IST 707

Final Project Report

Jaclyn Karboski and Ash Wan

Table of Contents:

Dataset Description

Title: Customer personality analysis: analysis of company's ideal customers

Source information:
Customer personality analysis: Analysis of company's ideal customers. (2021). [Data file]. Retrieved from https://www.kaggle.com/imakash3011/customer-personality-analysis/metadata

Description:
The dataset is a customer personality analysis that tracks customers' demographics, their purchase history, reactions to promotions, and place of purchase. The set consists of 29 variables and 2240 rows of data, for 64,960 objects.

Attributes:
Personal:
ID: Unique customer ID (num)
Year_Birth: Birth year (num)
Education: Highest level of education (chr)
Marital_Status: Marital Status (chr)
Income: Household income (num)
Kidhome: Number of children < 13 in the household (num)
Teenhome: Number of teens > 12 in the household (num)
Dt_Customer: Date of enrollment in system (chr)
Recency: Days since last purchase (num)
Complain: Made a complaint (num)

Product spending in last 2 years:
MntWines: Amount spent on wine (num)
MntFruits: Amount spent on fruits (num)
MntMeatProducts: Amount spent on meat (num)
MntFishProducts: Amount spent on fish (num)
MntSweetProducts: Amount spent on sweets (num)
MntGoldProds: Amount spent on gold (num)

Place of purchase:
NumWebPurchases: Number of web purchases (num)
NumCatalogPurchases: Number of catalog purchases(num)
NumStorePurchases: Number of in-store purchases (num)
NumWebVisitsMonth: Number of web visits per month (num)

Promotion reactions:
NumDealsPurchases: Number of purchases with a discount (num)

AcceptedCmp1: Accepted offer in 1st campaign (num)
AcceptedCmp2: Accepted offer in 2nd campaign (num)
AcceptedCmp3: Accepted offer in 3rd campaign (num)
AcceptedCmp4: Accepted offer in 4th campaign (num)
AcceptedCmp5: Accepted offer in 5th campaign (num)

Data Acquisition

For this project, we wanted to work with a dataset that a met a number of criteria. First, the dataset had to be accessible and ethically sourced. We were able to find datasets online through Kaggle.com and find a dataset that was available. The marketing data seemed to fit the purpose of our project and we wanted to work with a dataset that our group had familiarity with so we can understand the elements within and how they relate to one another.

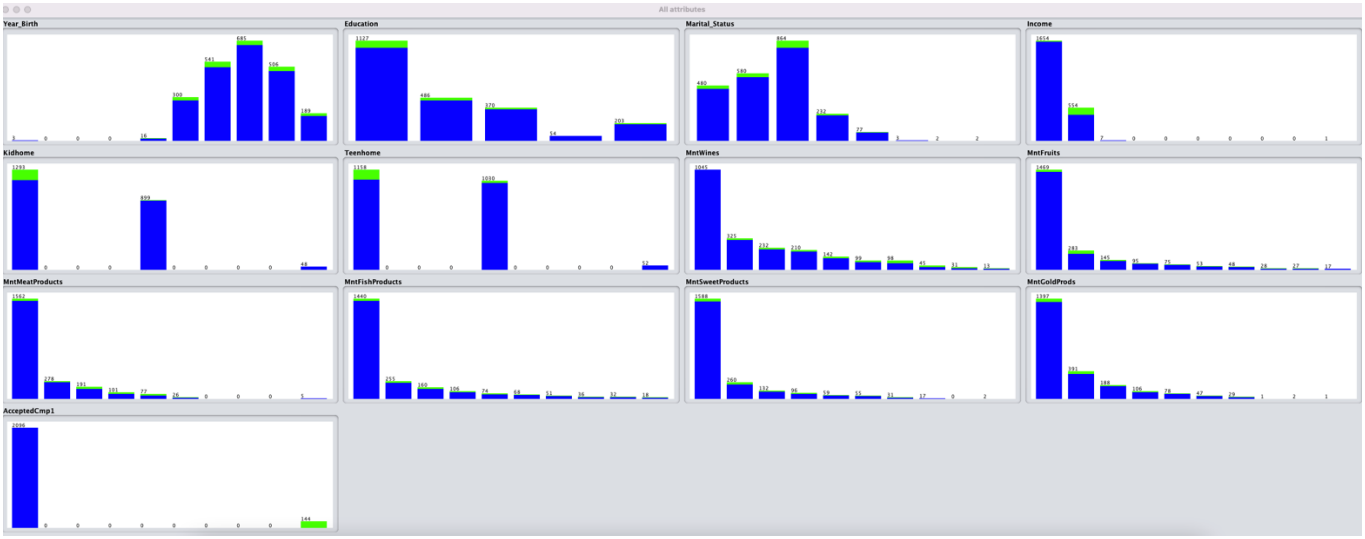
Next, we needed the dataset to be a viable option with a limited number of at least 10,000 values. In deciding to utilize the “marketing campaign” dataset we had a good mixture of values in the form of characters, dates, and integers that would allow us a variety of analysis and to have a more complete dataset to work with.

Association Rule Mining: Apriori Algorithm

Our initial business question is to understand if there are variables that will predict if a customer is likely to accept a campaign offer. To assess this, we chose the Apriori algorithm, implemented within Weka, for analysis.

Preprocessing

As a first step, data was loaded into Weka for preprocessing. Sixteen attributes were removed from the data set as they are considered not relevant for the analysis. Secondly, all numeric attributes were converted to nominal values. Refer below for the visualization of all data by attribute.



Algorithm Implementation/Model Build

An initial run of the Apriori algorithm was executed against the population at a set minimum confidence of 0.8, delta of 0.05 and an upper bound of support 1.0. The following 15 rules were returned, which ranged in confidence from 0.91 to 0.99:

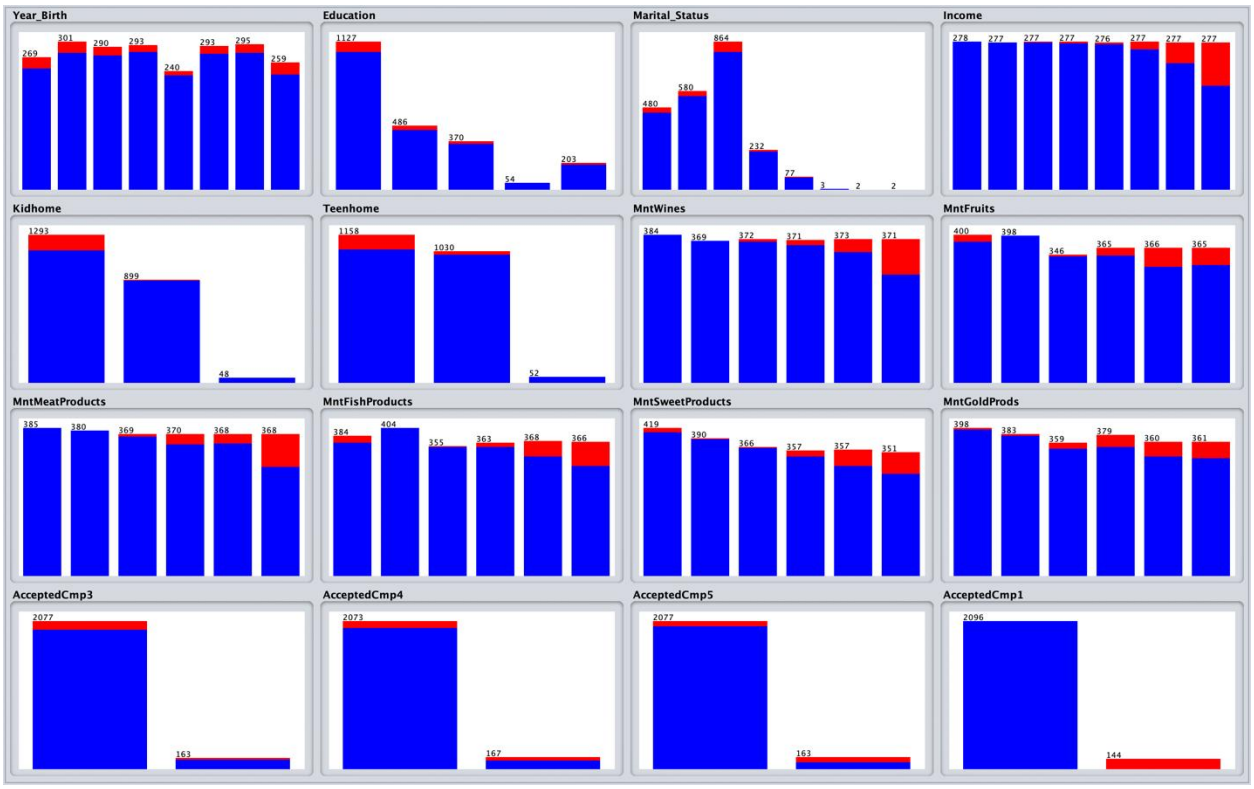
```
=== Run information ===

Scheme:       weka.associations.Apriori -N 15 -T 0 -C 0.8 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     Marketing-weka.filters.unsupervised.attribute.Remove-R1,8-9,16-21,26-29-weka.filters.unsupervised.attribute.Discretize-B10-M-1.0-Rfirst-last-precision6-weka.filte
Instances:    2240
Attributes:   13
              Year_Birth
              Education
              Marital_Status
              Income
              Kidhome
              Teenhome
              MntWines
              MntFruits
              MntMeatProducts
              MntFishProducts
              MntSweetProducts
              MntGoldProds
              AcceptedCmp1
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.6 (1344 instances)
Minimum metric <confidence>: 0.8
Number of cycles performed: 8

Generated sets of large itemsets:

Size of set of large itemsets L(1): 7

Size of set of large itemsets L(2): 12

Size of set of large itemsets L(3): 3

Best rules found:

 1. Income='(-inf-68223.6]' MntMeatProducts='(-inf-172.5]' 1448 ==> AcceptedCmp1='(-inf-0.1]' 1434    <conf:(0.99)> lift:(1.06) lev:(0.04) [79] conv:(6.21)
 2. Income='(-inf-68223.6]' MntSweetProducts='(-inf-26.3]' 1415 ==> AcceptedCmp1='(-inf-0.1]' 1401    <conf:(0.99)> lift:(1.06) lev:(0.03) [76] conv:(6.06)
 3. MntMeatProducts='(-inf-172.5]' MntSweetProducts='(-inf-26.3]' 1403 ==> AcceptedCmp1='(-inf-0.1]' 1387    <conf:(0.99)> lift:(1.06) lev:(0.03) [74] conv:(5.31)
 4. Income='(-inf-68223.6]' 1654 ==> AcceptedCmp1='(-inf-0.1]' 1633    <conf:(0.99)> lift:(1.06) lev:(0.04) [85] conv:(4.83)
 5. MntFishProducts='(-inf-25.9]' 1440 ==> AcceptedCmp1='(-inf-0.1]' 1412    <conf:(0.98)> lift:(1.05) lev:(0.03) [64] conv:(3.19)
 6. MntMeatProducts='(-inf-172.5]' 1562 ==> AcceptedCmp1='(-inf-0.1]' 1526    <conf:(0.98)> lift:(1.04) lev:(0.03) [64] conv:(2.71)
 7. MntFruits='(-inf-19.9]' 1469 ==> AcceptedCmp1='(-inf-0.1]' 1433    <conf:(0.98)> lift:(1.04) lev:(0.03) [58] conv:(2.55)
 8. MntSweetProducts='(-inf-26.3]' 1588 ==> AcceptedCmp1='(-inf-0.1]' 1546    <conf:(0.97)> lift:(1.04) lev:(0.03) [60] conv:(2.37)
 9. MntGoldProds='(-inf-36.2]' 1397 ==> AcceptedCmp1='(-inf-0.1]' 1347    <conf:(0.96)> lift:(1.03) lev:(0.02) [39] conv:(1.76)
10. MntMeatProducts='(-inf-172.5]' AcceptedCmp1='(-inf-0.1]' 1526 ==> Income='(-inf-68223.6]' 1434    <conf:(0.94)> lift:(1.27) lev:(0.14) [307] conv:(4.29)
11. MntFishProducts='(-inf-25.9]' 1440 ==> MntSweetProducts='(-inf-26.3]' 1347    <conf:(0.94)> lift:(1.32) lev:(0.15) [326] conv:(4.46)
12. MntMeatProducts='(-inf-172.5]' 1562 ==> Income='(-inf-68223.6]' 1448    <conf:(0.93)> lift:(1.26) lev:(0.13) [294] conv:(3.55)
13. MntMeatProducts='(-inf-172.5]' 1562 ==> Income='(-inf-68223.6]' AcceptedCmp1='(-inf-0.1]' 1434    <conf:(0.92)> lift:(1.26) lev:(0.13) [295] conv:(3.28)
14. MntFruits='(-inf-19.9]' 1469 ==> MntSweetProducts='(-inf-26.3]' 1345    <conf:(0.92)> lift:(1.29) lev:(0.14) [303] conv:(3.42)
15. MntFruits='(-inf-19.9]' 1469 ==> Income='(-inf-68223.6]' 1344    <conf:(0.91)> lift:(1.24) lev:(0.12) [259] conv:(3.05)
```

Model Tuning

To understand which variables best indicate if a customer will accept a campaign, the right-hand side/AcceptedCmp1 had to be constrained. Documented below are the top 15 rules:

```
Apriori
=======

Minimum support: 0.55 (1232 instances)
Minimum metric <confidence>: 0.8
Number of cycles performed: 9

Generated sets of large itemsets:

Size of set of large itemsets L(1): 6

Size of set of large itemsets L(2): 10

Size of set of large itemsets L(3): 8

Size of set of large itemsets L(4): 2

Best rules found:

 1. Income='(-inf-68223.6]' MntFishProducts='(-inf-25.9]' MntSweetProducts='(-inf-26.3]' 1279 ==> AcceptedCmp1='(-inf-0.1]' 1268    conf:(0.99)
 2. Income='(-inf-68223.6]' MntMeatProducts='(-inf-172.5]' MntFishProducts='(-inf-25.9]' MntSweetProducts='(-inf-26.3]' 1247 ==> AcceptedCmp1='(-inf-0.1]' 1236    conf:(0.99)
 3. MntMeatProducts='(-inf-172.5]' MntFishProducts='(-inf-25.9]' MntSweetProducts='(-inf-26.3]' 1281 ==> AcceptedCmp1='(-inf-0.1]' 1269    conf:(0.99)
 4. Income='(-inf-68223.6]' MntMeatProducts='(-inf-172.5]' MntFishProducts='(-inf-25.9]' 1274 ==> AcceptedCmp1='(-inf-0.1]' 1262    conf:(0.99)
 5. Income='(-inf-68223.6]' MntMeatProducts='(-inf-172.5]' 1448 ==> AcceptedCmp1='(-inf-0.1]' 1434    conf:(0.99)
 6. Income='(-inf-68223.6]' MntMeatProducts='(-inf-172.5]' MntSweetProducts='(-inf-26.3]' 1343 ==> AcceptedCmp1='(-inf-0.1]' 1330    conf:(0.99)
 7. Income='(-inf-68223.6]' MntFishProducts='(-inf-25.9]' 1324 ==> AcceptedCmp1='(-inf-0.1]' 1311    conf:(0.99)
 8. MntMeatProducts='(-inf-172.5]' MntFishProducts='(-inf-25.9]' 1316 ==> AcceptedCmp1='(-inf-0.1]' 1303    conf:(0.99)
 9. Income='(-inf-68223.6]' MntSweetProducts='(-inf-26.3]' 1415 ==> AcceptedCmp1='(-inf-0.1]' 1401    conf:(0.99)
10. Income='(-inf-68223.6]' MntFruits='(-inf-19.9]' MntMeatProducts='(-inf-172.5]' 1289 ==> AcceptedCmp1='(-inf-0.1]' 1276    conf:(0.99)
11. MntFruits='(-inf-19.9]' MntFishProducts='(-inf-25.9]' MntSweetProducts='(-inf-26.3]' 1261 ==> AcceptedCmp1='(-inf-0.1]' 1248    conf:(0.99)
12. Income='(-inf-68223.6]' MntFruits='(-inf-19.9]' MntMeatProducts='(-inf-172.5]' MntSweetProducts='(-inf-26.3]' 1253 ==> AcceptedCmp1='(-inf-0.1]' 1240    conf:(0.99)
13. Income='(-inf-68223.6]' MntFruits='(-inf-19.9]' MntSweetProducts='(-inf-26.3]' 1278 ==> AcceptedCmp1='(-inf-0.1]' 1264    conf:(0.99)
14. MntFruits='(-inf-19.9]' MntMeatProducts='(-inf-172.5]' 1331 ==> AcceptedCmp1='(-inf-0.1]' 1316    conf:(0.99)
15. MntMeatProducts='(-inf-172.5]' MntSweetProducts='(-inf-26.3]' 1403 ==> AcceptedCmp1='(-inf-0.1]' 1387    conf:(0.99)
```

Upon further inspection of the rules, all confidence levels were at 99% and driven by variables which had nominal attributes that contained the majority of the population. To fix this, we went back and re-discretized the variables. The majority were discretized from equal width to equal frequency. Refer below for the updated visualization of the data:



After adjusting data within the preprocessing step, we reran the Apriori algorithm and obtained far less (4 in total) association rules with confidence above 0.9.

```
Apriori
=======

Minimum support: 0.4 (896 instances)
Minimum metric <confidence>: 0.8
Number of cycles performed: 12

Generated sets of large itemsets:

Size of set of large itemsets L(1): 6

Size of set of large itemsets L(2): 4

Best rules found:

 1. Teenhome='(0.666667-1.333333]' 1030 ==> AcceptedCmp1='(-inf-0.5]' 1003    <conf:(0.97)> lift:(1.04) lev:(0.02) [39] conv:(2.36)
 2. Education=Graduation 1127 ==> AcceptedCmp1='(-inf-0.5]' 1045    <conf:(0.93)> lift:(0.99) lev:(-0) [-9] conv:(0.87)
 3. Teenhome='(-inf-0.666667]' 1158 ==> AcceptedCmp1='(-inf-0.5]' 1043    <conf:(0.9)> lift:(0.96) lev:(-0.02) [-40] conv:(0.64)
 4. Kidhome='(-inf-0.666667]' 1293 ==> AcceptedCmp1='(-inf-0.5]' 1160    <conf:(0.9)> lift:(0.96) lev:(-0.02) [-49] conv:(0.62)
```

Next, we reset the right hand side constraint to equal AcceptedCmp1 which resulted in the following 15 rules:

```
Apriori
=======

Minimum support: 0.2 (448 instances)
Minimum metric <confidence>: 0.1
Number of cycles performed: 16

Generated sets of large itemsets:

Size of set of large itemsets L(1): 9

Size of set of large itemsets L(2): 6

Best rules found:

 1. Kidhome='(0.666667-1.333333]' Teenhome='(-inf-0.666667]' 503 ==> AcceptedCmp1='(-inf-0.5]' 498    conf:(0.99)
 2. Kidhome='(0.666667-1.333333]' 899 ==> AcceptedCmp1='(-inf-0.5]' 890    conf:(0.99)
 3. Teenhome='(0.666667-1.333333]' 1030 ==> AcceptedCmp1='(-inf-0.5]' 1003    conf:(0.97)
 4. Education=Graduation Teenhome='(0.666667-1.333333]' 511 ==> AcceptedCmp1='(-inf-0.5]' 497    conf:(0.97)
 5. Kidhome='(-inf-0.666667]' Teenhome='(0.666667-1.333333]' 625 ==> AcceptedCmp1='(-inf-0.5]' 603    conf:(0.96)
 6. Marital_Status=Together 580 ==> AcceptedCmp1='(-inf-0.5]' 548    conf:(0.94)
 7. Education=PhD 486 ==> AcceptedCmp1='(-inf-0.5]' 456    conf:(0.94)
 8. Marital_Status=Single 480 ==> AcceptedCmp1='(-inf-0.5]' 449    conf:(0.94)
 9. Education=Graduation 1127 ==> AcceptedCmp1='(-inf-0.5]' 1045    conf:(0.93)
10. Marital_Status=Married 864 ==> AcceptedCmp1='(-inf-0.5]' 801    conf:(0.93)
11. Teenhome='(-inf-0.666667]' 1158 ==> AcceptedCmp1='(-inf-0.5]' 1043    conf:(0.9)
12. Kidhome='(-inf-0.666667]' 1293 ==> AcceptedCmp1='(-inf-0.5]' 1160    conf:(0.9)
13. Education=Graduation Teenhome='(-inf-0.666667]' 593 ==> AcceptedCmp1='(-inf-0.5]' 527    conf:(0.89)
14. Education=Graduation Kidhome='(-inf-0.666667]' 650 ==> AcceptedCmp1='(-inf-0.5]' 575    conf:(0.88)
15. Kidhome='(-inf-0.666667]' Teenhome='(-inf-0.666667]' 638 ==> AcceptedCmp1='(-inf-0.5]' 528    conf:(0.83)
```

Results Interpretation

The following rules returned greater than 95% confidence:

#	Rule Criteria	English Translation
1	Kidhome='(0.666667-1.333333]' Teenhome='(-inf-0.666667]'==> AcceptedCmp1='(-inf-0.5]' 498 conf:(0.99)	No kids **and** teenagers at home will not accept the campaign (with 99% confidence)
2	Kidhome='(0.666667-1.333333]' 899==> AcceptedCmp1='(-inf-0.5]' 890 conf:(0.99) 1030	No kids at home will not accept the campaign (with 97% confidence)
3	Teenhome='(0.666667-1.333333]' ==> AcceptedCmp1='(-inf-0.5]' 1003 conf:(0.97)	No teenagers at home will not accept the campaign (with 97% confidence)
4	Education=Graduation Teenhome='(0.666667-1.333333]' 511 ==> AcceptedCmp1='(-inf-0.5]' 497 conf:(0.97)	Graduate level education **and** no teens at home will not accept the campaign (with 97% confidence)
5	Kidhome='(-inf-0.666667]' Teenhome='(0.666667-1.333333]' 625==> AcceptedCmp1='(-inf-0.5]' 603 conf:(0.96)	No kids at home **and** one teenager at home will not accept the campaign (with 96% confidence)

We started the association rule process thinking that we would discover rules that would indicate who is accepting campaigns, instead, all the best rules indicate who is **not** accepting campaigns. This deviation from the expected, lead to an interesting conclusion; it is

with high confidence that there is a whole group of shoppers – made up by individuals who do not have children in the house, of any age – who are not accepting campaigns. This has identified a marketing opportunity and a chance to customize marketing to this group to drive the adoption/onboarding of these customers to the campaign. Next step opportunities would include Associate Rule Mining (LHS = KidsHome/Teenshome) understanding what this target audience tends to purchase as a way to further refine the messaging.

K-Means

Preprocessing

As part of our K-Means analysis, we removed unnecessary columns from the data set. The following variables were considered non-essential in our prediction model.

Variables Removed:
ID
Dt_Customer
Recency
Complain
Z_CostContact
Z_Revenue
Response
Education
Marital_Status
NumDealsPurchases
AcceptCmp2
AcceptCmp3
AcceptCmp4
AcceptCmp5

We search for any NA's/missing values. The only variable that contained missing values was Income. We applied the interpolation function to the variable so that the missing values would be replaced by the average of the income before and after the missing values.

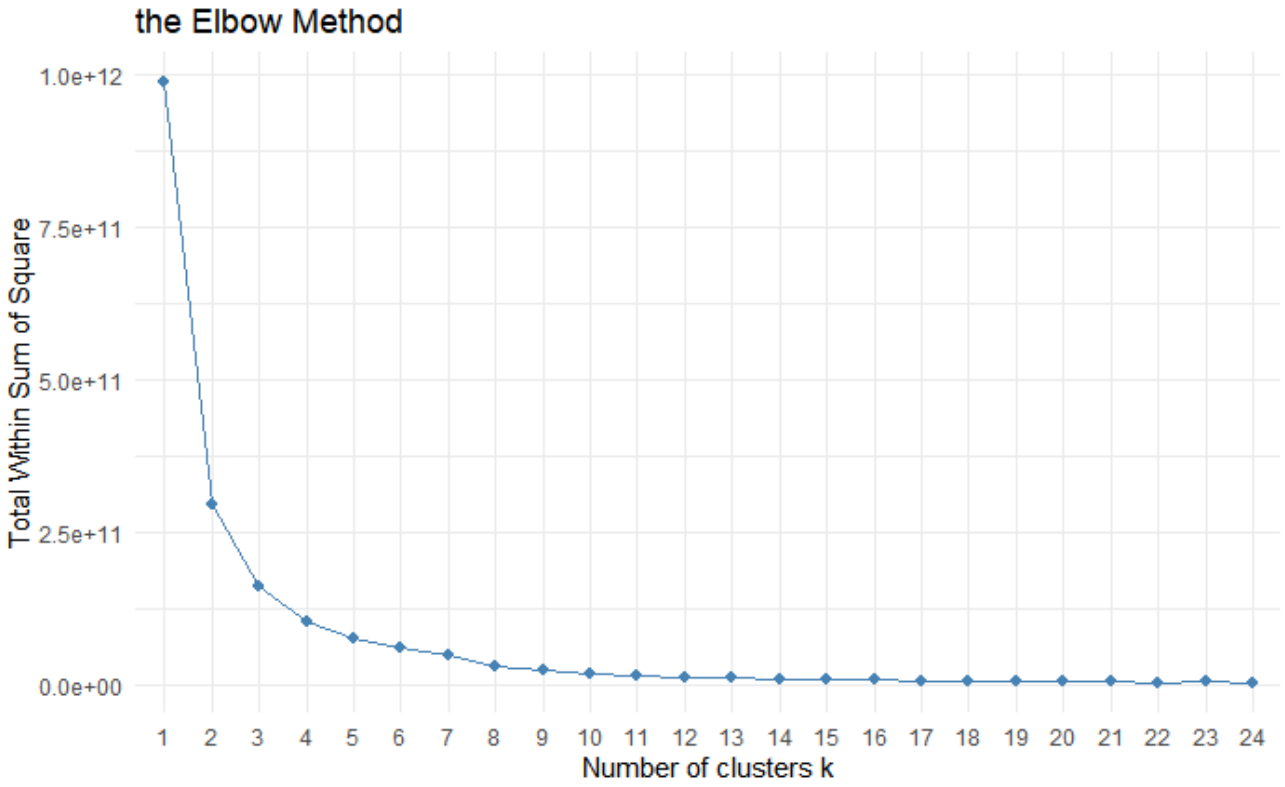
Missing Values/NA
Income

Lastly, we removed any outliers that was identified and removed to get an accurate result for the K-Means clusters.

Outliers:
Year_Birth
Income
MntMeatProducts
MntSweetProducts
MntGoldProds

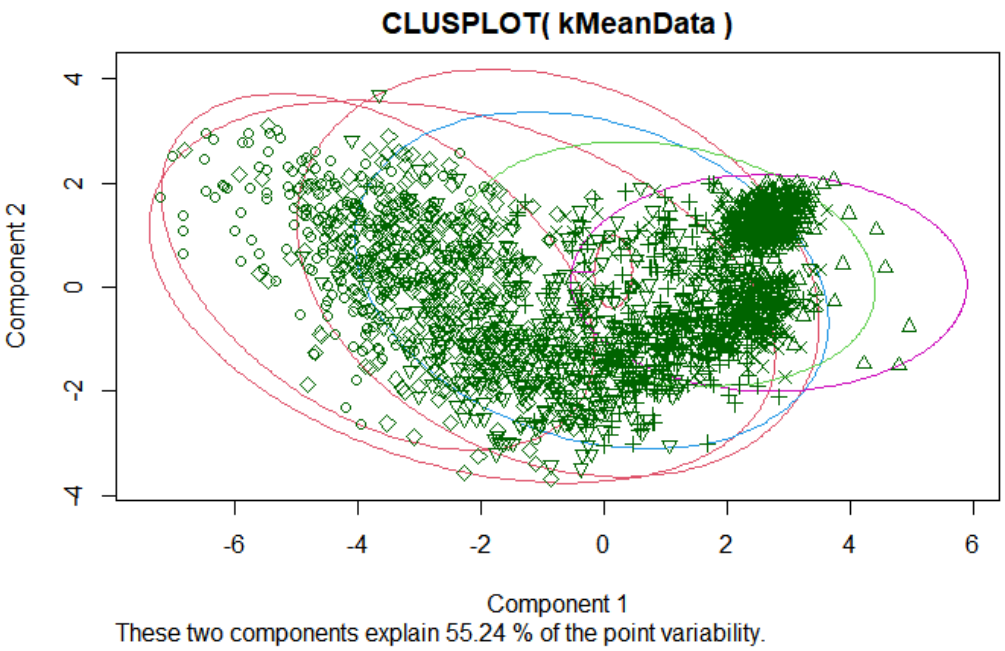
Algorithm Implementation/Model Build

To run the K-Means algorithm, first we needed to identify which value of k (number of clusters) would produce the best result. To do this, we performed the Elbow method. The Elbow method looks at the percentage of variance explained as a function of the number of clusters. We chose 7 to be our number of clusters because the numbers of 7 doesn't give a much better result.



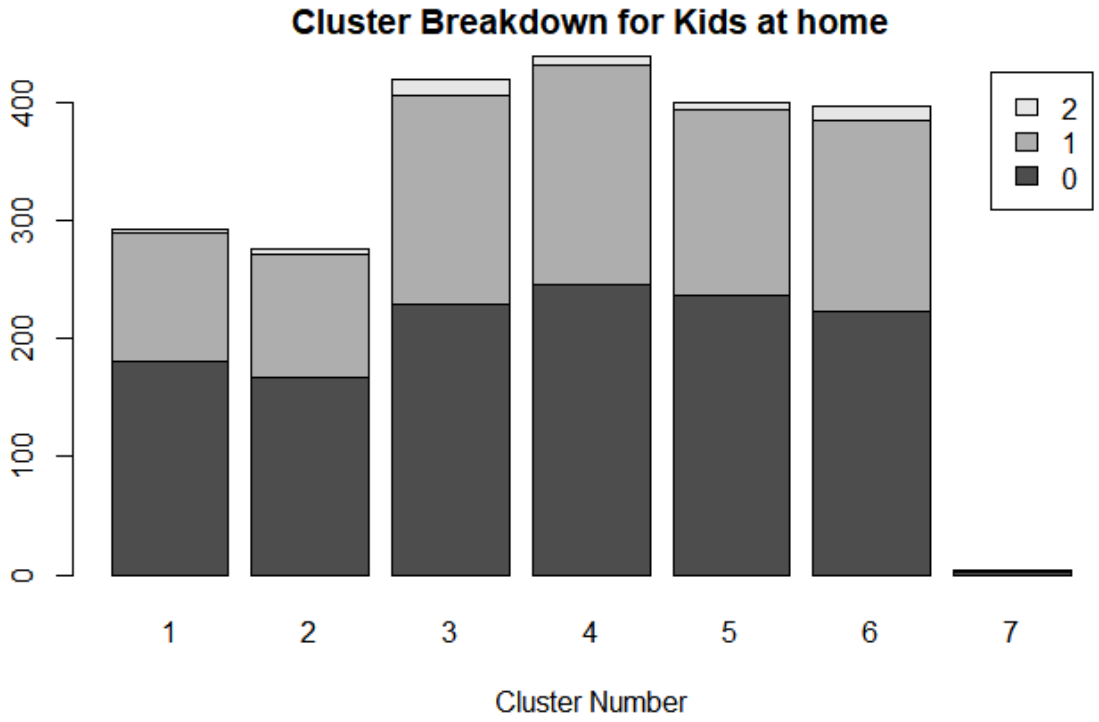
Documented below is the result of our K-Means and its plot.

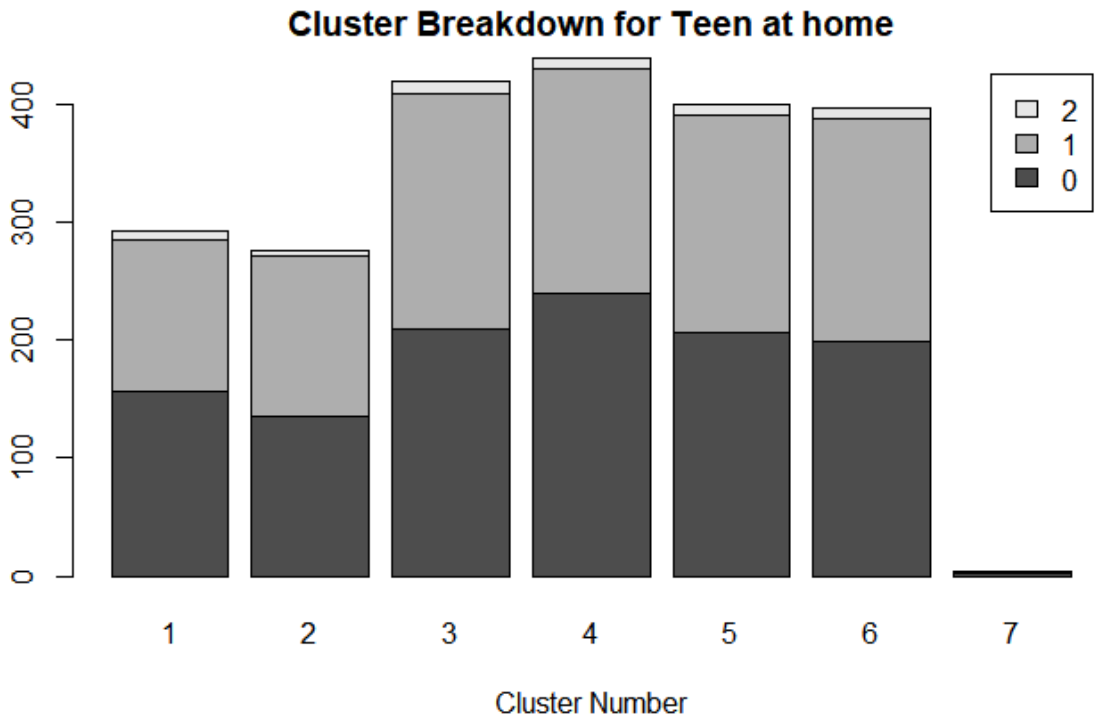
```
K-means clustering with 7 clusters of sizes 293, 276, 419, 439, 400, 396, 4

Cluster means:
  Year_Birth    Income    Kidhome  Teenhome  MntWines MntFruits MntMeatProducts MntFishProducts MntSweetProducts MntGoldProds NumWebPurchases
1   1967.819  83580.29 0.05119454 0.1945392 686.38225 66.361775       487.64164        93.877133        71.160410     72.96246        5.266212
2   1975.659  19340.62 0.75724638 0.1557971  11.68841  5.833333        14.43841         7.721014         6.184783     16.12319        1.815217
3   1967.181  45631.59 0.56085919 0.7541766 162.04296  9.634845        57.14081        15.214797         9.801909     32.00477        3.778043
4   1971.649  33355.16 0.84965831 0.4009112  40.45103  5.776765        27.18907         9.840547         5.781321     16.82232        2.202733
5   1967.320  70134.12 0.11750000 0.4800000 575.49750 50.587500       320.87250        75.032500        51.552500     69.12250        5.572500
6   1965.268  58199.70 0.27777778 0.8686869 402.76768 27.411616       136.22222        34.181818        25.717172     58.22222        5.590909
7   1967.750 157744.50 0.50000000 0.2500000  31.75000  1.000000         7.00000         1.500000         0.750000      3.00000        0.250000
  NumCatalogPurchases NumStorePurchases NumWebVisitsMonth AcceptedCmp1
1           6.1604096          8.481229          2.580205  0.296928328
2           0.4202899          2.768116          7.278986  0.000000000
3           1.4057279          4.625298          6.198091  0.011933174
4           0.5284738          3.261959          6.715262  0.002277904
5           4.7575000          8.490000          3.725000  0.097500000
6           3.0378788          7.421717          5.267677  0.030303030
7           0.0000000          0.500000          0.500000  0.000000000

Within cluster sum of squares by cluster:
[1] 8734852879 8434464598 5433280341 6062263451 5276966184 4996642045   36920717
 (between_SS / total_SS =  96.1 %)
```



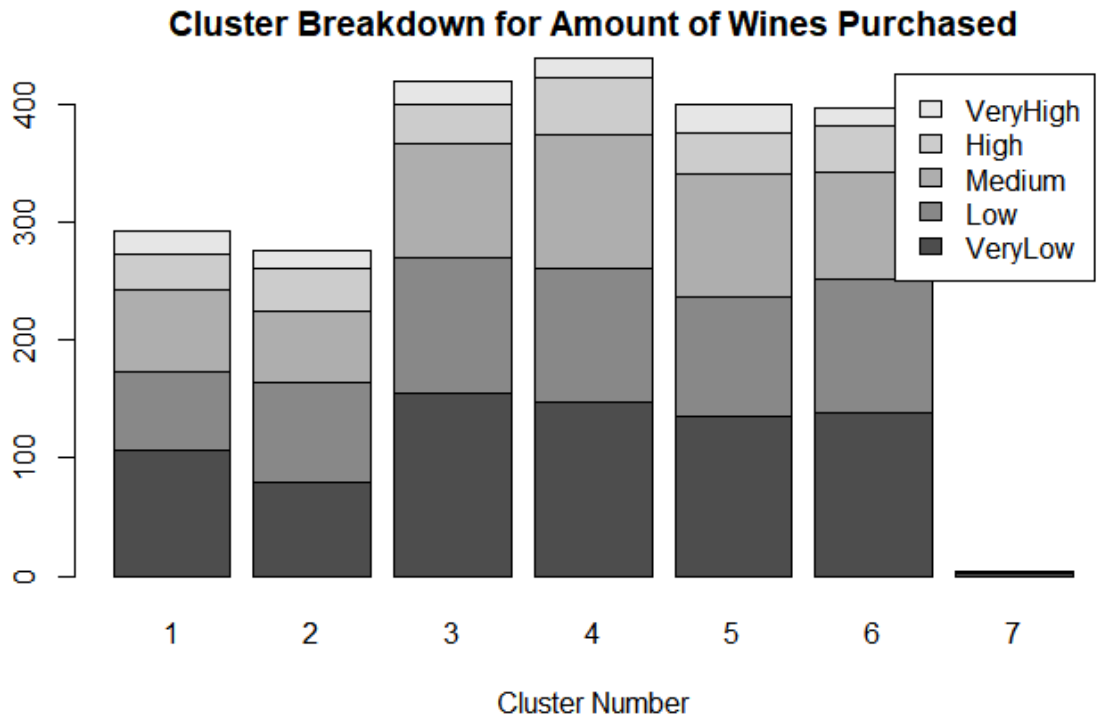
Interpreting Results

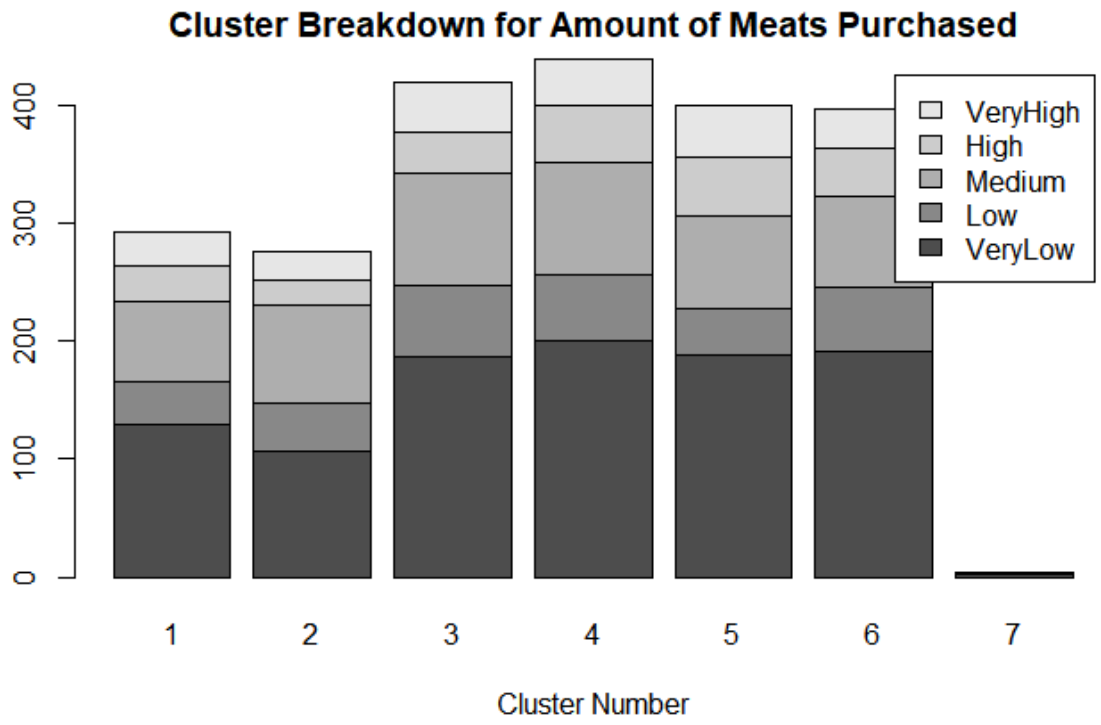
Documented below are the bar graphs showing the cluster breakdown for Kidshome and Teenhome variable which was pointed out as important from the Apriori. Below are also graphs for the top 5 variables identified during our prediction section. For additional details, refer to the RMD document submitted with this report.

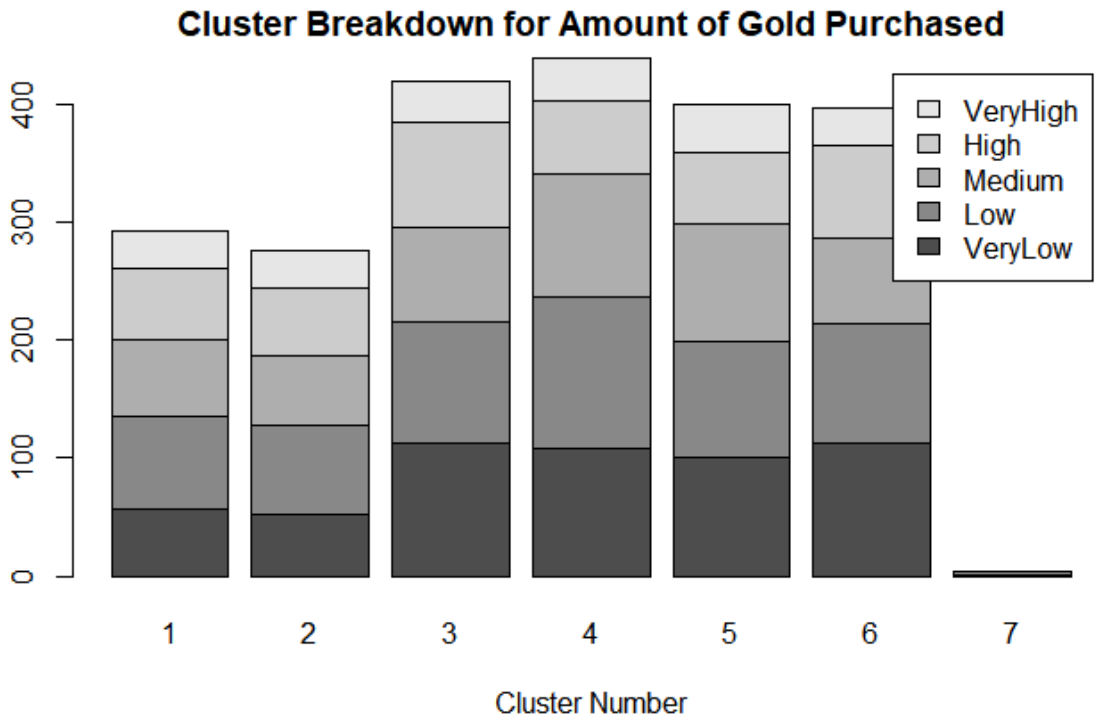


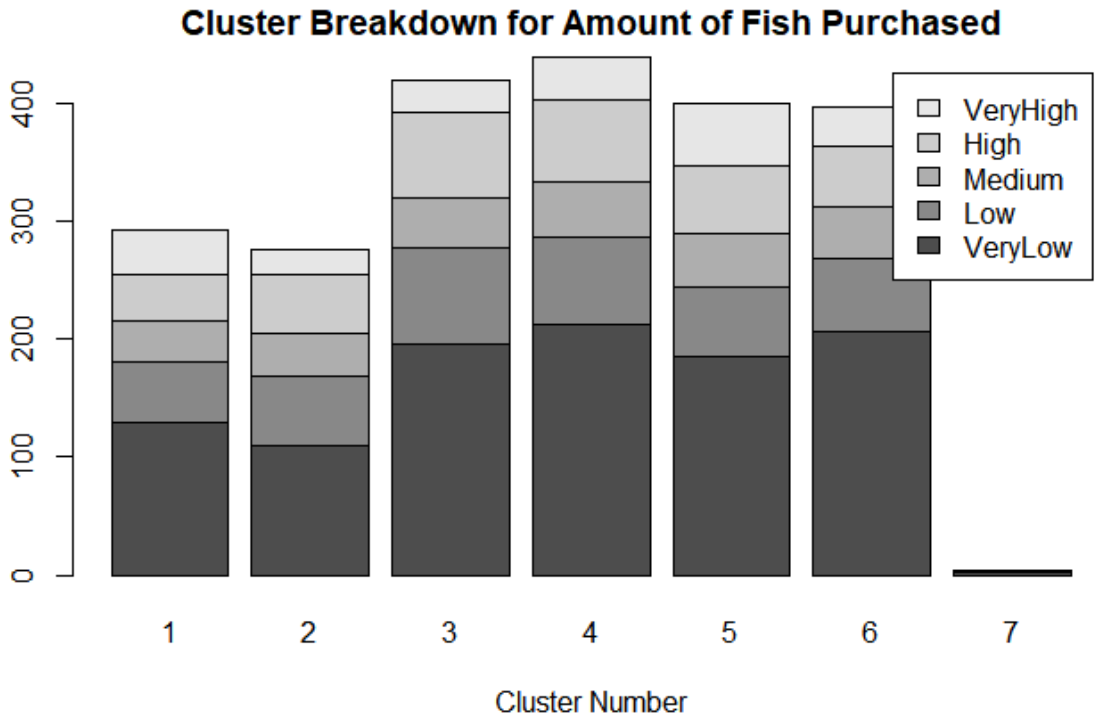

Cluster Breakdown for Teen at home
0
100
200
300
400
2
1
0
1
2
3
4
5
6
7
Cluster Number


Cluster Breakdown for Income
0
100
200
300
400
VeryHigh
High
Medium
Low
VeryLow
1
2
3
4
5
6
7
Cluster Number


Cluster Breakdown for Amount of Wines Purchased
0
100
200
300
400
1
2
3
4
5
6
7
Cluster Number
VeryHigh
High
Medium
Low
VeryLow


Cluster Breakdown for Amount of Meats Purchased
0
100
200
300
400
1
2
3
4
5
6
7
Cluster Number
VeryHigh
High
Medium
Low
VeryLow


Cluster Breakdown for Amount of Gold Purchased
0
100
200
300
400
1
2
3
4
5
6
7
Cluster Number
VeryHigh
High
Medium
Low
VeryLow


Cluster Breakdown for Amount of Fish Purchased
0
100
200
300
400
1
2
3
4
5
6
7
Cluster Number
VeryHigh
High
Medium
Low
VeryLow

Model Prediction – Decision Tree, Naïve Bayes, Random Forest, SVM, KNN

Preprocessing

As part of our model prediction analysis, we removed unnecessary columns from the data set. The following variables were considered non-essential in our prediction model.

Variables Removed:
ID
Dt_Customer
Recency
Z_CostContact
Z_Revenue
Response
NumDealsPurchases
AcceptCmp2
AcceptCmp3
AcceptCmp4
AcceptCmp5

As part of the model criteria, the following attributes we required to be discretized from numeric to factors.

Variables for Discretization
Education
Marital_Status
AcceptedCmp1

Lastly, we search for NA's/missing values. The only variable that contained missing values was Income. To solve for this, we sorted the data set by education as we believe that education would be the biggest indicator of income. Next, we applied the interpolation function to the Income variable so that missing incomes would be replaced by the average of the income before and after the missing value. Refer below:

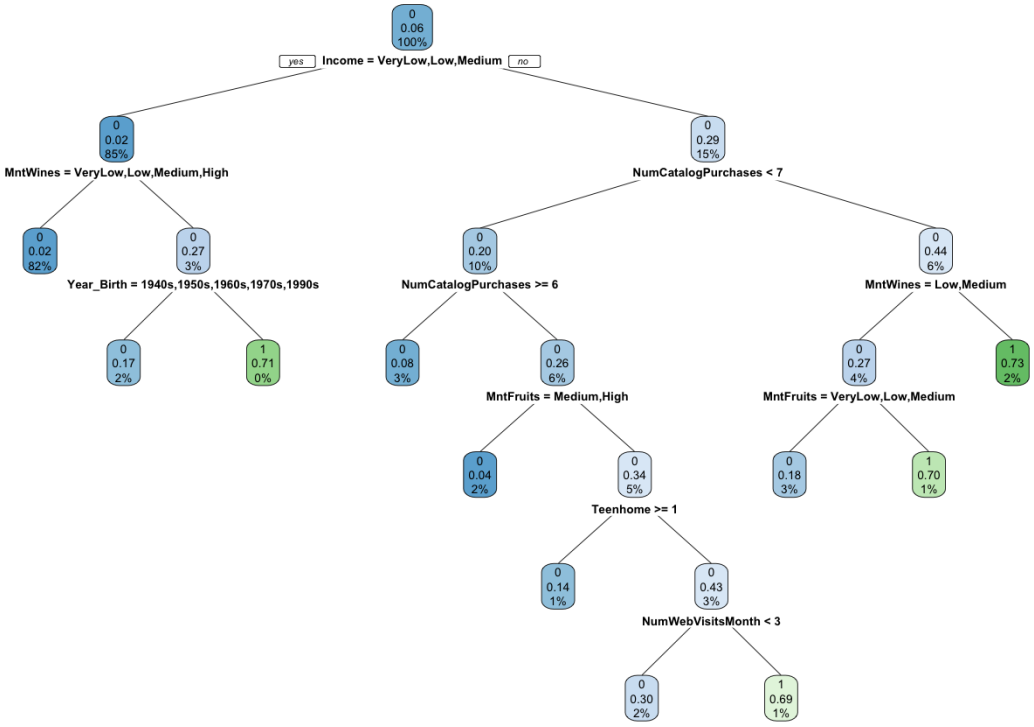
Missing Values/NA
Income

Now that the data has been fully preprocessed, we split the data set to form a training and testing set for purposed of model building. We used the standard rule of thumb for splitting data, two-thirds used for training (i.e. 0.66) and one-third used for testing.

Algorithm Implementation/Model Build

The following models were built in R using the training data set, refer for below for each model's summary results. For additional details, refer to the RMD document submitted with this report.

Decision Tree Model Plot:



As noted above, the top 5 variables driving the decision tree are:

Variable	Weight
Income	100%
MntWines	85%
NumCatalogPurchases	15%
MntFruits	6%
Teenhome	5%

Naïve Bayes:

Documented below is the initial Naïve Bayes model:

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
         0          1
0.93851351 0.06148649

Conditional probabilities:
   Year_Birth
Y       [,1]     [,2]
  0 1968.646 11.79611
  1 1969.275 13.56635

   Education
Y     2n Cycle      Basic Graduation     Master        PhD
  0 0.09325681 0.02510760 0.49426112 0.17503587 0.21233859
  1 0.08333333 0.01041667 0.56250000 0.11458333 0.22916667

   Marital_Status
Y         Absurd        Alone     Divorced      Married       Single     Together
  0 0.0007158196 0.0028632785 0.1095204009 0.3808160344 0.2047244094 0.2655690766
  1 0.0202020202 0.0101010101 0.1010101010 0.3333333333 0.2727272727 0.2020202020
   Marital_Status
Y          Widow         YOLO
  0 0.0336435218 0.0021474588
  1 0.0505050505 0.0101010101

   Income
Y       [,1]     [,2]
```

As documented below, the initial prediction results, when the model was executed against the test data set, returned an accuracy of 0.7947:

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 556   5
         1 151  48

               Accuracy : 0.7947
                 95% CI : (0.7642, 0.8229)
    No Information Rate : 0.9303
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3043

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.7864
            Specificity : 0.9057
         Pos Pred Value : 0.9911
         Neg Pred Value : 0.2412
             Prevalence : 0.9303
         Detection Rate : 0.7316
   Detection Prevalence : 0.7382
      Balanced Accuracy : 0.8460

       'Positive' Class : 0
```

To improve on this, we attempted to tune the original model through the following updates:

```r
model_nb2 <- train(AcceptedCmp1 ~ ., data = trainModel, method = "nb",
             trControl = trainControl(method = "cv", number = 3),
             tuneGrid = expand.grid(fL = 1:3, usekernel = c(TRUE, FALSE), adjust = 1:3))

predNB2=predict(model_nb2, newdata=testModel, type=c("raw"))

confusionMatrix(predNB2, testModel$AcceptedCmp1)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 709  51
         1   0   0

               Accuracy : 0.9329
                 95% CI : (0.9127, 0.9496)
    No Information Rate : 0.9329
    P-Value [Acc > NIR] : 0.5372

                  Kappa : 0

 Mcnemar's Test P-Value : 2.534e-12

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.9329
         Neg Pred Value :    NaN
             Prevalence : 0.9329
         Detection Rate : 0.9329
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 0
```

As documented within the results, when the updated NB model executed against the same test data set, it returned results with 0.9329 accuracy, a substantial improvement.

Random Forest:

Documented below, is the first version of the random forest model:

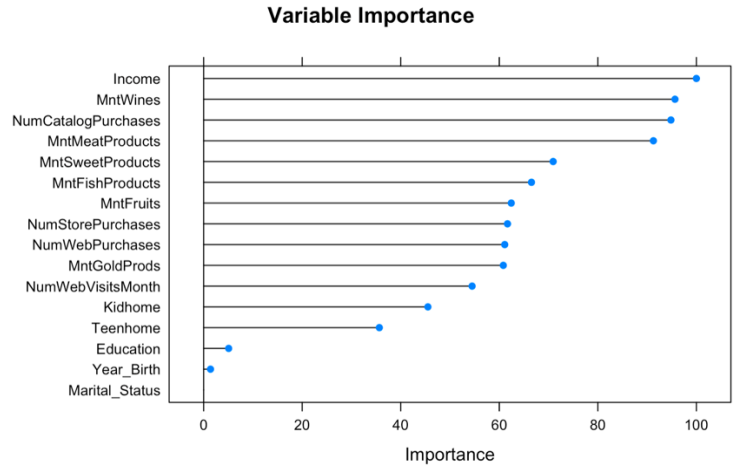
```
Call:
 randomForest(formula = AcceptedCmp1 ~ ., data = trainModel, ntree = 10,      na.action =
na.exclude)
               Type of random forest: classification
                     Number of trees: 10
No. of variables tried at each split: 4

        OOB estimate of  error rate: 6.28%
Confusion matrix:
     0  1 class.error
0 1346 27  0.01966497
1   65 26  0.71428571
```

As noted within the model parameters, the model executed with 10 trees and returned an accuracy of 0.9303 when executed against the training dataset.

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 707  53
         1   0   0

               Accuracy : 0.9303
                 95% CI : (0.9098, 0.9473)
    No Information Rate : 0.9303
    P-Value [Acc > NIR] : 0.5365

                  Kappa : 0

 Mcnemar's Test P-Value : 9.148e-13

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.9303
         Neg Pred Value :    NaN
             Prevalence : 0.9303
         Detection Rate : 0.9303
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 0
```

To compare our results against the decision tree results (also executed with nodes=10), we found that although the accuracy of the Naive Bayes was much lower, both models ranked the variables similarly in their order of importance for the calculations.

Random Forest Variables of Importance:

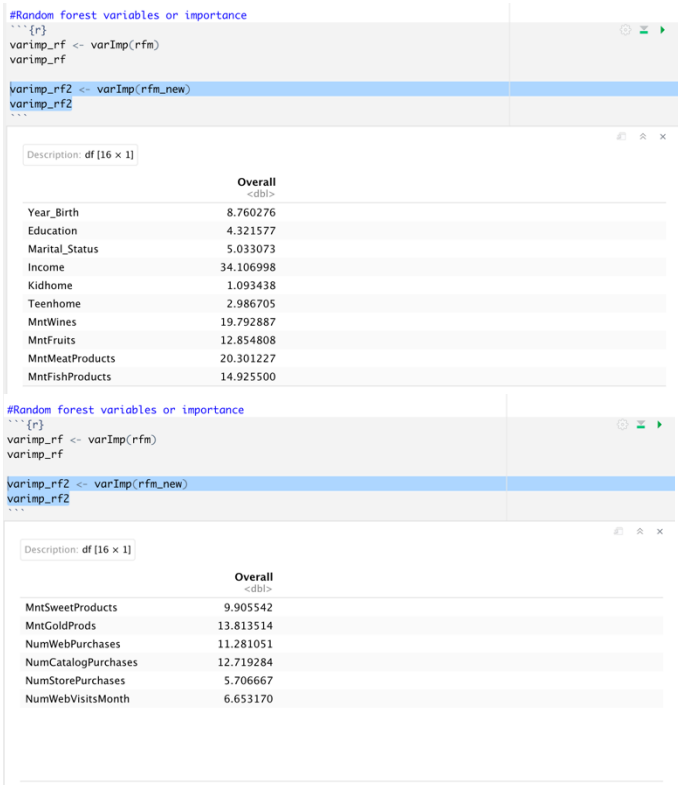


To understand if this could be improved, we re-ran the random forest with ntree = 13, to coincide with K-means results (noted below). After executing our new random forest model against the test data, we saw up uptick in accuracy to 0.9368.

```{r}
rfm_new <- randomForest(AcceptedCmp1~., data=trainModel, ntree=13, na.action=na.exclude)

predRF_new <- predict(rfm_new, testModel, type=c("class"))

confusionMatrix(predRF_new, testModel$AcceptedCmp1)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 704  43
         1   5   8

               Accuracy : 0.9368
                 95% CI : (0.9171, 0.9531)
    No Information Rate : 0.9329
    P-Value [Acc > NIR] : 0.3654

                  Kappa : 0.229

 Mcnemar's Test P-Value : 9.27e-08

            Sensitivity : 0.9929
            Specificity : 0.1569
         Pos Pred Value : 0.9424
         Neg Pred Value : 0.6154
             Prevalence : 0.9329
         Detection Rate : 0.9263
   Detection Prevalence : 0.9829
      Balanced Accuracy : 0.5749

       'Positive' Class : 0
```

To understand what makes up and drives the incremental increase in accuracy of the random forest model, we completed a deeper dive of the model variables, refer below:



#Random forest variables or importance

```{r}
varimp_rf <- varImp(rfm)
varimp_rf

varimp_rf2 <- varImp(rfm_new)
varimp_rf2
```

Description: df [16 × 1]

	Overall <dbl>
Year_Birth	8.760276
Education	4.321577
Marital_Status	5.033073
Income	34.106998
Kidhome	1.093438
Teenhome	2.986705
MntWines	19.792887
MntFruits	12.854808
MntMeatProducts	20.301227
MntFishProducts	14.925500

#Random forest variables or importance

```{r}
varimp_rf <- varImp(rfm)
varimp_rf

varimp_rf2 <- varImp(rfm_new)
varimp_rf2
```

Description: df [16 × 1]

	Overall <dbl>
MntSweetProducts	9.905542
MntGoldProds	13.813514
NumWebPurchases	11.281051
NumCatalogPurchases	12.719284
NumStorePurchases	5.706667
NumWebVisitsMonth	6.653170

As noted above, the top 5 variables driving the random forest model are:

Variable	**Weight**
Income	34.1
MntMeatProducts	20.3

MntWines	19.7
MntFishProducts	14.9
MntGoldProds	13.8

(Refer to results section for a side-by-side comparison of the decision tree and random forest model variables).

SVM Model Summary:

```
Call:
svm(formula = AcceptedCmp1 ~ ., data = trainModel, na.action = na.exclude)


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

Number of Support Vectors:  283
```

Documented above is the original SVM model, refer below for the initial prediction results using the test data. Note, accuracy was reported at 0.9303.

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 707  53
         1   0   0

               Accuracy : 0.9303
                 95% CI : (0.9098, 0.9473)
    No Information Rate : 0.9303
    P-Value [Acc > NIR] : 0.5365

                  Kappa : 0

 Mcnemar's Test P-Value : 9.148e-13

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.9303
         Neg Pred Value :    NaN
             Prevalence : 0.9303
         Detection Rate : 0.9303
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 0
```

To see if additional tuning could improve the model, we standardized the data set and re-built a new SVM model. As noted, the accuracy reported by the new SVM model against the same test data increased to 0.9382. Refer below:

```r
predSVMnew=predict(model_svm_rbf, newdata=testModel, type=c("raw"), na.action=na.exclude)

confusionMatrix(predSVMnew, testModel$AcceptedCmp1)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 709  47
         1   0   4

               Accuracy : 0.9382
                 95% CI : (0.9186, 0.9542)
    No Information Rate : 0.9329
    P-Value [Acc > NIR] : 0.3115

                  Kappa : 0.137

 Mcnemar's Test P-Value : 1.949e-11

            Sensitivity : 1.00000
            Specificity : 0.07843
         Pos Pred Value : 0.93783
         Neg Pred Value : 1.00000
             Prevalence : 0.93289
         Detection Rate : 0.93289
   Detection Prevalence : 0.99474
      Balanced Accuracy : 0.53922

       'Positive' Class : 0
```

KNN:

```
k-Nearest Neighbors

1480 samples
  16 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 1480, 1480, 1480, 1480, 1480, 1480, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.9192012  0.12562743
  7  0.9253868  0.12721900
  9  0.9272014  0.09246066

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
```

Refer above for first KNN model built with the training data set. As noted below, the model results returned an accuracy of 0.9342.

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 707  50
         1   0   3

               Accuracy : 0.9342
                 95% CI : (0.9142, 0.9508)
    No Information Rate : 0.9303
    P-Value [Acc > NIR] : 0.3677

                  Kappa : 0.1004

 Mcnemar's Test P-Value : 4.219e-12

            Sensitivity : 1.0000
            Specificity : 0.0566
         Pos Pred Value : 0.9339
         Neg Pred Value : 1.0000
             Prevalence : 0.9303
         Detection Rate : 0.9303
   Detection Prevalence : 0.9961
      Balanced Accuracy : 0.5283

       'Positive' Class : 0
```

As noted above, the KKN model was originally built via the bootstrap method, in an effort to further tune the model to get the best results, we standardized the training dataset – refer below.

```
#KNN - highest results, so used this model to continue refinement
```{r}
pre_process <- preProcess(trainModel, method = c("scale", "center"))
pre_process
```

Created from 1480 samples and 17 variables

Pre-processing:
  - centered (14)
  - ignored (3)
  - scaled (14)
```

```
  Year_Birth          Education    Marital_Status     Income            Kidhome
Min.   :-6.3095   2n Cycle  :134   Married :557    Min.   :-1.87910   Min.   :-0.822
1st Qu.:-0.8138   Basic     : 34   Together:396    1st Qu.:-0.62424   1st Qu.:-0.822
Median : 0.1022   Graduation:745   Single  :330    Median :-0.05526   Median :-0.822
Mean   : 0.0000   Master    :242   Divorced:144    Mean   : 0.00000   Mean   : 0.000
3rd Qu.: 0.6850   PhD       :325   Widow   : 49    3rd Qu.: 0.58961   3rd Qu.: 1.030
Max.   : 2.2671                    Alone   :  3    Max.   :22.95773   Max.   : 2.881
                                    (Other) :  1
   Teenhome          MntWines         MntFruits       MntMeatProducts   MntFishProducts
Min.   :-0.9188   Min.   :-0.9078   Min.   :-0.6593   Min.   :-0.7419   Min.   :-0.6801
1st Qu.:-0.9188   1st Qu.:-0.8371   1st Qu.:-0.6095   1st Qu.:-0.6710   1st Qu.:-0.6243
Median :-0.9188   Median :-0.3966   Median :-0.4600   Median :-0.4452   Median :-0.4569
Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
3rd Qu.: 0.9188   3rd Qu.: 0.6073   3rd Qu.: 0.1875   3rd Qu.: 0.2768   3rd Qu.: 0.2172
Max.   : 2.7564   Max.   : 3.4911   Max.   : 4.2967   Max.   : 6.8983   Max.   : 4.1365

MntSweetProducts   MntGoldProds     NumWebPurchases   NumCatalogPurchases
Min.   :-0.6511   Min.   :-0.8425   Min.   :-1.4459   Min.   :-0.9118
1st Qu.:-0.6271   1st Qu.:-0.6671   1st Qu.:-0.7279   1st Qu.:-0.9118
Median :-0.4592   Median :-0.3748   Median :-0.3689   Median :-0.2394
Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
3rd Qu.: 0.1403   3rd Qu.: 0.2099   3rd Qu.: 0.7080   3rd Qu.: 0.4330
Max.   : 5.6563   Max.   : 6.2123   Max.   : 8.2470   Max.   : 8.5016

NumStorePurchases NumWebVisitsMonth AcceptedCmp1
Min.   :-1.7678   Min.   :-2.1638   0:1387
1st Qu.:-0.8562   1st Qu.:-0.9374   1:  93
Median :-0.2484   Median : 0.2889
Mean   : 0.0000   Mean   : 0.0000
3rd Qu.: 0.6632   3rd Qu.: 0.6977
Max.   : 2.1826   Max.   : 6.0119
```

With the standardized data set, the new model had an initial accuracy of 0.9329 – refer below.

```
#New model using a standardized data set
```{r}
model_knn1 <- train(AcceptedCmp1 ~ ., data = KNNcampaign_train1, method = "knn")
predict_knn1 <- predict(model_knn1, newdata = KNNcampaign_test1)

confusionMatrix(predict_knn1, KNNcampaign_test1$AcceptedCmp1)
```

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 708 50
         1   1   1

               Accuracy : 0.9329
                 95% CI : (0.9127, 0.9496)
    No Information Rate : 0.9329
    P-Value [Acc > NIR] : 0.5372

                  Kappa : 0.0328

 Mcnemar's Test P-Value : 1.801e-11

            Sensitivity : 0.99859
            Specificity : 0.01961
         Pos Pred Value : 0.93404
         Neg Pred Value : 0.50000
             Prevalence : 0.93289
         Detection Rate : 0.93158
   Detection Prevalence : 0.99737
      Balanced Accuracy : 0.50910

       'Positive' Class : 0
```

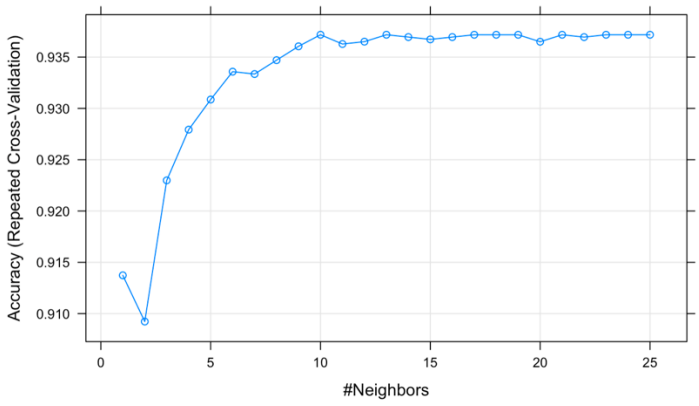
The new model's results were a decrease in accuracy from the first pass, so we continued to tune. After second tuned KNN model also leveraged the new standardized data, results outlined below. The highest accuracy for the KNN model was reported as 0.9371 with a k=13. The tuned results for this new KNN model are a slight increase over the original model (0.934).

```r
model_knn2 <- train(AcceptedCmp1 ~ ., data = KNNcampaign_train1, method = "knn",
                    tuneGrid = data.frame(k = seq(1, 25)),
                    trControl = trainControl(method = "repeatedcv",
                                             number = 10, repeats = 3))
print(model_knn2)
```

```
k-Nearest Neighbors

1480 samples
  16 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 1331, 1332, 1332, 1333, 1333, 1331, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.9137317   0.2059048219
   2  0.9092286   0.1881207563
   3  0.9229875   0.1077026396
   4  0.9279304   0.1019733907
   5  0.9308630   0.0772391002
   6  0.9335795   0.1300151858
   7  0.9333497   0.0860621558
   8  0.9347011   0.0950835455
   9  0.9360494   0.0692861586
  10  0.9371771   0.0720290762
  11  0.9362747   0.0484919553
  12  0.9365030   0.0268750333
  13  0.9371771   0.0296268276
  14  0.9369519   0.0173656132
  15  0.9367267   0.0110272640
  16  0.9369504  -0.0004117768
  17  0.9371741   0.0000000000
  18  0.9371726   0.0059251262
```

Refer below for the graphical representation of the KNN model with its K measurements.

```r
plot(model_knn2)
```



Interpreting Results

Documented below is the model accuracy summarized for each of our models when executed against the test data set. For additional details, refer to the RMD document submitted with this report.

Algorithm	Model Accuracy	Tuned Model Accuracy	Key Summary Points
Naïve Bayes	0.794	0.9329	Dramatic increase in accuracy with tuned results

Random Forest	0.930	0.9367	Model tuned based on KNN results for identification of k and increased accuracy
SVM	0.930	0.9382	Tuned with standardized data and reran.
KNN	0.934	0.9371	Tuned with standardized data and reran. Final results indicated when K= 13, accuracy is highest

Important Variables for Model Prediction Analysis:

Top Decision Tree Variables at 10 nodes	**Top Random Forest Variables at k=10**	**Top Random Forest Variables at k=13**
Income	Income	Income
MntWines	MntWines	MntMeatProducts
NumCatalogPurchases	NumCatalogPurchases	MntWines
MntFruits	MntMeatProducts	MntFishProducts
MntFishProducts	MntSweetProducts	MntGoldProds
	Accuracy = 0.930	Accuracy = 0.9367

To have the ability to predict when a customer will accept a campaign is powerful for the business, especially in combination with the previous Apriori results. The Apriori results proved valuable to us in accurately identifying who is **not** accepting the campaigns. This allows us to now target a whole group of untapped individuals **and** because of the strong models we have built we can now start to predict if they will accept the campaign. Additionally, we noted the following interdependencies within model results:

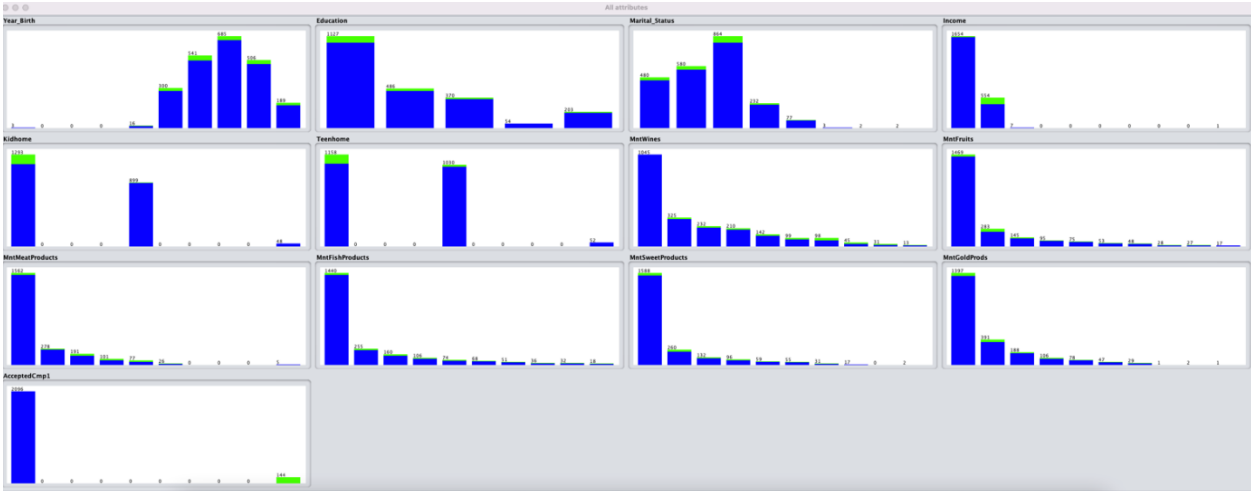
- We saw that the Decision Tree and Random Forest identified the same important variables for prediction
- We noted that when results from KNN were used (k=13) and applied to the Random Forest model, we were able to drive an even higher prediction accuracy.

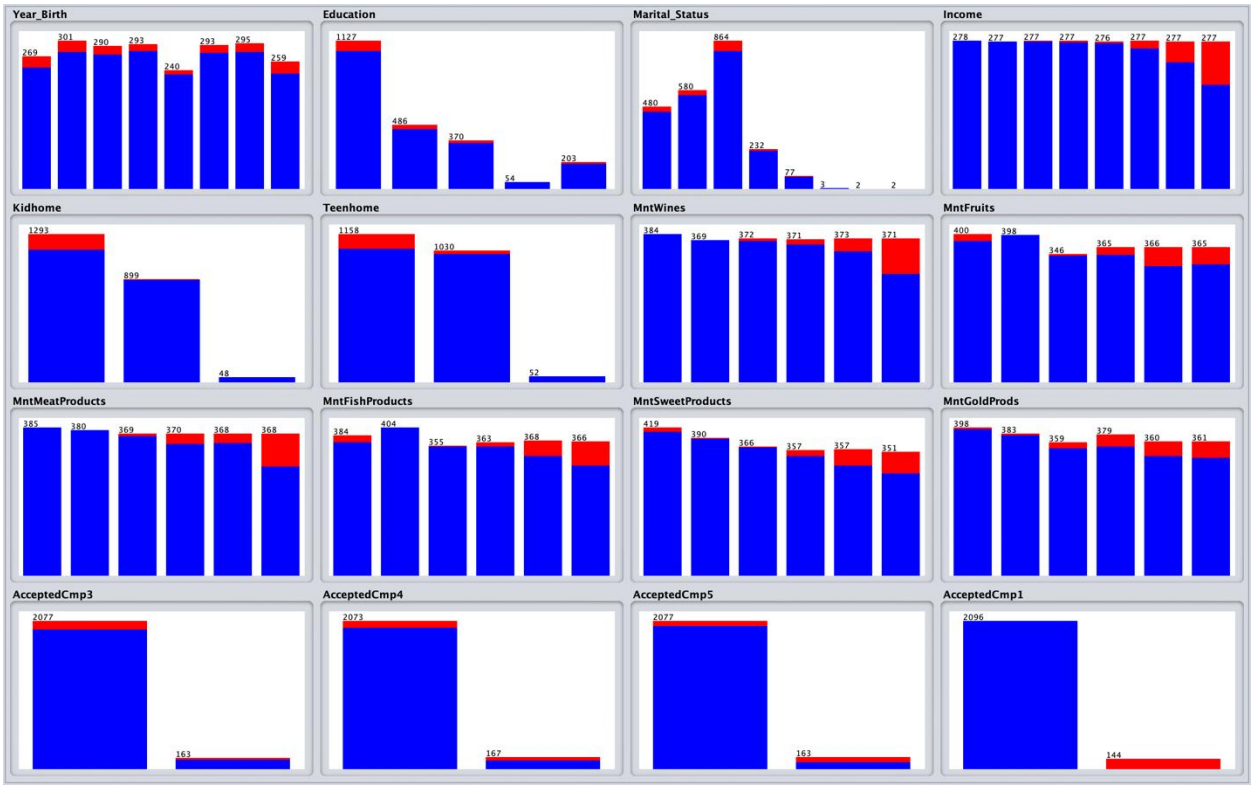
Conclusion

Based on the results found above, we concluded that we should be targeting our campaign to customers who have less than 1 kid or teen at home, with a medium or higher income range, who spends a decent amount of money on meat, wines, fish and/or gold. Based on the K-Mean clustering, cluster group number 4 is our target customer group for our upcoming campaign since it has the highest count of customer who meet our requirements.

Appendix – WEKA Documentation

Pre-processing:





Final Results:

Documented below is the Weka criteria for obtaining the final results:

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 15 -T 0 -C 0.1 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -A -c -1
Relation:     Marketing-weka.filters.unsupervised.attribute.Remove-R26-29-weka.filters.unsupervised.attribute.Remove-R1,8-9,16-20-weka.filters.unsupervised.attribute.Discretize
Instances:    2240
Attributes:   13
              Year_Birth
              Education
              Marital_Status
              Income
              Kidhome
              Teenhome
              MntWines
              MntFruits
              MntMeatProducts
              MntFishProducts
              MntSweetProducts
              MntGoldProds
              AcceptedCmp1
=== Associator model (full training set) ===
```

Documented below are the final results leveraged from Weka:

```
Apriori
=======

Minimum support: 0.2 (448 instances)
Minimum metric <confidence>: 0.1
Number of cycles performed: 16

Generated sets of large itemsets:

Size of set of large itemsets L(1): 9

Size of set of large itemsets L(2): 6

Best rules found:

 1. Kidhome='(0.666667-1.333333]' Teenhome='(-inf-0.666667]' 503 ==> AcceptedCmp1='(-inf-0.5]' 498    conf:(0.99)
 2. Kidhome='(0.666667-1.333333]' 899 ==> AcceptedCmp1='(-inf-0.5]' 890    conf:(0.99)
 3. Teenhome='(0.666667-1.333333]' 1030 ==> AcceptedCmp1='(-inf-0.5]' 1003    conf:(0.97)
 4. Education=Graduation Teenhome='(0.666667-1.333333]' 511 ==> AcceptedCmp1='(-inf-0.5]' 497    conf:(0.97)
 5. Kidhome='(-inf-0.666667]' Teenhome='(0.666667-1.333333]' 625 ==> AcceptedCmp1='(-inf-0.5]' 603    conf:(0.96)
 6. Marital_Status=Together 580 ==> AcceptedCmp1='(-inf-0.5]' 548    conf:(0.94)
 7. Education=PhD 486 ==> AcceptedCmp1='(-inf-0.5]' 456    conf:(0.94)
 8. Marital_Status=Single 480 ==> AcceptedCmp1='(-inf-0.5]' 449    conf:(0.94)
 9. Education=Graduation 1127 ==> AcceptedCmp1='(-inf-0.5]' 1045    conf:(0.93)
10. Marital_Status=Married 864 ==> AcceptedCmp1='(-inf-0.5]' 801    conf:(0.93)
11. Teenhome='(-inf-0.666667]' 1158 ==> AcceptedCmp1='(-inf-0.5]' 1043    conf:(0.9)
12. Kidhome='(-inf-0.666667]' 1293 ==> AcceptedCmp1='(-inf-0.5]' 1160    conf:(0.9)
13. Education=Graduation Teenhome='(-inf-0.666667]' 593 ==> AcceptedCmp1='(-inf-0.5]' 527    conf:(0.89)
14. Education=Graduation Kidhome='(-inf-0.666667]' 650 ==> AcceptedCmp1='(-inf-0.5]' 575    conf:(0.88)
15. Kidhome='(-inf-0.666667]' Teenhome='(-inf-0.666667]' 638 ==> AcceptedCmp1='(-inf-0.5]' 528    conf:(0.83)
```