JOURNEY ESTIMATION USING MACHINE LEARNING IN PYTHON

A Project Report

Submitted by:

1.DEBAL GHOSH (17010307016)

2.SHWETA KUMARI (17010307063)

3.TABNA SHAHID (17010307082)

in partial fulfillment for the award of the degree

of

Bachelor of Technology in Computer Science & Engineering

Department of Computer Science and Engineering

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata -700 125

December 2020

JOURNEY ESTIMATION USING MACHINE LEARNING IN PYTHON

Submitted by:

1.DEBAL GHOSH (17010307016)

2.SHWETA KUMARI (17010307063)

3.TABNA SHAHID (17010307082)

in partial fulfilment for the award of the degree

of

Bachelor of Technology in Computer Science & Engineering

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata -700 125

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata -700 125

Department of Computer Science and Engineering

<u>BONAFIDE CERTIFICATE</u>

Certified that this project report "**Journey Estimation Using Machine Learning in Python**" is the bonafide work of:

1.DEBAL GHOSH (17010307016)

2.SHWETA KUMARI (17010307063)

3.TABNA SHAHID (17010307082)

who carried out the project work under my supervision.

Signature of the Head of the Department

Mr. Jayanta Aich
Head of the Department
Assistant Professor
Dept. of Computer Sc. & Engineering
Brainware University, Barasat

Signature of the Supervisor

Mr. Shiplu Das
Supervisor
Assistant Professor
Dept. of Computer Sc. & Engineering
Brainware University, Barasat

TABLE OF CONTENTS

ABSTRACT

In this project, we were asked to experiment with a real-world dataset, and to explore how machine learning algorithms can be used to find the patterns in data. We were expected to gain experience using a common Machine Learning approach using Python and were expected to submit a report about the dataset and the algorithms used. After performing the required tasks on a dataset of our choice, herein lies our final report.

LIST OF TABLES

LIST OF FIGURES

1.Introduction

1.1. Problem Statement:

Implement a Journey Estimation software in python that determines the time and date of arrival when a distance taken as input is covered in fragments by separate modes of transportation, taking into account whether the journey is occurring on a weekday or a weekend.

1.1.1. Purpose

Travelling agencies like trip advisors and inter-state business meetings require optimal journeys to be scheduled in order to minimize cost and latency. An automated graphical user-friendly window aids the estimation of the exact time of arrival.

1.1.2. Mathematical Procedures Used

Linear regression is a powerful tool that can aid the intuition of a best-fitting-line/curve where one dependent variable can be algebraically computed from a one or more independent variables.

1.2. Dataset

Sample data used for training the regression model was collected from ResearchGate and Wikipedia.

Table 1: Sample Dataset

Track	Track distance (km)	Average speed of fastest 10 cars 2014 (km/h)	Standard deviation 2014 (km/h)	Average speed of fastest 10 cars 2015 (km/h)	Standard deviation 2015 (km/h)	Percentage increase in average speed
Melbourne, Australia	5.303	190.0	6.23	215.7	0.96	13.56
Sakhir, Bahrain	5.412	205.7	0.65	207.0	0.75	0.64
Shanghai, China	5.451	169.2	1.03	201.1	0.80	18.85
Kuala Lumpur, Malaysia	5.543	165.7	1.66	191.3	5.45	15.44
Montmeló, Spain	4.655	192.2	0.84	194.0	0.73	0.93
Monte Carlo, Monaco	3.337	155.2	0.48	156.8	0.46	1.04
Montreal, Canada	4.361	206.3	0.67	207.4	0.62	0.53
Silverstone, UK	5.891	213.4	2.83	226.6	0.59	6.20
Budapest, Hungary	4.381	186.2	1.10	189.1	0.57	1.56
Spa, Belgium	7.004	196.1	1.34	231.66	0.47	18.13
Monza, Italy	5.793	243.8	0.70	247.0	0.67	1.33
Singapore	5.065	171.02	0.53	172.61	0.80	0.93
Suzuka, Japan	5.807	222.11	1.46	222.97	1.31	0.38
Sochi, Russia	5.853	211.85	1.13	213.70	1.26	0.87
Abu Dhabi, UAE	5.554	196.38	1.05	196.46	1.18	0.04

1.2.1. Dataset Used to Train Regression Mode

Distance Covered is measured in Kilometer(km)

Weekday / Week-end is a discrete value capable of being either a weekend or a weekday

Time Taken is measured in hours

Table 2: Data collected for Mode of Transportation: Car

Distance Covered	Weekday /Week-end	Time Taken
30	Weekday	0.75
50	Weekday	1.25
70	Weekday	1.75
90	Weekday	2.26
110	Weekday	2.75
130	Weekday	3.25
150	Weekday	3.75
170	Weekday	4.25
190	Weekday	4.75
210	Weekday	5.25
20	Weekend	0.25
30	Weekend	0.375
40	Weekend	0.5
50	Weekend	0.625
60	Weekend	0.75
70	Weekend	0.875
80	Weekend	1.0
90	Weekend	1.125
100	Weekend	1.25
110	Weekend	1.375

Table 3: Data Collected for mode of transportation: Bike

Distance Covered	Weekday / Weekend	Time taken
10	Weekday	0.66
15	Weekday	1.00
20	Weekday	1.33
25	Weekday	1.66
30	Weekday	2.00
35	Weekday	2.33
40	Weekday	2.66
45	Weekday	3.00
50	Weekday	3.33
55	Weekday	3.67
5	Weekend	0.25
10	Weekend	0.50
15	Weekend	0.75
20	Weekend	1.00
25	Weekend	1.25
30	Weekend	1.50
35	Weekend	1.75
40	Weekend	2.00
45	Weekend	2.25
50	Weekend	2.50

Table 4: Data collected for mode of transportation: Train

Distance Covered	Weekday / Weekend	Time Taken
200	Weekend	0.80
210	Weekend	0.84
220	Weekend	0.88
230	Weekend	0.92
240	Weekend	0.96
250	Weekend	1.00
260	Weekend	1.04
270	Weekend	1.08
280	Weekend	1.12
290	Weekend	1.16
200	Weekday	1.33
215	Weekday	1.43
230	Weekday	1.53
245	Weekday	1.63
260	Weekday	1.73
275	Weekday	1.83
290	Weekday	1.93
305	Weekday	2.03
320	Weekday	2.13
335	Weekday	2.23

Table 5: Data collected for mode of transportation: Foot

Distance Covered	Weekday / Weekend	Time Taken
02	Weekday	0.66
03	Weekday	1.00
04	Weekday	1.33
05	Weekday	1.66
06	Weekday	2.00
07	Weekday	2.33
08	Weekday	2.66
09	Weekday	3.00
10	Weekday	3.33
11	Weekday	3.66
02	Weekend	0.40
03	Weekend	0.60
04	Weekend	0.80
05	Weekend	1.00
06	Weekend	1.20
07	Weekend	1.40
08	Weekend	1.60
09	Weekend	1.80
10	Weekend	2.00
11	Weekend	2.20

Table 6: Data Collected for mode of transportation: Bus

Distance Covered	Weekday / Weekend	Time Taken
100	Weekday	2.50
120	Weekday	3.0ù0
140	Weekday	3.50
160	Weekday	4.00
180	Weekday	4.50
200	Weekday	5.00
220	Weekday	5.50
240	Weekday	6.00
260	Weekday	6.50
280	Weekday	7.00
100	Weekend	1.66
105	Weekend	1.75
110	Weekend	1.83
115	Weekend	1.91
120	Weekend	2.00
125	Weekend	2.08
130	Weekend	2.16
135	Weekend	2.25
140	Weekend	2.33
145	Weekend	2 41

1.2.2. Regression Curve Representation

X-axis: Distance Covered

Y-axis: Time Taken

Figure 1: Regression Curve for Mode of Transportation: Bike

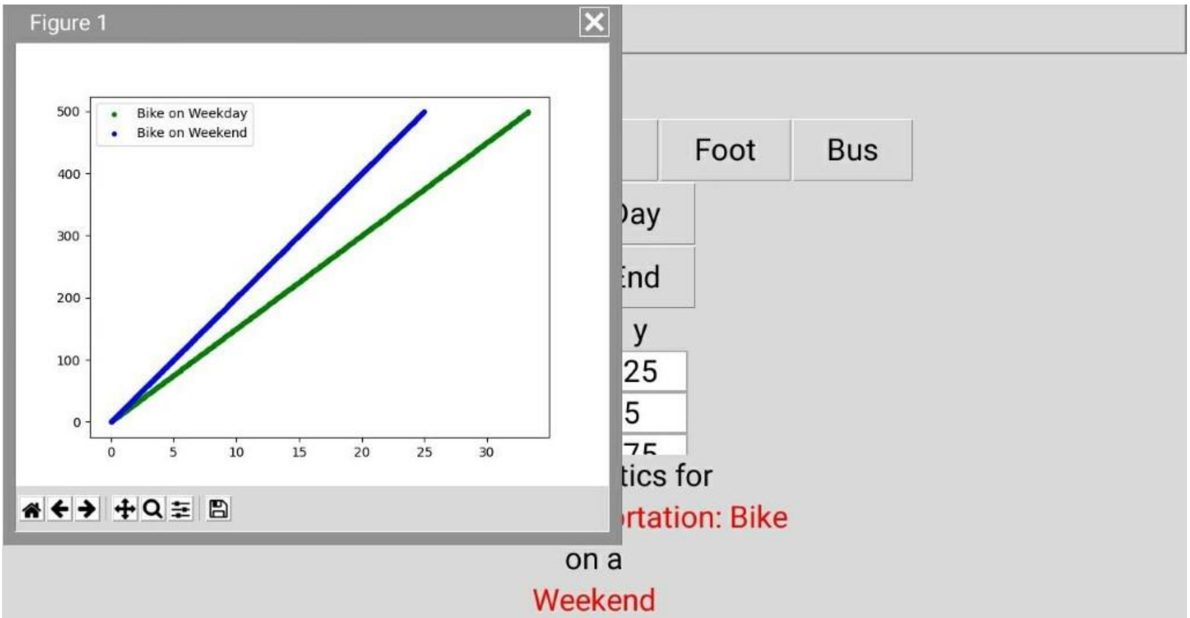


Figure 2: Regression Curve for Mode of Transportation: Car

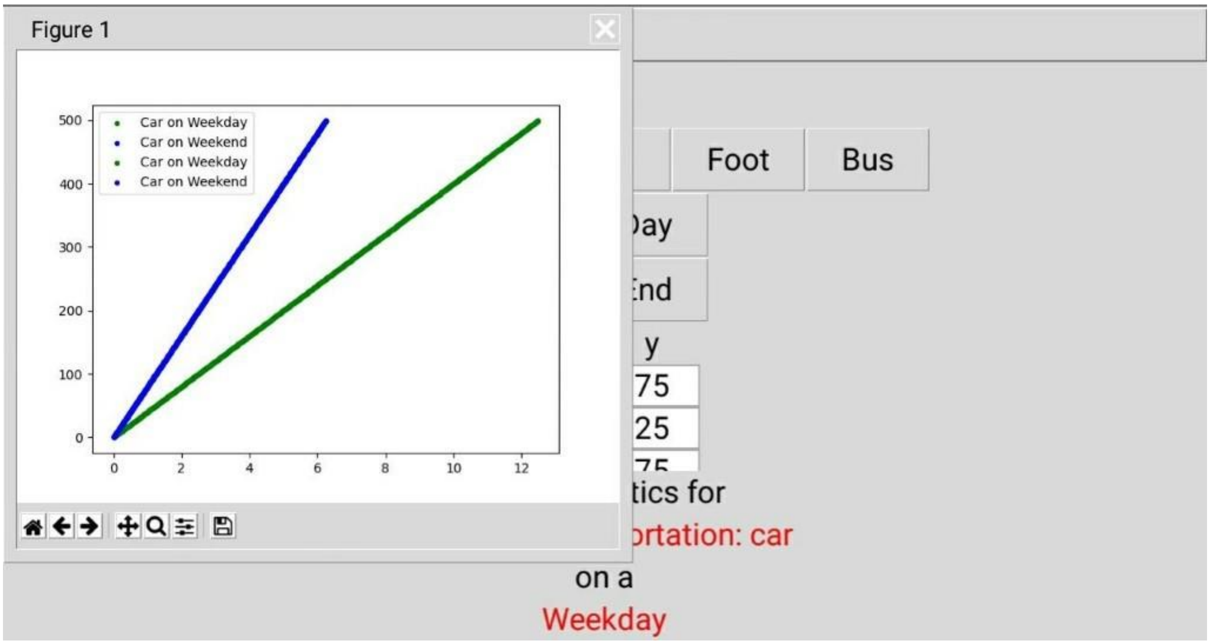


Figure 3: Regression Curve for Mode of Transportation: Train

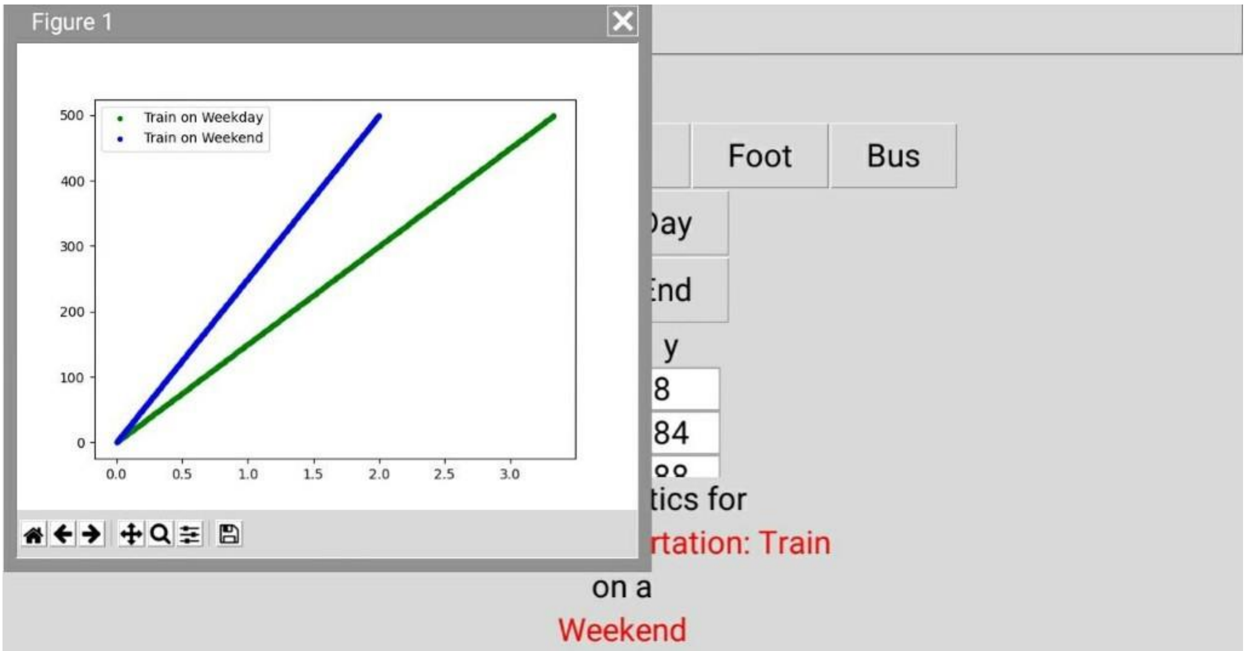


Figure 4: Regression Curve for Mode of Transportation: Foot

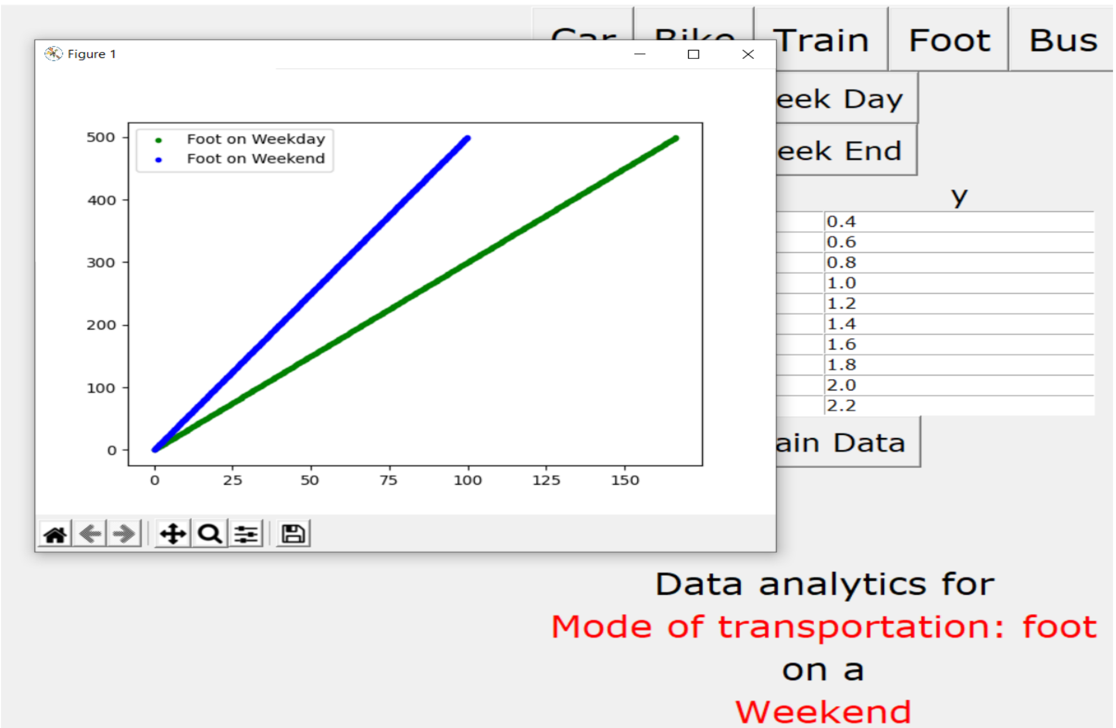
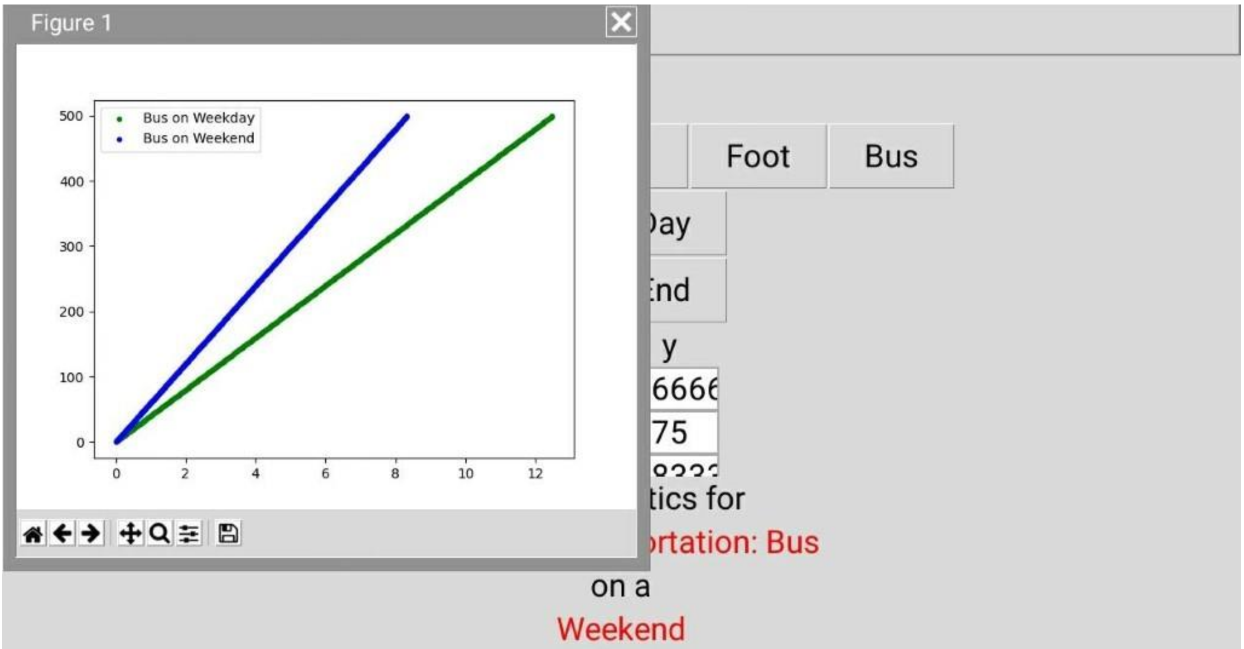


Figure 5: Regression Curve for Mode of Transportation: Bus



2.Chapters

2.1. Introduction to Least Square Curve Fitting Methods

We list below the equations of some common types of lines and curves:

1. y=a+bx [Straight Line]
2. Parabola: $y=ax+bx+cx^2$

Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points, possibly subject to constraints. Curve fitting can involve either interpolation, where an exact fit to the data is required, or smoothing in which a "smooth" function is constructed that approximately fits the data. A related topic is regression analysis, which focuses more on questions of statistical inference such as how much uncertainty is present in a curve that is fit to data observed with random errors. Fitted curves can be used as an aid for data visualization, to infer values of a function where no data are available, and to summarize the relationships among two or more variables. Extrapolation refers to the use of a fitted curve beyond the range of the observed data, and is subject to a degree of uncertainty since it may reflect the method used to construct the curve as much as it reflects the observed data.

The first-degree polynomial equation

{\displaystyle y=ax+b\;} y=ax+b

is a line with slope *a*. A line will connect any two points, so a first-degree polynomial equation is an exact fit through any two points with distinct x coordinates.

If the order of the equation is increased to a second-degree polynomial, the following results:

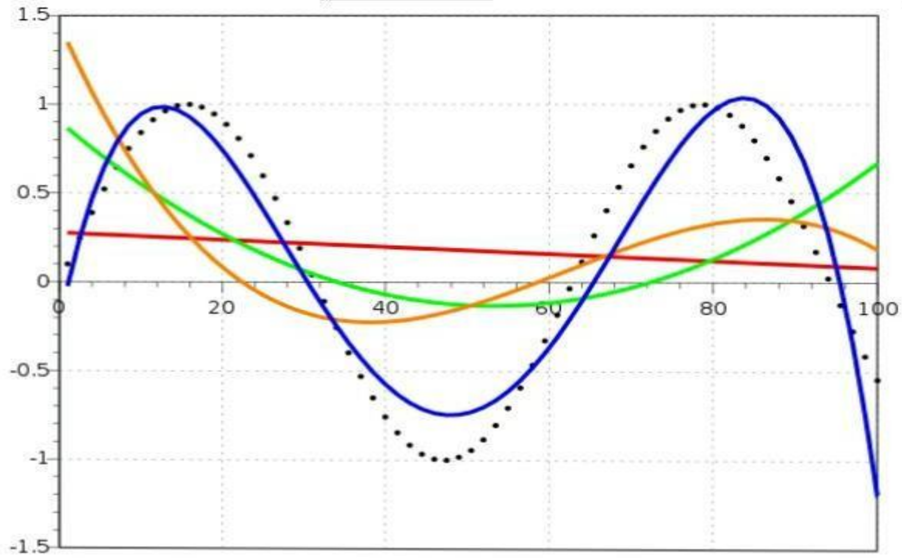
{\displaystyle y=ax^{2}+bx+c\;} $y=a_1 x^2 + a_2 x + c$

This will exactly fit a simple curve to three points.

If the order of the equation is increased to a third-degree polynomial, the following is obtained:

{\displaystyle y=ax^{3}+bx^{2}+cx+d\;.}

Figure 6: Diagrammatic Representation of Polynomial Curves



Polynomial curves fitting points generated with a sine function. The black dotted line is the "true" data, the red line is a first degree polynomial, the green line is second degree, the orange line is third degree and the blue line is fourth degree.

2.1.1. Matrix Algebra and Multivariable Regression

Matrix algebra is widely used for the derivation of multiple regression because it permits a compact, intuitive depiction of regression analysis.

In addition, matrix notation is flexible in that it can handle any number of independent variables.

A matrix is a rectangular array of numbers with rows and columns. As noted, operations performed on matrices are performed on all elements of a matrix simultaneously. In this section we provide the basic understanding of matrix algebra that is necessary to make sense of the expression of multiple regression in matrix form.

The resulting first degree, second degree of third-degree polynomial has variables that can be classified under two categories:

- Independent Variables
- Dependent Variation

Independent Variables are the variables used to predict the value of the dependent variable. Independent variables assign numerical values to the measures of factors that determine a predicted output which is the dependent variable.

Dependent Variables are the values we are attempting to predict through least squares method or regression analysis.

In a regression analysis procedure two commonly discussed impediments are as follows:

- Overfitting
- Multicollinearity

An unwanted circumstance occurs when an independent variable has a low Karl Pearson's Correlation coefficient value (less than 0.75) with respect to the Dependent Variable, in which case the independent variable in question is discarded.

This unwanted circumstance is known as

Overfitting - the existence of an unnecessary independent variable that does not contribute in a significant manner in determining the dependent variable.

Multicollinearity - is an impediment that arises when an independent variable has a high Karl Pearson's Correlation coefficient value with respect to another independent variable.

Ideally all independent variables must have low Karl Pearson's Correlation coefficient value with respect to each other.

2.1.2. A Matrix Formulation of the Multiple Regression Model

In the multiple regression setting, because of the potentially large number of predictors, it is more efficient to use matrices to define the regression model and the subsequent analyses. Here, we review basic matrix algebra, as well as learn some of the more important multiple regression formulas in matrix form.

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \qquad \text{for } i = 1, \ldots, n$$

If we actually let $i = 1, \ldots, n$, we see that we obtain n equations:

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_1 + \epsilon_1 \\ y_2 &= \beta_0 + \beta_1 x_2 + \epsilon_2 \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_n + \epsilon_n \end{aligned}$$

By taking advantage of this pattern, we can instead formulate the above simple linear regression function in matrix notation:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

That is, instead of writing out the n equations, using matrix notation, our simple linear regression function reduces to a short and simple statement:

$$Y = X\beta + \epsilon$$

- ***X*** is an $n \times 2$ **matrix**.
- ***Y*** is an $n \times 1$ **column vector**, ***β*** is a 2×1 column vector, and ***ε*** is an $n \times 1$ column vector.
- The matrix ***X*** and vector ***β*** are multiplied together using the techniques of **matrix multiplication**.
- And, the vector ***Xβ*** is added to the vector ***ε*** using the techniques of **matrix addition**.

2.1.3. Definition of a Matrix

An ***r* × *c* matrix** is a rectangular array of symbols or numbers arranged in *r* rows and *c* columns. A matrix is almost always denoted by a single capital letter in boldface type.

Here are three examples of simple matrices. The matrix ***A*** is a 2 × 2 **square matrix** containing numbers:

$$A = \begin{bmatrix} 1 & 2 \\ 6 & 3 \end{bmatrix}$$

The matrix ***B*** is a 5 × 3 matrix containing numbers:

$$B = \begin{bmatrix} 1 & 80 & 3.4 \\ 1 & 92 & 3.1 \\ 1 & 65 & 2.5 \\ 1 & 71 & 2.8 \\ 1 & 40 & 1.9 \end{bmatrix}$$

And, the matrix ***X*** is a 6 × 3 matrix containing a column of 1's and two columns of various *x* variables:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ 1 & x_{31} & x_{32} \\ 1 & x_{41} & x_{42} \\ 1 & x_{51} & x_{52} \\ 1 & x_{61} & x_{62} \end{bmatrix}$$

2.1.4. Definition of a Vector and a Scalar

A **column vector** is an *r* × 1 matrix, that is, a matrix with only one column. A vector is almost often denoted by a single lowercase letter in boldface type. The following vector ***q*** is a 3 × 1 column vector containing numbers:

$$q = \begin{bmatrix} 2 \\ 5 \\ 8 \end{bmatrix}$$

A **row vector** is a 1 × *c* matrix, that is, a matrix with only one row. The vector ***h*** is a 1 × 4 row vector containing numbers:

$$h = [21 \quad 46 \quad 32 \quad 90]$$

A 1 × 1 "matrix" is called a **scalar**, but it's just an ordinary number, such as 29 or σ^2.

2.1.5. Definition of the Transpose of a Matrix

$$A = \begin{bmatrix} 1 & 5 \\ 4 & 8 \\ 7 & 9 \end{bmatrix}$$

is the 2 × 3 matrix ***A***':

$$A' = A^T = \begin{bmatrix} 1 & 4 & 7 \\ 5 & 8 & 9 \end{bmatrix}$$

The **transpose** of a matrix ***A*** is a matrix, denoted ***A'*** or ***A***^{T}, whose rows are the columns of ***A*** and whose columns are the rows of ***A*** — all in the same order.

For example, the transpose of the 3 × 2 matrix ***A* -** the ***X'X*** matrix in the simple linear regression setting must be:

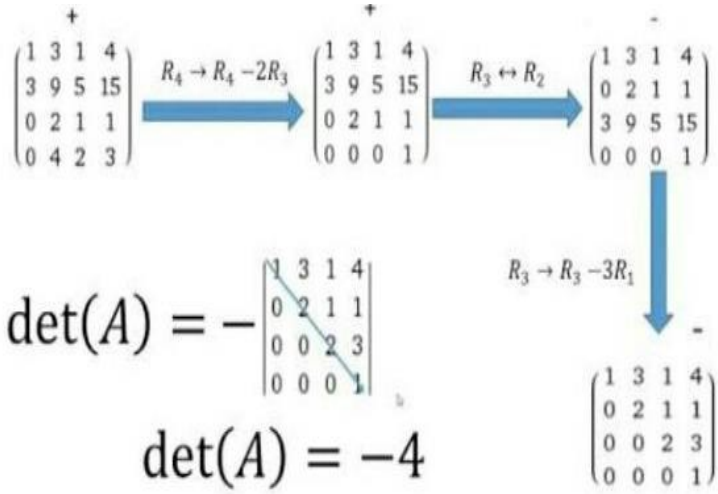
$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

the ***X'X*** matrix in the simple linear regression setting must be:

$$X'X = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & x_n \\ 1 & \end{bmatrix} = \begin{bmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{bmatrix}$$

2.2. Objective

- Implementing algorithms for computing the determinant of matrices and solving a system of Linear Equations



- Designing three X-window interfaces using Python Tkinter Library
    1. One that displays a 10×2 grid filled with entry widgets with the purpose of inputting user specific training data
    2. One that displays respective entry widgets for inputting the date and time in which the journey begins
    3. A list that displays fragments of the entire journey covered via several means of transportation, along with the time of arrival.
    4. Buttons that are positioned alongside aforementioned widgets for ease of processing.
    5. A Menu bar for the ease of navigation

- Collection of Data: Collection of scatter plots for the distance covered and time taken by several vehicles on weekends and weekdays from ResearchGate and Wikipedia.

- Evaluating the closest fitting curve:

$y=ax^2 + bx + c$

Where, y: Time taken to cover distance and

x: Distance covered for the following modes of transportation:

1. Car
2. Bus
3. Train
4. Bike

5. Foot

Separately on weekends and weekdays.

- Importing the datetime module in python for ease of implementing datetime operations.
- Importing Matplotlib for displaying graphs of closest fitting curves.
- Resizing aforementioned widgets and labels in the user interface for clearer visibility
- Testing the code rigorously for bugs, errors and exceptions.

2.3. Planning

Week-1**:** Evaluate and explore the numerous methods in solving a system of Linear Equations, that best serves the purpose of the project.

Week-2**:** Code, debug and save three separate python functions in a .py file:

1. One that returns the determinant of a matrix
2. One that takes as parameters three matrices corresponding to the Matrix representation of a system of Linear Algebra:

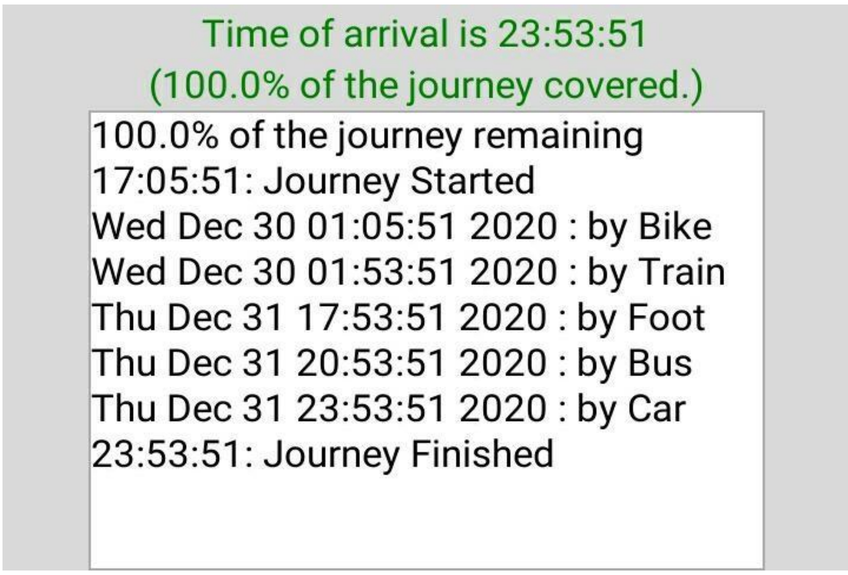
   Y=AX+B   and produces a solution

3. One that takes as parameters a datetime object and a float value representative of the number of hours and returns a datetime object that is the chronological summation of two

Week-3**:** Collect scatter plots and data for training purposes

Week-4**:** Build the user interface as dictated by the design

Figure 7: Panel to Show the Details of the Journey Covered

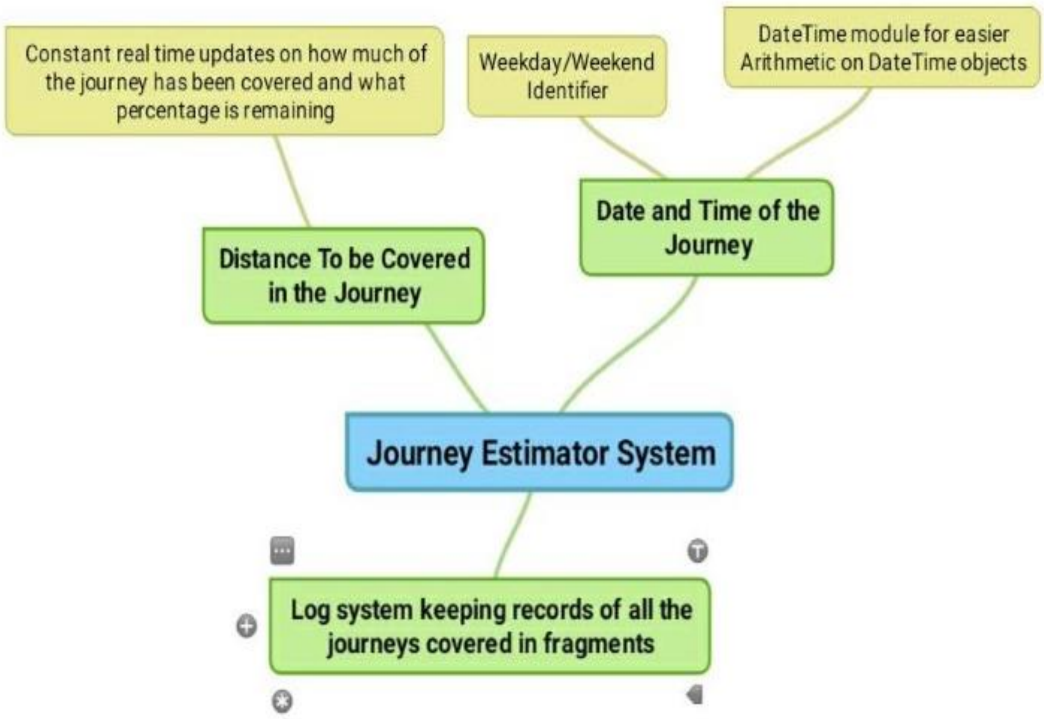


2.4. Working Prototype and Requirement Analysis

2.4.1. Requirement Analysis

- Python tkinter library for implementing graphical user interfaces
- Python matplotlib used for plotting graphs

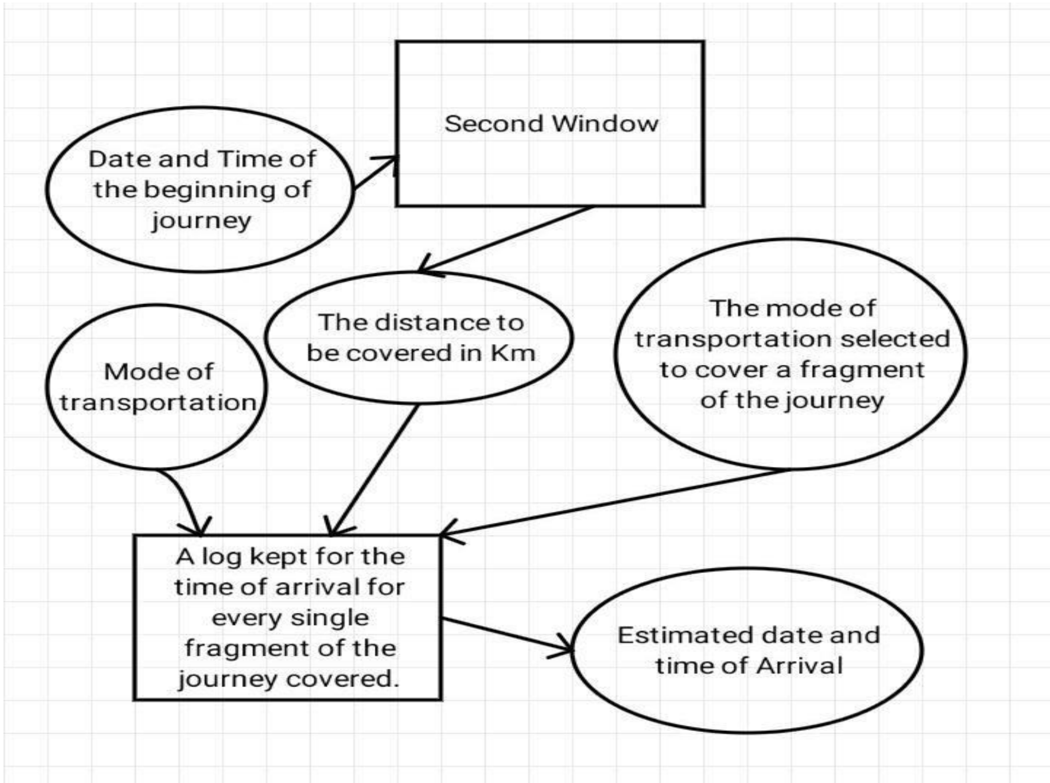
Figure 8: Diagrammatic Flow of the Algorithm



- User-defined / in built functions for matrix multiplications and matrix inversions

2.5. System Flow

Figure 9: Rough Outline (DFD) of the Project



2.6. Proposed Design

- On launching the project, the opening window should look familiar to the following orientation:

Figure 10

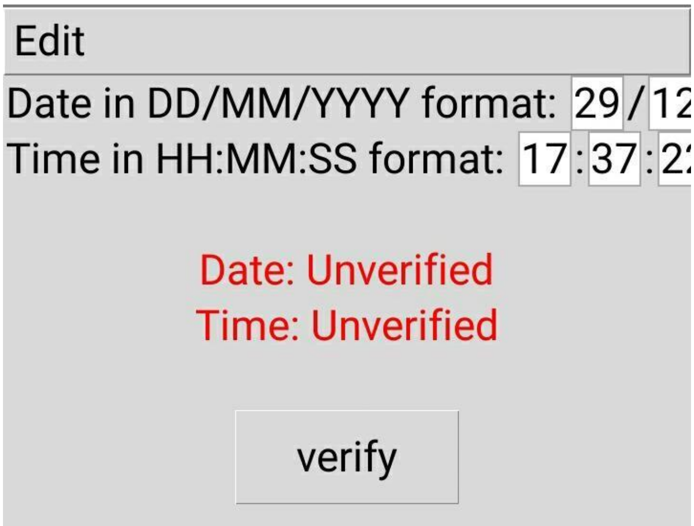
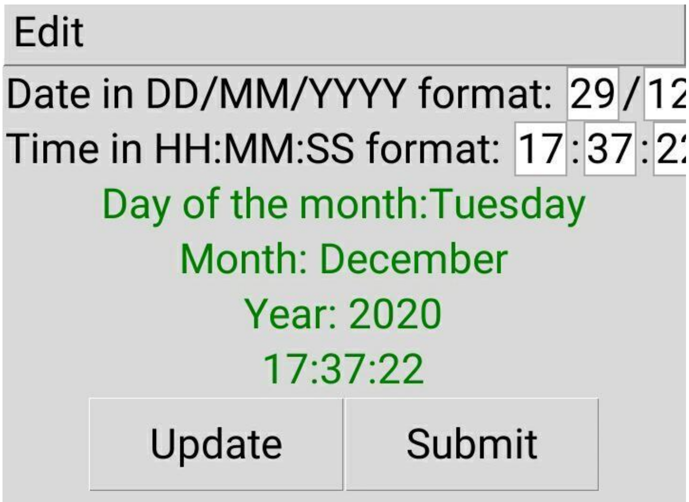
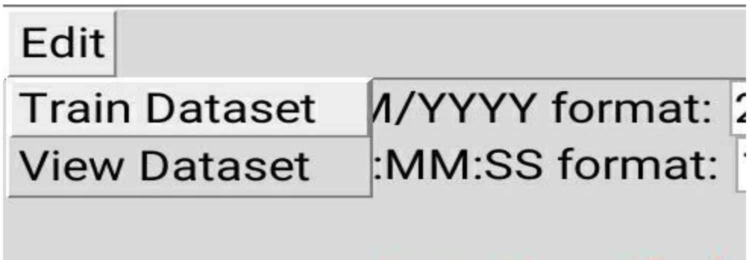


Figure 11



- On clicking the Verify button, an onClick () event should be called that verifies the datetime input and displays the window shown above.

Figure 12

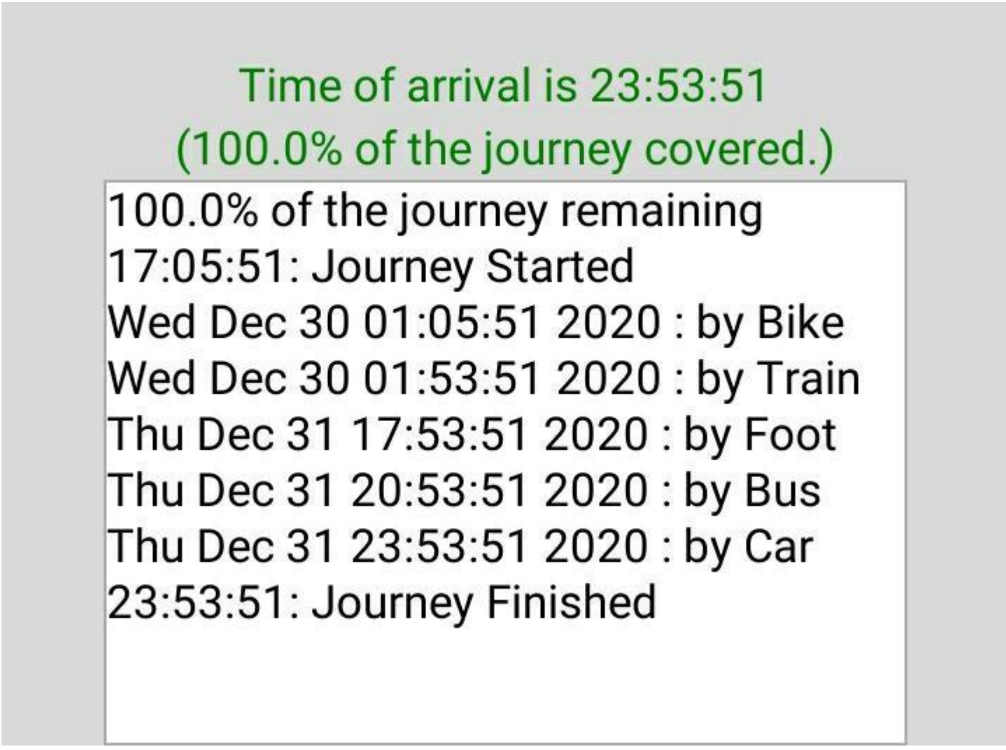


- The menu bar helps the user navigate through the edit option which helps the user navigate to the Train Dataset window

Figure 13

- The main computation is logged in a list structure.

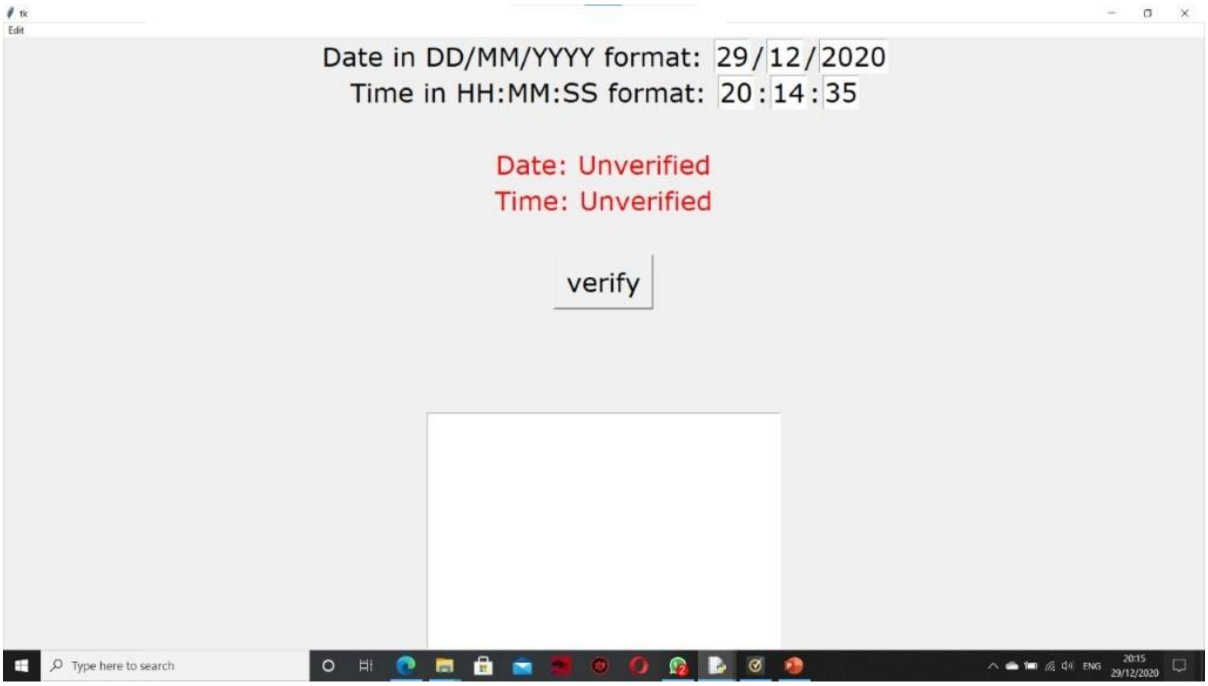
Figure 14



2.7. Experimental Result

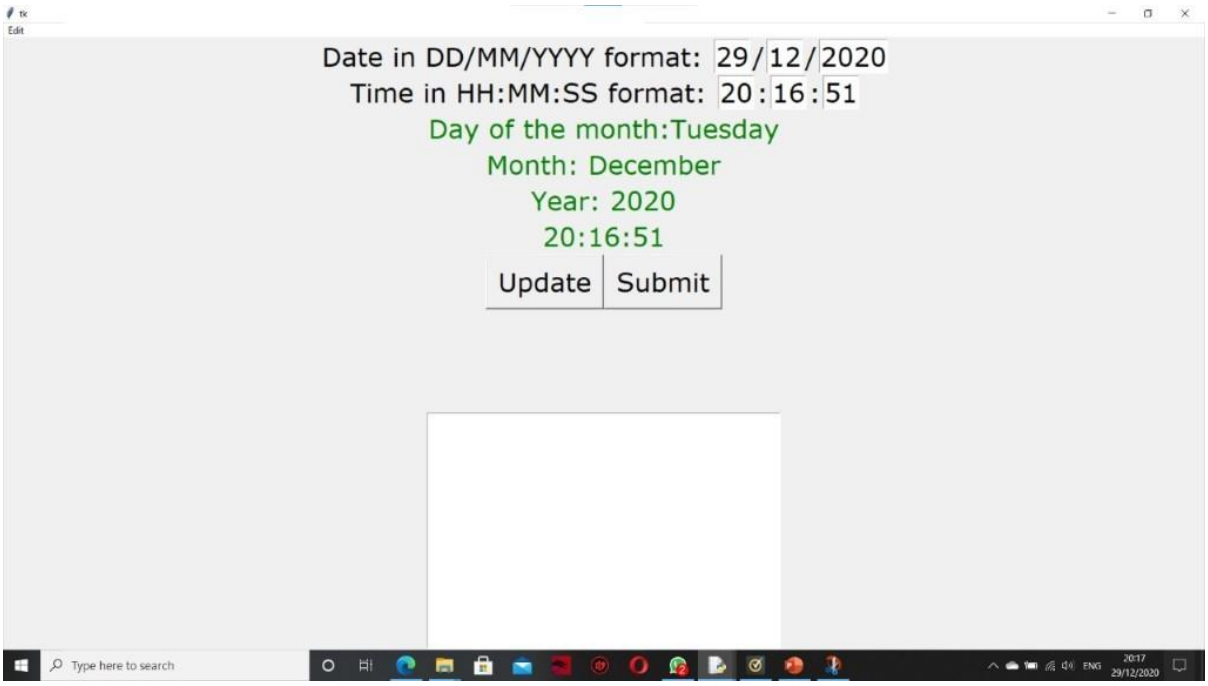
- Opening Window

Figure 15



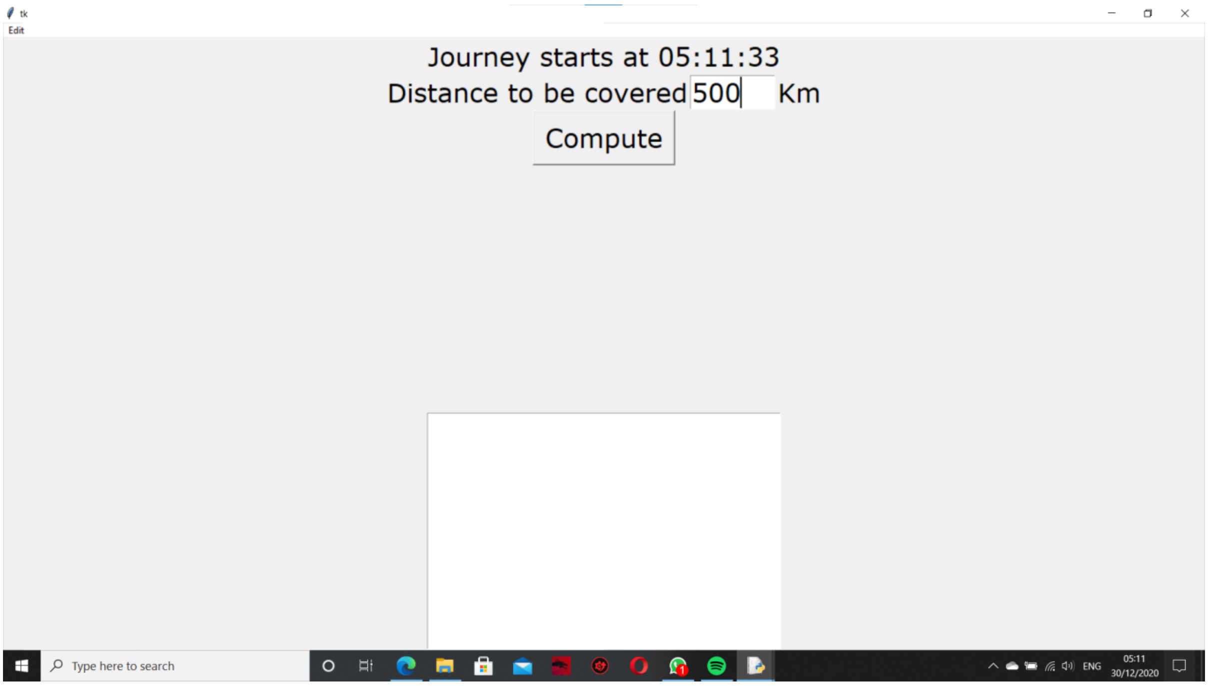
- Verify clicked

Figure 16



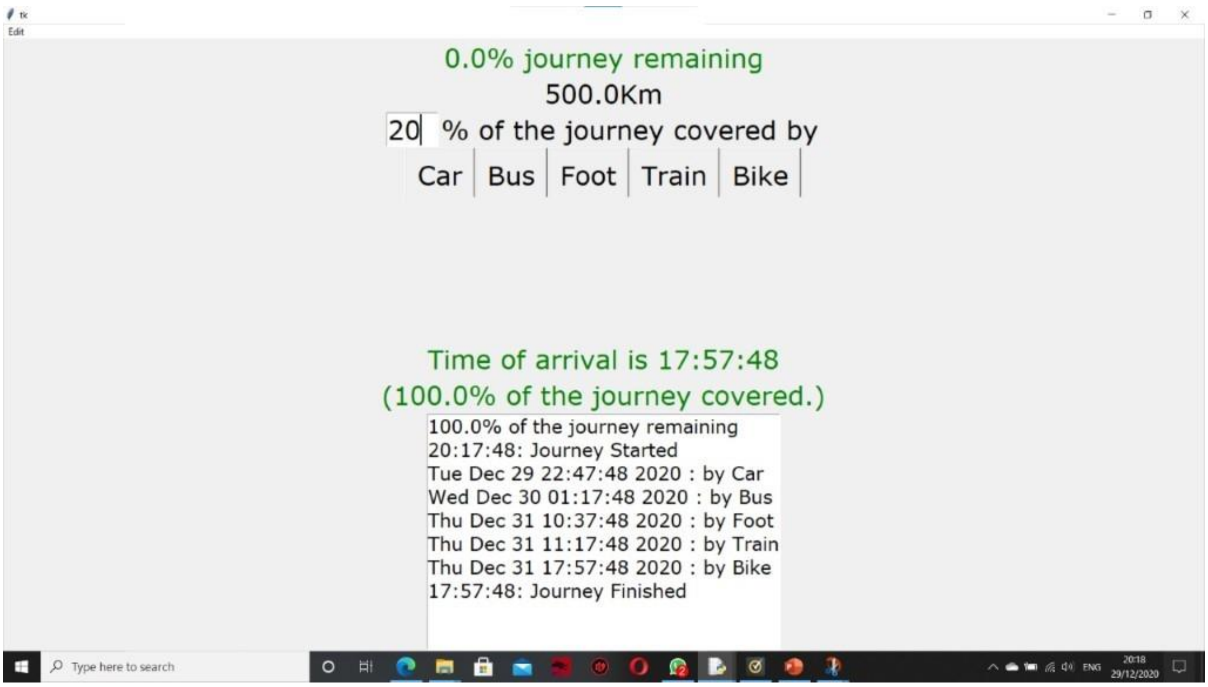
- Submit clicked

Figure 17



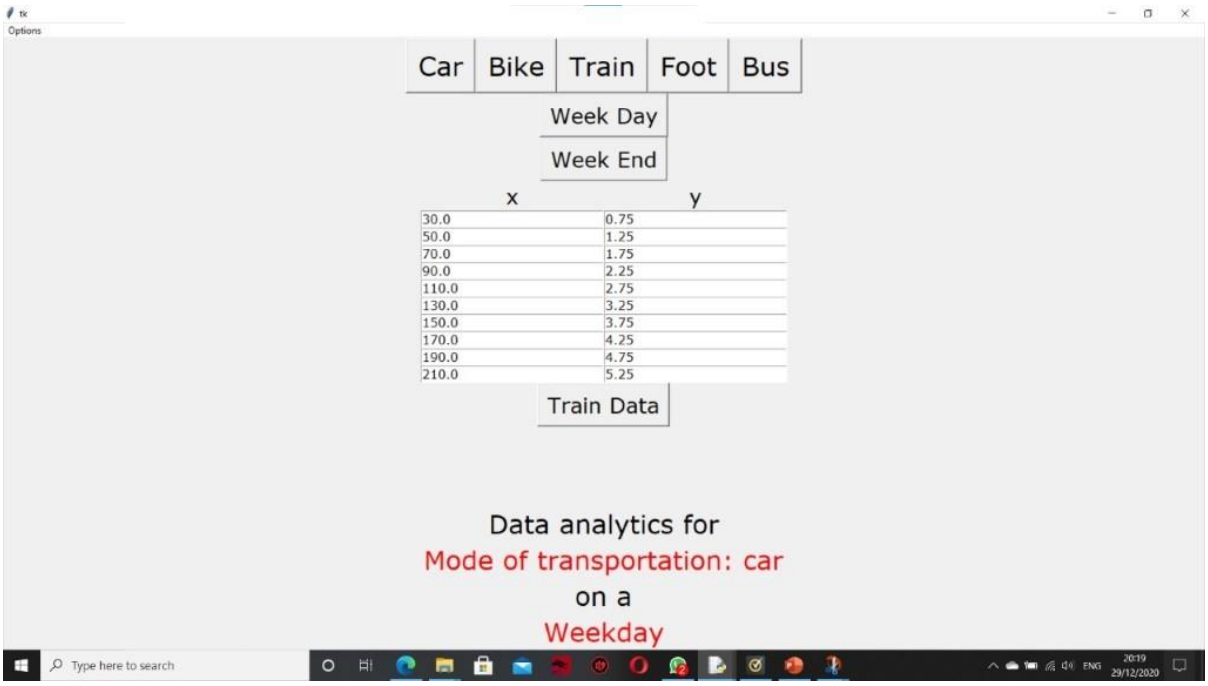
- Compute button clicked

Figure 18



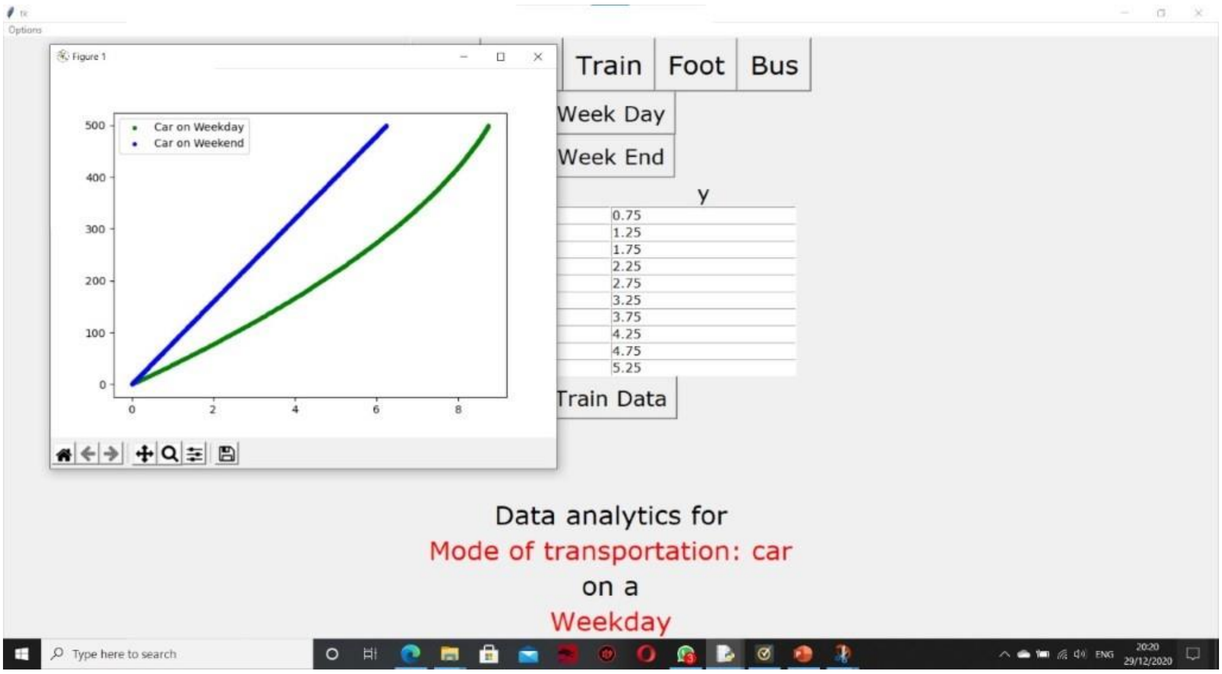
- Edit menu selected

Figure 19



- Train Data button clicked

Figure 20



2.8. Future Scope

- Justification:
    1. Travelling agencies and the tourism industry is burdened with the activity of constantly scheduling trips for customers.
    2. On site business trips can be optimized taking into account whether the journey is covered on a weekday or a weekend.

- Project Exclusions:
    1. The alternative of covering customized fragment of the journey on air (flight) was not implemented but is a major improvement that we plan on incorporating in the future.
    2. National holidays, bank holidays and social festivals were not taken into account in determining the arrival time of the journey, also an addition we plan on making in the near future.

- Constraints:
    1. Taking weather conditions into account in the process of determining the arrival time of the journey was highly unconventional without access to paid datasets available only through academia.
    2. Training the algorithm with a huge dataset would significantly slow the system down, hitherto the number of labelled datasets were kept countable.

- Assumption: The following assumptions were made in building this project:
    3. Factors like weather conditions and public holidays were neglected in determining the arrival time of the journey.

2.9. Conclusion

- How the results support or contradict the hypothesis:

  The computed regression curves evaluated via user data were consistent with the scatter plot. Root mean square error was negligible.

- The relationship between Independent and dependent variables:

  The independent variable was linearly correlated with the dependent variables with the pre-defined data, however once the user decides the data to have somewhat for a curve a first-degree polynomial is closely drawn using the least squares method.

- Changes we plan on making and further study:
  - A flight module is currently under development that will successfully predict the arrival time of flights.
  - Areas of future study: The gradient descent method of linear regression as opposed to the Matrix inversion method is an area, we plan on researching further to evaluate more optimal results.

3. CODE

```
from datetime import timedelta
from tkinter.ttk import *
from tkinter import *
import datetime
import matplotlib.pyplot as plt
import threading

def cofac(m,t,p,q,n):
i=0
j=0
for row in range(0,n):
for column in range(0,n):
if(row!=p and column!=q):
t[i][j]=m[row][column]
j+=1
if(j==n-1):
i+=1
j=0
def det(m,n):
t=[[float(0) for x in range(n)]for y in range(n)]
d=float(0)
if(n==1):
return(m[0][0])
sign=1
for f in range(0,n):
cofac(m,t,0,f,n)
d+=sign*m[0][f]*det(t,n-1)
```

```
sign*=(-1)
return(d)
def solve(m,s,a,n):
for i in range(0,n):
c=[[float(0) for x in range(n)] for y in range(n)]
for p in range(0,n):
for q in range(0,n):
c[p][q]=m[p][q]
for j in range(0,n):
c[j][i]=a[j]
s[i]=det(c,n)/det(m,n)

car=int(0)
bike=int(1)
bus=int(2)
foot=int(3)
train=int(4)
weekday=int(0)
weekend=int(1)
a=[[float(0) for x in range(2)] for y in range(5)]
b=[[float(0) for x in range(2)] for y in range(5)]
c=[[float(0) for x in range(2)] for y in range(5)]


x=0
y=1
car_weekday=[[float(0) for x in range(10)] for y in range(2)]
car_weekend=[[float(0) for x in range(10)] for y in range(2)]
bike_weekday=[[float(0) for x in range(10)] for y in range(2)]
bike_weekend=[[float(0) for x in range(10)] for y in range(2)]
```

```
bus_weekday=[[float(0) for x in range(10)] for y in range(2)]
bus_weekend=[[float(0) for x in range(10)] for y in range(2)]
train_weekday=[[float(0) for x in range(10)] for y in range(2)]
train_weekend=[[float(0) for x in range(10)] for y in range(2)]
foot_weekday=[[float(0) for x in range(10)] for y in range(2)]
foot_weekend=[[float(0) for x in range(10)] for y in range(2)]


for j in range(0,10):
car_weekend[x][j]=20+10*float(j)
car_weekend[y][j]=car_weekend[x][j]/80
for j in range(0,10):
car_weekday[x][j]=30+20*float(j)
car_weekday[y][j]=car_weekday[x][j]/40
for j in range(0,10):
bike_weekday[x][j]=10+5*float(j)
bike_weekday[y][j]=bike_weekday[x][j]/15
for j in range(0,10):
bike_weekend[x][j]=5+5*float(j)
bike_weekend[y][j]=bike_weekend[x][j]/20
for j in range(0,10):
bus_weekday[x][j]=100+20*float(j)
bus_weekday[y][j]=bus_weekday[x][j]/40
for j in range(0,10):
bus_weekend[x][j]=100+5*float(j)
bus_weekend[y][j]=bus_weekend[x][j]/60

for j in range(0,10):
train_weekday[x][j]=200+15*float(j)
train_weekday[y][j]=train_weekday[x][j]/150
```

```
for j in range(0,10):
train_weekend[x][j]=200+10*float(j)
train_weekend[y][j]=train_weekend[x][j]/250
for j in range(0,10):
foot_weekday[x][j]=2+1*float(j)
foot_weekday[y][j]=foot_weekday[x][j]/3
for j in range(0,10):
foot_weekend[x][j]=2+1*float(j)
foot_weekend[y][j]=foot_weekend[x][j]/5


def trainData(item,vehicle,day):
m=[[float(0) for x in range(3)] for y in range(3)]

sum=float(0)
for i in range(0,10):
sum+=(item[x][i])*(item[x][i])
m[0][0]=sum
m[1][1]=sum
m[2][2]=sum

sum=float(0)
for i in range(0,10):
sum+=(item[x][i])*(item[x][i])*(item[x][i])
m[1][0]=sum
m[2][1]=sum

sum=float(0)
for i in range(0,10):
sum+=(item[x][i])*(item[x][i])*(item[x][i])*(item[x][i])
```

```
m[2][2]=sum

sum=float(0)
for i in range(0,10):
sum+=(item[x][i])
m[0][1]=sum
m[1][2]=sum

m[0][2]=10

ans=[float(0) for x in range(3)]
sum=float(0)
for i in range(0,10):
sum=sum+item[y][i]
ans[0]=sum

sum=float(0)
for i in range(0,10):
sum=sum+item[y][i]*item[x][i]
ans[1]=sum

sum=float(0)
for i in range(0,10):
sum=sum+item[y][i]*item[x][i]*item[x][i]
ans[2]=sum

s=[float(0) for x in range(3)]
solve(m,s,ans,3)
a[vehicle][day]=s[0]
b[vehicle][day]=s[1]
```

```
c[vehicle][day]=s[2]

trainData(bus_weekday,bus,weekday)
trainData(bus_weekend,bus,weekend)
trainData(car_weekday,car,weekday)
trainData(car_weekend,car,weekend)
trainData(bike_weekday,bike,weekday)
trainData(bike_weekend,bike,weekend)
trainData(foot_weekday,foot,weekday)
trainData(foot_weekend,foot,weekend)
trainData(train_weekday,train,weekday)
trainData(train_weekend,train,weekend)


def distanceCovered(vehicle,d,day):
sum=float(0)
sum=a[vehicle][day]*d*d+b[vehicle][day]*d+c[vehicle][day]
return(sum)

graphWindow=Tk()
totalDist=float(0)

large_font = ('Verdana',25)
small_font = ('Verdana',20)

var1=var = StringVar(value='')

timeLabel=Frame(graphWindow)
timeLabel1=Label(timeLabel,text="  ")
timeLabel1.config(font=large_font)
```

```
timeLabel2=Label(timeLabel,text="  ")
timeLabel2.config(font=large_font)
timeLabel1.pack(side="top")
timeLabel2.pack(side="top")

menubar=Menu(graphWindow)
edit=Menu(menubar,tearoff=0)

menubar.add_cascade(label ='Edit', menu = edit)
remaining=float(100)

m=Menu(graphWindow)
ed=Menu(m,tearoff=0)
m.add_cascade(label ='Options', menu = ed)

def goBack():
global f1
global f15
global f2
global f3
global lBottom1
global lBottom2
global lBottom3
global lBottom4
global menubar
global cal
global ver
global but
global butFrame
global newBut
```

```
global timeEntry
global list
global totalPercentage
global timeLabel
global timeLabel1
global timeLabel2
global dEntry1

dEntry1.insert(0,"0")
timeLabel1.config(text="  ")
timeLabel2.config(text="  ")
totalPercentage=100
f1.pack_forget()
f15.pack_forget()
f2.pack_forget()
f3.pack_forget()
lBottom1.pack_forget()
lBottom2.pack_forget()
lBottom3.pack_forget()
lBottom4.pack_forget()
but.grid_forget()
newBut.grid_forget()
but.pack(side="top")

but.config(text="Verify")
graphWindow.config(menu=menubar)
cal.pack(side="top")
timeEntry.pack(side="top")
ver.pack(side="top")
butFrame.pack(side="top")
```

```
list.pack(side="bottom")
timeLabel.pack(side="bottom")


ed.add_command(label ='Back', command = goBack)

ed.add_command(label ='View Dataset', command = None)

def go():
global f1
global f15
global f2
global f3
global graphWindow
global cal
global ver
global but
global label1
global list
global button
global dist
global superLabel
global final
global timeEntry
global timeLabel
global m
global lBottom1
global lBottom2
global lBottom3
global lBottom4
```

```
timeLabel.pack_forget()
timeEntry.pack_forget()
superLabel.pack_forget()
final.pack_forget()
button.pack_forget()
dist.pack_forget()
list.pack_forget()
label1.pack_forget()
but.pack_forget()
cal.pack_forget()
ver.pack_forget()

f1.pack(side="top")
f15.pack(side="top")
f2.pack(side="top")
f3.pack(side="top")
lBottom4.pack(side="bottom")
lBottom3.pack(side="bottom")
lBottom2.pack(side="bottom")
lBottom1.pack(side="bottom")
graphWindow.config(menu=m)

final=Frame(graphWindow)
final2=Frame(final)
p=Entry(final2,width=3,font=large_font)
textP=Label(final2,text="% of the journey covered by")
textP.config(font=large_font)
final1=Frame(final)
```

```
totalPercentage=float(100)
def computeCar():
global p
global index
global totalDist
global list
global current
global totalPercentage
global label1
global car
global weekday
global weekend
xDay=int(current.strftime("%w"))
dayOfJourney=int(0)
if(xDay==0 or xDay==6):
dayOfJourney=weekend
else:
dayOfJourney=weekday


d=float(p.get())
if(totalPercentage==100):
mystr=current.strftime("%X")+": Journey Started"
list.insert((index),mystr)
index+=1
totalPercentage=totalPercentage-d
string=(str(totalPercentage)+"% journey remaining")
if(totalPercentage>=0 and totalPercentage<=100):
label1.config(text=string,fg="green")
```

```
d=(d*totalDist)/100
d=distanceCovered(car,d,dayOfJourney)
if(totalPercentage>=0):
current=current+timedelta(hours=d)
myString=current.strftime("%c")+" : by Car"

myString1="Time of arrival is "+current.strftime("%X")
myString2="("+str(100-totalPercentage)+"% of the journey covered.)"
if(totalPercentage>=0):
list.insert(index,myString)
index+=1
global timeLabel1
global timeLabel2
if(totalPercentage>=0):
timeLabel1.config(text=myString1,fg="green")
timeLabel2.config(text=myString2,fg="green")
if(totalPercentage==0):
mystr=current.strftime("%X")+": Journey Finished"
list.insert((index),mystr)
index+=1

def computeBus():
global p
global index
global totalDist
global list
global current
global totalPercentage
global label1
```

```
global bus
global weekday
global weekend
xDay=int(current.strftime("%w"))
dayOfJourney=int(0)
if(xDay==0 or xDay==6):
dayOfJourney=weekend
else:
dayOfJourney=weekday


d=float(p.get())

if(totalPercentage==100):
mystr=current.strftime("%X")+": Journey Started"
list.insert((index),mystr)
index+=1
totalPercentage=totalPercentage-d
string=(str(totalPercentage)+"% journey remaining")
if(totalPercentage>=0 and totalPercentage<=100):
label1.config(text=string,fg="green")
d=(d*totalDist)/100
d=distanceCovered(bus,d,dayOfJourney)
if(totalPercentage>=0):
current=current+timedelta(hours=d)
myString=current.strftime("%c")+" : by Bus"

myString1="Time of arrival is "+current.strftime("%X")
myString2="("+str(100-totalPercentage)+"% of the journey covered.)"
if(totalPercentage>=0):
```

```
list.insert(index,myString)
index+=1
global timeLabel1
global timeLabel2
if(totalPercentage>=0):
timeLabel1.config(text=myString1,fg="green")
timeLabel2.config(text=myString2,fg="green")
if(totalPercentage==0):
mystr=current.strftime("%X")+": Journey Finished"
list.insert((index),mystr)
index+=1


def computeFoot():
global p
global index
global totalDist
global list
global current
global totalPercentage
global label1
global foot
global weekday
global weekend
xDay=int(current.strftime("%w"))
dayOfJourney=int(0)
if(xDay==0 or xDay==6):
dayOfJourney=weekend
else:
dayOfJourney=weekday
```

```
d=float(p.get())

if(totalPercentage==100):
mystr=current.strftime("%X")+": Journey Started"
list.insert((index),mystr)
index+=1
totalPercentage=totalPercentage-d
string=(str(totalPercentage)+"% journey remaining")
if(totalPercentage>=0 and totalPercentage<=100):
label1.config(text=string,fg="green")
d=(d*totalDist)/100
d=distanceCovered(foot,d,dayOfJourney)
if(totalPercentage>=0):
current=current+timedelta(hours=d)
myString=current.strftime("%c")+" : by Foot"

myString1="Time of arrival is "+current.strftime("%X")
myString2="("+str(100-totalPercentage)+"% of the journey covered.)"
if(totalPercentage>=0):
list.insert(index,myString)
index+=1
global timeLabel1
global timeLabel2
if(totalPercentage>=0):
timeLabel1.config(text=myString1,fg="green")
timeLabel2.config(text=myString2,fg="green")
if(totalPercentage==0):
mystr=current.strftime("%X")+": Journey Finished"
list.insert((index),mystr)
index+=1
```

```
def computeTrain():
global p
global index
global totalDist
global list
global current
global totalPercentage
global label1
global train
global weekday
global weekend
xDay=int(current.strftime("%w"))
dayOfJourney=int(0)
if(xDay==0 or xDay==6):
dayOfJourney=weekend
else:
dayOfJourney=weekday


d=float(p.get())
if(totalPercentage==100):
mystr=current.strftime("%X")+": Journey Started"
list.insert((index),mystr)
index+=1

totalPercentage=totalPercentage-d
string=(str(totalPercentage)+"% journey remaining")
if(totalPercentage>=0 and totalPercentage<=100):
label1.config(text=string,fg="green")
```

```
d=(d*totalDist)/100
d=distanceCovered(train,d,dayOfJourney)
if(totalPercentage>=0):
current=current+timedelta(hours=d)
myString=current.strftime("%c")+" : by Train"

myString1="Time of arrival is "+current.strftime("%X")
myString2="("+str(100-totalPercentage)+"% of the journey covered.)"
if(totalPercentage>=0):
list.insert(index,myString)
index+=1
global timeLabel1
global timeLabel2
if(totalPercentage>=0):
timeLabel1.config(text=myString1,fg="green")
timeLabel2.config(text=myString2,fg="green")
if(totalPercentage==0):
mystr=current.strftime("%X")+": Journey Finished"
list.insert((index),mystr)
index+=1


def computeBike():
global p
global index
global totalDist
global list
global current
global totalPercentage
global label1
```

```
global bike
global weekday
global weekend
xDay=int(current.strftime("%w"))
dayOfJourney=int(0)
if(xDay==0 or xDay==6):
dayOfJourney=weekend
else:
dayOfJourney=weekday


d=float(p.get())
if(totalPercentage==100):
mystr=current.strftime("%X")+": Journey Started"
list.insert((index),mystr)
index+=1

totalPercentage=totalPercentage-d
string=(str(totalPercentage)+"% journey remaining")
if(totalPercentage>=0 and totalPercentage<=100):
label1.config(text=string,fg="green")
d=(d*totalDist)/100
d=distanceCovered(bike,d,dayOfJourney)
if(totalPercentage>=0):
current=current+timedelta(hours=d)
myString=current.strftime("%c")+" : by Bike"

myString1="Time of arrival is "+current.strftime("%X")
myString2="("+str(100-totalPercentage)+"% of the journey covered.)"

if(totalPercentage>=0):
```

```
list.insert(index,myString)
index+=1
global timeLabel1
global timeLabel2
if(totalPercentage>=0):
timeLabel1.config(text=myString1,fg="green")
timeLabel2.config(text=myString2,fg="green")
if(totalPercentage==0):
mystr=current.strftime("%X")+": Journey Finished"
list.insert((index),mystr)
index+=1


b1=Button(final1,text="Car",command=computeCar)
b2=Button(final1,text="Bus",command=computeBus)
b3=Button(final1,text="Foot",command=computeFoot)
b4=Button(final1,text="Train",command=computeTrain)
b5=Button(final1,text="Bike",command=computeBike)

b1.config(font=large_font)
b2.config(font=large_font)
b3.config(font=large_font)
b4.config(font=large_font)
b5.config(font=large_font)


b1.grid(row=0,column=0)
b2.grid(row=0,column=1)
b3.grid(row=0,column=2)
b4.grid(row=0,column=3)
b5.grid(row=0,column=4)
```

```
p.grid(row=0,column=0)
textP.grid(row=0,column=1)
final2.pack(side="top")
final1.pack(side="top")
font2=('Verdana', 25)
superLabel=Label(graphWindow,text=" ",font=font2)
font3=('Verdana',18)
list=Listbox(graphWindow,height=10,width=30,font=font3)
list.pack(side="bottom")
timeLabel.pack(side="bottom")

edit.add_command(label ='Train Dataset', command = go)

edit.add_command(label ='View Dataset', command = None)

graphWindow.config(menu=menubar)


dist=Frame(graphWindow)
dLabel=Label(dist,text="Distance to be covered")
dLabel1=Label(dist,text="Km")
dLabel.config(font=large_font)
dLabel1.config(font=large_font)

dEntry1=Entry(dist,width=5,textvariable=var1,font=large_font)
dLabel.grid(row=0,column=0)
dEntry1.grid(row=0,column=1)
dLabel1.grid(row=0,column=2)

def compute():
global dist
```

```
global button
global totalDist
global final
global index
global list
global dEntry1
totalDist=float(dEntry1.get())
st=str(totalDist)+"Km"
dist.pack_forget()
button.pack_forget()
superLabel.config(text=st)
superLabel.pack(side="top")
final.pack(side="top")
s=str(remaining)+"%"+" of the journey remaining"
list.insert(index,s)
index+=1


button=Button(graphWindow,text="Compute")
button.config(font=large_font)


cal=Frame(graphWindow)
label1=Label(cal,text="Date in DD/MM/YYYY format: ")
label2=Label(cal,text="/")
label3=Label(cal,text="/")

label1.config(font=large_font)
label2.config(font=large_font)
label3.config(font=large_font)
```

```
day=Entry(cal,width=2,font=large_font)
month=Entry(cal,width=2,font=large_font)
year=Entry(cal,width=4,font=large_font)
label1.grid(row=0,column=0)
day.grid(row=0,column=1)
label2.grid(row=0,column=2)
month.grid(row=0,column=3)
label3.grid(row=0,column=4)
year.grid(row=0,column=5)
timeEntry=Frame(graphWindow)

iLabel=Label(timeEntry,text="Time in HH:MM:SS format: ")
iLabel.config(font=large_font)

hourEntry=Entry(timeEntry,width=2,font=large_font)
hourLabel=Label(timeEntry,text=":")
hourLabel.config(font=large_font)
minEntry=Entry(timeEntry,width=2,font=large_font)
minLabel=Label(timeEntry,text=":")
minLabel.config(font=large_font)
secEntry=Entry(timeEntry,width=2,font=large_font)
iLabel.grid(row=0,column=0)
hourEntry.grid(row=0,column=1)
hourLabel.grid(row=0,column=2)
minEntry.grid(row=0,column=3)
minLabel.grid(row=0,column=4)
secEntry.grid(row=0,column=5)



cal.pack(side="top")
timeEntry.pack(side="top")
```

```
current=datetime.datetime.now()

day.insert(0,current.strftime("%d"))
month.insert(0,current.strftime("%m"))
year.insert(0,current.strftime("%Y"))
hourEntry.insert(0,current.strftime("%H"))
minEntry.insert(0,current.strftime("%M"))
secEntry.insert(0,current.strftime("%S"))

ver=Frame(graphWindow)
lab1=Label(ver,text="")
lab2=Label(ver,text="Date: Unverified", fg="red")
lab3=Label(ver,text="Time: Unverified",fg="red")
lab4=Label(ver,text="")

lab1.config(font=large_font)
lab2.config(font=large_font)
lab3.config(font=large_font)
lab4.config(font=large_font)

lab1.pack(side="top")
lab2.pack(side="top")
lab3.pack(side="top")
lab4.pack(side="top")
ver.pack(side="top")
dCal=int(0)
mCal=int(0)
yCal=int(0)
hourCal=int(0)
minCal=int(0)
```

```
secCal=int(0)
label1=Label(graphWindow,text="")
label1.config(font=large_font)

index=int(0)

def sub():
global cal
global ver
global butFrame
global timeEntry
timeEntry.pack_forget()
cal.pack_forget()
ver.pack_forget()
butFrame.pack_forget()
global dCal
global mCal
global yCal
global label1
global current
global dist
global button
global list
global index
s="Journey starts at "+current.strftime("%H")+":"+current.strftime("%M")+":"+current.strftime("%S")
label1.config(text=s)
label1.pack(side="top")
dist.pack(side="top")
button.config(command=compute)
button.pack(side="top")
```

```
def verifyDate():
global day
global month
global year
global lab1
global lab3
global lab2
global lab4
global dCal
global mCal
global yCal
global butFrame
global current
global hourCal
global minCal
global secCal
global hourEntry
global minEntry
global secEntry
global but
global newBut
dCal=int(day.get())
mCal=int(month.get())
yCal=int(year.get())
hourCal=int(hourEntry.get())
minCal=int(minEntry.get())
secCal=int(secEntry.get())
```

```
current=datetime.datetime(year=yCal,month=mCal,day=dCal,hour=hourCal,minute=minCal,
second=secCal)
str1="Day of the month:"+current.strftime("%A")
str2="Month: "+current.strftime("%B")
str3="Year: "+current.strftime("%Y")

but.pack_forget()
but.grid(row=0,column=1)
newBut.grid(row=0,column=2)
but.config(text="Update",command=verifyDate)
str4=current.strftime("%H")+":"+current.strftime("%M")+":"+current.strftime("%S")


lab1.config(text=str1,fg="green")
lab2.config(text=str2,fg="green")
lab3.config(text=str3,fg="green")
lab4.config(text=str4,fg="green")
butFrame=Frame(graphWindow)
but=Button(butFrame,text="verify",command=verifyDate)
but.config(font=large_font)
newBut=Button(butFrame,text="Submit",command=sub)
newBut.config(font=large_font)
but.pack(side="top")
butFrame.pack(side="top")


daySelected=0
vehicleSelected=0
transport=int(0)
travelDay=int(0)
trans="nothing"
```

```
lBottom4=Label(graphWindow,text="    ",fg="red")
lBottom3=Label(graphWindow,text="  ")
lBottom2=Label(graphWindow,text="   ",fg="red")
lBottom1=Label(graphWindow,text="  ")

lBottom4.config(font=large_font)
lBottom3.config(font=large_font)
lBottom2.config(font=large_font)
lBottom1.config(font=large_font)

lBottom4.pack(side="bottom")
lBottom3.pack(side="bottom")
lBottom2.pack(side="bottom")
lBottom1.pack(side="bottom")

 def enterCar():
global car
global travelDay
global vehicleSelected
global transport
global daySelected
global lBottom1
global lBottom2
global lBottom3
global lBottom4
global trans
trans="car"
vehicleSelected=1
transport=car
if(daySelected==0):
```

```
str="Medium of transport: Car"
str1="Please specify whether the journey commences"
str2="on week day or week end."
lBottom2.config(text=str)
lBottom3.config(text=str1)
lBottom4.config(text=str2)
else:
populate(car,travelDay)
str1="Data analytics for"
str2="Mode of transportation: Car"
str3=" on a "
str4=None
if(travelDay==0):
str4="Weekday"
else:
str4="Weekend"
lBottom1.config(text=str1)
lBottom2.config(text=str2)
lBottom3.config(text=str3)
lBottom4.config(text=str4)


def enterBike():
global bike
global travelDay
global vehicleSelected
global transport
global daySelected
global lBottom1
global lBottom2
global lBottom3
```

```
global lBottom4
global trans
trans="bike"
vehicleSelected=1
transport=bike
if(daySelected==0):
str="Medium of transport: Bike"
str1="Please specify whether the journey commences"
str2="on week day or week end."
lBottom2.config(text=str)
lBottom3.config(text=str1)
lBottom4.config(text=str2)
else:
populate(bike,travelDay)
str1="Data analytics for"
str2="Mode of transportation: Bike"
str3=" on a "
str4=None
if(travelDay==0):
str4="Weekday"
else:
str4="Weekend"
lBottom1.config(text=str1)
lBottom2.config(text=str2)
lBottom3.config(text=str3)
lBottom4.config(text=str4)


def enterBus():
global bus
global travelDay
```

```
global vehicleSelected
global transport
global daySelected
global lBottom1
global lBottom2
global lBottom3
global lBottom4
global trans
trans="bus"
vehicleSelected=1
transport=bus
if(daySelected==0):
str="Medium of transport: Bus"
str1="Please specify whether the journey commences"
str2="on week day or week end."
lBottom2.config(text=str)
lBottom3.config(text=str1)
lBottom4.config(text=str2)
else:
populate(bus,travelDay)
str1="Data analytics for"
str2="Mode of transportation: Bus"
str3=" on a "
str4=None
if(travelDay==0):
str4="Weekday"
else:
str4="Weekend"
lBottom1.config(text=str1)
lBottom2.config(text=str2)
lBottom3.config(text=str3)
```

```
lBottom4.config(text=str4)


def enterTrain():
global train
global travelDay
global vehicleSelected
global transport
global daySelected
global lBottom1
global lBottom2
global lBottom3
global lBottom4
global trans
trans="train"
vehicleSelected=1
transport=train
if(daySelected==0):
str="Medium of transport: Train"
str1="Please specify whether the journey commences"
str2="on week day or week end."
lBottom2.config(text=str)
lBottom3.config(text=str1)
lBottom4.config(text=str2)
else:
populate(train,travelDay)
str1="Data analytics for"
str2="Mode of transportation: Train"
str3=" on a "
str4=None
if(travelDay==0):
```

```
str4="Weekday"
else:
str4="Weekend"
lBottom1.config(text=str1)
lBottom2.config(text=str2)
lBottom3.config(text=str3)
lBottom4.config(text=str4)


def enterFoot():
global foot
global travelDay
global vehicleSelected
global transport
global daySelected
global lBottom1
global lBottom2
global lBottom3
global lBottom4
global trans
trans="foot"
vehicleSelected=1
transport=foot
if(daySelected==0):
str="Medium of transport: Foot"
str1="Please specify whether the journey commences"
str2="on week day or week end."
lBottom2.config(text=str)
lBottom3.config(text=str1)
lBottom4.config(text=str2)
else:
```

```
populate(foot,travelDay)
str1="Data analytics for"
str2="Mode of transportation: Foot"
str3=" on a "
str4=None
if(travelDay==0):
str4="Weekday"
else:
str4="Weekend"
lBottom1.config(text=str1)
lBottom2.config(text=str2)
lBottom3.config(text=str3)
lBottom4.config(text=str4)


f1=Frame(graphWindow)
tCar=Button(f1,text="Car",command=enterCar)
tBike=Button(f1,text="Bike",command=enterBike)
tTrain=Button(f1,text="Train",command=enterTrain)
tFoot=Button(f1,text="Foot",command=enterFoot)
tBus=Button(f1,text="Bus",command=enterBus)

tCar.config(font=large_font)
tBike.config(font=large_font)
tTrain.config(font=large_font)
tFoot.config(font=large_font)
tBus.config(font=large_font)

tCar.grid(row=0,column=0)
tBike.grid(row=0,column=1)
tTrain.grid(row=0,column=2)
```

```
tFoot.grid(row=0,column=3)
tBus.grid(row=0,column=4)

f1.pack(side="top")

def setWeekday():
global travelDay
global daySelected
global vehicleSelected
global lBottom1
global lBottom2
global lBottom3
global lBottom4
global trans
global transport
global weekday
global weekend
travelDay=weekday
daySelected=1
if(vehicleSelected==0):
str1="Travelling on a WeekDay"
str2="Specify mode of transport"
lBottom3.config(text=str1)
lBottom4.config(text=str2)
else:
populate(transport,travelDay)
str1="Data analytics for"
str2="Mode of transportation: "+trans
str3=" on a "
str4=None
if(travelDay==weekday):
```

```
str4="Weekday"
else:
str4="Weekend"
lBottom1.config(text=str1)
lBottom2.config(text=str2)
lBottom3.config(text=str3)
lBottom4.config(text=str4)


def setWeekend():
global travelDay
global daySelected
global vehicleSelected
global lBottom1
global lBottom2
global lBottom3
global lBottom4
global trans
global transport
global weekend
global weekday
travelDay=weekend
daySelected=1
if(vehicleSelected==0):
str1="Travelling on a WeekEnd"
str2="Specify mode of transport"
lBottom3.config(text=str1)
lBottom4.config(text=str2)
else:
populate(transport,travelDay)
str1="Data analytics for"
```

```
str2="Mode of transportation: "+trans
str3=" on a "
str4=None
if(travelDay==weekday):
str4="Weekday"
else:
str4="Weekend"
lBottom1.config(text=str1)
lBottom2.config(text=str2)
lBottom3.config(text=str3)
lBottom4.config(text=str4)

f15=Frame(graphWindow)
tWeekday=Button(f15,text="Week Day",command=setWeekday)
tWeekend=Button(f15,text="Week End",command=setWeekend)
tWeekday.config(font=small_font)
tWeekend.config(font=small_font)

tWeekday.pack(side="top")
tWeekend.pack(side="top")
f15.pack(side="top")



f2=Frame(graphWindow)
l1=Label(f2,text="x")
l1.config(font=small_font)
l2=Label(f2,text="y")
l2.config(font=small_font)
l1.grid(row=0,column=0)
l2.grid(row=0,column=1)
font1=('Verdana',12)
```

```
E=[[None for x in range(2)] for y in range(10)]
for i in range(0,10):
    for j in range(0,2):
        var = StringVar(value='')
        E[i][j]=Entry(f2,width=23,textvariable=var,font=font1)
        E[i][j].grid(row=i+1,column=j)



def populate(veh,day):
global E
global car
global bike
global bus
global foot
global train
global weekend
global weekday
global transport
transport=veh
if(veh==car and day==weekday):
for i in range(0,2):
for j in range(0,10):
E[j][i].delete(0,END)
E[j][i].insert(0,car_weekday[i][j])
if(veh==car and day==weekend):
for i in range(0,2):
for j in range(0,10):
E[j][i].delete(0,END)
E[j][i].insert(0,car_weekend[i][j])
```

```
if(veh==bike and day==weekday):
for i in range(0,2):
for j in range(0,10):
E[j][i].delete(0,END)
E[j][i].insert(0,bike_weekday[i][j])
if(veh==bike and day==weekend):
for i in range(0,2):
for j in range(0,10):
E[j][i].delete(0,END)
E[j][i].insert(0,bike_weekend[i][j])

if(veh==bus and day==weekday):
for i in range(0,2):
for j in range(0,10):
E[j][i].delete(0,END)
E[j][i].insert(0,bus_weekday[i][j])
if(veh==bus and day==weekend):
for i in range(0,2):
for j in range(0,10):
E[j][i].delete(0,END)
E[j][i].insert(0,bus_weekend[i][j])
if(veh==train and day==weekday):
for i in range(0,2):
for j in range(0,10):
E[j][i].delete(0,END)
E[j][i].insert(0,train_weekday[i][j])
if(veh==train and day==weekend):
for i in range(0,2):
for j in range(0,10):
E[j][i].delete(0,END)
E[j][i].insert(0,train_weekend[i][j])
```

```
if(veh==foot and day==weekday):
for i in range(0,2):
for j in range(0,10):
E[j][i].delete(0,END)
E[j][i].insert(0,foot_weekday[i][j])
if(veh==foot and day==weekend):
for i in range(0,2):
for j in range(0,10):
E[j][i].delete(0,END)
E[j][i].insert(0,foot_weekend[i][j])


dataItem=[[float(0) for x in range(10)] for y in range(2)]

 f2.pack(side="top")
def submit():
global graphWindow
global dataItem
global E
global car
global bike
global bus
global train
global foot
global weekday
global weekend
global transport
global car_weekday
global car_weekend
global bike_weekday
global bike_weekend
```

```
global train_weekday
global train_weekend
global bus_weekday
global bus_weekend
global foot_weekday
global foot_weekend

global travelDay
for i in range(0,2):
for j in range(0,10):
yx=E[j][i].get()
dataItem[i][j]=float(yx)
if(transport==car and travelDay==weekday):
car_weekday[i][j]=yx
if(transport==car and travelDay==weekend):
car_weekend[i][j]=yx
if(transport==bike and travelDay==weekday):
bike_weekday[i][j]=yx
if(transport==bike and travelDay==weekend):
bike_weekend[i][j]=yx
if(transport==bus and travelDay==weekday):
bus_weekday[i][j]=yx
if(transport==bus and travelDay==weekend):
bus_weekend[i][j]=yx
if(transport==foot and travelDay==weekday):
foot_weekday[i][j]=yx
if(transport==foot and travelDay==weekend):
foot_weekend[i][j]=yx
if(transport==train and travelDay==weekday):
train_weekday[i][j]=yx
if(transport==train and travelDay==weekend):
```

```
train_weekend[i][j]=yx
trainData(dataItem,transport,travelDay)

xi=[0 for c in range(500)]
yi=[0 for c in range(500)]
for c in range(0,500):
xi[c]=c
yi[c]=distanceCovered(transport,xi[c],weekday)
string="car"
if(transport==car):
string="Car"
if(transport==bike):
string="Bike"
if(transport==train):
string="Train"
if(transport==bus):
string="Bus"
if(transport==foot):
string="Foot"
plt.scatter(yi,xi,marker=".",label=string+" on Weekday",color="green")

for c in range(0,500):
xi[c]=c
yi[c]=distanceCovered(transport,xi[c],weekend)
plt.scatter(yi,xi,marker=".",label=string+" on Weekend",color="blue")
plt.legend()

plt.show()

f3=Frame(graphWindow)
trainDat=Button(f3,text="Train Data",command=submit)
```

```
trainDat.config(font=small_font)
trainDat.pack(side="top")
f3.pack(side="top")

f1.pack_forget()
f15.pack_forget()
f2.pack_forget()
f3.pack_forget()
graphWindow.mainloop()
```

4. REFERENCES

- **BookDown.org**

  https://bookdown.org/ripberjt/qrmbook/introduction-to-multiple-regression.html

- **Wikipedia**

  https://en.m.wikipedia.org/wiki/Non-linear_least_squares

  https://en.m.wikipedia.org/wiki/Linear_algebra