FA582 Assignment 4 Report

Naveen Mathews Renji - 20016323

Problem 1 - OJ data set

Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

1. Load the OJ dataset from the CSV file.
2. I converted the Purchase and Store7 variables in the oj_data dataset to factors, as these were initially character types.
3. Set a seed for reproducibility
4. Randomly select 800 observations to be included in the training set.
5. Split the dataset into a training set and a test set based on these indices

Fit a tree to the training data, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

I fitted a decision tree to the training data using the tree package in R. The response variable is 'Purchase', representing whether the customer bought Citrus Hill (CH) or Minute Maid (MM) Orange Juice, and the predictors are the other variables in the dataset

```
> summary(oj_tree)

Classification tree:
tree(formula = Purchase ~ ., data = train_set)
Variables actually used in tree construction:
[1] "LoyalCH"   "PriceDiff"
Number of terminal nodes:  8
Residual mean deviance:  0.7625 = 603.9 / 792
Misclassification error rate: 0.165 = 132 / 800
```

The summary of the tree revealed that **'LoyalCH'** and **'PriceDiff'** were the primary variables used, indicating their significant influence on orange juice brand choice. The model comprised **8 terminal nodes**, suggesting a nuanced understanding of customer behavior while maintaining model simplicity.

Key performance metrics included a **residual mean deviance of 0.7625**, reflecting the model's goodness of fit, and a misclassification **error rate of 16.5%** on the training set. This error rate, while reasonable, suggests potential areas for model improvement.

These findings underscore the importance of brand loyalty and price in influencing purchase decisions.

Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed

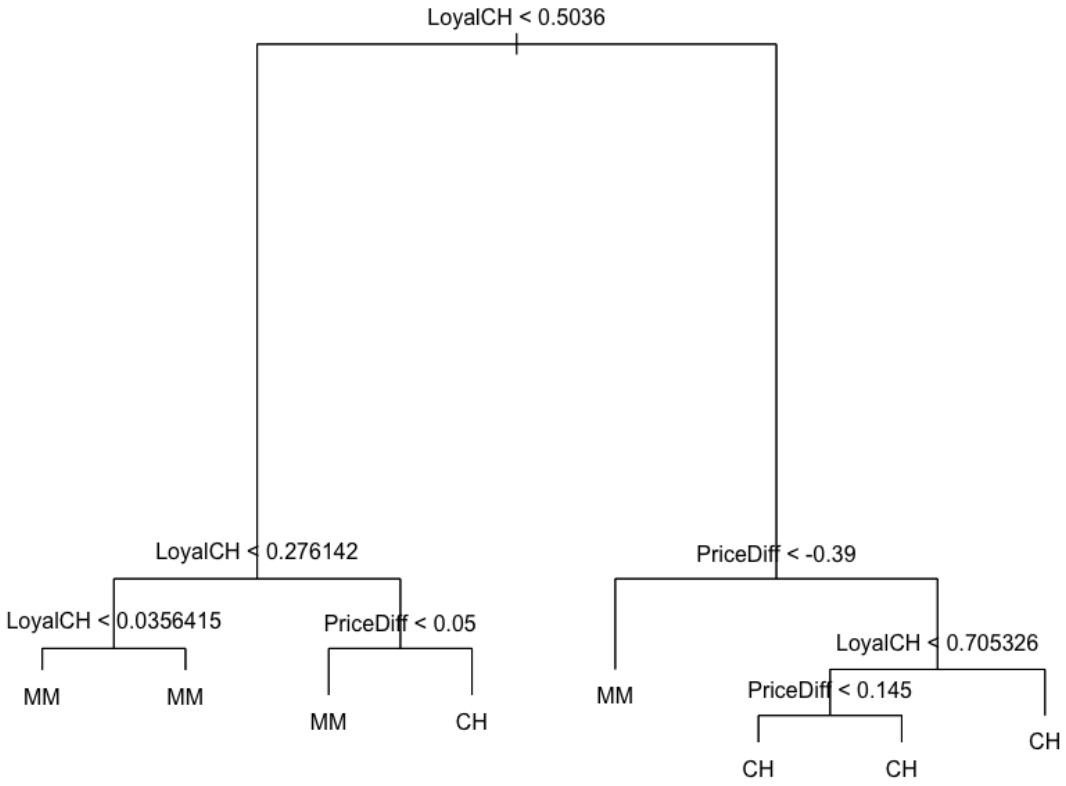
```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 800 1071.00 CH ( 0.60875 0.39125 )
   2) LoyalCH < 0.5036 350  415.10 MM ( 0.28000 0.72000 )
     4) LoyalCH < 0.276142 170  131.00 MM ( 0.12941 0.87059 )
       8) LoyalCH < 0.0356415 56   10.03 MM ( 0.01786 0.98214 ) *
       9) LoyalCH > 0.0356415 114  108.90 MM ( 0.18421 0.81579 ) *
     5) LoyalCH > 0.276142 180  245.20 MM ( 0.42222 0.57778 )
      10) PriceDiff < 0.05 74   74.61 MM ( 0.20270 0.79730 ) *
      11) PriceDiff > 0.05 106  144.50 CH ( 0.57547 0.42453 ) *
   3) LoyalCH > 0.5036 450  357.10 CH ( 0.86444 0.13556 )
     6) PriceDiff < -0.39 27   32.82 MM ( 0.29630 0.70370 ) *
     7) PriceDiff > -0.39 423  273.70 CH ( 0.90071 0.09929 )
      14) LoyalCH < 0.705326 130  135.50 CH ( 0.78462 0.21538 )
        28) PriceDiff < 0.145 43   58.47 CH ( 0.58140 0.41860 ) *
        29) PriceDiff > 0.145 87   62.07 CH ( 0.88506 0.11494 ) *
      15) LoyalCH > 0.705326 293  112.50 CH ( 0.95222 0.04778 ) *
```

Looking at the detailed output of the oj_tree, I focused on interpreting node 8, a terminal node defined by LoyalCH < 0.0356415. This node:

1. Targets customers with very low loyalty to Citrus Hill (CH), specifically those with a loyalty score below approximately 0.036.
2. Contains 56 observations with a low deviance of 10.03, suggesting a good model fit for these data points.
3. Predicts a strong preference for Minute Maid (MM) with about 98.21% of observations in this node choosing MM over CH.

This implies that customers with minimal loyalty to CH are overwhelmingly likely to purchase MM. This insight is particularly relevant for understanding consumer behavior and tailoring marketing strategies. It highlights a market segment where CH could potentially increase its influence through targeted marketing efforts, aiming to enhance brand loyalty among these consumers.

Create a plot of the tree, and interpret the results



- The plotted decision tree illustrates the hierarchical structure of the decision-making process for predicting customer purchases of orange juice brands (Citrus Hill or Minute Maid). The tree is split based on the values of two primary predictors: 'LoyalCH' and 'PriceDiff'.
- The first split is made on 'LoyalCH', dividing the dataset into two major groups: customers with loyalty to Citrus Hill less than 0.5036 and those with greater loyalty.
- For customers with 'LoyalCH' less than 0.5036, further splits are based on 'LoyalCH' and 'PriceDiff'. Customers with very low loyalty (less than 0.0356415) are predicted to choose Minute Maid (MM), reflecting a strong preference for MM in this segment, likely due to low attachment to the CH brand.
- For those with higher loyalty to CH ('LoyalCH' greater than 0.5036), the tree uses 'PriceDiff' for further decisions. A significant negative price difference (less than -0.39) leads to a prediction of MM purchase, possibly indicating price sensitivity despite higher brand loyalty.

- Terminal nodes represent the final predictions based on the combination of conditions leading to them. The tree has a mix of nodes predicting both CH and MM, highlighting different customer segments and their predicted behavior.

Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

Using the decision tree model fitted to the training set, I predicted the Purchase variable on the test set. The predict() function in R was utilized for this purpose, specifying the new data as the test set and setting the type parameter to "class" to obtain actual class predictions. Following the predictions, a confusion matrix was generated using the table() function, which cross-tabulated the actual test labels against the predicted test labels.

```
> print(conf_matrix)
    test_pred
      CH  MM
  CH 150  16
  MM  34  70
```

- There were 166 actual purchases of CH. Of these, 150 were correctly predicted as CH (true positives), and 16 were incorrectly predicted as MM (false negatives).
- There were 104 actual purchases of MM. Of these, 70 were correctly predicted as MM (true negatives), and 34 were incorrectly predicted as CH (false positives).

```
> print(test_error_rate)
[1] 0.1851852
```

The test error rate, which represents the proportion of incorrect predictions out of all predictions made on the test set, is approximately 18.52% (printed as 0.1851852 in the output). This means that the decision tree model incorrectly predicted the Purchase outcome for about 18.52% of the test set observations.

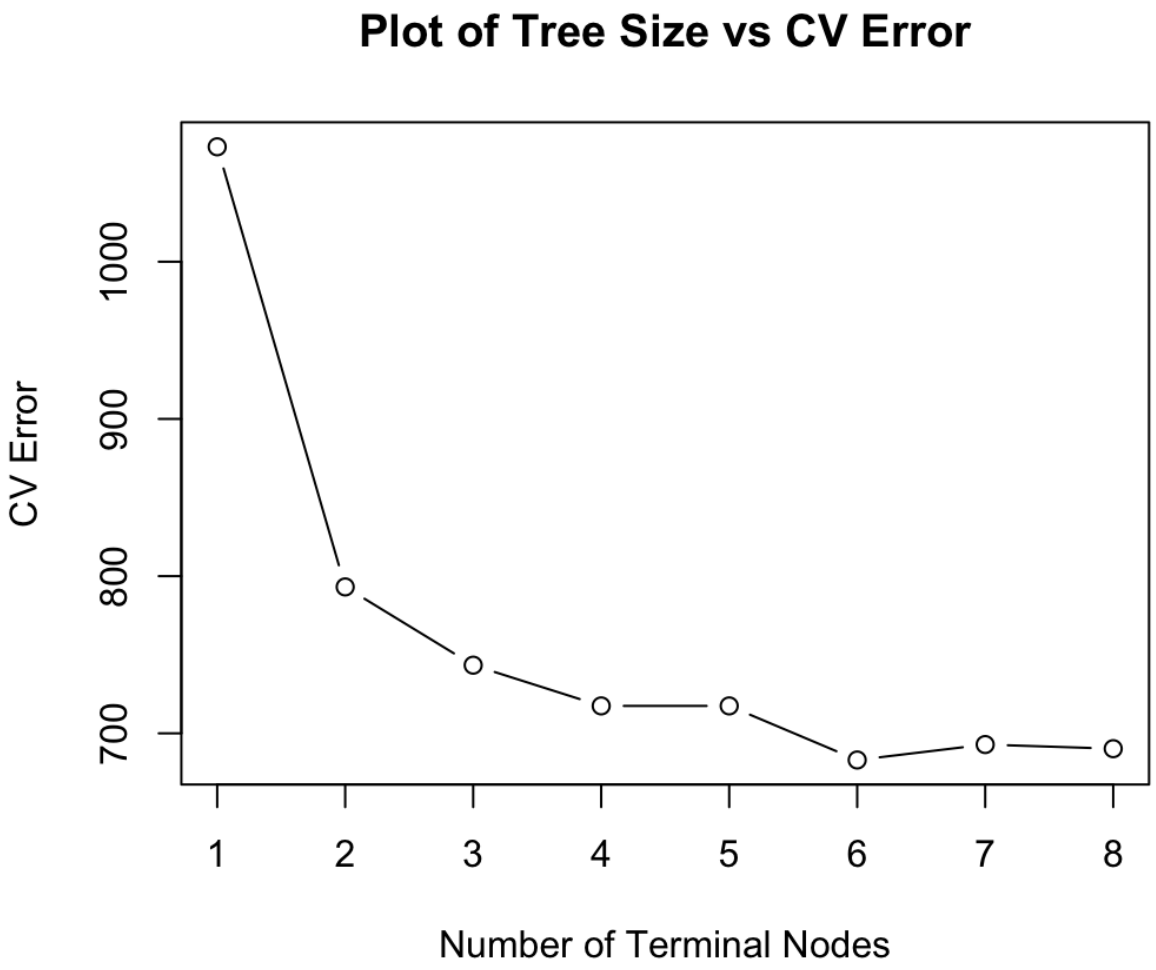
Apply the cv.tree() function to the training set in order to determine the optimal tree size

To find the optimal size for the decision tree, I employed the cv.tree() function, which performs cross-validation on the tree model fitted to the training data. This function iterates over different sizes of the tree to find the one that minimizes the cross-validation error

```
$size
[1] 8 7 6 5 4 3 2 1

$dev
[1]  690.2346  692.8691  683.0423  717.4446  717.4446  743.3604  793.1266 1073.0916

$k
[1]      -Inf  12.03823  14.92474  25.76707  26.02613  38.91686  50.61655 298.68751

$method
[1] "deviance"

attr(,"class")
[1] "prune"         "tree.sequence"
```

The tree with 6 terminal nodes has the lowest cross-validation error, with a deviance of 683.0423. This indicates that a tree of this size is likely to perform best when generalizing to unseen data. Other sizes result in higher error rates, with smaller trees not fitting the data well enough and larger trees potentially overfitting. Hence, the optimal tree size for the dataset appears to be the one with 6 terminal nodes.

Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis



The plot shows us that as the number of terminal nodes increases from 1 to 6, the cross-validated error decreases, indicating an improvement in the model's accuracy. However, after reaching 6 terminal nodes, the error rate stabilizes, suggesting that additional complexity (more nodes) does not lead to significant improvements in model accuracy.

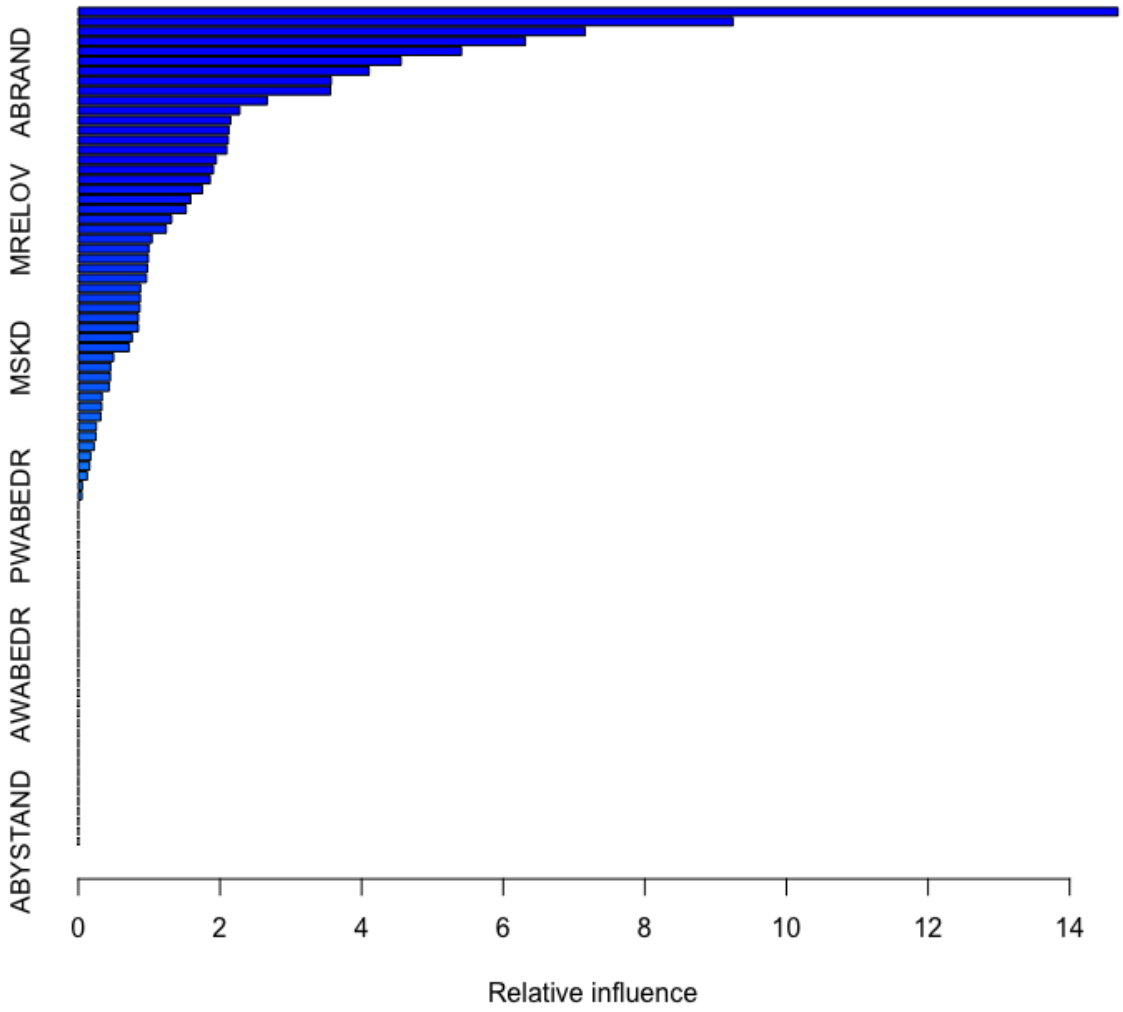
Problem 2 - Caravan data set

First I analyzed the dataset and found that the ‘Purchasing’ column needs to convert to a factor with Yes as 1 and No as 0. Then I encountered an error while training the model due to variables with no variation. So I remove any predictor variables that have no variation (like 'PVRAAUT' and 'AVRAAUT’)

Create a training set consisting of the first 1,000 observations, and a test set consisting of the remaining observations.

For the analysis of the Caravan dataset, the data was divided into two sets: a training set and a test set. The training set was created with the first 1,000 observations, and the test set comprised the remaining observations.

Fit a boosting model to the training set with Purchase as the response and the other variables as predictors. Use 1,000 trees, and a shrinkage value of 0.01. Which predictors appear to be the most important?



I utilized a gradient boosting model (gbm) with 1,000 trees and a shrinkage rate of 0.01 to predict caravan insurance purchases. The model's variable importance plot revealed that the number of fire policies (ABRAND) is the most influential predictor. This suggests a potential correlation where customers with fire policies are more likely to purchase caravan insurance, perhaps indicating a general propensity for insurance products.

Marital status (MRELGE) and social class (MSKD) also emerged as significant factors. MRELGE represents married individuals, while MSKD corresponds to the lower social class, indicating these demographic aspects are influential in predicting insurance purchasing behavior.

These findings are highly informative. The prominence of ABRAND in the model can guide targeted cross-selling strategies. Likewise, understanding the influence of MRELGE and MSKD enables a more nuanced approach to marketing, allowing for campaigns that resonate with specific customer groups. This analytical approach is invaluable for optimizing resource allocation in marketing strategies.

Use the boosting model to predict the response on the test data. Predict that a person will make a purchase if the estimated probability of purchase is greater than 20 %. Form a confusion matrix. What fraction of the people predicted to make a purchase do in fact make one? How does this compare with the results obtained from applying KNN or logistic regression to this data set?

Boosting Model

In this analysis, I compared the performance of a boosting model and a logistic regression model in predicting caravan insurance purchases. The objective was to predict if a person would make a purchase based on an estimated probability greater than 20%.

For the boosting model, after fitting with 1,000 trees and a shrinkage value of 0.01, I predicted the test set responses. I classified a response as a purchase if the predicted probability exceeded 20%. The confusion matrix for the boosting model was as follows:

```
> print(conf_matrix)
   test_pred_class
       0    1
  0 4420  113
  1  256   33
> print(fraction_correct)
[1] 0.2260274
```

From the confusion matrix, out of the 146 cases where the model predicted a purchase, 33 were true positives, meaning the customer did indeed make a purchase. The fraction of correct predictions among those predicted to make a purchase is approximately 22.60%

Logistic Regression

The confusion matrix from the logistic regression model, with the same 20% probability threshold for predicting a purchase, is presented as follows

```
> print(conf_matrix_logistic)
   logistic_pred_class
       0    1
  0 4183  350
  1  231   58
> print(logistic_precision)
[1] 0.1421569
```

From the logistic regression predictions, out of the 408 cases predicted as a purchase, there are 58 true positives where the customers did indeed make a purchase. We get 14.21% as the accuracy or fraction of correct prediction. Since it identified a higher number of potential buyers, this could be preferable in scenarios where the cost of false positives is less significant compared to the potential revenue from true positives.

Comparison between Boosting and Logistic Regression

The boosting model showed higher precision at 22.60% compared to logistic regression's 14.22%. This suggests that boosting is more effective in accurately identifying potential insurance buyers with fewer false positives. On the other hand, logistic regression, despite being less precise, might be better for broader marketing strategies due to its ability to identify more potential customers. The choice between these models hinges on whether we prioritize precision or the overall number of identified potential customers

Problem 3 - Simulated Data

In this problem, I will generate simulated data, and then perform PCA and K-means clustering on the data.

Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables. Hint: There are a number of functions in R that you can use to generate data. One example is the rnorm() function; runif() is another option. Be sure to add a mean shift to the observations in each class so that there are three distinct classes.

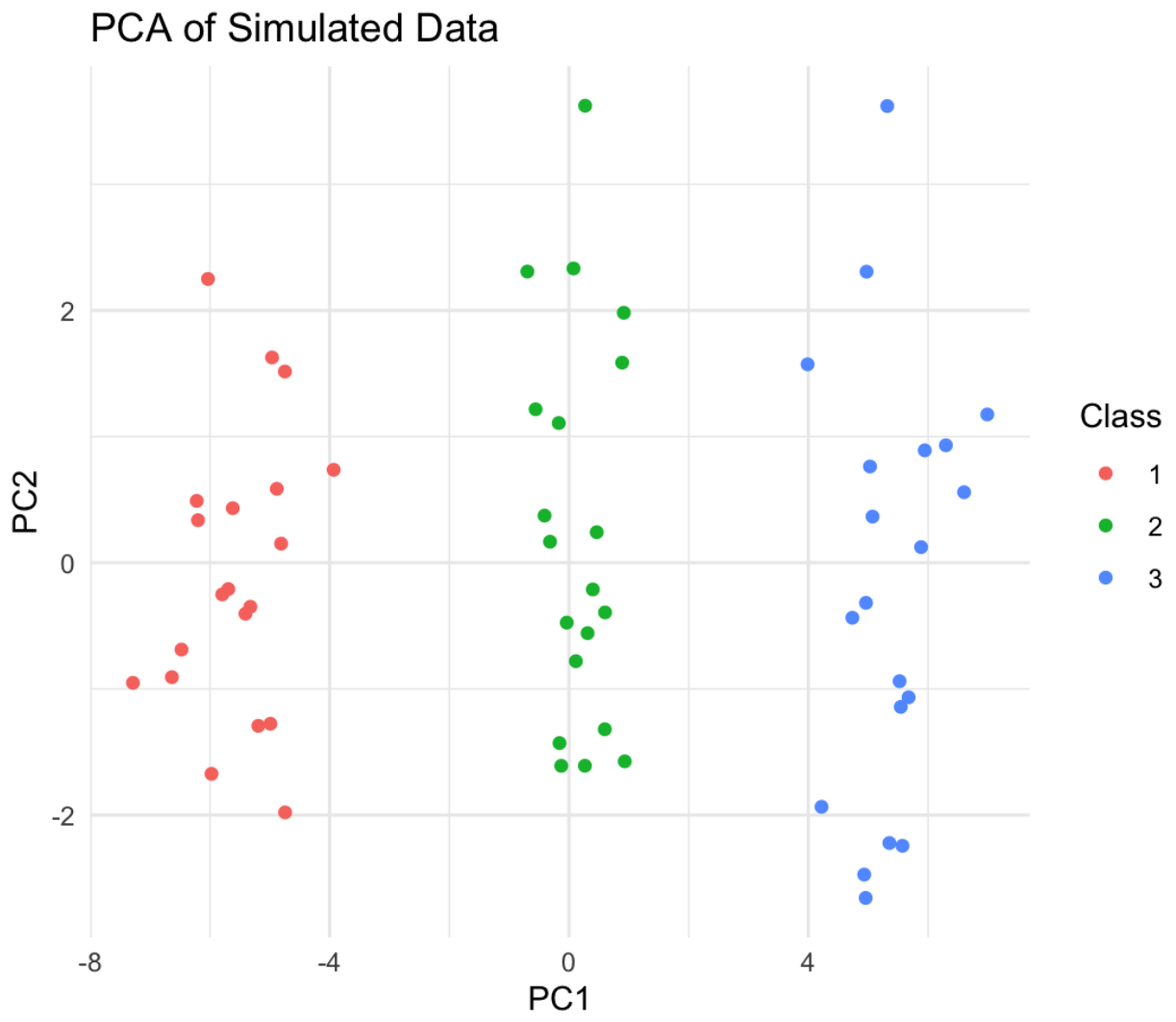
For generating the simulated data, I used R's rnorm() function to create 50 variables for each observation. To ensure three distinct classes, each class was assigned a unique mean shift. This mean shift differentiates the classes by altering the central tendency of the data points in each class. The dataset consists of 20 observations per class, resulting in a total of 60 observations. The mean for each class is shifted by 0, 1, and 2 respectively. Finally, the matrices are combined into one dataset using rbind. This method ensures that the simulated data has distinct classes.

```
> str(data)
 num [1:60, 1:50] -0.5605 -0.2302 1.5587
```

The structure of the dataset, as verified by the str() function, confirms the appropriate dimensions and data type.

Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes. Do not continue to part (c) until the three classes show at least some separation in the first two principal component score vectors.

Used prcomp() function to perform PCA in the data and then created a dataframe with the PCA scores. Then I plot the first two PCA score vectors.



The PCA score plot of the simulated dataset clearly shows three distinct clusters corresponding to the three classes. The first principal component (PC1) and the second principal component (PC2) capture the majority of the variance and separate the classes well. Each class is color-coded—red for Class 1, green for Class 2, and blue for Class 3—demonstrating the effectiveness of the mean shift introduced during the data generation process in creating distinct groups.

With this successful separation in the PCA score plot, we can proceed to perform K-means clustering with K=3 on the dataset.

Perform K-means clustering of the observations with K = 3. How well do the clusters that you obtained in K-means clustering compare to the true class labels? Hint: You can use the table() function in R to compare the true class labels to the class labels obtained by clustering. Be careful how you interpret the results: K-means clustering will arbitrarily number the clusters, so you cannot simply check whether the true class labels and clustering labels are the same.

In this analysis, I performed K-means clustering on the dataset with the intention of grouping the 60 observations into three clusters (K=3). I then compared the clusters obtained from K-means with the true class labels. To compare the true labels with the K-means clusters, I used the table() function:

```
> print(comparison_table)
           Cluster
TrueLabels  1  2  3
         1  0 20  0
         2 20  0  0
         3  0  0 20
```

For each true class label (1, 2, and 3), all 20 observations were correctly grouped into their respective clusters (2, 1, and 3 respectively). The cluster numbering does not align with the original class labels because K-means clustering assigns cluster numbers based on the order in which it encounters the clusters during the algorithm's execution. The clustering algorithm's ability to perfectly match the true class labels, despite its arbitrary numbering, suggests that the simulated data was well-separated into distinct classes.

This result indicates that the K-means clustering algorithm was able to perfectly identify and group the observations according to the true classes in the simulated dataset. The separation seen in the PCA score plot was a good predictor of the clustering algorithm's

success. Such clear demarcation and grouping are not always achievable with real-world data, which often has more overlap and less distinct clustering.

Perform K-means clustering with K = 2. Describe your results

```
> print(kmeans_result_k2$cluster)
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[58] 1 1 1
>
> # Compare K-means clusters to true class labels
> comparison_table_k2 <- table(TrueLabels = pca_scores$class, Cluster = kmeans_result_k2$cluster)
> print(comparison_table_k2)
          Cluster
TrueLabels  1  2
         1  0 20
         2 20  0
         3 20  0
>
```

The table indicates that K-means clustering with K=2 clusters has perfectly grouped Class 1 into Cluster 2 but has combined Classes 2 and 3 into Cluster 1. This was expected since we're forcing the model to create only two groups out of three distinct classes. The model consolidates the two classes that are closer together in the feature space into a single cluster.

Now perform K-means clustering with K = 4, and describe your results.

```
> print(kmeans_result_k4$cluster)
 [1] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 1 1 1 2 2 1 1 2 1 1 2 2 1 1 1 1 2 2 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[58] 3 3 3
>
> # Compare K-means clusters to true class labels
> comparison_table_k4 <- table(TrueLabels = pca_scores$class, Cluster = kmeans_result_k4$cluster)
> print(comparison_table_k4)
          Cluster
TrueLabels  1  2  3  4
         1  0  0  0 20
         2 13  7  0  0
         3  0  0 20  0
```

With K=4, the model has introduced an additional cluster, which has led to some fragmentation. Class 1 is still perfectly isolated in Cluster 4. Class 3 is perfectly grouped into Cluster 3. However, Class 2 has been split between Clusters 1 and 2, indicating that within Class 2, the model has detected what it interprets as two subgroups. So, with K-means, when we increase the number of clusters beyond the true number of classes; the algorithm will attempt to find additional structure, even if none exists. **This behavior underscores the importance of selecting the appropriate K value.**

Now perform K-means clustering with K = 3 on the first two principal component score vectors, rather than on the raw data. That is, perform K-means clustering on the 60 × 2 matrix of which the first column is the first principal component score vector, and the second column is the second principal component score vector. Comment on the results.

To perform K-means clustering on the first two principal component score vectors, we will use the scores obtained from the PCA analysis earlier. We will cluster the 60x2 matrix where each row is an observation and the two columns are the first and second principal component scores.

```
> print(kmeans_result_pca$cluster)
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[58] 3 3 3
>
> # Compare K-means clusters to true class labels
> comparison_table_pca <- table(TrueLabels = pca_scores$class, Cluster = kmeans_result_pca$cluster)
> print(comparison_table_pca)
          Cluster
TrueLabels  1  2  3
         1  0 20  0
         2 20  0  0
         3  0  0 20
```

Each class has been exclusively assigned to a single cluster: Class 1 to Cluster 2, Class 2 to Cluster 1, and Class 3 to Cluster 3, without any misclassification. This indicates that the first two principal components effectively captured the underlying structure of the data and that the clusters in this reduced space are well-separated.

Performing K-means on the PCA-reduced dataset has yielded results that mirror the true class distribution precisely. This demonstrates the power of PCA to distill the most informative aspects of the data, enhancing the performance of clustering algorithms like K-means. In real-world scenarios, such clear-cut clustering might be less common, but the technique of combining PCA with K-means can often lead to more meaningful clustering by reducing noise and focusing on the most significant variation within the data.

Using the scale() function, perform K-means clustering with K = 3 on the data after scaling each variable to have standard deviation one. How do these results compare to those obtained in (b)? Explain.

I do scale(data) which performs this operation on dataset data. Each column in will have a mean of approximately zero and a standard deviation of one after this operation, ensuring that all variables contribute equally to the K-means clustering process, regardless of their original scale

The results from the K-means clustering on the scaled data show a perfect correspondence between the clusters and the true class labels:

```
> print(comparison_table_scaled)
           Cluster
TrueLabels  1  2  3
         1  0 20  0
         2 20  0  0
         3  0  0 20
```

Just like in part (b), where clustering was performed on the raw data, each class is exclusively assigned to a single cluster with no misclassifications. For both the raw and scaled data, the clusters perfectly match the true class labels, demonstrating that the underlying structure of the classes is strong and clearly defined, which is easily captured by K-means clustering.

This perfect clustering outcome might be less likely with more complex real-world data, where classes are not as well-separated and the data contains more noise. However, in this simulated example, both the PCA and scaling approaches have led to clear and distinct clusters that align perfectly with the true class labels. This suggests that the variables contribute to the class separation equally, and thus, scaling has not changed the relative distances between observations from different classes.