Student Name: K.K.D.Kariyawasam
Student ID: 200289U

EN3150 Assignment 01

Learning from data and related challenges and linear models for regression

---

1 Data preprocessing

1. Index number without letters and leading zeros - 200289
2.

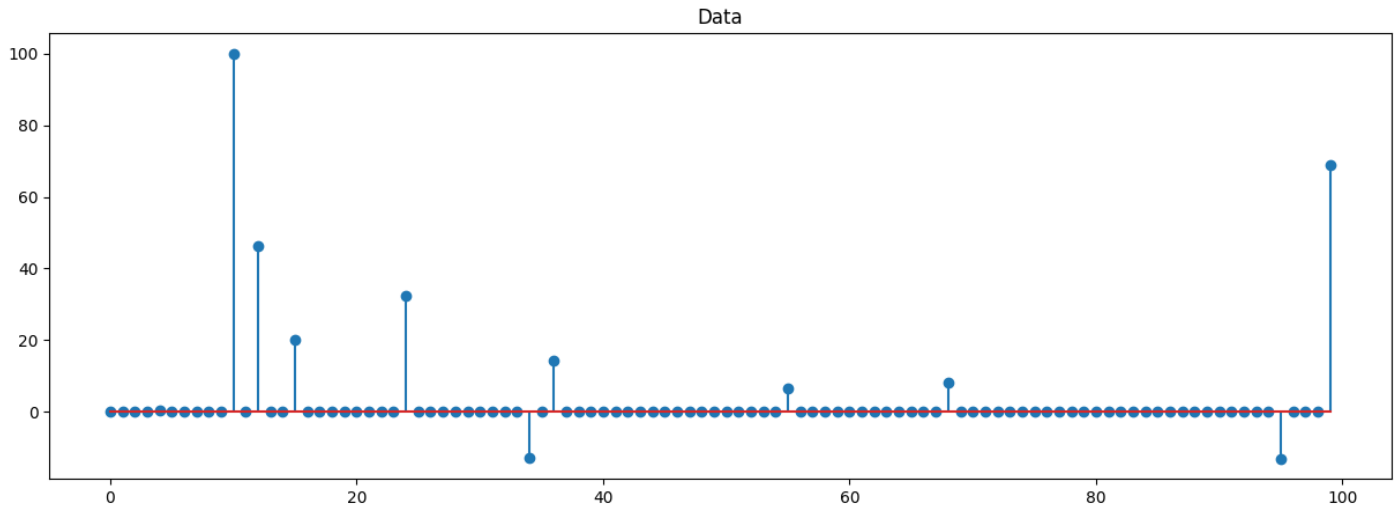

Figure 1: Generated data (signal)

3. (a) Maximum absolute scaling was performed on data by importing MaxAbsScalar class from scikit-learn's preprocessing module. Implented code snippet is shown below.

```
# MaxAbsScaler
import sklearn
from sklearn import preprocessing

max_scaled = preprocessing.MaxAbsScaler().fit_transform(signal)

```

(b) The min max scaler was implemented using the code given in the pdf, and min max scaling was performed on the generated signal. Related code is shown below.

```
# Min Max scaler
def min_max_scale(data):
    min_val = np.min(data)
    max_val = np.max(data)
    print("min of data", min_val, "max of data", max_val)
    scaled_data = (data - min_val) / (max_val - min_val)
    return scaled_data

scaled_data_min_max = min_max_scale(signal)

```

(c) The standard normalization function was implemented using the standardization scaling equation. The it was applied on the generated signal. Relevant defined function is shown below.

```
# Standard normalization
def standard_normalization(data):
    mean = np.mean(data)
    stan_deviation = np.std(data)
    x_normalized = (data - mean)/stan_deviation
    return x_normalized

standard_normalized_data = standard_normalization(signal)

```

4.

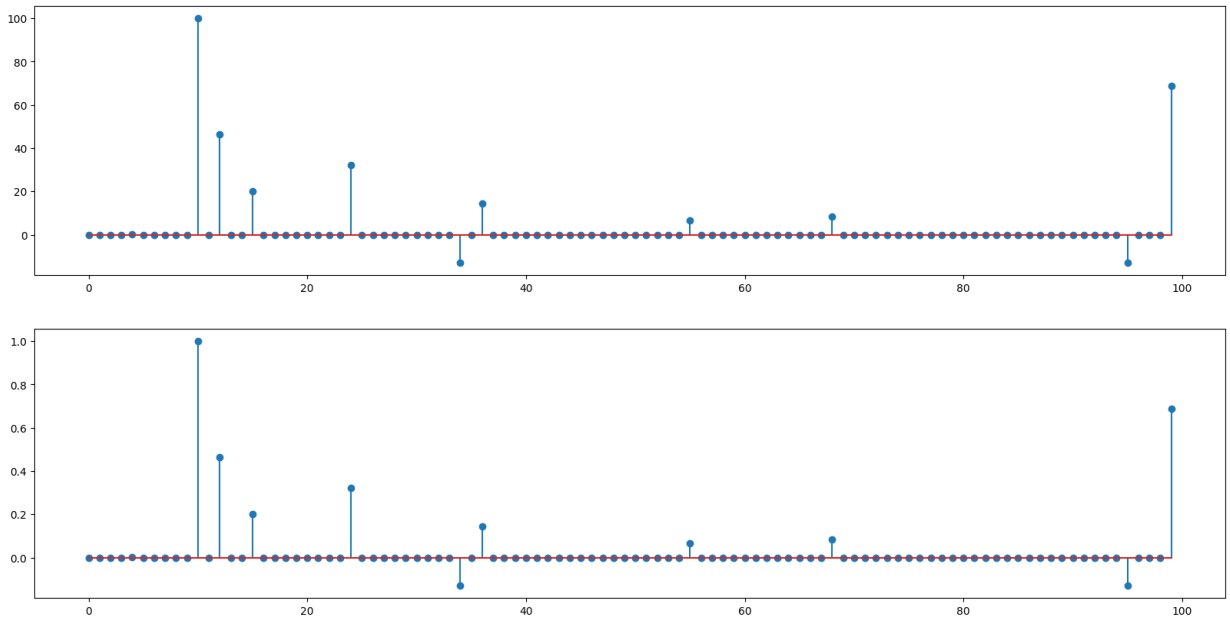

Figure 2: Before and after maximum absolute scaling the generated data

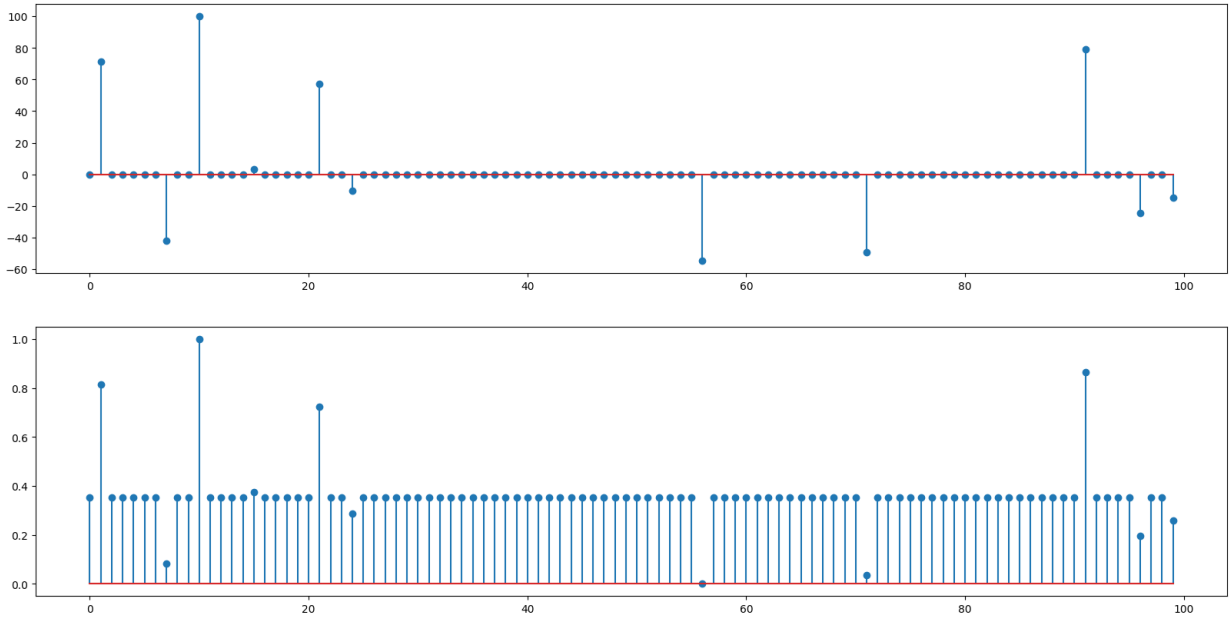

Figure 3: Before and after min-max scaling the generated data

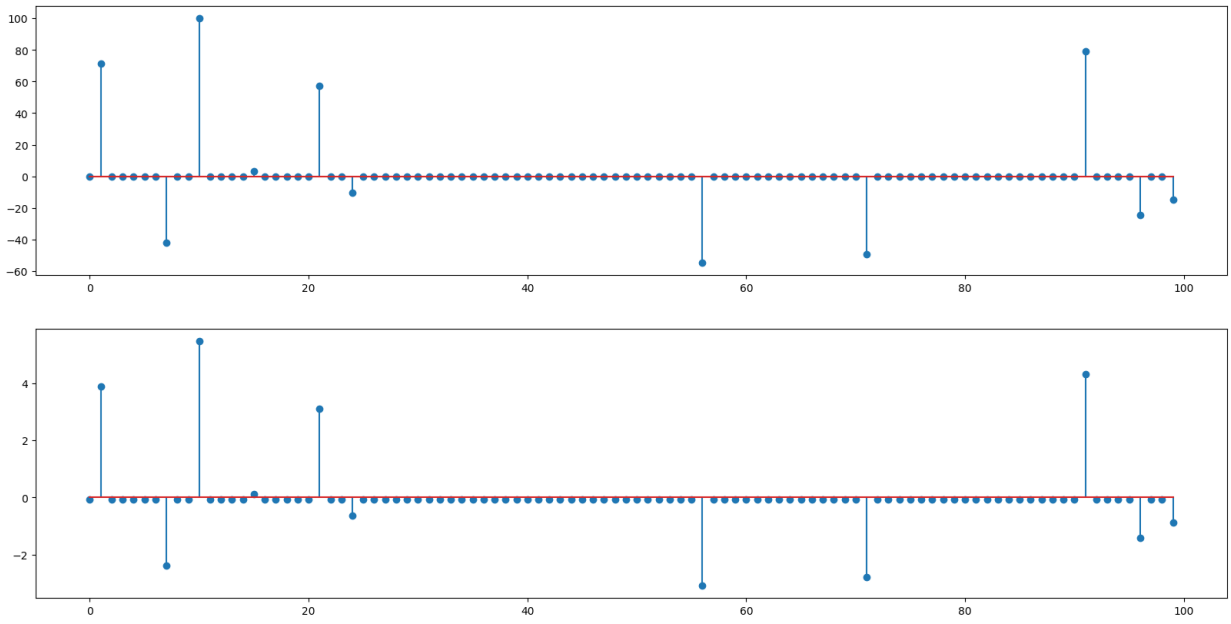

Figure 4: Before and after standardization scaling the generated data

- Data was visualized before and after each scaling method and plotted. Plots are shown above.

5. To count the non-zero elements before and after each normalization *np.count_nonzero* function from the Numpy library was used and results obtained are listed below.

   - Number of non-zero elements before normalization: 11
   - Number of non-zero elements after min-max-scaling: 99
   - Number of non-zero elements after maximum absolute scaling: 11
   - Number of non-zero elements after standardization scaling: 100

   According to the obtained results non-zero elements before and after maximum absolute normalization are equal. That is it preserves the sparsity of the data. In other two normalization methods non-zero elements before and after normalization are different.

6. **Min-max scaling**

   - Scaling formula:

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

   - As Figure 4 illustrates after the min-max scaling the features are in the range of [0,1]. That means this method transform features in the dataset to the range of [0,1]. Also, according to the formula and the Figure 4 minimum and maximum features(variable) have the values 0 and 1 respectively.
   - Impacts of data:
     - × After applying this scaling method model can precisely interpret data.
     - × It preserves the relationship between the data points.
     - × Min-max scaling is sensitive to outliers. So, if there are Therefore, handling outliers before performing the scaling may be essential.

Maximum absolute scaling

   - Scaling formula:

$$x_{\text{scaled}} = \frac{x}{abs(x_{\max})}$$

   - Maximum absolute scaler scales data by dividing each individual feature by maximum absolute value of each feature. This scales features into the range of [-1,1].
   - Impacts of data:
     - × It preserves the relative relationship between the data points.
     - × Maximum absolute scaler is not much sensitive to outliers as min-max scaler.
     - × Since the non-zero elements before and after the maximum absolute scaling is same, it preserves the sparsity of data.

Standardization scaling

   - Scaling formula:

$$x_{\text{scaled}} = \frac{x}{abs(x_{\max})}$$

- Maximum absolute scaler scales data by dividing each individual feature by maximum absolute value of each feature. This scales features into the range of [-1,1].
- Impacts of data:
  - × It preserves the relative relationship between the data points.
  - × Maximum absolute scaler is not much sensitive to outliers as min-max scaler.
  - × Since the non-zero elements before and after the maximum absolute scaling is same, it preserves the sparsity of data.

7. **MaxAbsScaler:**

- Data distribution: This method does not change the distribution's shape. It preserves the relative distances between data points.
- Structure: It preserves the data's structure.
- Scale: It scales features within the range of [-1, 1].

Min-Max Scaling:

- Distribution: This scaling method also does not change the distribution's shape. Also it preserves the relative distances between data points.
- Structure: It preserves the data's structure.
- Scale: It scales features within the range of [0, 1].

Standardization Scaling:

- Distribution: This method centers the data around 0 (that is the resulting distribution will have a 0 mean) and scales it to have a standard deviation of 1. So, it may be change the distribution's shape. It makes the data more Gaussian-like.
- Structure: It can change data structure, especially if the data is not normally distributed.
- Scale: This scales features into the range of [-1,1].

Among these scaling methods min-max scaling is a better choice for this dataset. Reasons for that is features in the dataset are of different scales. Also normalized values have the same meaning after normalization, only difference is range changed to [0,1].

2 Linear regression on real world data

1. The given dataset was loaded by using the *pd.read_csv* function using Pandas library. Then the separation of dependent and independent variables was done using the *iloc* function.

```
import pandas as pd

# printing data head
df = pd.read_csv("Advertising.csv")

# Seperating independent variables
X = df.iloc[:,1:4]

# Seperating dependent variables
y = df.iloc[:,4]

```

2. Data was splitted into training and testing sets with 80% and 20% using the *train_test_split* function from Scikit-Learn library. To ensure the same randomization used when run the code each time *random_state* parameter was set to 42.

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,
    random_state=42)

```

3. *LinearRegression* model from Scikit-Learn library was imported as the linear regression model and trained using the dataset. Then the coefficients of the model was obtained by *model.coef_* function and the intercept was obtained using *model.intercept_* function.

```
from sklearn.linear_model import LinearRegression

# Training a regression model
model = LinearRegression()
model.fit(X_train,y_train)
coef_tv = model.coef_[0]            # coefficient of TV
coef_radio = model.coef_[1]          # coefficient of radio
coef_newspaper = model.coef_[2]      # coefficient of newspaper
# Printing the value of intercept
print("Intercept:",model.intercept_)

```

Obtained results are,

- Intercept: 2.979067338122629
- Coefficient for TV: 0.044729517468716326
- Coefficient for Radio: 0.18919505423437652
- Coefficient for Newspaper: 0.0027611143413671935

4. Residual sum of squares (RSS), Residual Standard Error (RSE), Mean Squared Error (MSE), R^2 statistic, Std. Error for each feature, t-statistic for each feature and p-value for each feature were calculated as below. (Note that only a part of the code is shown below.) Trained model was evaluated on testing data using the function *model.predict()*.

```
y_pred_train = model.predict(X_train)
# Calculating RSS
RSS_train = np.sum((y_train - y_pred_train)**2)

# Calculating RSE
N_train = len(X_train)
d = X_train.shape[1] + 1
RSE_train = np.sqrt(RSS_train / (N_train - d))

# Calculating MSE
MSE_train = RSS_train/N_train

# Calculating R**2 statistic
TSS_train = np.sum((y_train - np.mean(y_train))**2)
R2_train = 1 - (RSS_train/TSS_train)

```

	Training data	Testing data
Residual sum of squares	432.8207	126.9639
Residual Standard Error	1.6657	1.8779
Mean Squared Error	2.7051	3.1741
R^2 statistic	0.8957	0.8994

Table 1: Calculated statistic values for testing and training data.

Feature	Standard Error	t-values	p-values
const	0.353517	8.426953	2.202287e-14
TV	0.001567	28.543587	8.166150e-64
radio	0.009693	19.517950	1.016134e-43
newspaper	0.007048	0.391761	6.957694e-01

Table 2: Standard Errors, t-Values, and p-Values for Each Feature

5. The R^2 statistic value for training and testing sets are 0.8957 and 0.8994 respectively. Since it is close to 1, we can say that there is a strong relationship between the advertising budgets and the sales.

6. When looking at the values of the coefficients of the each feature radio has the highest value, which is 0.189195. So, radio contributes more on sales out of the independent variables.

7. After predicting sales value for each scene given, the obtained sale values are as follows.

   - Spending \$25000 for each TV and radio: 8.82718
   - Spending \$50000 for TV: 5.21554
   - Spending \$50000 for radio: 12.43882

According to that the maximum sales value is obtained when spending $50000 for radio advertising individually.

3 Linear regression impact on outliers

1. To store the given table two numpy arrays were created one for dependent variable and one for independent variable.

2. A linear regression model was implemented using the Scikit-Learn library. The plotted scatter plot and the linear regression model is shown below.

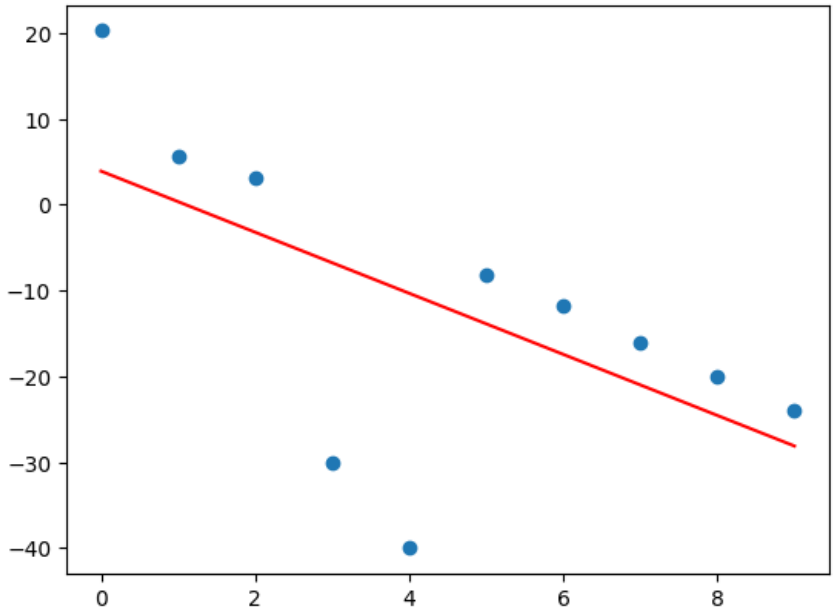

Figure 5: Scatter plot of x and y values and linear regression model.

3. The given two models were implemented as below.

```
# Defining model1
def model1(x):
    y = (-4*x) + 12
    return y

# Defining model2
def model2(x):
    y = (-3.55*x) + 3.91
    return y

```

4. The given loss function was then implemented as follow and the loss for each model were calculated using that loss function. The obtained loss values are 0.4354, 0.9728 for model1 and model2 respectively. So, the loss of model1 is small than model2.

```
# Defining loss function
def loss_function(model, x, y, beta=1, N=10):
    y_hat = model(x)
    diff_sq = (y - y_hat)**2
    loss = np.sum(diff_sq/(diff_sq + (beta**2)))
    loss = loss/N
    return loss

# Computing loss for each model
model1_loss = loss_function(model1, xi, yi)
model2_loss = loss_function(model2, xi, yi)

```

5. According to the losses computed for two models model1 has the lowest values for the loss among the two models. Lower the loss value better the model for the given dataset. So, among the two models given model1 was chosen as the most suitable model.

6. The impact of the outliers are reduced by the robust estimator by assigning lower weights to the outliers when calculating the loss function. Therefore, outliers have less influence on the estimated model parameters.

7. Plot of models and the data points

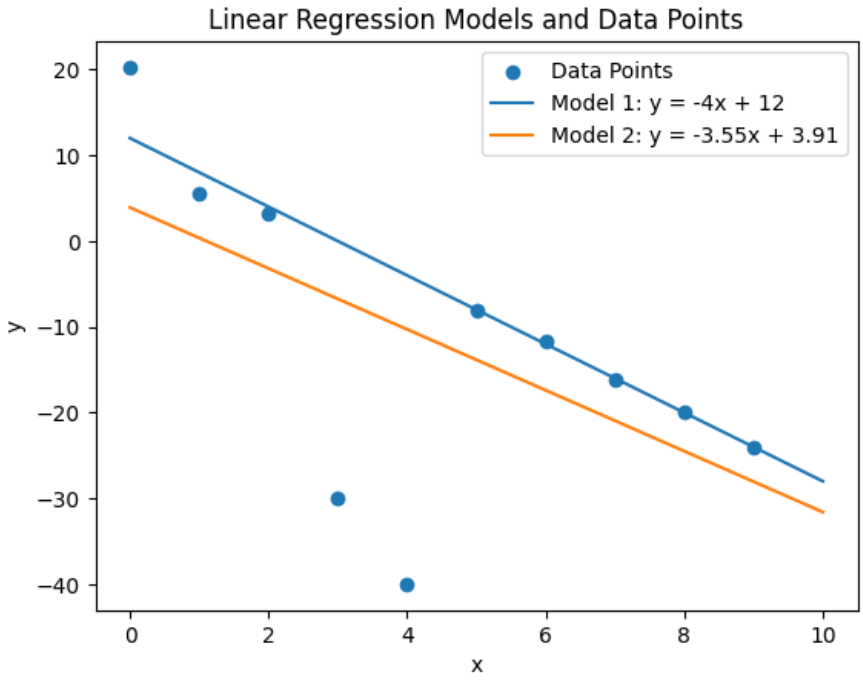

Figure 6: Caption

8. As β increases, the impact of outliers decreases, making the model more robust to extreme data points. The loss function of each model with beta values is shown below.

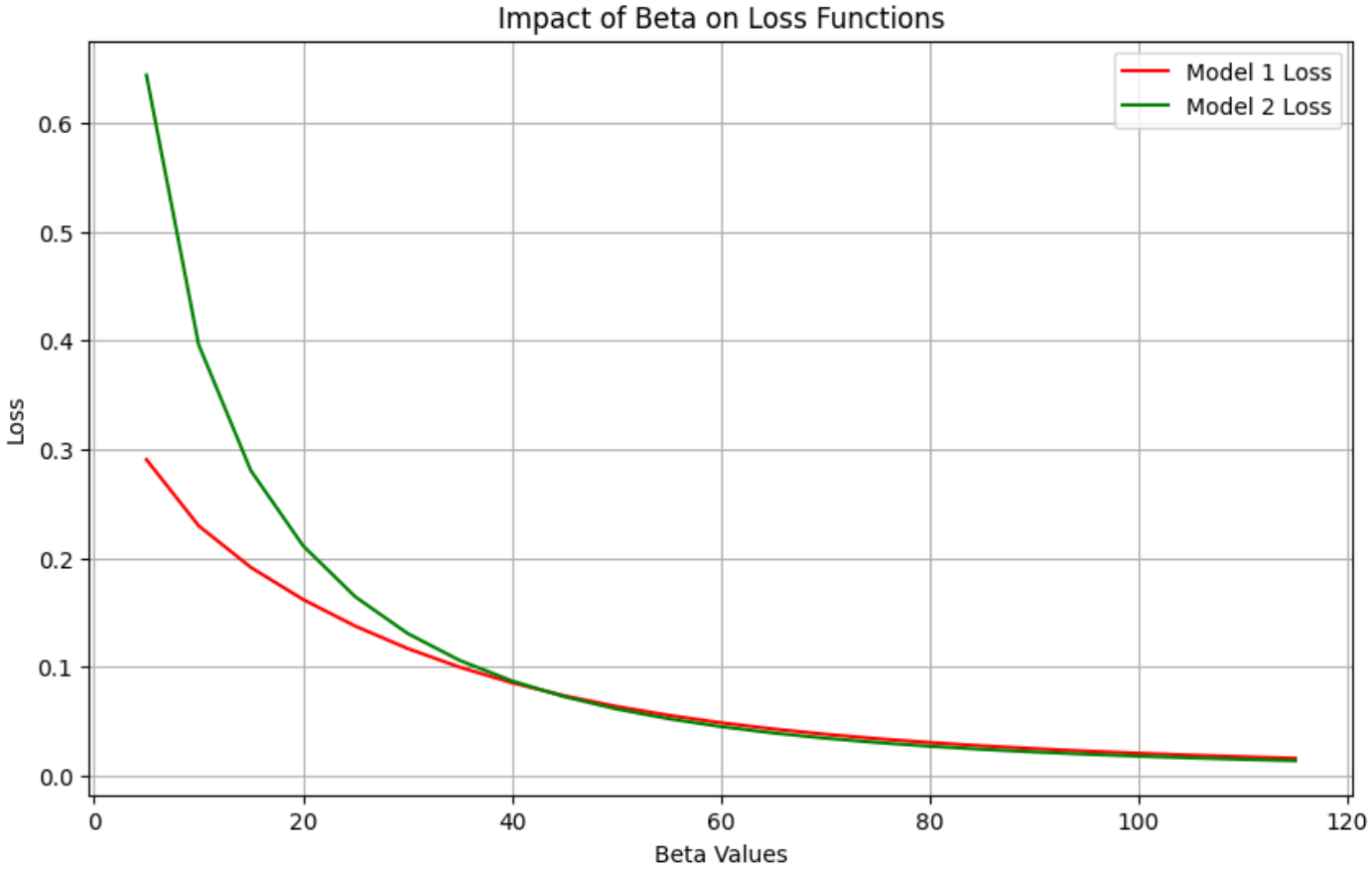

Figure 7: Impact of β on loss function