ACT Land Tax Detection Algorithm

Project Documentation

Rafi Rahman, Ashwin Mayilvahanan, Matej Mitrev, and Abuzar Lone

06/10/2021

Abstract

This documentation, created for the ACT Revenue Office, outlines the work done for the data analysis, preprocessing, and modelling of the ACT_Properties dataset provided by the ACT Revenue Office for UC Capstone Project Group 10004/10005-1.

Contents

1 Overview

Packages Used: tidyverse, factoextra, cowplot, caret, fastAdaboost, imbalance, randomForest, and FNN

Dataset: ACT_Properties.csv (as provided by the ACT Revenue Office)

Modelling Techniques: K-Nearest Neighbour, Random Forest, Decision Tree, and Adaptive Boosting

Highest Achieved Accuracy: ~91%

2 Data Preprocessing

2.1 Initialisation

```r
# Packages
library(tidyverse)
library(factoextra) # For clustering (assist for EDA)
library(cowplot)
library(caret) # Install all dependencies as well
library(fastAdaboost) # Faster AdaBoost
library(imbalance) # For oversampling
library(randomForest)
library(FNN) # Faster KNN

# Import dataset
ACTproperties <- read.table(file.choose(), header = T, sep = ",")
```

Functions:

```r
# Find most common value
calc_mode_property <- function(x) {
  distinct_values <- unique(x)

  # If there are more than two values then we can guarantee that if one of the
  # values is Unknown, there is something to replace it with
  if (length(distinct_values) >= 2) {
    distinct_values <- distinct_values[distinct_values != "Unknown"]
  }

  distinct_tabulate <- tabulate(match(x, distinct_values))
  distinct_values[which.max(distinct_tabulate)]
}

# Find most common value (numeric)
calc_mode_numeric <- function(x) {
  distinct_values <- unique(x)
  dist_length <- length(distinct_values)

  # If >= 3 we can always safely remove 0's and NA's
  if (dist_length >= 3) {
    distinct_values <- distinct_values[distinct_values != 0]
    distinct_values <- distinct_values[!is.na(distinct_values)]
  }

  # If == 2 we have to test before we can remove 0's and NA's
  if (dist_length == 2) {
    # 0 & NA test case
    if (is.element(0, distinct_values) & is.element(NA, distinct_values)){
      distinct_values <- distinct_values[!is.na(distinct_values)]
      length(distinct_values)
    }

    # 0 & >= 1 test case
    if (is.element(0, distinct_values) & !is.element(NA, distinct_values)) {
      distinct_values <- distinct_values[distinct_values != 0]
    }

    # >= 1 & NA test case
    if (!is.element(0, distinct_values) & is.element(NA, distinct_values)) {
      distinct_values <- distinct_values[!is.na(distinct_values)]
    }
  }

  distinct_tabulate <- tabulate(match(x, distinct_values))
  distinct_values[which.max(distinct_tabulate)]
}
```

2.2 Dataset Insights

```
# Check NA's
sapply(ACTproperties, function(i) sum(is.na(i))/nrow(ACTproperties)*100)
```

```
##                       X          AssessmentId     OriginalAddressTxt
##              0.00000000            0.00000000             0.00000000
##                SuburbNm  PropertyValuationAmt          SalesPriceAmt
##              0.00000000            0.02170819            21.04368171
##            SalesPriceDt          PropertyType               RentalDt
##             22.46496539            0.00000000             0.00000000
## ElectricityConsumption      WaterConsumption      WeeklyRentPriceAmt
##             41.12255481           71.24870354             0.01326612
##            TotalBondAmt            BedroomCnt            BathroomCnt
##              0.00000000            0.00000000            42.98342941
##             CarSpaceCnt         RoomRentalFlg             LandtaxFlg
##             48.15118551            0.00000000             0.00000000
##               TenantCnt
##              0.00000000
```

High percent of NA's in SalesPriceAmt, SalePriceDt, BathroomCnt, CarSpaceCnt, WaterConsumption, and ElectricityConsumption.

```
# Check for imbalance in data
count(ACTproperties, LandtaxFlg)
```

```
##   LandtaxFlg      n
## 1          0  15920
## 2          1 149916
```

Ten times as many 1's as 0's, meaning the dataset is quite imbalanced.

```r
# Check LandtaxFlg-RoomRentalFlg 1-1 and 0-0
count(ACTproperties, LandtaxFlg, RoomRentalFlg)
```

```
##   LandtaxFlg RoomRentalFlg      n
## 1          0             0  15757
## 2          0             1    163
## 3          1             0 149867
## 4          1             1     49
```

There are too many 0-0 rows, so we made the assumption that that they are meant to be 0-1. As for the 1-1 rows, they can be safely dropped.

```r
ACTproperties <- ACTproperties[!(ACTproperties$LandtaxFlg == 1 &
                                   ACTproperties$RoomRentalFlg == 1), ]

# Drop useless variables
ACTproperties <- subset(ACTproperties, select = -c(X,
                                                    SalesPriceAmt,
                                                    SalesPriceDt,
                                                    RoomRentalFlg,
                                                    RentalDt,
                                                    WaterConsumption,
                                                    ElectricityConsumption,
                                                    OriginalAddressTxt))
```

RoomRentalFlg is dropped as it is supposed to be the inverse of LandtaxFlg. WaterConsumption and ElectricityConsumption had way too many NA's. The rest are unnecessary for modelling and will not have an impact on LandtaxFlg, the target variable.

```r
# Reformat datatypes
ACTproperties <- ACTproperties %>% mutate_if(is.numeric, as.integer)

# Unique Values
unique(ACTproperties$PropertyType)
```

```
## [1] "Flat/Unit"        "Separated House" "Townhouse/Semi-" "Unknown"
## [5] "UNK"
```

```r
unique(ACTproperties$BedroomCnt)
```

```
##  [1]  1  0  3 99  2  5  4  6  7 26  8 30  9 88 40 12 10 19 11 22 25
```

```r
unique(ACTproperties$BathroomCnt)
```

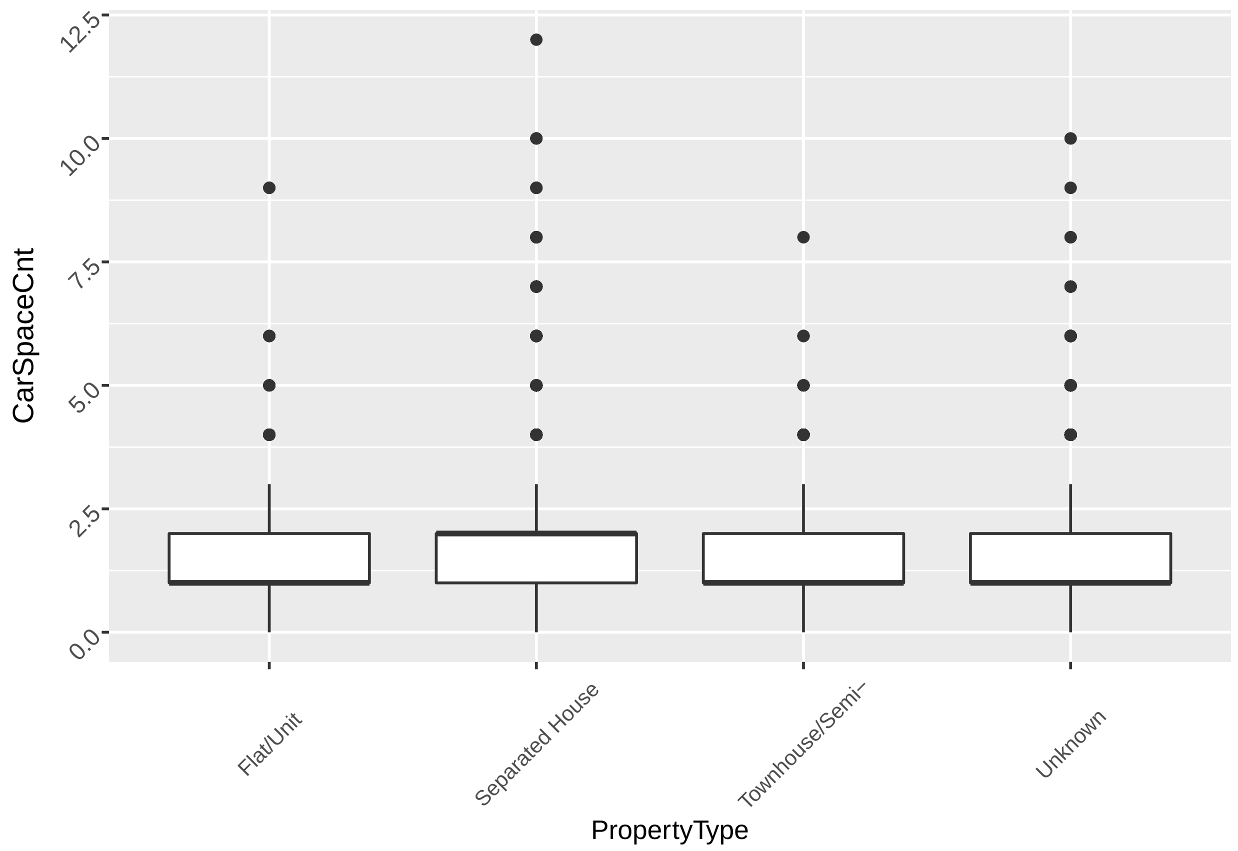
```
##  [1] NA  1  2  3  4  5  0  6  8  7
```

```r
unique(ACTproperties$CarSpaceCnt)
```

```
##  [1] NA  1  0  2  4  3  5  8  9  7  6 10 12
```

2.3 Cleaning

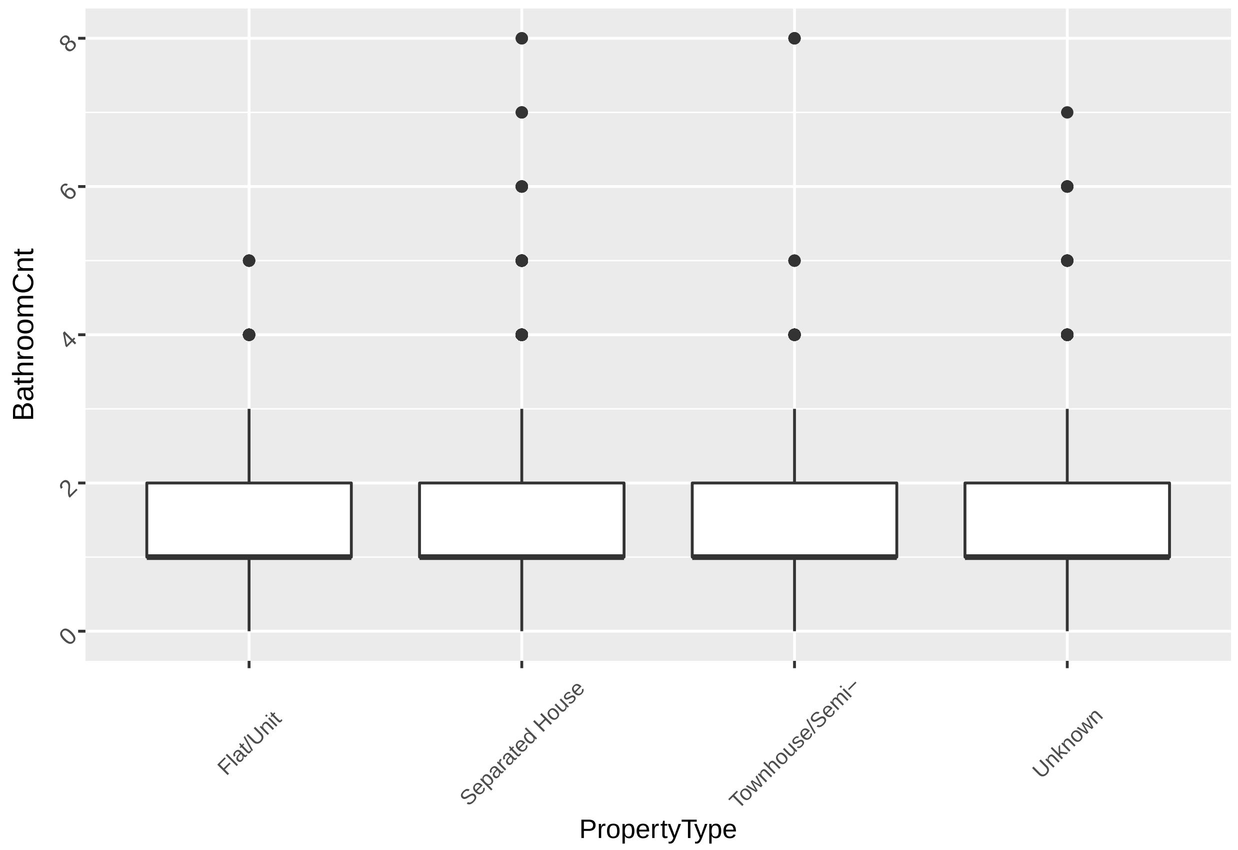
```r
# Change UNK to Unknown for consistency in PropertyType
ACTproperties$PropertyType[ACTproperties$PropertyType == "UNK"] <- "Unknown"

# Find outliers in CarSpaceCnt
ggplot(ACTproperties, aes(x = PropertyType, y = CarSpaceCnt)) +
  geom_boxplot() +
  theme(plot.title = element_text(size = "12", face = "bold"),
        axis.title.x = element_text(hjust = "0.5", size = "10"),
        axis.text.x = element_text(size="8", vjust = 0.5),
        axis.text = element_text(angle = 45))
```



We decided to set 7 as the upper outlier threshold in this instance.

```r
# Remove outliers
ACTproperties$CarSpaceCnt[ACTproperties$CarSpaceCnt >= 7] <- NA
```

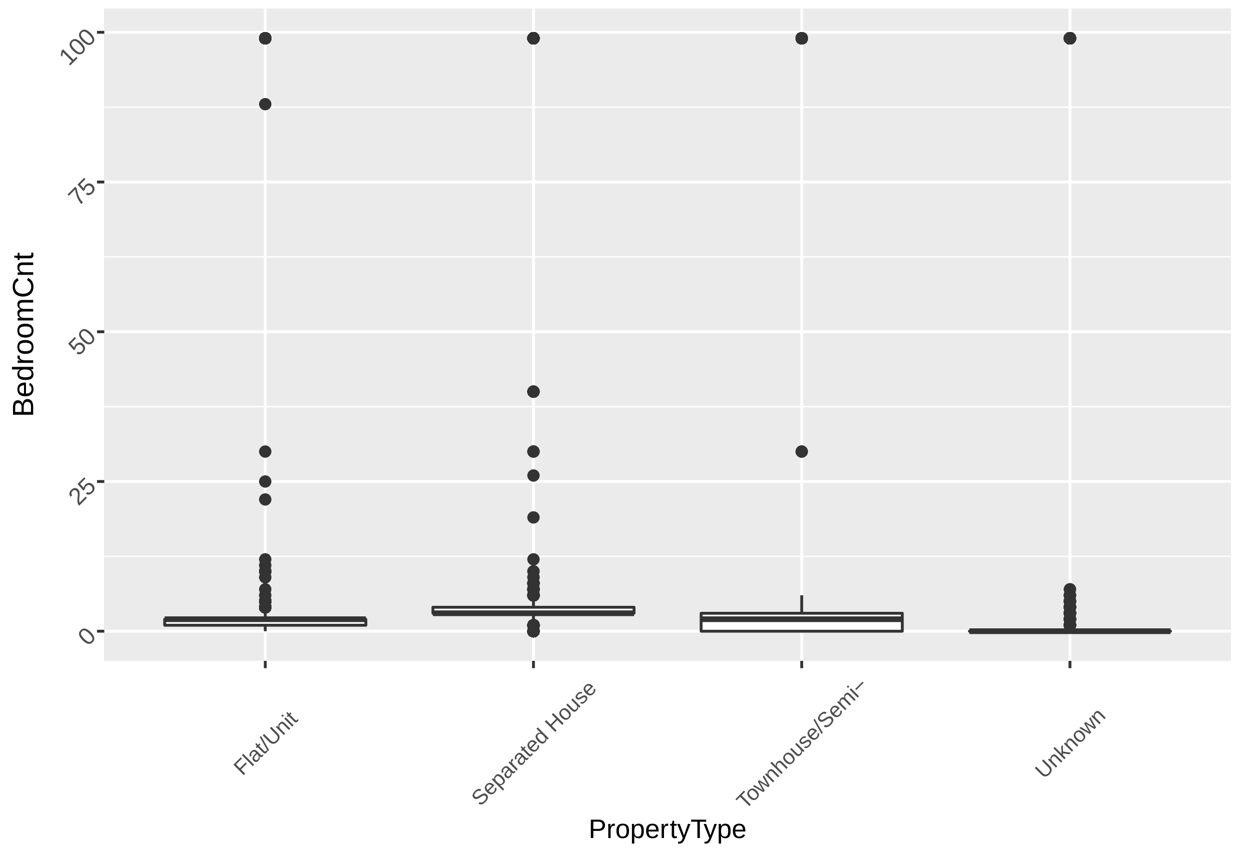
```
# Find outliers in BathroomCnt
ggplot(ACTproperties, aes(x = PropertyType, y = BathroomCnt)) +
  geom_boxplot() +
  theme(plot.title = element_text(size = "12", face = "bold"),
        axis.title.x = element_text(hjust = "0.5", size = "10"),
        axis.text.x = element_text(size="8", vjust = 0.5),
        axis.text = element_text(angle = 45))
```



For this instance, 6 was determined as the upper outlier threshold. Because it is highly unlikely that a property would have 0 bathrooms, those have been considered outliers as well.

```
# Remove outliers
ACTproperties$BathroomCnt[ACTproperties$BathroomCnt >= 6] <- NA
ACTproperties$BathroomCnt[ACTproperties$BathroomCnt == 0] <- NA
```

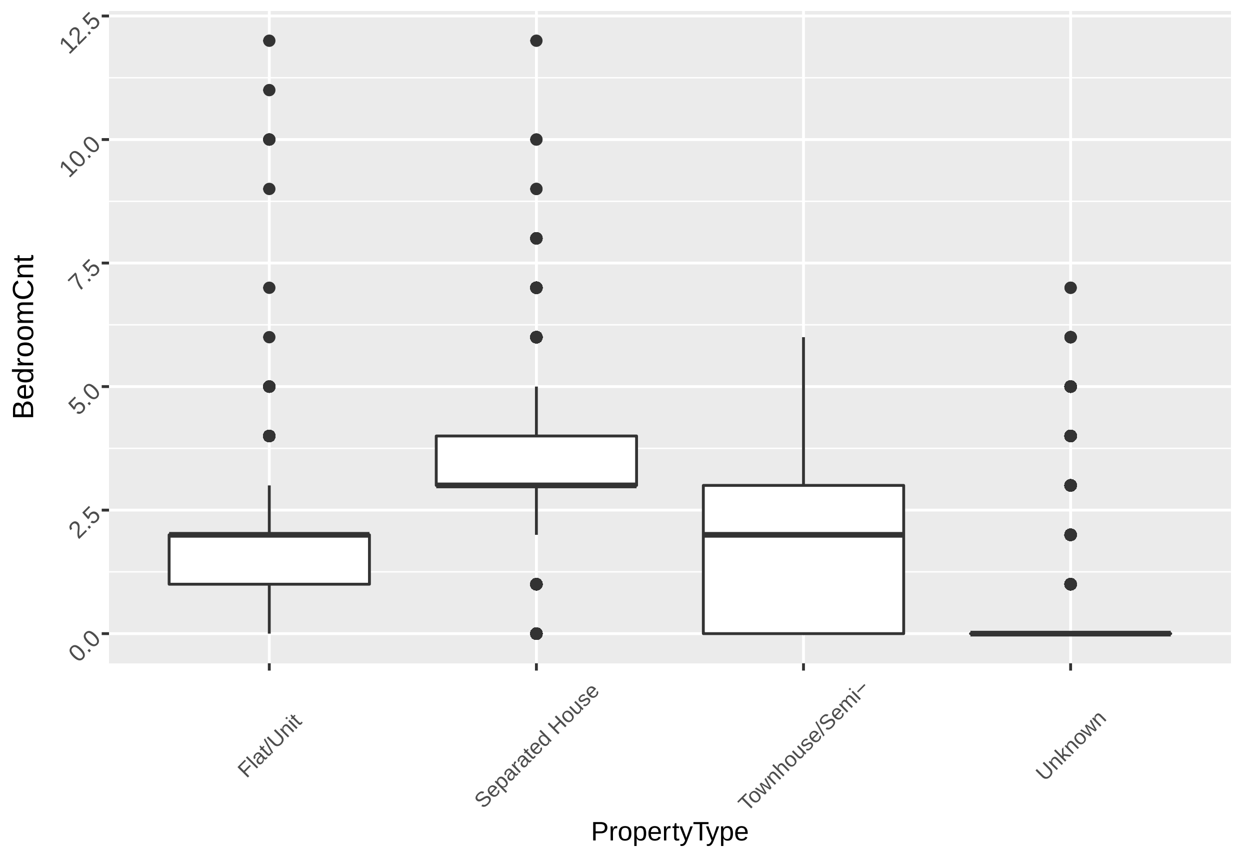
```r
# Find outliers in BedroomCnt
ggplot(ACTproperties, aes(x = PropertyType, y = BedroomCnt)) +
  geom_boxplot() +
  theme(plot.title = element_text(size = "12", face = "bold"),
        axis.title.x = element_text(hjust = "0.5", size = "10"),
        axis.text.x = element_text(size="8",vjust = 0.5),
        axis.text = element_text(angle = 45))
```



For a better view, we removed outliers over 15 first.

```r
# Remove outliers
ACTproperties$BedroomCnt[ACTproperties$BedroomCnt >= 15] <- NA
```

```
# Find more outliers in BedroomCnt
ggplot(ACTproperties, aes(x = PropertyType, y = BedroomCnt)) +
  geom_boxplot() +
  theme(plot.title = element_text(size = "12", face = "bold"),
        axis.title.x = element_text(hjust = "0.5", size = "10"),
        axis.text.x = element_text(size="8",vjust = 0.5),
        axis.text = element_text(angle = 45))
```



```
# Remove outliers
ACTproperties$BedroomCnt[ACTproperties$BedroomCnt >= 7] <- NA
ACTproperties$BathroomCnt[ACTproperties$BathroomCnt == 0] <- NA
```

2.4 Imputing

```
# Replace by most prevalent value for PropertyType, BedroomCnt, BathroomCnt, and
# CarSpaceCnt for each address
ACTproperties <- ACTproperties %>%
  group_by(AssessmentId) %>%
  mutate(PropertyType = calc_mode_property(PropertyType),
         BedroomCnt = calc_mode_numeric(BedroomCnt),
         BathroomCnt = calc_mode_numeric(BathroomCnt),
         CarSpaceCnt = calc_mode_numeric(CarSpaceCnt))
```

```r
# Remove remaining NA observations
ACTproperties <- na.omit(ACTproperties)
ACTproperties <- ACTproperties[!(ACTproperties$PropertyType == "Unknown"), ]

# Random undersampling of LandtaxFlg = 1
true_properties <- ACTproperties[(ACTproperties$LandtaxFlg == 1), ]
false_properties <- ACTproperties[(ACTproperties$LandtaxFlg == 0), ]
true_properties <- true_properties[sample(1:nrow(true_properties),
                                          60000 - nrow(false_properties)), ]
ACTproperties <- rbind(false_properties, true_properties)
rm(true_properties, false_properties)

# Convert char to factor
ACTproperties$PropertyType <- as.factor(ACTproperties$PropertyType)
ACTproperties$SuburbNm <- as.factor(ACTproperties$SuburbNm)

# Dataset of property types with matching indices
propertytypes <- as.data.frame(sapply(ACTproperties$PropertyType,
                                      levels)[, 1]) %>%
  rename(Name = `sapply(ACTproperties$PropertyType, levels)[, 1]`)

# Dataset of suburb names with matching indices
suburbs <- as.data.frame(sapply(ACTproperties$SuburbNm, levels)[, 1]) %>%
  rename(Name = `sapply(ACTproperties$SuburbNm, levels)[, 1]`)

# Convert factor to integer so oversampling can be done
ACTproperties$PropertyType <- as.integer(ACTproperties$PropertyType)
ACTproperties$SuburbNm <- as.integer(ACTproperties$SuburbNm)

# Remove non-numeric variables
ACTproperties <- ungroup(ACTproperties)
ACTproperties <- as.data.frame(ACTproperties %>% select(-AssessmentId))

# Balance data by oversampling LandtaxFlg = 0
ACTproperties <- oversample(ACTproperties, method = "ADASYN",
                            classAttr = "LandtaxFlg")
```

Finalise for Modelling:

```r
# Categorise LandtaxFlg
ACTproperties$LandtaxFlg <- as.factor(ACTproperties$LandtaxFlg)

# Reorder variables so that LandtaxFlg is last
ACTproperties <- ACTproperties %>% select(-LandtaxFlg, everything())

# Reset row indices
row.names(ACTproperties) <- NULL

# Ensure that ACTproperties remains a data frame
ACTproperties <- as.data.frame(ACTproperties)
```

3 Modelling and Prediction

```r
# Split Dataset Train 75% and Test 25%
inTrain <- createDataPartition(
  y = ACTproperties$LandtaxFlg,
  p = .75,
  list = F
)

train <- as.data.frame(ACTproperties[inTrain, ])
test <- as.data.frame(ACTproperties[-inTrain, ])
rm(inTrain)
```

The custom bias score follows the following metric:

- Negative values: Bias towards 0
- 0 to 25%: Low, with 0% meaning no bias at all
- 25 to 50%: Moderate
- 50 to 75%: High
- 75 to 100%: Very high, with 100% meaning a complete bias towards 1

3.1 K-Nearest Neighbour

```r
prediction1 <- knn(train %>% select(-LandtaxFlg),
                   test %>% select(-LandtaxFlg),
                   train$LandtaxFlg,
                   k = 15,
                   prob = FALSE,
                   algorithm = c("kd_tree", "cover_tree", "brute"))

test$Prediction1 <- prediction1
```

Result:

```r
# Confusion matrix
confusionMatrix(test$Prediction1, test$LandtaxFlg, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 10018  4549
##          1  2561  8248
##
##                Accuracy : 0.7198
##                  95% CI : (0.7142, 0.7253)
##     No Information Rate : 0.5043
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.4403
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6445
##             Specificity : 0.7964
##          Pos Pred Value : 0.7631
##          Neg Pred Value : 0.6877
##              Prevalence : 0.5043
##          Detection Rate : 0.3250
##    Detection Prevalence : 0.4260
##       Balanced Accuracy : 0.7205
##
##        'Positive' Class : 1
##
```

```
# Custom bias score
error <- count(test, LandtaxFlg, Prediction1)$n
print(error[2]*2/(error[2] + error[3]) - 1)
```

```
## [1] -0.2796062
```

```
rm(error)
```

3.2 Random Forest

```
RF_model <- randomForest(LandtaxFlg ~ .,
                         data = train,
                         ntree = 100,
                         mtry = 2,
                         importance = TRUE)

prediction2 <- predict(RF_model, test)
test$Prediction2 <- prediction2
```

Result:

```
# Confusion matrix
confusionMatrix(test$Prediction2, test$LandtaxFlg, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 10189  2733
##          1  2390 10064
##
##                Accuracy : 0.7981
##                  95% CI : (0.7931, 0.803)
##     No Information Rate : 0.5043
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5963
##
##  Mcnemar's Test P-Value : 1.769e-06
##
##             Sensitivity : 0.7864
##             Specificity : 0.8100
##          Pos Pred Value : 0.8081
##          Neg Pred Value : 0.7885
##              Prevalence : 0.5043
##          Detection Rate : 0.3966
##    Detection Prevalence : 0.4908
##       Balanced Accuracy : 0.7982
##
##        'Positive' Class : 1
##
```

```
# Custom bias score
error <- count(test, LandtaxFlg, Prediction2)$n
print(error[2]*2/(error[2] + error[3]) - 1)
```

```
## [1] -0.06695296
```

```
rm(error)
```

3.3 Decision Tree

```
decisiontree_model <- train(LandtaxFlg ~ .,
                            data = train,
                            method = 'rpart',
                            preProc = c("center", "scale"),
                            tuneLength = 15,
                            trControl = trainControl(method = "cv"))

prediction3 <- predict(decisiontree_model, test)
test$Prediction3 <- prediction3
```

Result:

```
# Confusion matrix
confusionMatrix(table(test$Prediction3, test$LandtaxFlg), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
##         0    1
##    0 7879 3990
##    1 4700 8807
##
```

```
##                Accuracy : 0.6576
##                  95% CI : (0.6517, 0.6634)
##     No Information Rate : 0.5043
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3147
##
##  Mcnemar's Test P-Value : 2.835e-14
##
##             Sensitivity : 0.6882
##             Specificity : 0.6264
##          Pos Pred Value : 0.6520
##          Neg Pred Value : 0.6638
##              Prevalence : 0.5043
##          Detection Rate : 0.3471
##    Detection Prevalence : 0.5323
##       Balanced Accuracy : 0.6573
##
##        'Positive' Class : 1
##
```

```
# Custom bias score
error <- count(test, LandtaxFlg, Prediction3)$n
print(error[2]*2/(error[2] + error[3]) - 1)
```

```
## [1] 0.08170311
```

```
rm(error)
```

3.4 Adaptive Boosting

```
adaboost_model <- adaboost(LandtaxFlg ~ ., train, 50)
prediction4 <- predict(adaboost_model, test)
test$Prediction4 <- prediction4$class
```

Result:

```
# Confusion matrix
confusionMatrix(table(test$Prediction4, test$LandtaxFlg), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
##         0     1
##   0 11346   983
##   1  1233 11814
##
##                Accuracy : 0.9127
##                  95% CI : (0.9091, 0.9161)
##     No Information Rate : 0.5043
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8253
##
##  Mcnemar's Test P-Value : 1.227e-07
##
##             Sensitivity : 0.9232
##             Specificity : 0.9020
##          Pos Pred Value : 0.9055
##          Neg Pred Value : 0.9203
##              Prevalence : 0.5043
##          Detection Rate : 0.4656
##    Detection Prevalence : 0.5141
##       Balanced Accuracy : 0.9126
##
##        'Positive' Class : 1
##
```

```
# Custom bias score
error <- count(test, LandtaxFlg, Prediction4)$n
print(error[2]*2/(error[2] + error[3]) - 1)
```

```
## [1] 0.1128159
```

```
rm(error)
```

Overall model run time:

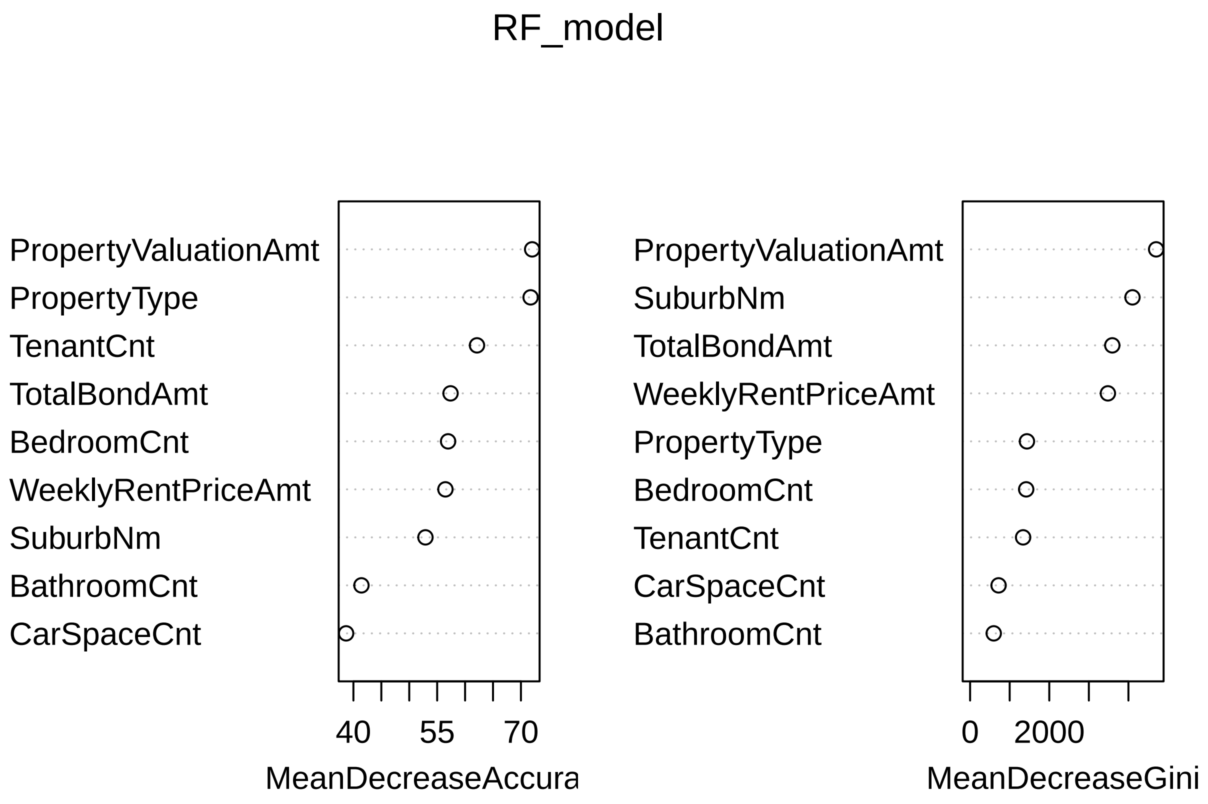
```
## Time difference of 5.817716 mins
```

4 Analysis

The custom bias score was a method to indicate that data imbalance was largely taken care of, and hence is not too relevant. The better metrics are sensitivity, specificity, and balanced accuracy as shown in the confusion matrices.

Also, another important note: the values have slight changes with each run.

AdaBoost, the best performing model, does not include feature importance in the faster package used, so instead we decided to look at Random Forest for the feature importance, as it was the second best performing model.



PropertyValuationAmt, PropertyType, TenantCnt, WeeklyRentPriceAmt, BedroomCnt, SuburbNm, and TotalBondAmt all seem to be the important metrics for determining the value of LandtaxFlg.

Suburb name might be a bit of a stretch here, but it seemed to score pretty high for MeanDecreaseGini. If it is to be used as a metric when testing on new data, the value inputted should correspond to the *suburbs* dataset created right before oversampling was done. Same goes for property type (*propertytypes*).

Bathroom count and car space count seem to be relatively unimportant (compared to the other features; they might still be useful), which does make sense, as bedroom count would naturally be the better indicator.

Note: Accuracy and balanced accuracy are within 1 significant figure of each other, which is good because that means the dataset is properly balanced. For the purposes of analysis they'll represent the same value, and will be discussed using the same terminology.

Modeling provided some very hopeful results, with all of the models accuracy scoring above the 60% target. With the models using ensemble techniques scoring higher than the traditional models. Decision Tree scored the lowest accuracy with 65%, KNN scoring better accuracy with 71%, Random Forest scoring the second highest accuracy with 80%, and Adaptive Boosting scoring the highest accuracy with 91%.

This is only natural, as random forest and AdaBoost are both ensemble machine learning techniques. Boosting in particular is quite powerful.

A closer look into the confusion matrix for Adaptive Boosting shows:

- True Negative being 11,319. Accounting for 44.5% of the test data **correctly** identified as *LandTaxFlg = 0.*
- True Positive being 11,858. Accounting for 46.7% of the test data **correctly** identified as *LandTaxFlg = 1.*
- False Negative being 939. Accounting for 3.7 % of the test data **incorrectly** identified as *LandTaxFlg = 0.*
- False Positive being 1,288. Accounting for 5.1% of the test data **incorrectly** identified as *LandTaxFlg = 1.*

Sensitivity and specificity evaluate a model's ability to predict true positives and true negatives. Adaptive Boosting attains a sensitivity score of 92.7%; this indicates that the model can predict the true positives very well. The specificity indicates the opposite class, at 89.8%, suggesting the model can also predict the true negatives very well.

5 Recommendation and Conclusion

The best way to evaluate the performance of these models is to test it on new data, just to be 100% sure. However, the results here are a pretty good indicator of what to expect.

We would recommend the use of AdaBoost, as we got the highest scores with it. The other models can be used as well alongside it, especially random forest, as it has a pretty good score. As the system for detecting noncompliance utilises multiple indicators to identify properties of interest, it is reasonable to implement multiple models as individual indicators. These indicators produced by the models can be weighted appropriately.

Oversampling and undersampling were both utilised for this dataset due to the imbalance; the data should be undersampled to a reasonable amount first, then oversampled. In our case, we undersampled it down to 60,000 rows (a value we came upon accidentally) before oversampling. Too much oversampling can be a detriment to the performance of the models. As an example, when we initially dropped the *LandtaxFlg-RoomRentalFlg* 0-0 rows (resulting in only around 160 *LandtaxFlg = 0* rows) the modelling resulted in a custom bias score of around -98%, flipping the bias heavily towards *LandtaxFlg = 0.*

The assumptions we made should also be taken into account, as you may disagree with them, e.g. the assumption we made about the *LandtaxFlg-RoomRentalFlg* 0-0 rows being 0-1 rows, which we made largely so that we could have more *LandtaxFlg = 0* values. We figured that if it was listed like that, that meant it was not flagged for land tax, which would in turn mean it was a boarder lodger agreement.

If data is acquired where the data imbalance problem is less prevalent, the oversampling and undersampling method can be disregarded. This also applies to the assumptions we have made. However as mentioned – *"Unfortunately, real world data sucks." ~ Rino Ciaccia, circa 2021.*

In terms of the scope of the project, we believe it is possible to accurately predict when a rental property is a boarder/rental lodgement through the use of machine learning algorithms. Many of the features in the dataset seemed to have a significant impact on *LandtaxFlg.* Detection of non-compliant individuals should be possible by applying these methods and manually cross checking to verify that the property owner's tax obligations are being met.