Methods & Dataset

Problem Description

By defining the success of the marketing campaign as the response rate, this report aims to provide a method of analysing retail marketing campaign strategies. This is performed by visualising responses to marketing campaigns for each customer cluster created through Customer Segmentation.

Through this, it can also be hypothesized whether the retail company adopts target marketing and serve as potential supporting information for future marketing strategies.

Customer Segmentation will be performed through unsupervised machine learning clustering algorithms where customers are clustered according to the similarity in demographics and purchasing patterns. To optimise clustering, required data would be common demographics and purchasing information. Additionally, these customer's responses to marketing campaigns are required to map each cluster to response type.

Dataset Introduction

The dataset used for this report is obtained from MarketingCampaign, 2019 (https://www.kaggle.com/rodsaldanha/arketing-campaign). It contains customer demographic information, purchasing information, and response to Marketing Campaigns. This dataset has a total of 29 attributes and 2240 observations.

To analyse campaign response, customer segmentation will be performed to gain a visualisation of the customer clusters within the retail.

3.3 Dataset Feature Description

This section aims to describe features found within the MarketingCampaign, 2019 dataset. The 27 attributes are grouped into 5 categories according to the informative purpose.

- Category 1: Campaign Response

Variable	Description
AcceptedCmp1	Denotes customer’s response to marketing offer in the 1st campaign

AcceptedCmp2	Denotes customer's response to marketing offer in the 2nd campaign
AcceptedCmp3	Denotes customer's response to marketing offer in the 3rd campaign
AcceptedCmp4	Denotes customer's response to marketing offer in the 4th campaign
AcceptedCmp5	Denotes customer's response to marketing offer in the 5th campaign
Response	Denotes customer's response to marketing offer in the 6th campaign

- Category 2: Demographic Information

Variable	Description
Year_Birth	Customers year of birth
DtCustomer	Date of customer's enrolment with the company
Education	Customer's level of education
Marital	Customer's marital status
Kidhome	Number of children in customer's household
Teenhome	Number of teenagers in customer's household
Income	Customer's annual household income

- Category 3: Spending amount on retail categories

Variable	Description
MntFishProducts	Monetary amount spent on fish products in the last 2 years
MntMeatProducts	Monetary amount spent on meat products in the last 2 years
MntFruits	Monetary amount spent on fruits products in the last 2 years
MntSweetProducts	Monetary amount spent on sweet products in the last 2 years
MntWines	Monetary amount spent on wine products in the last 2 years
MntGoldProds	Monetary amount spent on gold products in the last 2 years

- Category 4: Amount of purchases through multiple channels (the information here maybe can remove)

Variable	Description
NumDealsPurchases	Frequency of purchases made when products are on discount
NumCatalogPurchases	Frequency of purchases made through a catalogue
NumStorePurchases	Frequency of purchases made directly in stores
NumWebPurchases	Frequency of purchases made through company's web site

- Category 5: Additional Information

Variable	Description
Complain	Indicates whether the customer has made complaints within the last 2 years
NumWebVisitsMonth	Number of visits to company's web site in the last month
Recency	Number of days since the last purchase from the store

Data Exploration

Initial Data Exploration

The datatypes of all attributes are converted to numerical, factor, or date types according to their purposes through running code shown in Code Snippet 1. After handling datatypes of the attributes, datatype of each attribute is observed in Figure 1.

Variables converted in each category (Categories are listed in Section 3.3):

- To numeric type: Year_Birth, KidHome, Teenhome and Income from Category 2, all Category 3 variables, all Category 4 variables and Recency from Category 5.
- To date type: Dt_Customer from Category 2
- To factor type: Education, Martial_Status from Category 2, all Category 1 variables.

```
#convert numeric columns to hold numeric data
#columns not to convert to numeric: Education, Marital_Status, Dt_Customer
numer_col <- col.names[c(c(2,5,6,7,9),seq(10,20))]
marketing <- mutate_at(marketing, numer_col, as.numeric)

#convert Dt_customer column to date type
marketing <- mutate_at(marketing, col.names[8], as.Date)

#convert columns to factor type: Education, Marital_Status, all AcceptedCmpn variables
fact_col <- col.names[c(c(3,4),seq(21,26))]
marketing <- mutate_at(marketing, fact_col, as.factor)

str(marketing)
```

Code Snippet 1. Handling datatypes

```
> str(marketing)
'data.frame':   2240 obs. of  27 variables:
 $ ID                 : chr  "5524" "2174" "4141" "6
 $ Year_Birth         : num  1957 1954 1965 1984 198
 $ Education          : Factor w/ 5 levels "2n Cycle
 $ Marital_Status     : Factor w/ 8 levels "Absurd",
 $ Income             : num  58138 46344 71613 26646
 $ Kidhome            : num  0 1 0 1 1 0 0 1 1 1 ...
 $ Teenhome           : num  0 1 0 0 0 1 1 0 0 1 ...
 $ Dt_Customer        : Date, format: "2012-09-04" "
 $ Recency            : num  58 38 26 26 94 16 34 32
 $ MntWines           : num  635 11 426 11 173 520 2
 $ MntFruits          : num  88 1 49 4 43 42 65 10 0
 $ MntMeatProducts    : num  546 6 127 20 118 98 164
 $ MntFishProducts    : num  172 2 111 10 46 0 50 3
 $ MntSweetProducts   : num  88 1 21 3 27 42 49 1 3
 $ MntGoldProds       : num  88 6 42 5 15 14 27 23 2
 $ NumDealsPurchases  : num  3 2 1 2 5 2 4 2 1 1 ...
 $ NumWebPurchases    : num  8 1 8 2 5 6 7 4 3 1 ...
 $ NumCatalogPurchases: num  10 1 2 0 3 4 3 0 0 0 ..
 $ NumStorePurchases  : num  4 2 10 4 6 10 7 4 2 0 .
 $ NumWebVisitsMonth  : num  7 5 4 6 5 6 6 8 9 20 ..
 $ AcceptedCmp3       : Factor w/ 2 levels "0","1":
 $ AcceptedCmp4       : Factor w/ 2 levels "0","1":
 $ AcceptedCmp5       : Factor w/ 2 levels "0","1":
 $ AcceptedCmp1       : Factor w/ 2 levels "0","1":
 $ AcceptedCmp2       : Factor w/ 2 levels "0","1":
 $ Complain           : Factor w/ 2 levels "0","1":
 $ Response           : Factor w/ 2 levels "0","1":
```

Figure 1. Final data types after cleaning

Identifying Missing Values

In the R environment, blank data are not treated as missing values. Therefore, a mutate function is adopted to ensure all blank data are converted to missing values. The final number of missing values can then be identified. The code used is shown in Code Snippet 2. As shown in Figure 2, 24 missing values are found.

```
#check for missing values
#replace all blanks with NA
marketing <- mutate_all(marketing,na_if,"")
sum(is.na(marketing))
```

Code Snippet 2. Identifying number of missing values

```
> sum(is.na(marketing))
[1] 24
```

Figure 2. Number of missing values found

To ensure that missing values are only within the Income attribute, visualisations are generated in Figure 3 and 4. It is shown that the missing values are all found in the Income attribute. These missing values make up 1.07% of the data.

Through adopting the VIM package, a combined aggregate plot is produced as shown in Figure 4. The combined aggregate plot visualizes the combination of row types available in the dataset: Rows with all data present, and rows with missing Income observation. The values shown on the right indicate the frequencies of the respective combination.

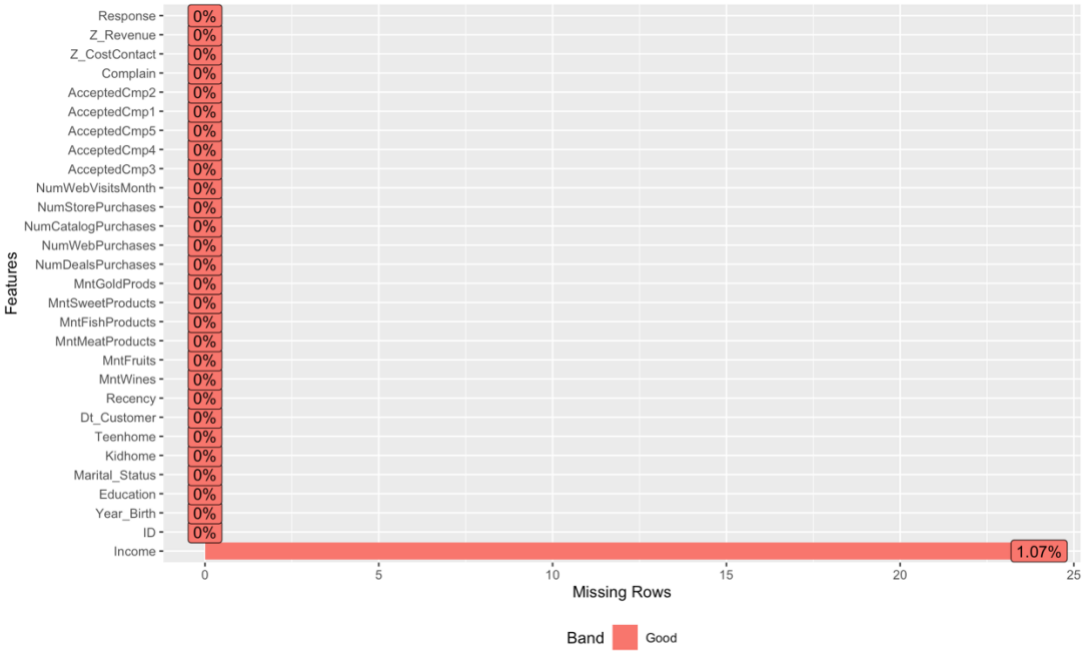

Figure 3. Plot for missing values found in variables

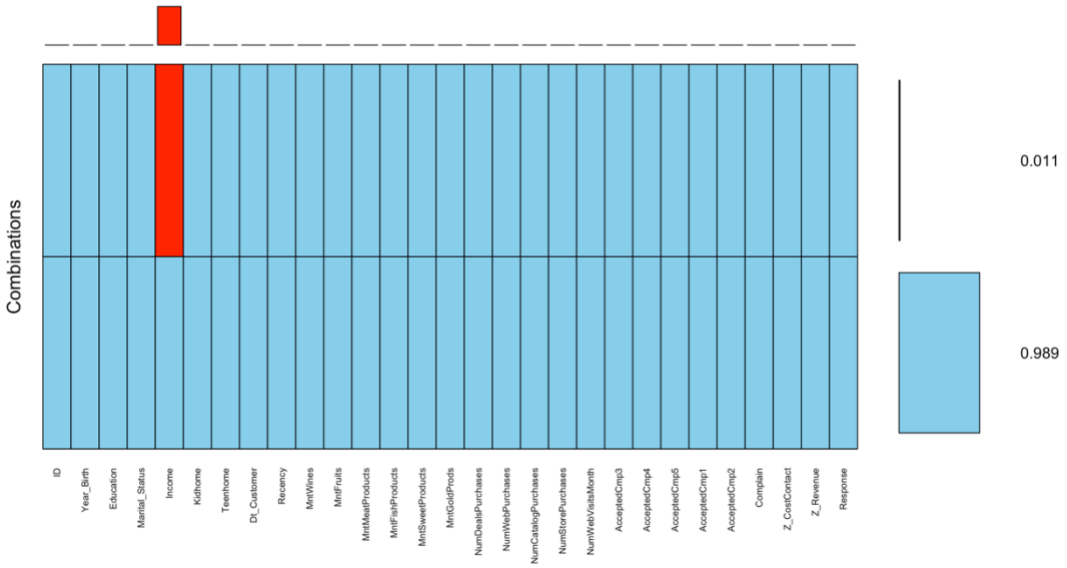

Figure 4. Aggregate plot for missing values

Exploration Data Analysis

Numerical Correlation Matrix

A numerical correlation plot visualizes the correlation between all numerical variables in the dataset. The correlation coefficient is expressed as a range of -1 to 1 where -1 indicates a strong negative correlation, 0 indicates no correlation, and 1 indicates a strong positive correlation. Commonly, the boundary which defined correlation is set at 0.5. Within the numerical correlation plot shown in Figure 4, it is observed that recency does not correlate with any other numerical variables while the customer's number of visits to the website has a consistent negative correlation to all of the numerical variables, except for the number of purchases made on deals.

Amongst the negative correlations between the number of web visits per month and other variables, the most significant is the number of store purchases and the amount spend on meat products. This can be interpreted into customers who visit the retailer's web store are less likely to perform purchases in the physical stores and spend less on meat products.

The numeric variables with the strongest correlation are number of purchases made through catalogues and amount spent on meat products. This implies that customers tend to have higher spending on meat products when purchases are being made through catalogues.

```
#correlation plot between numeric vectors
ggcorr(marketing,low = "steelblue", mid = "white", high = "darkred", label=TRUE,
label_size=3,label_alpha=TRUE,hjust=0.85,size=3)
```

Code Snippet 3. Numerical Correlation Plot

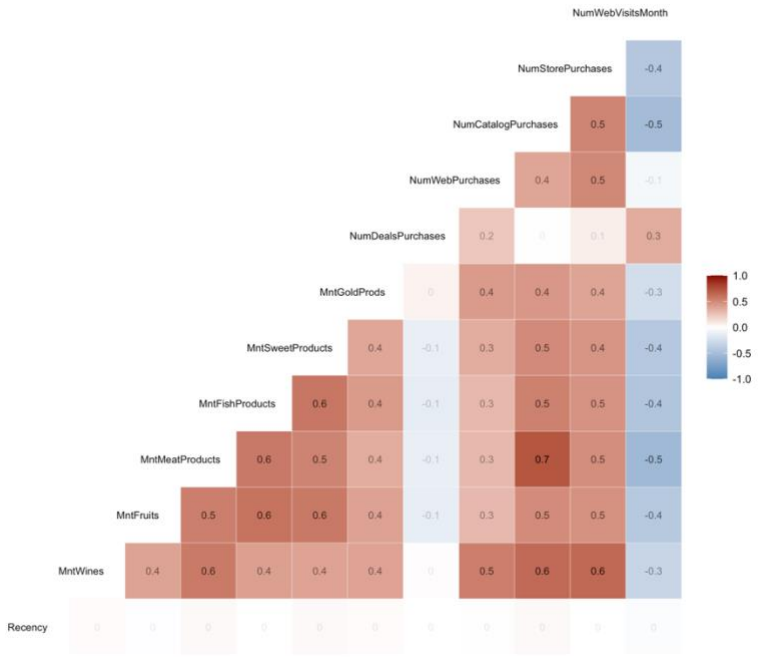

Figure 5. Numerical Correlation Plot

Univariate Analysis

Categorical Features

- **Marital_Status: Marital Status of customers**

There is a total of 8 Marital Status amongst customers: Married, Together, Single, Divorced, Widow, Alone, Absurd and YOLO.

In order to create a more efficient grouping system, Marital Status factor levels can be further minimised into Coupled and Single through grouping levels of "Married" and "Together" into a Coupled level, and "Single", "Divorced", "Widow", and "Alone" into the Single level as shown in Code Snippet 6 and Figure 10. As the true meaning of Martial Status "Absurd" and "YOLO" are unclear and only has 1.8% proportion, it was be removed.

```
#Marital status variable
marketing %>%
```

```
group_by(Marital_Status) %>%
summarise(Count = n(),Perc=round(n()/nrow(.)*100,2)) %>% arrange(desc(Count))
```

Code Snippet 4. Exploring 'Marital_Status' variable

```
# A tibble: 8 x 3
  Marital_Status Count  Perc
  <fct>          <int> <dbl>
1 Married          864 38.6
2 Together         580 25.9
3 Single           480 21.4
4 Divorced         232 10.4
5 Widow             77  3.44
6 Alone              3  0.13
7 Absurd             2  0.09
8 YOLO               2  0.09
```

Figure 6. 'Marital_Status' variable information

```
#Marital Status
#Create New Cohesive Categories
marketing$Rel_Status[marketing$Marital_Status %in% c('Alone', 'Divorced', 'Widow',
'Single')] <- 'Single'
marketing$Rel_Status[marketing$Marital_Status %in% c('Married', 'Together')] <- 'Coupled'
```

Code Snippet 5. Feature Engineering for Marital Status

```
> table(marketing$Rel_Status)

Coupled  Single
   1444     792
```

Figure 7. Engineered Marital Status Feature

- Education: Education level of customers

Amongst retail customers, there are 5 education levels: Graduation, PhD, Master, 2n Cycle, and Basic. In order to disambiguate the implications of each of these education levels, brief descriptions are provided:

Graduation: Completion of a degree or foundation studies are a college or university.

PhD: The highest level of academics.

Master: A postgraduate level of studies which is a level further from a Degree programme.

2n Cycle: Second Cycle Degree Programme which is a second level of university programme mostly carried out in European countries. It is comparable to a Master's programme.
Basic: Completion of high school.

The results shown in Figure 911shows that the majority of 50.3% amongst the retail's customers achieve Graduation level of education while only 2.41% achieve a Basic level of education. Feature engineering is performed on Education feature through Code Snippet 7 to categorise Education levels into Below Degree, Degree, and Postgraduate as shown in Figure 12.

```
#Education variable
marketing %>%
 group_by(Education) %>%
  summarise(Count = n(),Perc=round(n()/nrow(.)*100,2)) %>% arrange(desc(Count))
```

Code Snippet 6. Exploring 'Education' variable

```
# A tibble: 5 x 3
  Education  Count  Perc
  <fct>      <int> <dbl>
1 Graduation  1127 50.3
2 PhD          486 21.7
3 Master       370 16.5
4 2n Cycle     203  9.06
5 Basic         54  2.41
```

Figure 8. 'Education' variable information

```
#Education into Below Degree, Degree, and Postgraduate
table(marketing$Education)
marketing$EducationLevel[marketing$Education %in% c('Basic')] <- 'Below Degree'
marketing$EducationLevel[marketing$Education %in% c('Graduation')] <- 'Degree'
marketing$EducationLevel[marketing$Education %in% c('PhD','Master','2n Cycle')] <-
'Postgraduate'
table(marketing$EducationLevel)
```

Code Snippet 7. Feature Engineering of Education Feature

```
> table(marketing$EducationLevel)

Below Degree       Degree Postgraduate
          54         1127         1059
```

Figure 9. Education Level feature created through Feature Engineering

Numerical Features

- **Income**

A boxplot is created through running code shown in Code Snippet 8 to visualize customers' range of annual household income as shown in Figure 13. Within the boxplot, it is observed that there is a glaring outlier where household income is greater than 600,000. This can be interpreted as either an exceptionally wealthy customer or an error in data. Since it is a sole data point that falls out of the average income range of our customer base, it is an outlier that has to be treated during Data Manipulation.

```
income_box = boxplot(marketing$Income,main = 'Customer Income Range', ylab='Customer Annual Household Income')
```

Code Snippet 8. Boxplot for Customer Income

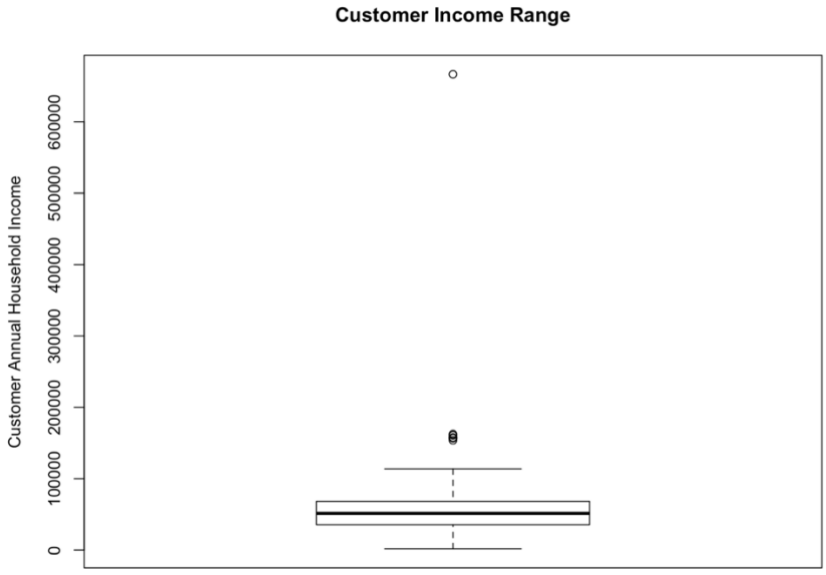

Figure 10. Boxplot for Customer Income

- **Age**

Visualising customers' age consists of two steps found which are included in Code Snippet 9. Firstly, an age column is created by performing calculation with the Year_Birth column in the dataset. Then, the histogram can is plotted by executing the hist() function. Within the histogram shown in Figure 14, it can be seen that majority of customers lie within the ages 40-60. Additionally, outliers are also observed where age is recorded to be beyond 120. This outlier may be assumed to be caused by an error and has to be treated during Data Manipulation.

```
#create age column
marketing$age = 2021 - marketing$Year_Birth


#plot age
age_hist = hist(marketing$age,main = "Histogram of Customer Age",xlab='Customer
Age') ;age_hist
```

Code Snippet 9. Feature Engineering of Age Feature

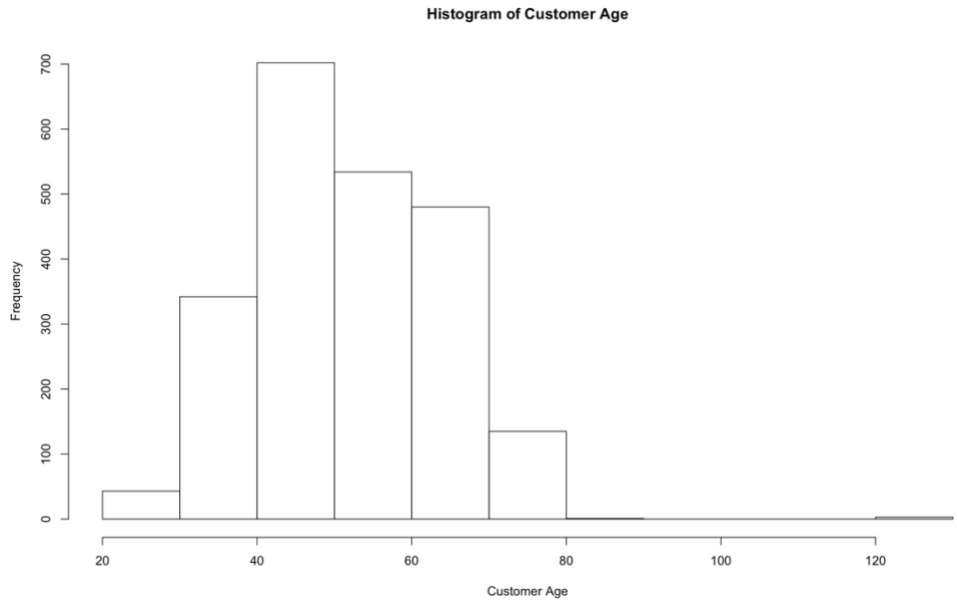

Figure 11. Histogram of Customer Age

- **Marketing Campaign Responses**

Marketing Campaign Responses are plotted to observe the performance of each marketing campaign held. As shown in Figure 15, responses to marketing campaigns are relatively low, especially for Campaign 2. The most recent campaign held, Campaign 6 has the highest response rate and is treated as the most successful campaign held.

```
#plot marketing response
#Create a table for all campaign responses
t1 = as.data.frame(table(marketing$AcceptedCmp1)) ; t1
t2 = as.data.frame(table(marketing$AcceptedCmp2)) ; t2
t3 = as.data.frame(table(marketing$AcceptedCmp3)) ; t3
t4 = as.data.frame(table(marketing$AcceptedCmp4)) ; t4
t5 = as.data.frame(table(marketing$AcceptedCmp5)) ; t5
t6 = as.data.frame(table(marketing$Response)) ; t6

#Combine all responses into a table
responses.df <- c(t1[,2])
responses.df <- rbind(responses.df, c(t2[,2]))
responses.df <- rbind(responses.df, c(t3[,2]))
responses.df <- rbind(responses.df, c(t4[,2]))
responses.df <- rbind(responses.df, c(t5[,2]))
responses.df <- rbind(responses.df, c(t6[,2]))

colnames(responses.df) <- c("0","1")
rownames(responses.df) <- c("Campaign 1","Campaign 2",'Campaign 3',"Campaign
4","Campaign 5",'Campaign 6')

responses.df <- melt(responses.df)
responses.df$Var2 <- as.factor(responses.df$Var2)
responses.df

ggplot(responses.df, aes(fill=Var2, y=value, x=Var1)) +
```

```
  geom_bar(position="dodge", stat="identity") + scale_fill_manual(values =
c("#009E73","#0072B2")) +
  xlab("Marketing Campaigns") + ylab("Reponse Frequency") + ggtitle("Customers Responses
to Marketing Campaigns",subtitle="0 = No Response, 1 = Reponded") +
  guides(fill=guide_legend(title="Customer Response"))
```

Code Snippet 10. Plot stacked barchart for marketing responses

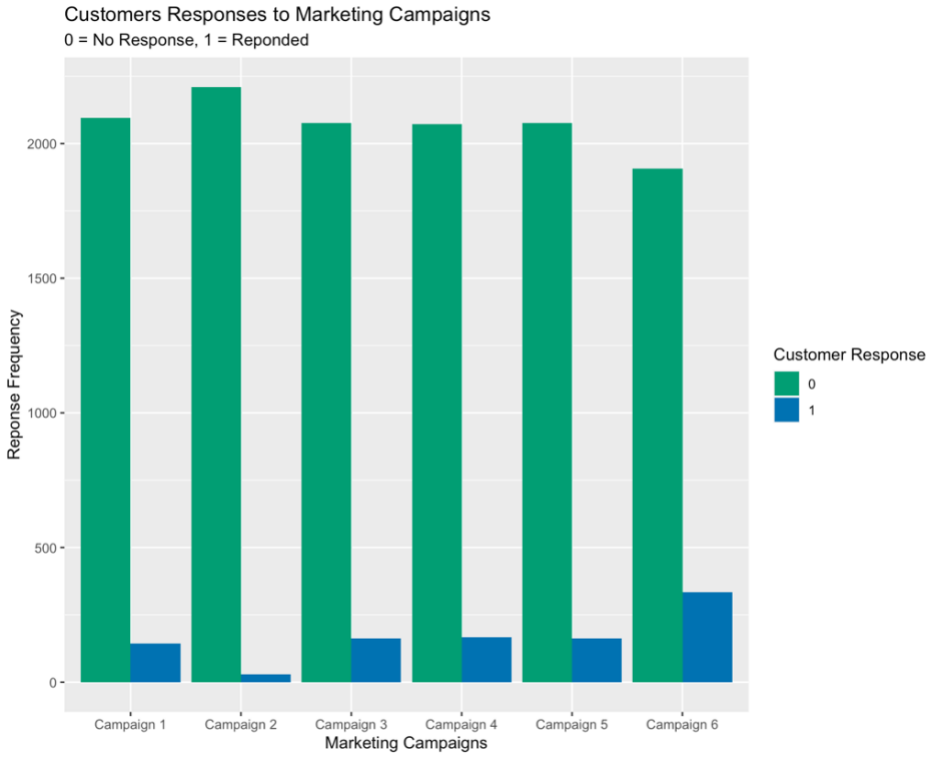

Figure 12. Customer Response to Marketing Campaigns

```
#PastCampaignResponse
marketing$CampaignResponse = as.numeric(marketing$AcceptedCmp1) +
  as.numeric(marketing$AcceptedCmp2) + as.numeric(marketing$AcceptedCmp3) +
  as.numeric(marketing$AcceptedCmp4) + as.numeric(marketing$AcceptedCmp5) +
  as.numeric(marketing$Response)
```

Code Snippet 11. Feature Engineering for Past Campaign Responses

- **Kid_home and Teen_home**

As both of the features indicate the number of children customers have, the two features are aggregated through code shown in code snippet X to sum up the number of children of each customer.

```
marketing$Num_Child = marketing$Kidhome + marketing$Teenhome
```

Code Snippet 12. Feature Engineering for Number of Children feature

Bivariate Analysis

- **Relationships between Income and Total Spending**

To obtain a visualization of the relationship between a customer's annual household income and the total amount spent within 2 years, a total spending column is created which totals the value of all Category 3 variables.

The scatter plot generated is shown in Figure 16 whereby a directly proportional relationship is observed between Annual Household Income and Total Spending. This is logical as customers with a higher annual household income can afford to have higher spending.

```
#plot income vs total spending
#create new column for total spending
marketing$total_spending <- marketing$MntFishProducts + marketing$MntMeatProducts +
marketing$MntFruits + marketing$MntSweetProducts + marketing$MntWines +
marketing$MntGoldProds

#plot income vs total spending
plot(marketing$total_spending, marketing$Income , xlab = 'Total spendings (in 2 years)',
ylab="Annual Household Income",main = 'Houshold Income vs Total Spendings',
pch=19,cex=0.5)
```

Code Snippet 13. Scatter Plot for Annual Household Income against Total Spending in 2 years

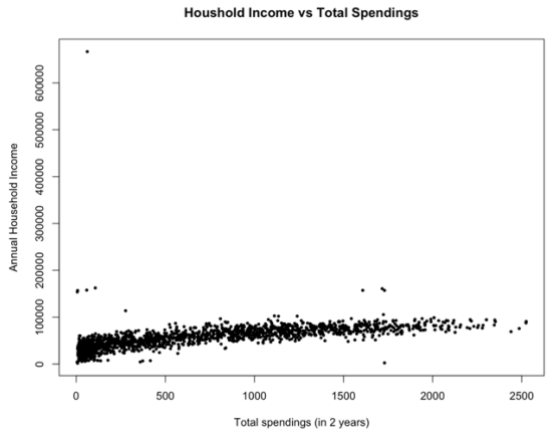

Figure 13. Code Snippet 25. Scatter Plot for Annual Household Income against Total Spending in 2 years

- **Relationship Between Recency and Total Spending**

The relationship between Recency and Total Spending may provide an insight into customer satisfaction towards the products sold in the retail business. If there is a majority of customers with long recency associated with low total spending, it can be assumed that the product range available in the store is too narrow. Hence, customers were not able to purchase what they wished and had not returned to the store recently.

After executing the code in Code Snippet 15, the scatter plot is produced in Figure 17 shows that there is no relationship and correlation between recency and total spending.

```
#plot recency vs total spending
recencyplot = plot(marketing$Recency, marketing$total_spending, pch=19, cex=0.5,
xlab="Recency",ylab="Total spending in 2 years", main='Total spending vs Recency')
```

Code Snippet 14. Scatter plot for Total Spending against Recency

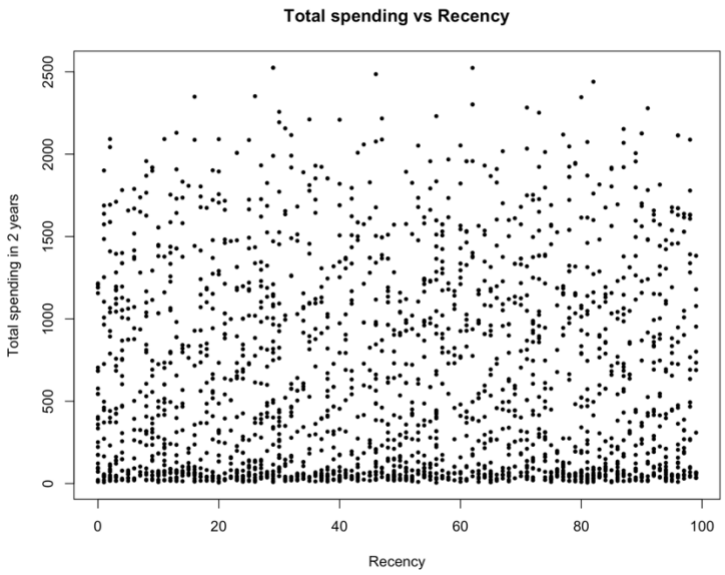

Figure 14. Scatter plot for Total Spending against Recency

Data Preparation

Handling Missing Values

In this report, MICE is implemented through executing code shown in Code Snippet X to perform missing data imputation. MICE is a multiple imputation method which replaces missing data that assumes that data are Missing At Random (MAR). The advantage of adopted MICE for missing data imputation would be the ability of incorporating statistical uncertainties into imputation.

```
library("mice")

marketing_miceimpute <- mice(marketing, m=3)
marketing_miceimpute <- complete (marketing_miceimpute)

marketing_miceimpute%>%
summarise( is_NULL=sum(is.na(Income)==1),is_NOT_NULL=sum(!is.na(Income)==1),Max
=max(Income), Min=min(Income), Mean=mean(Income), Median=median(Income),
QUA1=quantile(Income,1/4), QUA3=quantile(Income,3/4), IQR=IQR(Income)
)
```

Code Snippet 15. MICE Imputation for Missing Values

One Hot Encoding

One Hot encoding is the process of representing categorical variables as integers to allow better performance of machine learning algorithms

```
#Label Encoding for Marital Status
marketing$Rel_Single[marketing$Rel_Status %in% c('Single')] <- 1
marketing$Rel_Single[marketing$Rel_Status %in% c('Coupled')] <- 0
marketing$Rel_Coupled[marketing$Rel_Status %in% c('Single')] <- 0
marketing$Rel_Coupled[marketing$Rel_Status %in% c('Coupled')] <- 1

#Label Encoding for Education Level
marketing$Ed_Below_Degree[marketing$EducationLevel%in% c('Below Degree')] <- 1
marketing$Ed_Below_Degree[marketing$EducationLevel %in% c('Degree','Postgraduate')] <-
0
marketing$Ed_Degree[marketing$EducationLevel %in% c('Degree')] <- 1
marketing$Ed_Degree[marketing$EducationLevel%in% c('Below Degree','Postgraduate')] <-
0
marketing$Ed_Postgraduate[marketing$EducationLevel %in% c('Postgraduate')] <- 1
marketing$Ed_Postgraduate[marketing$EducationLevel%in% c('Degree', 'Below Degree')] <-
0
```

Feature Selection

In Feature Selection, irrelevant columns and features previously used for engineering are removed.

```
#Remove ZCost and ZContact variables - no information provided for these variables
marketing <- marketing [ ,-c(27,28)]



#Remove ID variable – irrelevant column
marketing <- marketing [ ,-c(1)]



#Remove columns used to perform feature engineering (IYearofBirth, Education, Marital Status, KidHome, TeenHome,
marketing <- marketing[-c(1,2,3,4,5,6,7,8,29,30)]
```

Data Splitting

Prior to model implementing, data splitting is executed to obtain a Training Set and Testing Set. In this report, the Pareto Principle is adopted where an 80:20 split is used. Data Splitting is performed through executing code shown in Code Snippet 16. After performing data splitting, it is observed from Figure 18 that training set has dimensions of 1788 rows x 32 columns while test set has dimensions of 447 rows x 32 columns.

```
#Perform 80:20 split
split = sample.split(marketing$Response, SplitRatio = 0.8)

train_set = subset(marketing, split == TRUE)
test_set = subset(marketing, split == FALSE)

dim(train_set)
dim(test_set)
```

Code Snippet 16. 80:20 Data Splitting

```
> dim(train_set)
[1] 1788   32
> dim(test_set)
[1] 447  32
```

Figure 15. Dimensions of Train and Test Set

Data Balancing: Oversampling with SMOTE

As class imbalance is observed through Figure 19, oversample is performed through implementing of the SMOTE technique described in Section 5.4. SMOTE function from DMwR package is used to perform oversampling as shown in Code Snippet 17. After execution of SMOTE, class distribution is shown in Figure 20 where classes are balanced with a 50% : 50% distribution.

```
> prop.table(table(train_set$Response))

        0         1
0.8512304 0.1487696
```

Figure 16. Class Imbalance in Train Set

```
#~~~~~~~~~~~~~SMOTE Oversampling~~~~~~~~~~~~~~~~~~~~~~~~~~

library(DMwR)

#perc.over = oversampling of minority class
#perc.under = undersampling of majority class
balanced_set = SMOTE(Response ~., train_set, perc.over = 100, perc.under=200)

#After implementing SMOTE
prop.table(table(balanced_set$Response)) #50% of class 0 and 50% of class 1
```

Code Snippet 17. SMOTE Oversampling

```
> prop.table(table(balanced_set$Response))

  0   1
0.5 0.5
```

Figure 17. Class Balance after SMOTE

Model Implementation and Validation

This section aims to present the implementation and evaluation of experiments to achieve the optimisation of customer's marketing response prediction. The design of each experiment is presented below in Table 2, 3, and 4. In this study, parameters used to evaluate model performances are: Accuracy, F1 Score, and AUC.

Experiment 1: Decision Tree

Model Name	Model Used	Hyperparameter Tuning
DecisionTree_Basic	Decision Tree	No
DecisionTree_Tuned	Decision Tree	Yes

Table 1. Decision Tree Experiment Design

Experiment 2: Logistic Regression

Experiment Name	Model Used	Hyperparameter Tuning
LogRegression_Basic	Logistic Regression	No

Table 2. Logistic Regression Experiment Design

Experiment 3: Support Vector Machine

Experiment Name	Model Used	Hyperparameter Tuning
SVM_Basic	Support Vector Machine	No
SVM_Tuned	Support Vector Machine	Yes

Table 3. Support Vector Machine Experiment Design

Experiment 1: Decision Tree

Decision Tree Implementation: DecisionTree_Basic

The Decision Tree for Classification algorithm is built through running code shown in Code Snippet 18. Through generated outputs shown in Figure 21, it is observed that a complexity parameter (cp) value of 0.01 is recommended to achieve a minimum cross validation error of 0.3985, denoted by xerror. Following this, CP value of 0.01 is used to build the Decision Tree Classifier.

```
#Perform Tree classification
tree = rpart(Response~ ., data=balanced_train_set, method="class") #basic tree
rpart.plot(tree, extra = 101)
printcp(tree)
plotcp(tree)

tree = rpart(Response~ ., data=balanced_train_set, method="class", cp=0.01)

#Perform prediction with training set
train_pred <- predict(tree, balanced_train_set, type='class')
#Perform prediction with test set
test_pred <- predict(tree, test_set, type='class')
```

Code Snippet 18. DecisionTree_Basic Model

```
Classification tree:
rpart(formula = Response ~ ., data = balanced_trai
n_set, method = "class")

Variables actually used in tree construction:
[1] CampaignResponse    Ed_Degree           Income
              Membership          MntMeatProducts
    NumCatalogPurchases NumStorePurchases
[8] NumWebVisitsMonth   Recency

Root node error: 532/1064 = 0.5

n= 1064

        CP nsplit rel error  xerror     xstd
1 0.454887      0   1.00000 1.09962 0.030504
2 0.046053      1   0.54511 0.54511 0.027302
3 0.030075      3   0.45301 0.48872 0.026347
4 0.016917      4   0.42293 0.47180 0.026032
5 0.015038      5   0.40602 0.46053 0.025813
6 0.014411      6   0.39098 0.45489 0.025702
7 0.011278      9   0.34774 0.42857 0.025159
8 0.010652     12   0.31203 0.40602 0.024663
9 0.010000     15   0.28008 0.39850 0.024491
```

Figure 18. Decision Tree Classifier Results

DecisionTree_Basic Model Validation

To evaluate the performance of DecisionTree_Basic, confusion matrix and ROC curve is generated in figures 22, 23, and 24 to compute performance evaluation parameters: Accuracy, F1 Score, and AUC . The performance parameters and summarised in Table 5.

Through comparing the Train Set Accuracy of 86% and Test Set Accuracy of 74%, it can be seen that the model's performance of train set and test set are satisfactory. Hence, it is concluded that the DecisionTree_Basic model is not overfitted or underfitted. Furthermore, an F1 score of 0.74 and AUC of 77.5% is achieved.

Model	Parameters Used	Parameter Tuning	Train Set Accuracy	Test Set Accuracy	F1 Score	AUC
Decision Tree	Minsplit = 1, Maxdepth = 8, Cp = 0.01	No	86%	74%	0.74	77.5%

Table 4. DecisionTree_Basic Model Performance

```
> confusionMatrix(train_pred, balanced_train_set$Response, positive='1')
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 458  75
         1  74 457

               Accuracy : 0.86
                 95% CI : (0.8376, 0.8803)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.7199

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8590
            Specificity : 0.8609
         Pos Pred Value : 0.8606
         Neg Pred Value : 0.8593
             Prevalence : 0.5000
         Detection Rate : 0.4295
   Detection Prevalence : 0.4991
      Balanced Accuracy : 0.8600

       'Positive' Class : 1
```

Figure 19. Decision Tree Train Set Confusion Matrix

```
> confusionMatrix(test_pred, test_set$Response, positive='1')
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 283  17
         1  98  49

               Accuracy : 0.7427
                 95% CI : (0.6996, 0.7826)
    No Information Rate : 0.8523
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3219

 Mcnemar's Test P-Value : 8.65e-14

            Sensitivity : 0.7424
            Specificity : 0.7428
         Pos Pred Value : 0.3333
         Neg Pred Value : 0.9433
             Prevalence : 0.1477
         Detection Rate : 0.1096
   Detection Prevalence : 0.3289
      Balanced Accuracy : 0.7426

       'Positive' Class : 1
```

Figure 20. Decision Tree Test Set Confusion Matrix

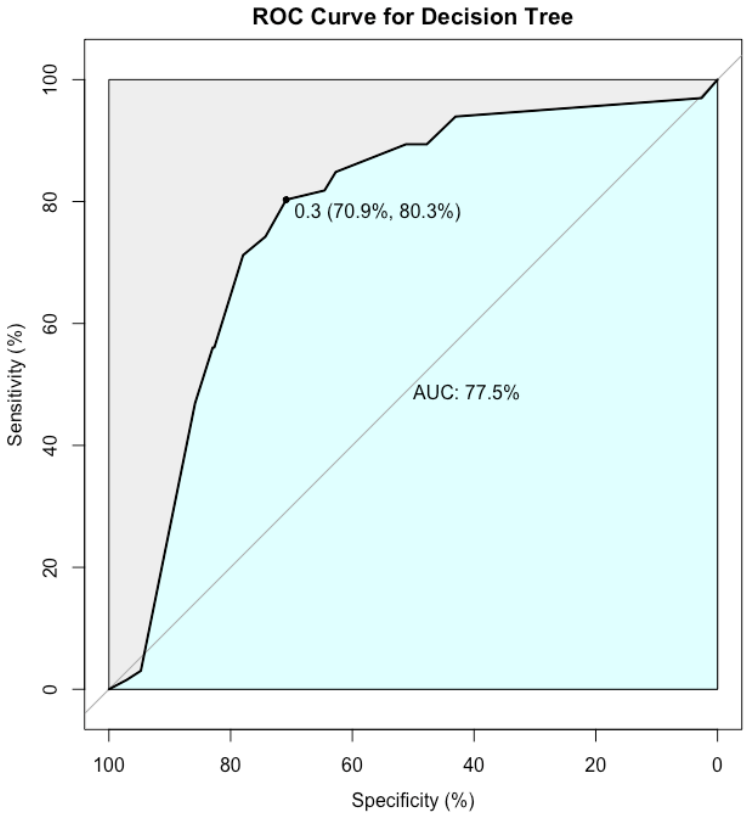

Figure 21. ROC Curve for Decision Tree

Decision Tree with Hyperparameter Tuning: DecisionTree_Tuned

To perform hyperparameter tuning, the Grid Search technique is adopted to optimise minsplit, maxdepth, and complexity parameter (cp) of the Decision Tree. Minsplit represents the minimum number of observations required to perform further splits. Maxdepth represents the maximum depth of the tree which is the length of the longest branch in the tree, and the complexity parameter in the minimum improvement required within each node of the tree.

Code shown in Code Snippet 19 is executed to obtain the tuned parameters as shown in Figure 25. The optimum parameters identified through Grid Search are: Minsplit = 9, Maxdepth = 1, CP = 0.01

The tuned parameters used in the Decision Tree classifier through Code Snippet 20 to perform final predictions on customers' responses to the marketing campaign.

```
#Parameter Tuning with Grid Search
tree_grid <- list(minsplit=seq(1,10,2), maxdepth=seq(1,10,2), cp=seq(0.01,1,0.01)) %>%
  cross_df() # Convert to data frame grid
tree_grid

tree_func <- function (...){tree}

tree_grid = tree_grid %>% mutate(fit = pmap(tree_grid , tree_func))
tree_grid

#Create function to compute accuracy
compute_accuracy <- function(fit, test_features, test_labels) {
  predicted <- predict(fit, test_features, type = "class")
  mean(predicted == test_labels)
}

test_features <- balanced_train_set %>% select(-Response)
test_labels   <- balanced_train_set$Response
tree_grid <- tree_grid %>%
  mutate(test_accuracy = map_dbl(fit, compute_accuracy,
                                 test_features, test_labels))

tree_grid <- tree_grid %>% arrange(desc(test_accuracy), desc(minsplit))
tree_grid

#selected parameters: minsplit=9, maxdepth=1, cp=0.01
```

Code Snippet 19. Grid Search for Decision Tree Hyperparameter Tuning

```
> tree_grid
# A tibble: 2,500 x 5
   minsplit maxdepth    cp fit     test_accuracy
      <dbl>    <dbl> <dbl> <list>          <dbl>
 1        9        1  0.01 <rpart>         0.727
 2        9        3  0.01 <rpart>         0.727
 3        9        5  0.01 <rpart>         0.727
 4        9        7  0.01 <rpart>         0.727
 5        9        9  0.01 <rpart>         0.727
 6        9        1  0.02 <rpart>         0.727
 7        9        3  0.02 <rpart>         0.727
 8        9        5  0.02 <rpart>         0.727
 9        9        7  0.02 <rpart>         0.727
10        9        9  0.02 <rpart>         0.727
# … with 2,490 more rows
```

Figure 22. Grid Search for Decision Tree Tuning

```
#build tree according to tuned parameters
tree = rpart(Response~ ., data=balanced_train_set, method="class", minsplit=9,maxdepth=1, cp=0.01)
rpart.plot(tree, extra = 101)

#Perform prediction with training set
train_pred <- predict(tree, balanced_train_set, type='class')
#Perform prediction with test set
test_pred <- predict(tree, test_set, type='class')
```

Code Snippet 20. DecisionTree_Tuned Model

DecisionTree_Tuned Model Validation

After performing predictions for train and test datasets with the tuned parameters as described in Section 6.1.3, confusion matrix and ROC curve is generated to compute performance parameters: Accuracy, F1 Score, and AUC. The performance of DecisionTree_Basic and DecisionTree_Tuned are summarised and shown in Table 6.

It is observed that after performing hyperparameter tuning, Test Set Accuracy increases while Train Set Accuracy decreases. This implies that the fitness of model is improved. However, it is observed the F1 score and AUC has decreased.

Accuracy describes the proportions of observations classified correctly whereby F1 score is a harmonic mean between precision and recall. When dataset has class imbalance, F1 score is often prioritised as the performance measurement metric as F1 score focuses on the performance of ‘positive’ classifications.

In this report, SMOTE has been performed to achieve a class balance. Thus, accuracy is prioritised and defined as the target function. Through this, it can be concluded that the performance of DecisionTree_Tuned is better than DecisionTree_Basic.

Model	Parameters Used	Parameter Tuning	Train Set Accuracy	Test Set Accuracy	F1 Score	AUC
DecisionTree_Basic	Minsplit = 1, Maxdepth = 8, Cp = 0.01	No	86%	74%	0.74	77.5%
Decision_Tree_Tuned	Minsplit = 9, Maxdepth = 1, Cp = 0.01	Yes	72.4%	79.8%	0.43	68.8%

Table 5. DecisionTree_Basic vs DecisionTree_Tuned Model Performance

```
> confusionMatrix(train_pred, balanced_train_set$Response, positive='1')
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 465 223
         1  67 309

               Accuracy : 0.7274
                 95% CI : (0.6996, 0.754)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4549

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.5808
            Specificity : 0.8741
         Pos Pred Value : 0.8218
         Neg Pred Value : 0.6759
             Prevalence : 0.5000
         Detection Rate : 0.2904
   Detection Prevalence : 0.3534
      Balanced Accuracy : 0.7274

       'Positive' Class : 1
```

Figure 23. Confusion Matrix for DecisionTree_Tuned on Train Set

```
> confusionMatrix(test_pred, test_set$Response, positive='1')
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 322  31
         1  59  35

               Accuracy : 0.7987
                 95% CI : (0.7584, 0.8349)
    No Information Rate : 0.8523
    P-Value [Acc > NIR] : 0.999147

                  Kappa : 0.3194

 Mcnemar's Test P-Value : 0.004427

            Sensitivity : 0.5303
            Specificity : 0.8451
         Pos Pred Value : 0.3723
         Neg Pred Value : 0.9122
             Prevalence : 0.1477
         Detection Rate : 0.0783
   Detection Prevalence : 0.2103
      Balanced Accuracy : 0.6877

       'Positive' Class : 1
```

Figure 24. Confusion Matrix for DecisionTree_Tuned on Test Set

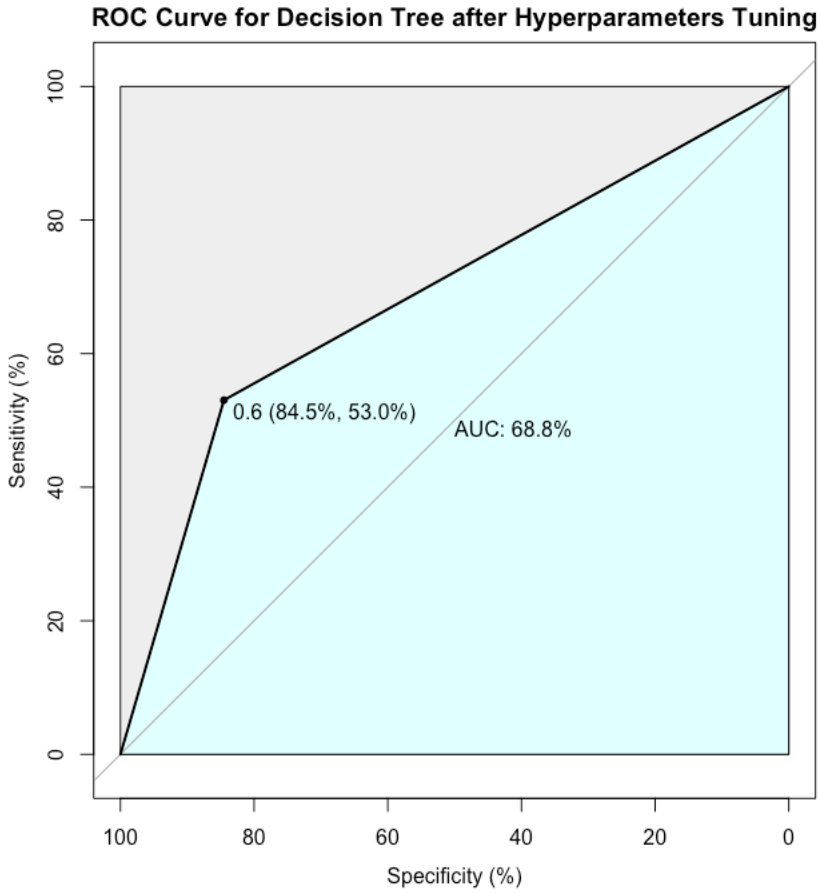

Figure 25. ROC Curve for DecisionTree_Tuned

Feature Importance in Decision Tree

In Decision Trees, Feature Importance can be calculated to denote the importance of each feature to performing classification. Through Figure 29, the top 20 most important features are shown with the three most important features being: NumCatalogPurchases, CampaignResponse, and Num_Child.

```
> FeatImp = FeatImp %>% arrange(desc(Overall))
> FeatImp
                       Overall
NumCatalogPurchases 138.843636
CampaignResponse    137.403464
Num_Child           125.379328
MntGoldProds        114.210773
TotalSpending        78.869944
Recency              73.393234
Membership           54.315320
NumStorePurchases    41.400224
NumWebVisitsMonth    35.321559
Income               29.513850
MntMeatProducts      25.913896
Rel_Single           20.396640
Ed_Postgraduate      20.184975
age                  17.896295
Rel_Coupled          17.089890
Ed_Degree            16.538650
MntWines             15.012373
MntFruits            12.757651
MntFishProducts      12.537394
NumDealsPurchases     3.377525
MntSweetProducts      0.000000
NumWebPurchases       0.000000
Complain              0.000000
Ed_Below_Degree       0.000000
```

Figure 26. Feature Importance of Decision Tree

Experiment 2: Logistic Regression

Logistic Regression Implementation: LogRegression

Logistic Regression Classification is performed through executing code shown in Code Snippet 21 where the classifier is defined in Figure 30.

Through Figure 30, it can be interpreted that the features with highest importance in the LogRegression classifier are: Recency, MntMeatProducts, NumStorePurchases, AcceptedCmp1, AcceptedCmp3, AcceptedCmp4, AcceptedCmp5, NumChild, Membership, Rel_Single, Ed_BelowDegree, and Ed_Degree.

Furthermore, the difference between Null deviance and Residual deviance can be used to obtain insight to the model fit. Null deviance represents response performance when the classifier contains only the intercept while Residual deviance represents the response performance when all variables are incorporated. A greater difference between the Null and Residual deviance represents a model of better fit. In the LogRegression classifier, Null deviance is 1475.02 on 1063 degrees of freedom and Residual deviance is 775.32 on 1037 degrees of freedom. This difference shows that the LogRression classifier built is of good fit.

```
#Logistic Regression
library(tidyverse)
library(caret)

#Build classifier
classifier = glm(Response ~.,
                 balanced_train_set,
                 family = binomial)

summary(classifier)

prob_pred_train = predict(classifier, type = 'response', balanced_train_set[ ,-15] )
y_pred_train = ifelse(prob_pred_train > 0.5, 1, 0)

pred_prob_test = classifier %>% predict(test_set, type = 'response')
y_pred_test = ifelse (pred_prob_test > 0.5 , 1, 0)
```

Code Snippet 21. LogRegression Model

```
> summary(classifier)

Call:
glm(formula = Response ~ ., family = binomial, data = balanced_train_set)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.0659  -0.5446  -0.0181   0.5191   2.3741

Coefficients: (4 not defined because of singularities)
                     Estimate Std. Error z value Pr(>|z|)
(Intercept)        -1.758e+01  1.847e+00  -9.514  < 2e-16 ***
Income             -1.609e-06  9.520e-06  -0.169 0.865798
Recency            -3.699e-02  3.718e-03  -9.950  < 2e-16 ***
MntWines           -5.633e-04  5.260e-04  -1.071 0.284223
MntFruits           5.277e-03  3.826e-03   1.379 0.167830
MntMeatProducts     3.534e-03  7.057e-04   5.009 5.48e-07 ***
MntFishProducts     5.066e-05  2.593e-03   0.020 0.984413
MntSweetProducts    5.408e-03  3.282e-03   1.648 0.099438 .
MntGoldProds       -4.319e-03  2.303e-03  -1.875 0.060727 .
NumDealsPurchases   6.222e-02  6.546e-02   0.950 0.341879
NumWebPurchases     5.759e-02  4.459e-02   1.292 0.196520
NumCatalogPurchases 2.990e-02  6.324e-02   0.473 0.636331
NumStorePurchases  -2.732e-01  4.602e-02  -5.937 2.91e-09 ***
NumWebVisitsMonth   9.781e-02  6.630e-02   1.475 0.140159
AcceptedCmp3        2.838e+00  3.719e-01   7.630 2.35e-14 ***
AcceptedCmp4        1.495e+00  3.923e-01   3.812 0.000138 ***
AcceptedCmp5        1.785e+00  4.047e-01   4.411 1.03e-05 ***
AcceptedCmp1        1.501e+00  4.177e-01   3.594 0.000326 ***
AcceptedCmp2        2.438e+02  9.491e+03   0.026 0.979507
Complain           -2.308e-01  1.495e+00  -0.154 0.877257
age                 1.492e-02  8.691e-03   1.717 0.086026 .
TotalSpending              NA         NA      NA       NA
Num_Child          -4.187e-01  1.951e-01  -2.146 0.031885 *
CampaignResponse           NA         NA      NA       NA
Membership          5.707e-03  5.913e-04   9.653  < 2e-16 ***
Rel_Single          1.289e+00  1.943e-01   6.637 3.20e-11 ***
Rel_Coupled                NA         NA      NA       NA
Ed_Below_Degree    -2.599e+00  7.219e-01  -3.600 0.000318 ***
Ed_Degree          -4.322e-01  1.961e-01  -2.204 0.027512 *
Ed_Postgraduate            NA         NA      NA       NA
clusterNum          1.712e-01  1.717e-01   0.997 0.318544
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1475.02  on 1063  degrees of freedom
Residual deviance:  775.32  on 1037  degrees of freedom
AIC: 829.32

Number of Fisher Scoring iterations: 20
```

Figure 27. Summary of LogRegression Model

LogRegression Model Validation

Similar to the model validation performed for Experiment 1: Decision Tree, confusion matrix and ROC curve shown in Figure 31, 32 and 33 to evaluate the performance of LogRegression classifier. The computed performance measurement values are populated within Table 7.

As the Train Set Accuracy, 83.6% and Test Set Accuracy, 79.6% do not have a large difference, it is concluded that the LogRegression Model is not overfitted or underfitted. Additionally, the AUC of 86.7% achieved denotes that LogRegression Model is able to achieve high True Positive rate and True negative rates.

Model	Train Set Accuracy	Test Set Accuracy	F1 Score	AUC
Logistic Regression	83.6%	79.6%	0.52	86.7%

Table 6. LogRegression Model Performance

```
> cm_train = table(balanced_train_set$Response,y_pred_train)
> cm_train
   y_pred_train
      0   1
  0 452  80
  1  95 437
> accuracy_train = sum(diag(cm_train))/sum(cm_train)
> accuracy_train
[1] 0.8355263
```

Figure 28. Confusion Matrix for LogRegression classifier on Train Set

```
> cm_test = table(test_set$Response,y_pred_test)
> cm_test
   y_pred_test
      0   1
  0 307  74
  1  17  49
> accuracy_test = sum(diag(cm_test))/sum(cm_test)
> accuracy_test
[1] 0.7964206
```

Figure 29. Confusion Matrix for LogRegression classifier on Test Set

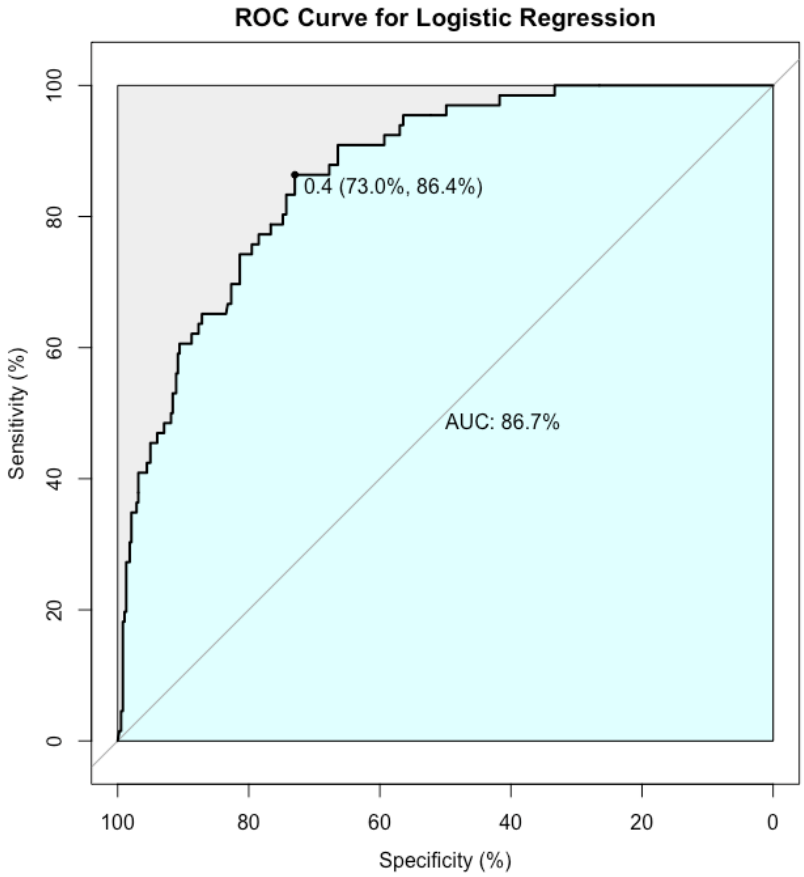

Figure 30. ROC Curve for LogRegression Model

Experiment 3: Support Vector Machine

Support Vector Machine Implementation: SVM_Basic

The Support Vector Machine classifier is built through execution of code shown in Code Snippet 22. The main parameters of the SVM model is the cost function and gamma.

As SVM is a maximum margin classifier, the cost function supports the control of overfitting through empowering modification to degree of Hard and Soft Classification. The lower value of cost function denotes a softer classification and increased margin violation while a higher value of cost function leads to a harder classification with decreased margin violation.

The gamma function defines the degree of influence a single training example holds to the classification. With high values of gamma, the performance of classifier will be highly influenced by the observations closest to the hyperplanes.

Through interpreting results of running code in Code Snippet 22 as shown in Figure 34, it is seen that the SVM_Basic classifier has a cost function of value 1 and gamma of value 0.04.

```
#Support Vector Machine
library(caTools)
library(ggplot2)
library(e1071)
set.seed(123)

svm_rbf <- svm(Response~., data = balanced_train_set)
summary(svm_rbf)
svm_rbf$gamma
svm_rbf$cost

#Performing predictions on train and test set
pred_train = predict (svm_rbf, balanced_train_set)
pred_test = predict (svm_rbf, test_set)
```

Code Snippet 22. SVM_Basic Model

```
> summary(svm_rbf)

Call:
svm(formula = Response ~ ., data = balanced_train_set)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  538

 ( 270 268 )


Number of Classes:  2

Levels:
 0 1
```

```
> svm_rbf$gamma
[1] 0.04166667
> svm_rbf$cost
[1] 1
```

Figure 31. Summary, Cost and Gamma Value of SVM_Basic Model

SVM_Basic Model Validation

The performance of SVM_Basic classifier is evaluate through confusion matrix in Figure 35 and 36, and ROC curve in Figure 37. It is observed that the model is not overfitted or underfitted and performance of classifier is satisfactory on Train Set and Test Set.

Model	Parameters Used	Parameter Tuning	Train Set Accuracy	Test Set Accuracy	F1 Score	AUC
SVM_Basic	Gamma = 0.2, Cost = 1	No	91.2%	79.6%	0.53	78.1%

Table 7. SVM_Basic Model Performance

```
> cm_train = table(Predicted = pred_train, Actual = balanced_train_set$Response)
> cm_train
         Actual
Predicted   0   1
        0 476  38
        1  56 494
> accuracy_train = sum(diag(cm_train))/sum(cm_train)*100
> accuracy_train
[1] 91.16541
```

Figure 32. Confusion Matrix of SVM_Basic on Train Set

```
> cm_test = table(Predicted = pred_test, Actual = test_set$Response)
> cm_test
         Actual
Predicted   0   1
        0 304  14
        1  77  52
> accuracy_test = sum(diag(cm_test))/sum(cm_test)*100
> accuracy_test
[1] 79.64206
```

Figure 33. Confusion Matrix of SVM_Basic on Test Set

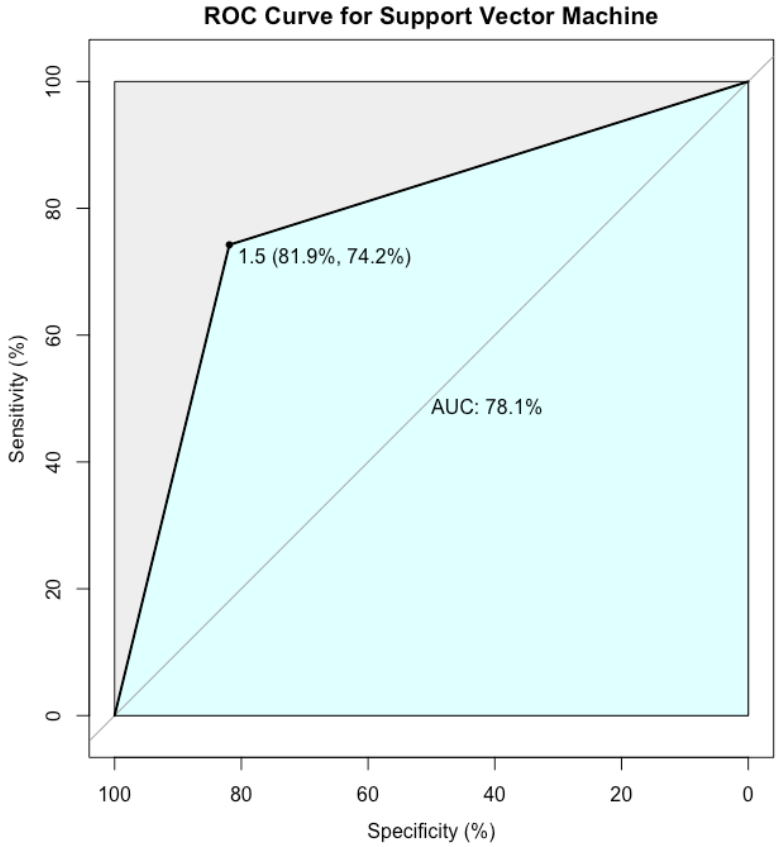

Figure 34. ROC Curve for SVM_Basic Model

Support Vector Machine with Hyperparameter Tuning: SVM_Tuned

To identify an optimum set of parameters: cost function and gamma, code shown in Code Snippet 23 is implemented to obtain a cost function value of 0.9 and gamma value of 0.2 shown in Figure 38.

SVM_Tuned model is built based on the derived parameters where the summary of tuned model is displayed in Figure 29.

```
# ~~~~~~~~~~~~~~~~~~~~  Model Tuning  ~~~~~~~~~~~~~~~~~~~~
set.seed(123)
# tune function tunes the hyperparameters of the model using grid search method
tuned_model = tune(svm, Response~., data=balanced_train_set,
                   ranges = list(gamma = seq (0, 1, 0.2), cost = seq(0.1, 1, 0.2)))
plot (tuned_model)
summary (tuned_model)

opt_model = tuned_model$best.model
summary(opt_model)
opt_model$gamma
```

Code Snippet 23. Hyperparameter Tuning of SVM

```
> summary(opt_model)

Call:
best.tune(method = svm, train.x = Response ~ ., data =
 balanced_train_set, ranges = list(gamma = seq(0, 1, 0.
2), cost = seq(0.1, 1,
    0.2)))


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.9

Number of Support Vectors:  782

 ( 359 423 )


Number of Classes:  2

Levels:
 0 1



> opt_model$gamma
[1] 0.2
```

Figure 35. Optimum parameters for SVM_Basic

```
> svm_tuned <- svm (Response~., data = balanced_train_set, gamma = 0.2, cost = 0.9)
> summary(svm_tuned)

Call:
svm(formula = Response ~ ., data = balanced_train_set, gamma = 0.2, cost = 0.9)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.9

Number of Support Vectors:  782

 ( 359 423 )


Number of Classes:  2

Levels:
 0 1
```

Figure 36. Summary of SVM_Tuned Model

SVM_Tuned Model Validation

After implementation of SVM_Tuned, the confusion matrix shown in Figure 40 and 41, and ROC curve shown in Figure 42 are used to obtain performance measurement parameters. The performance of SVM_Basic and SVM_Tuned models are summarised in Table 9.

Through comparison of SVM_Basic and SVM_Tuned classifiers, it is observed that SVM_Basic outperforms SVM_Tuned as the accuracy, F1 Score and AUC is higher in SVM_Basic classifier. Additionally, overfitting is observed in SVM_Tuned as accuracy obtained through performing classification on train set is at an almost perfect, 98.5% while accuracy obtained on test set is only 71.4%.

Hence, it is concluded that the parameter values used in the SVM_Basic model: Gamma = 0.4, Cost = 1 is optimal compared to parameter values used in SVM_Tuned model.

Model	Parameters Used	Parameter Tuning	Train Set Accuracy	Test Set Accuracy	F1 Score	AUC
SVM_Basic	Gamma = 0.4, Cost = 1	No	91.2%	79.6%	0.53	78.1%
SVM_Tuned	Gamma = 0.2, Cost = 0.9	Yes	98.5%	71.4%	0.47	72.5%

Table 8. Model Performance of SVM_Basic and SVM_Tuned

```
> cm_train
          Actual
Predicted   0   1
        0 520   8
        1  12 524
> accuracy_train = sum(diag(cm_train))/sum(cm_train)*100
> accuracy_train
[1] 98.1203
```

Figure 37. Confusion Matrix for SVM_Tuned on Train Set

```
> cm_test
          Actual
Predicted   0   1
        0 307  14
        1  74  52
> accuracy_test = sum(diag(cm_test))/sum(cm_test)*100
> accuracy_test
[1] 80.3132
```

Figure 38. Confusion Matrix for SVM_Tuned on Test Set

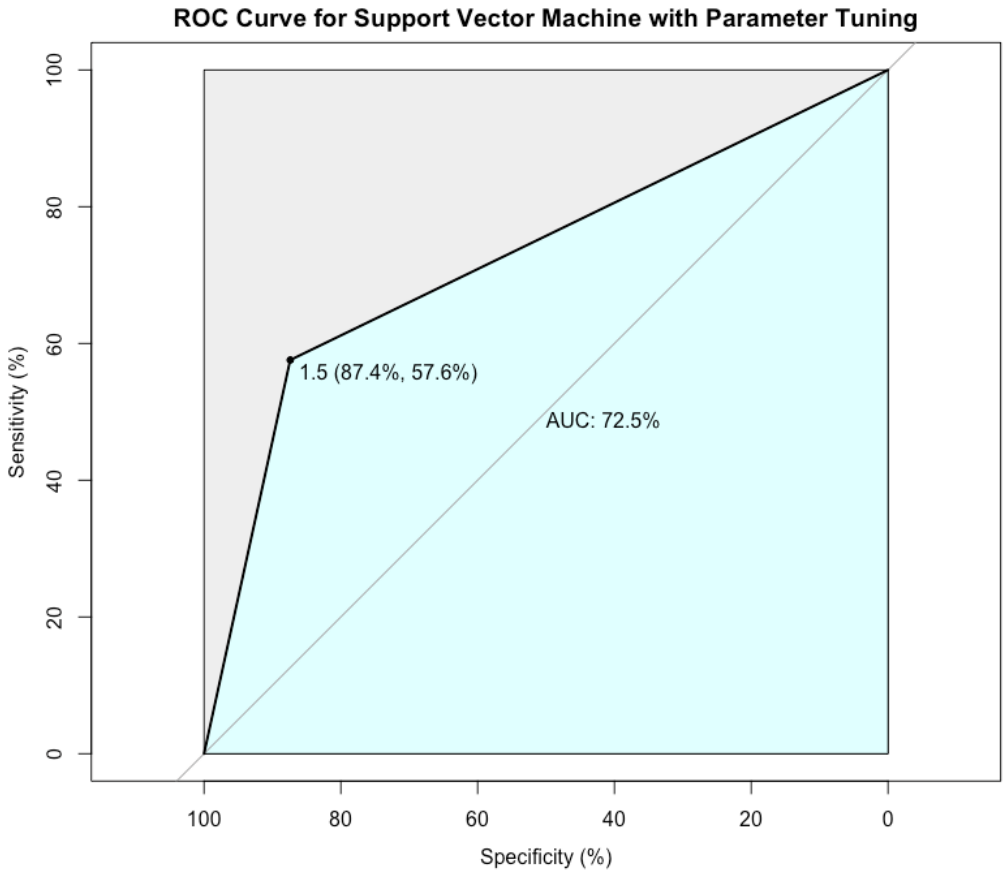

Figure 39. ROC Curve for SVM_Tuned Model

Comparison of Models

The performance of all models implemented within Experiment 1, 2, and 3 are summarised and tabulated within Table 10.

Amongst the models implemented, it is observed that the Test Set Accuracy obtained for DecisionTree_Tuned, LogRegression_Basic, and SVM_Basic have obtained the highest Test Set Accuracies. The accuracies obtained for the three models are similar. However, the difference between Train Set Accuracy and Test Set Accuracy is minimum with LogRegression_Basic. This further implies that fitness of LogRegression_Basic model is the best amongst the three models. Finally, the optimum AUC obtained through classification with LogRegression_Basic model explains that the True Positive Rate and True Negative Rate is highest through implementation of Logistic Regression.

Model	**Parameters Used**	**Parameter Tuning**	**Train Set Accuracy**	**Test Set Accuracy**	**F1 Score**	**AUC**
Experiment 1: Decision Tree						
DecisionTree_Basic	Minsplit = 1, Maxdepth = 8, Cp = 0.01	No	86%	74%	0.74	77.5%
DecisionTree_Tuned	Minsplit = 9, Maxdepth = 1, Cp = 0.01	Yes	72.4%	79.8%	0.43	68.8%
Experiment 2: Logistic Regression						
LogRegression_Basic	**-**	**No**	**83.6%**	**79.6%**	**0.52**	**86.7%**
Experiment 3: Support Vector Machine (SVM)						
SVM_Basic	Gamma = 0.4, Cost = 1	No	91.2%	79.6%	0.53	78.1%
SVM_Tuned	Gamma = 0.2, Cost = 0.9	Yes	98.5%	71.4%	0.47	72.5%

Table 9. Performance of Decision Tree, Logistic Regression, and SVM Models

Analysis and Recommendations

Critical Analysis of Model Performances

In this report, three traditional machine learning methods are implemented to predict customers' response to marketing campaigns through classification. The machine learning methods implemented are Decision Tree, Logistic Regression, and Support Vector Machine. As the dataset used is smaller in dimensions and holds lower complexity, it is expected to perform well with the traditional methods implemented. To obtain optimum model performance, hyperparameter tuning is implemented in Decision Tree and SVM.

Decision Trees and SVM are often used due to the ability to reliably handle non-linearity and split observation into smaller regions. Thus, it was initially expected for Decision Trees and SVM to achieve better performance than the Logistic Regression classifier. However, it is observed through Section 6.0 that Logistic Regression classifier was concluded to achieve optimum performance. Although the accuracies of all three classifiers were comparable, Logistic Regression obtained the higher AUC.

This can be potentially explained by two main factors: Firstly, a high correlation with the target variable, Response and a few of other variables. Additionally, the observations may be well-separated which requires a simple decision boundary for classification. These two factors would have led a higher suitability to perform classification with Logistic Regression.

Analysis of Hyperparameter Tuning

Model	Parameters Used	Parameter Tuning	Train Set Accuracy	Test Set Accuracy	F1 Score	AUC
Basic Decision Tree vs Tuned Decision Tree						
DecisionTree_Basic	Minsplit = 1, Maxdepth = 8, Cp = 0.01	No	86%	74%	0.74	77.5%
DecisionTree_Tuned	Minsplit = 9, Maxdepth = 1, Cp = 0.01	Yes	72.4%	79.8%	0.43	68.8%
Basic Support Vector Machine vs Tuned Support Vector Machine						
SVM_Basic	Gamma = 0.4, Cost = 1	No	91.2%	79.6%	0.53	78.1%
SVM_Tuned	Gamma = 0.2, Cost = 0.9	Yes	98.5%	71.4%	0.47	72.5%

Table 10. Model Performance of Basic and Tuned Models

Hyperparameter Tuning was implemented through the Grid Search technique for Decision Tree and SVM. During Decision Tree implementation, the accuracy increased as expected. However, there was a decrease is F1 Score and AUC. As SMOTE has been adopted to achieve class balance, the prioritised performance measurement parameter is accuracy. Thus, it is concluded that the performance of Decision Tree model improved after parameter tuning.

However, the opposite was observed through performing parameter tuning with SVM. It is observed that the performance of tuned SVM model got worse through adopted the cost function and gamma value derived through the Grid Search technique. Additionally, the SVM_Tuned model has been identified to be overfitted as Train Set Accuracy is 98.5% while Test Set Accuracy is only 71.4%.

The impairment of model performance after hyperparameter tuning is due to the cost function and gamma value intialised in implementing the Grid Search technique. However, due to a time constraints, further investigation and optimisation was not performed.

7.3 Recommendation and Future Works

The recommendations and improvement areas derived through this report to predict response to marketing campaigns are:

Dataset

- Collection of more data to increase reliability of results

Data Preparation

- Feature engineering to obtain total number of past responses to marketing campaign
- Feature engineering to obtain length of membership of each customer
- Perform oversampling with SMOTE to tackle class imbalance

Model Implementation and Validation

- Logistic Regression to be implemented for classification
- Optimisation of SVM model through improved initialisation of parameter values