DMAW_MIDTERMS_KHAFAJI

Exploring Customer Churn

Our data set involves data relevant to customer churn. It has the following columns:

- CustomerID: The customer's ID
- Gender: the customer's gender
- SeniorCitizen: if the customer is a senior citizen
- Partner: If the customer is a partner
- Dependents: if the customer has dependents
- Tenure: The tenure of the customer
- PhoneService: If the customer has phone service
- InternetService: The internet service of the customer, if they have one
- Contract: Their contract type
- MonthlyCharges: Their monthly charge for the service
- TotalCharges: Total charged amount
- Churn: If customer left the service or not.

Data Mining

loading data

First, let's load our data set into a variable called cust_churn

```
cust_churn <- read_csv("customer_churn.csv")
```

```
## Rows: 10000 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (8): CustomerID, Gender, Partner, Dependents, PhoneService, InternetServ...
## dbl (4): SeniorCitizen, Tenure, MonthlyCharges, TotalCharges
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(cust_churn)
```

```
## # A tibble: 6 x 12
##   CustomerID Gender SeniorCitizen Partner Dependents Tenure PhoneService
##   <chr>      <chr>          <dbl> <chr>   <chr>       <dbl> <chr>
## 1 CUST00001  Male               0 No      No             65 Yes
## 2 CUST00002  Male               0 No      No             26 Yes
## 3 CUST00003  Male               0 Yes     No             54 Yes
## 4 CUST00004  Female             0 Yes     Yes            70 Yes
## 5 CUST00005  Male               0 No      No             53 Yes
## 6 CUST00006  Female             0 No      Yes            45 Yes
## # i 5 more variables: InternetService <chr>, Contract <chr>,
## #   MonthlyCharges <dbl>, TotalCharges <dbl>, Churn <chr>
```

now, let's get the summary of the dataset:

```r
cat("The structure of the data set is:\n")
```

```
## The structure of the data set is:
```

```r
print(str(cust_churn))
```

```
## spc_tbl_ [10,000 x 12] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ CustomerID     : chr [1:10000] "CUST00001" "CUST00002" "CUST00003" "CUST00004" ...
##  $ Gender         : chr [1:10000] "Male" "Male" "Male" "Female" ...
##  $ SeniorCitizen  : num [1:10000] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Partner        : chr [1:10000] "No" "No" "Yes" "Yes" ...
##  $ Dependents     : chr [1:10000] "No" "No" "No" "Yes" ...
##  $ Tenure         : num [1:10000] 65 26 54 70 53 45 35 20 48 33 ...
##  $ PhoneService   : chr [1:10000] "Yes" "Yes" "Yes" "Yes" ...
##  $ InternetService: chr [1:10000] "Fiber optic" "Fiber optic" "Fiber optic" "DSL" ...
##  $ Contract       : chr [1:10000] "Month-to-month" "Month-to-month" "Month-to-month" "One year" ...
##  $ MonthlyCharges : num [1:10000] 20 65.1 49.4 31.2 103.9 ...
##  $ TotalCharges   : num [1:10000] 1303 1694 2667 2183 5505 ...
##  $ Churn          : chr [1:10000] "No" "No" "No" "No" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   CustomerID = col_character(),
##   ..   Gender = col_character(),
##   ..   SeniorCitizen = col_double(),
##   ..   Partner = col_character(),
##   ..   Dependents = col_character(),
##   ..   Tenure = col_double(),
##   ..   PhoneService = col_character(),
##   ..   InternetService = col_character(),
##   ..   Contract = col_character(),
##   ..   MonthlyCharges = col_double(),
##   ..   TotalCharges = col_double(),
##   ..   Churn = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
## NULL
```

```r
#cat("The summary of the data set is:\n")
#summary(cust_churn)

cat("\n\nThe Summary Statistics for the data set are:\n")
```

```
##
##
## The Summary Statistics for the data set are:
```

```r
skimr::skim(cust_churn)
```

Table 1: Data summary

Name	cust_churn
Number of rows	10000
Number of columns	12

Column type frequency:

character	8
numeric	4
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
CustomerID	0	1	9	9	0	10000	0
Gender	0	1	4	6	0	2	0
Partner	0	1	2	3	0	2	0
Dependents	0	1	2	3	0	2	0
PhoneService	0	1	2	3	0	2	0
InternetService	0	1	2	11	0	3	0
Contract	0	1	8	14	0	3	0
Churn	0	1	2	3	0	2	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
SeniorCitizen	0	1	0.15	0.36	0.00	0.00	0.00	0.00	1.00	
Tenure	0	1	35.22	20.79	0.00	17.00	35.00	53.00	71.00	
MonthlyCharges	0	1	70.18	29.03	20.02	44.88	70.56	95.77	119.99	
TotalCharges	0	1	2455.81	1854.59	0.00	961.21	2025.58	3610.98	8425.57	

```r
cat("\n\n")
```

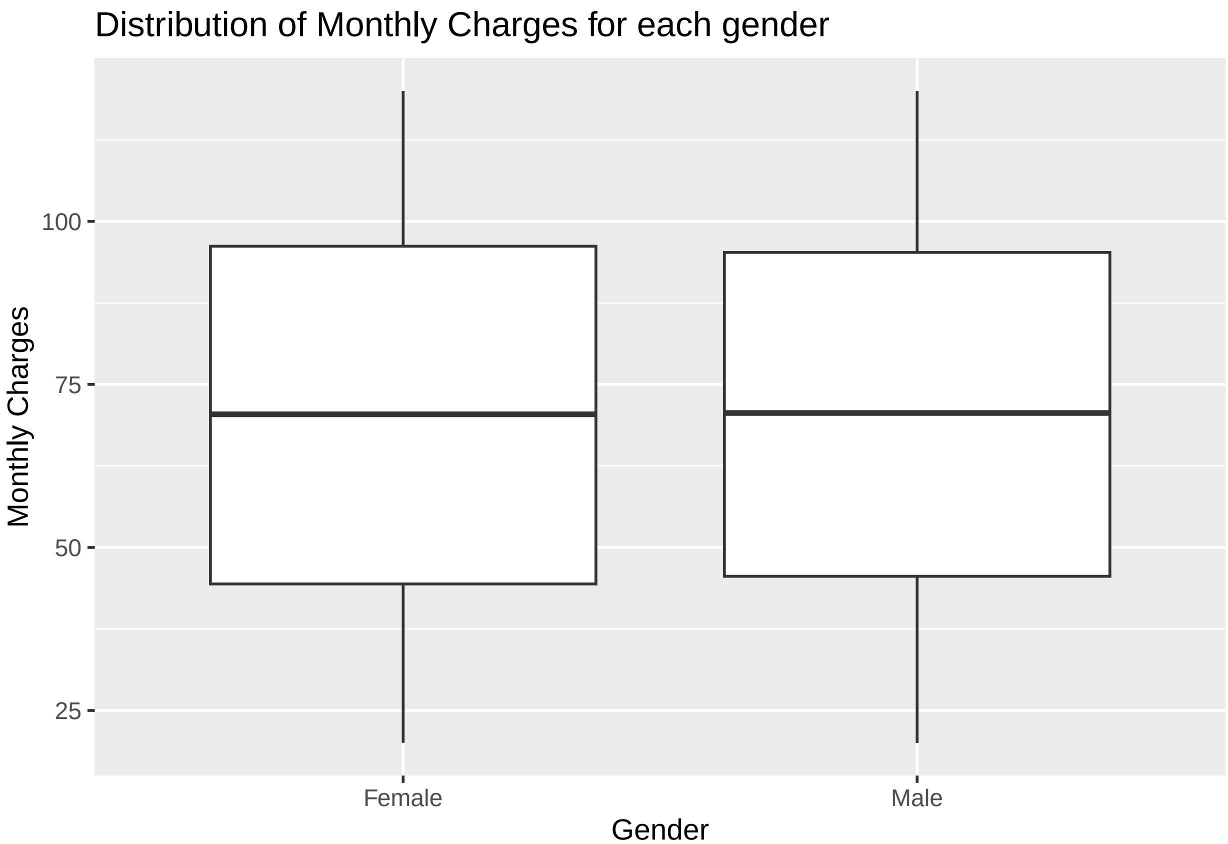
We can see from our summary that we have 1 ID table, 7 categorical variables, and three numerical variables. We also have no missing values.

Data Visualization

Let's take a quick look at what our data says.

Let's take a look at monthly charges per gender.

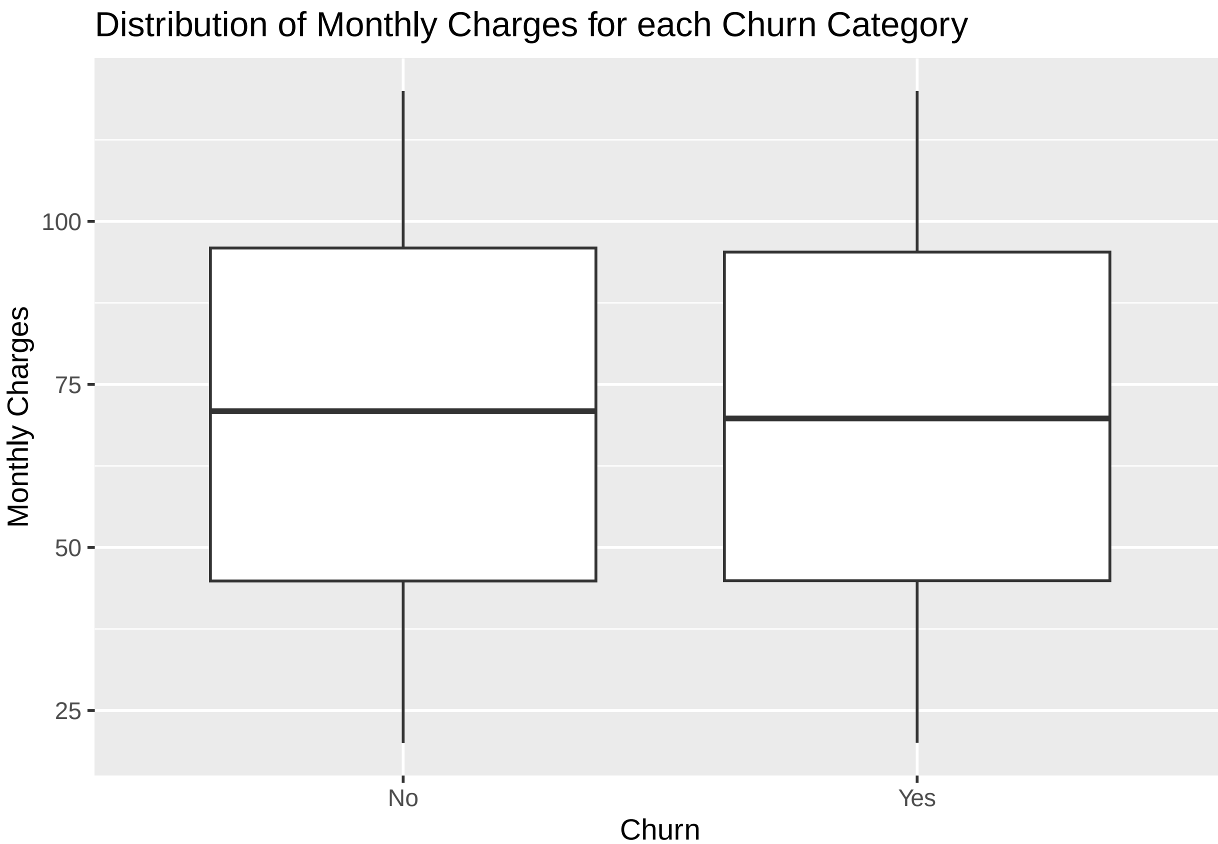
```r
cust_churn %>% ggplot(aes(x=Gender, y=MonthlyCharges)) +
  geom_boxplot()+
  labs(title="Distribution of Monthly Charges for each gender", y="Monthly Charges", x= "Gender")
```



We can see that the median monthly charges for each gender is roughly equal, if not slightly less for female customers. However, the interquartile range of the monthly charges for females is slightly larger than for their male counterparts.

Next, let's look at the distribution of monthly charges for each churn category.

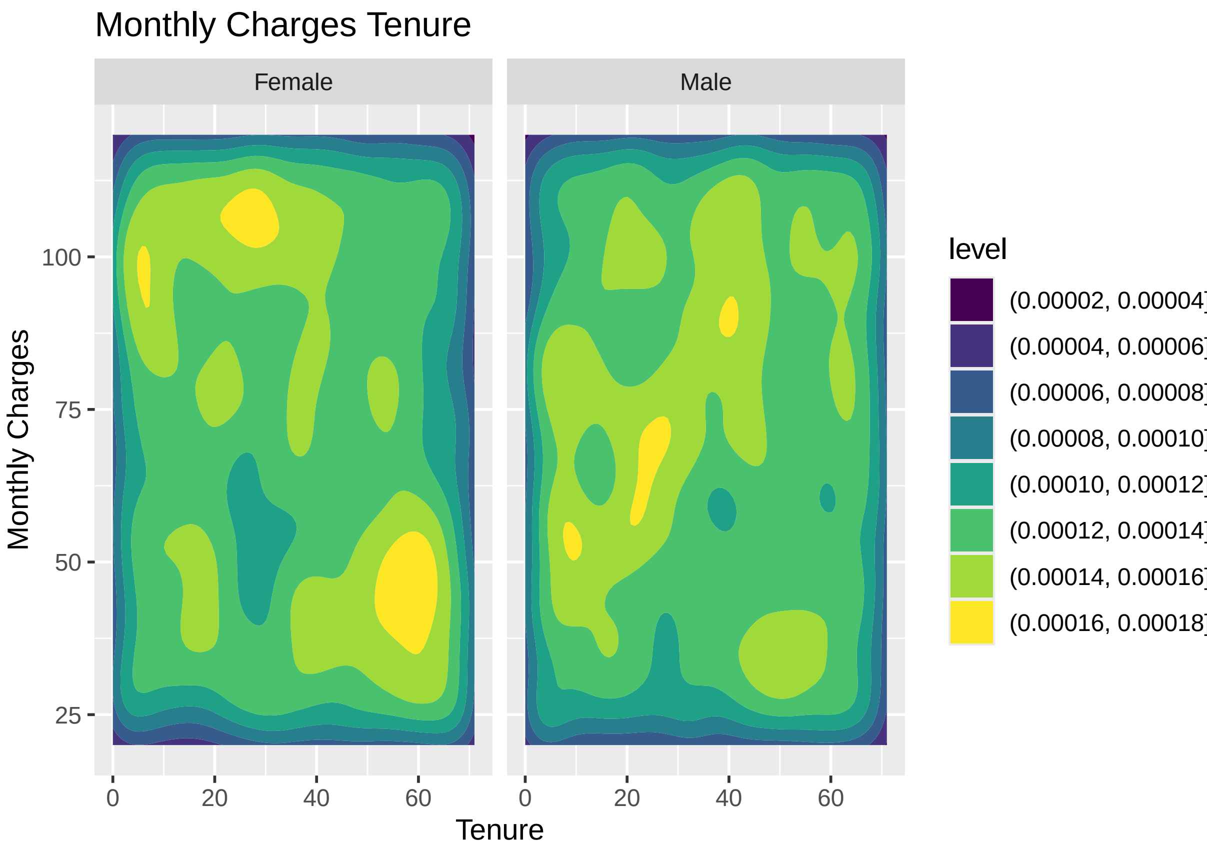
```
cust_churn %>% ggplot(aes(x=Churn, y=MonthlyCharges)) +
  geom_boxplot()+
  labs(title="Distribution of Monthly Charges for each Churn Category", y="Monthly Charges", x= "Churn")
```



We can see that the median monthly charges for churned customer is slightly less than current customers. This also follows for the 1st and 3rd quartile.

Lastly, let's look at monthly charges for vs tenure, faceted by gender.

```
cust_churn %>% ggplot(aes(x=Tenure, y=MonthlyCharges)) +
  geom_density2d_filled()+
  facet_wrap(~Gender)+
  labs(title="Monthly Charges Tenure", y="Monthly Charges", x= "Tenure")
```



The 2d Density plot shows us that, while the distribution isn't really uniform, The monthly charges for female customers rises with their tenure. In contrast, for males, their monthly charge is relatively higher for customers that has short tenures, lower for customers with medium or long length of tenures.

Data Transformation

Now, since we don't have missing values, all we have to do is convert categorical variables to factor variables. We would also normalize or standardize numerical features, if necessary.

let's first convert our categorical variables to factor variables:

```
cust_churn <- cust_churn %>% mutate( SeniorCitizen = case_when(
                                                                SeniorCitizen == 0 ~ "Not Senior",
                                                                SeniorCitizen == 1 ~ "Senior Citizen"
                                                                )) %>%
  mutate_at( c('Gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService', 'InternetService', '(

str(cust_churn)
```

```
## tibble [10,000 x 12] (S3: tbl_df/tbl/data.frame)
##  $ CustomerID      : chr [1:10000] "CUST00001" "CUST00002" "CUST00003" "CUST00004" ...
##  $ Gender          : Factor w/ 2 levels "Female","Male": 2 2 2 1 2 1 1 1 1 1 ...
##  $ SeniorCitizen   : Factor w/ 2 levels "Not Senior","Senior Citizen": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Partner         : Factor w/ 2 levels "No","Yes": 1 1 2 2 1 1 2 2 2 1 ...
##  $ Dependents      : Factor w/ 2 levels "No","Yes": 1 1 1 2 1 2 1 2 2 1 ...
##  $ Tenure          : num [1:10000] 65 26 54 70 53 45 35 20 48 33 ...
##  $ PhoneService    : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 1 2 ...
```

```
##  $ InternetService: Factor w/ 3 levels "DSL","Fiber optic",..: 2 2 2 1 1 2 3 2 3 3 ...
##  $ Contract       : Factor w/ 3 levels "Month-to-month",..: 1 1 1 2 1 1 2 1 1 3 ...
##  $ MonthlyCharges : num [1:10000] 20 65.1 49.4 31.2 103.9 ...
##  $ TotalCharges   : num [1:10000] 1303 1694 2667 2183 5505 ...
##  $ Churn          : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 1 1 ...
```

Now, let's normalize our numeric data:

```
cust_churn_norm <- cust_churn %>% mutate(across(where(is.numeric), ~ as.numeric(scale(.x))))

head(cust_churn_norm)
```

```
## # A tibble: 6 x 12
##   CustomerID Gender SeniorCitizen Partner Dependents Tenure PhoneService
##   <chr>      <fct>  <fct>         <fct>   <fct>       <dbl> <fct>
## 1 CUST00001  Male   Not Senior    No      No          1.43  Yes
## 2 CUST00002  Male   Not Senior    No      No         -0.444 Yes
## 3 CUST00003  Male   Not Senior    Yes     No          0.903 Yes
## 4 CUST00004  Female Not Senior    Yes     Yes         1.67  Yes
## 5 CUST00005  Male   Not Senior    No      No          0.855 Yes
## 6 CUST00006  Female Not Senior    No      Yes         0.470 Yes
## # i 5 more variables: InternetService <fct>, Contract <fct>,
## #   MonthlyCharges <dbl>, TotalCharges <dbl>, Churn <fct>
```

As we can see, our numeric data is now z-score normalized.

Data Wrangling

Since we have a relatively huge data set, let's filter outliers, if our data has them. Since we have normalized our numerical data, we can simply filter if the absolute value of their z-score is greater than 3.

```
cust_churn_norm <- cust_churn_norm %>%
  dplyr::filter(across(is.numeric, ~ abs(.x) < 3))
```

```
## Warning: Using `across()` in `filter()` was deprecated in dplyr 1.0.8.
## i Please use `if_any()` or `if_all()` instead.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: There was 1 warning in `dplyr::filter()`.
## i In argument: `across(is.numeric, ~abs(.x) < 3)`.
## Caused by warning:
## ! Use of bare predicate functions was deprecated in tidyselect 1.1.0.
## i Please use wrap predicates in `where()` instead.
##   # Was:
##   data %>% select(is.numeric)
##
##   # Now:
##   data %>% select(where(is.numeric))
```

```
cust_churn_norm
```

```
## # A tibble: 9,972 x 12
##    CustomerID Gender SeniorCitizen Partner Dependents  Tenure PhoneService
##    <chr>      <fct>  <fct>         <fct>   <fct>        <dbl> <fct>
##  1 CUST00001  Male   Not Senior    No      No           1.43   Yes
##  2 CUST00002  Male   Not Senior    No      No          -0.444  Yes
```

```
##   3 CUST00003  Male   Not Senior    Yes      No           0.903   Yes
##   4 CUST00004  Female Not Senior    Yes      Yes          1.67    Yes
##   5 CUST00005  Male   Not Senior    No       No           0.855   Yes
##   6 CUST00006  Female Not Senior    No       Yes          0.470   Yes
##   7 CUST00007  Female Not Senior    Yes      No          -0.0106  Yes
##   8 CUST00008  Female Not Senior    Yes      Yes         -0.732   Yes
##   9 CUST00009  Female Not Senior    Yes      Yes          0.615   No
##  10 CUST00010  Female Not Senior    No       No          -0.107   Yes
## # i 9,962 more rows
## # i 5 more variables: InternetService <fct>, Contract <fct>,
## #   MonthlyCharges <dbl>, TotalCharges <dbl>, Churn <fct>
```

Review

In this chapter, we simply loaded up the data, and cleaned/transformed it so that it is much better suited for use in machine learning. Some of the things we did is transformed categorical variables into an R factor data type, which would make it easier for the machine to use. We also Z-score normalized our data, and removed outliers (removing data points with a z-score of > 3).

We also explored the distribution of the Monthly Charges for gender and churn category, finding minimal differences. We also found that the monthly charges differed for males and females with regards to tenure.

Tuning Predictive Models

Model Complexity

Now, let's try fitting a decision tree and a logistic regression model to our data set.

Let's first double check our data:

```r
cust_churn_norm_features <- cust_churn_norm %>% dplyr::select(-CustomerID)

vars_to_check <- cust_churn_norm_features %>%
  dplyr::select(where(is.factor) ) %>%
  dplyr::select(-Churn) %>% names()

for (var in vars_to_check) {

  formula <- as.formula(paste("~ Churn +", var))

  tbl <- xtabs(formula, data = cust_churn_norm_features)

  print(tbl)
}
```

```
##      Gender
## Churn Female Male
##   No    3663 3614
##   Yes   1402 1293
##      SeniorCitizen
## Churn Not Senior Senior Citizen
##   No        6175           1102
##   Yes       2298            397
##      Partner
```

```
## Churn   No  Yes
##   No  3615 3662
##   Yes 1368 1327
##      Dependents
## Churn   No  Yes
##   No  5082 2195
##   Yes 1913  782
##      PhoneService
## Churn   No  Yes
##   No   685 6592
##   Yes  277 2418
##      InternetService
## Churn  DSL Fiber optic   No
##   No  2888        2926 1463
##   Yes 1086        1100  509
##      Contract
## Churn Month-to-month One year Two year
##   No            4320     1514     1443
##   Yes           1639      494      562
```

As we can see, we have data points in all of the intersections of churn and other factor variables. However, what's worrying is the intersection of "yes" in churn and Senior Citizens. But let's find out later if it will be a problem.

Let's first create the splits:

```r
set.seed(123)

trainIndex <- createDataPartition(cust_churn_norm_features$Churn, p = .8,
                                  list = FALSE,
                                  times = 1)

churnTrain <- cust_churn_norm_features[ trainIndex,]
churnTest  <- cust_churn_norm_features[-trainIndex,]
```

Let's now create our logistic regression.

```r
model_logistic <- glm(formula = Churn ~ ., family = "binomial", data=cust_churn_norm_features)

summary(model_logistic)
```

```
##
## Call:
## glm(formula = Churn ~ ., family = "binomial", data = cust_churn_norm_features)
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 -0.783674   0.086842  -9.024  < 2e-16 ***
## GenderMale                  -0.069111   0.045218  -1.528  0.12642
## SeniorCitizenSenior Citizen -0.032045   0.063528  -0.504  0.61396
## PartnerYes                  -0.042498   0.045173  -0.941  0.34682
## DependentsYes               -0.059519   0.049621  -1.199  0.23035
## Tenure                       0.069629   0.058866   1.183  0.23687
## PhoneServiceYes             -0.102348   0.075195  -1.361  0.17348
## InternetServiceFiber optic   0.001558   0.050266   0.031  0.97527
## InternetServiceNo           -0.081882   0.062681  -1.306  0.19144
```

```
## ContractOne year            -0.155395   0.059461  -2.613  0.00897 **
## ContractTwo year             0.025314   0.057619   0.439  0.66042
## MonthlyCharges               0.026633   0.045000   0.592  0.55395
## TotalCharges                -0.079574   0.070277  -1.132  0.25751
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 11638  on 9971  degrees of freedom
## Residual deviance: 11619  on 9959  degrees of freedom
## AIC: 11645
##
## Number of Fisher Scoring iterations: 4
```

```r
ll.null <- model_logistic$null.deviance/-2
ll.proposed <- model_logistic$deviance/-2

cat("\n\nThe Psuedo R^2 is:", (ll.null-ll.proposed)/ll.null)
```

```
##
##
## The Psuedo R^2 is: 0.001593432
```

```r
cat("\nAnd the p-value is: ", 1-pchisq(2*(ll.proposed-ll.null), df=(length(model_logistic$coefficients)
```

```
##
## And the p-value is:  0.1001508
```

Given by the p-value (R^2 = 0.0016, p-val = 0.1) , we can say that the regression does not properly predict Churn. However, we can see that the contract length does have a significant effect in the regression (Z = -1.989, p-value = 0.467), specifically the one year contracts.

Let's try only using the contracts.

```r
model_logistic_2 <- glm(formula = Churn ~ Contract, family = "binomial", data=cust_churn_norm_features)

summary(model_logistic_2)
```

```
##
## Call:
## glm(formula = Churn ~ Contract, family = "binomial", data = cust_churn_norm_features)
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -0.96917    0.02901 -33.408   <2e-16 ***
## ContractOne year -0.15081    0.05938  -2.540   0.0111 *
## ContractTwo year  0.02619    0.05757   0.455   0.6491
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 11638  on 9971  degrees of freedom
## Residual deviance: 11630  on 9969  degrees of freedom
## AIC: 11636
##
```

```
## Number of Fisher Scoring iterations: 4
```

```r
ll.null <- model_logistic_2$null.deviance/-2
ll.proposed <- model_logistic_2$deviance/-2

cat("\n\nThe Psuedo R^2 is:", (ll.null-ll.proposed)/ll.null)
```

```
##
##
## The Psuedo R^2 is: 0.0006711029
```

```r
cat("\nAnd the p-value is: ", 1-pchisq(2*(ll.proposed-ll.null), df=(length(model_logistic_2$coefficient
```

```
##
## And the p-value is:  0.02014065
```

We can see that the model did better than the last, getting a better result (R^2 = 0.00067, p-val = 0.02).

let's now get the AUC of both:

```r
predicted_probs <- predict(model_logistic, type = "response")
roc_curve <- roc(cust_churn_norm_features$Churn, predicted_probs)
```

```
## Setting levels: control = No, case = Yes
```

```
## Setting direction: controls < cases
```

```r
auc_value <- pROC::auc(roc_curve)
cat("AUC of first model:", auc_value, "\n")
```

```
## AUC of first model: 0.5267711
```

```r
predicted_probs <- predict(model_logistic_2, type = "response")
roc_curve <- roc(cust_churn_norm_features$Churn, predicted_probs)
```

```
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
```

```r
auc_value <- pROC::auc(roc_curve)
cat("AUC of second model:", auc_value, "\n")
```

```
## AUC of second model: 0.5139753
```

Despite the first model getting a lower p-value than the second, we can see that the first model, albeit slightly, performs better than the second. Overall, both models are bad, and are no better than making our own guesses.

Now, let's try using decision trees.

```r
churn.tree <- train(
  Churn ~ .,
  data = churnTrain,
  method="rpart",
  trControl = trainControl(method = "cv", number=10, classProbs = TRUE, summaryFunction = twoClassSumma
  tuneGrid = data.frame(cp = seq(0, 0.01, by = 0.001)),
  weights = ifelse(churnTrain$Churn == "Yes",12, 1),
  metric = "ROC"

  )
```

```r
churn.tree
```

```
## CART
##
## 7978 samples
##   10 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 7181, 7181, 7179, 7180, 7179, 7181, ...
## Resampling results across tuning parameters:
##
##   cp     ROC        Sens          Spec
##   0.000  0.4962488  0.2636684880  0.7337554
##   0.001  0.4861030  0.0353863474  0.9624268
##   0.002  0.5018550  0.0046379964  0.9990719
##   0.003  0.4997937  0.0005154639  0.9990719
##   0.004  0.5000252  0.0005154639  0.9995349
##   0.005  0.5000000  0.0000000000  1.0000000
##   0.006  0.5000000  0.0000000000  1.0000000
##   0.007  0.5000000  0.0000000000  1.0000000
##   0.008  0.5000000  0.0000000000  1.0000000
##   0.009  0.5000000  0.0000000000  1.0000000
##   0.010  0.5000000  0.0000000000  1.0000000
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.002.
```

```r
predicted_probs <- predict(churn.tree, newdata = churnTest, type = "prob")$Yes

# Calculate the ROC curve
roc_curve <- roc(churnTest$Churn, predicted_probs)
```

```
## Setting levels: control = No, case = Yes
```

```
## Setting direction: controls < cases
```

```r
# Compute the AUC
auc_value <- pROC::auc(roc_curve)
print(paste("ROC AUC:", auc_value))
```

```
## [1] "ROC AUC: 0.502061855670103"
```

Setting the weight of Churn = “Yes” to 12, yields the best results (ROC AUC = 0.505).

Bias-Variance Trade Off

The Bias-Variance trade-off is more apparent with our decision tree, in which we are adusting the cp, which is related to how many splits our decision tree makes. The lower the cp, the more splits it makes, and the more complex the model is.

By increasing the complexity, or decreasing bias, we tend to capture the trends better, but a higher complexity leads to over fitting, making the model unable to properly function for new data points. But if we did not highten the complexity, in our case, we would underfit, i.e. we won’t be able to make accurate predictions because the model does not see the trends.

In relation to our logistic regression, our complexity is related to the number of predictors used. When we decreased the complexity, we increased our bias, leading to our model to underfit worse than the model with the complete predictors.

Cross-Validation

Let's use caret to create our cross validation model, and accuracy, precision, recall, and F1-score:

```
train_control <- trainControl(
  method = "repeatedcv",
  repeats = 3,
  number = 10,
  classProbs = TRUE,
  summaryFunction = function(data, lev = NULL, model = NULL) {
    default <- defaultSummary(data, lev, model)
    pr <- prSummary(data, lev, model)
    c(default, pr)
  },
  savePredictions = TRUE
)

model_cv <- train(
  Churn ~ .,
  data = churnTrain,
  method = "rpart",
  trControl = train_control,
  #weights = ifelse(churnTrain$Churn == "Yes",12, 1),
  metric = "Accuracy"
)

# Print the model summary
print(model_cv)
```

```
## CART
##
## 7978 samples
##   10 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 7180, 7180, 7179, 7181, 7181, 7181, ...
## Resampling results across tuning parameters:
##
##   cp            Accuracy   Kappa         AUC        Precision  Recall
##   0.0006029685  0.7121250  -0.005970814  0.6256356  0.7288211  0.9642768
##   0.0006957328  0.7229446  -0.003192634  0.3364527  0.7292829  0.9865438
##   0.0007168157  0.7236137  -0.002601137  0.3131209  0.7293700  0.9877466
##   F
##   0.8299766
##   0.8385476
##   0.8390416
##
## Accuracy was used to select the optimal model using the largest value.
```

```
## The final value used for the model was cp = 0.0007168157.
```

```r
# View the cross-validation results
cv_results <- model_cv$results
print(cv_results %>% dplyr::select(cp, Accuracy, Precision, Recall, F))
```

```
##             cp  Accuracy Precision    Recall         F
## 1 0.0006029685 0.7121250 0.7288211 0.9642768 0.8299766
## 2 0.0006957328 0.7229446 0.7292829 0.9865438 0.8385476
## 3 0.0007168157 0.7236137 0.7293700 0.9877466 0.8390416
```

the cp value of 0.0006029685 had the lowest accuracy (0.7121250), Recall (0.9642768), and F-stat (0.8299766), but it has the highest precision (0.7288211). It is also the most complex iteration of the model.

The model that the caret library deemed optimal is the cp value of 0.0007168157, giving the highest accuracy (0.7236137), recall (0.9877466), and F statistic (0.8390416), with the least complexity.

cp value of 0.0006957328 gave the same results, but is slightly more complex than the optimal.

Classification

Let's try using a Random Forest Classifier model to predict customer churn.

```r
train_control <- trainControl(
  method = "cv",
  number = 10,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  search = "random",
  savePredictions = TRUE
)

model_rf <- train(
  Churn ~ .,
  data = churnTrain,
  method = "rf",
  trControl = train_control,
  #weights = ifelse(churnTrain$Churn == "Yes",12, 1),
  metric = "ROC",
  tuneLength  = 10
)
```

Next, let's check the results of our random forest model:

```r
print(model_rf)
```

```
## Random Forest
##
## 7978 samples
##   10 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 7179, 7179, 7180, 7181, 7180, 7180, ...
## Resampling results across tuning parameters:
```

```
##
##   mtry  ROC        Sens       Spec
##    3    0.4854184  0.9958781  0.001856158
##    4    0.4908241  0.9723471  0.019952627
##    5    0.4917721  0.9507061  0.040366064
##    6    0.4914627  0.9453841  0.044993540
##    7    0.4929483  0.9414331  0.052405254
##    8    0.4904217  0.9374824  0.057506460
##   10    0.4912006  0.9366239  0.058438846
##   11    0.4903445  0.9335337  0.057504307
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 7.
```

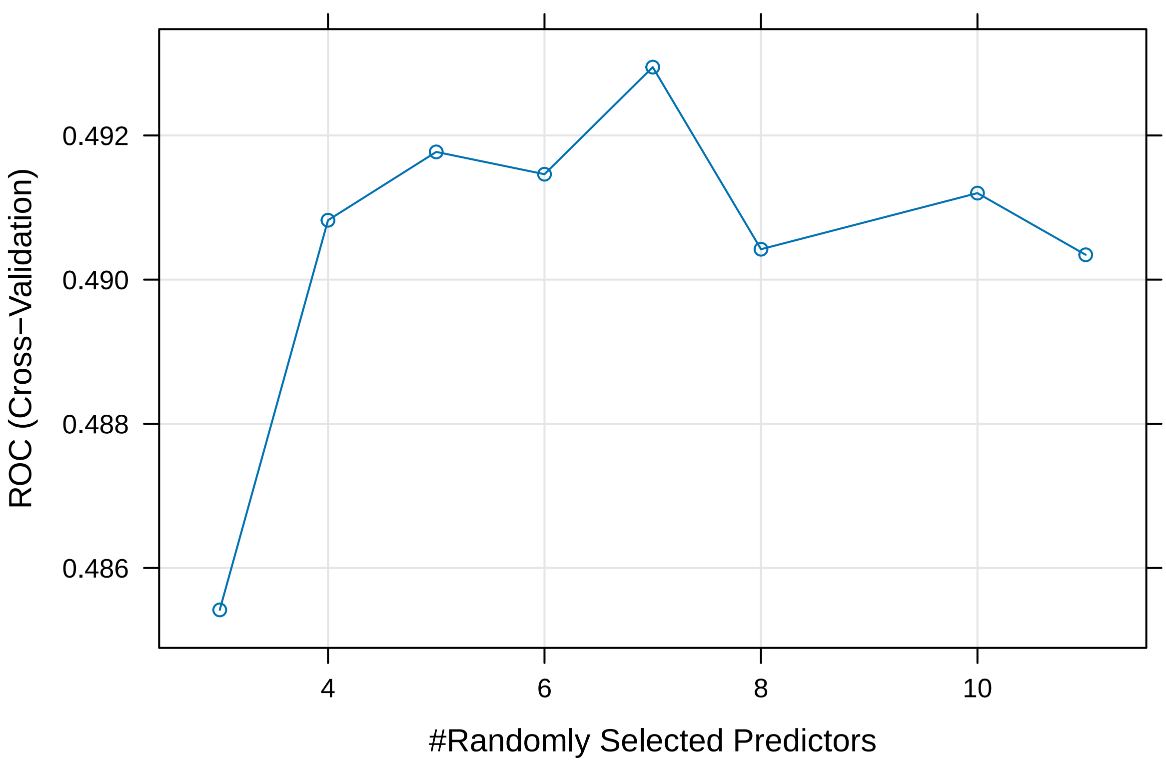
```r
print(model_rf$results)
```

```
##   mtry       ROC      Sens        Spec      ROCSD      SensSD      SpecSD
## 1    3 0.4854184 0.9958781 0.001856158 0.02544269 0.002171940 0.003241181
## 2    4 0.4908241 0.9723471 0.019952627 0.02479683 0.006187996 0.009299380
## 3    5 0.4917721 0.9507061 0.040366064 0.02590795 0.008860844 0.011639741
## 4    6 0.4914627 0.9453841 0.044993540 0.03058226 0.011949691 0.009793337
## 5    7 0.4929483 0.9414331 0.052405254 0.03121598 0.013242892 0.013267511
## 6    8 0.4904217 0.9374824 0.057506460 0.02929051 0.012394051 0.009791159
## 7   10 0.4912006 0.9366239 0.058438846 0.03146156 0.011744879 0.014366406
## 8   11 0.4903445 0.9335337 0.057504307 0.02920184 0.012160493 0.013822762
```

```r
print(model_rf$bestTune)
```

```
##   mtry
## 5    7
```

```r
plot(model_rf)
```



The best hyperparameter for the model is mtry = 8. However, the ROC is lower than 0.5, which indicates that random guessing might be better than using the model.

Overall, the classification model didn't capture the trends of the data.

Regression-Based Methods

We have previously tried classification methods. Now, let's try regression methods.

Logistic Regression

Let's fit a logistic regression model using Churn as the dependent variable and Tenure, MonthlyCharges, and TotalCharges as independent variables. Then, let's Interpret the coefficients and assess model significance using p-values.

```r
model_logistic_fin <- glm(formula = Churn ~ Tenure + MonthlyCharges + TotalCharges, family = "binomial"

summary(model_logistic_fin)
```

```
##
## Call:
## glm(formula = Churn ~ Tenure + MonthlyCharges + TotalCharges,
##     family = "binomial", data = churnTrain)
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -0.99388    0.02522 -39.408   <2e-16 ***
## Tenure          0.05692    0.06611   0.861    0.389
## MonthlyCharges  0.02075    0.05085   0.408    0.683
## TotalCharges   -0.05710    0.07886  -0.724    0.469
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9310.3  on 7977  degrees of freedom
## Residual deviance: 9309.3  on 7974  degrees of freedom
## AIC: 9317.3
##
## Number of Fisher Scoring iterations: 4
```

```r
ll.null <- model_logistic_fin$null.deviance/-2
ll.proposed <- model_logistic_fin$deviance/-2

cat("\n\nThe Psuedo R^2 is:", (ll.null-ll.proposed)/ll.null)
```

```
##
##
## The Psuedo R^2 is: 0.0001060756
```

```r
cat("\nAnd the p-value is: ", 1-pchisq(2*(ll.proposed-ll.null), df=(length(model_logistic_fin$coefficie
```

```
##
## And the p-value is:  0.8042524
```

It seems like no coefficient correlated significantly to Churn. The closest would be tenure (Z = 0.861, p-val = 0.389), followed by Total Charges (z = -0.724, p-val = 0.469), and finally, the

Monthly Charges (z = 0.408, p-val = 0.683). All p-value is insignificant.

The overall p-value of the model is 0.804, with a Psuedo R^2 of 0.0001

This all shows that the model is not able to predict customer churn, as all variables do not correlate with it.

Regression in High Dimensions

Having a high-dimensional model can be pretty problematic.

First, they can be pretty hard on the machine, needing long training times.

Second, they can cause overfitting, leading to the model not being able to see the pattern of the data. This leads to errors in predicting the result when a new data point comes.,

third, is that most of the time, predictors correlate with each other, which can ruin the training of the model.

One way to combat this is to use Principal Component Analysis (PCA) on numerical features to be use variables that correlate together as one component, instead of using all of them.

Let's try creating a PCA of Tenure, MonthlyCharges, and TotalCharges.

```
numeric_scaled_data <- cust_churn_norm_features %>% dplyr::select(where(is.numeric))

pca_result <- prcomp(numeric_scaled_data, center = TRUE, scale. = TRUE)

print(pca_result)
```

```
## Standard deviations (1, .., p=3):
## [1] 1.383578 1.015932 0.231504
##
## Rotation (n x k) = (3 x 3):
##                       PC1         PC2        PC3
## Tenure         -0.5811163  0.57012040 -0.5807466
## MonthlyCharges -0.3921231 -0.82146001 -0.4140566
## TotalCharges   -0.7131222 -0.01289089  0.7009212
```

```
cat("\n\n")
```

```
summary(pca_result)
```

```
## Importance of components:
##                           PC1    PC2     PC3
## Standard deviation     1.3836 1.0159 0.23150
## Proportion of Variance 0.6381 0.3440 0.01786
## Cumulative Proportion  0.6381 0.9821 1.00000
```

We can see that the first component is negatively correlated to both Tenure (R=-0.5811163) and Total Charges (R=-0.7131222), and significantly so. We can then say that, when tenure decreases, the total charge also decreases. This makes sense, because customers with higher tenure pay more in the long run.

The second component is significantly correlated with Tenure (R=0.57012040), while is negatively correlated with Monthly Charges (R=-0.82146001). This tells us that when the monthly charges increase, the tenure increases, or vice versa.

The last componenent is then the opposite of the first component, where it says that it is negatively correlated to Tenure (R=-0.5807466), but is positively correlated to total charges (R=0.7009212)

Ridge Regression

Implement Ridge Regression using Churn as the target variable and Tenure, MonthlyCharges, TotalCharges, and additional customer demographic features as predictors.

Identify the optimal lambda using cross-validation.

```r
predictors <- c("Tenure", "MonthlyCharges", "TotalCharges")

encoded_contract <- model.matrix(~ Contract - 1, data = churnTrain)
encoded_contract <- as.data.frame(encoded_contract)
num_preds <- churnTrain  %>% dplyr::select(all_of(predictors)) %>% as.data.frame()
preds <- cbind(num_preds, encoded_contract)

enc_cont_test <- model.matrix(~ Contract - 1, data = churnTest)
enc_cont_test <- as.data.frame(enc_cont_test)
preds_num_test <- churnTest  %>% dplyr::select(all_of(predictors)) %>% as.data.frame()
preds_test <- cbind(preds_num_test, enc_cont_test)


y <- churnTrain  %>% dplyr::select(Churn)

# Set up cross-validation
ctrl <- trainControl(
  method = "cv",
  number = 10,
)

weight = 2.70037105751
ridge_model <- train(
  x = preds,
  y = y$Churn,
  method = "glmnet",
  trControl = ctrl,
  tuneGrid = expand.grid(alpha = 0, lambda = seq(0, 50, length = 51)),
  weights = ifelse(churnTrain$Churn == "Yes",weight, 1),
)

cat("The optimal lambda is:")
```

```
## The optimal lambda is:
```

```r
print(ridge_model$bestTune$lambda)
```

```
## [1] 18
```

```r
# Make predictions and evaluate the model
predictions <- predict(ridge_model, newdata = preds_test)
predictions <- factor(predictions, levels = levels(y$Churn))
confusionMatrix(predictions, churnTest$Churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1455  539
##        Yes    0    0
```

```
##
##                Accuracy : 0.7297
##                  95% CI : (0.7096, 0.7491)
##     No Information Rate : 0.7297
##     P-Value [Acc > NIR] : 0.5116
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.7297
##          Neg Pred Value :    NaN
##              Prevalence : 0.7297
##          Detection Rate : 0.7297
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

The in this particular ridge regression model, the most optimal lambda value is 50. I wager that if we increase the lambda value, it will take the highest possible value.

Looking at the confusion matrix, the model only gives out a prediction of "No", Changing the weight of "Yes" in training seemes to be able to improve things, but after multiple iterations, the model only gives out "Yes" (true positives and false positives), or "No" (True Negatives or False Negatives) with each change. A better interpolation approach would be required into making this into a working model.

However, due to the prevalence of "No", it is correct 0.7297 accounting to "No" being some 73% of the dataset. The p-value given by the model is 0.5116.

Lasso Regression

Now, let's try the Lasso Regression

```r
lasso_model <- train(
  x = preds,
  y = y$Churn,
  method = "glmnet",
  trControl = ctrl,
  tuneGrid = expand.grid(alpha = 0, lambda = seq(0, 50, length = 51)),
  weights = ifelse(churnTrain$Churn == "Yes",weight, 1),
)

cat("The optimal lambda is:")
```

```
## The optimal lambda is:
```

```r
print(lasso_model$bestTune$lambda)
```

```
## [1] 50
```

```r
# Make predictions and evaluate the model
predictions_lasso <- predict(lasso_model, newdata = preds_test)
```

```r
predictions_lasso <- factor(predictions_lasso, levels = levels(y$Churn))
confusionMatrix(predictions_lasso, churnTest$Churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1455  539
##        Yes    0    0
##
##                Accuracy : 0.7297
##                  95% CI : (0.7096, 0.7491)
##     No Information Rate : 0.7297
##     P-Value [Acc > NIR] : 0.5116
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.7297
##          Neg Pred Value :    NaN
##              Prevalence : 0.7297
##          Detection Rate : 0.7297
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

The lasso regression provides the same exact results as the ridge regression.