GLM Advance Data Mining Assignment 1

Part - A

QA1. What is the main purpose of regularization when training predictive models?

Answer:

Regularization is an effective methodology to control the model from overfitting. It tries to optimize the performance of the model on the training data so that the model doesn't underfit but it also adds a penalty to the model when the model turns to be complex to avoid overfitting. Overfitting takes place when the model gets too attuned to the training data and tries to comprehend the training data by closely capturing both the signals and noise. Here the model captures too many details and loses its ability to generalize well. There are several types of regularization techniques like

1. ***L1 Regularization*** *also known as Lasso Regularization adds the absolute value of the magnitude of the coefficient as a penalty term to the loss function.*

2. ***L2 Regularization*** *also known as Ridge Regularization adds the squared magnitude of the coefficient as the penalty term to the loss function.*

3. ***Dropout Regularization*** *drops some of the units in the network or the model during the training. (Mostly used in image classification, speech recognition and natural language processing tasks).*

Lasso or L1 Regularization is mostly used for Variable Selection, this method doesn't show any pity towards having unnecessary variables in the model which don't carry any importance towards the objective function. So, we get to see variable elimination in L1 Regularization. Ridge or L2 Regularization is the most commonly used methodology to control the model from overfitting, it doesn't remove or eliminate any variables but it does reduce the coefficients or weights of the attributes. Most of the regression models use L1 for variable selection and then use L2 for regularizing the model.

By reducing the complexity of the model and by preventing the model from overfitting, regularization can help to improve the generalization performance of the model on unseen data. This is very important because the ultimate goal of predictive modelling is to create models that can make accurate predictions on new, unseen data.

QA2. What is the role of a loss function in a predictive model? And name two common loss functions for regression models and two common loss functions for classification models.

Answer:

The loss function in a predictive model is used as a method to evaluate the performance of the model therefore built. It calculates the difference between the values that are predicted to that with the actual values. The end goal of the predictive model is to minimize this difference, which is commonly known as the "error" or "cost".

Loss Functions –

Regression Models:

Mean Squared Error (MSE) *–*

This loss function calculates the mean squared difference between the predicted and actual values of the target variable.

MSE = (1/n) Σ(yi - ŷi)^2

n is the number of samples,
yi is the actual value of the target variable for the ith sample,
ŷi is the predicted value of the target variable for the ith sample.

Mean Absolute Error (MAE) *–*

This loss function measures the average absolute difference between the predicted and actual values of the target variable.

MAE = (1/n) Σ|yi - ŷi|

n is the number of samples,

yi is the actual value of the target variable for the ith sample,

ŷi is the predicted value of the target variable for the ith sample.

Classification Models:

Binary Cross-Entropy Loss –

This loss function is used for binary classification problems where the target variable has two possible values. It measures the difference between the predicted probabilities and the actual binary labels.

BCE = - (1/n) Σ(yi * log(ŷi) + (1-yi) * log(1-ŷi))

n is the number of samples,

yi is the actual binary label (0 or 1) for the ith sample,

ŷi is the predicted probability of the positive class for the ith sample.

Categorical Cross-Entropy Loss –

This loss function is used for multi-class classification problems where the target variable has more than two possible values. It measures the difference between the predicted class probabilities and the actual class labels.

CCE = - (1/n) Σ(yij * log(ŷij))

n is the number of samples,

yij is the actual probability of the jth class for the ith sample (1 if the sample belongs to the jth class and 0 otherwise),

ŷij is the predicted probability of the jth class for the ith sample.

QA3. Consider the following scenario. You are building a classification model with many hyperparameters on a relatively small data set. You will see that the training error is extremely small. Can you fully trust this model? Discuss the reason.

Answer:

No, we fully cannot trust this model which has an extremely small training error rate. The reason is that the model may have overfit the training data which resulted in a small error rate in the training data whereas the same model will most likely not generalize well to new, unseen data. The end goal of building a predictive model is to generalize well, if the model doesn't generalize well then there is no point in building a predictive model.

While building a machine learning model it is common to observe the trend of a decrease in training error as we increase the complexity of the model. The complexity of the model depends on the size of the data and the algorithm which we are trying to use to build the model.

When a model has many hyperparameters and a relatively small data set, there is a high risk of overfitting. Overfitting occurs when a model has become too attuned to the training data and it starts to fit the nose and random fluctuations in the training data instead of following the underlying pattern that generalizes well to the unseen data. As a result, the model may have a very low error on the training set but when predicted on the unseen data the model may not perform well because of overfitting.

All in all, a model with an extremely small training error rate indicates overfitting and may not generalize well to new data. We could possibly use techniques such as "cross-validation" and "regularization" to avoid overfitting so that the model performs well on unseen data.

QA4. What is the role of the lambda parameter in regularized linear models such as Lasso or Ridge regression models?

Answer:

The lambda parameter is also known as the regularization parameter. It plays a crucial role in regularized linear models such as Lasso or Ridge Regression Models.

The goal of regularized linear models is to minimize a function i.e., loss function by adding a penalty term to the loss function of the model. The lambda parameter controls the weight of the penalty term.

A higher value of lambda will result in a stronger penalty and a simpler model, this may also lead the model to underfit since the model is too simple and is not able to capture the underlying patterns in the data. While a lower value of lambda will result in a weaker penalty and a more complex model, this will eventually lead to overfitting where the model focused more on the training data and not generalizing well on the test data.

For L1 regularization (also known as Lasso Regularization), increasing lambda tends to increase the sparsity of the model which will eventually turn the coefficients to zero. L1 Regularization is helpful for variables/feature selection and also to reduce the dimensionality of the problem.

For L2 regularization (also known as Ridge Regularization), increasing lambda tends to decrease the magnitude of the coefficients, which can help prevent overfitting. L2 doesn't eliminate any of the input attributes it generally tries to reduce the coefficients of the input attributes.

In both Lasso and Ridge regression, the lambda parameter is typically chosen through a process of cross-validation, where different values of lambda are tried and the one that results in the best performance on a validation set is selected. The end optimal value of lambda will be chosen based on the specific data set and the complexity of the model.

Part - B

Setting default values to get a clean output

```
knitr::opts_chunk$set(message = FALSE)
knitr::opts_chunk$set(warning = FALSE)
```

Loading Required Packages

```
library("class")
library("caret")
library("ISLR")
library("ggplot2")
library("esquisse")
library("tinytex")
library("tidyverse")
library("dplyr")
library("glmnet")
library("glmnetUtils")
library("corrplot")
library("moments")
```

Loading Carseats Data

```
data <- Carseats
```

Data Subsetting

Selecting a Subset of Data

```
new_data <- data %>%
select("Sales","Price","Advertising","Population","Age","Income","Education")
```

Data Cleaning

Checking for Null Values

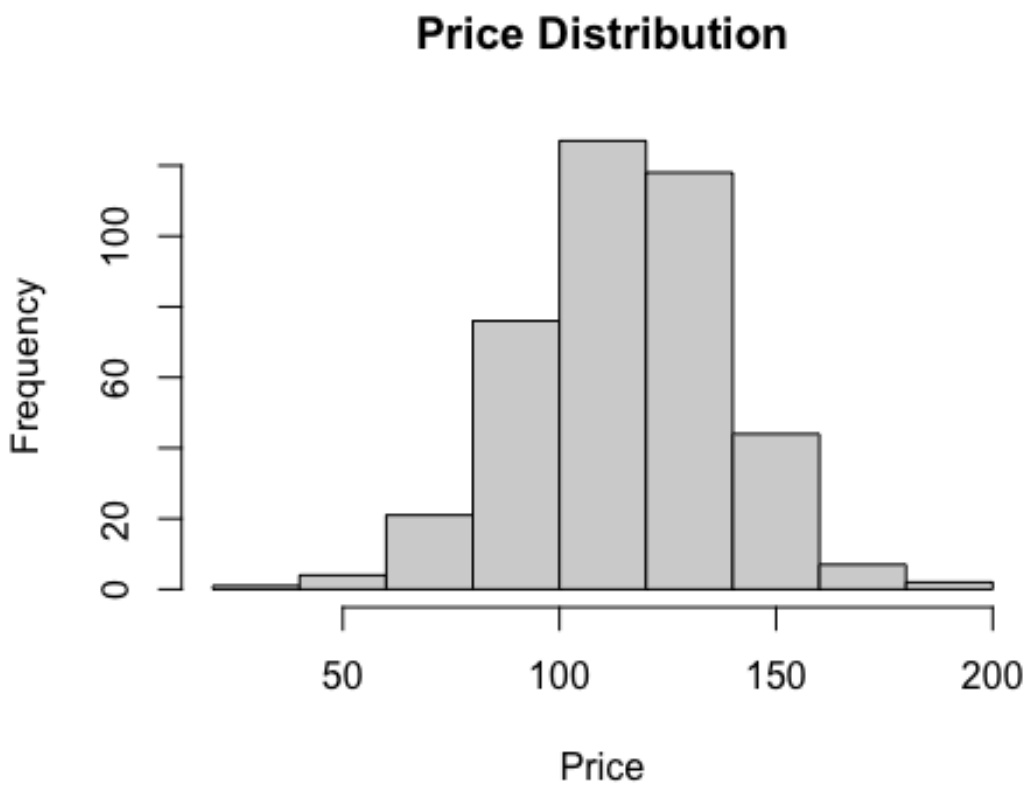
```
colMeans(is.na(new_data))
```

```
##       Sales       Price Advertising  Population         Age      Income
##           0           0           0           0           0           0
##   Education
##           0
```

Exploratory Data Analysis

Checking if the input attributes are normally distributed or not

```
#Price
hist(new_data$Price, xlab = "Price", main = "Price Distribution")
```

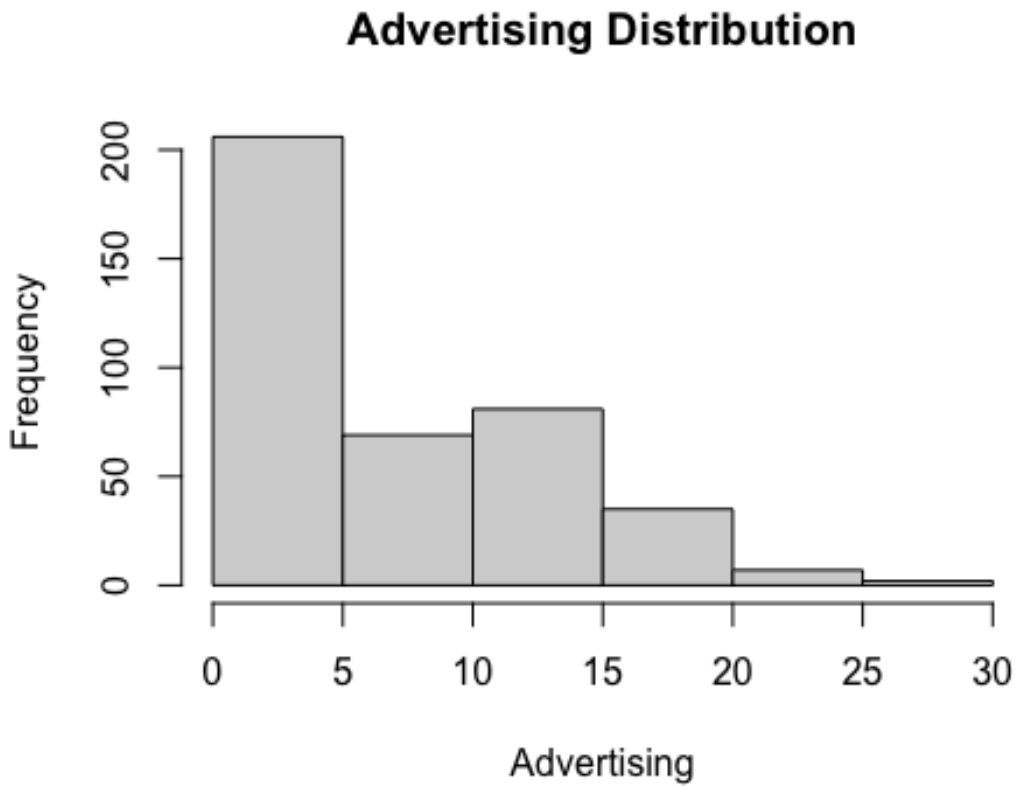



```
skewness(new_data$Price)
```

```
## [1] -0.1248159
```

As we can see the data seems to be normally distributed and the skewness value of -0.12 is close to 0, so there shouldn't be a need to transform the Price Variable.

```
#Advertising
hist(new_data$Advertising, xlab = "Advertising", main = "Advertising
Distribution")
```

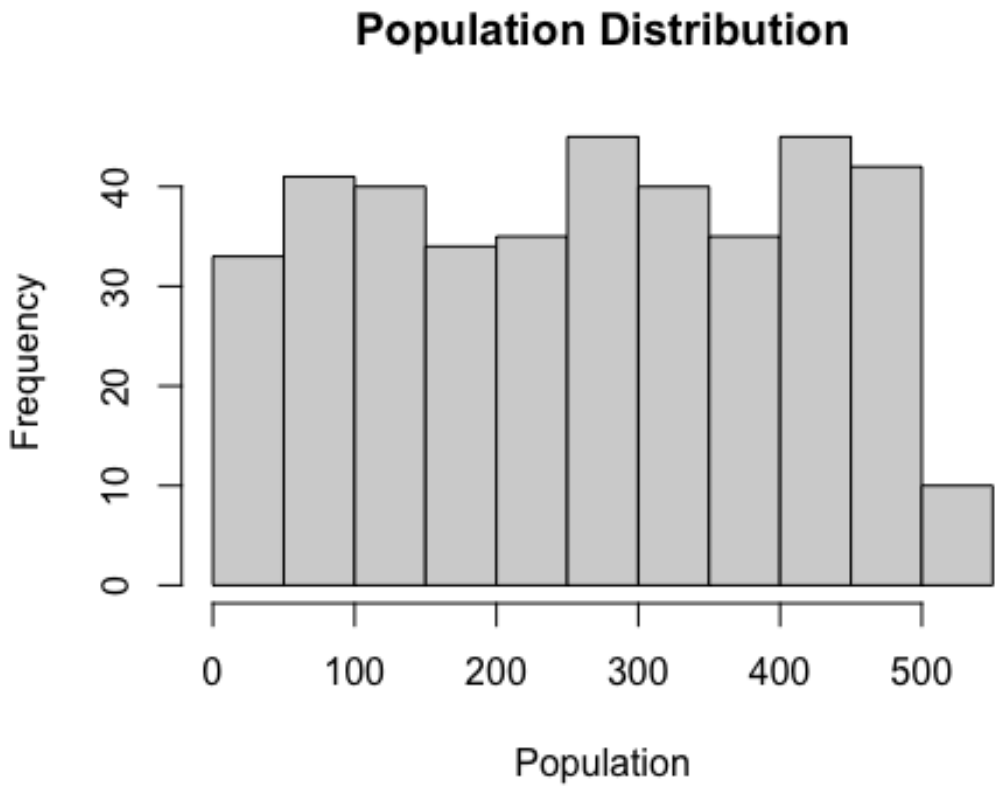



```
skewness(new_data$Advertising)
```

```
## [1] 0.6371848
```

Here the advertising attribute seems to be skewed as it has a long tail towards the right indicating that the data is "right skewed", so we need to possibly transform the data in order to get it normally distributed. Also, the skewness values of 0.63 is greater and nowhere close to 0.

```
#Population
hist(new_data$Population, xlab = "Population", main = "Population
Distribution")
```

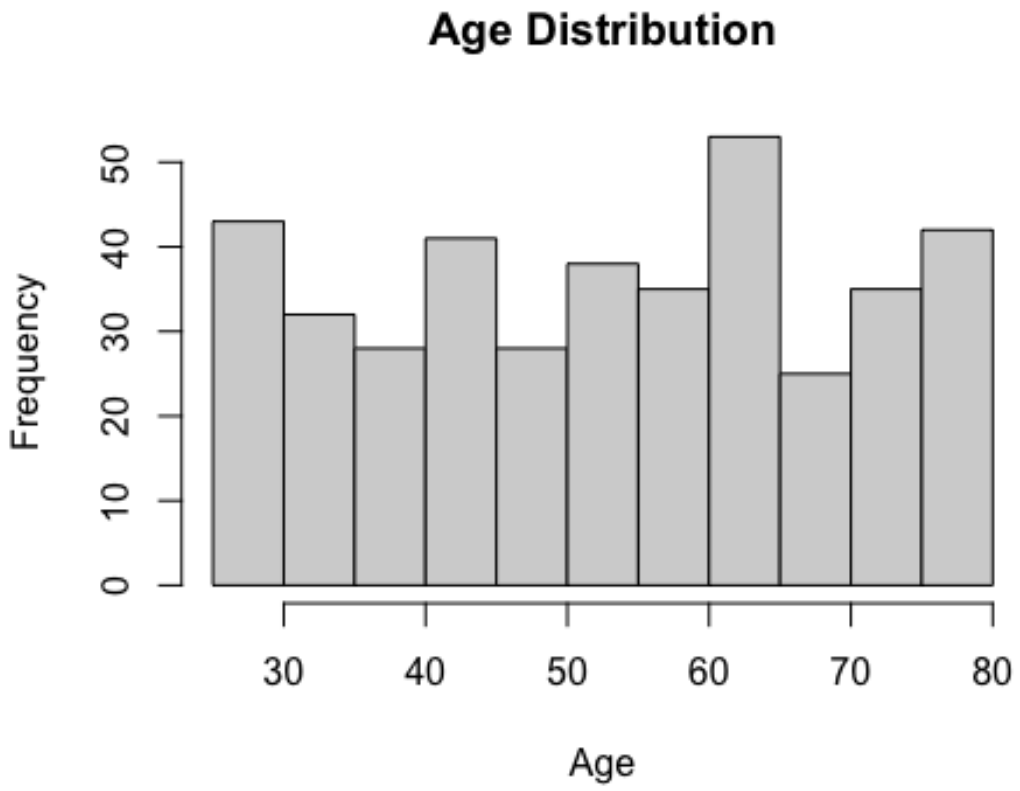



```
skewness(new_data$Population)

## [1] -0.05103434
```

Although the data doesn't have a bell shaped curve the skewness value of -0.05 is close to 0, so there shouldn't be a need to transform the Population Variable.

```
#Age
hist(new_data$Age, xlab = "Age", main = "Age Distribution")
```

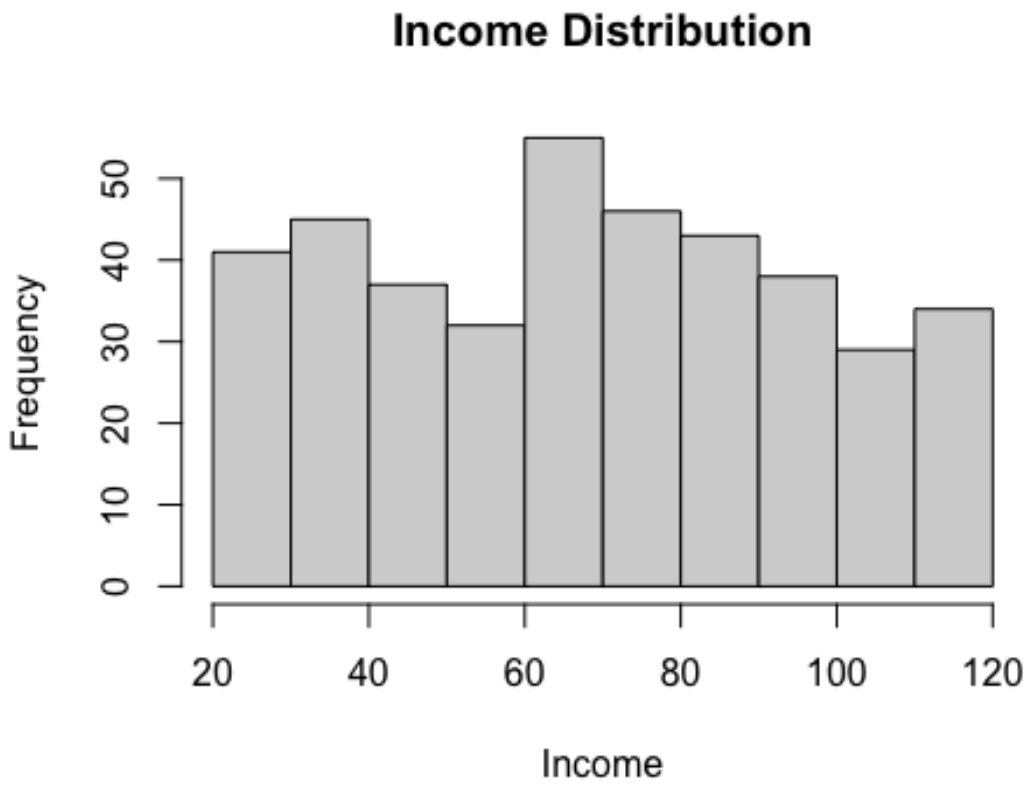



```
skewness(new_data$Age)
```

```
## [1] -0.076892
```

Although the data doesn't have a bell shaped curve the skewness value of -0.07 is close to 0, so there shouldn't be a need to transform the Age Variable.

```r
#Income
hist(new_data$Income, xlab = "Income", main = "Income Distribution")
```

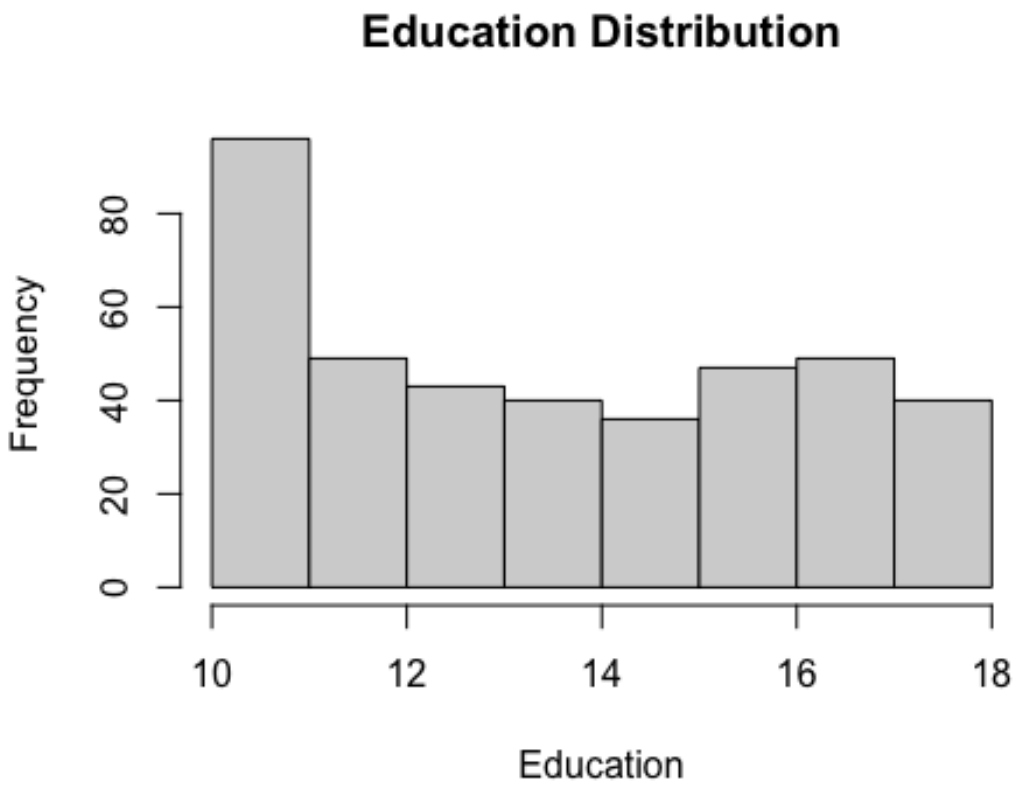



```r
skewness(new_data$Income)
```

```
## [1] 0.04925888
```

Although the data doesn't have a bell shaped curve the skewness value of 0.04 is close to 0, so there shouldn't be a need to transform the Income Variable.

```
#Education
hist(new_data$Education, xlab = "Education", main = "Education Distribution")
```




```
skewness(new_data$Education)
```

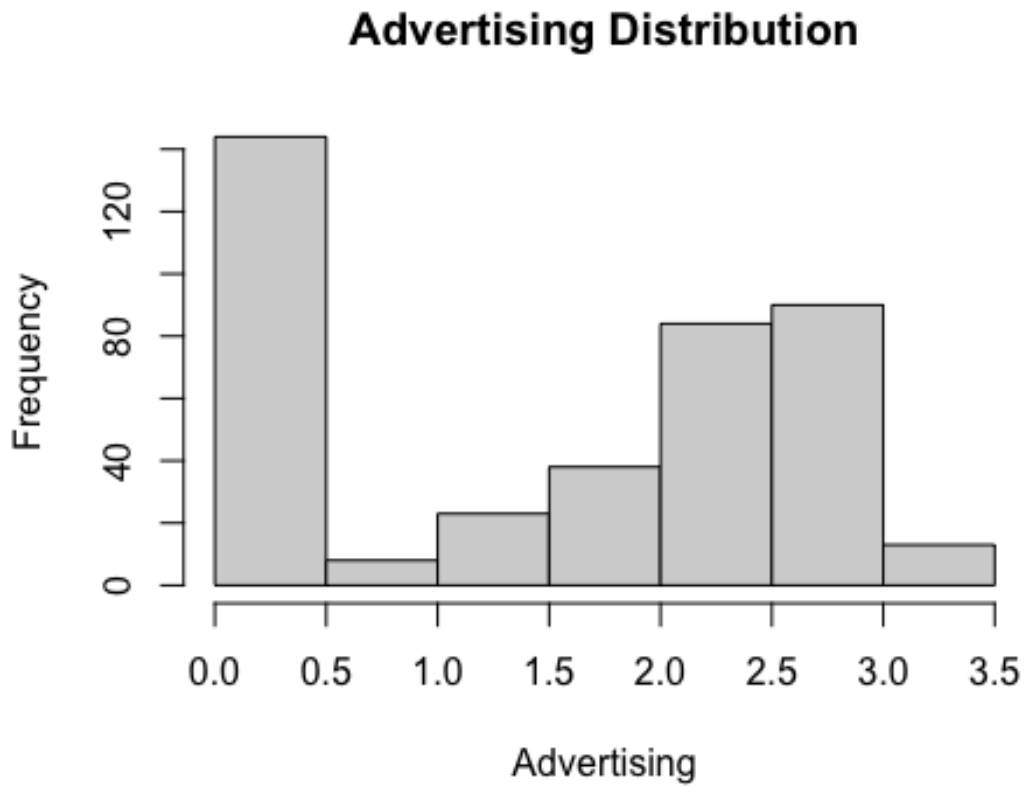
```
## [1] 0.04384163
```

Although the data doesn't have a bell shaped curve the skewness value of 0.04 is close to 0, so there shouldn't be a need to transform the Education Variable.

Data Transformation

Transforming only the Advertising Column

```
new_data$Advertising <- log(new_data$Advertising + 1)
hist(new_data$Advertising, xlab = "Advertising", main = "Advertising
Distribution")
```



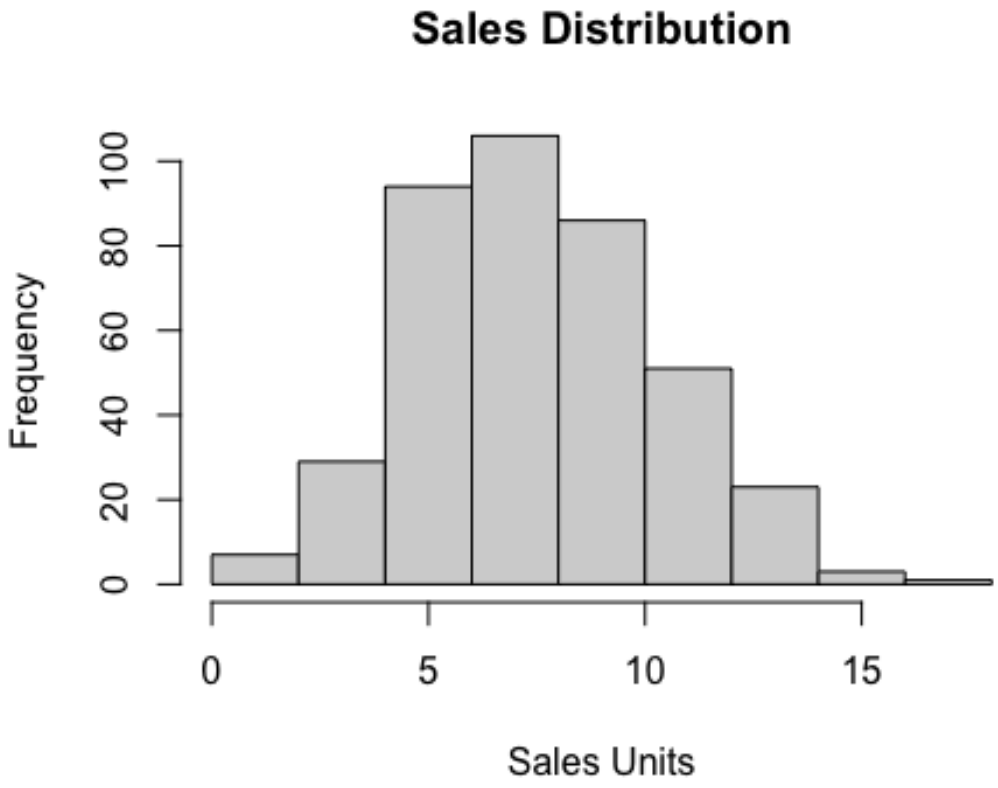

```
skewness(new_data$Advertising)
```

```
## [1] -0.1977802
```

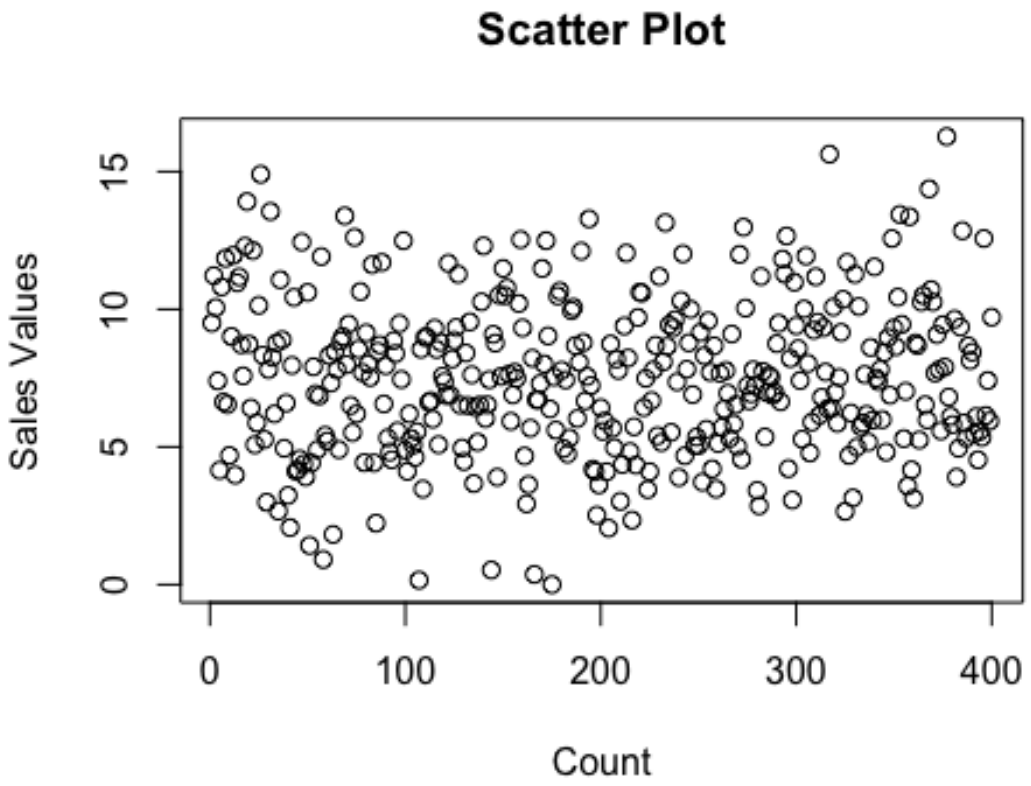
Since the Advertising variable had many zero values in them we couldn't perform log() transformation, we performed log(x+1) transformation where x is the data frame. Now after the transformation we could see that the skewness of the data is -0.19 which is closer to 0 when compared to the non-transformed value of 0.63.

Checking if the Target Variable is Skewed or Evenly Distributed

```
hist(new_data$Sales, xlab = "Sales Units", main="Sales Distribution")
```




```
plot(new_data$Sales, xlab = "Count", ylab="Sales Values",main="Scatter Plot")
```




```
skewness(new_data$Sales)
```

```
## [1] 0.1848638
```

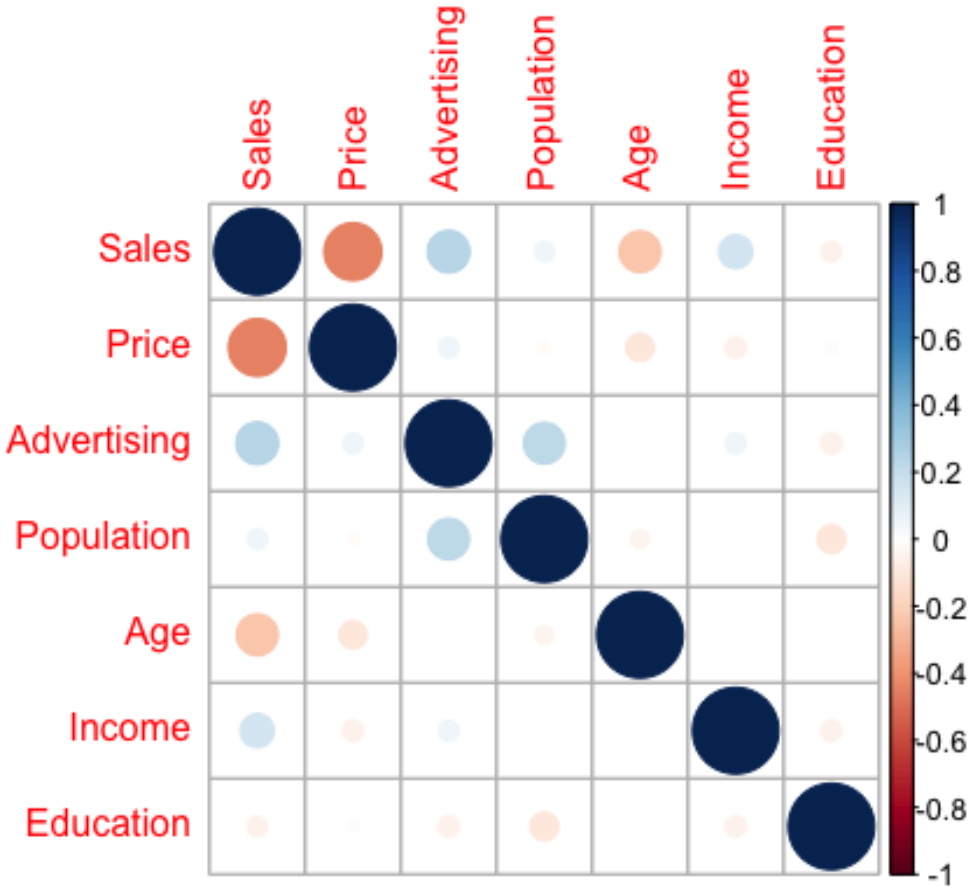
Usually we don't normalize the target or the dependent variable here in our case i.e. Sales. But there might be a situation where we might have to normalize the target variable as well, example: if the data is skewed and the has outliers.

In this case we can see that the data is properly distributed by the histogram plot and we could also see that there are no outliers through the scatter plot.

Finally the skewness value for the Sales attribute is 0.18 which is closer to zero. So, we can skip normalizing the target variable.

Correlation Plot

```
corrplot(cor(new_data))
```



Finally to end the "Exploratory Data Analysis" we are using correlation plot to check the distinct relation of the target variable i.e. Sales with that to the other input variables.

Sales has negative correlation with Price and Age (orange shaded circles).

Sales has positive correlation with Advertising and Income (blue shaded circles).

Preparing for Data Modelling

For the given set of questions there isn't a need to partition the data into train, validate and test. We could use Cross Validation in order to tune the hyperparameters.

Data Normalization

Scaling and Centering the input variables using Z score Normalization

```
set.seed(456)
Norm_Train <- preProcess(new_data[,-1], method = c("scale","center"))
#-1 refers to excluding the target variable from normalization

Normalized_Train <- predict(Norm_Train, new_data)
```

Since glmnet accepts the independent variables to be in a matrix format and the dependent variable to be in a vector format, we are transforming these variables.

Data Transformation

Input Variables - Matrix Format

Target Variable - Vector

```
train_x <- as.matrix(Normalized_Train[2:7])
#Independent Variables/Input Variables

train_y <- Normalized_Train[[1]]
#Dependent Variables/Output Variable
```

1. Build a Lasso regression model to predict Sales based on all other attributes ("Price", "Advertising", "Population", "Age", "Income" and "Education"). What is the best value of lambda for such a lasso model?

```
set.seed(789)
cvfit = cv.glmnet(train_x,train_y,data=Normalized_Train,nfolds=5,alpha=1)
cvfit

##
## Call:  glmnet::cv.glmnet(x = train_x, y = train_y, data =
Normalized_Train,      nfolds = 5, alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure     SE Nonzero
## min 0.01908    46   5.262 0.3073       6
## 1se 0.25810    18   5.524 0.2896       4
```

Best Lambda Value

cv.glmnet is a useful package that helps us know the lambda.min which represent the lambda value that is optimal and minimizes the cross validation mean square of the error.

```
cvfit$lambda.min
```

```
## [1] 0.01907519
```

As we can see 0.019 can be considered as the best lambda value for the lasso model which we have built.

Lambda 1SE

```
cvfit$lambda.1se
```

```
## [1] 0.2580965
```

There's also another lambda value which we can use if it is accepted to have a bigger lambda value which will result in a more regularized model.

This ensures that the cross validation error is still only up to one standard deviation bigger than the optimal value. For this model built the lambda value is 0.258.

Looking at the coefficients that was eliminated

```
coef(cvfit, s="lambda.min")
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  7.49632500
## Price       -1.33844979
## Advertising  0.72886034
## Population  -0.05828673
## Age         -0.77461557
## Income       0.28818371
## Education   -0.05429357
```

When the lambda is set to lambda.min i.e. 0.01907519 we can see that none of the attributes has been eliminated from the model.

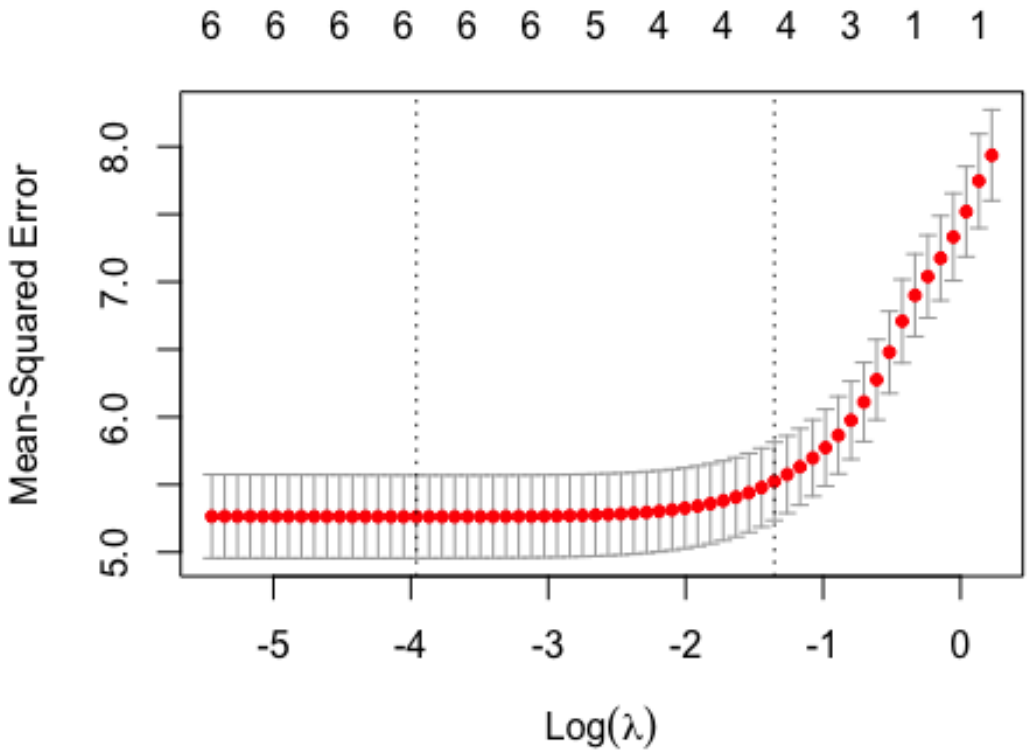
```
coef(cvfit, s="lambda.1se")
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  7.49632500
## Price       -1.07007879
## Advertising  0.47571907
## Population   .
## Age         -0.50618157
## Income       0.08235159
## Education    .
```

When the lambda is set to lambda.1se i.e. 0.2580965 we can see that two of the attributes have been eliminated from the model i.e. "Population" and "Education" have been eliminated.

Plotting the Model

```
plot(cvfit)
```



The Optimal (Min) and 1SE lambda values with respect to the above graph

```
lambda_min <- log(0.01907519)
lambda_min

## [1] -3.959367

lambda_1se <- log(0.2580965)
lambda_1se

## [1] -1.354422
```

2. What is the coefficient for the price (normalized) attribute in the best model (i.e. model with the optimal lambda)?

```
coef(cvfit, s="lambda.min")
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  7.49632500
## Price       -1.33844979
## Advertising  0.72886034
## Population  -0.05828673
## Age         -0.77461557
## Income       0.28818371
## Education   -0.05429357
```

The coefficient for the "Price Attribute" when the lambda is at minimum i.e. Optimal Lambda is -1.33844979.

3. How many attributes remain in the model if lambda is set to 0.01? How that number changes if lambda is increased to 0.1? Do you expect more variables to stay in the model (i.e., to have non-zero coefficients) as we increase lambda?

```
#Lambda = 0.01
coef(cvfit,s=0.01)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  7.49632500
## Price       -1.34888753
## Advertising  0.74064437
## Population  -0.07158094
## Age         -0.78526659
## Income       0.29537251
## Education   -0.06350835
```

When lambda (s) = 0.01 we have all the input attributes.

```
#Lambda = 0.1
coef(cvfit,s=0.1)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept)  7.4963250
## Price       -1.2468705
## Advertising  0.6362182
```

```
## Population   .
## Age         -0.6822203
## Income       0.2212576
## Education    .
```

Whereas when we increased the lambda value from 0.01 to 0.1 we see that the "Education" and "Population" attribute has been removed by the model.

In general when we increase the value of the lambda we expect the number of attributes to reduce, the lambda value controls the strength of the L1 penalty term, which shrinks the coefficients towards zero. When lambda is very small, the L1 penalty has a negligible effect and the regression model behaves like ordinary least squares (OLS) regression. In this case, the model may include all the available attributes, resulting in overfitting (s = 0.01)

As lambda increases, the L1 penalty term becomes more dominant, causing some coefficients to shrink towards zero and some to become exactly zero. The coefficients of the attributes that become exactly zero are eliminated from the model, resulting in a reduction in the number of attributes (s = 0.1).

Also, increasing the lambda will also result in reducing the coefficients of the attributes

i.e. Price with 0.01 lambda = -1.34888753

whereas Price with 0.1 lambda = -1.2468705

4. Build an elastic-net model with alpha set to 0.6. What is the best value of lambda for such a model?

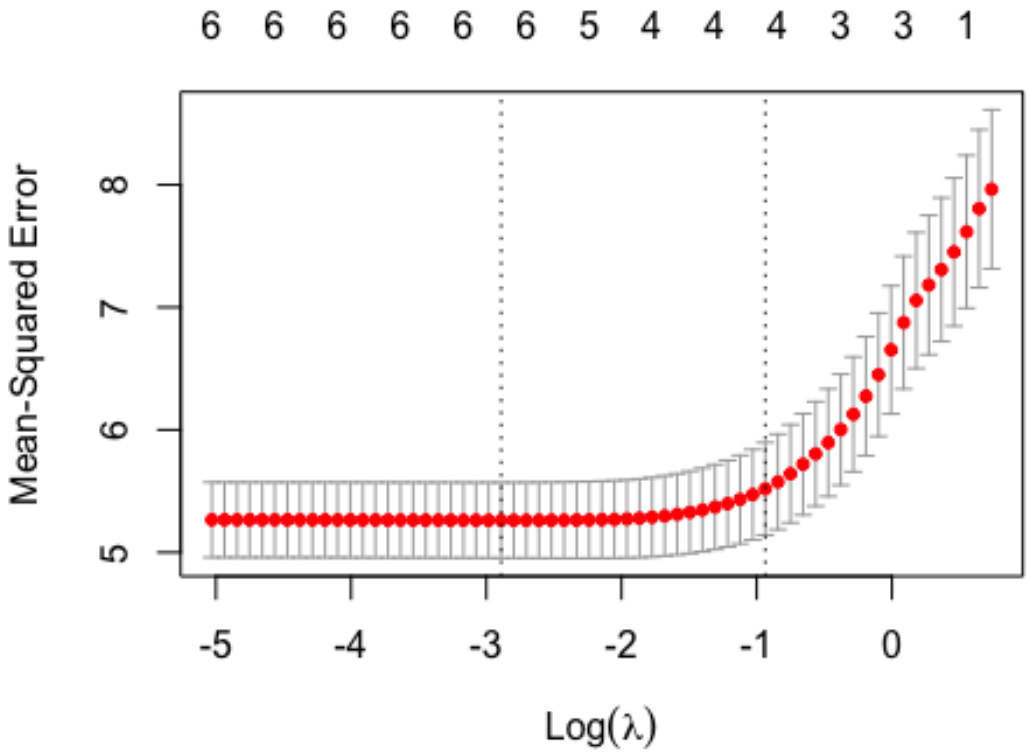
```
cvfit1 <-
cv.glmnet(train_x,train_y,data=Normalized_Train,nfolds=10,alpha=0.6)
cvfit1

##
## Call:  glmnet::cv.glmnet(x = train_x, y = train_y, data =
Normalized_Train,      nfolds = 10, alpha = 0.6)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure     SE Nonzero
```

```
## min 0.0556    40   5.262 0.3083       6
## 1se 0.3919    19   5.519 0.3781       4
```

cv.glmnet() plot

```
plot(cvfit1)
```



Looking at the attributes that were eliminated when the model was at lambda.min and alpha = 0.6

```
coef(cvfit1,s="lambda.min")
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  7.49632500
## Price       -1.31068027
## Advertising  0.70373819
## Population  -0.03517919
## Age         -0.75070051
```

```
## Income       0.27575862
## Education   -0.03989332
```

The cvfit1 model at lambda.min resulted in eliminating none of the input attributes.

***Optimal Lambda*:**

cvfit1 has a wrapper to it called as lambda.min which let's us know the best lambda value.

```
cvfit1$lambda.min
```

```
## [1] 0.0555574
```

The best value of lambda when the alpha = 0.6 is 0.0555574