Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

# Assessment Submission Form

| | |
|---|---|
| **Student Name** | Ross Finnegan |
| **Student ID Number** | 15320532 |
| **Course Title** | Management Science and Information Systems Studies (MSISS) |
| **Module Title** | Data Analytics (ST4003) |
| **Lecturer(s)** | Dr. Myra O'Regan |
| **Assessment Title** | Assignment |
| **Date Submitted** | 03/12/2018 |

Signed …Ross Finnegan……………………………………………………… Date …03/12/2018………

# ST4003- Data Analytics Assignment

Ross Finnegan - 15320532

## Contents

## 1. Preliminary Analysis

### 1.1 The Datasets

Two datasets were provided for this assignment: “Churn” and “Kingscounty House data”.

The Churn dataset contains 7,043 rows of 21 variables outlined below. Each row tracks a set of variables for each unique customer and the binary outcome variable Churn which recorded whether the customer churned (left the company) or remained.

| Variable | Description |
|---|---|
| **customerID** | Unique ID number for each customer |
| **Gender** | Customer gender (male/female) |
| **SeniorCitizen** | 1: Yes 0: No |
| **Partner** | Does customer have a partner? |
| **Dependents** | Does customer have any dependent children? |
| **Tenure** | How long has the customer been with the company? (measured in months) |
| **PhoneService** | Whether the customer has phone service or not |
| **MultipleLines** | Whether the customer has multiple phone lines |
| **InternetService** | Internet service method (DSL, Fiber optic, None) |
| **OnlineSecurity** | Does the customer have online security software? |
| **OnlineBackup** | Does the customer have online backup? |
| **DeviceProtection** | Does the customer avail of device protection |
| **TechSupport** | Whether the customer has tech support or not |
| **StreamingTV** | Whether the customer has TV streaming or not |
| **StreamingMovies** | Whether the customer has movie streaming or not |
| **Contract** | Customer’s contract plan (month-to-month, one year, two year) |
| **PaperlessBilling** | Does the customer avail of paperless billing? |
| **PaymentMethod** | Customer’s payment method (electronic check, mailed check, bank transfer, credit card) |
| **MonthlyCharges** | Amount charged to the customer per month |
| **TotalCharges** | Total amount charged to the customer since the outset |
| **Churn (Target Variable)** | Whether the customer churned or not (yes/no) |

The Kingscounty House Data contains 21,613 rows of 21 variables outlined below. Each row records details for each unique house sold in Kings County, Washington over the period May 2014-May 2015 and the price the house was sold (continuous outcome variable).

| Variable | Description |
|---|---|
| **id** | Unique ID for each house sold |
| **date** | Date of sale |
| **price (Target Variable)** | Price of each home sold |
| **bedrooms** | Number of bedrooms |
| **bathrooms** | Number of bathrooms (0.5 accounts for a room with a toilet but no shower) |
| **sqft_living** | Square footage of the home interior living space |
| **sqft_loft** | Square footage of the land space |
| **floors** | Number of floors |
| **waterfront** | Whether the apartment was overlooking the waterfront or not |
| **view** | Index 0-4 how good the view of the property is |
| **condition** | Index 1-5 on the condition of the apartment |
| **grade** | Index 1-13 on the quality of construction & design. 1-3 Low, 4-10 Average, 11-13 High. |
| **sqft_above** | Square footage of the interior housing above ground level |
| **sqft_basement** | Square footage of the interior housing space below ground level |
| **yr_built** | Year the house was built |
| **yr_renovated** | Year of the house's most recent renovation |
| **zipcode** | What zipcode area the house is in |
| **lat** | Latitude |
| **long** | Longitude |
| **sqft_living15** | Average square footage of the interior living space of the nearest 15 neighbours |
| **sqft_lot15** | Average square footage of the lots of the nearest 15 neighbours |

## 1.2 Data Cleansing

Data quality is essential to the quality of the predictions from any model. Garbage in, garbage out! (or nothing out!) Therefore, before any analysis is done, the data must be inspected and prepared.

The '*vis_dat'* function from the visdat library is a great way to quickly visualise a full dataset. The churn dataset required many of the variables to be recoded to factors. Figure 1 also indicates (with NA datatypes) that there are a small number of missing values in the total charges column and that this should be investigated further.

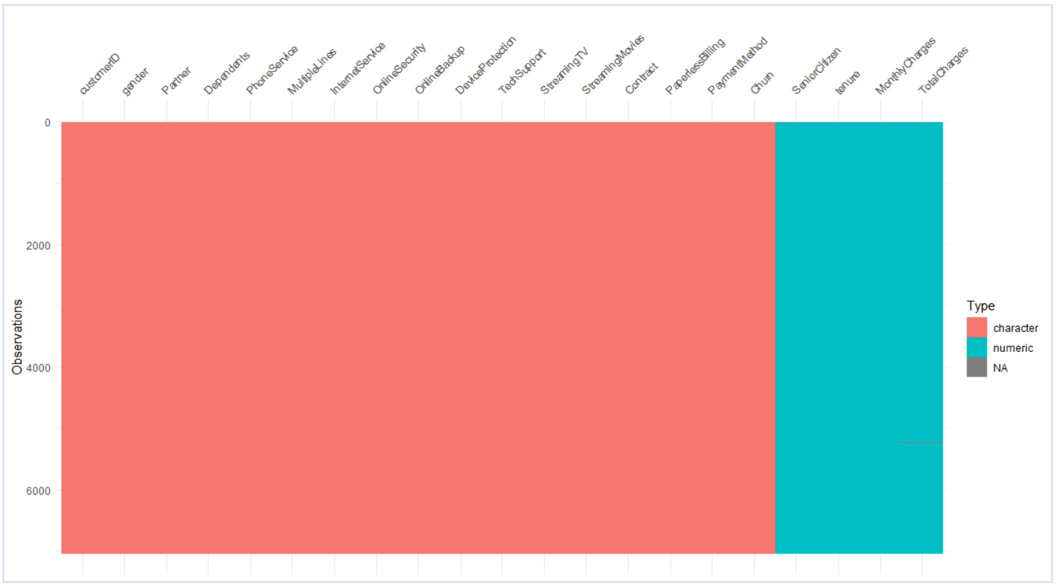
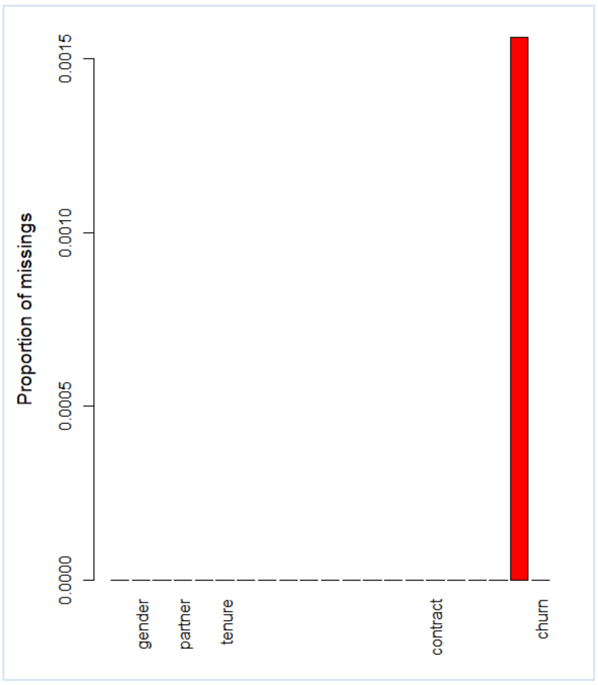
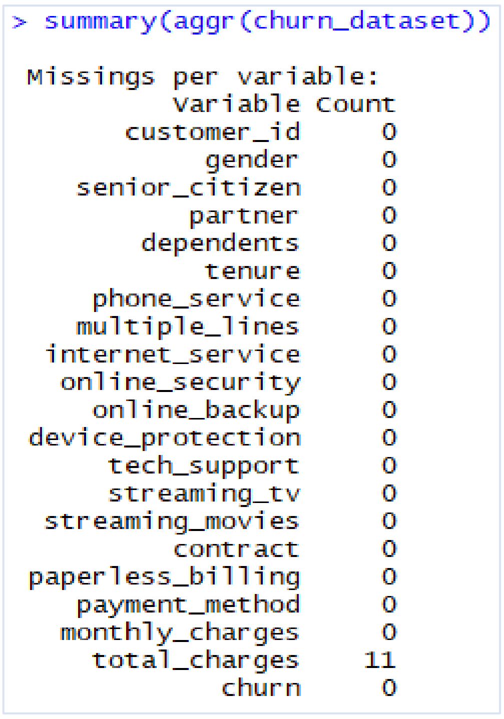


*Figure 1: Preliminary Churn Dataset Visualisation*

By isolating the row with missing data for 'total_charges', it became clear these 11 customers (Figure 2) all had a tenure of 0 months and as a result did not have a 'total_charges' value. As a result, the NA values were recoded as zeros.





```
> summary(aggr(churn_dataset))

 Missings per variable:
          Variable Count
       customer_id     0
            gender     0
    senior_citizen     0
           partner     0
        dependents     0
            tenure     0
     phone_service     0
    multiple_lines     0
  internet_service     0
   online_security     0
     online_backup     0
 device_protection     0
      tech_support     0
      streaming_tv     0
  streaming_movies     0
          contract     0
 paperless_billing     0
    payment_method     0
   monthly_charges     0
     total_charges    11
             churn     0
```

*Figure 2: Identifying missing values using aggr function*

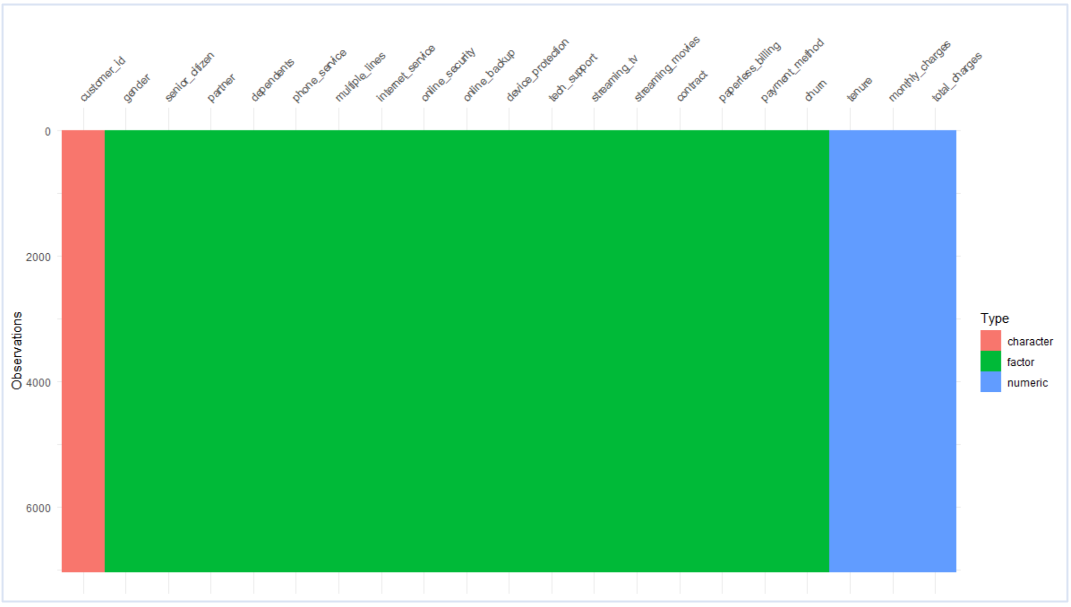


*Figure 3: Churn dataset cleaned*

The Kingscounty Data did not have any missing data. As with the Churn dataset, a number of variables were recoded to factors to assist later analysis. The date variable was in an awkward format and recoded to a more useable format. For the sake of predictive analysis, Unique ID identifiers were removed from each dataset. This is because a powerful classifier will use that column to perfectly fit the training set which would result in a useless model for future use.

After applying these changes, the *‘clean_names’* function from the janitor package was used on both datasets to ensure uniformity between variable names. Both datasets were checked for duplicated rows using the ‘*duplicated’* function and neither contained any duplicates. Following these provisional checks complete, the data was ready to be evaluated.

## 1.3 Derived Variables

In order to make the process more creative and to gain a greater understanding of the raw data, a number of variables were derived for both datasets.

Churn:

- Tenure_group: Tenure is measured in number of months. By categorising the number of months into groups such as 1-2 years,3-4 years etc. the data is easier to interpret.
- Phone_and_internet_serv: The data identifies a customer as a phone service customer or an internet service customer. Customers who avail of both services from the provider may be more “integrated” into this providers system and as a result less likely to churn.

Kingscounty:

- Sqft_living_percent_diff: The sqft_living15 provides an interesting comparison of the living space of the house that was sold to that of the surrounding 15 nearest neighbours. By standardising the difference between these living spaces, one can quickly evaluate the relative size of the property in comparison to those nearby.

- Renovation_type: The renovation year variable stores the year of the most recent renovation to the property. Many properties have never been renovated or were renovated many years previously, with years ranging from 1934 to 2015. To make this data more applicable, the renovation_type variable categorises renovation_yr into None, 1900s, 2000s, 2010-2013 or Last Two Years.
- Grade_class: The dataset descriptor outlined three categories for the grade variable which assesses the quality of the design and construction of the building. However, these categories were not included as a variable. Grade_class categorises into Low, Average and High as illustrated by the dataset descriptor

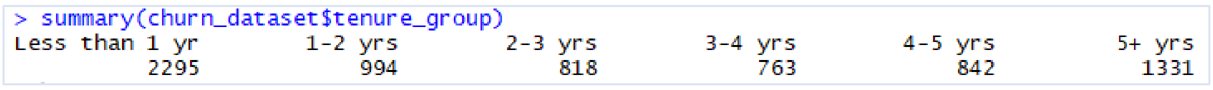
## 1.4 Summary Statistics

Basic summary statistics are very effective when trying to communicate as large an amount of information as possible. For the sake of brevity, three variables from each dataset will be evaluated to gain a better understanding of the data.

### 1.4.1 Churn Summary Statistics

**1) Tenure_group**:
Tenure_group is a derived variable which groups the number of months a customer has been with the service provider into the number of years.

The summary function (Figure 4) in R provides a quick breakdown of the groups however a histogram will provide a better illustration (Figure 5). It is clear from the graph that there are a considerable number (33%) of customers with contracts less than one year.



```
> summary(churn_dataset$tenure_group)
Less than 1 yr        1-2 yrs        2-3 yrs        3-4 yrs        4-5 yrs         5+ yrs
          2295            994            818            763            842           1331
```

*Figure 4: Summary statistics for Tenure Group*

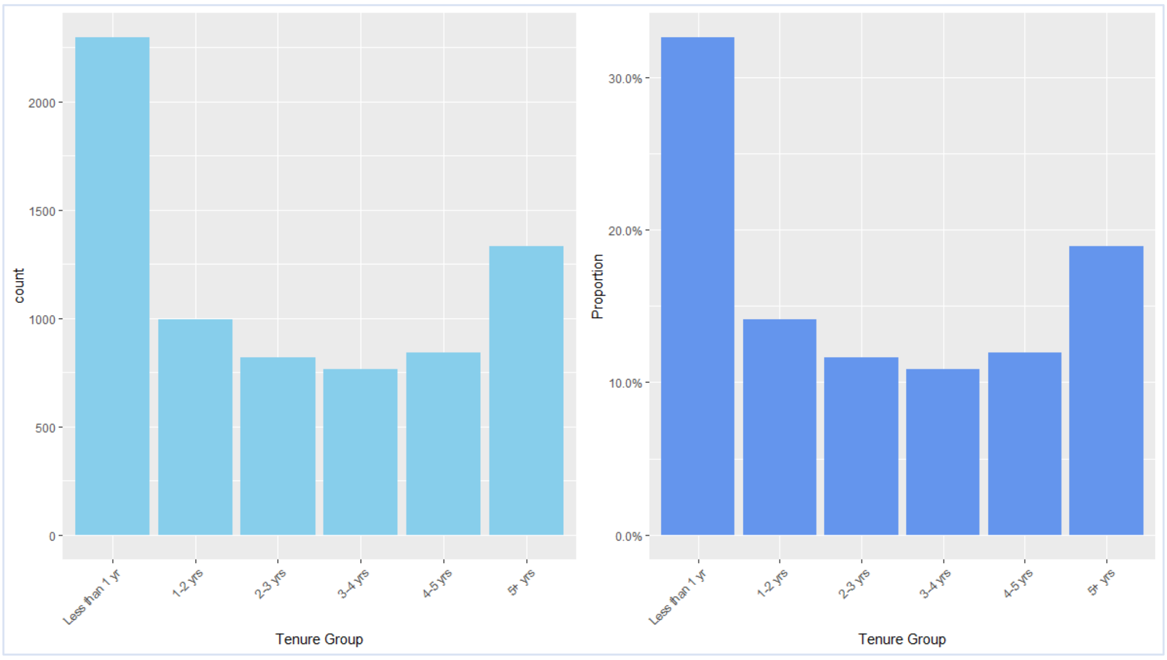


*Figure 5: Boxplots outlining count and proportion of each Tenure Group*

It is clear from Figure 6 that the likelihood of a customer churning within the first 12 months of joining is far greater than any other tenure group. The proportion of customers churning decreases steadily as the tenure increases.

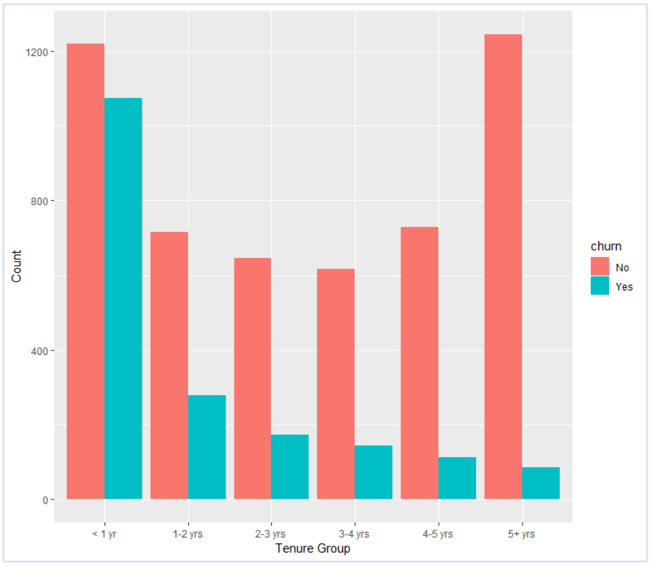
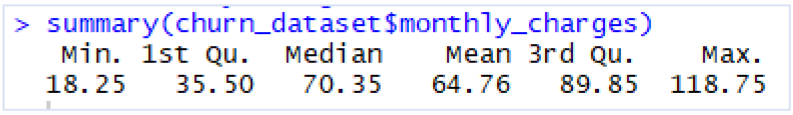


*Figure 6: Proportion of No:Yes to Churn for each Tenure Group*

**2) Monthly_charges:**

The monthly_charges variable is one of the few quantitative variables in the churn dataset. A greater understanding of the amount each customer is paying monthly, and their likelihood of churning provides an interesting insight into the importance of this variable.



```
> summary(churn_dataset$monthly_charges)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  18.25   35.50   70.35   64.76   89.85  118.75 
```

*Figure 7: Summary statistics for Monthly Charges*

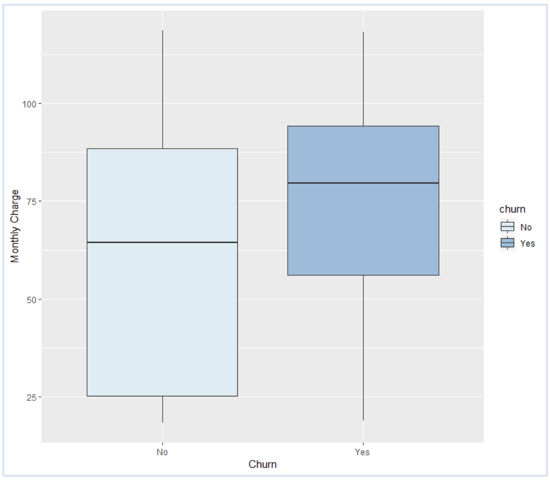


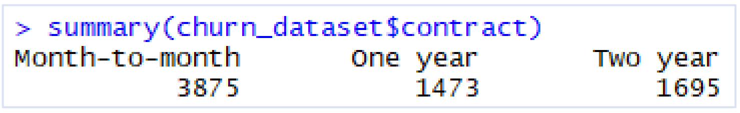
*Figure 8: Boxplot of Monthly Charge for Churn and Remain*

Figure 7 illustrates the wide range of charges incurred by customers each month. It is interesting to note that the range of monthly charges for Remain is a close match to the overall range however the range for customers who Churn is far narrower (Figure 8).

Having evaluated ‘tenure_group’ and ‘monthly_charges’ it is clear that the provider would benefit from targeting customers whose tenure is less than 1 year and are paying a monthly fee greater than $60.

**3) Contract:**

Contract is a categorical variable which describes the type of contract each customer has with the provider. The data provided contains three categories: Month-to-month, One year and Two year. Therefore, Month-to-month is the only “pay-as-you-go” model, while One year and Two year represent a fixed term, presumably binding, contract.



```
> summary(churn_dataset$contract)
Month-to-month       One year       Two year
          3875           1473           1695
```

*Figure 9: Summary statistics for Contract*

A quick look at the breakdown of customers per group (Figure 9) illustrates that there are approximately 50% of customers locked into a contract (One year or Two year) and 50% on a Month-to-month plan.

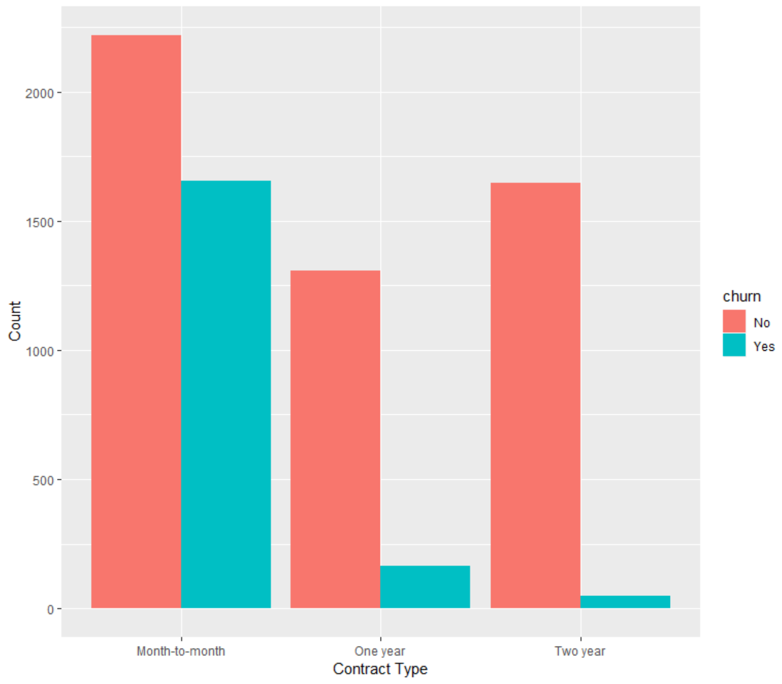


*Figure 10: Proportion of those who Churned or not for each contract type*

Figure 10 demonstrates the relationship of contract type to churn. It is clear that customers on a Month-to-month plan are far more likely to churn, while it is a rare event for a fixed-term contract to be broken. Data is not provided for the reasons for churn, however give there are so few cases of churn for customers on One-year or Two-year contracts, perhaps relocation could be a primary reason for these cases.

## 1.4.2 Kingscounty Summary Statistics

**1) Location:**

The longitude and latitude of each house allows for some insightful visualisations which provide a greater understanding of the role of location on sale price. The 'gmap' function in R provides an excellent mapping capability using the latitude and longitude of each point. Figure 11 is a heat map

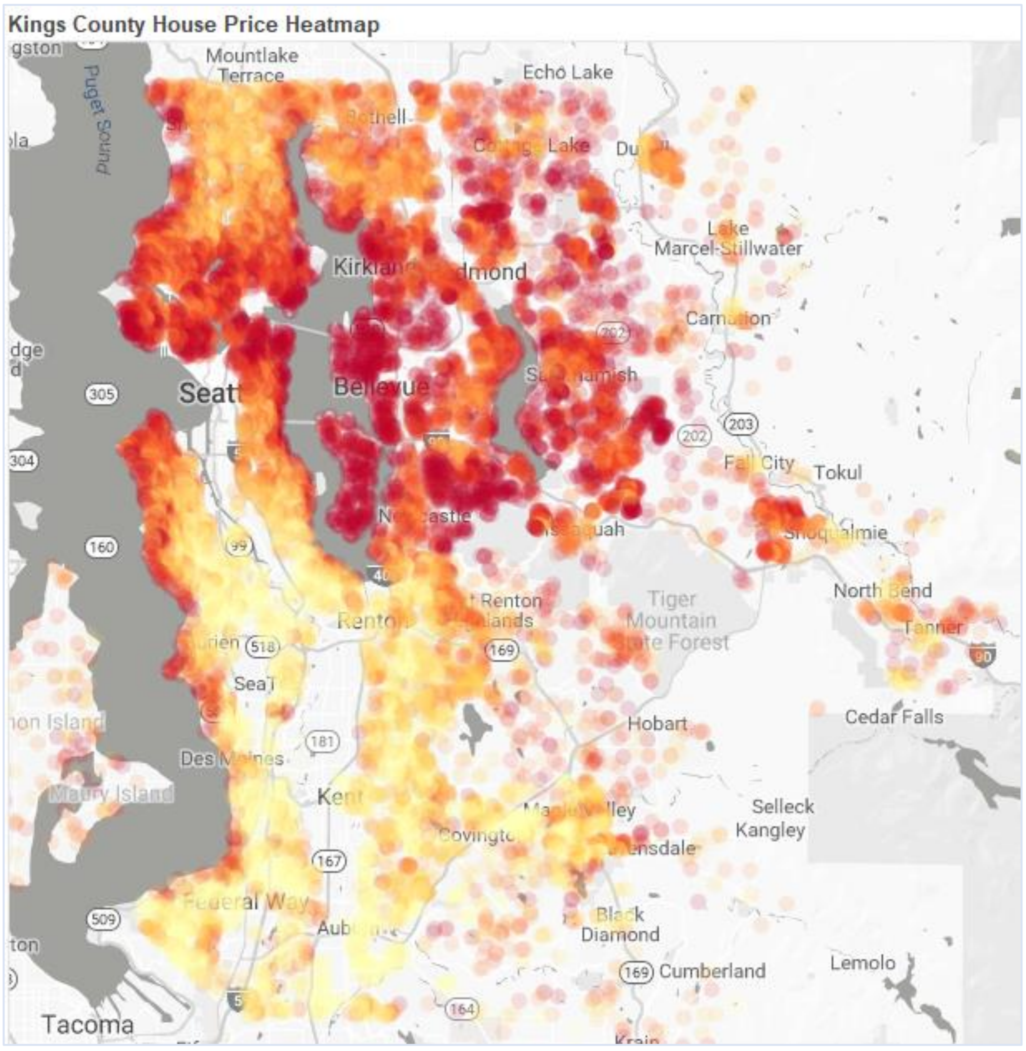


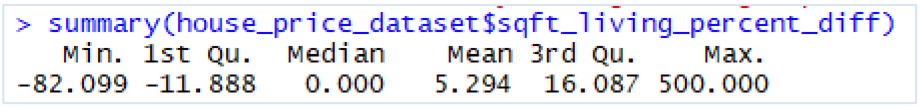
*Figure 11: House Price Heatmap*

of each sale with increasing sale price resulting in a darker red colour. It is clear that there are high value areas of Kings County. Any property by the water appears to go for a far higher price. Bellevue is an expensive area to live while the further out in the suburbs prices tend to be lower.

**2) Sqft_lot_percent_diff:**

This variable is a derived variable designed to give a better understanding of a property's relative size when compared to its nearest neighbours. Figure 12 illustrates that for the dataset provided the mean difference in living space of houses sold is slightly above the average of the fifteen nearest neighbours (5.29%). The range of values is quite narrow with an interquartile range of -11.89% to

16.09%. This is illustrated further the boxplot shown in Figure 13. It is shown that the interquartile range is very narrow when compared to the outliers and minimum/maximum values.



```
> summary(house_price_dataset$sqft_living_percent_diff)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
-82.099 -11.888   0.000   5.294  16.087 500.000 
```

Figure 12: Summary statistics

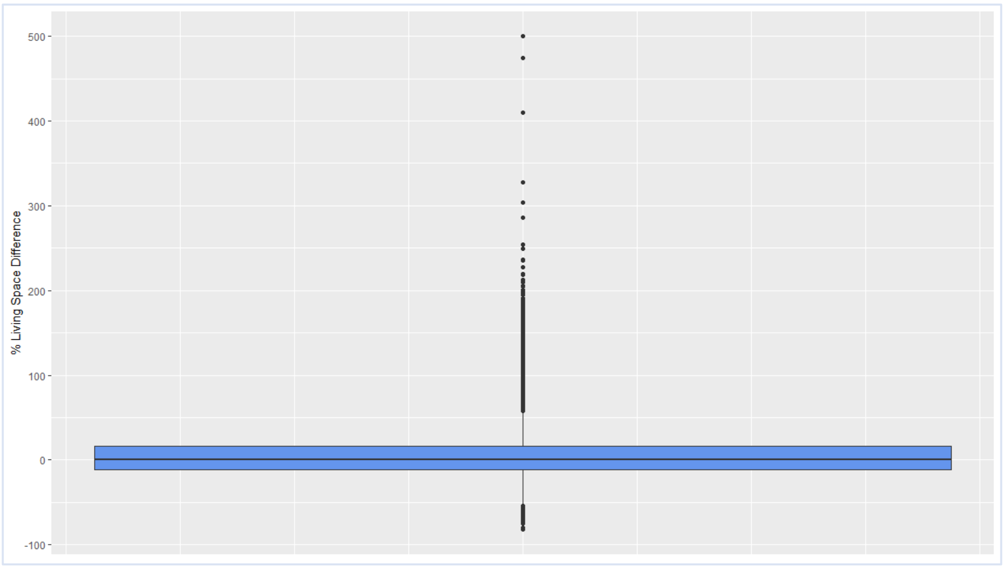


Figure 13: Boxplot showing % Difference in Living Space

Another interesting insight is to plot this % difference against sale price (Figure 14). Given the intensity of data points around the origin the stat_binhex() function is called as a parameter to improve the aesthetic by grouping nearby values and shading according to the count within each hexagon. This graph indicates that majority of houses sold have a slightly larger living space than their neighbours. Outlier properties selling for values greater than approximately $3.4 million all are considerably larger than their neighbours.

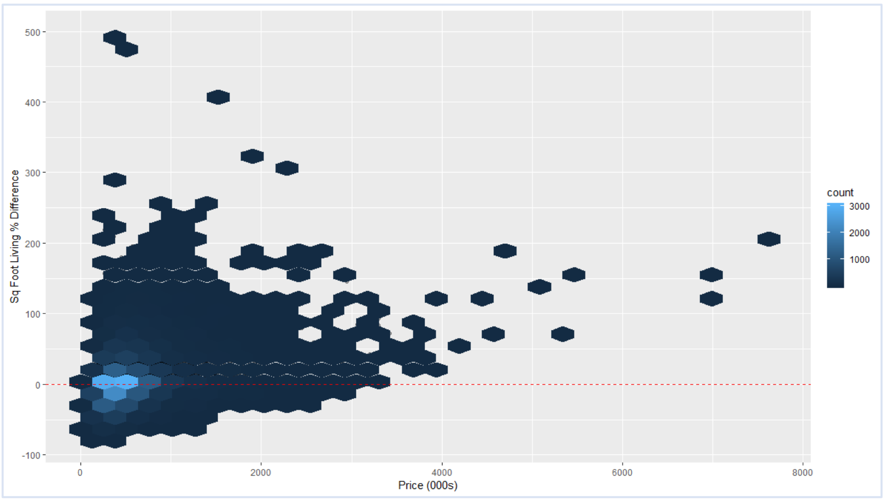


Figure 14: Price vs % Difference

**<u>3) Yr_built:</u>**

An inspection of the year each house sold was built will set some background for the property market in the Kings County property market. Figure 15 illustrates the cyclical nature of construction with the number of builds rising gradually over time. It is interesting to note the effect of the 2008-09 financial crisis and the rapid decline in annual house builds. The jump from 2013 to a record high in 2014 eludes to the recovery of the US economy and the prevalence of new housing in Kings County.

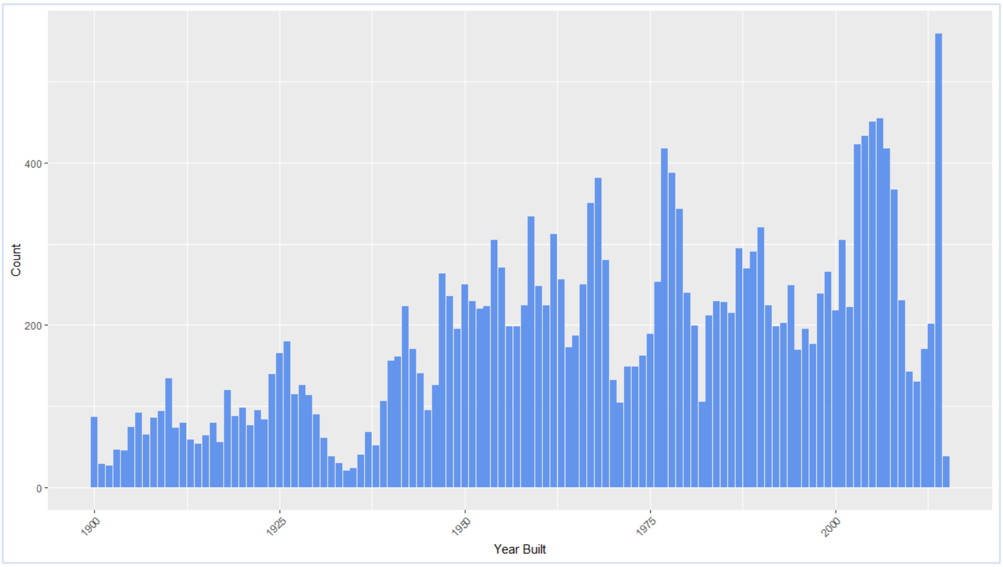


*Figure 15: Count of Annual House Builds*

Intuitively, I assumed that old houses would sell for less than newer buildings however when we analyse the range of sale price for each year it is clear that these properties retain their value over time. Figure 16 shows no significant increase in value of a house based on how recently it was built. Greater number of outliers arise purely from increased number of builds.

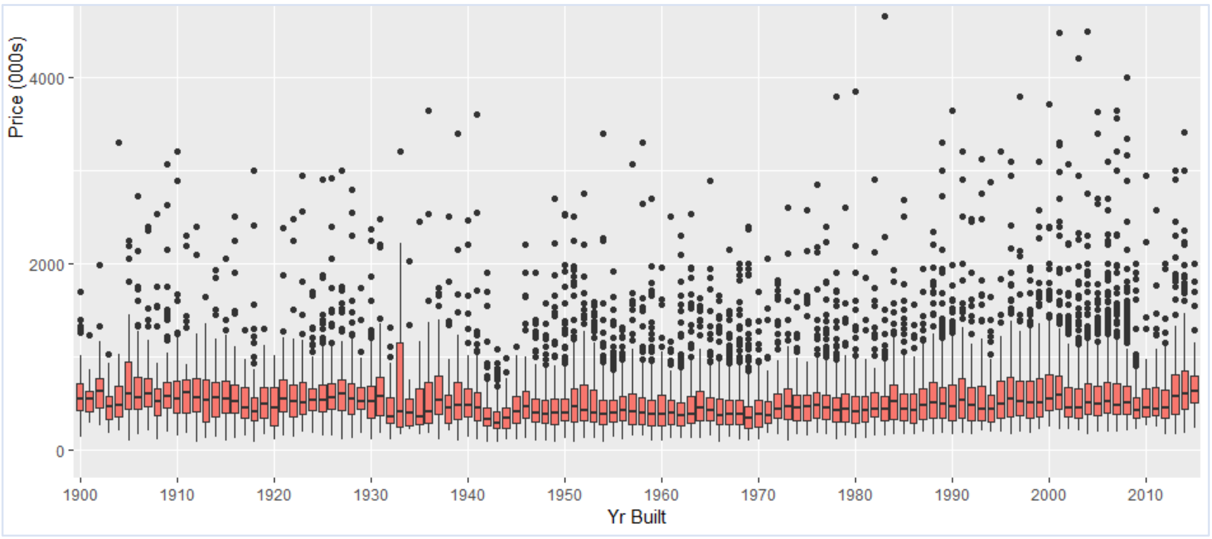


*Figure 16: Price Range for Each Year*

# 2. Trees: CART

## 2.1 Introduction

Decision Trees are a supervised learning method used for classification and regression. The goal is to create a set of rules to assist in the classification/prediction of future data subjects. There are numerous tree algorithms with different approaches for how to grow a tree. They differ in the possible structure of the tree (e.g. number of splits per node), criteria for how to find the splits and when to stop splitting. In the 1970s, Leo Breiman and his colleagues developed Classification and regression trees (CART). CART is one of the more effective algorithms for tree building.

- Classification Trees are used when the target variable is categorical.
- Regression Trees are used when the target variable is continuous.

Trees have a number of advantages over linear regression models and logistic regression. Trees can account for non-linearity between features and the outcome variable as well as where features interact with each other.

Using our two datasets, we will build a Classification Tree for the Churn dataset to help predict customers who are more likely to churn and a Regression Tree for the Kingscounty dataset to define a set of rules that will allow us to predict the sale price of houses.

## 2.2 Splitting the Data

In order to assess the quality of the data, it is necessary to partition the data into a training set which is used to build the tree and a test set which is used to evaluate the accuracy of the tree. It is assumed that the test set will be a representative sample of future data and that the model will perform similarly in the future.

It is important to note that the 'set.seed()' function is vital to ensure repeatability when sampling and reproducing the same tree. The column which has indicator should be removed from the data frame after splitting to ensure that it does not impact the model.

## 2.3 Rpart

The most popular function used to grow CART trees in R is the RPART (Recursive Partitioning and Regression Trees). The rpart algorithm splits the data recursively. Subsets from these splits are split further. Each split is made by evaluating each independent variable and selecting the variable that results in the greatest possible reduction in determining the target variable.

Splitting rules can be constructed in different ways, all are based on the concept of impurity - a measure of the degree of heterogeneity of the leaf nodes. Impurity = 0 suggests that a leaf node is made up entirely of one class. The rpart algorithm allows Entropy and Gini to be set as the measure of impurity.

$$\text{Entropy} = -\sum_{j=1}^{c} p(j\,|\,t)\log_2(p(j\,|\,t)\quad)$$

$$\text{Gini} = \sum_{i=1}^{c}\sum_{\substack{j=1\\ j\neq i}}^{c} p(j\,|\,t)p(i\,|\,t) = 1-\sum_{j=1}^{c} p^2(j\,|\,t)$$

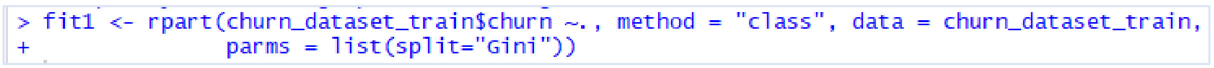
*Figure 17: Common Impurity Functions*

Rpart builds trees using a bottom down approach whereby it grows the maximal tree and then prunes branches using an appropriate criterion such as cross validation error. In general, a larger tree will have lower classification error, however, this runs the risk of overfitting. Therefore, pruning is an essential part of growing trees.

## 2.4 Churn - Classification Tree

### 2.4.1 Growing & Evaluating

Figure 18 shows the call of the rpart function and the different parameters called. The notation “churn_dataset_train$churn ~.” is shorthand for using all the variables as predictors for churn in our model.



```
> fit1 <- rpart(churn_dataset_train$churn ~., method = "class", data = churn_dataset_train,
+               parms = list(split="Gini"))
```

*Figure 18: rpart function*

The “class” method tells the function that the output variable is categorical and a classification tree should be used. As will be demonstrated, method = “anova” is used to build regression trees to create a predictor for continuous outcome variables.

In order to produce a tree which is easy to interpret visually we will let our complexity parameter = 0.01 (rpart default).  Gini is called as the splitting criterion. Rpart.plot offers a wide number of parameters which can be used to edit the appearance of the tree. In Figure 19, shadow.col has been set to azure1, a light blue colour; cex and tweak both used to edit the text size and type sets the style of the tree.

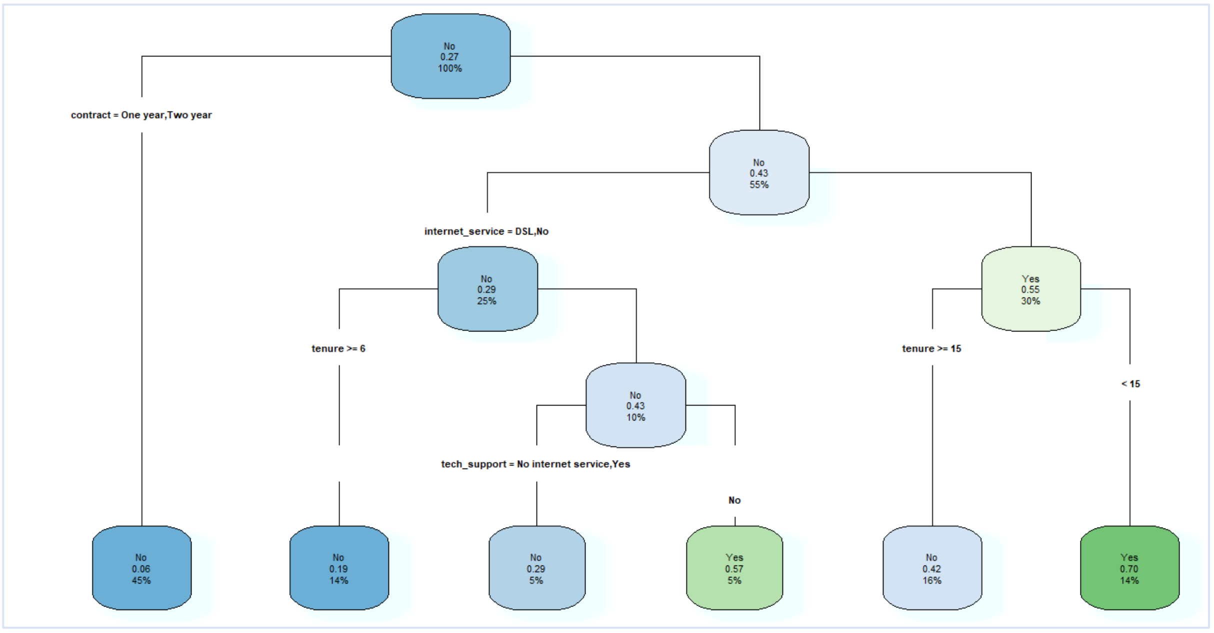


*Figure 19: Classification Tree: rpart.plot(fit,shadow.col="azure1",cex=0.6, type = 4, tweak = 1.1)*

Each node shows the predicted class (Churn yes or no), the predicted probability of churning and the percentage of observations in the node (R-bloggers, 2018). In Figure 19 there are 6 terminal nodes, two which predict churn and four that predict remain.

Figure 20 shows a portion of the summary output of the model. The summary of rpart objects can be very long as each node is analysed fully. It is clear from the output that this tree may be pruned to high as the complexity parameter (CP) and cross validation error (x error) both decrease and should this trend continue a lower CP value should produce a better model.

```
> summary(fit)
Call:
rpart(formula = churn_dataset_train$churn ~ ., data = churn_dataset_train,
    method = "class", parms = list(split = "Gini"), control = rpart.control(cp = 0.01))
  n= 5282

          CP nsplit rel error    xerror       xstd
1 0.05955777      0 1.0000000 1.0000000 0.02288983
2 0.01283880      3 0.7810271 0.8045649 0.02124421
3 0.01000000      5 0.7553495 0.7774608 0.02097861
```

*Figure 20: Output from summary(fit)*

As illustrated by the R code, the splitting parameter was changed to “Information” which rpart recognises as the Entropy splitting criterion outlined in Section 2.2. When plotted and summarised there was no difference in the output using this criterion.

**<u>Overfitting:</u>**

In order to build a maximal tree rpart was ran again this time with the parameter:

`control = rpart.control(cp = 0.001)`

This will ensure that the model returns a very large tree that will require pruning. As shown in Figure 21, the tree is no longer simple to interpret visually. Sometimes this may be the case, however this is highly unlikely. The summary function of this model is now over 1000 lines and therefore cannot be viewed in the RStudio console. As an alternative, we will call “cp$table” in order to assess where to prune the tree.

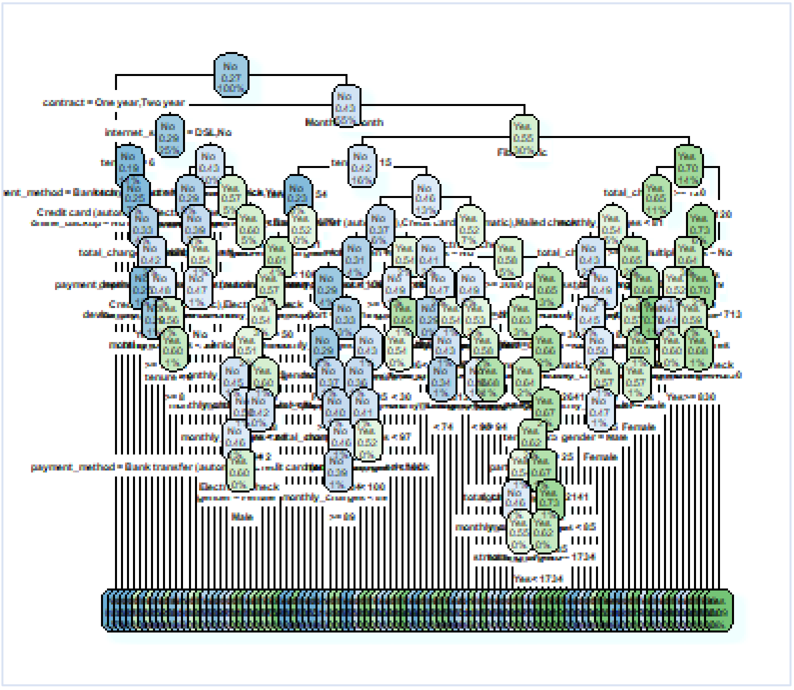
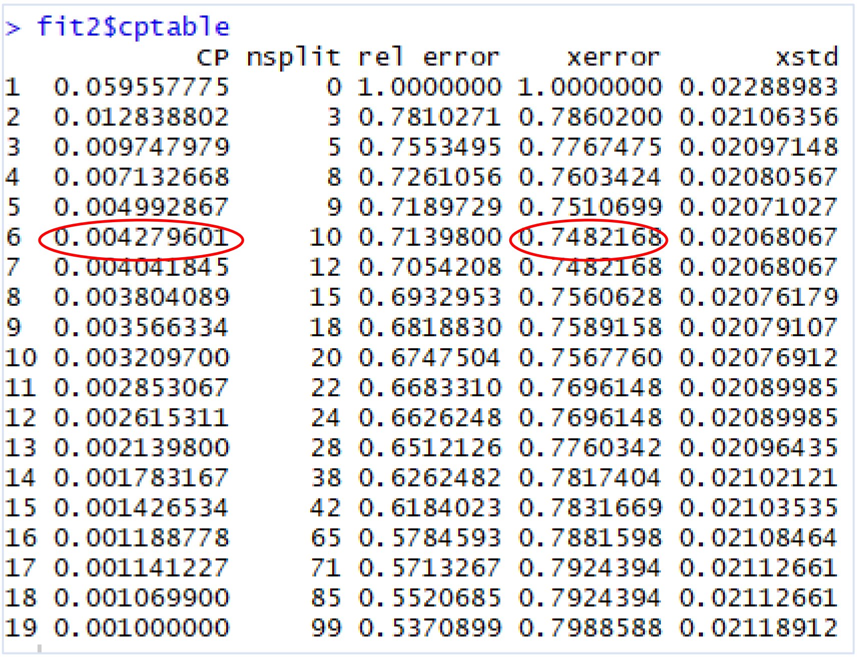


*Figure 21: CP = 0.001 - Overfitted*

By evaluating Figure 22 it is clear that the cross-validation error begins to increase at CP = 0.004279601. It is clear that this is the point at which our model begins to overfit and therefore we will select CP = 0.0043 as our final complexity parameter to build our classification tree. Figure 23 is a visualisation of the CP table which illustrates the cross-validation error at the set of possible costcomplexity pruning. The horizontal blue line is drawn 1 standard error above the minimum of the curve; a good choice of cp is typically the leftmost value which is under the horizontal line



```
> fit2$cptable
            CP nsplit rel error    xerror       xstd
1  0.059557775      0 1.0000000 1.0000000 0.02288983
2  0.012838802      3 0.7810271 0.7860200 0.02106356
3  0.009747979      5 0.7553495 0.7767475 0.02097148
4  0.007132668      8 0.7261056 0.7603424 0.02080567
5  0.004992867      9 0.7189729 0.7510699 0.02071027
6  0.004279601     10 0.7139800 0.7482168 0.02068067
7  0.004041845     12 0.7054208 0.7482168 0.02068067
8  0.003804089     15 0.6932953 0.7560628 0.02076179
9  0.003566334     18 0.6818830 0.7589158 0.02079107
10 0.003209700     20 0.6747504 0.7567760 0.02076912
11 0.002853067     22 0.6683310 0.7696148 0.02089985
12 0.002615311     24 0.6626248 0.7696148 0.02089985
13 0.002139800     28 0.6512126 0.7760342 0.02096435
14 0.001783167     38 0.6262482 0.7817404 0.02102121
15 0.001426534     42 0.6184023 0.7831669 0.02103535
16 0.001188778     65 0.5784593 0.7881598 0.02108464
17 0.001141227     71 0.5713267 0.7924394 0.02112661
18 0.001069900     85 0.5520685 0.7924394 0.02112661
19 0.001000000     99 0.5370899 0.7988588 0.02118912
```

*Figure 22: cptable for overfitted tree*

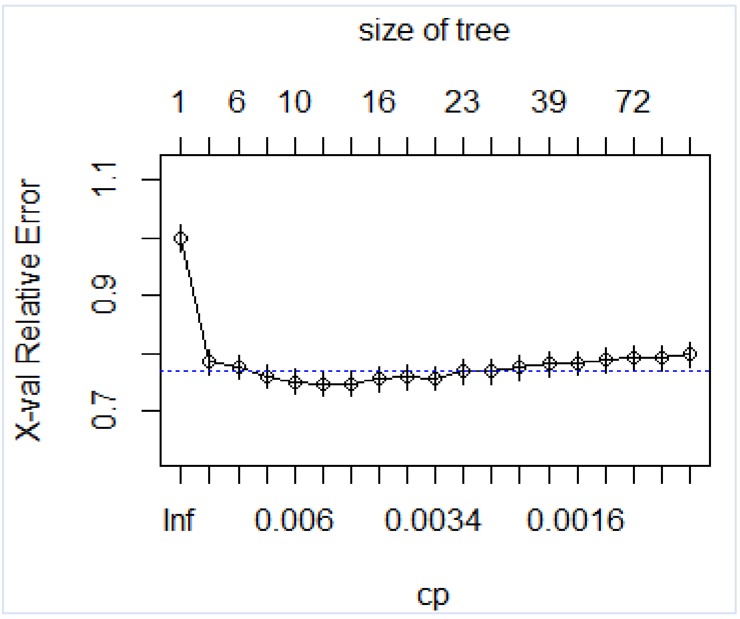


*Figure 23: plotcp(fit2, col = "blue")*

Having finalised the training model, we can now evaluate the model's performance on the test set which was created. Using the predict function, the three models are assessed against the test set to predict the outcome by passing in the model and the test set as parameters.

The prediction() function is used to create prediction objects. This function requires two arguments: firstly, the list of predictions created using the predict() function and also the list of true labels, in this case that is the churn column in the churn test set.

The Receiver Operating Characteristic (ROC) curve (Figure 24) is a quick way to measure the performance of classification models. The ROC curve is plotted with the True Positive

Rate/Sensitivity/Recall (TPR) against the False Positive Rate/1-Specificity (FPR) where TPR is on y-axis and FPR is on the x-axis (Towards Data Science, 2018).

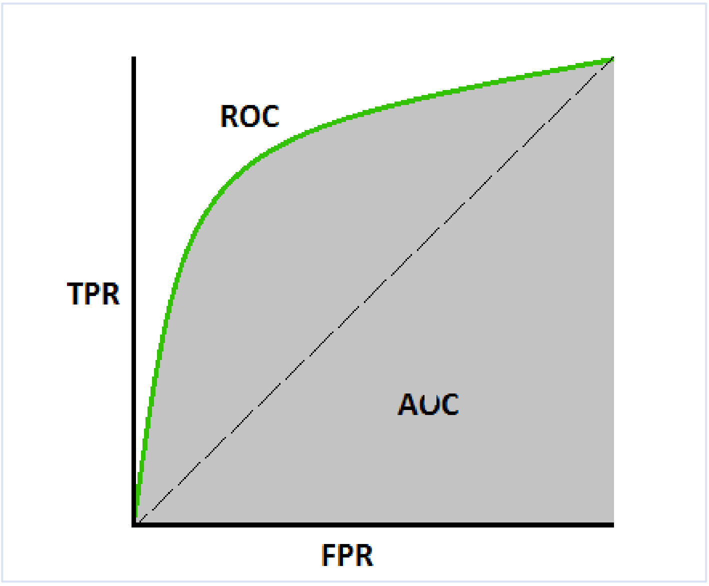


*Figure 24: Roc Curve (https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5)*

The area under the curve (AUC) tells how capable the model is at distinguishing between classes. An excellent model will have AUC near to 1. An AUC of 0.5 suggests the model is no better than random selection in determining classification. In order to compare our classification tree models, we will run a linear regression model on the churn data and also plot the performance of this model against the trees.

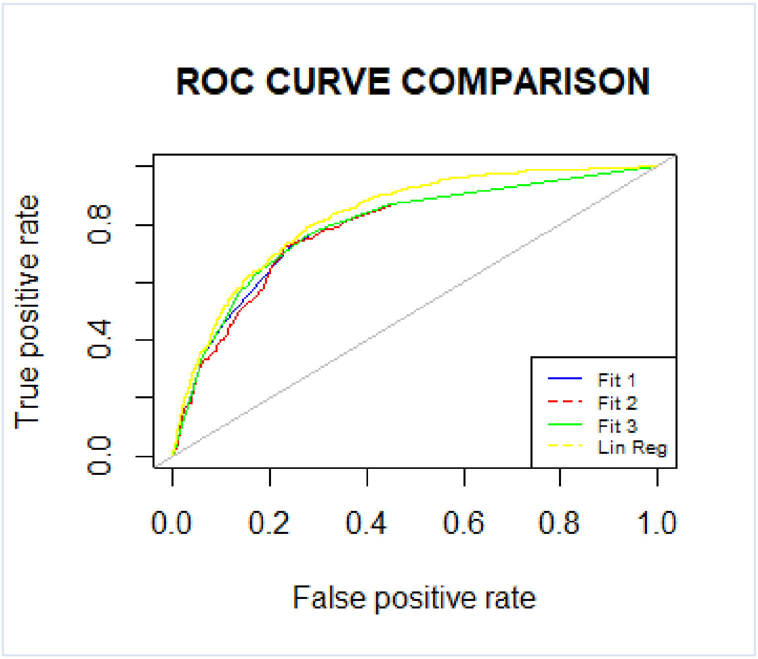


*Figure 25: ROC Curve comparison*

Figure 25 demonstrates a marginal improvement in the classification trees as the complexity parameter is tuned, however the performance is not as good as the linear regression model. To take a closer look at the model performance a Confusion Matrix may help to identify areas of weakness in the model.

The Confusion Matrix for Fit3 (Figure 26) demonstrates that the model performs very well for detecting negatives however the accuracy for detecting positives is very low (~58%). This may be caused by such a low frequency of positive churn values. Therefore, in order to improve our model, we may introduce a method for increasing the size of a minority group in a data sample, known as SMOTE.

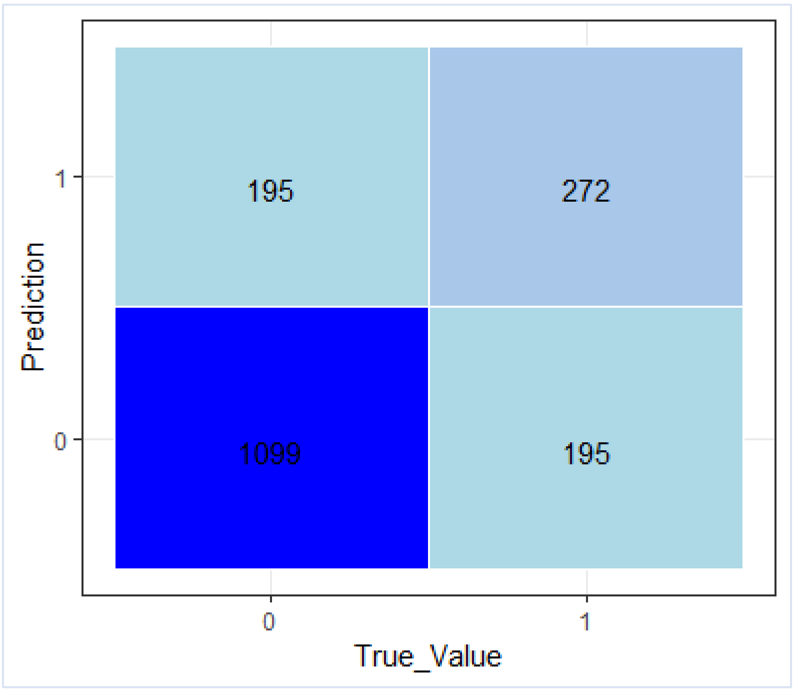


*Figure 26: Confusion Matrix for Fit 3*

### 2.4.2 SMOTE

Unbalanced classification problems cause problems to many learning algorithms. These problems are characterized by the uneven proportion of cases that are available for each class of the problem (Rdocumentation.org, 2018). Given the low frequency of Churn “Yes” values, SMOTE is an algorithm that can be used to combat this problem.

The SMOTE, or Synthetic Minority Oversampling Technique, method is to artificially generate new examples of the minority class using the nearest neighbours of these cases. The majority class examples are also under-sampled, leading to a more balanced dataset (Rdocumentation.org, 2018).

```
new_churn <- SMOTE(churn ~., churn_dataset , perc.under = 200)
```

*Figure 27: Calling SMOTE()*

Figure 27 demonstrates the initiation of the SMOTE function. The perc.under parameter controls the amount of under-sampling of the majority classes. In order to apply SMOTE, our churn dataset is reset and resampled into test and train data as before. The “SMOTEd” dataset returns a very different tree as shown in Figure 28. It is far too difficult to interpret visually however by plotting the Confusion Matrix (Figure 29) on the test set it is clear that there is a significant improvement in the prediction of true positives.

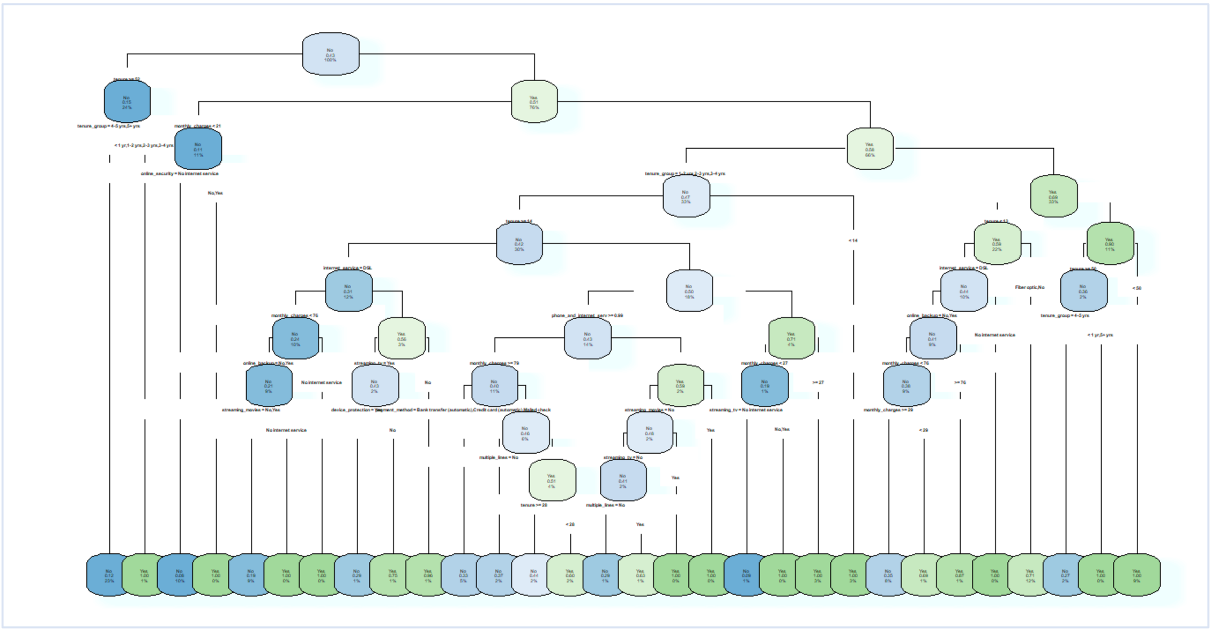


*Figure 29: SMOTE Classification Tree*

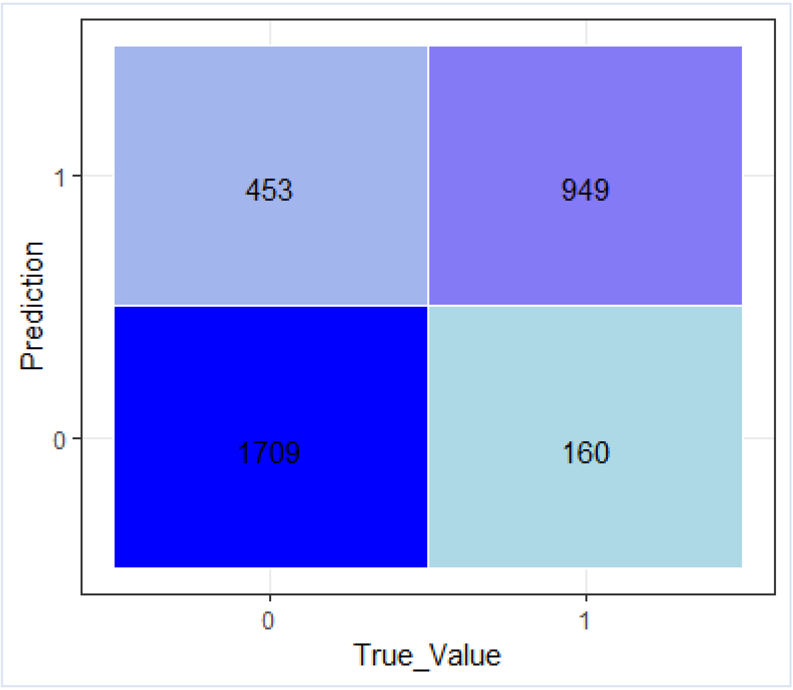
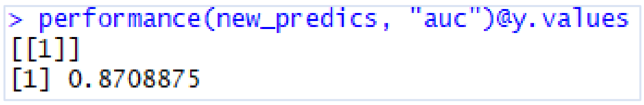


*Figure 28: SMOTE Classification Matrix*

This improvement should result in a considerable improvement in the Area Under the Curve of the model's ROC curve. Plotting the ROC curve of the SMOTEd model (Figure 31), this is the case and is by far the best model built to predict churn with an area under the curve of 87% (Figure 30). Figure 32 shows the considerable rise in Specificity, albeit at the slight detriment of Sensitivity, however given the company's supposed interest in identifying those who **are** likely to switch, this should be a welcomed trade off.



```
> performance(new_predics, "auc")@y.values
[[1]]
[1] 0.8708875
```

Figure 30: SMOTE AUC =87%

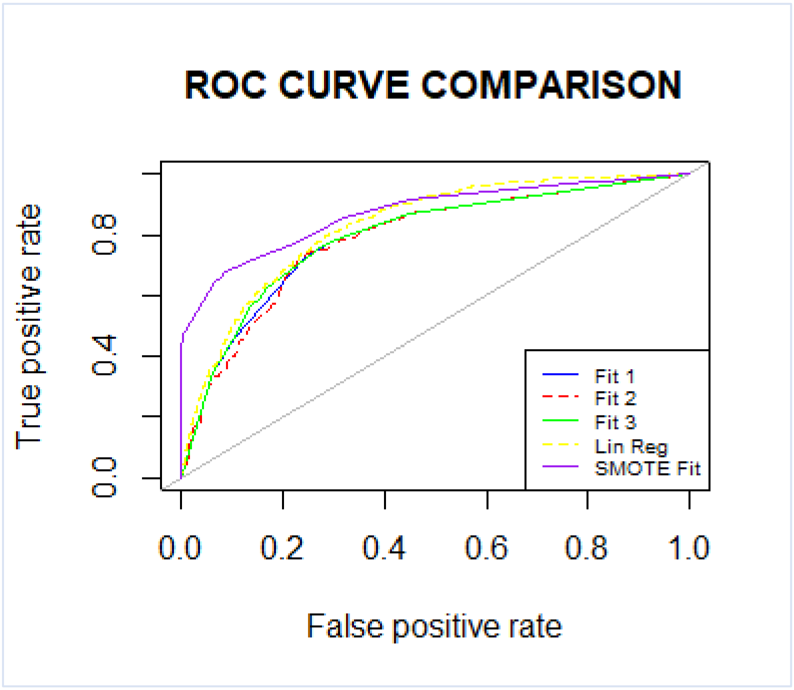
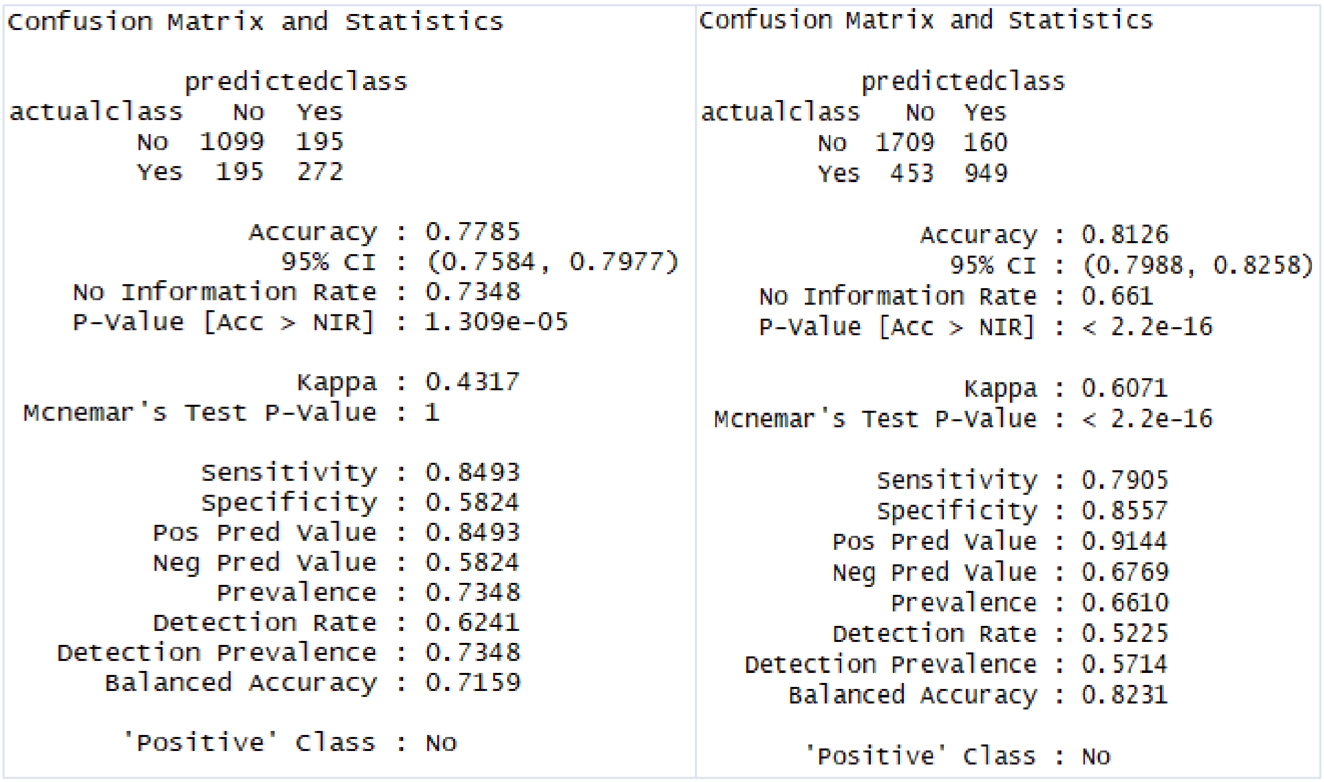


Figure 31: SMOTE ROC Curve added



```
Confusion Matrix and Statistics

           predictedclass
actualclass   No  Yes
        No  1099  195
        Yes  195  272

               Accuracy : 0.7785
                 95% CI : (0.7584, 0.7977)
    No Information Rate : 0.7348
    P-Value [Acc > NIR] : 1.309e-05

                  Kappa : 0.4317
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8493
            Specificity : 0.5824
         Pos Pred Value : 0.8493
         Neg Pred Value : 0.5824
             Prevalence : 0.7348
         Detection Rate : 0.6241
   Detection Prevalence : 0.7348
      Balanced Accuracy : 0.7159

       'Positive' Class : No
```

```
Confusion Matrix and Statistics

           predictedclass
actualclass   No  Yes
        No  1709  160
        Yes  453  949

               Accuracy : 0.8126
                 95% CI : (0.7988, 0.8258)
    No Information Rate : 0.661
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6071
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.7905
            Specificity : 0.8557
         Pos Pred Value : 0.9144
         Neg Pred Value : 0.6769
             Prevalence : 0.6610
         Detection Rate : 0.5225
   Detection Prevalence : 0.5714
      Balanced Accuracy : 0.8231

       'Positive' Class : No
```

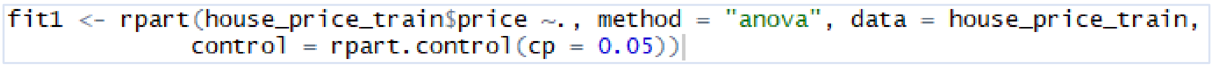
Figure 32: Confusion Matrices of Fit 3 vs SMFit (Smote)

## 2.5 Kingscounty House Price - Regression Tree

### 2.5.2 Growing and Evaluating

In order to remove numbers with scientific notation in the nodes of the trees, the price column in the Kingscounty data is divided by 1000. Therefore, all predictions are measured in thousands.

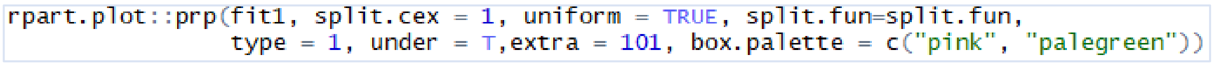
Figure 33 shows the rpart function that is called on the Kingscounty House Price training set. For continuous outcome variables, the parameter method = “anova”. This approach evaluates the Sum of Squares for the splits at each node. The model will look to maximise the Sum of Squares at each node and every incremental gain in $R^2$ is recorded as “improvement”.



```
fit1 <- rpart(house_price_train$price ~., method = "anova", data = house_price_train,
              control = rpart.control(cp = 0.05))
```

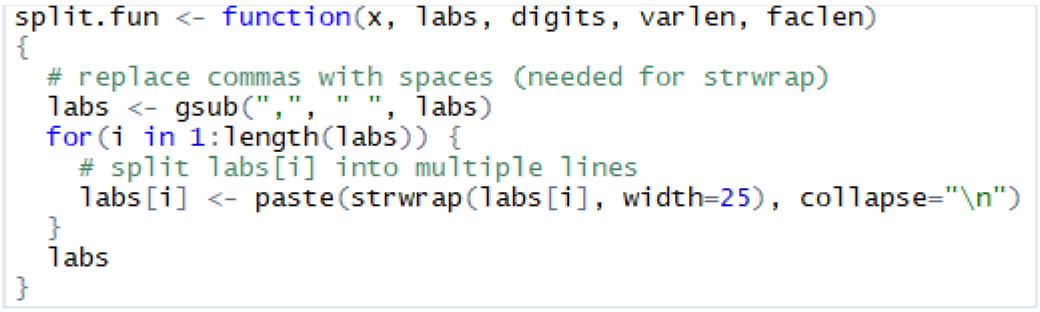
*Figure 33: rpart Anova*

Given the large number of factors in certain variables in the Kingscounty dataset (such as zipcodes), the trees require additional tuning to become visually interpretable. The split.fun (Figure 35) function is used to split labels into multiple lines to assist with the layout of the regression tree. Figure 34 shows the more complicated rpart.plot code used to graph the model. The prp function allow for greater customisation of the tree features such as tree type, node information and colour scheme used.



```
rpart.plot::prp(fit1, split.cex = 1, uniform = TRUE, split.fun=split.fun,
                type = 1, under = T,extra = 101, box.palette = c("pink", "palegreen"))
```

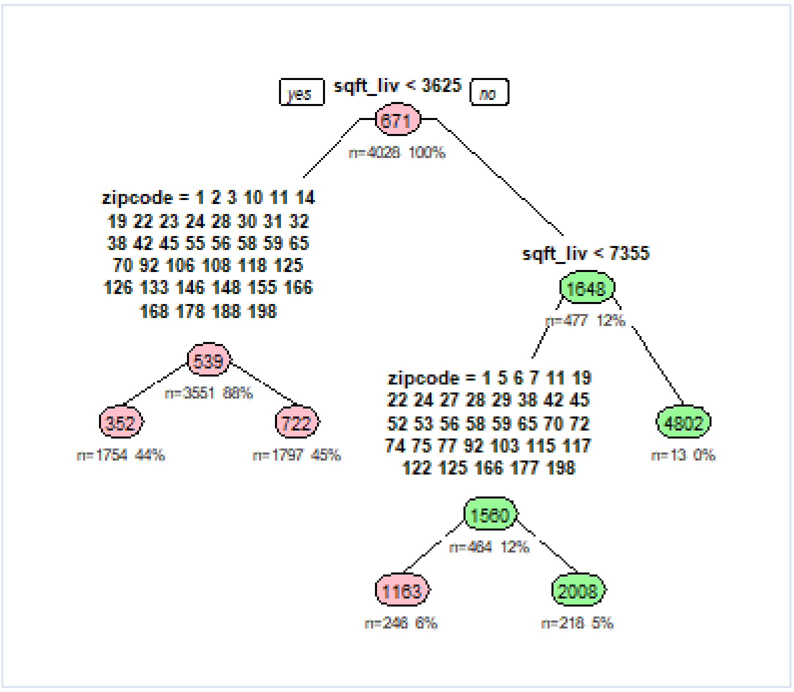
*Figure 34: Plot the regression tree*



```
split.fun <- function(x, labs, digits, varlen, faclen)
{
  # replace commas with spaces (needed for strwrap)
  labs <- gsub(",", " ", labs)
  for(i in 1:length(labs)) {
    # split labs[i] into multiple lines
    labs[i] <- paste(strwrap(labs[i], width=25), collapse="\n")
  }
  labs
}
```

*Figure 35: Custom function used to format split labels*

The regression tree plot (Figure 36) is similar to the classification tree however the layout has been tweaked to allow for greater readability. Each node shows the predicted value (in thousands), the number of observations in the node and the percentage of observations in the node.



yes
sqft_liv < 3625
no
671
n=4028 100%
zipcode = 1 2 3 10 11 14 19 22 23 24 28 30 31 32 38 42 45 55 56 58 59 65 70 92 106 108 118 125 126 133 146 148 155 166 168 178 188 198
sqft_liv < 7355
1648
n=477 12%
539
n=3551 88%
352
n=1754 44%
722
n=1797 45%
zipcode = 1 5 6 7 11 19 22 24 27 28 29 38 42 45 52 53 56 58 59 65 70 72 74 75 77 92 103 115 117 122 125 166 177 198
4802
n=13 0%
1560
n=464 12%
1163
n=246 6%
2008
n=218 5%

*Figure 36: Regression Tree plot*

As with the Churn tree, the complexity parameter will be decreased significantly, and the model regrown to build the maximal tree. As expected, the CP table for this model is very large, which indicates a great degree of splitting. Again, the tree is pruned at the point of inflection of the cross-validation error (Figure 37).

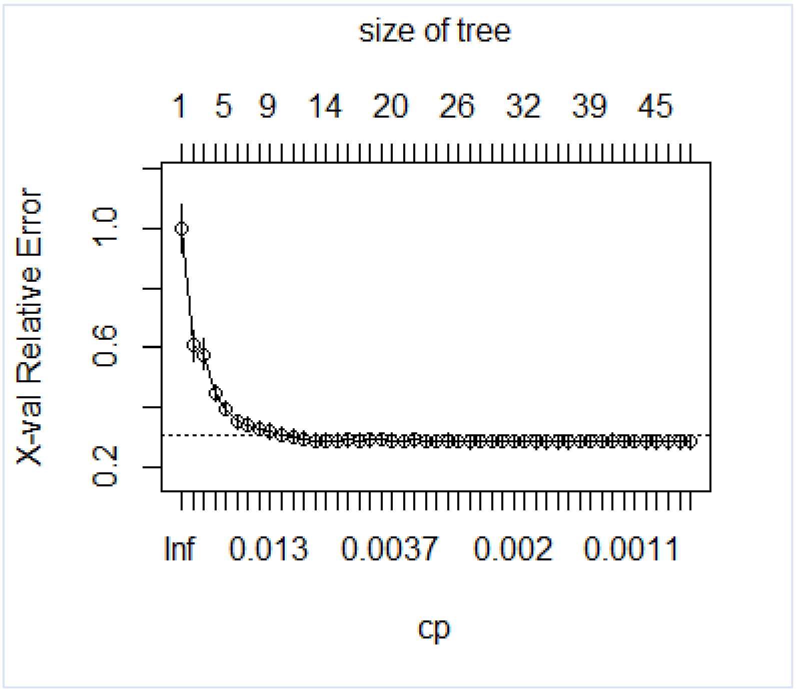


*Figure 37: plotcp(Fit2)*

Following pruning the tree, we can evaluate the model by using the rsq.rpart function. The function plots (Figure 38) the approximate R-Square for the different splits. It is clear that the first few splits capture the greatest increase in R-Square and decrease in cross validation error. These results suggest that a tree with five splits would capture the majority of the value.

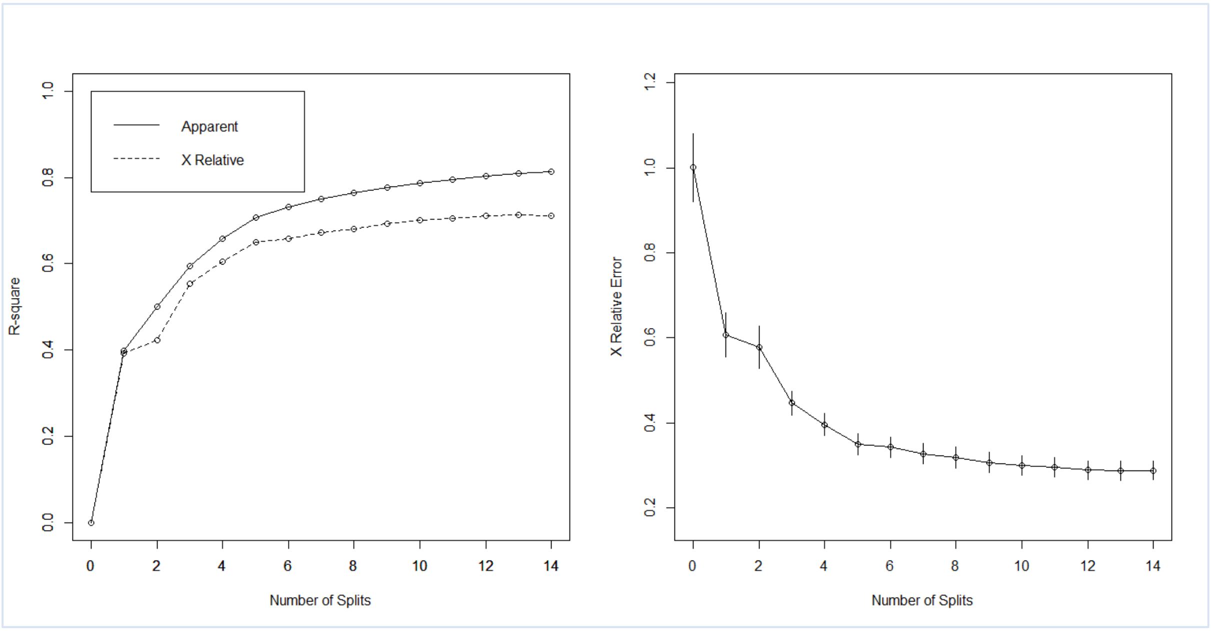
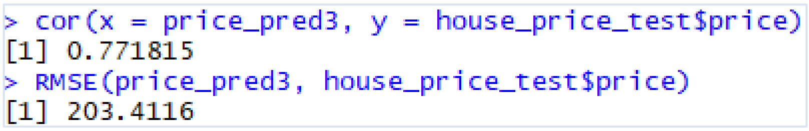


*Figure 38: rsq.rpart(fit3)*

Similar to the churn dataset, we can now evaluate the performance of the model on the test set using the predict function, however we must use the method = "anova" parameter to return a vector of predicted prices. There are several ways to assess the performance of our regression model. The correlation and root mean square error are a good assessment (Figure 39). These results



```
> cor(x = price_pred3, y = house_price_test$price)
[1] 0.771815
> RMSE(price_pred3, house_price_test$price)
[1] 203.4116
```

*Figure 39*

suggest a strong positive correlation between the predictions and the actual house prices. The RMSE is best assessed in comparison to another model. Using inferences from the rsq.rpart output, we will develop another model using just five splits (Figure 40) and compare the results. Figure 41 demonstrates that this model has both lower correlation between predictions and actual as well as a greater RMSE. This serves to further validate our original model - Fit 3.

We can isolate the residuals of the model by subtracting the real house price from the predictions:
`residuals = (price_pred3 - house_price_test$price)`

Plotting these residuals (Figure 42) demonstrates that the model tends to undervalue the house price and there are considerable outliers.

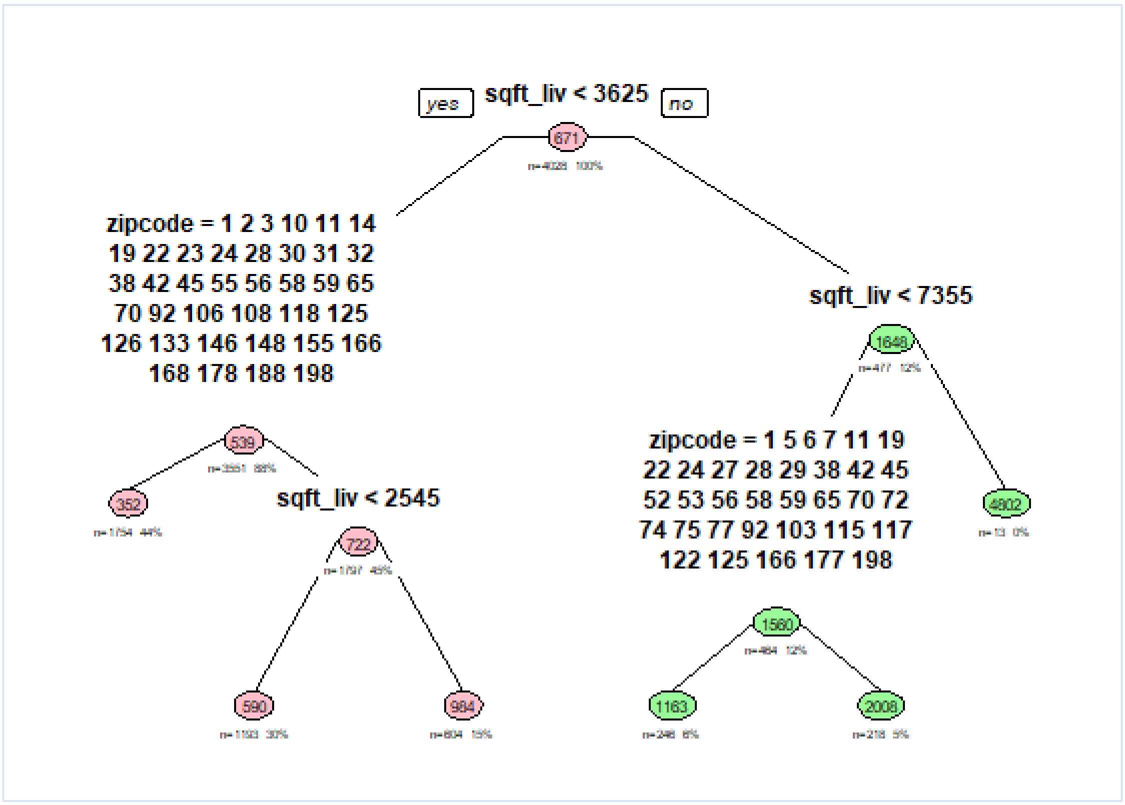
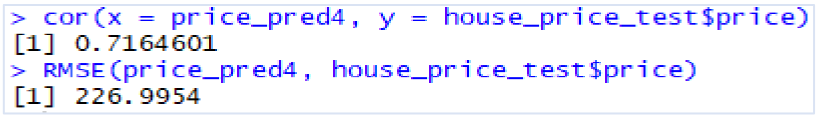

yes
sqft_liv < 3625
no
671
n=4028 100%
zipcode = 1 2 3 10 11 14 19 22 23 24 28 30 31 32 38 42 45 55 56 58 59 65 70 92 106 108 118 125 126 133 146 148 155 166 168 178 188 198
539
n=3551 88%
352
n=1754 44%
sqft_liv < 2545
722
n=1797 45%
590
n=1193 30%
984
n=604 15%
sqft_liv < 7355
1648
n=477 12%
zipcode = 1 5 6 7 11 19 22 24 27 28 29 38 42 45 52 53 56 58 59 65 70 72 74 75 77 92 103 115 117 122 125 166 177 198
1560
n=464 12%
1163
n=246 6%
2008
n=218 5%
4802
n=13 0%

Figure 40: Fit 4 - five splits



```
> cor(x = price_pred4, y = house_price_test$price)
[1] 0.7164601
> RMSE(price_pred4, house_price_test$price)
[1] 226.9954
```

Figure 41: Fit 4 - correlation & RMSE

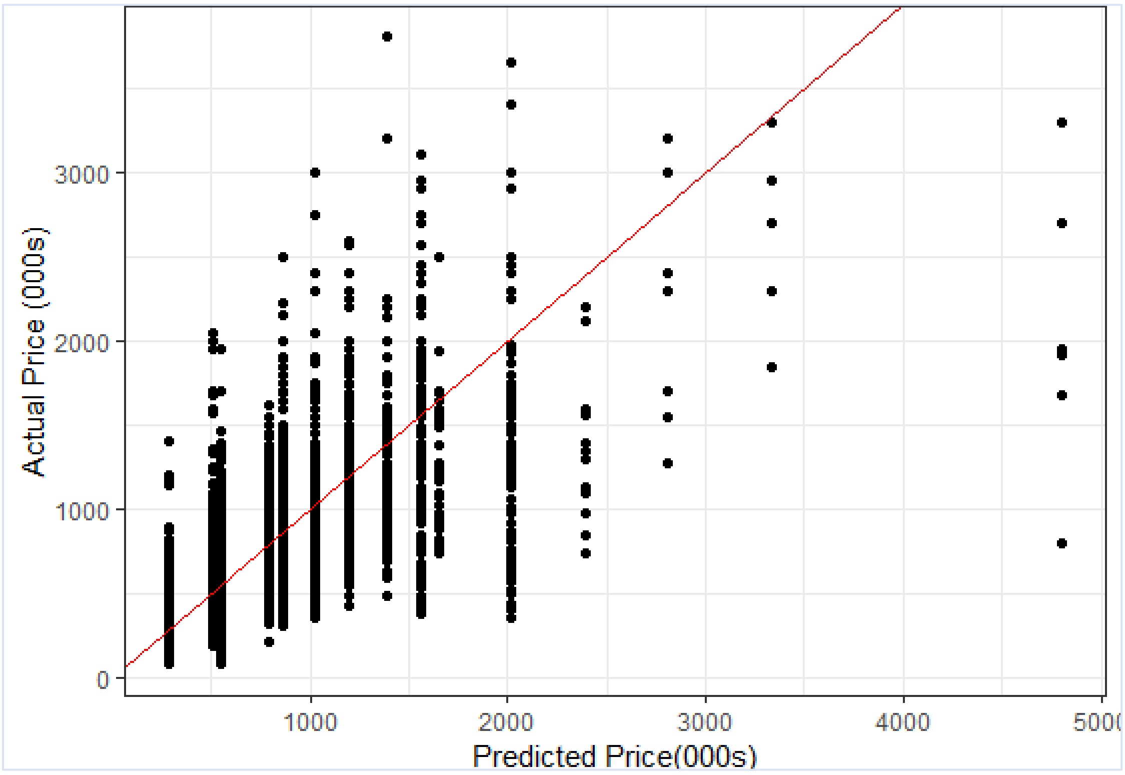


Figure 42: Plot of Actual vs Predicted

## 3. Trees: Other Approach - CHAID

CART Trees are the most popular decision trees used today however there are a number of interesting alternatives. Another useful tool for your analytical “toolbox” is CHAID. CHAID, or Chi-square Automatic Interaction Detection, is a robust statistical method that is particularly good at explaining how a model arrives at its prediction or classification.

The technique was developed in 1980 by Gordon Kass and is based on the Chi-square test and is a strong exploratory technique when investigating large quantities of categorical data. One limitation of note is that CHAID can only accept nominal or ordinal categorical predictors.

In order to implement the CHAID algorithm on our Churn dataset, we must manipulate the data. As outlined in section 2.4.2, we will make use of the SMOTE algorithm to expand the minority case. The numeric factors must be assessed to determine if they can be categorised. There are three numeric categories in the Churn dataset: tenure, monthly charges and total charges. In Section 1.3, we created a derived categorical variable tenure_group, which will capture the data from tenure however we do not want to lose the information from the charges variable as these variables had high levels of importance in our CART model.

```
churn_dataset <- within(churn_dataset, monthlycharge_quartile <- as.integer(cut(monthly_charges,
                                                          quantile(monthly_charges, probs=0:4/4),
                                                          include.lowest=TRUE)))
churn_dataset$monthlycharge_quartile = as.factor(churn_dataset$monthlycharge_quartile)
```

*Figure 43: Monthly charge quantiles*

Figure 43 demonstrates how we can capture the information in the numeric monthly_charges variable in the categorical variable monthlycharge_quantile. We can now proceed to develop the CHAID model for our churn dataset with only factor variables, fact_churn (Figure 44).

```
chaidattrit1 <- chaid(churn ~ . , data = fact_churn)
plot(chaidattrit1, type = "simple", gp = gpar(fontsize = 4, theme = "blue",
                                              lty = "solid",
                                              lwd = 1))
```

*Figure 44*

The plot of this CHAID tree is impossible to interpret (Figure 45), and therefore requires some tuning. Chaid_control (not surprisingly) controls the behaviour of the model building (Chuck Powell, 2018). We set the number of observations in a split at which no response is further desired to 200 using the minsplit parameter and the minimum frequency of observations in valid terminal nodes to

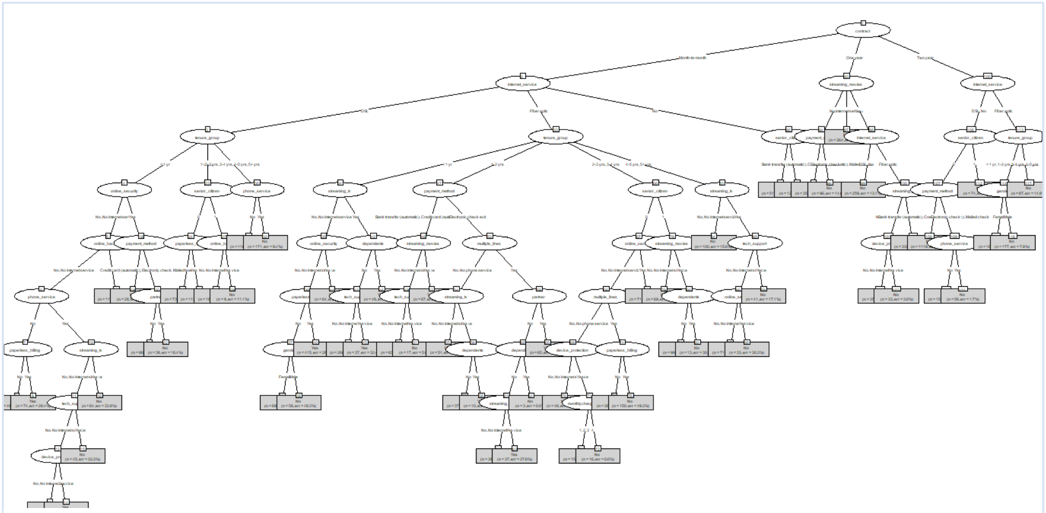


*Figure 45: First CHAID Tree*

5%. Visualising this tree (Figure 46), there are less nodes and the probability buckets are now visible, however there is still too much going on and requires further refinement. Setting the maxheight = 3, we now can assess the CHAID tree visually (Figure 47).

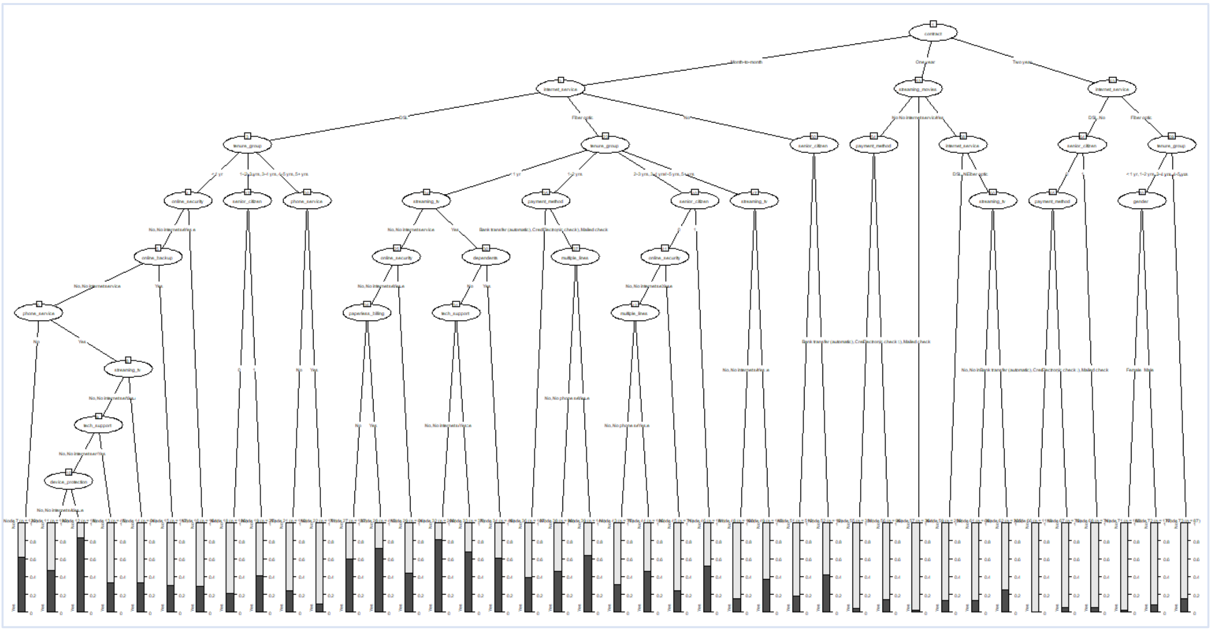


Figure 46: CHAID tree with control parameter

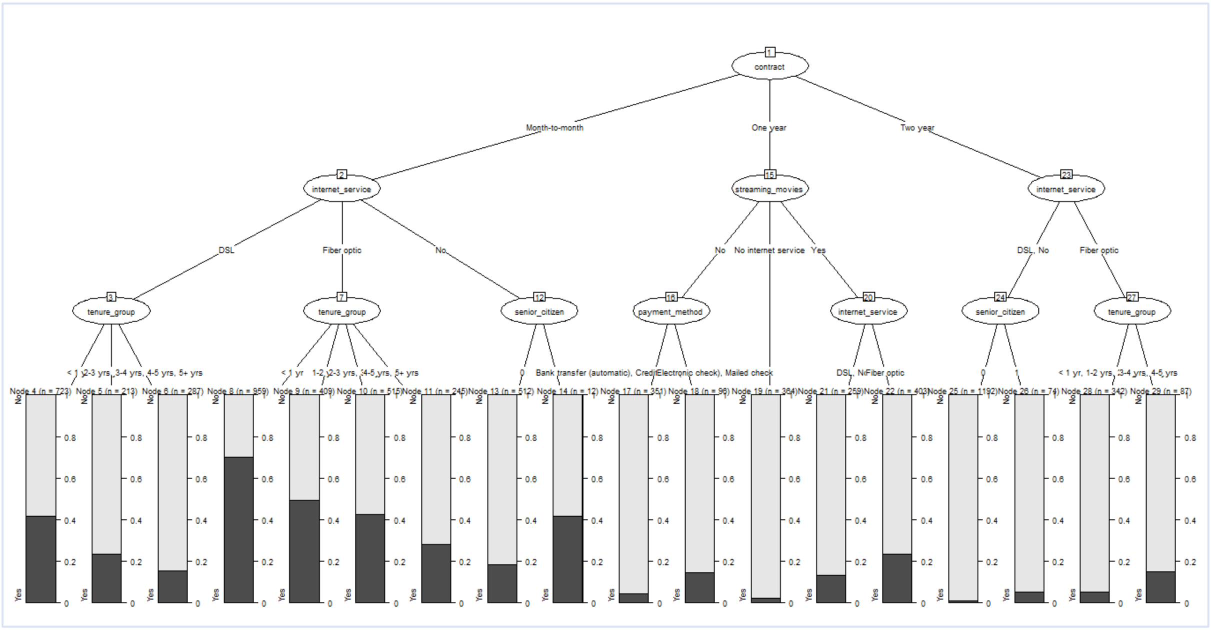


Figure 47: CHAID tree - Max height = 3

Each terminal node indicates how many observations there are and the predicted probabilities for observations in that node. To evaluate these models, we will use the predict function again and generate a Confusion Matrix for each (Figure 48). It is clear that in setting the max height so shallow, as in CHAID model 3, a greater degree of error is introduced. All three of the models perform very poorly when determining churns.

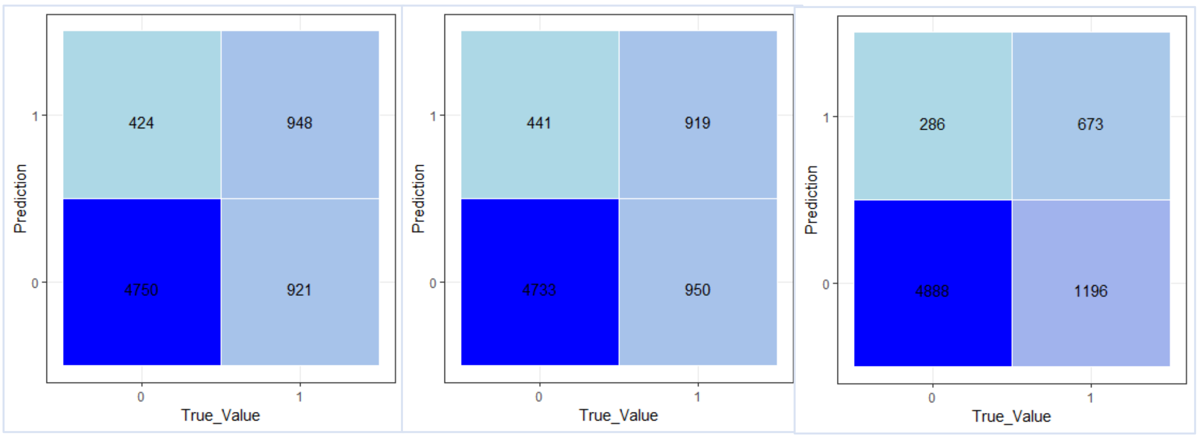
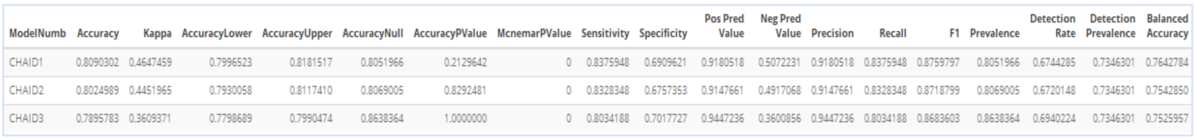


*Figure 48: Confusion Matrices for CHAID models 1, 2, 3*

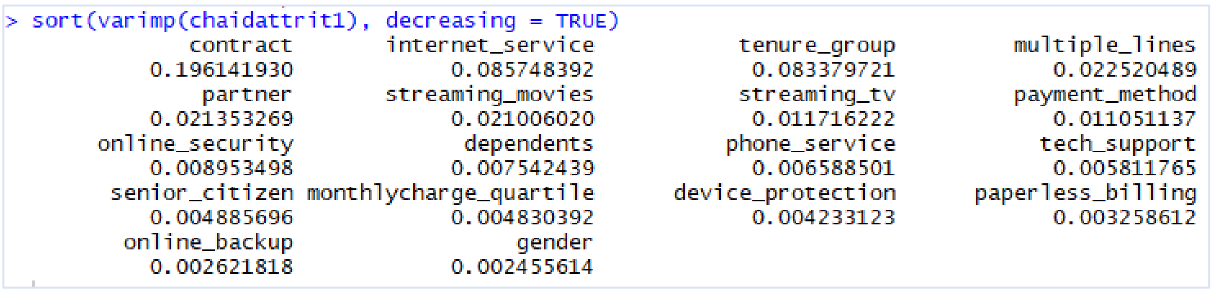


| ModelNumb | Accuracy | Kappa | AccuracyLower | AccuracyUpper | AccuracyNull | AccuracyPValue | McnemarPValue | Sensitivity | Specificity | Pos Pred Value | Neg Pred Value | Precision | Recall | F1 | Prevalence | Detection Rate | Detection Prevalence | Balanced Accuracy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CHAID1 | 0.8090302 | 0.4647459 | 0.7996523 | 0.8181517 | 0.8051966 | 0.2129642 | 0 | 0.8375948 | 0.6909621 | 0.9180518 | 0.5072231 | 0.9180518 | 0.8375948 | 0.8759797 | 0.8051966 | 0.6744285 | 0.7346301 | 0.7642784 |
| CHAID2 | 0.8024989 | 0.4451965 | 0.7930058 | 0.8117410 | 0.8069005 | 0.8292481 | 0 | 0.8328348 | 0.6757353 | 0.9147661 | 0.4917068 | 0.9147661 | 0.8328348 | 0.8718799 | 0.8069005 | 0.6720148 | 0.7346301 | 0.7542850 |
| CHAID3 | 0.7895783 | 0.3609371 | 0.7798689 | 0.7990474 | 0.8638364 | 1.0000000 | 0 | 0.8034188 | 0.7017727 | 0.9447236 | 0.3600856 | 0.9447236 | 0.8034188 | 0.8683603 | 0.8638364 | 0.6940224 | 0.7346301 | 0.7525957 |

*Figure 49: Kable table*

The Kable function provides a very easy to interpret table to compare models on for numerous metrics. The evidence from the table suggests that CHAID1 and CHAID2 are very similar models as their metrics are very close and follow similar trends. CHAID3 is less accurate on most metrics however does rank highest in Specificity. This demonstrates the trade-off nature of bias-variance that must be considered when building models.

This method for tree building is not as accurate as the CART algorithm, however it is able to identify important variables and display that information in an easy to interpret format (Figure 50). Just as in our CART trees, the variable contract was the most important and customers who were locked into a contract had a very low likelihood of churning.



```
> sort(varimp(chaidattrit1), decreasing = TRUE)
           contract        internet_service             tenure_group          multiple_lines
        0.196141930             0.085748392              0.083379721             0.022520489
            partner         streaming_movies             streaming_tv          payment_method
        0.021353269             0.021006020              0.011716222             0.011051137
    online_security               dependents            phone_service            tech_support
        0.008953498             0.007542439              0.006588501             0.005811765
     senior_citizen monthlycharge_quartile        device_protection       paperless_billing
        0.004885696             0.004830392              0.004233123             0.003258612
      online_backup                   gender
        0.002621818             0.002455614
```

*Figure 50:CHAID variable importance*

# 4. Random Forests

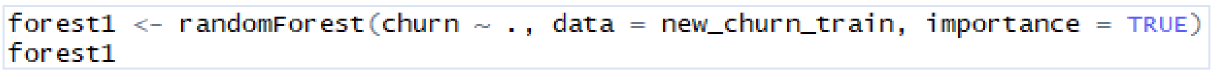
## 4.1 Introduction

Ensembling is a “type of supervised learning technique where multiple models are trained on a training dataset and their individual outputs are combined by some rule to derive the final output.” (Analytics, 2018). Random Forest is a very powerful ensembling algorithm which works by creating multiple decision trees and then combining the output generated by each of the trees. Random Forest works on the same principle as Decision Tress; however, it does not select all the data points and variables in each of the trees. It randomly samples data points and variables in each of the tree that it creates and then combines the output at the end. It removes the bias that a decision tree model might introduce in the system. Also, it can significantly improve the predictive power (Analytics, 2018).

## 4.2 Churn - Random Forest

Again, it is important to split the data into a training and testing set. We will use the SMOTE churn dataset to ensure an adequate representation of the minority class. Figure 51 illustrates the basic build for the first random forest. In this build we allow the parameters to be set to default. We can tune the model by changing the number of trees (ntree) and the number of random variables sampled at each stage (mtry).

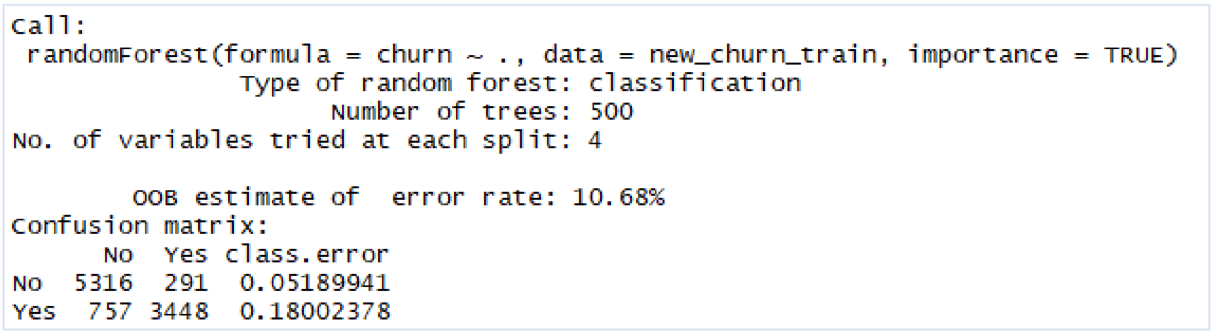
- Ntree: Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.
- Mtry: Number of variables randomly sampled as candidates at each split. Note that the default values are different for classification (sqrt(p) where p is number of variables in x) and regression (p/3)



```
forest1 <- randomForest(churn ~ ., data = new_churn_train, importance = TRUE)
forest1
```

*Figure 51:Random Forest with default parameters*

The output for the Random Forest output is very well laid out and easy to interpret. Figure 52 shows the output which features a summary of the model ran, the number of trees and the number of variables tried at each split. Also included in the output is the OOB estimate of error rate and the confusion matrix. The OOB estimate of error rate stands for the Out of Bagging estimate of error rate and denotes the generalisation error of the classifier.



```
Call:
 randomForest(formula = churn ~ ., data = new_churn_train, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 10.68%
Confusion matrix:
      No  Yes class.error
No  5316  291  0.05189941
Yes  757 3448  0.18002378
```

*Figure 52: Random Forest R output*

The Confusion matrix in Figure 52 shows that the model performs relatively well, with classification error of 5.2% on No Churn and 18% on Yes Churn.

We now rerun the randomForest algorithm tuning the default mtry and ntrees parameters to improve the model's performance (Figure 53). However, as illustrated by the OOB error rate and the Confusion matrix tuning the parameters mtry = 3 does not improve the model's performance. Using

```
Call:
 randomForest(formula = churn ~ ., data = new_churn_train, ntree = 500,      mtry = 3, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 11.25%
Confusion matrix:
      No  Yes class.error
No  5301  306  0.05457464
Yes  798 3407  0.18977408
```

*Figure 54: Random Forest with tuned parameters*

a for loop, we can cycle through a number of values for the number of variables tried at each split to assess the best value to tune mtry. (Figure 54)

*Figure 53: Looping through values for mtry*

By plotting the accuracy predictions of each model, it is clear that the greatest accuracy is seen between 3 and 4 variables tried at each split (Figure 55). Therefore, our first random forest model should perform best on future data.

By overlaying the ROC curve from this model onto the plot from the decision trees in section 2.4.2, we can see that the Random Forest model is our best performing model on the test set so far with a AUC of 96%. This is a great improvement from our original decision tree models which could not

```
for (i in 3:8) {
  model3 <- randomForest(churn ~ ., data = new_churn_train, ntree = 500, mtry = i, importance = TRUE)
  predValid <- predict(model3, new_churn_test, type = "class")
  a[i-2] = mean(predValid == new_churn_test$churn)
}
```

perform better than a standard linear regression model!

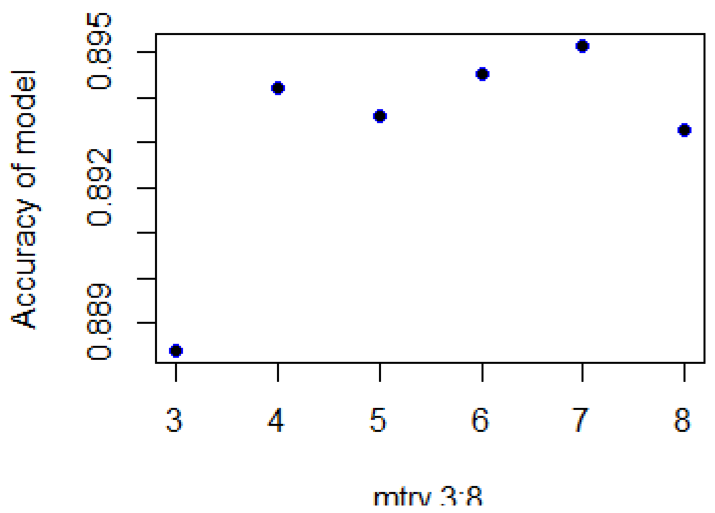


*Figure 55: mtry accuracy comparison plot*

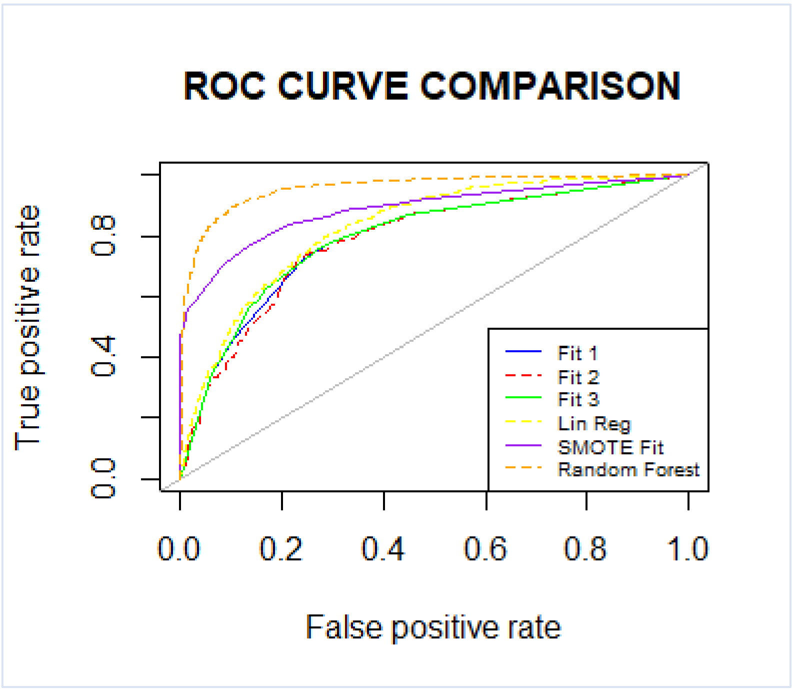


*Figure 56: RF ROC Curve added*

Given that Random Forest tests variables on many occasions in many different tree constructions, the algorithm can be excellent at assessing the importance of variables on the target variable. Figure 57 uses the varImpPlot function which works only on Random Forest objects to map variable importance.

It is clear that monthly charges and tenure have the greatest importance in determining the likelihood of a customer to churn. This is also consistent with the results of the CART and CHAID trees.

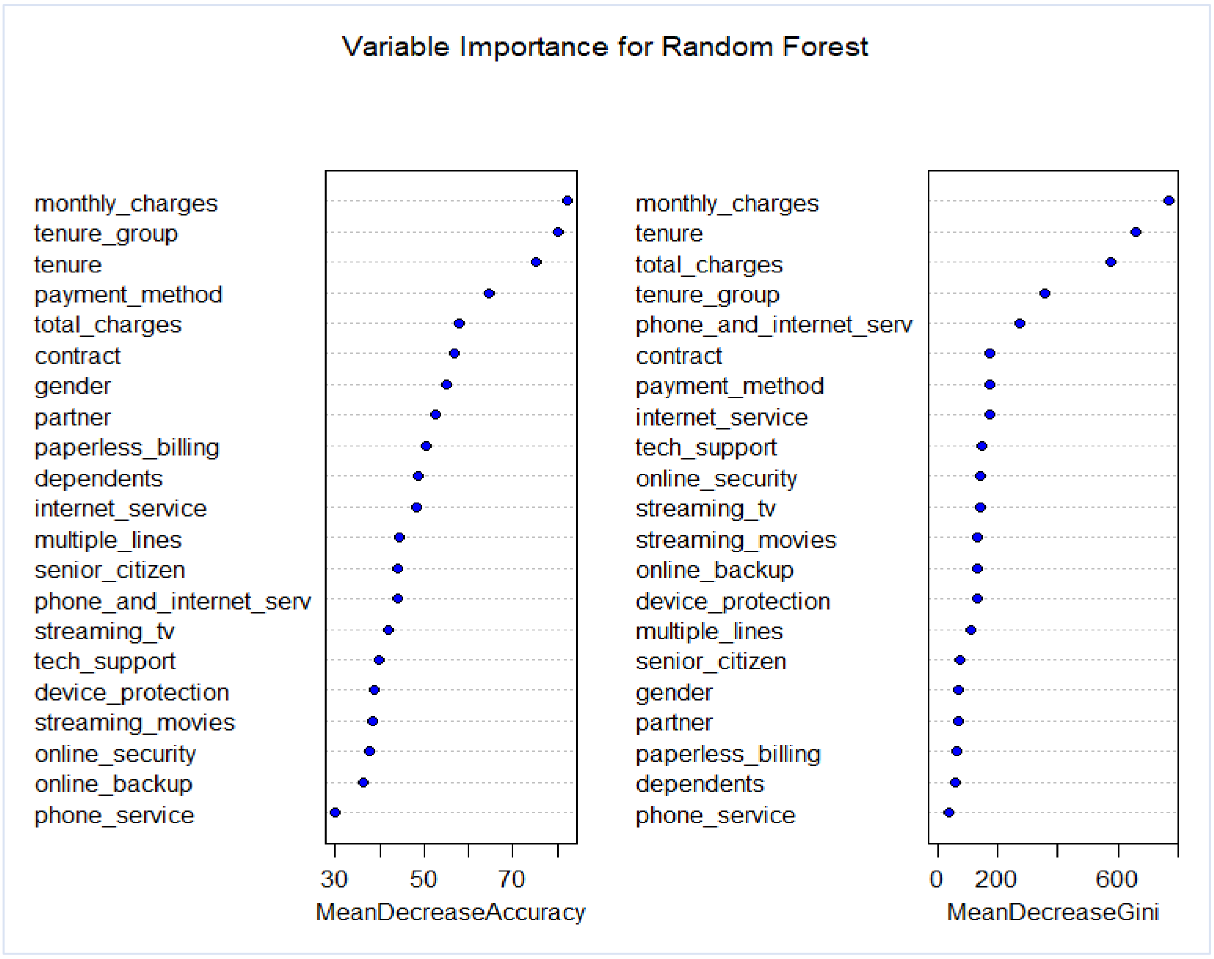
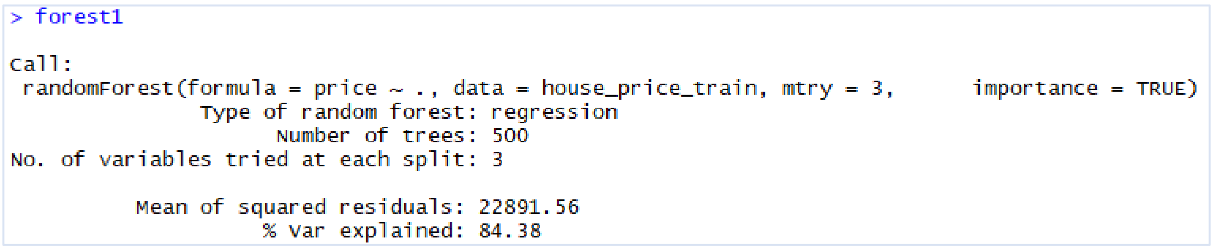


*Figure 57: varImpPlot()*

## 4.3 Kingscounty House Price - Random Forest

One limitation of the Random Forest algorithm is that it cannot manage categorical variables with more than 53 categories. This gave rise to a number of errors when calling the function on the Kingscounty data set. As a result, it was necessary to declare the yr_built and zipcode variables as numeric variables.

The R output for the Random Forest on a continuous variable appears quite different to the previous classification example. Figure 58 demonstrates the output which highlights the model implemented as well as the same tuning parameters used for classification. It is clear that the mean squared



```
> forest1

Call:
 randomForest(formula = price ~ ., data = house_price_train, mtry = 3,      importance = TRUE) 
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 22891.56
                    % Var explained: 84.38
```

*Figure 58: RandomForest on House Price Data*

residuals have a very similar value as the value derived from the CART tree and the high level of variance explained demonstrates that this is a strong model. As shown in the Churn data example, Random Forests are excellent predictors of variable importance. Figure 59 shows a ranking of the most important variables in the Kings County house price data. %IncMSE denotes the estimated increase in error in the absence of that variable. It is clear that location plays a major role in determining the price as well as the year the house was built and the square footage. The date of sale and any renovation information do not seem to play a major role at all.

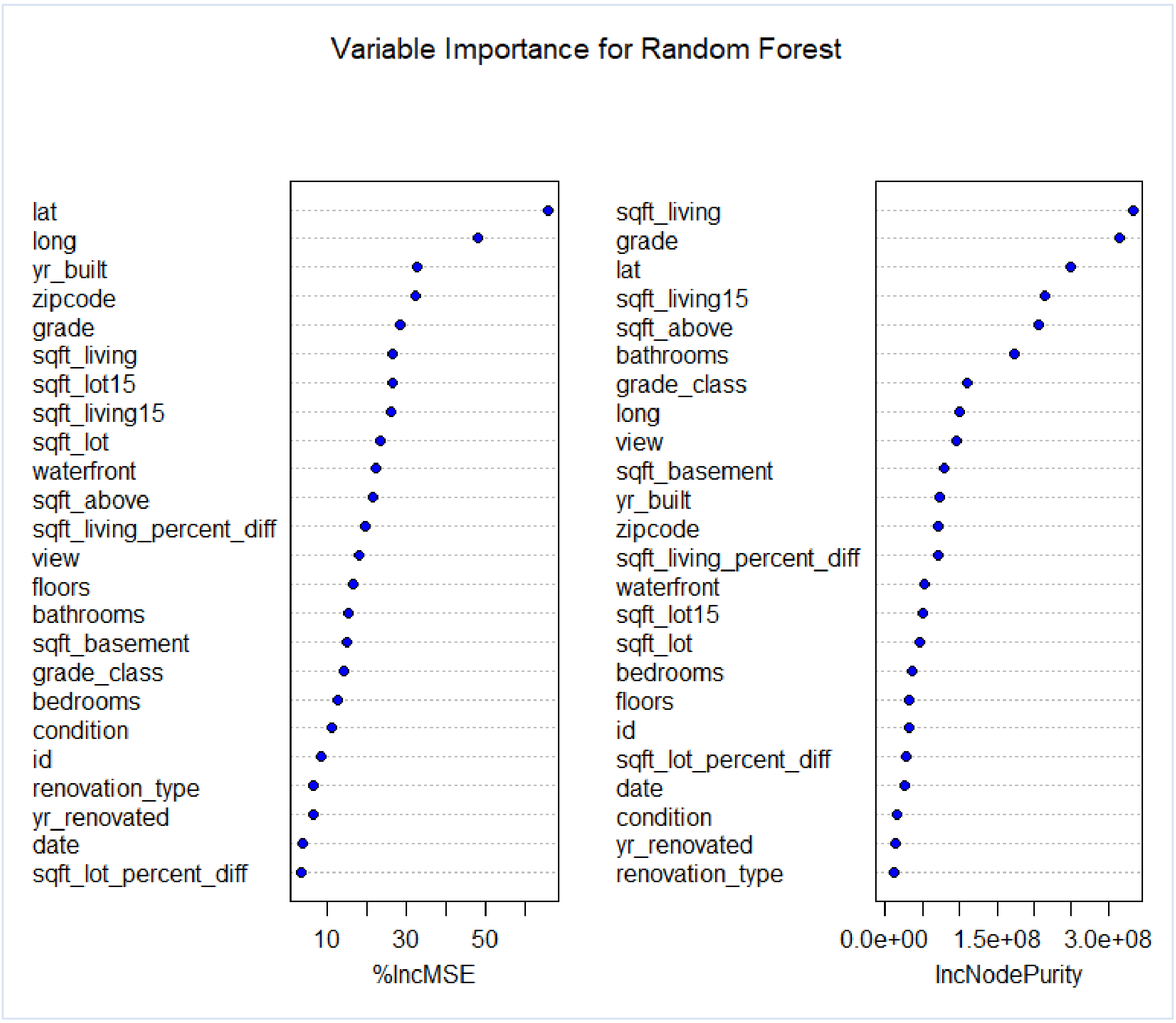
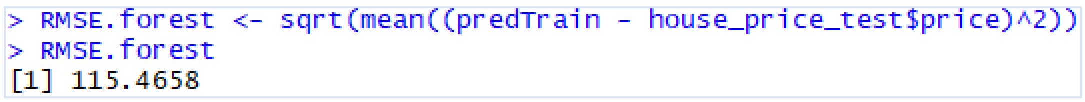


*Figure 59: Variable Importance*

As before, it is important to predict and evaluate on the test set. Figure 60 indicates the RMSE of the random forest model and as expected there is a significant improvement when compared to the results in Section 2.5.2.



```
> RMSE.forest <- sqrt(mean((predTrain - house_price_test$price)^2))
> RMSE.forest
[1] 115.4658
```

*Figure 60: RMSE of Random Forest*

## References

Analytics, P. (2018). *How to implement Random Forests in R*. [online] R-bloggers. Available at: https://www.r-bloggers.com/how-to-implement-random-forests-in-r/ [Accessed 2 Dec. 2018].

Auguié, B. (2018). *Laying out multiple plots on a page*. [online] Cran.r-project.org. Available at: https://cran.r-project.org/web/packages/egg/vignettes/Ecosystem.html [Accessed 2 Dec. 2018].

Chuck Powell. (2018). *CHAID and R – When you need explanation – May 15, 2018*. [online] Available at:https://ibecav.github.io/chaidtutor1/?fbclid=IwAR0l8kobvPjNnK2CFUSbp5WcV4Ak_8EdEqz0TNuXkPU4OQ4lpflyhsqQj74 [Accessed 2 Dec. 2018].

R-bloggers. (2018). *Part 4a: Modelling – predicting the amount of rain*. [online] Available at: https://www.r-bloggers.com/part-4a-modelling-predicting-the-amount-of-rain/ [Accessed 2 Dec. 2018].

Rdocumentation.org. (2018). *SMOTE function | R Documentation*. [online] Available at: https://www.rdocumentation.org/packages/DMwR/versions/0.4.1/topics/SMOTE [Accessed 2 Dec. 2018].

Towards Data Science. (2018). *Understanding AUC - ROC Curve – Towards Data Science*. [online] Available at: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 [Accessed 2 Dec. 2018].

## RCode

```
## ST4003 - Data Analytics: Assignment

## Ross Finnegan, 15320532 31/10/2018


##############

#Library - Need to arrange alphabetically

#################

install.packages("VIM")

install.packages("janitor")

install.packages("rbokeh")

install.packages("ggplot2")

install.packages("rpart.utils")

install.packages("caret")

install.packages(("prp"))

install.packages("rattle")

install.packages("ROCR")

install.packages("pROC")

install.packages("gplots")

install.packages("gridExtra")

install.packages("caTools")

install.packages("naniar")

install.packages(c("maps", "mapdata"))

install.packages("DMwR")

install.packages("randomForest")



library(readxl)

library(rbokeh)

library(randomForest)

library(caTools)

library(RColorBrewer)
library(pROC)
library(gridExtra)
library(VIM)
library(janitor)
library(dplyr)
library(tidyr)
library(ggplot2)
library(ROCR)
library(caret)
library(visdat)
library(rattle)
library(rpart)
library(maps)
library(mapdata)
library(DMwR)

#Set working directory - please change as required
setwd("C:/Users/Ross/Documents/MSISS/Fourth Year/ST4003 - Data Analytics/Assignment")
###############
## PRELIMINARY ANALYSIS
################

#Reading in the datasets
file_one <- "churn .xlsx"
file_two <- "Kingscounty house data.xlsx"

#churn_dataset: Binary Outcome - Churn Yes/No
#house_price: Continous Outcome - Price
churn_dataset <- read_excel(file_one)
house_price_dataset <- read_excel(file_two)
```

```
#Types of Attributes - lots of character types. Redefine as factors
#Remove customer id
sapply(churn_dataset, class)
factor_cols <- c(2,3,4,5,7,8,9,10,11,12,13,14,15,16,17,18,21)
churn_dataset[factor_cols] <- lapply(churn_dataset[factor_cols], factor)
#Remove unique ID
churn_dataset <- churn_dataset[,-1]



sapply(house_price_dataset, class)
#Zipcode shortened as every code has the same first two digits. This will make for easier analysis
house_price_dataset$zipcode <- house_price_dataset$zipcode%%1000
factor_cols <- c(4,5,8,9,10,11,12,15,17)
house_price_dataset[factor_cols] <- lapply(house_price_dataset[factor_cols], factor)
house_price_dataset$date <- as.Date(house_price_dataset$date, "%Y%m%dT000000" )

# Check for Duplicates - None & Clean data names
table(duplicated(churn_dataset))
table(duplicated(house_price_dataset))
churn_dataset <- clean_names(churn_dataset)
house_price_dataset <- clean_names(house_price_dataset)

#Missing data
#Summary gives general overview
summary(churn_dataset)
summary(house_price_dataset)

#Aggr calculates the amount of missing/imputed values - only 11 missing values in churn$total
charges
vis_dat(churn_dataset)
vis_miss(churn_dataset)
```

```
vis_dat(house_price_dataset)

summary(aggr(churn_dataset))

summary(aggr(house_price_dataset))


missing_data <- churn_dataset[!complete.cases(churn_dataset),]

summary(missing_data$tenure)

#Set total charges to 0

churn_dataset$total_charges[is.na(churn_dataset$total_charges)] <-0




#Cleaned data :)


#Near Zero variance vars

ch = nearZeroVar(churn_dataset, saveMetric= TRUE)

hp =nearZeroVar(house_price_dataset, saveMetric= TRUE)


##############

#Derived Variables

###################


#Churn

#tenure group

churn_dataset = churn_dataset %>% mutate(tenure_group = cut(tenure, breaks = c(-
1,13,25,37,49,61, Inf), labels = c("< 1 yr", "1-2 yrs", "2-3 yrs", "3-4 yrs", "4-5 yrs", "5+ yrs")))

#both phone and internet service

churn_dataset = churn_dataset %>% mutate(phone_and_internet_serv = (phone_service == "Yes" &
internet_service != "No"))


#House Prices
```

```
#sqft difference

house_price_dataset = house_price_dataset %>% mutate(sqft_living_percent_diff = ((sqft_living - sqft_living15)/sqft_living15)*100, sqft_lot_percent_diff = ((sqft_lot - sqft_lot15)/sqft_lot15)*100)

#renovation type

house_price_dataset = house_price_dataset %>% mutate(renovation_type = cut(yr_renovated, breaks = c(-Inf,0,1999,2009,2013,Inf), labels = c("None","1900s", "2000s", "2010-2014","Last Two Years")))

#grade class

house_price_dataset = house_price_dataset %>% mutate(grade_class = cut(as.numeric(grade), breaks = c(-Inf,2,9,Inf), labels = c("Low", "Average", "High")))


############################

#Summary Statistics

###########################

#Churn

##############################


#1 - Tenure Group


summary(churn_dataset$tenure_group)


tg1 <- ggplot(churn_dataset, aes(x = `tenure_group`)) +

  xlab("Tenure Group") +

  geom_bar(fill = "#87CEEB") +

  theme(axis.text.x = element_text(angle = 45, hjust = 1))


# Proportions bar charts

tg2 <- ggplot(churn_dataset, aes(x = `tenure_group` )) +

  geom_bar(fill = "#6495ED", aes(y = (..count..)/sum(..count..))) +

  xlab("Tenure Group") +

  scale_y_continuous(labels = scales::percent, name = "Proportion") +

  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
grid.arrange(tg1, tg2, nrow = 1)



ggplot(churn_dataset, aes(tenure_group, ..count..)) +
  geom_bar(aes(fill = churn), position = "dodge") +
  xlab("Tenure Group") + ylab("Count")

#2 - Monthly charges
summary(churn_dataset$monthly_charges)

mc1 <- ggplot(churn_dataset, aes(x = churn, y = monthly_charges, fill= churn)) +
  geom_boxplot()+
  xlab("Churn") + ylab("Monthly Charge")+
  scale_fill_brewer(palette="BuPu")
mc1
#3 - Contract
c1 <- ggplot(churn_dataset, aes(x = `contract`)) +
  xlab("Contract") +
  geom_bar(fill = "#87CEEB") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
c1

ggplot(churn_dataset, aes(contract, ..count..)) +
  geom_bar(aes(fill = churn), position = "dodge") +
  xlab("Contract Type") + ylab("Count")
#############
#Kings
##############
#Location
minihouse <- house_price_dataset %>% select(lat, long, price)
```

```
# creates a color gradient of length 5
colours <- brewer.pal(5, "YlOrRd")
# computes quantiles to later break up the data set based on price
breaks <- quantile(minihouse$price, probs = seq(0, 1, 0.2))

cat1 <- minihouse %>% filter(price >= breaks[1] & price <= breaks[2])
cat2 <- minihouse %>% filter(price > breaks[2] & price <= breaks[3])
cat3 <- minihouse %>% filter(price > breaks[3] & price <= breaks[4])
cat4 <- minihouse %>% filter(price > breaks[4] & price <= breaks[5])
cat5 <- minihouse %>% filter(price > breaks[5] & price <= breaks[6])
cat1$color <- colours[1]
cat2$color <- colours[2]
cat3$color <- colours[3]
cat4$color <- colours[4]
cat5$color <- colours[5]

minihouse <- rbind(cat1, cat2, cat3, cat4, cat5)
minihouse %>% head

grayscale = '[{"featureType":"landscape","stylers":[{"saturation":-100},{"lightness":65},
{"visibility":"on"}]},{"featureType":"poi","stylers":[{"saturation":-100},{"lightness":51},
{"visibility":"simplified"}]},{"featureType":"road.highway","stylers":[{"saturation":-100},
{"visibility":"simplified"}]},{"featureType":"road.arterial","stylers":[{"saturation":-100},
{"lightness":30},{"visibility":"on"}]},{"featureType":"road.local","stylers":[{"saturation":-100},
{"lightness":40},{"visibility":"on"}]},{"featureType":"transit","stylers":[{"saturation":-100},
{"visibility":"simplified"}]},{"featureType":"administrative.province","stylers":
[{"visibility":"off"}]},{"featureType":"water","elementType":"labels","stylers":
[{"visibility":"on"},{"lightness":-25},{"saturation":-20}]},
{"featureType":"water","elementType":"geometry","stylers":[{"hue":"#ffff00"},
{"lightness":-25},{"saturation":-97}]}]'
```

```
gmap(lat = 47.530096, lng = -122.162723, zoom = 10,
   width = 800, height = 1000, map_style = grayscale, title = "Kings County House Price Heatmap")
%>%
 ly_points(long, lat, data = minihouse,
        fill_alpha = 0.15, line_alpha = 0, col = color)

#Sqft_ft_living_difference

mc1 <- ggplot(house_price_dataset, aes(x = 0, y = sqft_living_percent_diff)) +
 geom_boxplot(fill = "#6495ED")+
 xlab("") + ylab("% Living Space Difference")+
 scale_fill_brewer(palette="BuPu") +
 theme(axis.title.x=element_blank(),
     axis.text.x=element_blank(),
     axis.ticks.x=element_blank())

plot(house_price_dataset$sqft_living_percent_diff)

ggplot(house_price_dataset,aes(x = price/1000, sqft_living_percent_diff )) +
 geom_point(alpha = 0.3) + stat_binhex() +
 xlab ("Price (000s)") + ylab ("Sq Foot Living % Difference") +
 geom_hline(yintercept=0, linetype="dashed", color = "red")

#Yr_built
house_price_dataset$real_year = as.numeric(as.character(house_price_dataset$yr_built))
ggplot(house_price_dataset, aes(x = real_year)) +
 xlab("Year Built") + ylab("Count") +
 geom_bar(fill = "#6495ED") +
 theme(axis.text.x = element_text(angle = 45, hjust = 1))

mc1 <- ggplot(house_price_dataset, aes(x = factor(yr_built), y = price/1000, fill = "#6495ED")) +
  geom_boxplot()+
  scale_x_discrete(breaks = c(seq(from = 1900, to = 2010, by = 10))) +
  xlab("Yr Built") + ylab("Price (000s)")
mc1



##############
#Trees
########################
##########
#Churn
##########
table(churn_dataset$churn)

#Split into test & training
set.seed(123)
churn_dataset$spl = sample.split(churn_dataset$churn, SplitRatio = 0.75)
churn_dataset_train <- subset(churn_dataset, spl == "TRUE")
nrow(churn_dataset_train)
ncol(churn_dataset_train)

churn_dataset_test <- subset(churn_dataset, spl == "FALSE")
#Removing the boolean set for sampling
churn_dataset_train <- churn_dataset_train[,-23]
churn_dataset_test <- churn_dataset_test[,-23]




# CP default to build shallow tree
fit1 <- rpart(churn_dataset_train$churn ~., method = "class", data = churn_dataset_train,
        parms = list(split="Gini"))
```

```
rpart.plot::rpart.plot(fit1,shadow.col="azure1",cex=0.6, type = 4, tweak = 1.1)



info_fit <-rpart(churn_dataset_train$churn ~., method = "class", data = churn_dataset_train,
           parms = list(split="Information"))
rpart.plot::rpart.plot(info_fit,shadow.col="azure 1",cex=0.6, type = 4, tweak = 1.1)

#Purposely overwriting cp to have a big tree.
fit2 <- rpart(churn_dataset_train$churn ~., method = "class", data = churn_dataset_train,
         control = rpart.control(cp = 0.001), parms = list(split="Gini"))
rpart.plot::rpart.plot(fit2,shadow.col="azure1",cex=0.3, type = 4)
plotcp(fit2, col = "blue")

#cptable of fit 2 shows xerror starts to increase after xerror value of 0.7482168.
#The # of splits for this xerror is 10 so we will use cp = 0.0043 and rerun
fit3 <- prune(fit2,cp = 0.0043)
rpart.plot::rpart.plot(fit3,shadow.col="azure1",cex=0., type = 4)

printcp(fit3)
plotcp(fit3)

churn_predict1 <- predict(fit1, churn_dataset_test)
churn_predict2 <- predict(fit2, churn_dataset_test)
churn_predict3 <- predict(fit3, churn_dataset_test)



predics1 <- prediction(churn_predict1[,2], churn_dataset_test$churn)
predics2 <- prediction(churn_predict2[,2], churn_dataset_test$churn)
predics3 <- prediction(churn_predict3[,2], churn_dataset_test$churn)
```

```
auc1 <- performance(predics1,"auc")@y.values

auc2 <- performance(predics2,"auc")@y.values

auc3 <- performance(predics3,"auc")@y.values


auc1

auc2

auc3


fit1roc = performance(predics1,"tpr","fpr")

fit2roc = performance(predics2, "tpr", "fpr")

fit3roc = performance(predics3, "tpr", "fpr")


plot(fit1roc, col="blue", main = "ROC CURVE COMPARISON")

plot(fit2roc, col="red", add = T, lty =2)

plot(fit3roc, col = "green", add =T)

legend("bottomright",legend=c("Fit 1", "Fit 2", "Fit 3", "Lin Reg", "SMOTE Fit", "Random Forest"),

    col=c("blue", "red", "green", "yellow", "purple", "orange"), lty=1:2, cex=0.6)


linreg<-glm(churn~.,family=binomial,data=(churn_dataset_train))

lin_pred <- predict(linreg, churn_dataset_test)

lin_predics <- prediction(lin_pred, churn_dataset_test$churn)

perf_lin = performance(lin_predics, "tpr", "fpr")

plot(perf_lin, col="yellow", add = TRUE, lty = 2)

abline(a=0,b=1, col = "grey")


churn_dataset_test$churn_predicted <- predict(fit3, churn_dataset_test, type = 'class')

xlab <- table(actualclass = churn_dataset_test$churn, predictedclass =
churn_dataset_test$churn_predicted)

cm <- confusionMatrix(xlab)
```

```
True_Value <- factor(c(0, 0, 1, 1))
Prediction <- factor(c(0, 1, 0, 1))
Y      <- c(cm$table[1], cm$table[2]
          , cm$table[3], cm$table[4])
df <- data.frame(True_Value, Prediction, Y)

ggplot(data =  df, mapping = aes(x = True_Value, y = Prediction)) +
  geom_tile(aes(fill = Y), colour = "white") +
  geom_text(aes(label = sprintf("%1.0f", Y)), vjust = 1) +
  scale_fill_gradient(low = "lightblue", high = "blue") +
  theme_bw() + theme(legend.position = "none")

#SMOTE
churn_dataset <- churn_dataset[,-23]
churn_dataset <- as.data.frame(churn_dataset)

new_churn <- SMOTE(churn ~., churn_dataset , perc.under = 200)

set.seed(123)
new_churn$spl = sample.split(new_churn$churn, SplitRatio = 0.75)
new_churn_train <- subset(new_churn, spl == "TRUE")

new_churn_test <- subset(new_churn, spl == "FALSE")
#Removing the boolean set for sampling
new_churn_train <- new_churn_train[,-23]
new_churn_test <- new_churn_test[,-23]



#Purposely overwriting cp to have a big tree.
SMfit1 <- rpart(new_churn_train$churn ~., method = "class", data = new_churn_train,
             control = rpart.control(cp = 0.001), parms = list(split="Gini"))
```

```
summary(SMfit1)

SMfit1$cptable

#Xerror increases at 0.4297265, CP = 0.001426873, 56 nsplit

SMfit2 <- prune(SMfit1, cp =0.0024)

rpart.plot::rpart.plot(SMfit2,shadow.col="azure1",cex=0.3, type = 4)



new_churn_predict <- predict(SMfit2, new_churn_test)

new_churn_test$churn_predicted <- predict(SMfit2, new_churn_test, type = "class")

new_xlab <- table(actualclass = new_churn_test$churn, predictedclass =
new_churn_test$churn_predicted)

cm2 <- confusionMatrix(new_xlab)


True_Value <- factor(c(0, 0, 1, 1))

Prediction <- factor(c(0, 1, 0, 1))

Y      <- c(cm2$table[1], cm2$table[2]

          , cm2$table[3], cm2$table[4])

df2 <- data.frame(True_Value, Prediction, Y)


ggplot(data =  df2, mapping = aes(x = True_Value, y = Prediction)) +

  geom_tile(aes(fill = Y), colour = "white") +

  geom_text(aes(label = sprintf("%1.0f", Y)), vjust = 1) +

  scale_fill_gradient(low = "lightblue", high = "blue") +

  theme_bw() + theme(legend.position = "none")



new_predics <- prediction(new_churn_predict[,2], new_churn_test$churn)


cm

cm2

SMroc <- performance(new_predics, "tpr","fpr")
```

```
plot(SMroc, col = "purple", add =T)

performance(new_predics, "auc")@y.values


############
#House Price
#############
house_price_dataset$price <- house_price_dataset$price/1000
set.seed(123)
house_price_dataset$spl <- sample.split(house_price_dataset$price, SplitRatio = 0)
house_price_train <- subset(house_price_dataset, spl == "TRUE")
house_price_test<-subset(house_price_dataset, spl == "FALSE")
house_price_train <- house_price_train[,-26]
house_price_test <- house_price_test[,-26]

fit1 <- rpart(house_price_train$price ~., method = "anova", data = house_price_train,
              control = rpart.control(cp = 0.05))

split.fun <- function(x, labs, digits, varlen, faclen)
{
  # replace commas with spaces (needed for strwrap)
  labs <- gsub(",", " ", labs)
  for(i in 1:length(labs)) {
    # split labs[i] into multiple lines
    labs[i] <- paste(strwrap(labs[i], width=25), collapse="\n")
  }
  labs
}
rpart.plot::prp(fit1, split.cex = 1, uniform = TRUE, split.fun=split.fun,
               type = 1, under = T,extra = 101, box.palette = c("pink", "palegreen"))
```

```
summary(fit1)


fit2 <- rpart(house_price_train$price ~., method = "anova", data = house_price_train,
            control = rpart.control(cp = 0.001))
fit2$cptable
plotcp(fit2)



#Xerror increasing at 0.2874504, CP = 0.004621982, nsplit = 13
fit3 <- prune(fit2,cp = 0.004621)
rpart.plot::prp(fit3, split.cex = 1, uniform = TRUE, split.fun=split.fun,
              type = 1, under = T,extra = 101, box.palette = c("pink", "palegreen"))

par(mfrow=c(1,1))
rsq.rpart(fit3)

price_pred3 <-predict(fit3, house_price_test, method = "anova")

cor(x = price_pred3, y = house_price_test$price)
RMSE(price_pred3, house_price_test$price)

fit4 <- prune(fit3,cp = 0.047)
rpart.plot::prp(fit4, split.cex = 1.6, uniform = TRUE, split.fun=split.fun,
              type = 1, under = T,extra = 101, box.palette = c("pink", "palegreen"))

price_pred4 <- predict(fit4, house_price_test, method = "anova")
cor(x = price_pred4, y = house_price_test$price)
RMSE(price_pred4, house_price_test$price)

residuals = (price_pred3 - house_price_test$price)
qplot(x = price_pred3, y = house_price_test$price)+ xlab("Predicted Price(000s)")+
  ylab("Actual Price (000s)") + geom_abline(col = "red",intercept = 0, slope = 1)


###########
#Random Forests
##############
######
#Churn
#####
#Split into test & training
set.seed(123)
new_churn$spl = sample.split(new_churn$churn, SplitRatio = 0.75)
new_churn_train <- subset(new_churn, spl == "TRUE")
nrow(new_churn_train)
ncol(new_churn_train)

new_churn_test <- subset(new_churn, spl == "FALSE")
#Removing the boolean set for sampling
new_churn_train <- new_churn_train[,-23]
new_churn_test <- new_churn_test[,-23]

# Create a Random Forest model with default parameters

forest2 <- randomForest(churn ~ ., data = new_churn_train, importance = TRUE)
forest2

forest3 <- randomForest(churn ~ ., data = new_churn_train, ntree = 500, mtry = 3, importance =
TRUE)
forest3

a=c()
```

```
i=5

for (i in 3:8) {

  model3 <- randomForest(churn ~ ., data = new_churn_train, ntree = 500, mtry = i, importance =
TRUE)

  predValid <- predict(model3, new_churn_test, type = "class")

  a[i-2] = mean(predValid == new_churn_test$churn)

}


a


plot(3:8,a, xlab= "mtry 3:8", ylab = "Accuracy of model",col = "blue", bg = "black", pch = 21)


predTrain <- predict(forest2, new_churn_test, type = "prob")

# Checking classification accuracy

table(predTrain, new_churn_test$churn)

mean(predTrain == new_churn_test$churn)


predic_rf <- prediction(predTrain[,2], new_churn_test$churn)

rf_perf <- performance(predic_rf,"tpr", "fpr")

plot(rf_perf, col = "orange", add =T, lty = 2)



varImpPlot(forest2, main = "Variable Importance for Random Forest", bg = "blue")

#####

#House Price

#####

#Split into test & training

set.seed(123)

#Convert some factor vars to numeric

house_price_dataset$zipcode <- as.numeric(house_price_dataset$zipcode)

house_price_dataset$yr_built <- as.numeric(house_price_dataset$yr_built)
```

```
house_price_dataset$spl = sample.split(house_price_dataset$price, SplitRatio = 0.75)
house_price_train <- subset(house_price_dataset, spl == "TRUE")
nrow(house_price_train)
ncol(house_price_train)

house_price_test <- subset(house_price_dataset, spl == "FALSE")
#Removing the boolean set for sampling
house_price_train <- house_price_train[,-26]
house_price_test <- house_price_test[,-26]

# Create a Random Forest model with default parameters
set.seed(123)
forest1 <- randomForest(price ~ ., data = house_price_train, mtry=3,  importance = TRUE)
forest1

varImpPlot(forest1,main = "Variable Importance for Random Forest", bg = "blue")


predTrain <- predict(forest1, house_price_test)

RMSE.forest <- sqrt(mean((predTrain - house_price_test$price)^2))
RMSE.forest

# Checking classification accuracy
table(predTrain, house_price_test$price)
mean(predTrain == house_price_test$price)

########
#Other approach for growing trees - Chaid
#########
```

```
install.packages("partykit")
install.packages("CHAID", repos="http://R-Forge.R-project.org")
install.packages("rsample")
install.packages("kableExtra")


require(rsample) # for dataset and splitting also loads broom and tidyr
require(dplyr)
require(ggplot2)
theme_set(theme_bw()) # set theme
require(CHAID)
require(purrr)
require(caret)
library(kableExtra)

#Chaid can only take factors
churn_dataset %>%
  select_if(is.factor) %>%
  ncol

churn_dataset %>%
  select_if(is.numeric) %>%
  ncol

#Have to drop monthly charges & total charges, Tenure group will contain tenure
#Make monthly charges factor var using quartiles
churn_dataset <- within(churn_dataset, monthlycharge_quartile <- as.integer(cut(monthly_charges,
                                                    quantile(monthly_charges, probs=0:4/4),
                                                    include.lowest=TRUE)))
churn_dataset$monthlycharge_quartile = as.factor(churn_dataset$monthlycharge_quartile)
churn_dataset$phone_and_internet_serv = as.factor(churn_dataset$phone_and_internet_serv)
```

```
fact_churn <- churn_dataset %>%
  select_if(is.factor)
dim(fact_churn)



chaidattrit1 <- chaid(churn ~ ., data = fact_churn)
plot(chaidattrit1, type = "simple", gp = gpar(fontsize = 4, theme = "blue",
                                              lty = "solid",
                                              lwd = 1))

ctrl <- chaid_control(minsplit = 200, minprob = 0.05)

chaidattrit2 <- chaid(churn ~ ., data = fact_churn, control = ctrl)
print(chaidattrit2)
plot(chaidattrit2,  gp = gpar(fontsize = 4, theme = "blue",
                              lty = "solid",
                              lwd = 1))
ctrl <- chaid_control(maxheight = 3)
chaidattrit3 <- chaid(churn ~ ., data = fact_churn, control = ctrl)
print(chaidattrit3)
plot(chaidattrit3,gp = gpar(fontsize = 6.5, theme = "blue",
                            lty = "solid",
                            lwd = 1))
pmodel1 <- predict(chaidattrit1)
pmodel2 <- predict(chaidattrit2)
pmodel3 <- predict(chaidattrit3)

cm1 <-confusionMatrix(pmodel1, fact_churn$churn)
cm2 <-confusionMatrix(pmodel2, fact_churn$churn)
cm3 <-confusionMatrix(pmodel3, fact_churn$churn)
```

```
True_Value <- factor(c(0, 0, 1, 1))

Prediction <- factor(c(0, 1, 0, 1))

Y      <- c(cm3$table[1], cm3$table[2]

          , cm3$table[3], cm3$table[4])

df <- data.frame(True_Value, Prediction, Y)


ggplot(data =  df, mapping = aes(x = True_Value, y = Prediction)) +

  geom_tile(aes(fill = Y), colour = "white") +

  geom_text(aes(label = sprintf("%1.0f", Y)), vjust = 1) +

  scale_fill_gradient(low = "lightblue", high = "blue") +

  theme_bw() + theme(legend.position = "none")




modellist <- list(CHAID1 = chaidattrit1, CHAID2 = chaidattrit2, CHAID3 = chaidattrit3)

CHAIDResults <- map(modellist, ~ predict(.x)) %>%

  map(~ confusionMatrix(fact_churn$churn, .x)) %>%

  map_dfr(~ cbind(as.data.frame(t(.x$overall)),as.data.frame(t(.x$byClass))), .id = "ModelNumb")

kable(CHAIDResults, "html") %>%

  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),

              font_size = 9)

sort(varimp(chaidattrit1), decreasing = TRUE)

plot(sort(varimp(chaidattrit1), decreasing = TRUE))


#########
```