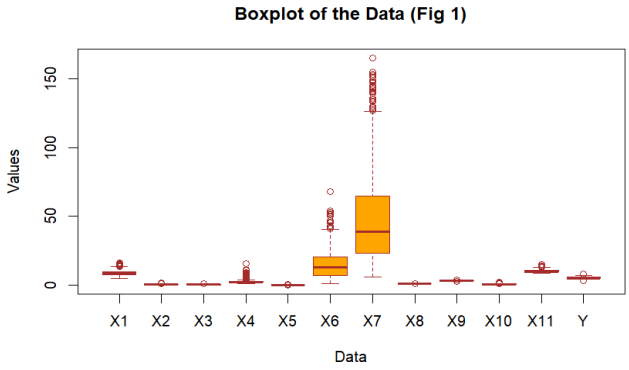
IE 7280 Spring 2023 Course Project

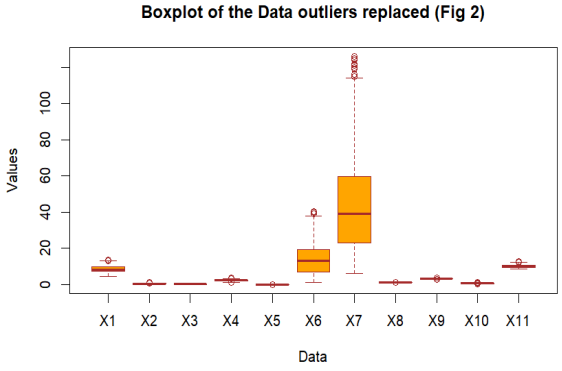
Anjali Patil, Archit Raj, Manwell Hanna, Shriya Kenkre

1.Abstract: The main objective of this project is to build an accurate multiple linear regression model to predict a response variable (Y) based on 11 regressors (X1 to X11). The dataset used for this project contains 999 observations, and we will be experimenting with various modeling techniques, ranging from basic linear regression on subsets of the regressors to more complex models that incorporate transformations of the regressors, such as polynomial regression. Additionally, we will explore regularization methods like LASSO regression to further improve model performance.

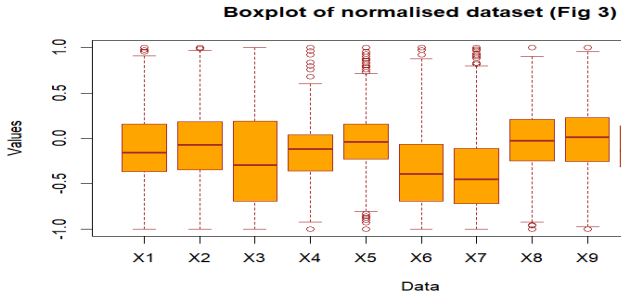
To begin the project, we will first conduct exploratory data analysis to understand the relationships between the response variable and the regressors. This will help us to determine which variables to include in our models and how to transform the data if necessary. After selecting the relevant features, we will train several linear regression models with different subsets of the regressors and compare their performances. We will then try more complex models, such as polynomial regression, and compare their results to the simpler models. Finally, we will implement regularization methods to improve model performance and select the best model based on its predictive accuracy. Overall, this project aims to build a robust model that accurately predicts the response variable based on the given regressors.

2.Data Tidying: Before modeling, we would like to perform data analysis to fix or remove irrelevant data. Data cleaning is an important step before feeding the dataset into the model so that we get more accurate results and increase the overall productivity. After examining the dataset, we found no duplicates or missing values. After plotting the boxplot (fig 1), we found that there are few outliers which might have a negative impact on our dataset. We replaced these outliers with median of the outliers (Fig 2)

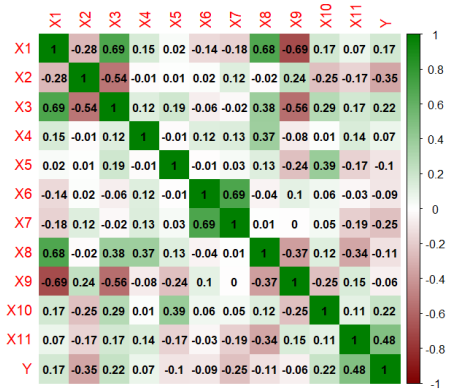




Then we normalized the dataset from -1 to 1. (Fig 3)



Then we wanted to look at the correlation between the variables. As you can see from the below plot, X1 and X3 have positive correlation and so do X1 and X8, and X1 and X9 are negatively correlated. Depending on how the model performs, we can either remove correlated variables or keep them.



3.Models:

1.MLE: Since this is a multiple regression problem, our first model is MLE. Multiple regression is a statistical technique that can be used to analyze the relationship between a single dependent variable(Y) and several independent variables (X1 to X11). Since we need to test our model, we divided the data set into training and testing dataset, with 70% of the dataset for training.

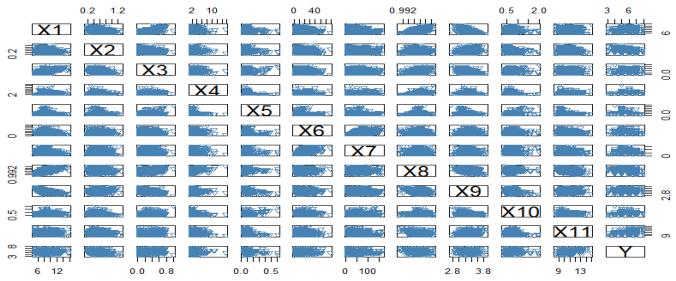
First, we built the model with all the variables.

```
model <-lm(Y~X1+X2+X3+X4+X5+X6+X7+X8+X10+X11, data = train_data)
```

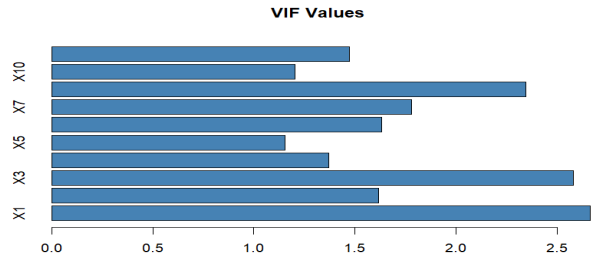
and we got RMSE =0.633 and Adjusted r_square=0.284.

We tried removing correlated variables, however there wasn't any significant improvement in the model. To investigate more, we decided to check the assumptions of MLE.

Assumption 1: - Linear Regression: The easiest way to determine if this assumption is met is to create a scatter plot of each predictor variable and the response variable(Y). It is difficult to see, but most of the variables have some linearity with the response variable.



Assumption 2:- No Multicollinearity: The easiest way to determine if this assumption is met is to calculate the VIF value for each predictor variable. If the VIF value is greater than 5 then this indicates potential multicollinearity. As you can see in the below figure, no variable has VIF value greater than 5, hence we can conclude that there's no multicollinearity.



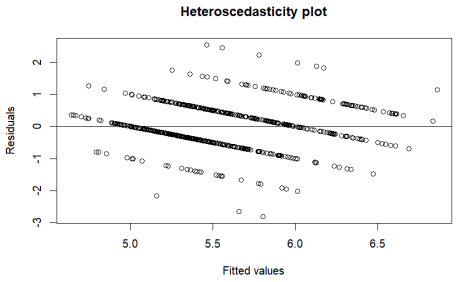
Assumption 3:- Independence : The simplest way to determine if this assumption is met is to perform a Durbin-Watson test. If the p-value is

less than 0.05, we can reject the null hypothesis and conclude that the residuals in this regression model are autocorrelated. In general, if the test statistic is between 1.5 and 2.5 then autocorrelation is likely not a cause for concern. As you can see from the below results, the test statistic is between 1.5 and 2.5, hence we conclude that autocorrelation is likely not a concern.

```
> durbinWatsonTest(model)
 lag Autocorrelation D-W Statistic p-value
   1     0.004540157      1.989991   0.856
 Alternative hypothesis: rho != 0
```

However, since p-value is greater than 0.05, we thought of adding lag of independent variables to our model. However, there wasn't any significant improvement in the model.

Assumption 4:- Homoscedasticity : The simplest way to determine if this assumption is met is to create a plot of standardized residuals versus predicted values. If the points in the scatter plot exhibit a pattern, then heteroscedasticity is present. As you can see from the below plot, standardized residuals are scattered about zero with a clear pattern, hence there is a problem of heteroscedasticity.



We further performed Breusch-pagan test. As you can see from the results, p-value less than 0.05 we reject the null hypothesis and conclude that heteroscedasticity is a problem in this model.

```
        studentized Breusch-Pagan test

data:  model
BP = 21.774, df = 10, p-value = 0.0163
```

The most common way to deal with heteroskedasticity is to transform the response variable by taking the log, square root, or cube root of all of the values of the response variable. This often causes heteroscedasticity to go away.

model3 <- lm(log(Y) ~ X1+X2+X3+X4+X5+X6+X7+X8+X9+X10+X11, data = train_data)

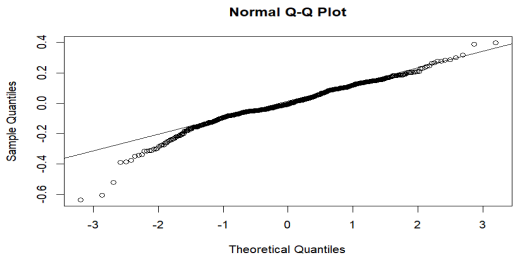
and we got RMSE =0.115 and Adjusted r_square=0.267.

```
        studentized Breusch-Pagan test

data:  model3
BP = 19.446, df = 11, p-value = 0.05354
```

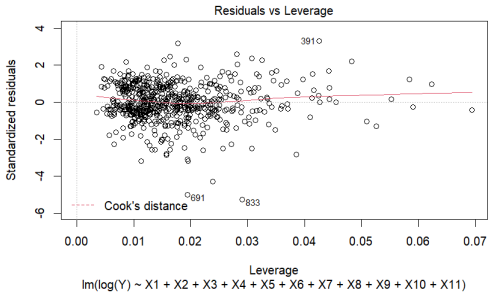
As we can see from the above results, p-value is close to 0.05 and hence this model works.

Assumption 5:- Multivariate Normality: We checked this assumption visually using the Q-Q plot for our latest model (model3-with log(Y)).



As you can see the points on the plot roughly form a straight diagonal line, hence the normality assumption is met.

Residuals vs. Leverage Plot: We then plotted residuals vs leverage plot and removed influential points and checked the performance.



However, there wasn't any significant improvement in the model and hence we decided to keep these points for our final model:

model3 <- lm(log(Y) ~ X1+X2+X3+X4+X5+X6+X7+X8+X9+X10+X11, data = train_data)

and we got RMSE =0.115 and Adjusted r_square=0.267.

git

We even tried the Forward,Backward and stepwise model. Below are the results:

```
RMSE for full model: 0.1159326
> cat("RMSE for forward model:", rmse_forward, "\n")
RMSE for forward model: 0.1159326
> cat("RMSE for backward model:", rmse_backward, "\n")
RMSE for backward model: 0.1149493
> cat("RMSE for stepwise model:", rmse_stepwise, "\n")
RMSE for stepwise model: 0.1149493
```

2.**Lasso Regression**:

The least absolute shrinkage method is applied to improve the accuracy by identifying the most important regressors which can be highly correlated with each other leading the coefficients to be highly sensitive to small variation in the data. To tackle this issue, LASSO includes a penalty measurement to the formula:

$$\sum_{i=1}^{n}(y_i - \sum_{j} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

A small value of lambda will result in a model with high variance and low bias, while a large value of lambda will result in a model with low variance and high bias.

First, we loaded the dataset and partitioned 80 – 20 for training and testing, placing a specific seed to avoid random data generation on each iteration of the model, while preparing "cross validation" as our hyperparameter tuning technique.

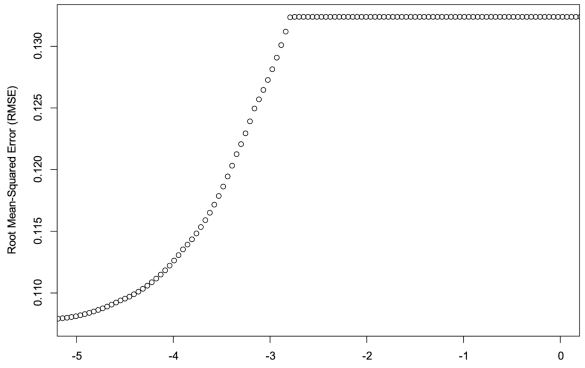
```
# Partition data and create index matrix of selected values
index <- createDataPartition(dataset$Y, p = 0.8, list=FALSE, times=1)

# Create test and train data frames
train_df <- dataset[index,]
test_df <- dataset[-index,]

# Specify 10-fold cross-validation as training method
ctrlspecs <- trainControl(method="cv",
                          number=200,
                          savePredictions="all")
```

We then loaded a range of possible lambda values into a vector using a built-in library and applied the cross-validation function to tune it. And select the most optimal lambda expression for our given dataset. It's important to note, that this method was applied previously on the unmodified dataset, giving us low estimations and measurements, therefore, the main edit was applying the training model and lambda optimization based on the " Logarithmic value of the dependent variable Y" and setting ALPHA - > 1 (*see figure*)

Plotting:



Based on the optimal value of lambda, we got an RMSE value of around 0.107

we then computed R_squared and got 0.502

```
```{r}
# Create function to identify Rsquared for best lambda,
# where x = Rsquared vector, y = lambda vector, &  z = optimal lambda value
Rsquared_lasso <- function(x, y, z){
  temp <- data.frame(x, y)
  colnames(temp) <- c("Rsquared", "lambda_val")
  rownum <- which(temp$lambda_val==z)
  print(temp[rownum,]$Rsquared) ^Rsquared_lasso
}

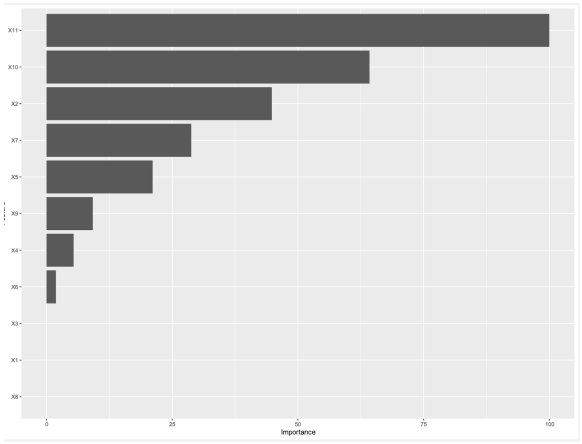
# Apply newly created Rsquared_lasso function
Rsquared_lasso(x=model1$results$Rsquared, # x = Rsquared vector
               y=model1$results$lambda,   # y = lambda vector
               z=model1$bestTune$lambda)  # z = optimal lambda value
```

[1] 0.5025853
```

note that this Value was computed by applying 200 folds for the k-fold cross validation.

Finally, we computed the most important variables to our dataset and omitted them in the calculations based on the lasso model.

(variable importance plot)



Finally, we applied this model to our testing dataset, with 10 folds for cross-validation,

```r
# Predict outcome using model from training data based on testing data
predictions1 <- predict(model1, newdata=test_df)
# Model performance/accuracy
mod1perf <- data.frame(RMSE=RMSE(predictions1, log(test_df$Y)),
                       Rsquared=R2(predictions1, log(test_df$Y)))

# Print model performance/accuracy results
print(mod1perf)
```

Table | Console

"RMSE"	"Rsquared"
0.120357027815281	0.338986875052593

4.Best Model

After a comprehensive evaluation of multiple linear regression models and Lasso regression, we concluded that the Lasso regression model is the optimal choice for the given dataset. Lasso regression outperforms the multiple linear regression models in terms of predictive accuracy, providing a more reliable and effective model for our data. The following points highlight the technical reasons behind selecting Lasso regression as the best model:

- **Predictive Accuracy:** The Lasso regression model has an RMSE of 0.107 and an adjusted R-squared of 0.502, whereas the best multiple linear regression model has an RMSE of 0.115 and an adjusted R-squared of 0.267. The improved accuracy of Lasso regression allows for more reliable predictions of the response variable (Y).

```r
# Create vector of potential lambda values
lambda_vector <- 10^seq(5, -5, length=500)

# Specify lasso regression model to be estimated using training data
# and k-fold cross-validation process
model1 <- train(log(Y) ~ .,
                data=train_df,
                preProcess=c("center",'scale','nzv','YeoJohnson'),
                method="glmnet",
                tuneGrid=expand.grid(alpha=1, lambda=lambda_vector),
                trControl=ctrlspecs,
                na.action=na.omit)


# Lasso regression model coefficients
coef(model1$finalModel, model1$bestTune$lambda)
```

- **Variable Selection:** Lasso regression incorporates a regularization term that penalizes the model for the inclusion of unnecessary variables, effectively shrinking the coefficients of less important variables towards zero. This feature allows the model to identify the most relevant variables in the dataset, simplifying the model and reducing the risk of overfitting.
- **Robustness:** Lasso regression is robust to multicollinearity issues that can arise when dealing with multiple independent variables. By shrinking coefficients of correlated variables, Lasso regression mitigates the impact of multicollinearity on the model's performance.
- **Model Complexity:** Lasso regression reduces the ability complexity of the model by promoting sparsity in the coefficient estimates, which results in a more interpretable and efficient model. This reduction in complexity is beneficial when working with datasets containing numerous independent variables, as it simplifies the analysis and interpretation of the model's results.
- **Cross-validation:** We implemented k-fold cross-validation with 200 folds to optimize the hyperparameter lambda, ensuring that the Lasso regression model generalizes well to unseen data. Cross-validation results indicate that the Lasso regression model consistently

provides better performance across different folds, further supporting its selection as the best model for our dataset.

Conclusively, the Lasso regression model stands out as the best model for this project due to its enhanced predictive accuracy, variable selection capabilities, robustness to multicollinearity, reduced model complexity, and strong cross-validation performance. Implementing the Lasso regression model for the given project enables accurate prediction of the response variable (Y) based on the selected regressors, leading to a more robust and efficient solution.