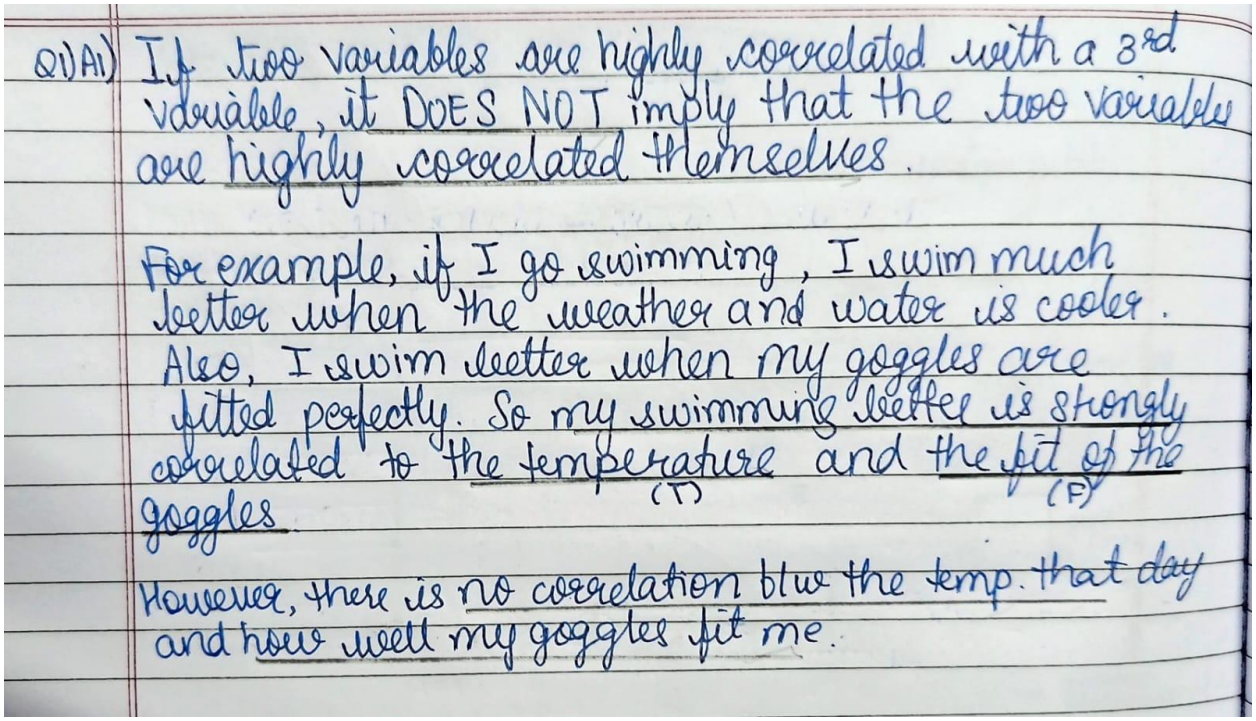
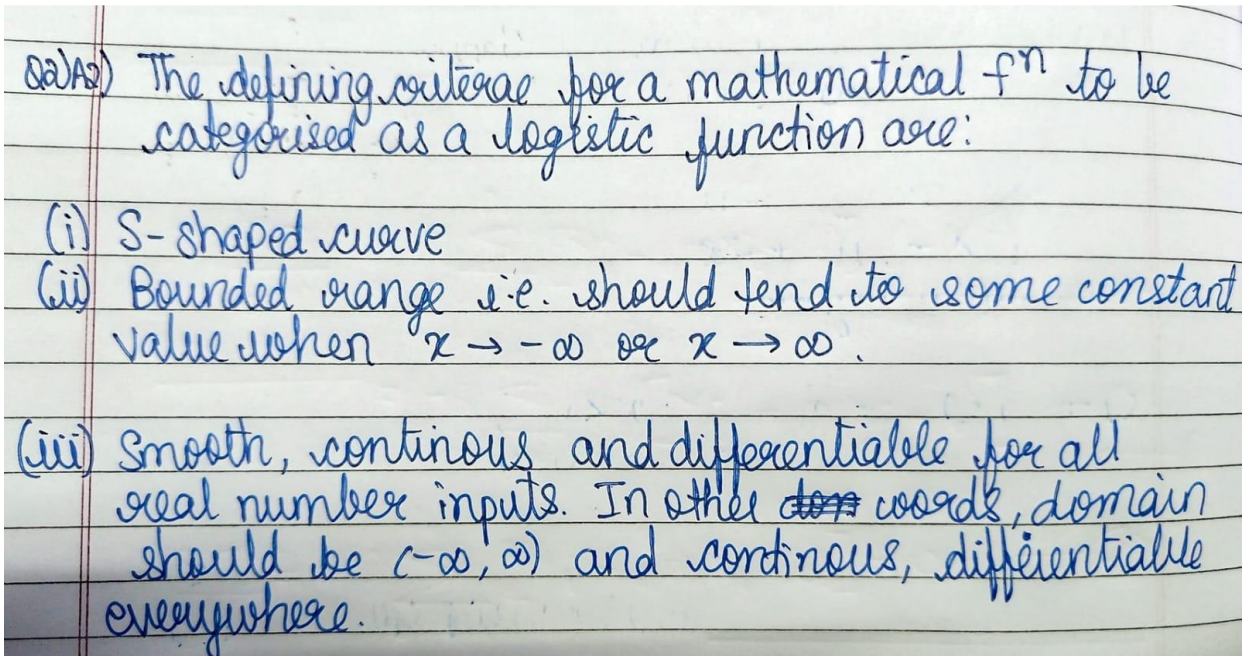
ML Assignment 1 - Report - Arnav Goel (Roll No: 2021519)

Theory Questions

Part A:



Part B:



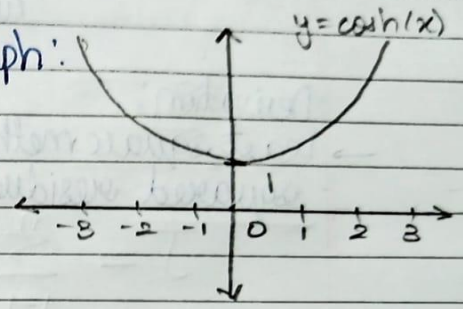
Out of the given functions:

(i) $\sinh(x)$: Domain: $(-\infty, \infty)$
Range: $(-\infty, \infty)$ → NOT BOUNDED
∴ NOT a valid logistic function

(ii) $\cosh(x)$: Domain: $(-\infty, \infty)$
Range: $[1, \infty)$

→ Range is unbounded as $x \to \infty$ and not S-shaped
∴ NOT a valid logistic fn

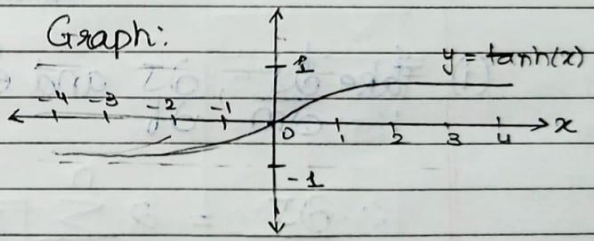
Graph:



(iii) $\tanh(x)$: Domain: $(-\infty, \infty)$
Range: $[-1, 1]$

→ Bounded range, S-shaped and domain, $x \in R$
∴ VALID logistic fn

Graph:



(iv) signum(x): $= \begin{cases} -1 & : x < 0 \\ 0 & : x = 0 \\ 1 & : x > 0 \end{cases}$

∴ Bounded range but not continous and non-diff at $x = 0$

∴ NOT a valid logistic function

Part C:

For very sparse datasets, the technique of 'Leave-one-out' cross validation is beneficial. This is because:

(i) Max. Data Utilisation:
In sparse data, each pt. contains valuable info thus using all (as much as possible for training) makes a better model.

(ii) This leads to better generalisation and robust models.

In K-fold cross validation, we divide the data into 'K' random subsets where we train the model 'K' times each time taking (K-1) randomly as train and the remaining one as test subset.

However in Leave-One-Out (LOO) cross-validation, K=n. That is, only one sample for testing and (n-1) samples for training. This is run 'n' times.

Part E:

Part-e:

$Y = \alpha + \beta x + \epsilon \; ; \; \epsilon = N(0, \sigma)$

∴ The parameters to be estimated are:

(i) α → intercept of reg. line

(ii) β → slope of reg. line

(iii) σ → Std. dev. of error term ϵ

Ans: (A)

Part D:

Part D

We have n data points of the form:

$$(x_i, y_i)$$

Now the least square regression line will be of the form:

$$Y = mX + b$$

We need to derive the coefficients m and b through the process of minimising the cost function or the sum of squared residuals which is:

$$J = \sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n}(Y_i - (mX_i + b))^2$$

We take $\frac{\mathrm{d}J}{\mathrm{d}m}$ and $\frac{\mathrm{d}J}{\mathrm{d}b}$ and equate them to zero to derive these coefficient values:

$$\frac{\mathrm{d}J}{\mathrm{d}m} = -2\sum_{i=1}^{n} X_i(Y_i - (mX_i + b)) = 0$$

$$\frac{\mathrm{d}J}{\mathrm{d}b} = -2\sum_{i=1}^{n}(Y_i - (mX_i + b)) = 0$$

Now taking $\frac{\mathrm{d}J}{\mathrm{d}b}$:

$$\sum_{i=1}^{n} Y_i - \sum_{i=1}^{n} b - \sum_{i=1}^{n} mX_i = 0$$

$$\sum_{i=1}^{n} Y_i - nb - m\sum_{i=1}^{n} X_i = 0$$

$$b = \frac{\sum_{i=1}^{n} Y_i - m\sum_{i=1}^{n} X_i}{n}$$

$$b = \bar{Y} - m\bar{X}$$

Now that we have derived b in terms of m, we are going to look at $\frac{\mathrm{d}J}{\mathrm{d}m}$ and substitute the value of b:

$$\sum_{i=1}^{n} X_iY_i - m\sum_{i=1}^{n} X_i^2 - (\bar{Y} - m\bar{X})\sum_{i=1}^{n} X_i = 0$$

Collecting terms with m:

$$\sum_{i=1}^{n} X_i(Y_i - \bar{Y}) = m\sum_{i=1}^{n} X_i(X_i - \bar{X})$$

$$m = \frac{\sum_{i=1}^{n} X_i(Y_i - \bar{Y})}{\sum_{i=1}^{n} X_i(X_i - \bar{X})}$$

Taking the corrected sum of squares of X:

$$m = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

$$b = \bar{Y} - m\bar{X}$$

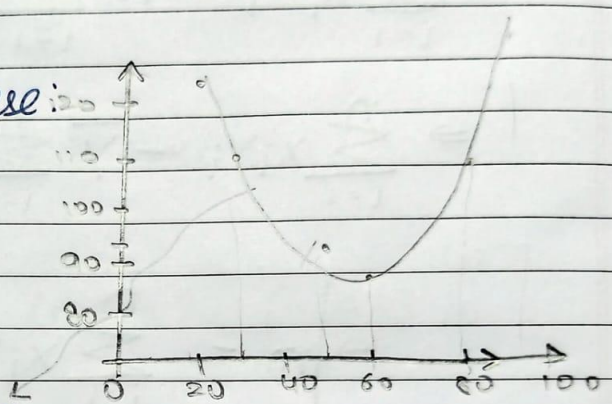
where X_i is the i^{th} sample in our data, Y_i is the corresponding i^{th} output value of the sample, $\bar{X}$ is the mean of all X_i and $\bar{Y}$ is the mean of all Y_i.

Part F:

Part-f:

X	Y
20	125
30	110
50	95
60	90
80	110
90	130

→ Plotting these:



this curve is a good model estimate as the value of 'y' dec. till an x_0 and then inc ⇒ UPWARD FACING PARABOLA

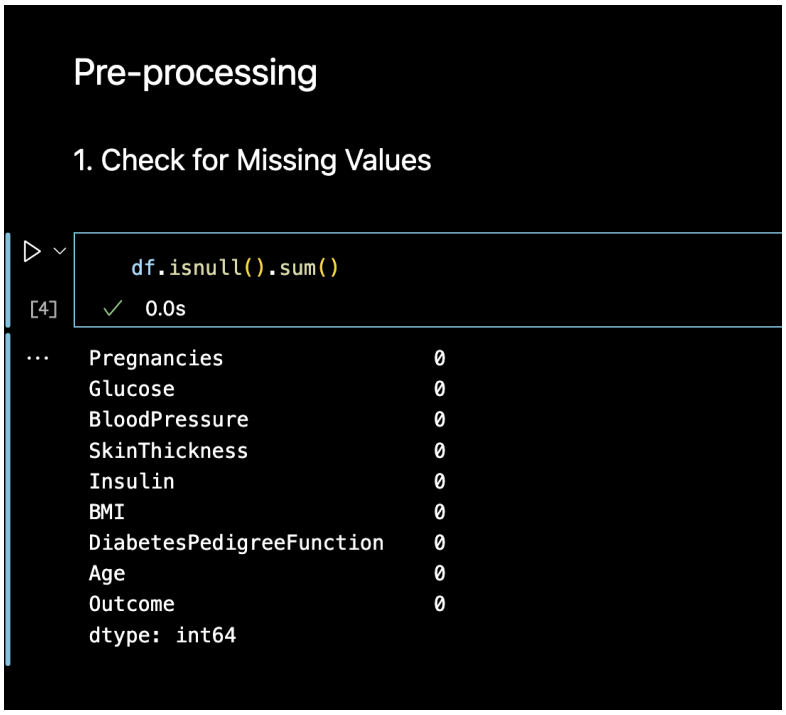
∴ The most likely is: $Y = \alpha + \beta_1 X + \beta_2 X^2 + \epsilon$; $\beta_2 > 0$ (∵ upward facing)

Ans: (D)

Programming Questions
Question 2:

Part A and B
We first performed pre-processing on the given dataset to us i.e.

- **Check for Missing Values:** There were no missing values so we just moved ahead.



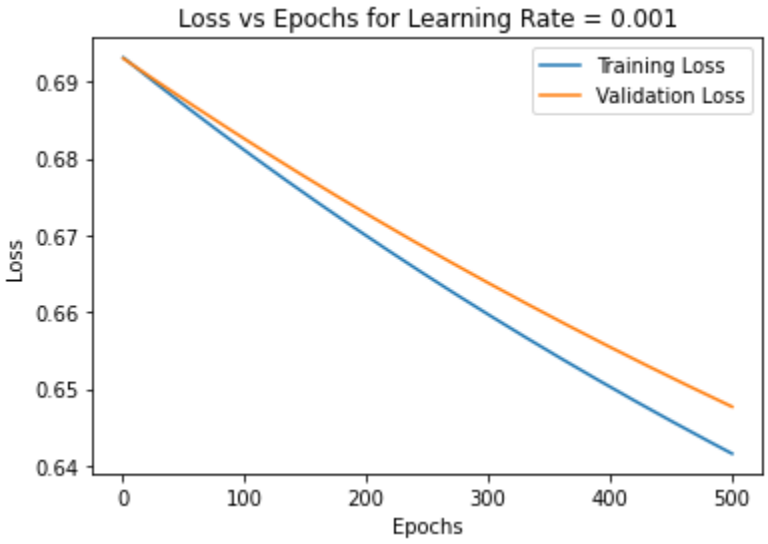
- **Standardise/Scale the data** using Z-Score Normalisation
- Since all the features of the dataset are numerical in nature, we don't need to perform any encoding the data and the data is ready for processing after performing the above pre-processing steps.
- We then divide the dataset into a **70:20:10 ratio for training, testing and validation.**

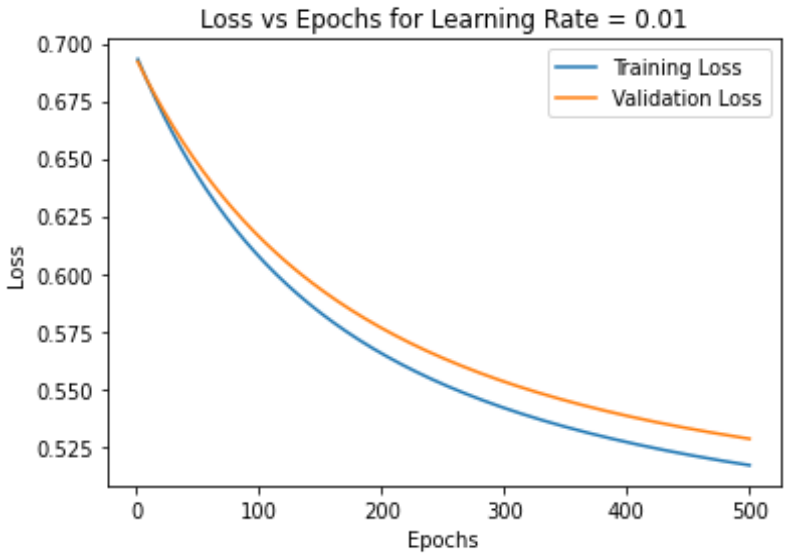
We then implement the training function for logistic regression from scratch with Stochastic Gradient Descent: (Code Explanation)

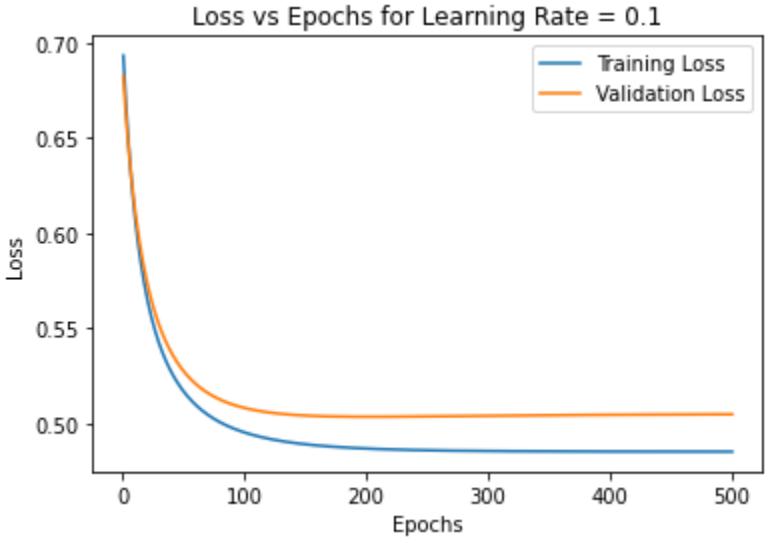
- We initialize the the weights to 0 and the bias to 0.
- We setup arrays for storing the training loss, training accuracy, validation loss and validation accuracy at each epoch.
- We then calculate the gradients for the points and updates the weight and bias (Gradient Descent)
- Using the weights and bias as a dummy model, we make predictions on the training and validation data.

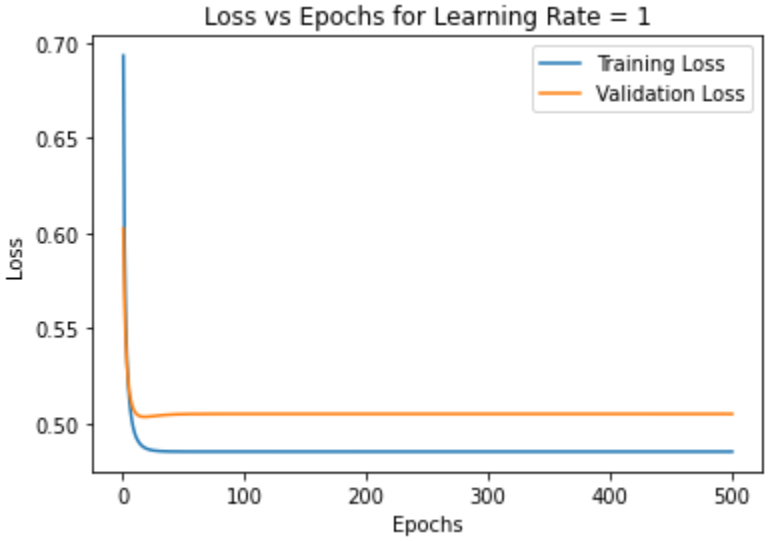
- I calculated the training loss, training accuracy, validation loss and validation accuracy using our defined functions and store them in our list.
- I tried 4 different learning rates i.e. **0.001, 0.01, 0.1 and 1** and for each learning rate train our model for **500 epochs.**
- I tried various different threshold values for our classifier **i.e. 0.3, 0.35, 0.4, 0.45 and 0.50** and found that the **threshold of 0.45** was giving the most accurate results. The plots are for the same threshold:

Loss vs Epochs Plot (Training and Validation Loss) for the 4 learning rates:







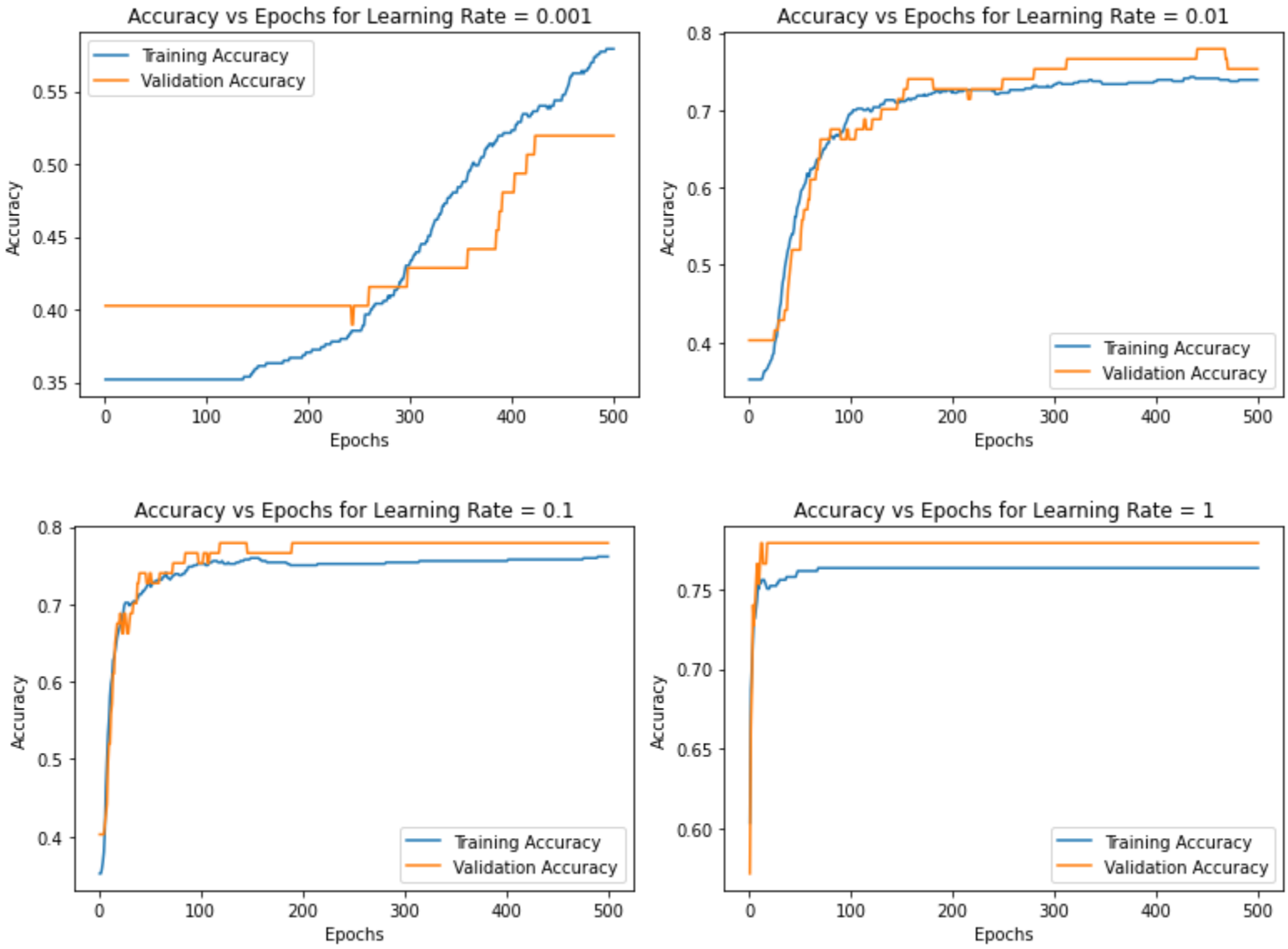


Comment on Convergence based on these Plots:

Note that these plots depict the **cross-entropy loss** on the training set and the validation set at each epoch. As we can see clearly that our losses converge very rapidly with learning rates of **0.1**

(around 200 epochs) and 1 (around 25 epochs). However for lower learning rates, loss tends to be high and does not converge even after 500 epochs. For both the learning rates of 0.1 and 1, final converged loss tends to be almost the same around 0.5. We can also analyse the effect on the accuracy through the plots given below:

Accuracy vs Epochs Plot (Training and Validation Accuracy) for the 4 learning rates:



As we can see from the accuracy vs loss plots as well, they mimic the behaviour of the loss vs epoch plots when it comes to convergence and attain higher accuracies in the cases of learning rates of **0.1 and 1.**

Part C (Evaluation Metrics)

- Here firstly, I calculated the predictions of each of these 4 models on our testing dataset.
- Then I wrote my own custom function to calculate the confusion matrix, accuracy, precision, recall and F1 scores based on the formulae discussed in class.

Learning Rate	0.001	0.01	0.1	1
Accuracy	0.558	0.801	0.811	0.818
Precision	0.412	0.672	0.711	0.711
Recall	0.979	0.770	0.667	0.667
F1 Score	0.580	0.718	0.688	0.688

- The confusion matrices can be found in the code files for each learning rate for this part.

Part D (Regularisation)

For this part we set the following hyperparameters:

1. Epochs : 200
2. Learning Rate: 0.1
3. Threshold: 0.45

L1 Regularisation

- Here only two things change in the original code for Logistic Regression which utilised Stochastic Gradient Descent.
- We add a linear penalty to our loss function and subsequently to our gradient descent function

i.e. $\lambda \sum_{i=1}^{n} (|w_i|)$

L2 Regularisation

- Here only two things change in the original code for Logistic Regression which utilised Stochastic Gradient Descent.
- We add a penalty to our loss function and subsequently to our gradient descent function i.e.

$\lambda \sum_{i=1}^{n} ({w_i}^2)$

- We experiment with different values of the regularisation parameter or lambda. The values we experiment are as follows:

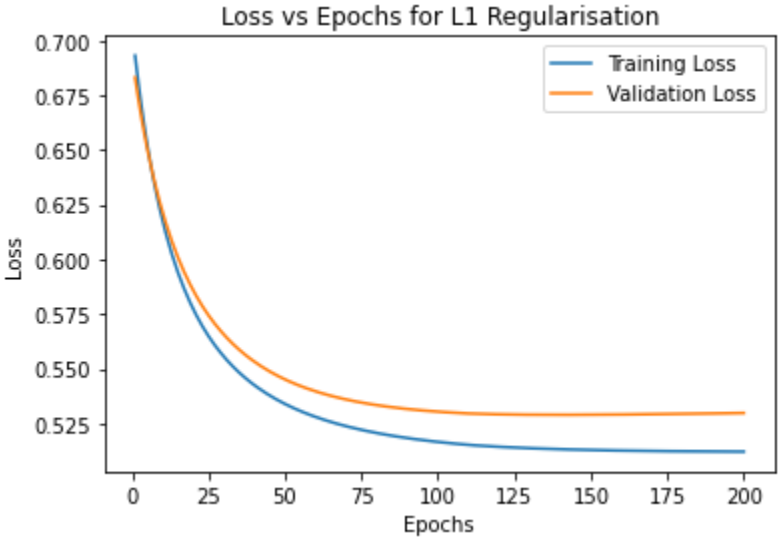
i) 0.001
ii) 0.01
iii) 0.1

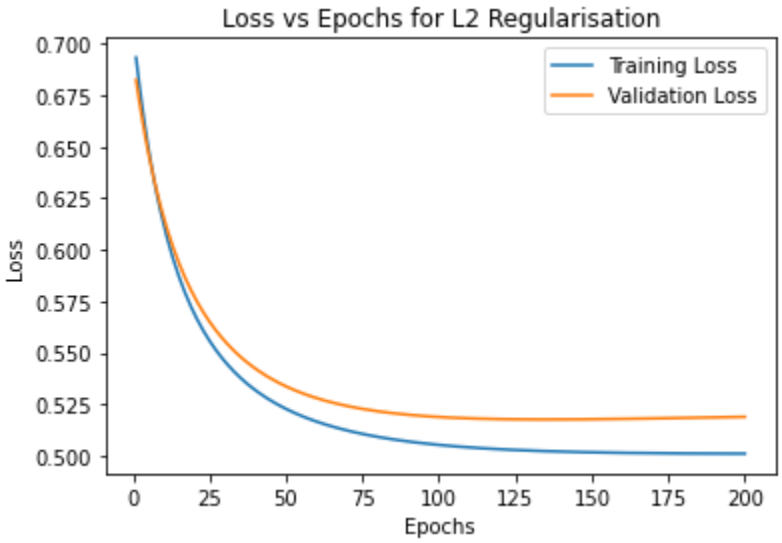
iv) 1

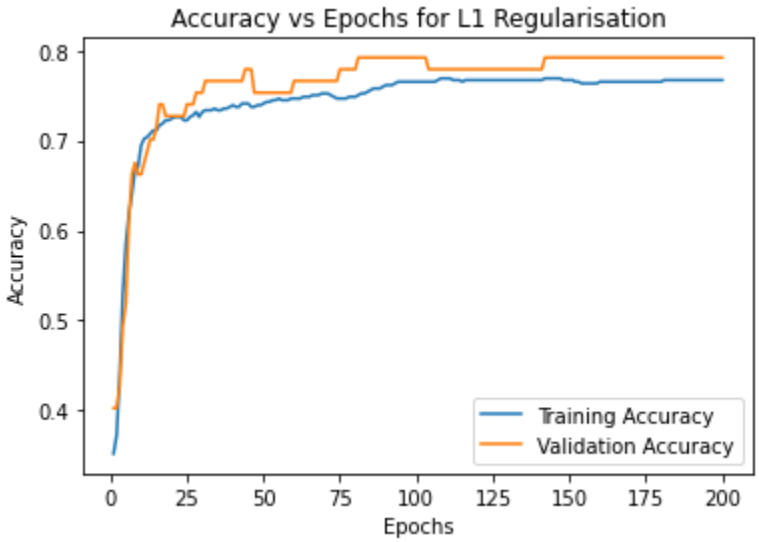
Plots: (Both Loss vs Epoch and Accuracy vs Epoch)

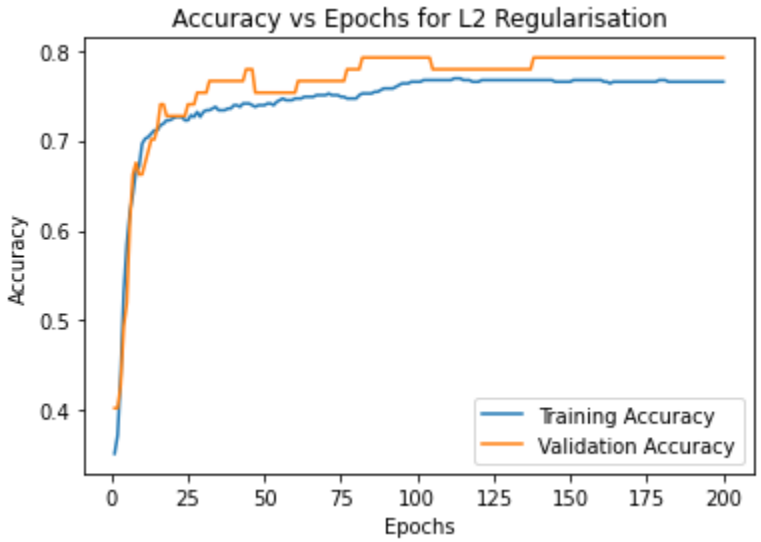
- Training and Validation Loss and Accuracy are in the same plot

Regularisation Parameter: 0.01



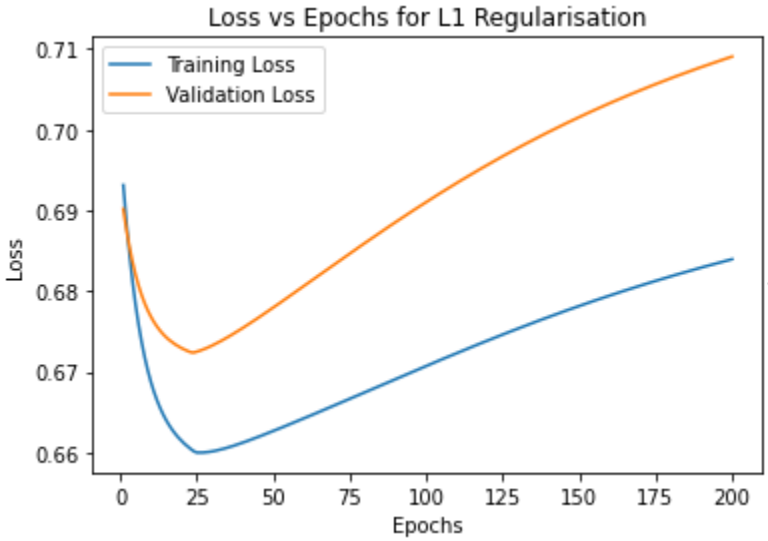


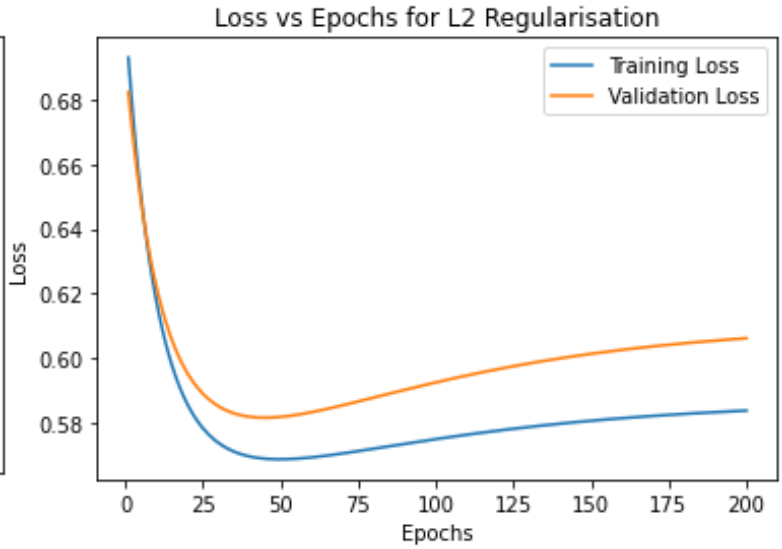


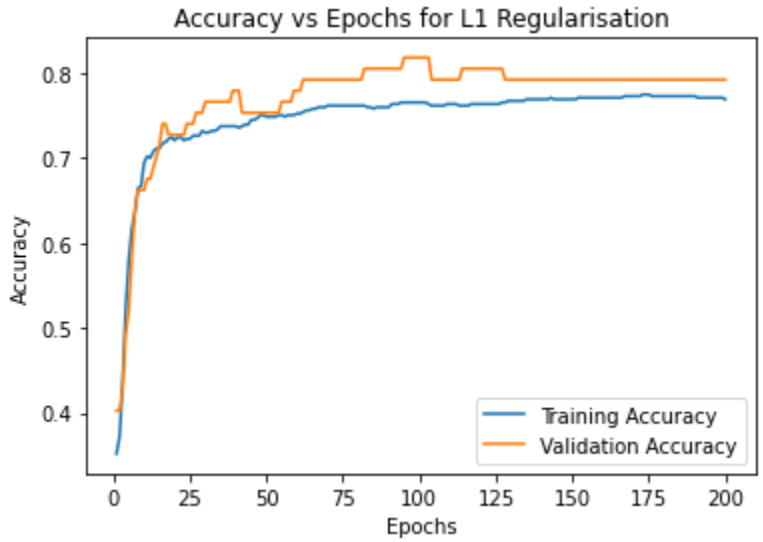


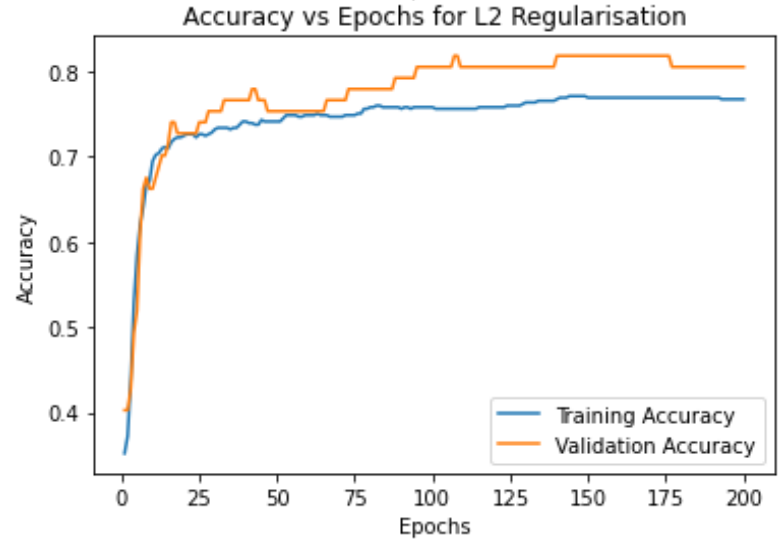
- At this value, both plots are more or less similar and training accuracy clearly plateaus out early indicating the use of regularisation to prevent overfitting.

Regularisation Parameter: 0.1

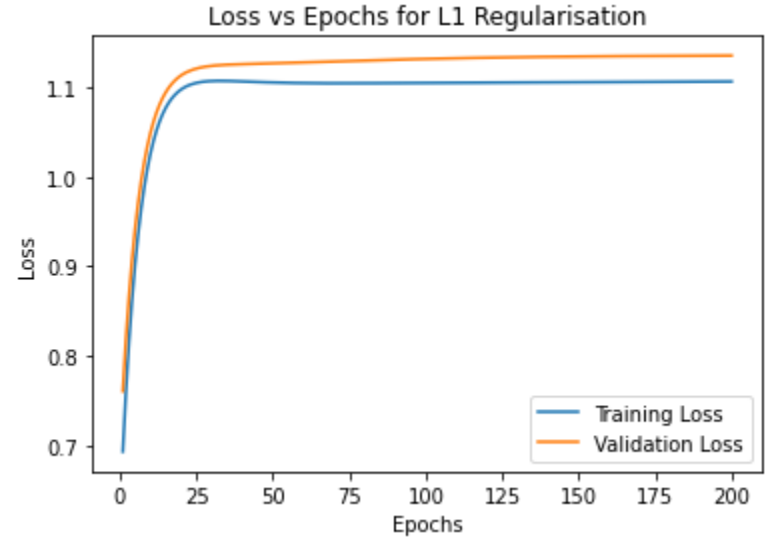


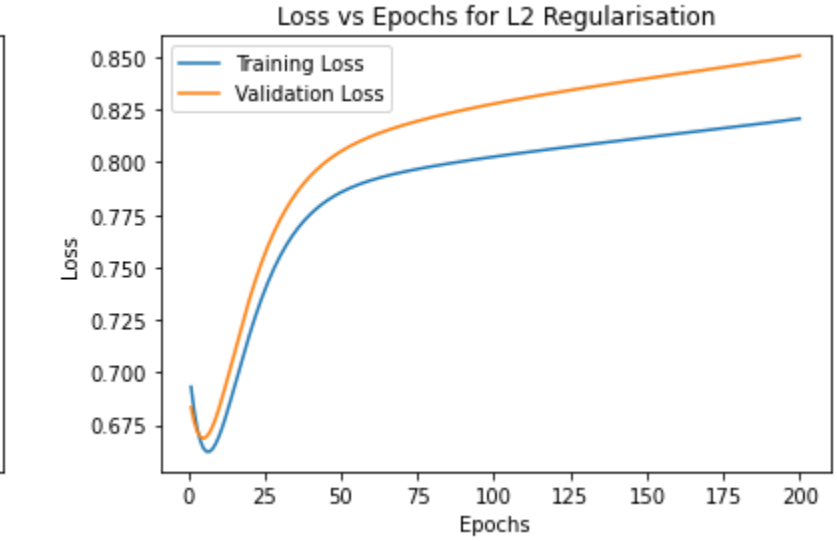


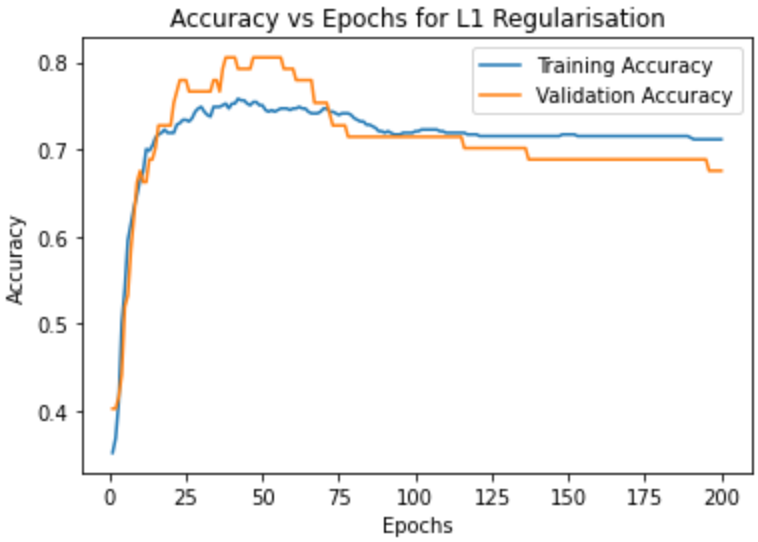


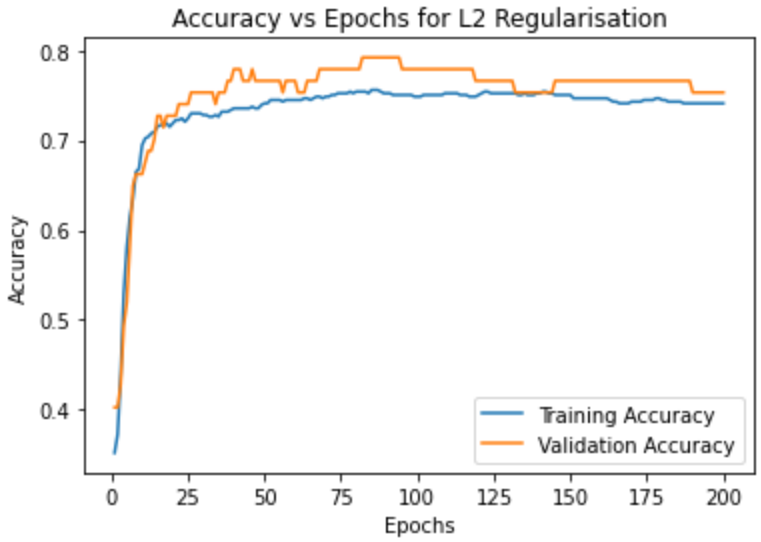


Regularisation Parameter: 1









- On analysing the plots we can see that increasing the value of the regularisation parameter to a number like 1 causes the accuracies to eventually fall on the validation set too especially with L1 Regularisation.
- However too small a value like 0.01, diminishes the role of regularisation in both L1 and L2 and it is not that powerful in stopping overfitting.
- Thus the more appropriate value in my opinion is somewhere in between as can be seen for the plots with regularisation parameter equal to **0.1.**

Part E (Tanh Activation)

- We use the `tanh()` function instead of `sigmoid`.
- The loss function we use for this is still the cross entropy loss as $-\log(f(x))$is always a convex function and thus continuing with cross entropy will still give us a convex loss function.
- However, since $\tanh(x)$maps from **(-∞, ∞)** → **[-1, 1]**, the original cross-entropy function needs to be tweaked a little.
- Therefore, the modified binary cross-entropy loss function becomes:

$$L = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}(\ln(\frac{1+\tanh(x^{(i)})}{2})) - (1-y^{(i)})(\ln(\frac{1-\tanh(x^{(i)})}{2})))$$

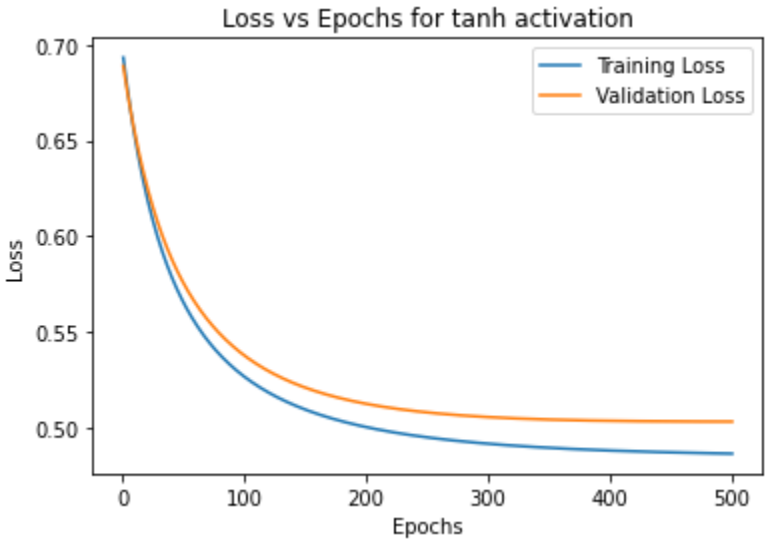
- As our loss function changes, so does our derivate changing our gradient descent formulation (upon simplification of the derivative):

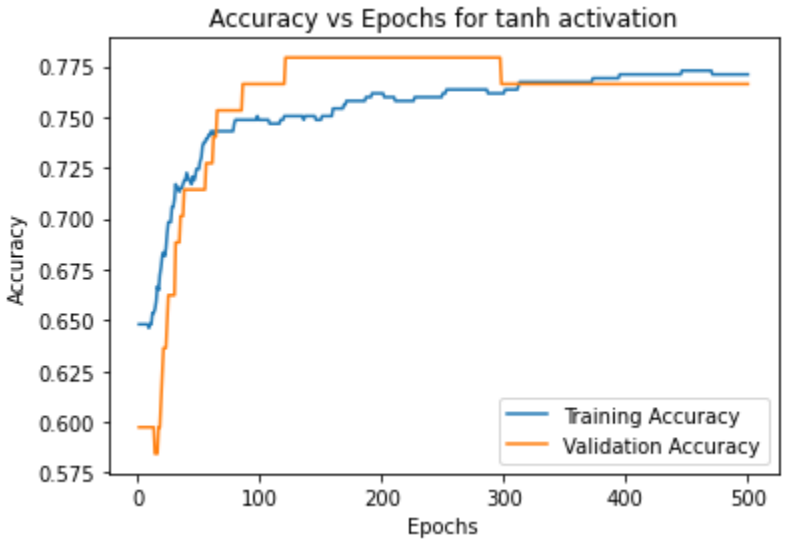
$$\frac{\mathrm{d}L}{\mathrm{d}w_j} = -\frac{1}{m}\sum_{i=1}^{m}(2y^{(i)} - \tanh(x^{(i)}) - 1)(x_j^{(i)})$$

-

$$\frac{\mathrm{d}L}{\mathrm{d}b} = -\frac{1}{m}\sum_{i=1}^{m}(2y^{(i)} - \tanh(x^{(i)}) - 1)$$

- **Hyperparameters:**
  i) **Epochs: 500**
  ii) Learning Rate: 0.01
  iii) Threshold: 0.45
  The loss vs epochs and the accuracy vs epochs plot for tanh:





- If we compare this **loss vs epochs plot for tanh activation** with the same learning rate of 0.01 and normal logistic regression, we notice that tanh activation causes the loss function to converge very quickly.
- If the learning rate is 0.01, the loss function did not even converge after 500 epochs with normal sigmoid logistic regression. However replacing sigmoid with tanh made it converge in about 200 epochs.
- This faster convergence is at a very minimal difference in accuracy which is a very positive sign.

Part F (Mini-Batch Gradient Descent)

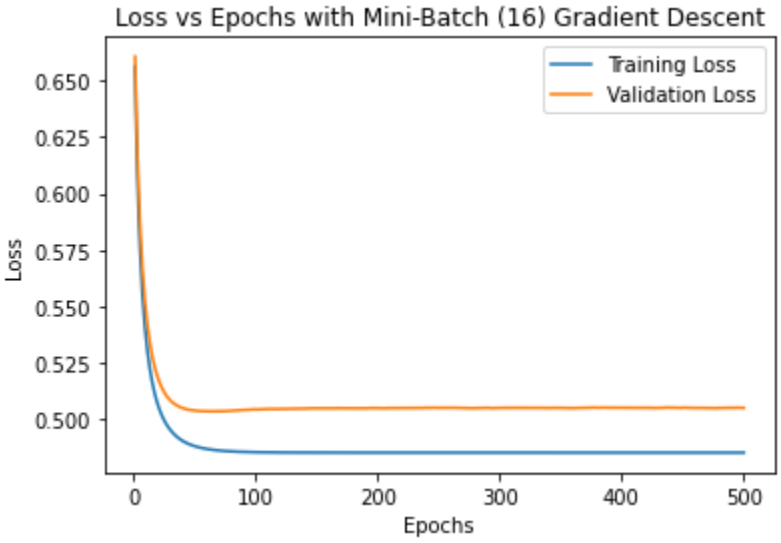
- Here we divided our training data into batches (the size of which we vary) and apply gradient descent batch-wise to update the weights.

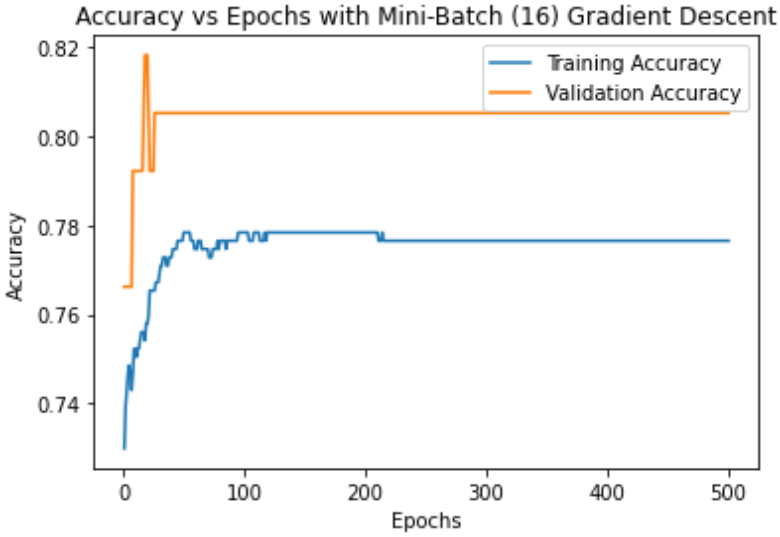
- Hyperparameters:
i) **Epochs: 500**
ii) Learning Rate: 0.01
iii) Threshold: 0.45

- We experiment with the following batch-size values:
i) 16
ii) 32
iii) 64
iv) 256

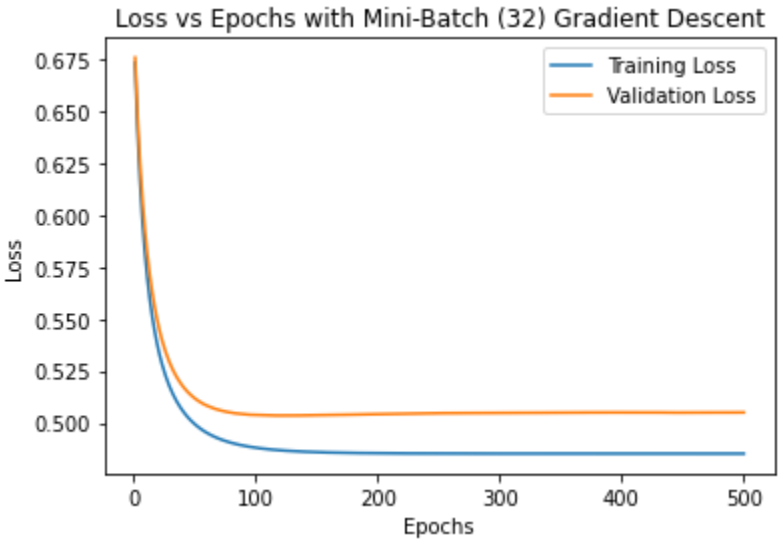
Plots: (Loss vs Epochs and Accuracy vs Epochs) - for each Batch Size

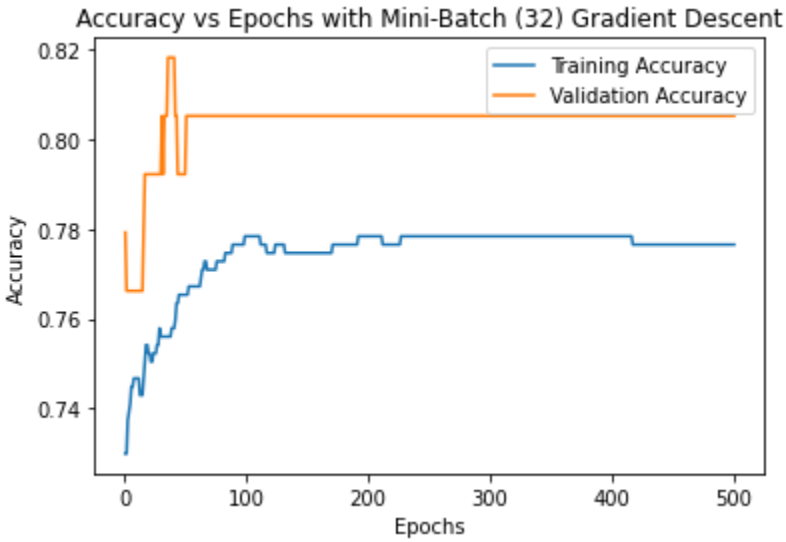
Batch Size: 16



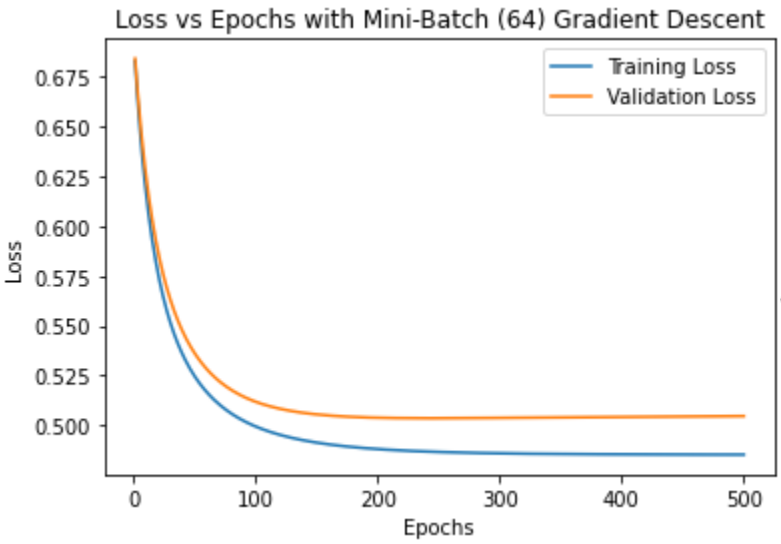


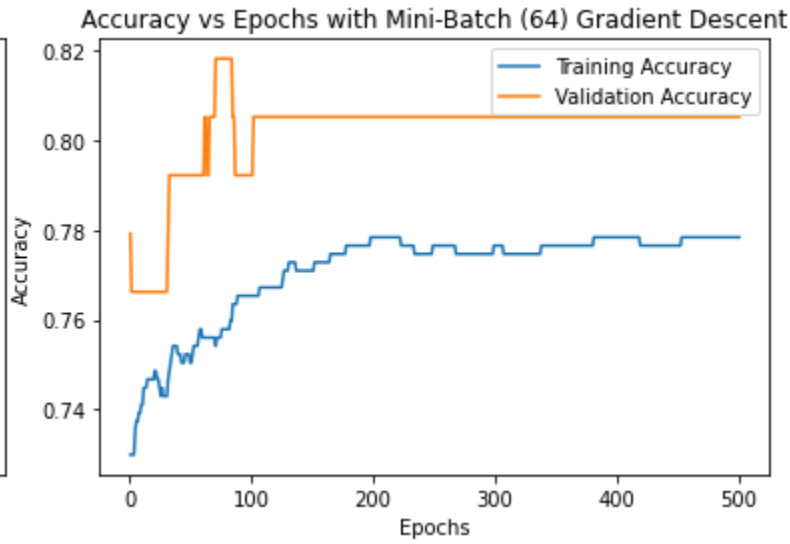
Batch Size: 32



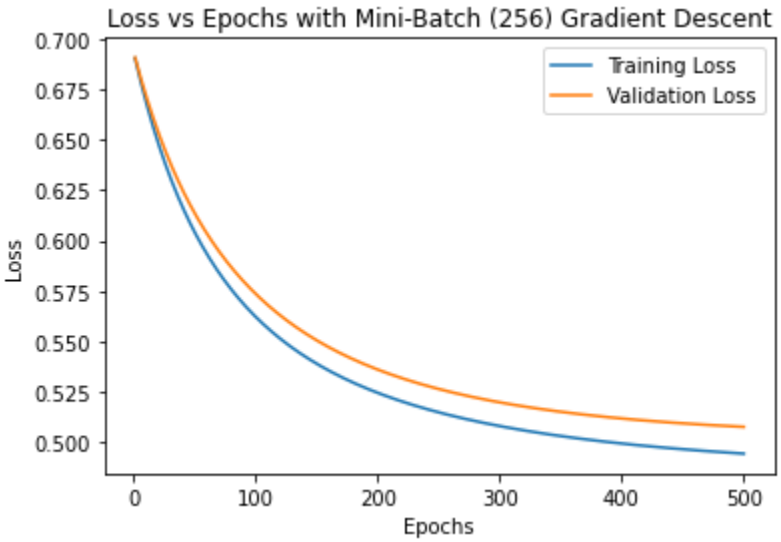


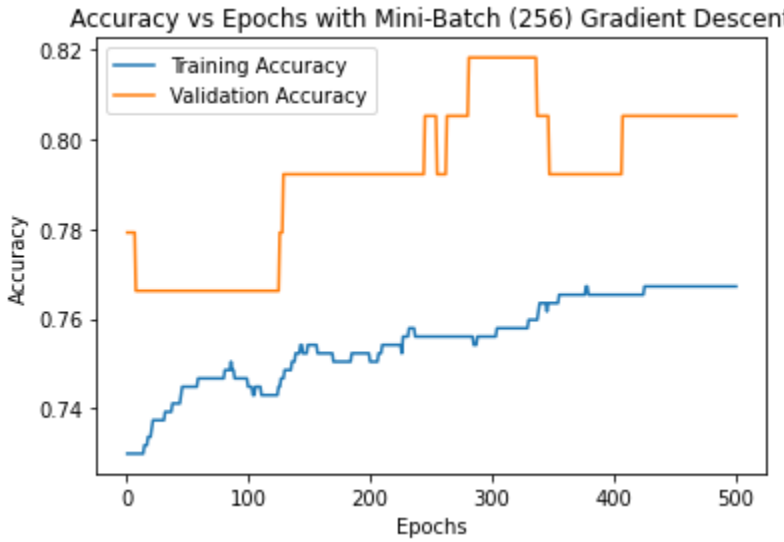
Batch Size: 64





Batch Size: 256





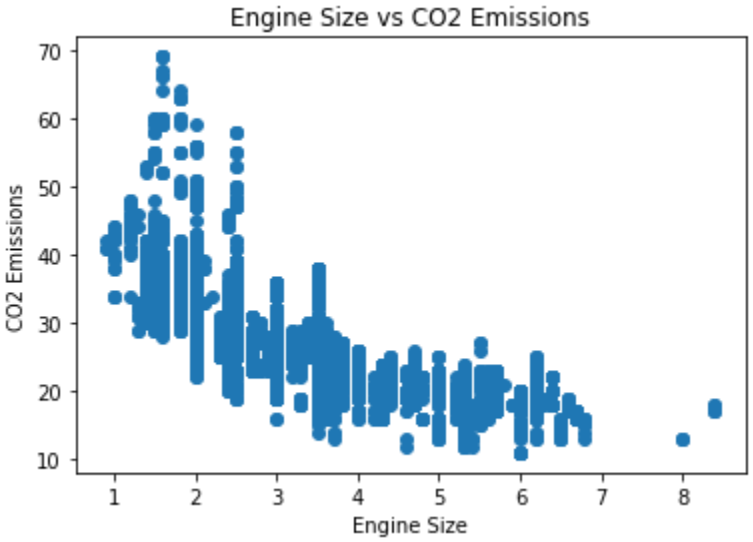
- Upon analysing the plots, we can easily decipher that as we increase the batch size the performance of mini-batch Gradient Descent equates Stochastic Gradient Descent with respect to convergence.
- Smaller batch sizes such as 16 and 32 converge the loss function way quicker than Stochastic Gradient Descent at the same hyperparameters like learning rate.
- For example, from the plots shown above at a **learning rate of 0.01, SGD cannot converge in even 500 epochs.** However **Mini-Batch GD with batch size 16** converges in only about **50 epochs** with a minimal difference in accuracy. Thus we can say that this leads to **faster convergence.**

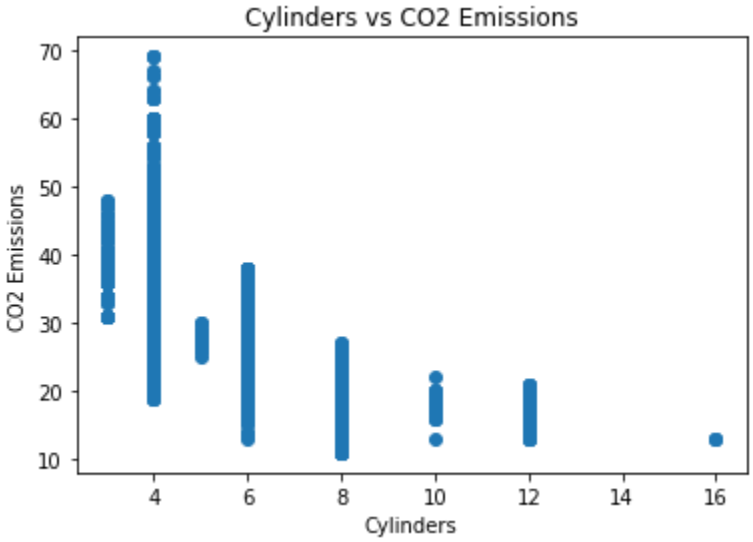
Question 3

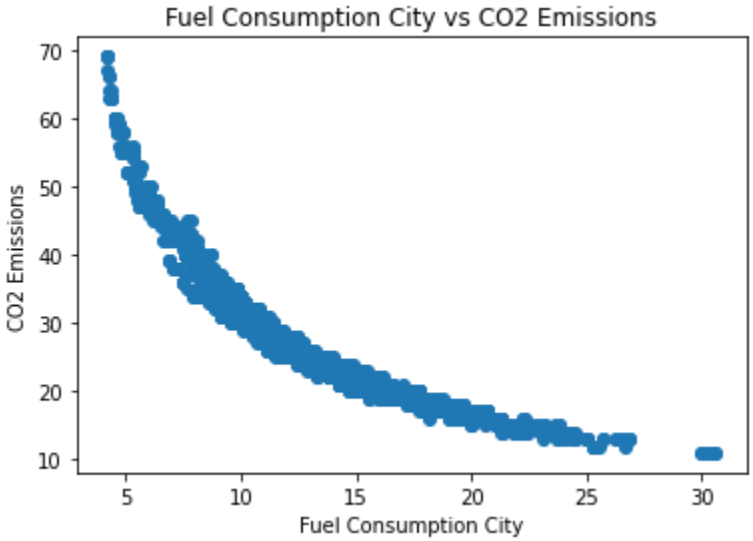
Part A (Exploratory Data Analysis - EDA):

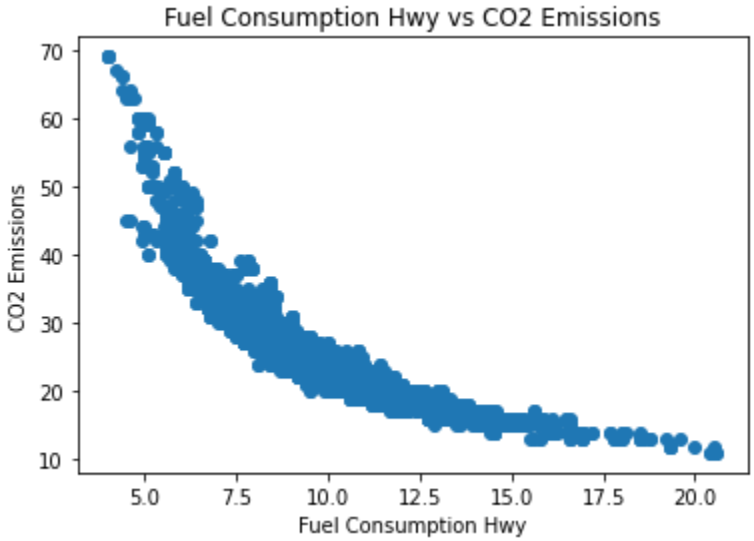
- We first open the dataset and divides the features into categorical and non-categorical features.
- We visualise the non-categorical features through the following plots:

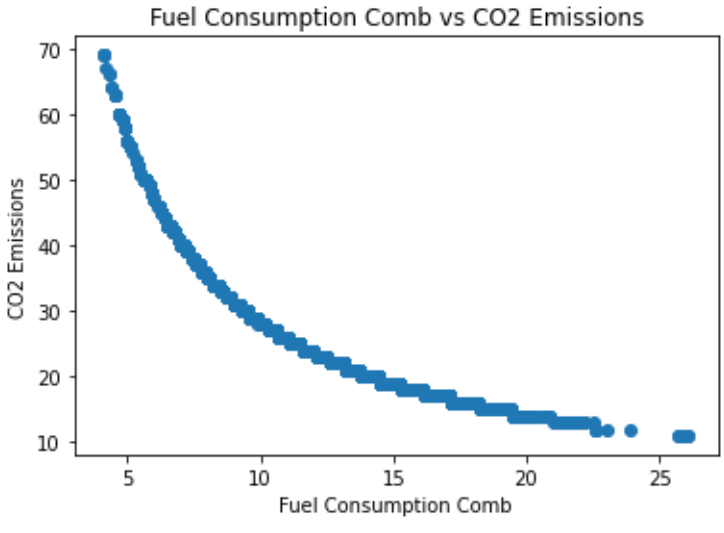
1. Scatter Plots:

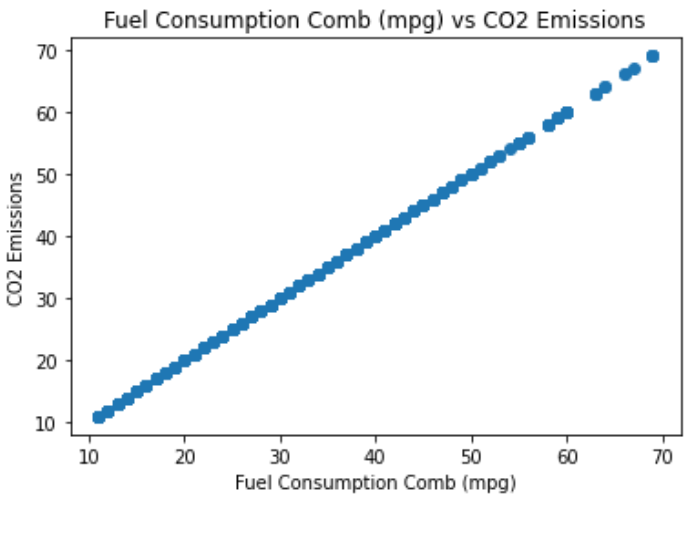




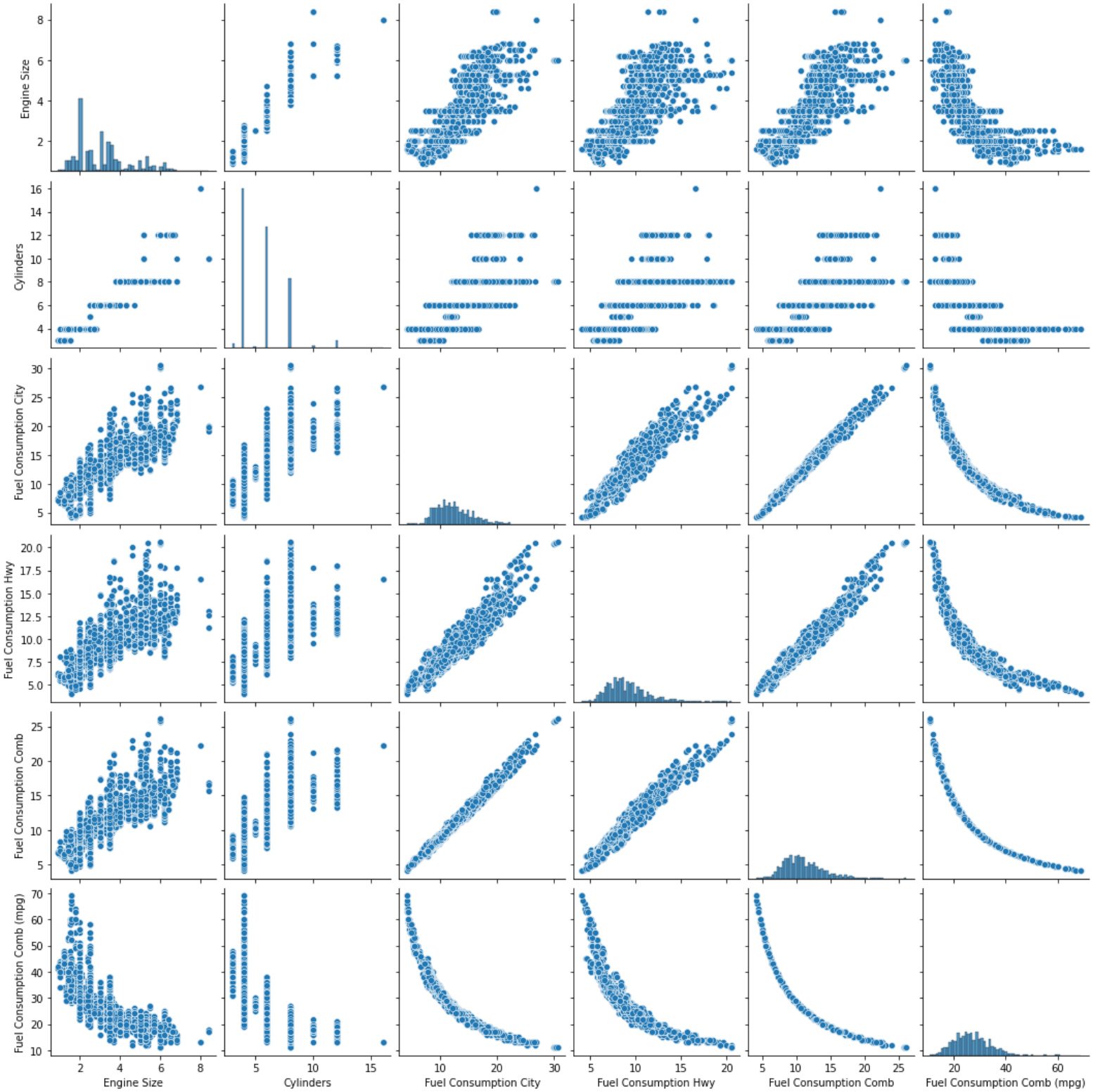




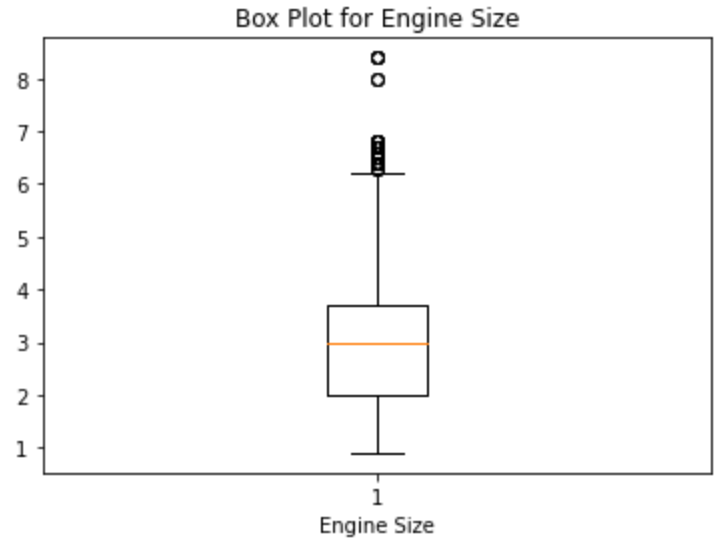


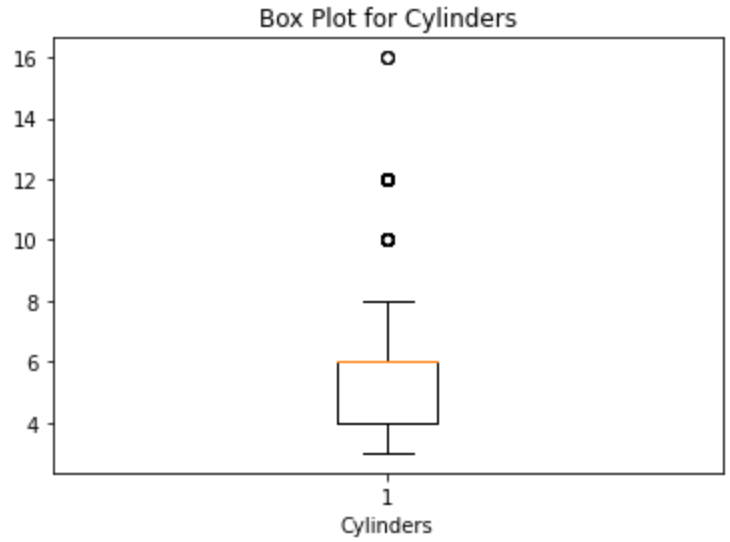


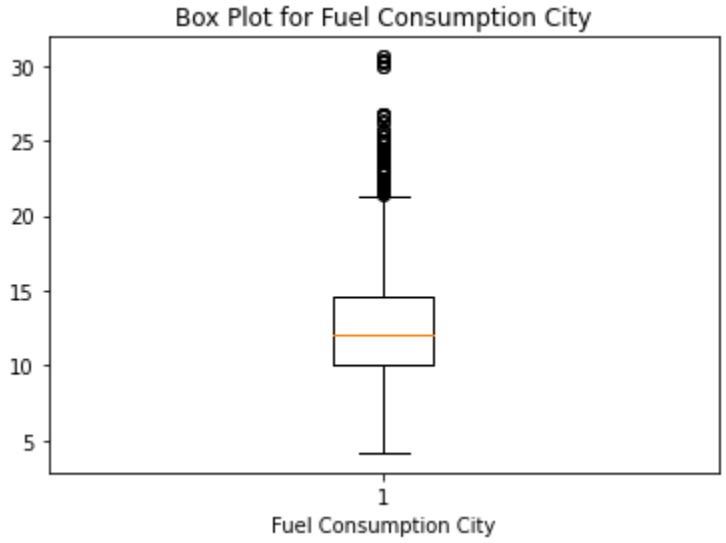
2. Pair Plots

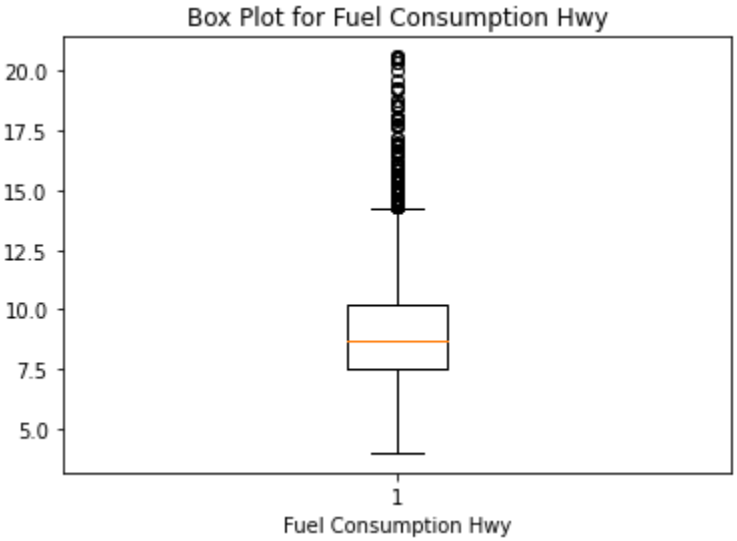

Engine Size
Cylinders
Fuel Consumption City
Fuel Consumption Hwy
Fuel Consumption Comb
Fuel Consumption Comb (mpg)

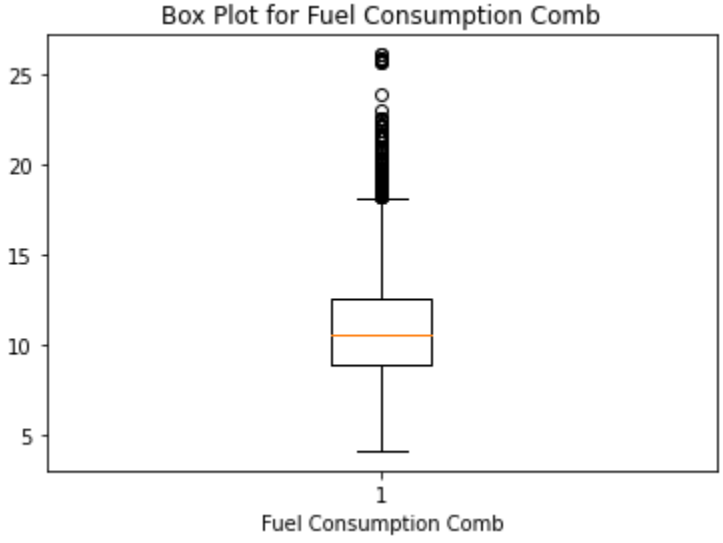
3. Box Plots:

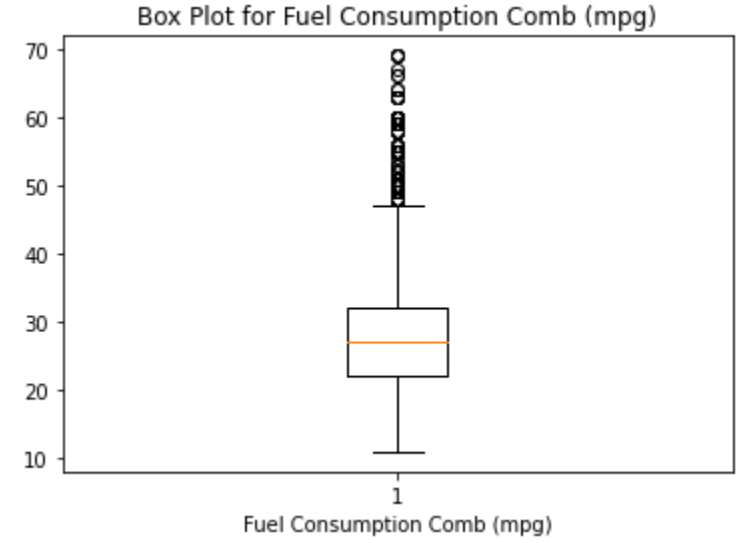

Box Plot for Engine Size
8
7
6
5
4
3
2
1
1
Engine Size


Box Plot for Cylinders
16
14
12
10
8
6
4
1
Cylinders


Box Plot for Fuel Consumption City
30
25
20
15
10
5
1
Fuel Consumption City

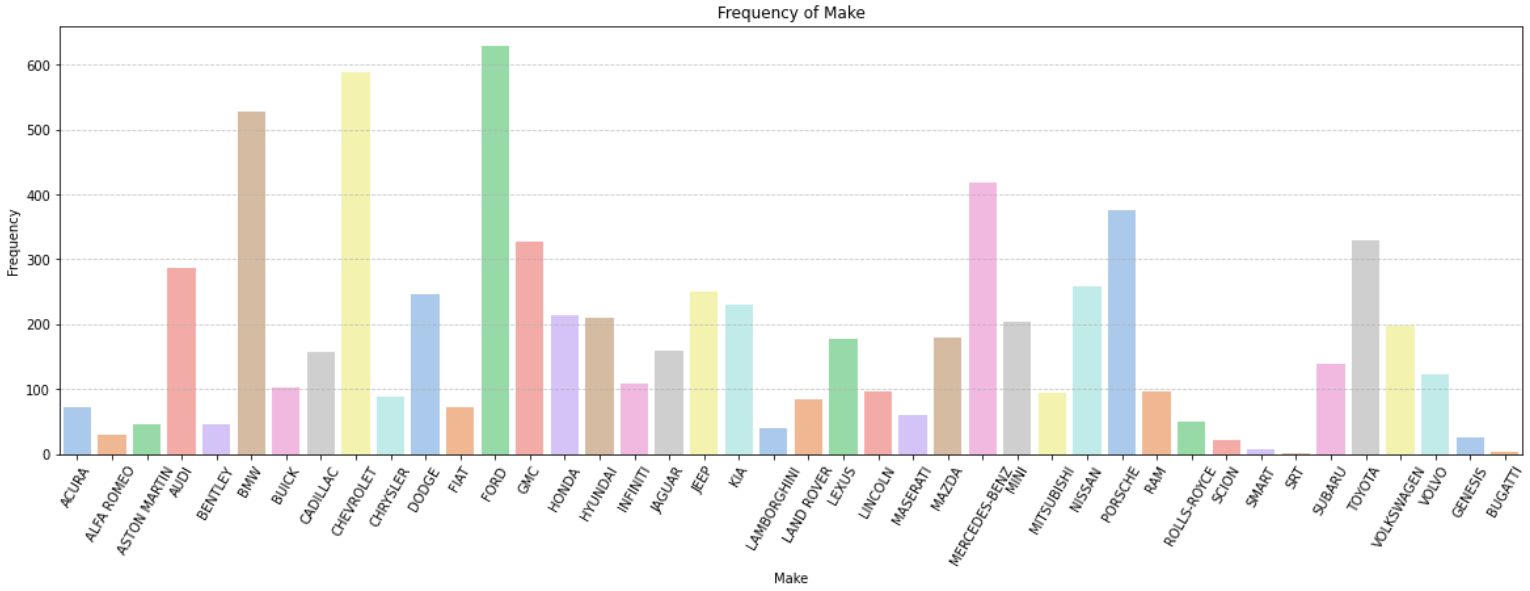

Box Plot for Fuel Consumption Hwy
20.0
17.5
15.0
12.5
10.0
7.5
5.0
1
Fuel Consumption Hwy

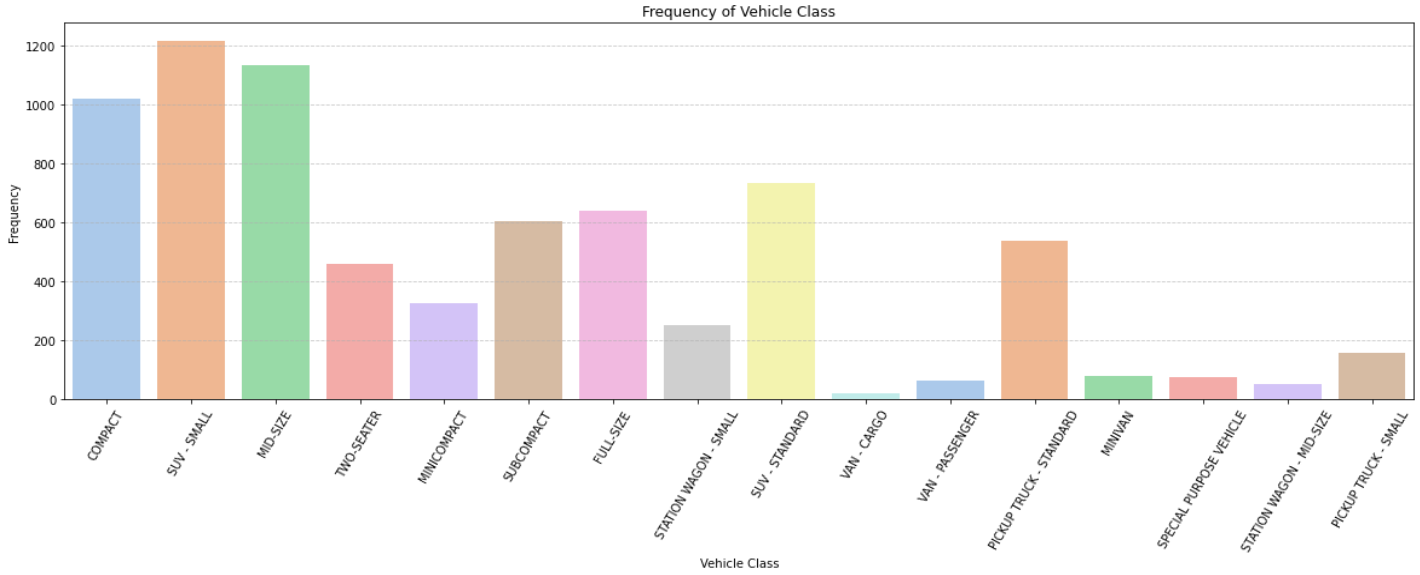

Box Plot for Fuel Consumption Comb
25
20
15
10
5
1
Fuel Consumption Comb

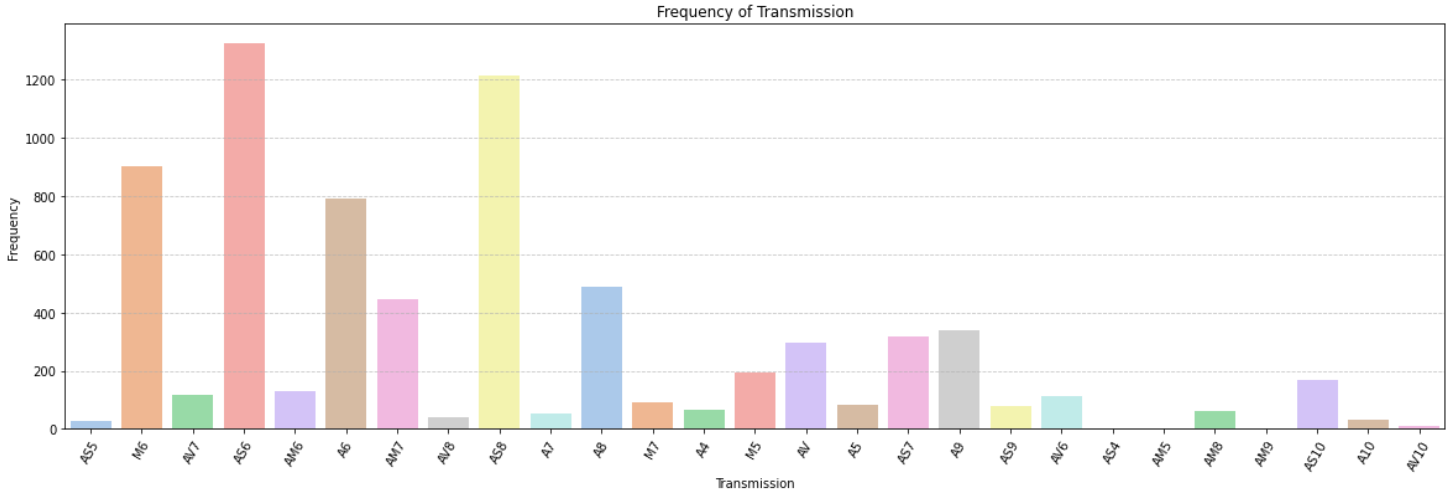

Box Plot for Fuel Consumption Comb (mpg)
70
60
50
40
30
20
10
1
Fuel Consumption Comb (mpg)

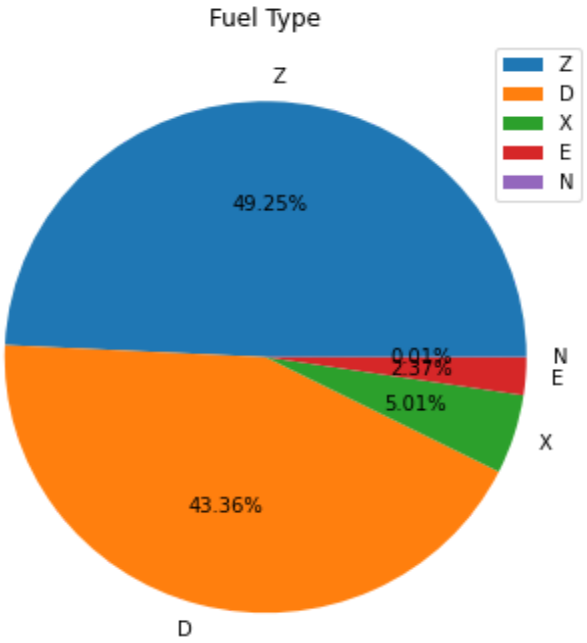
4. Correlation Heatmap

- Now we visualize the categorical features using Histograms and Pie-Charts:




Frequency of Vehicle Class
Frequency
Vehicle Class
COMPACT
SUV - SMALL
MID-SIZE
TWO-SEATER
MINICOMPACT
SUBCOMPACT
FULL-SIZE
STATION WAGON - SMALL
SUV - STANDARD
VAN - CARGO
VAN - PASSENGER
PICKUP TRUCK - STANDARD
MINIVAN
SPECIAL PURPOSE VEHICLE
STATION WAGON - MID-SIZE
PICKUP TRUCK - SMALL


Frequency of Transmission
Frequency
Transmission


Fuel Type
Z
D
X
E
N
49.25%
43.36%
5.01%
2.37%
0.01%

- Based on this EDA performed by me on the dataset, I have made the following **5 insights on the data:**

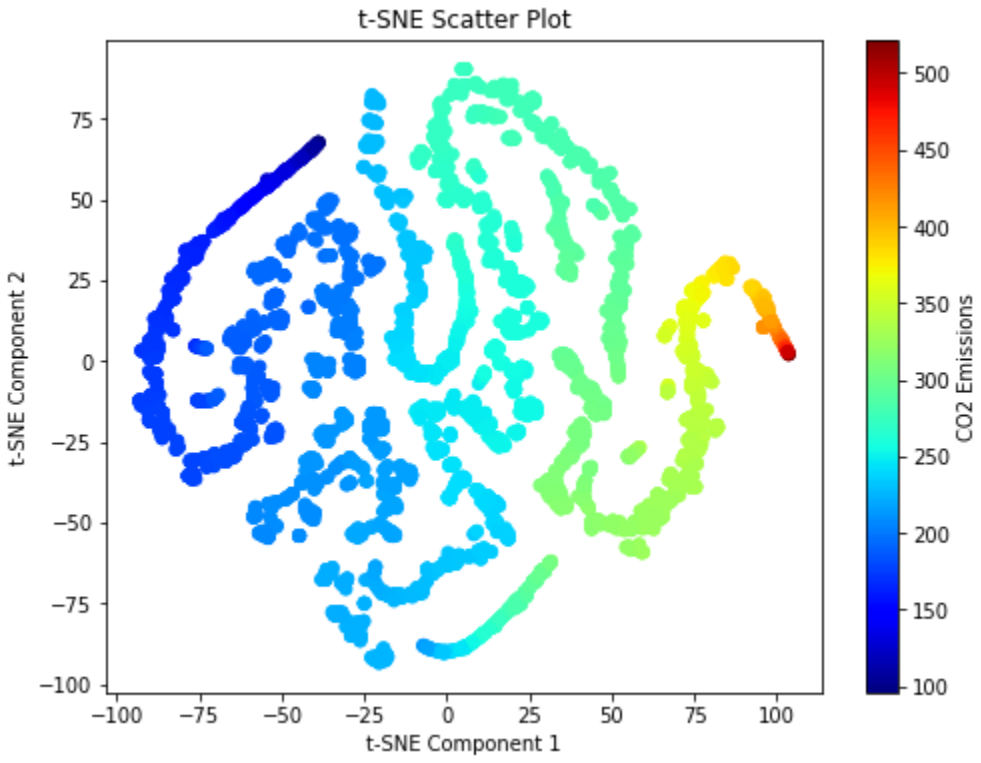
1. There is a clear majority of people using **Z and D type fuels as almost 92%** of the samples use these 2 fuel types only.
2. We see a very strong **positive correlation (0.93)** between the engine size and the number of cylinders which is also very intuitive.
3. **Fuel consumption comb (mpg) shows a negative correlation** with all the other features
4. **Cars with lesser (4) cylinders leader to highest CO2 Emissions** whereas cars with higher no of cylinders (16) lead to lesser CO2 Emissions
5. Almost **50% of the respondents prefer smaller, compact or mid-size vehicle classes.**

Part B (t-SNE Separation):

- Taking only the non-categorical features (i.e. numeric features), I perform t-SNE based dimensionality reduction on the dataset to reduce it to 2 dimensions.
- I plot the 2 on a scattermap with colour based on the value of corresponding CO2 emissions:

Commenting on the Separability:

- As we can see from the scatter plots, same ranges of CO2 values are found together in a cluster like formation.
- Red indicating highest CO2 emissions are only on the extreme right and if we start threading along the plot, we see a gradual decrease in the CO2 values as we progress towards the left or lesser values of t-SNE component 1,
- Thus we can say that t-SNE does a very good job at separating the classes in this case ranges of CO2 values from one another.



Part C (Label-based Encoding and Linear Regression):

- We perform label-based encoding on the categorical features by using Sklearn's Label Encoder.
- We then perform pre-processing on the data by checking for missing values, replacing them with the mean of the column and then subsequently scaling the data using Z-Score normalization.

- Following this we perform a 80:20 train-test split on the data
- We then perform Linear Regression using Sklearn's in-built implementation of linear regression.
- We make predictions on the testing and training dataset. We then calculate the following evaluation metrics:

i) MSE - from Sklearn
ii) RMSE - sqrt of MSE
iii) R2 - from Sklearn
iv) Adjusted R2

$$= 1 - \frac{(1 - R^2) * (N - 1)}{(N - K - 1)}$$

v) MAE - from Sklearn

The metric values on the training dataset and testing dataset are as follows:

```
Testing Data Accuracy:
Mean Squared Error (MSE):  295.3046695124161
Root Mean Squared Error (RMSE):  17.184431020910065
R2 Score:  0.9141463205390631
Adjusted R2 Score:  0.9135016854031789
Mean Absolute Error (MAE):  11.176630024206952

Training Data Accuracy:
Mean Squared Error (MSE):  285.9853836055968
Root Mean Squared Error (RMSE):  16.911102377006557
R2 Score:  0.9163430991931094
Adjusted R2 Score:  0.9161870228856339
Mean Absolute Error (MAE):  10.971129887345167
```

Part D (PCA and Linear Regression):

- We perform Principal Component Analysis using Sklearn to perform dimensionality reduction on our categorically encoded and pre-processed data.
- We then run it for PCs = 4, 6, 8 and 10 as asked and calculate the evaluation metrics as stated for the training and testing data. It is presented in the table given below:

1. Training Data - Evaluation Metrics

Evaluation Metric	PCs = 4	PCs = 6	PCs = 8	PCs = 10
MSE	303.748	300.355	287.013	286.010
RMSE	17.42	17.331	16.941	16.911
R2	0.911	0.912	0.916	0.916
Adjusted-R2	0.911	0.912	0.915	0.916
MAE	11.742	11.740	10.999	10.973

2. Testing Data - Evaluation Metrics

Evaluation Metric	PCs = 4	PCs = 6	PCs = 8	PCs = 10
MSE	313.453	306.528	297.171	295.404
RMSE	17.704	17.507	17.238	17.187
R2	0.908	0.911	0.913	0.914
Adjusted-R2	0.908	0.910	0.913	0.913
MAE	11.989	11.889	11.212	11.179

Part E (One-Hot Encoding):

- We take the categorical features from the original data and we apply one-hot encoding on them through **Sklearn's One-Hot Encoder** and then concatenate back the one-hot encoded categorical features with the non-categorical features.
- We then apply standard pre-processing which we applied in Part C.
- One-Hot Encoding increases the number of features in the data from just **11 to 2149!!**
- We then train a linear regression model on the one-hot encoded dataset and evaluate the given metrics after making predictions on the test dataset.
- The results are as follows:

```
Mean Squared Error (MSE):  1.0891751243635011e+30
Root Mean Squared Error (RMSE):  1043635532340434.0
R2 Score:  -3.166549725012001e+26
Adjusted R2 Score:  6.94476581592528e+26
Mean Absolute Error (MAE):  270462981971696.2
```

- The results are clearly quite messed up compared to categorical encoded dataset from Part C and the possible reason could be the exponential rise in the number of dimensions in the dataset.

Part F (One-Hot Encoding + PCA):

- We perform PCA with the following values of Principal Components:

1. PCs = 10
2. PCs = 50
3. PCs = 100
4. PCs = 200
5. PCs = 400

1. Training Data - Evaluation Metrics

Evaluation Metric	PCs = 10	PCs = 50	PCs = 100	PCs = 200	PCs = 400
MSE	464.828	320.53	176.53	140.067	150.261
RMSE	21.551	17.903	13.286	11.853	12.258
R2	0.864	0.906	0.948	0.959	0.956
Adjusted-R2	0.863	0.905	0.947	0.957	0.952

MAE	14.905	12.777	10.001	8.849	9.246

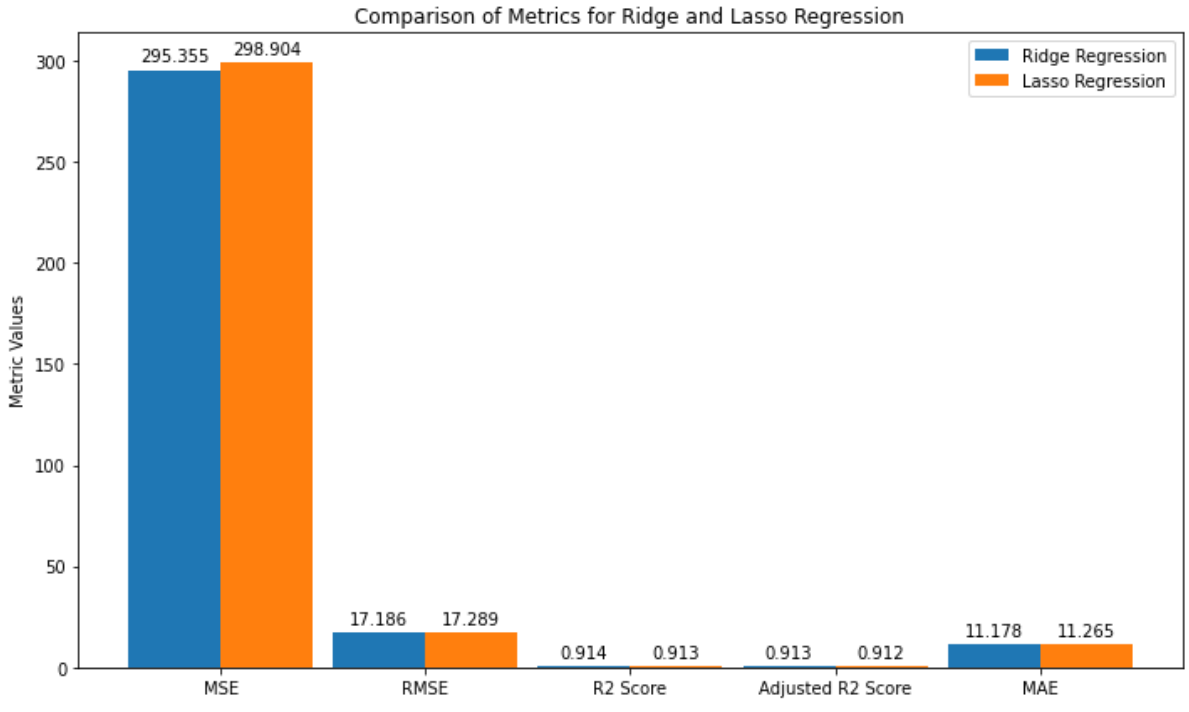
2. Testing Data - Evaluation Metrics

Evaluation Metric	PCs = 10	PCs = 50	PCs = 100	PCs = 200	PCs = 400
MSE	475.191	339.489	184.543	157.001	186.635
RMSE	21.798	18.425	13.584	12.530	13.661
R2	0.861	0.901	0.946	0.954	0.945
Adjusted-R2	0.860	0.897	0.942	0.947	0.925
MAE	14.941	12.912	10.096	9.145	10.067

- We see that the One-Hot Encoded Dataset when reduced to lower dimensions performs very well with a very good R2 score and a very low RMSE score.
- These scores are much better than the scores achieved by Linear Regression with Categorical Encoding whether that be with or without PCA.
- However there is a catch, the R2 score increases very well till 200 Principal Components and decreases when we have 400 Principal Components. The same holds true for other evaluation metrics.
- This validates our hypothesis that one-hot encoding due to high-dimensional data causes a poor performance on these metrics. Thus the ideal range for PCs would be between 200 and 300 PCs.

Part G (L1 and L2 Regularisation):

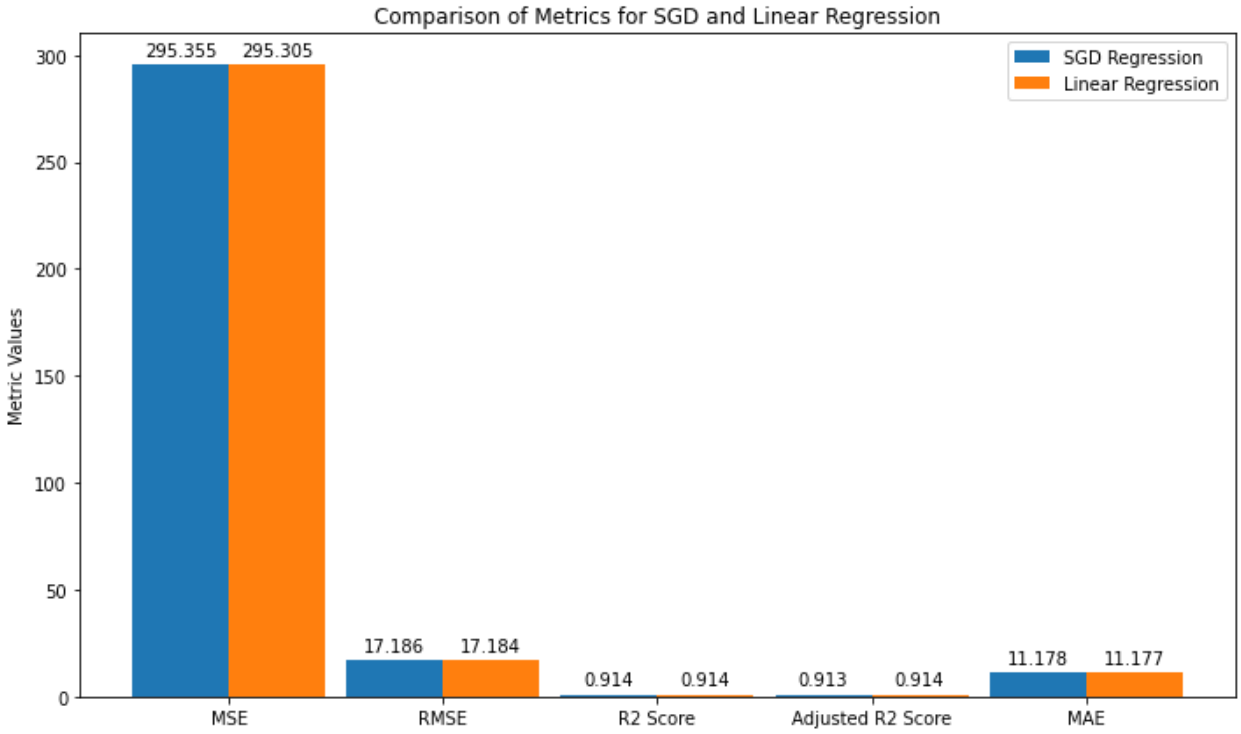
- We compare Lasso (L1) and Ridge (L2) Regression using Sklearn's default implementation and running it on the categorically encoded dataset from Part C.



- The comparison in results with the two techniques are captured by the bar-graph given above.
- Clearly Ridge (R2) Regression performs much better than Lasso (L1) Regression. Note that these scores are on the testing dataset.

Part H (SGD Regressor):

Use the SGDRegressor from Sklearn and run the regression on the same dataset as Linear Regression and compared the results using a bar-graph:



Both given almost similar results and good performance on the testing dataset.