```
---
title: "Homework 2"
author: "Kavana Manvi KrishnaMurthy"
date: "2024-04-24"
output: html_document
---
```

Problem 1

For this problem, you will load and perform some cleaning steps on a dataset in the provided BankData.csv, which is data about loan approvals from a bank in Japan (it has been modified from the original for our purposes in class, so use the provided version). Specifically, you will use visualization to examine the variables and normalization, binning and smoothing to change them in particular ways.

a. Visualize the distributions of the variables in this data. You can choose bar graphs, histograms and density plots. Make appropriate choices given each type of variables and be careful when selecting parameters like the number of bins for the histograms. Note there are some numerical variables and some categorical ones. The ones labeled as a 'bool' are Boolean variables, meaning they are only true or false and are thus a special type of categorical. Checking all the distributions with visualization and summary statistics is a typical step when beginning to work with new data.

Load and look at the summary of BankData dataset.

```{r}
bank <-read.csv("/Users/kavanamanvi/Desktop/FDS/HW2/BankData.csv")
summary(bank)
```

Histograms for numerical data

```{r}
library("ggplot2")
ggplot(bank, aes(x = X)) +
geom_histogram(binwidth = 60,bins = length(unique(bank$X))) +
labs(title = "Histogram of X", x = "X values", y = "count")
```

```{r}
ggplot(bank, aes(x = cont1)) +
geom_histogram(binwidth = 60,bins = length(unique(bank$cont1))) +
labs(title = "Histogram of cont1", x = "values", y = "count")
```

```{r}
ggplot(bank, aes(x = cont2)) +
geom_histogram(binwidth = 60,bins = length(unique(bank$cont2))) +
labs(title = "Histogram of cont2", x = "values", y = "count")
```

```{r}
ggplot(bank, aes(x = cont3)) +
geom_histogram(binwidth = 60,bins = length(unique(bank$cont3))) +
labs(title = "Histogram of cont3", x = "values", y = "count")
```

```{r}
ggplot(bank, aes(x = cont4)) +
geom_histogram(binwidth = 60,bins = length(unique(bank$cont4))) +
labs(title = "Histogram of cont4", x = "values", y = "count")
```

```{r}
ggplot(bank, aes(x = cont5)) +
geom_histogram(binwidth = 60) +
labs(title = "Histogram of cont5", x = "values", y = "count")
```

```{r}
ggplot(bank, aes(x = cont6)) +
geom_histogram(bins = length(unique(bank$cont6))) +
labs(title = "Histogram of cont6", x = "values", y = "count")
```

```{r}
ggplot(bank, aes(x = credit.score)) +
geom_histogram(binwidth = 60,bins = length(unique(bank$credit.score))) +
labs(title = "Histogram of credit.score", x = "values", y = "count")
```

```{r}
ggplot(bank, aes(x = ages)) +
geom_histogram(binwidth = 60,bins = length(unique(bank$ages))) +
labs(title = "Histogram of ages", x = "values", y = "count")
```

Bar graph for categorical variables

```{r}
ggplot(bank, aes(x=approval)) + geom_bar()+
  labs(title = "Histogram of approvals", x = "values", y = "count")
```

```{r}
ggplot(bank, aes(x=bool1)) + geom_bar() +
labs(title = "Histogram of bool1", x = "values", y = "count")
```

```{r}
ggplot(bank, aes(x=bool2)) + geom_bar()+
labs(title = "Histogram of bool2", x = "values", y = "count")
```

```{r}
ggplot(bank, aes(x=bool3)) + geom_bar()+
  labs(title = "Histogram of bool3", x = "values", y = "count")
```

b. Now apply normalization to some of these numerical distributions. Specifically, choose to apply z- score to one, min-max to another, and decimal scaling to a third. Explain your choices of which normalization applies to which variable in terms of what the variable means, what distribution itstarts with, and how the normalization will affect it.

Apply Z-score normalization to Age :For the Age feature, I would apply Z-score normalization. Z-score normalization is suitable for this variable because it does not contain extreme outliers, and it allows us to standardize the feature distribution to have a mean of 0 and a standard deviation of 1. This technique is particularly useful when dealing with algorithms that assume normally distributed data, such as many linear models. Unlike the min-max scaling technique, which restricts feature values to a specific range, Z-score normalization represents features in terms of the number of standard deviations they lie away from the mean.It reduces the range from 0 to 85 upto a range of -2 to 4.This technique is particularly useful when dealing with algorithms that assume normally distributed data, such as many linear models.

```{r}
library(caret)
# Convert bank$age to a data frame
age_df <- data.frame(age = bank$age)
# Preprocess the age variable
preproc_age <- preProcess(age_df, method = c("center", "scale"))
# Standardize the age variable
normalized_age <- predict(preproc_age, age_df)
# Summary of the normalized age variable
summary(normalized_age)

bank$ages_zscore <- normalized_age$age
```

Apply decimal scaling to credit score:Decimal scaling normalization is used for the credit score feature in a bank dataset because it allows us to scale the feature values such that the largest absolute value becomes less than 1. In the context of a credit score, which typically ranges from a few hundred to a few thousand, decimal scaling ensures that the scale of the credit score is consistent across different datasets or features. This is important because the absolute magnitude of the credit score matters more than its specific scale.

Normalization affects the credit score by scaling it to a range where the largest absolute value is less than 1. This makes it easier to compare credit scores across different individuals or datasets, as the scale is consistent. Additionally, normalization can help improve the performance of machine learning algorithms that are sensitive to the scale of the features, such as linear models
```{r}
bank$credit.score_decimal <- bank$credit.score / 1000
```

Apply min-max scaling to cont6:Here the data is scaled to a fixed range (usually 0 to 1). The impact is that we end up with smaller standard deviations, which can suppress the effect of outliers. The code is very similar to standardization.
Since it has a lot of outliers, min-max scaling supresses it.

```{r}
# Convert bank$cont6 to a data frame
cont6_df <- data.frame(cont6 = bank$cont6)

# Preprocess the cont6 variable
preproc_cont6 <- preProcess(cont6_df, method = c("range"))

# Apply the min-max scaling to cont6
normalized_cont6 <- predict(preproc_cont6, cont6_df)

# Summary of the normalized cont6 variable
summary(normalized_cont6)

bank$cont6_minmax <-normalized_cont6$cont6
```

Print the first 6 rows of normalized data
```{r}
head(bank[, c("ages", "ages_zscore", "credit.score", "credit.score_decimal", "cont6",
"cont6_minmax")])
```
c. Visualize the new distributions for the variables that have been normalized.
What has changed from the previous visualization?

Apply Z-score normalization to Age:Z-score normalization represents features in terms of the number of standard deviations they lie away from the mean.It reduces the range from 0 to 85 upto a range of -2 to 4.This technique is particularly useful when dealing with algorithms that assume normally distributed data, such as many linear models.
```{r}
ggplot(bank, aes(x = ages_zscore)) +
geom_histogram() +
```

```
labs(title = "Histogram of ages Z score normalization", x = "values", y = "count")
```

Apply decimal scaling to credit score:Decimal scaling normalization is used for the credit score feature in a bank dataset because it allows us to scale the feature values such that the largest absolute value becomes less than 1.Normalization affects the credit score by scaling it to a range where the largest absolute value is less than 1.

```{r}
ggplot(bank, aes(x = credit.score_decimal)) +
geom_histogram(bins = length(unique(bank$credit.score_decimal))) +
labs(title = "Histogram of credit.score- Decimal scaling", x = "values", y = "count")
```

Apply min-max scaling to cont2:It brings the values to a range of -1 to 1.in-max scaling to a range involves converting floating-point feature values from their original range (-30 to 30) into a standardized range, typically between 0 and 1.

```{r}
ggplot(bank, aes(x = cont6_minmax)) +
geom_histogram(bins = length(unique(bank$cont6_minmax))) +
labs(title = "Histogram of cont6 Min Max Normalization", x = "values", y = "count")
```

d. Choose one of the numerical variables to work with for this problem.
Let's call it v. Create a new variable called v_bins that is a binned version of that variable. This v_bins will have a new set of values like low, medium, high. Choose the actual new values (you don't need to use low, medium, high) and the ranges of v that they represent based on your understanding of v from your visualizations. You can use equal depth, equal width or custom ranges.
Explain your choices: why did you choose to create that number of values and those particular ranges?

```{r}
summary(bank$credit.score)
ggplot(bank, aes(y = credit.score)) +
  geom_boxplot() +
  labs(title = "Box Plot of Credit Score", y = "Credit Score")
```

Given that credit.score ranges from 583.7 to 806.0, we can divide it into three bins: low, medium, and high. The ranges for these bins can be defined as follows:

Low: Below the 1st quartile (Q1) value of credit.score (i.e., less than 666.7)
Medium: Between the 1st quartile (Q1) and the 3rd quartile (Q3) values of credit.score (i.e., between 666.7 and 726.4)
High: Above the 3rd quartile (Q3) value of credit.score (i.e., greater than 726.4)
We choose these ranges to evenly distribute the data into three categories based on quartiles, which gives us a good representation of the data distribution. Additionally, using quartiles ensures that each bin contains a similar number of observations, which can help in creating meaningful categories for analysis.

```{r}
library(dplyr)
bank <- bank %>%
  mutate(credit.score_bins = cut(credit.score,
                                 breaks=c(-Inf, 666.7, 726.4, Inf),
                                 labels=c("low", "medium", "high")))
head(bank$credit.score_bins)
```

e. Building on (d), use v_bins to create a smoothed version of v. Choose a smoothing strategy to create a numerical version of the binned variable and explain your choices.

Create Bins: Use the cut function to create equal width bins for the credit.score variable and store the result in a new column credit.score_bins.
Calculate Mean for Each Bin: Calculate the mean value of credit.score for each bin (low,

medium, high) and store the result in separate data frames for each bin.
Replace Original Values: Replace the original credit.score values with the mean value of each bin for the corresponding credit.score_bins.
Combine Results: Combine the separate data frames for each bin into a single data frame to get the smoothed version of credit.score_bins.

```{r}
# Create bins for credit score
bank <- bank %>%
  mutate(credit.score_bins = cut(credit.score, breaks = 3,
                                 labels = c("low", "medium", "high")))

# Calculate mean value for each bin and replace the original
#credit score with the mean
low <- bank %>%
  filter(credit.score_bins == 'low') %>%
  mutate(credit.score = mean(credit.score, na.rm = TRUE))

medium <- bank %>%
  filter(credit.score_bins == 'medium') %>%
  mutate(credit.score = mean(credit.score, na.rm = TRUE))

high <- bank %>%
  filter(credit.score_bins == 'high') %>%
  mutate(credit.score = mean(credit.score, na.rm = TRUE))

# Combine the results
smoothed_bank <- bind_rows(list(low, medium, high))
bind_rows(list(low, medium, high))

```

Problem 2

This is the first homework problem using machine learning algorithms. You will perform a straightforward training and evaluation of a support vector machine on the bank data from Problem 1. Start with a fresh copy, but be sure to remove rows with missing values first.

```{r}
bank <-read.csv("/Users/kavanamanvi/Desktop/FDS/HW2/BankData.csv")
bank <- na.omit(bank)
summary(bank)
```

Pre-processing: since SVM only works with numerical data, convert boolean values to 0 and 1 instead of t and f.

```{r}
# Convert boolean values to 0 and 1
bank$bool1 <- as.integer(bank$bool1 == "t")
bank$bool2 <- as.integer(bank$bool2 == "t")
bank$bool3 <- as.integer(bank$bool3 == "t")

# Print the updated dataset
head(bank)

```
a. Apply SVM to the data from Problem 1 to predict approval and report the accuracy using 10-fold cross validation.

```{r}
library(caret)
library(e1071)
train_control_cv = trainControl(method = "cv", number = 10)
svm_bank <- train(approval ~., data = bank, method = "svmLinear",
               trControl = train_control_cv)
svm_bank
```

```

b. Next, use the grid search functionality when training to optimize the C parameter of the SVM. What parameter was chosen and what is the accuracy?

The grid search functionality was used to optimize the C parameter of the SVM. The grid search tested a range of C values from 10^-5 to 10^2, with a step size of 0.5. The best C parameter chosen was 0.01, which resulted in an accuracy of approximately 86.35%. This means that the SVM model with a C parameter of 0.01 achieved the highest accuracy compared to other C values tested during the grid search.

```{r}
grid <- expand.grid(C = 10^seq(-5,2,0.5))
svm_grid <- train(approval ~., data = bank, method = "svmLinear",
                  trControl = train_control_cv, tuneGrid = grid)
svm_grid
```

c. Sometimes even if the grid of parameters in (b) includes the default value of C = 1 (used in (a)), the accuracy result will be different for this value of C. What could make that different?

The grid search in (a) explored a range of C values from 10^-5 to 10^2, with a step size of 0.5, and found that the optimal C value was 0.01, resulting in an accuracy of approximately 86.35%.

In contrast, the grid search in (b) used a fixed value of C = 1 and achieved an accuracy of approximately 86.33%.

The difference in accuracy between the two cases could be due to the nature of the dataset and the performance characteristics of the SVM algorithm. Grid search helps find the optimal C value by evaluating the model's performance across different values of C. Even though C = 1 is the default value, it may not always be the best choice for a given dataset. Factors such as the complexity of the dataset, the presence of outliers, and the distribution of the classes can all influence the performance of the SVM model with different values of C. As a result, the grid search may find a different optimal C value that better suits the characteristics of the dataset, leading to a different accuracy result compared to using the default value of C = 1.

```{r}
grid <- expand.grid(C = 1)
svm_grid <- train(approval ~., data = bank, method = "svmLinear",
                   trControl = train_control_cv, tuneGrid = grid)
svm_grid
```

Problem 3

We will take SVM further in this problem, showing how it often gets used even when the data are not suitable, by first engineering the numerical features we need. There is a Star Wars dataset in the dplyr library. Load that library and you will be able to see it (head(starwars)). There are some variables we will not use, so first remove films, vehicles, starships and name. Also remove rows with missing values.

```{r}
library(dplyr)
data(starwars)
starwars <- starwars %>% select(-c("films", "vehicles", "starships", "name"))
names(starwars)
#remove missing coloumns
starwars<- na.omit(starwars)
summary(starwars)
```
```

```

a. Several variables are categorical. We will use dummy variables to make it possible for SVM to use these. Leave the gender category out of the dummy
variable conversion to use as a categorical for prediction. Show the resulting head.
Convert categorical variables to dummy variables, leaving out 'gender'

```{r}
library(caret)

# Selecting only the categorical variables
categorical_vars <- c("hair_color", "skin_color", "eye_color", "homeworld",
"species","sex")

# Creating dummy variables, excluding 'gender'
dummy <- dummyVars(~ ., data = starwars[, categorical_vars], levelsOnly = TRUE)

# Using the dummy predictor to transform the dataset
dummies <- as.data.frame(predict(dummy, newdata = starwars[, categorical_vars]))

# Combining the dummy variables with the original dataset
starwars_dummies <- cbind(starwars, dummies)

# Removing the original categorical variables
starwars_dummies <- starwars_dummies[, -which(names(starwars_dummies) %in%
categorical_vars)]

head(starwars_dummies)

```


b. Use SVM to predict gender and report the accuracy.

```{r}
train_control_loocv = trainControl(method = "LOOCV", number = 10)
svm_starwars <- train(gender ~., data = starwars_dummies, method = "svmLinear",
              trControl = train_control_loocv)
svm_starwars
```


c. Given that we have so many variables, it makes sense to consider using PCA. Run PCA on the data
and determine an appropriate number of components to use. Document how you made the decision, including any graphs you used. Create a reduced version of the data with that number of principle components. Note: make sure to remove gender from the data before running PCA because it would be cheating if PCA had access to the label you will use. Add it back in after reducing
the data and show the result.

PCA only works with numerical values
To get a list of potential predictors as a set of column indices we can use nearZeroVar function.
```{r}
nzv <- nearZeroVar(starwars_dummies)
length(nzv)

```

```{r}
#Remove gender column
starwars_dummies_no_gender <- starwars_dummies %>% select(-c("gender"))
```
The Caret library version of the PCA requires an exact number components as a parameter. To determine this number we can use summary statistics and various visualizations available for the base R principal component function prcomp. Here the summary will give
```

you the cumulative proportions of the PCs. Scree plot will give you similar information visually.

```{r}
starwars.pca <- prcomp(starwars_dummies_no_gender)
summary(starwars.pca)
# Visualize the scree plot
screeplot(starwars.pca, type = "l") + title(xlab = "PCs")
```
Looking at these results we can say that we capture most of the variance by using 3 principal components at +95% variance. Now, we can use 3 PCs to model our data. Caret package preProcess function here can give you a new dataset with the PCs instead of the predictors.

```{r}
target <- starwars %>% dplyr::select(gender)
preProc <- preProcess(starwars_dummies, method="pca", pcaComp=3)
starwars.pc <- predict(preProc, starwars_dummies)
starwars.pc$gender <- starwars$gender
head(starwars.pc)
```

d. Use SVM to predict gender again, but this time use the data resulting from PCA. Evaluate the results
with a confusion matrix and at least two partitioning methods, using grid search on the C parameter
each time.

```{r}
library(caret)
library(e1071)

set.seed(123)
index = createDataPartition(y=starwars.pc$gender, p=0.7, list=FALSE)
train_set = starwars.pc[index,]
test_set = starwars.pc[-index,]
svm_split <- train(gender ~., data = train_set, method = "svmLinear")
pred_split <- predict(svm_split, test_set)
sum(pred_split == test_set$gender) / nrow(test_set)

#Confusion matrix:
test_set$Sgender <- pred_split

# Convert the Sgender column to a factor with levels from pred_split
test_set$Sgender <- factor(test_set$Sgender, levels = levels(pred_split))

confusionMatrix(test_set$Sgender, pred_split)

```
grid search:
```{r}
train_control = trainControl(method = "cv", number = 10)
grid <- expand.grid(C = 10^seq(-5,2,0.5))

# Fit the model
svm_grid <- train(gender ~., data = starwars.pc, method = "svmLinear",
              trControl = train_control, tuneGrid = grid)
# View grid search result
svm_grid
```

e. Whether or not it has improved the accuracy, what has PCA done for the complexity of the model?

PCA has reduced the complexity of the model by reducing the number of predictors from 66 to 3 while maintaining a high level of accuracy. The original dataset
had 66 predictors, which could lead to overfitting and increased computational complexity. By using PCA to reduce the dimensionality to 3 principal components, we have

a simpler model that still explains over 95% of the variance in the
data. This reduction in complexity can lead to a more interpretable model and
potentially better generalization to new data.

Problem 4

Use the Sacremento data from the caret library by running data(Sacremento) after loading caret. This data is about housing prices in Sacramento, California. Remove the zip and city variables.
a. Explore the variables to see if they have reasonable distributions and show your work. We will be predicting the type variable — does that mean we have a class imbalance?

Exploring variables

```{r}
data("Sacramento")
library(ggplot2)
ggplot(Sacramento, aes(x=city)) + geom_bar()+ggtitle("Bar Graph of City")
ggplot(Sacramento, aes(x=zip)) + geom_bar()+ggtitle("Bar Graph of zip codes")
ggplot(Sacramento, aes(x=beds)) + geom_bar()+ggtitle("Bar Graph of Beds")
ggplot(Sacramento, aes(x=baths)) + geom_bar()+ggtitle("Bar Graph of Bath")
ggplot(Sacramento, aes(sqft)) + geom_histogram(binwidth = 60)+ggtitle("Histogram of sqft")
ggplot(Sacramento, aes(price)) + geom_histogram(binwidth = 60)+ggtitle("Histogram of price")
ggplot(Sacramento, aes(x = longitude, y = latitude)) + geom_bin2d() + labs(x = "Longitude", y = "Latitude", title = "Heatmap of Latitude and Longitude")
```

Class imbalance: In the type variable-Residential has more values than condo and multi-family

```{r}
ggplot(Sacramento, aes(x=type)) + geom_bar()+ggtitle("Bar Graph of Types")
```

b. There are lots of options for working on the data to try to improve the performance of SVM, including (1) removing other variables that you know should not be part of the prediction, (2) dealing with extreme variations in some variables with smoothing, normalization or a log transform, (3) applying PCA, and (4) to removing outliers. Pick one now and continue.

(1) removing other variables that you know should not be part of the prediction
From the Sacramento dataset, it seems like the variables "city", "zip", "latitude", and "longitude" are not directly related to the prediction of the property type ("type"). These variables are more about the location and identification of the property, which might not be relevant for predicting its type. Therefore, I would consider removing these variables if they are not required for my prediction task.

```{r}
Sacramento<- na.omit(Sacramento)
Sacramento_cleaned <- Sacramento %>% select(-c("city", "zip", "latitude","longitude"))
head(Sacramento_cleaned)
```

c. Use SVM to predict type and use grid search to get the best accuracy you can. The accuracy may be good, but look at the confusion matrix as well. Report what you find. Note that the kappa value provided with your SVM results can also help you see this. It is a measure of how well the classifier performed that takes
into account the frequency of the classes.

```{r}
# Assuming 'Sacramento_cleaned' is your dataset

set.seed(123)
index = createDataPartition(y = Sacramento_cleaned$type, p = 0.7, list = FALSE)
```

```
train_set = Sacramento_cleaned[index, ]
test_set = Sacramento_cleaned[-index, ]

# Fit the SVM model using the training set
svm_model <- train(type ~., data = train_set, method = "svmLinear")

# Make predictions on the test set
pred <- predict(svm_model, test_set)

# Calculate accuracy
accuracy <- sum(pred == test_set$type) / nrow(test_set)
accuracy

# Generate confusion matrix
confusionMatrix(test_set$type, pred)
```

The overall accuracy of the model is 0.935, indicating that it correctly predicts the property type 93.5% of the time.
The model shows a high specificity for predicting Condo (0.94585) and Multi_Family (0.98917) properties, meaning it is good at correctly identifying these types when they are the actual class.
However, there is an issue with class imbalance, particularly with the Condo and Multi_Family classes, where the model fails to predict any instances correctly. This imbalance can skew the accuracy metrics and make it seem like the model performs better than it actually does.
For the Residential class, the sensitivity (true positive rate) is 0.935, indicating that the model is fairly good at identifying Residential properties when they are present.
It's important to note that the Precision, Recall, and F1-score metrics are not calculated due to the class imbalance issue.
Overall, while the model shows good accuracy, it may not be reliable for predicting Condo and Multi_Family properties due to the class imbalance. Further adjustments, such as resampling techniques or using different algorithms, may be necessary to improve the model's performance for these classes.

Grid search:

```{r}
train_control = trainControl(method = "cv", number = 10)
grid <- expand.grid(C = 10^seq(-5,2,0.5))

# Fit the model
svm_grid_sc <- train(type ~., data = Sacramento_cleaned, method = "svmLinear",
                     trControl = train_control, tuneGrid = grid)
 # View grid search result
 svm_grid_sc
```

d. Return to (b) and try at least one other way to try to improve the data before running SVM again, as in (c).

Apply PCA:

```{r}
nzv <- nearZeroVar(Sacramento)
length(nzv)
#remove type from Sacramento
sac_no_type <- Sacramento_cleaned %>% select(-c("type"))
#apply pca
sac.pca <- prcomp(sac_no_type)
summary(sac.pca)
#scree plot
screeplot(sac.pca, type = "l") + title(xlab = "PCs")
```

Looking at these results we can say that we capture most of the variance by using 1

principal component at +95% variance. Now, we can use 1 PC to model our data. Caret package preProcess function here can give you a new dataset with the PCs instead of the predictors.

```{r}
target <- Sacramento %>% dplyr::select(type)
preProc <- preProcess(Sacramento_cleaned, method="pca", pcaComp=1)
sac.pc <- predict(preProc, Sacramento_cleaned)
# Put back target column
sac.pc$type <- Sacramento$type
head(sac.pc)

```

SVM:

```{r}
set.seed(123)
index = createDataPartition(y=sac.pc$type, p=0.7, list=FALSE)
train_set = sac.pc[index,]
test_set = sac.pc[-index,]
svm_split <- train(type ~., data = train_set, method = "svmLinear")
pred_split <- predict(svm_split, test_set)
sum(pred_split == test_set$type) / nrow(test_set)

#Confusion matrix:
test_set$Stype <- pred_split

# Convert the Stype column to a factor with levels from pred_split
test_set$Stype <- factor(test_set$Stype, levels = levels(pred_split))

confusionMatrix(test_set$Stype, pred_split)

```

In the confusion matrix generated for the prediction of property types in the Sacramento dataset, there is a significant class imbalance. The dataset consists of 277 instances of residential properties and no instances of condos or multi-family properties in the test set. This class imbalance can lead to challenges in model evaluation, as the model may perform well in predicting the majority class (residential) but may struggle with the minority classes (condo and multi-family).

e. In the end, some data are just so imbalanced that a classifier is never going to predict the minority class. Dealing with this is a huge topic. One simple possibility is to conclude that we do not have enough data to support predicting the very infrequent class(es) and remove them. If they are not actually important to the reason we are making the prediction, that could be fine. Another approach is to force the data to be more even by sampling.

Create a copy of the data that includes all the data from the two smaller classes, plus a small random sample of the large class (you can do this by separating those data with a filter, sampling, then attaching them back on). Check the distributions of the variables in this new data sample to make sure they are reasonably close to the originals using visualization and/or summary statistics. We want to make sure we did not get a strange sample where everything was cheap or there were only studio apartments, for example. You can rerun the sampling a few times if you are getting strange results. If it keeps happening, check your process.
Use SVM to predict type one this new, more balanced dataset and report its performance with a confusion matrix and with grid search to get the best accuracy.

```{r}
# Filter the data
condo_data <- Sacramento_cleaned %>% filter(type == "Condo")
multi_family_data <- Sacramento_cleaned %>% filter(type == "Multi_Family")
residential_data <- Sacramento_cleaned %>% filter(type == "Residential")

# Sample the residential_data
set.seed(123) # for reproducibility
sample_residential_data <- residential_data %>% sample_n(size = nrow(condo_data) + nrow(multi_family_data))
```

```
# Combine the sampled residential_data with condo_data and multi_family_data
balanced_data <- rbind(condo_data, multi_family_data, sample_residential_data)

# Check the distributions
summary(balanced_data)

```

SVM to predict the class
```{r}
# Split the data into training and testing sets
index <- createDataPartition(y = balanced_data$type, p = 0.7, list = FALSE)
train_set <- balanced_data[index, ]
test_set <- balanced_data[-index, ]

# Train the SVM model
svm_model <- train(type ~ ., data = train_set, method = "svmLinear")

# Predict on the test set
pred <- predict(svm_model, test_set)

# Confusion matrix
confusionMatrix(test_set$type, pred)

# Grid search for best accuracy
grid <- expand.grid(C = 10^seq(-5, 2, 0.5))
svm_grid <- train(type ~ ., data = train_set, method = "svmLinear",
                  trControl = trainControl(method = "cv", number = 5),
                  tuneGrid = grid)

# Best model
best_model <- svm_grid$bestTune
best_model

```

Problem 5

To understand just how much different subsets can differ, create a 5 fold partitioning of the cars data included in R (mtcars) and visualize the distribution of the gears variable across the folds. Rather than use the fancy trainControl methods for making the folds, create them directly so you actually can keep track of which data points are in which fold. This is not covered in the tutorial, but it is quick. Here is code to create 5 folds and a variable in the data frame that contains the fold index of each point. Use that resulting data frame to create your visualization.
```
mycars <- mtcars # make a copy to modify
mycars$folds = 0 # initialize new variable to hold fold indices
# Create 5 folds, get a list of lists of indices.
# Take a look at this result so you understand what is happening.
# Note we are not passing the data frame directly, but a list of its
indices created by 1:nrow(mycars). If you don't understand how
that works, try the individual parts on their own first
flds = createFolds(1:nrow(mycars), k=5, list=TRUE)
# This loop sets all the rows in a given fold to have that fold's
index in the folds variable. Take a look at the result and use it
        to create the visualization.
      for (i in 1:5) { mycars$folds[flds[[i]]] = i}
```

```{r}
# Load the mtcars dataset
data(mtcars)

# Make a copy of the dataset
mycars <- mtcars
mycars$folds <- 0

# Create 5 folds for cross-validation
flds <- createFolds(1:nrow(mycars), k = 5, list = TRUE)
```

```
# Assign fold indices to the folds variable
for (i in 1:5) {
  mycars$folds[flds[[i]]] <- i
}

# Visualize the distribution of the gear variable across folds
library(ggplot2)
ggplot(mycars, aes(x = factor(folds), fill = factor(gear))) +
  geom_bar(position = "dodge") +
  labs(x = "Fold", y = "Count", fill = "Gear") +
  ggtitle("Distribution of Gear Variable Across Folds")
```