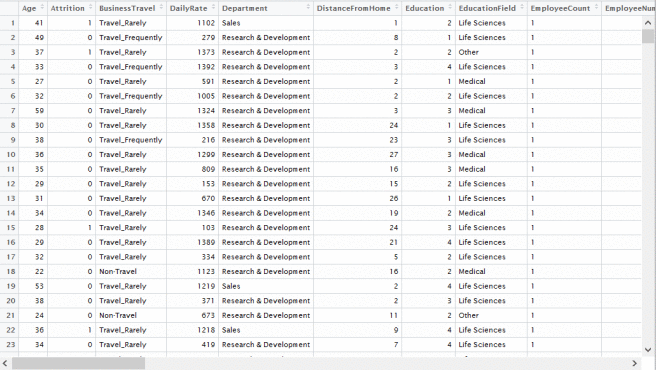
Employee attrition is costly. It can significantly affect a company’s growth and bottom line. Because of this, it has become increasingly popular to use data analysis methods and technology to understand and manage employee attrition.

This report shows how exploratory data analysis and predictive modeling can be done on a employee attrition/turnover data set using R and RStudio.

**Employee Turnover/Attrition Sample Data**

Before any data analysis can be done we first need to load the employee turnover sample data into RStudio

The sample data has 1,470 rows and 35 columns (i.e. 1,470 instances and 35 variables). Variables include each employee’s age, distance from home, amount of business travel, education level, whether or not the employee left the company, and several other employee characteristics.



**Examine the Data Structure**

Data loading and examining. In R, we use str to do this:

# Code: Process data and data types  
str(alldata)

We need to convert some fields into factors:

# Code: Convert some ints to factors  
alldata$Education <- as.factor(alldata$Education)  
alldata$EnvironmentSatisfaction <- as.factor(alldata$EnvironmentSatisfaction)  
alldata$JobInvolvement <- as.factor(alldata$JobInvolvement)  
alldata$JobSatisfaction <- as.factor(alldata$JobSatisfaction)  
alldata$PerformanceRating <- as.factor(alldata$PerformanceRating)  
alldata$RelationshipSatisfaction <- as.factor(alldata$RelationshipSatisfaction)  
alldata$WorkLifeBalance <- as.factor(alldata$WorkLifeBalance)  
alldata$JobLevel <- as.factor(alldata$JobLevel)  
alldata$StockOptionLevel <- as.factor(alldata$StockOptionLevel)

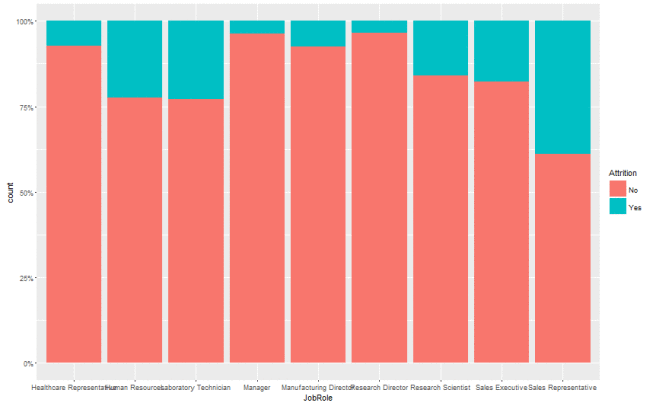
**Exploratory Data Analysis**

After processing and transforming our data, we can do some exploratory data analysis (EDA). The goal of EDA is to get to know our data, get some descriptive statistics, and investigate relationships that may exist. A lot of interesting insights can be gleaned from EDA. Moreover, some of these insights can be very useful in the future when we get into the modeling phase of the project.

For example, we might want to check if there are different attrition rates across different types of jobs. We use the ggplot2 package and this code to create two plots that show attrition rates across job roles.

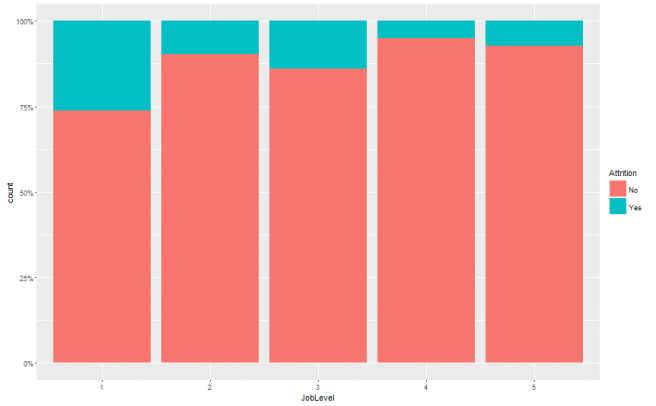
# Code: Hypothesis: Some JobRole have high attrition rates  
ggplot(train, aes(x = JobRole, fill = Attrition)) +  
stat\_count(width = 0.5) +  
xlab(“Job Role”) +  
ylab(“Count”) +  
labs(fill = “Attrition”)

ggplot(train, aes(x = JobRole)) + geom\_bar(aes(fill = Attrition), position = ‘fill’) +  
scale\_y\_continuous(labels = percent\_format())



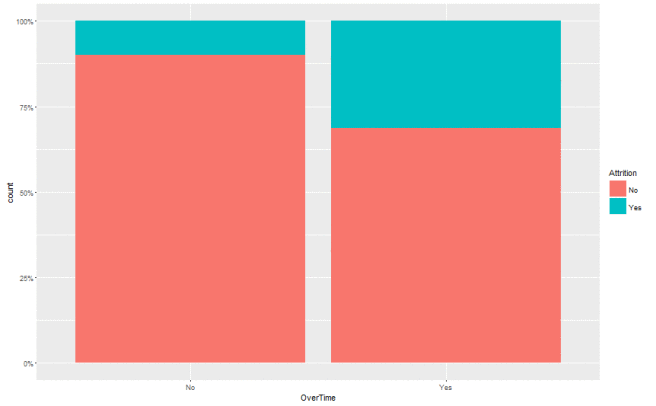
As we can see above in one of the plots we created, the Sales Representative job role has significantly more attrition relative to other job roles.

Here’s another plot that shows attrition across job levels.



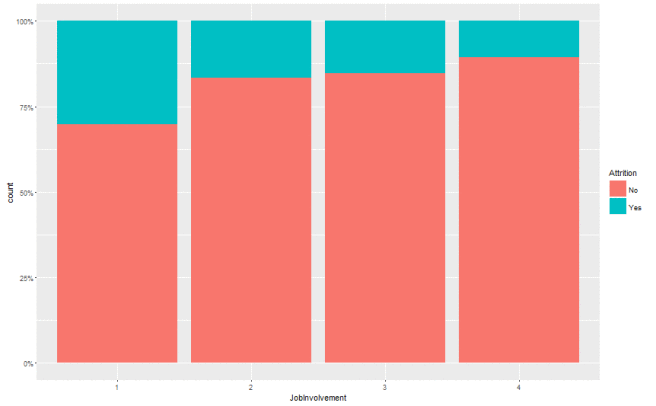
As we can see, attrition is a lot higher when job level = 1.

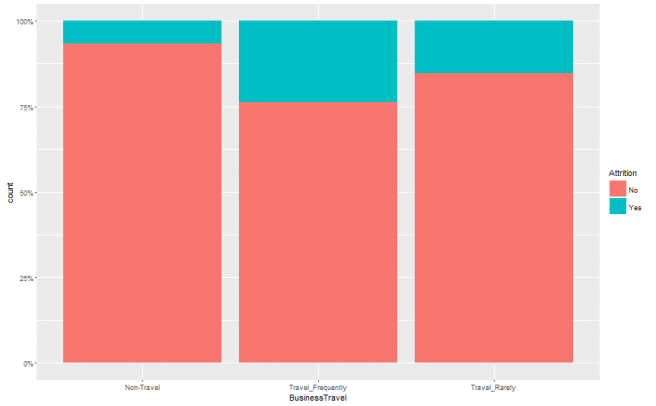
The next plot could also be very interesting. It shows how overtime (as a proxy for hourly vs. exempt employees) affects attrition.



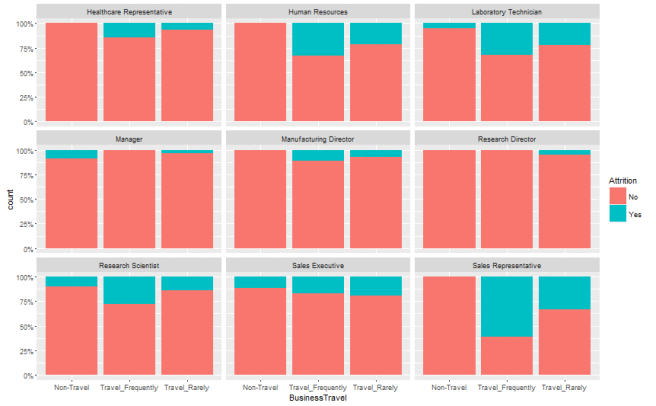
As can be seen above, hourly employees have significantly higher attrition rates as compared to exempt employees with no overtime.

Here are other interesting plots that show some interesting relationships and give clues on what variables affect attrition rates such as job involvement and business travel.





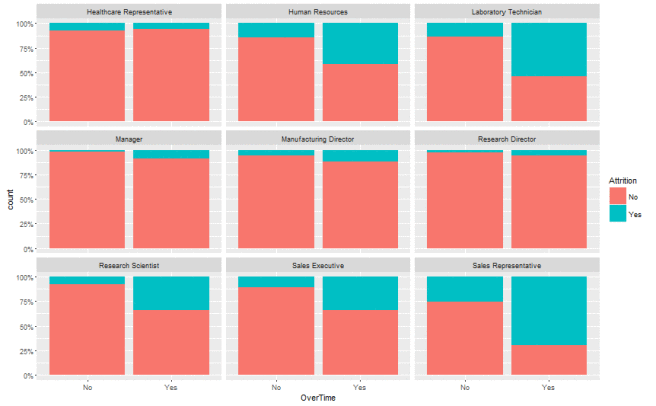
Another cool thing we are doing on R is check the relationship across three variables based on a hypothesis. For example, in the plot below, we check how job role and business travel affects attrition.



We find something very interesting in the plot above. We already know that sales representatives have high attrition rates. The plot above gives us even more information: We see that sales representatives that travel frequently have very high attrition rates (greater that 50%!!!) and that non-traveling sales representatives have very low attrition.

I ran the same analysis on some other factors and found some interesting insights below. I will try plotting them and check them out and figure out if any of these insights are actionable.

There is high attrition among hourly sales representatives and lab technicians.



Work-life balance seems to affect the attrition rates of lab technicians more than sales representatives.



**Scaling and Looking for High Correlations**

Before we do the Principal Component Analysis, it is usually a good idea to check the standard deviations of our variables to determine if we need to scale the data. In the code below, we check sd on our variables of the training data set.

# Code: standard deviation to determine if we need to scale  
sapply(train, sd)

This shows us the following:

Age Attrition

9.1072540 0.3666357

BusinessTravel DailyRate

0.6672492 401.2319634

Department DistanceFromHome

0.5258680 8.0478547

Education EducationField

1.0153433 1.3227162

EmployeeCount EmployeeNumber

0.0000000 601.8080750

EnvironmentSatisfaction Gender

1.0878315 0.4908889

HourlyRate JobInvolvement

20.3399360 0.7072229

JobLevel JobRole

1.0974566 2.4811670

JobSatisfaction MaritalStatus

1.1027981 0.7311559

MonthlyIncome MonthlyRate

4665.7415181 7057.4297356

NumCompaniesWorked Over18

2.4931237 0.0000000

OverTime PercentSalaryHike

0.4511352 3.6337468

PerformanceRating RelationshipSatisfaction

0.3552106 1.0847260

StandardHours StockOptionLevel

0.0000000 0.8542454

TotalWorkingYears TrainingTimesLastYear

7.6388123 1.2960934

WorkLifeBalance YearsAtCompany

0.6987070 5.9917422

YearsInCurrentRole YearsSinceLastPromotion

3.6327980 3.2020771

YearsWithCurrManager

3.5427293

I noticed significant differences in our sd values. It’s a good idea to scale our data. Below is some useful code to scale a data set with mixed variable data types. It finds the numeric variables, scales their values, and updates the data set with these new scaled values into a new data set train.scaled.

# Code: scale the data of numeric columns  
design.matrix <- train  
numeric.columns <- design.matrix[,unlist(lapply(design.matrix,is.numeric))]  
scaled.numeric.columns <- scale(numeric.columns)  
design.matrix[,unlist(lapply(design.matrix,is.numeric))] <- scaled.numeric.columns  
train.scaled <- design.matrix

I would also want to check which variables are the most highly correlated. This might help us remove some variables that are very highly correlated with other variables. We can use this code below to do this. It gets the numeric variables and checks correlation across all combinations.

# Code: get the most highly correlated variables  
train.numericonly <- design.matrix[,unlist(lapply(design.matrix,is.numeric))]  
mosthighlycorrelated <- function(mydataframe,numtoreport)  
{  
# find the correlations  
cormatrix <- cor(mydataframe)  
# set the correlations on the diagonal or lower triangle to zero,  
# so they will not be reported as the highest ones:  
diag(cormatrix) <- 0  
cormatrix[lower.tri(cormatrix)] <- 0  
# flatten the matrix into a dataframe for easy sorting  
fm <- as.data.frame(as.table(cormatrix))  
# assign human-friendly names  
names(fm) <- c(“First.Variable”, “Second.Variable”,”Correlation”)  
# sort and print the top n correlations  
head(fm[order(abs(fm$Correlation),decreasing=T),],n=numtoreport)  
}  
mosthighlycorrelated(train.numericonly, 5)

Running the code above, we find the 5 highest correlated variable pairs:

First.Variable Second.Variable Correlation

194 MonthlyIncome TotalWorkingYears 0.7720484

252 YearsAtCompany YearsInCurrentRole 0.7673168

286 YearsAtCompany YearsWithCurrManager 0.7639706

287 YearsInCurrentRole YearsWithCurrManager 0.7047477

188 Age TotalWorkingYears 0.6713113

This also gives us some insight on the data that will be useful later in the project.

**Principal Components Analysis (PCA)**

The purpose of PCA is variable reduction. Now that we’ve scaled the data, we can perform PCA.

# Code: PCA  
train.pca <- prcomp(train.numericonly.scaled)

We can then check the results to determine how many components we need. We discuss three of the most popular methods here:

1- Eigenvalue one criterion: based on this criterion we choose the first components with eigenvalues higher than 1.  
2- Amount of explained variance: based on this, the chosen factors should explain 70 to 80% of our variance at least.  
3- Scree plot: this is a graphical method in which we choose the factors until a break in the graph

For example, in our data, there are five components with eigenvalues greater than 1.

[1] 3.9848001 1.6959384 1.1114873 1.0670108 1.0451258

[6] 0.9994388 0.9680187 0.9091655 0.8968254 0.7225999

[11] 0.5199347 0.4625047 0.2860060 0.1863880 0.1447559

Or, also in our data, we need seven components to explain at least 70% of the variance:

Importance of components:

PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12 PC13

Standard deviation 1.9962 1.3023 1.0543 1.03296 1.02231 0.99972 0.98388 0.95350 0.94701 0.85006 0.72106 0.68008 0.53480

Proportion of Variance 0.2656 0.1131 0.0741 0.07113 0.06968 0.06663 0.06453 0.06061 0.05979 0.04817 0.03466 0.03083 0.01907

Cumulative Proportion 0.2656 0.3787 0.4528 0.52395 0.59362 0.66025 0.72479 0.78540 0.84519 0.89336 0.92802 0.95886 0.97792

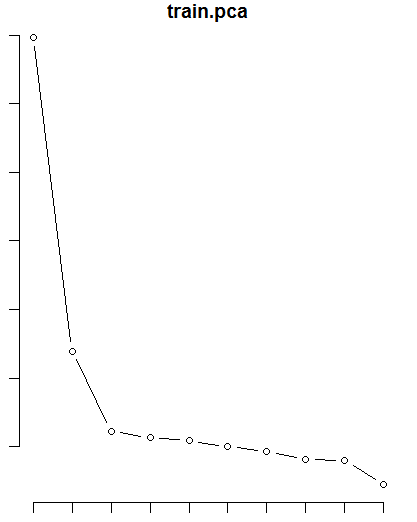
PC14 PC15

Standard deviation 0.43173 0.38047

Proportion of Variance 0.01243 0.00965

Cumulative Proportion 0.99035 1.00000

Or, in the screeplot below, there is an “elbow” after the third component.



This means, we can choose to retain 5, 7, or 2 components.

We can then check the PC loadings to see which variables comprise the components

# Code: PC loadings – Used to determine which variables belong to a component  
train.pca$rotation[,1]  
train.pca$rotation[,2]  
train.pca$rotation[,3]  
train.pca$rotation[,4]  
train.pca$rotation[,5]

Here we see component 1 seems to be influenced TotalWorkingYears and YearsAtTheCompany. Component 2 is influenced mainly by Age. C3 by DistanceFromHome. C4 by DailyRate and MonthlyIncome. C5 by hourly rate and salary hike.

**Factor Analysis**

Now that we know how many components we need, I will try some factor analysis. Using factanal, we get the following output:

Uniquenesses:

Age DailyRate

0.376 0.360

DistanceFromHome EmployeeNumber

0.997 0.990

HourlyRate MonthlyIncome

0.985 0.221

MonthlyRate NumCompaniesWorked

0.975 0.748

PercentSalaryHike TotalWorkingYears

0.994 0.101

TrainingTimesLastYear YearsAtCompany

0.992 0.005

YearsInCurrentRole YearsSinceLastPromotion

0.303 0.415

YearsWithCurrManager

0.225

Loadings:

Factor1 Factor2 Factor3 Factor4

Age 0.208 0.711 0.249

DailyRate 0.789

DistanceFromHome

EmployeeNumber

HourlyRate

MonthlyIncome 0.370 0.360 0.625

MonthlyRate

NumCompaniesWorked -0.124 0.480

PercentSalaryHike

TotalWorkingYears 0.500 0.601 0.505

TrainingTimesLastYear

YearsAtCompany 0.893 0.279

YearsInCurrentRole 0.830

YearsSinceLastPromotion 0.675

YearsWithCurrManager 0.853 0.122

Factor5 Factor6

Age 0.110

DailyRate -0.105

DistanceFromHome

EmployeeNumber

HourlyRate -0.107

MonthlyIncome 0.146 0.306

MonthlyRate 0.145

NumCompaniesWorked

PercentSalaryHike

TotalWorkingYears 0.140

TrainingTimesLastYear

YearsAtCompany 0.323 -0.121

YearsInCurrentRole

YearsSinceLastPromotion 0.242 0.236

YearsWithCurrManager -0.140 -0.105

Factor1 Factor2 Factor3 Factor4 Factor5

SS loadings 3.119 1.246 0.812 0.652 0.255

Proportion Var 0.208 0.083 0.054 0.043 0.017

Cumulative Var 0.208 0.291 0.345 0.389 0.405

Factor6

SS loadings 0.231

Proportion Var 0.015

Cumulative Var 0.421

Test of the hypothesis that 6 factors are sufficient.

The chi square statistic is 16.58 on 30 degrees of freedom.

The p-value is 0.977

This shows us high loadings on some factors on associated variables. This might help us make sense of the clusters and groupings we might find in our analysis down the road.

We’ll be running our data through some classification systems, evaluate prediction accuracy, and check variable importance.

**Random Forest Model**

The first algorithm I tried was Random Forest. We’ll use the code below to create a model that predicts whether or not an employee will attrite based on our known variables.

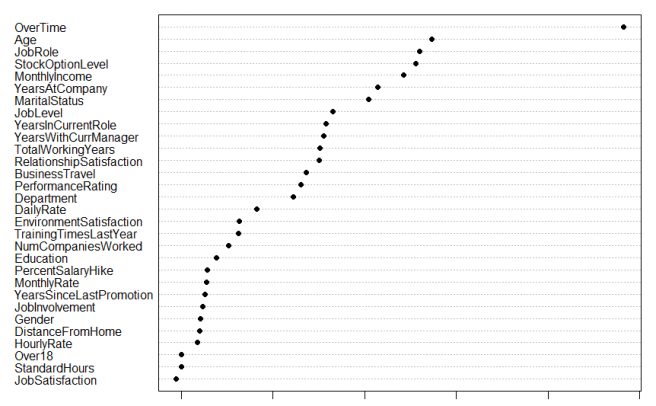
# Code: Random Forest Model  
randomForestModel <- randomForest(Attrition~Age + BusinessTravel + DailyRate + Department + DistanceFromHome + Education + EducationField + EnvironmentSatisfaction + Gender + HourlyRate + JobInvolvement + JobLevel + JobRole + JobSatisfaction + MaritalStatus + MonthlyIncome + MonthlyRate + NumCompaniesWorked + Over18 + OverTime + PercentSalaryHike + PerformanceRating + RelationshipSatisfaction + StandardHours + StockOptionLevel + TotalWorkingYears + TrainingTimesLastYear + WorkLifeBalance + YearsAtCompany + YearsInCurrentRole + YearsSinceLastPromotion + YearsWithCurrManager,data=train,ntree=100,mtry=5, importance=TRUE)  
print(randomForestModel)

Running the code above, we get the following output:

Type of random forest: classification Number of trees: 100 No. of variables tried at each split: 5 OOB estimate of error rate: 14.03% Confusion matrix: No Yes class.error No 976 12 0.01214575 Yes 153 35 0.81382979

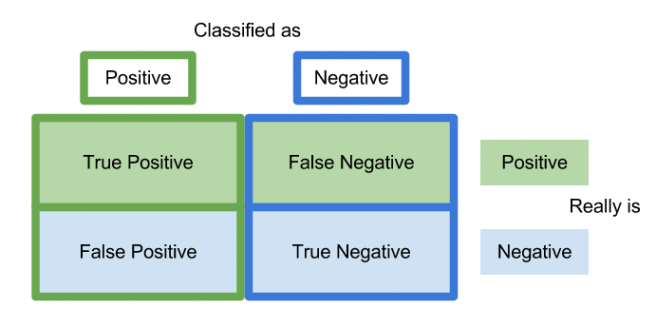
What this means is that we get a 14% error rate which is not too bad. BUT we get a 81% class error rate where our model predicts a whole lot of false positives. This is unacceptable.

However, we can still get a lot of information from this model – such as how important the variables are to the classification system. We can plot the importance of each variable to show how each variable ranks in importance.



This shows us that Overtime, by far, is the most important variable associated with attrition. Other important variables are age, job role, and stock option level. This is interesting because it coincides well with some of the exploratory analysis results we got in initially.

Meanwhile, here’s a handy cheat sheet on how to evaluate confusion matrices.



**Decision Tree Model**

Another algorithm we can try is the decision tree algorithm (which is very much related to random forest). We’ll use the code below to create a decision tree model to predict attrition.

# Code: Decision Tree Model  
decTree <- rpart(Attrition~Age + BusinessTravel + DailyRate + Department + DistanceFromHome + Education + EducationField + EnvironmentSatisfaction + Gender + HourlyRate + JobInvolvement + JobLevel + JobRole + JobSatisfaction + MaritalStatus + MonthlyIncome + MonthlyRate + NumCompaniesWorked + Over18 + OverTime + PercentSalaryHike + PerformanceRating + RelationshipSatisfaction + StandardHours + StockOptionLevel + TotalWorkingYears + TrainingTimesLastYear + WorkLifeBalance + YearsAtCompany + YearsInCurrentRole + YearsSinceLastPromotion + YearsWithCurrManager,data=train)

We can then evaluate this decision tree model.

# Code: Evaluate Models  
randomForestResult <- predict(randomForestModel, test, type=c(“class”))  
decTreeResult <- predict(decTree, test, type=c(“class”))  
## test$PAtt <– predict(randomForestModel, test, type=”response”)  
## prediction <- function(t) ifelse(randomForestResult > t, 1,0)  
randomForestModel  
decTreeResult  
table(decTreeResult, test$Attrition)

This gives us the following confusion matrix:

decTreeResult No Yes No 230 39 Yes 15 10

Similar to the random forest model, there are way too many false positives. With this model, we get okay accuracy and okay specificity – but horrible precision.

**C50 Model**

I wanted to try another model. This one is a R package called C5.0. This model is set up quite differently from the other models. We need to do a bit more data prep to get this to work. Here is the relevant code snippets below.

# Code: Create Independent and Dependent data  
alldataC50 <- alldataC50[ sample( nrow( alldataC50 ) ), ]  
C50x <- alldataC50[,-c(2)] # independent  
C50y <- alldataC50[,c(2)] # dependent

# Code: Create train and test sets for x and y  
C50xtrain <- C50x[1:1176, ]  
C50xtest <- C50x[1177:1470, ]  
C50ytrain <- C50y[1:1176]  
C50ytest <- C50y[1177:1470]

Now that we have prepped the data, we can create the C50 model.

# Code: Install and load C50  
library(C50)  
c50model <- C50::C5.0( C50xtrain, C50ytrain )  
summary( c50model )

Running the code above gives us 92.3% accuracy which is better than the previous models. However there are still too many false positives. Let’s try boosting and see how this helps our predictive characteristics.

# Code: Try Boosting with 10 trials  
c50model <- C50::C5.0( C50xtrain, C50ytrain, trials=10 )  
summary( c50model )

Boosting helped. Based on the new confusion matrix, we have significantly better specificity and a lot less false positives. We can further evaluate the model using our test data set.

# Code: Evaluate the model using the Test Set (y)  
p <- predict( c50model, C50xtest, type=”class” )  
sum( p == C50ytest ) / length( p )  
## 85.7% accurate

The code above gives us an accuracy of 85.7%.

We can also use the code below to give us the prediction for each instance in our test data set.

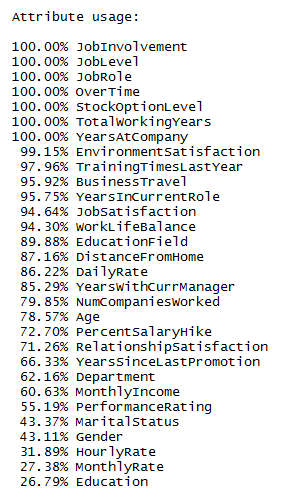
pprob <- predict( c50model, C50xtest, type=”prob” )  
pprob

The code above gives us output such as:

No Yes 355 0.5240595 0.47594046 503 0.8091050 0.19089501 1448 0.8159891 0.18401093 119 0.9233563 0.07664371 615 0.5177806 0.48221943 497 0.8322792 0.16772078 214 0.9196789 0.08032109  
…more data not shown here

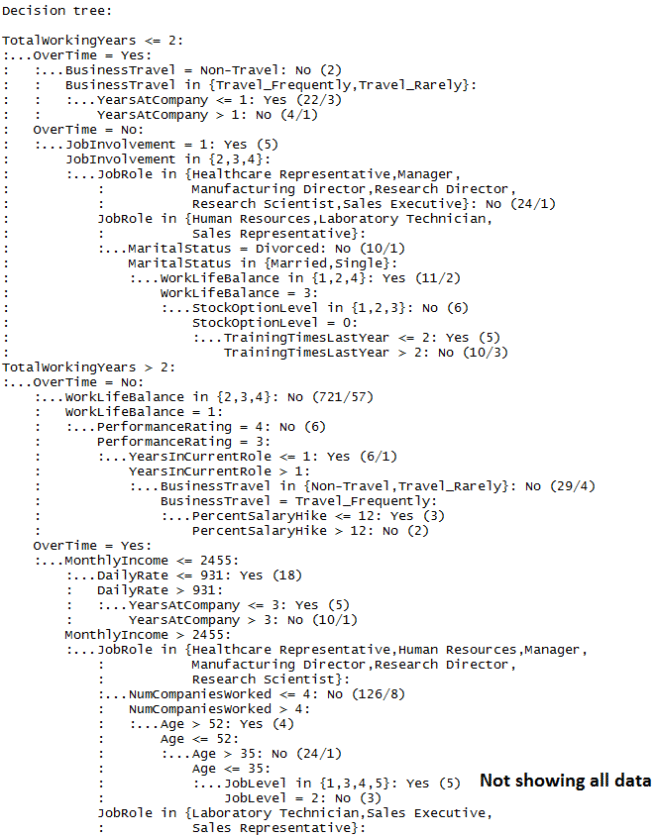
The output shows us the likelihood each instance might attrite which can be very useful to our target audience.

This model also gives us attribute usage and importance in the classification system.



As we can see above, the important attributes align pretty well with the variable importance plot we got from the random forest model and reinforces the importance of overtime and job role to attrition. This is also consistent with our exploratory data analysis graphs.

The C5.0 package also outputs the decision tree that was used for each trial which is a good way to see how the predictions were made.

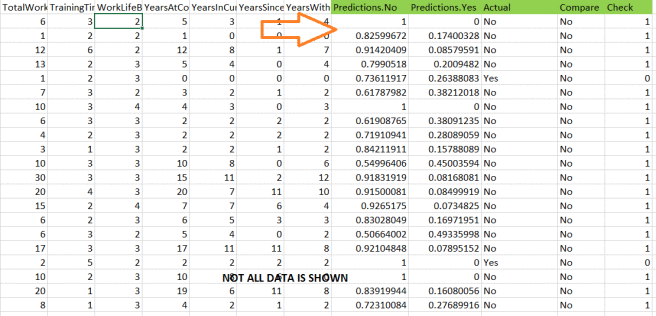


**Predictions and Likelihood of Attrition**

We can also check the prediction and predicted likelihood of attrition for each row in our test data set. The predictive model gives us the probability that an instance/employee will attrite.

# Code: Probability of Attrition  
pprob <- predict( c50model, C50xtest, type=”prob” )  
pprob

We can then write out these predictions and probabilities and look at it in a spreadsheet.



As seen in the screenshot above, there are probability values for both the “Yes” and “No” predictions. I added the Actual columns to evaluate the accuracy of the model.

This would be very helpful for clarity of understanding! We can give Human Resources users both a prediction and likelihood of attrition for each employee in a report.

**Conclusion**

We explored a few predictive models. We created and evaluated random forest and decision tree models. Unfortunately, these models predicted too many false positives. We then tried the C50 model. This model did better and gave us 85% predictive accuracy on our test data set. It also predicted very few false positives. The model summary information also listed the attribute usage when predicting attrition.

**Survival Package and Loading Employee Data**

To perform this analysis, we’ll install and load the R survival package. The R survival package can be found in cran.

# Code: Load Survival package  
library(survival)

# Code: Load and attach data  
alldata <- read.csv(“/home/user/Desktop/Data Analysis/”, header = TRUE)

attach(alldata)

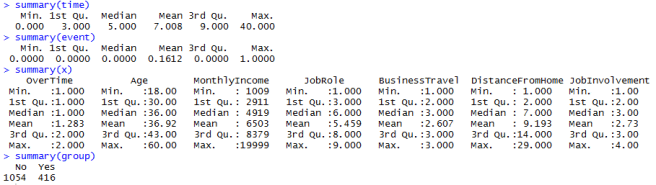
Once the data and the survival package has been loaded, we can define our variables.

# Code: Define variables  
time <- YearsAtCompany  
event <- Attrition  
x <- cbind(OverTime,Age,MonthlyIncome,JobRole,BusinessTravel,DistanceFromHome,JobInvolvement)  
group <- OverTime

We can also check some summary statistics on our variables.

# Code: Summary Statistics  
summary(time)  
summary(event)  
summary(x)  
summary(group)

This gives us the following console output.



It gives us mean, median, min, max, and quartile information on our variables. After reviewing this output, we can set up our survival analysis by creating a Surv object.

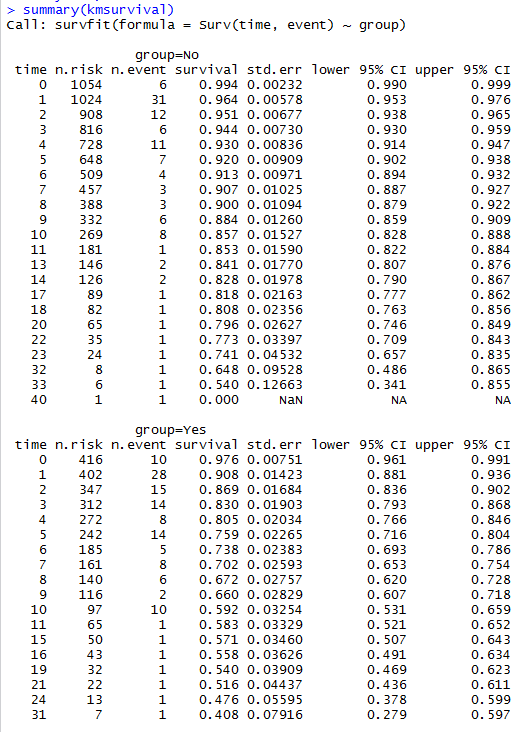
# Code: Create Surv object  
survival <- Surv(time,event)

We then use this Surv object in our survival analysis. The first method we’ll try is the Kaplan-Meier method.

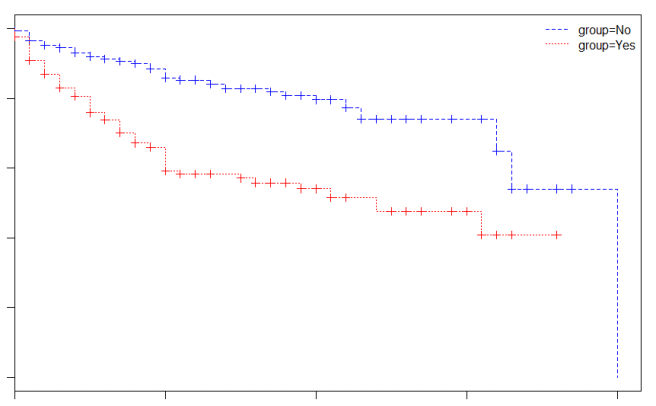
**Kaplan-Meier Non-parametric Analysis**

# Code: # Kaplan-Meier non-parametric analysis by group  
kmsurvival <- survfit(Surv(time,event) ~group)

The code above gives us a Kaplan-Meier method object we can use to analyze the data. The output is the following:



As we can see above, the KM object conveniently created the survival matrices for each group for us. This data is better visualized on a plot.



In the plot above, the x-axis is Time and the y-axis is Survival Probability. The plot shows a comparison of the survival probability of employees with overtime (“Yes” red line) and employees without overtime (“No” blue line) over Time. The plot shows that survival probability of the employees with overtime (“Yes” red line) drops faster as compared to the “No” group. If we use Overtime as a proxy for hourly vs. exempt, could this mean that exempt/salaried employees stay at the company longer than hourly employees?

**Cox Proportional Hazards Regression Model**

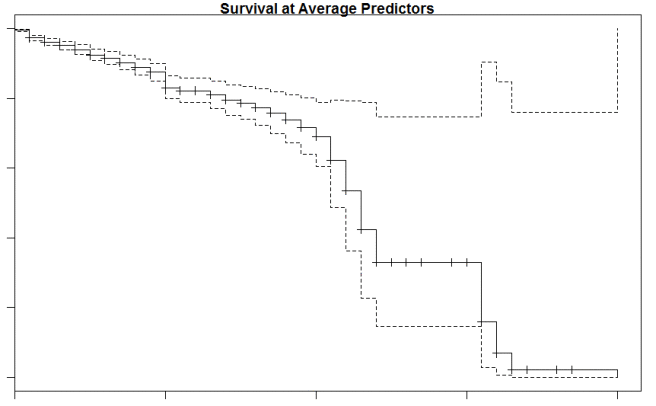
Another method we can use is coxph. In the survival package, the coxph:

Fits a Cox proportional hazards regression model. Time dependent variables, time dependent strata, multiple events per subject, and other extensions are incorporated using the counting process formulation of Andersen and Gill.

I tried this method by running the following code.

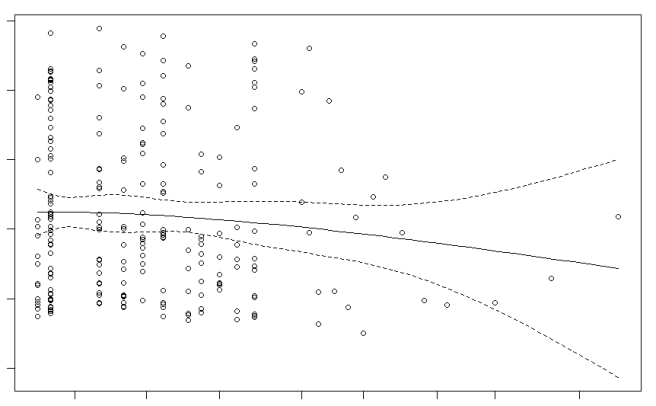
coxph <- coxph(survival ~ OverTime +  
Age +  
MonthlyIncome +  
DistanceFromHome, alldata)

We can then plot the result.



We can then compute a test for each covariate (in this case the covariates are overtime, age, monthly income, and distance from home). We can also plot the results of our computation.

cox.zph(coxph)  
plot(cox.zph(coxph))



This is just the tip of the iceberg in survival analysis.