Course Notes

# Functions TOC

A list of the functions created in this document.

ToDo:

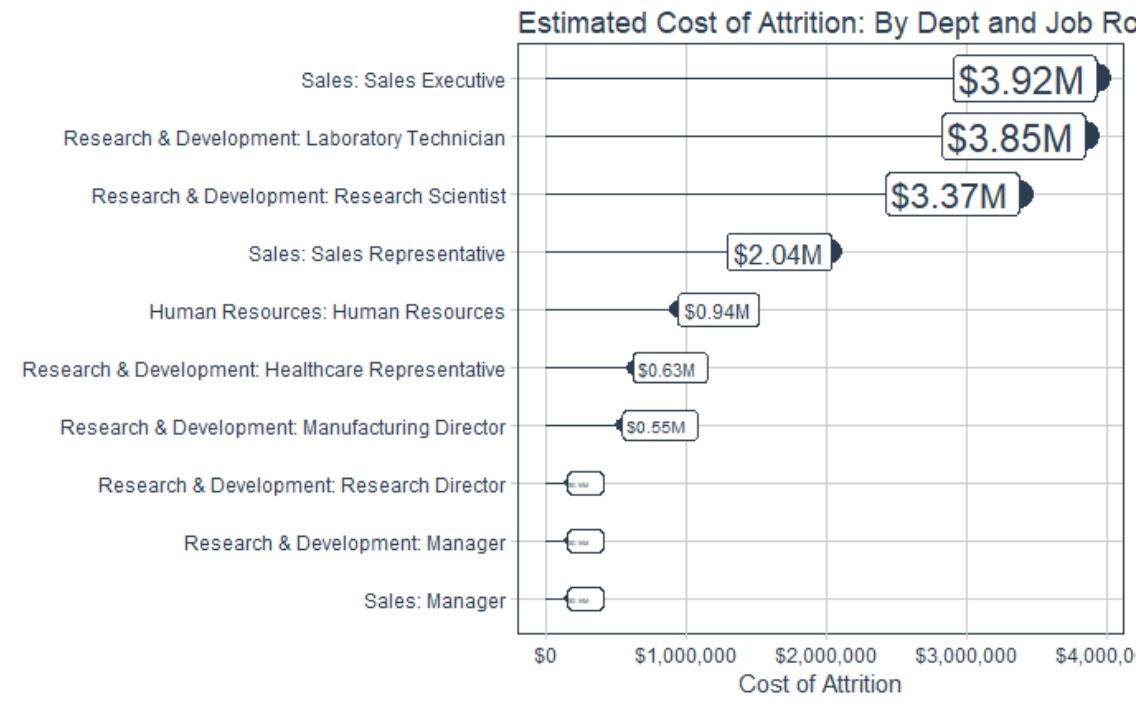
Add: calculate\_attrition\_cost, count\_to\_pct, plot\_attrition in scrips - assess\_attrition

Artifact

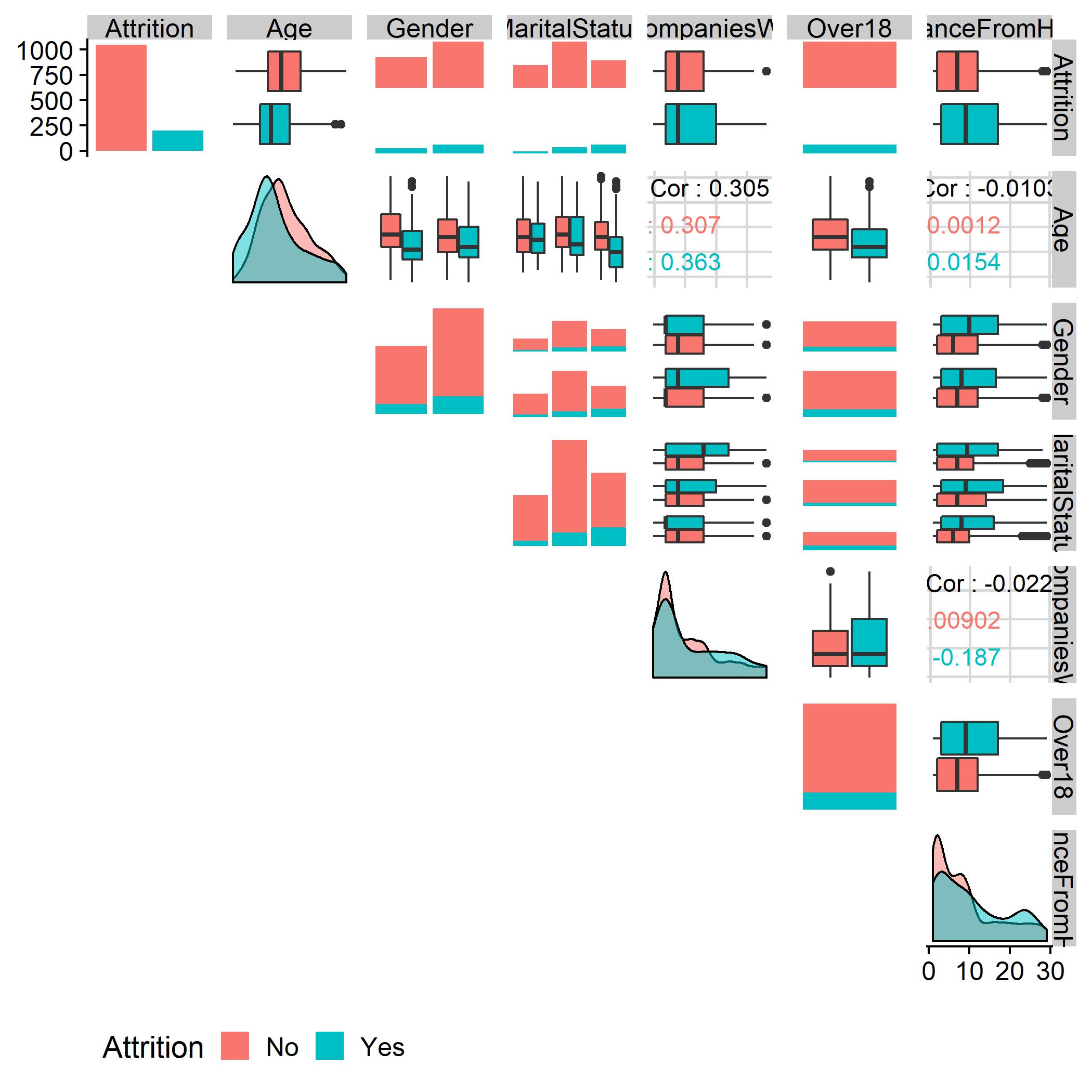
Thumbnail

Visualizations

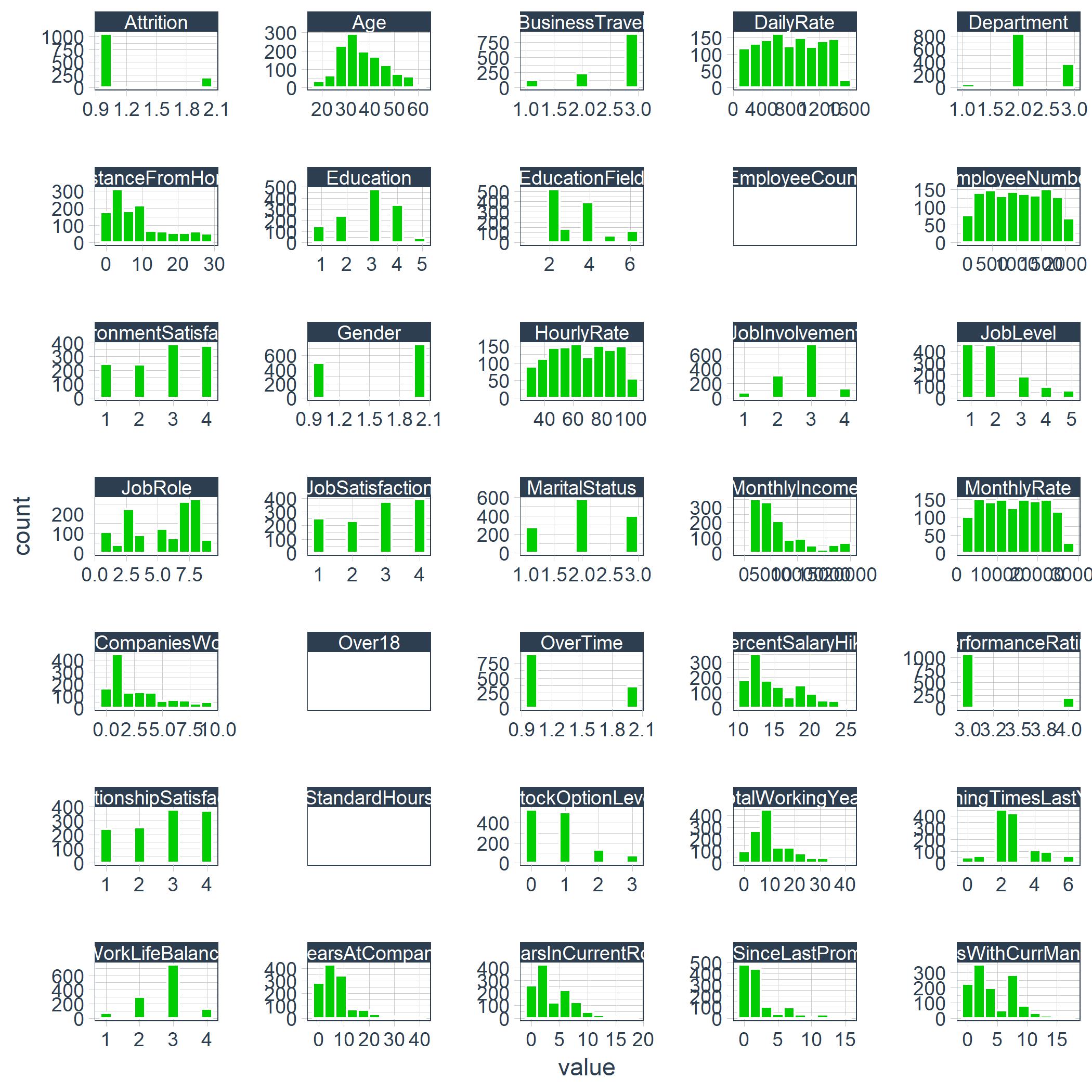
[plot\_attrition](#plot_attrition)



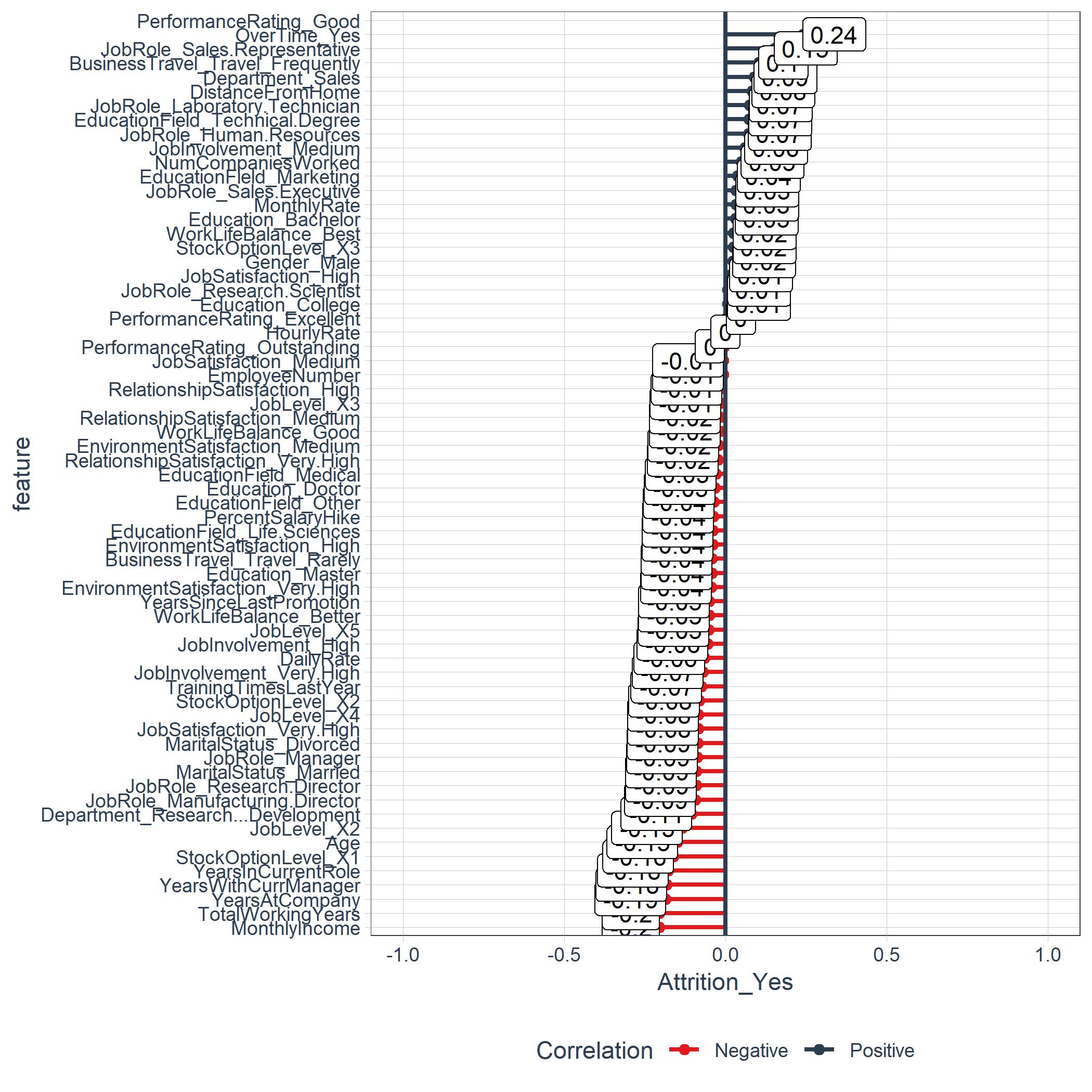
[plot\_ggpairs](#plot_ggpairs)



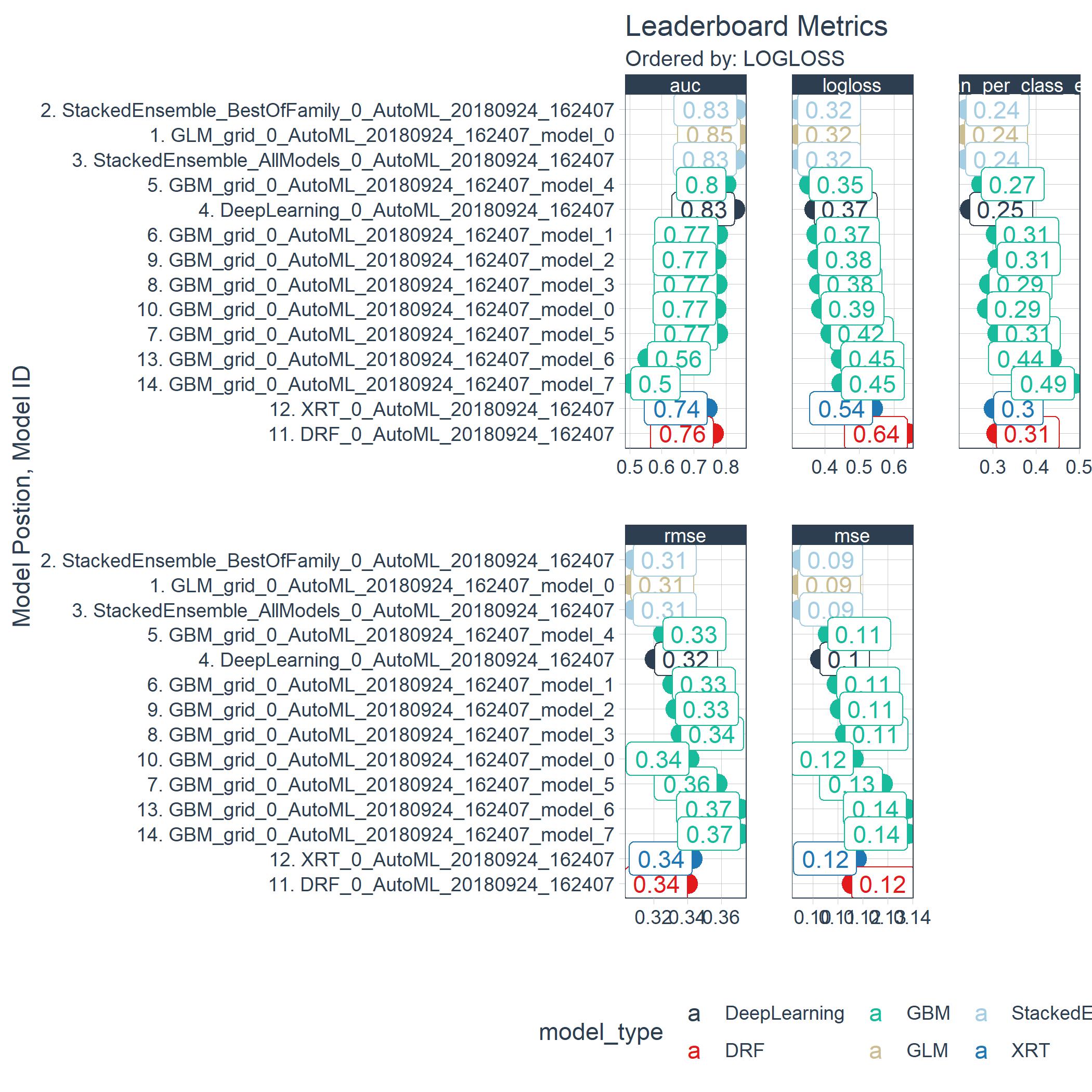
[plot\_hist](#plot_hist)



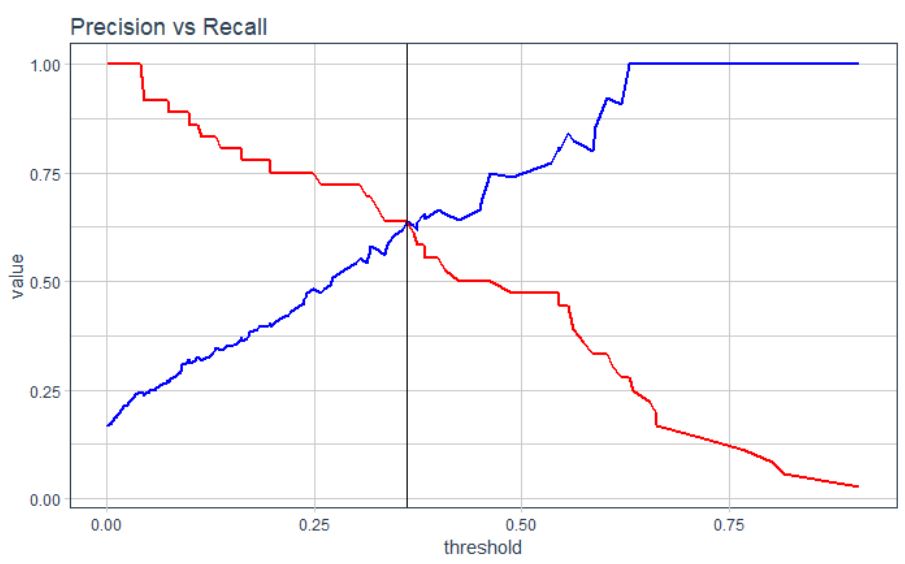
[plot\_cor](#plot_cor)



[plot\_leaderboard](#plt_leaderboard)



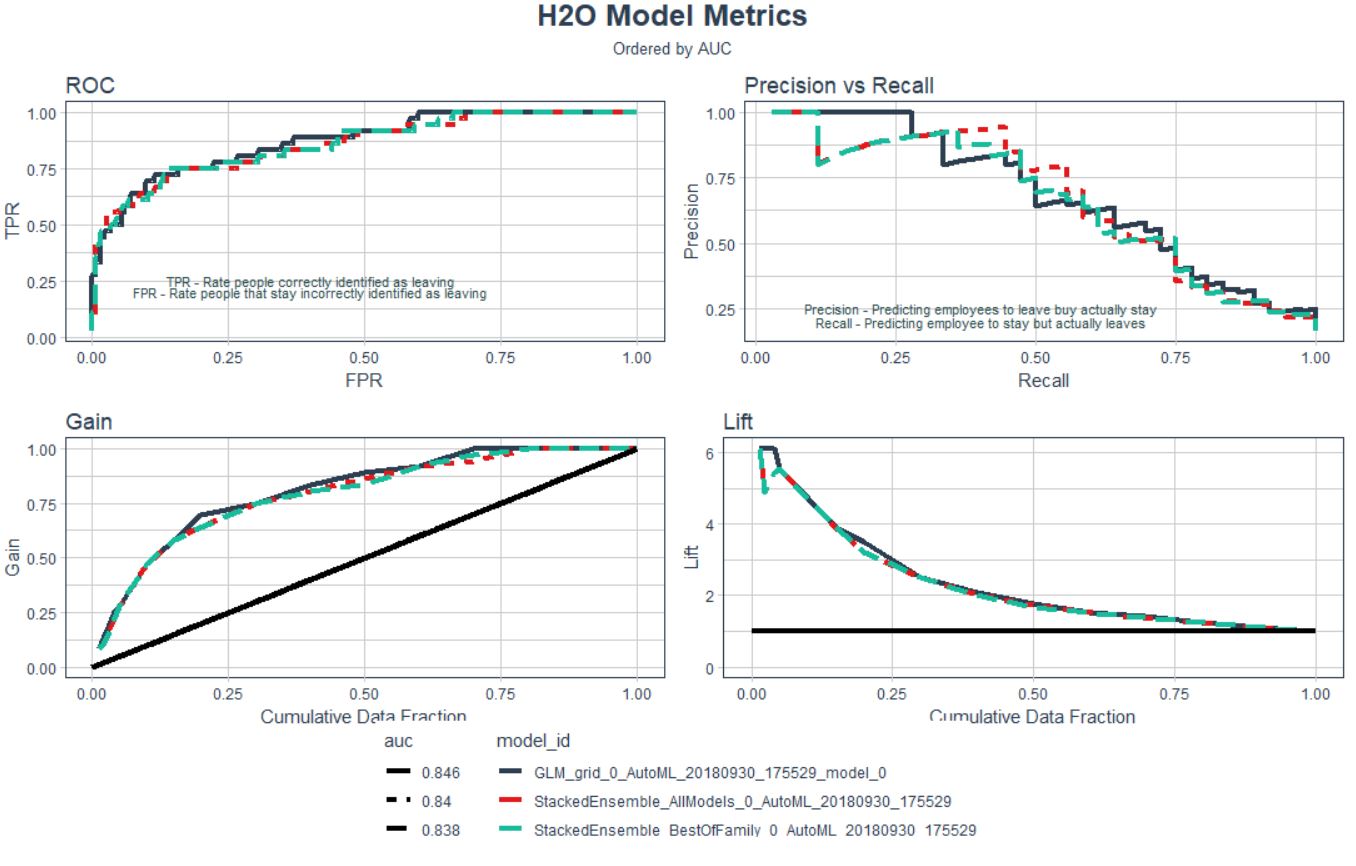
[Precision\_Recall\_plot](#precisionrecallplot)



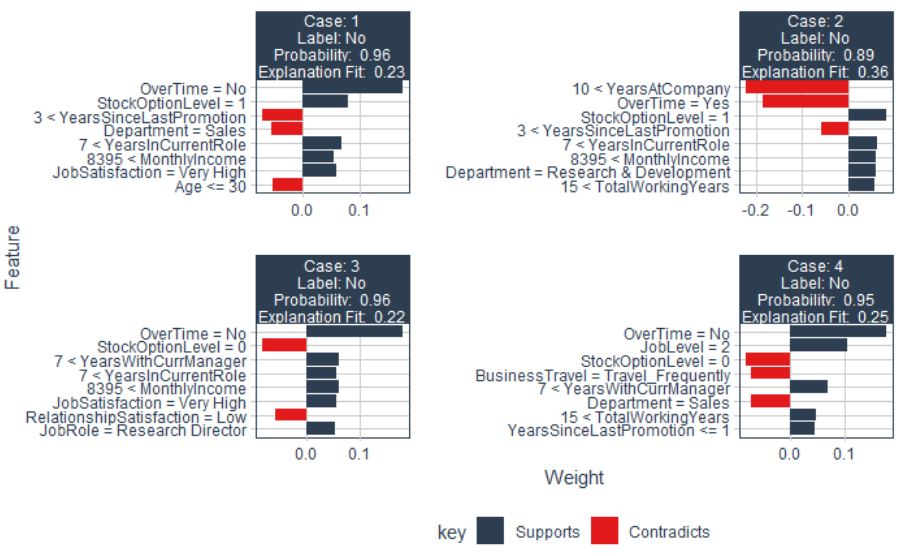
[ROC\_Plot](#ROCplot)



[H2O\_Performance\_Plot](#ploth20perf)



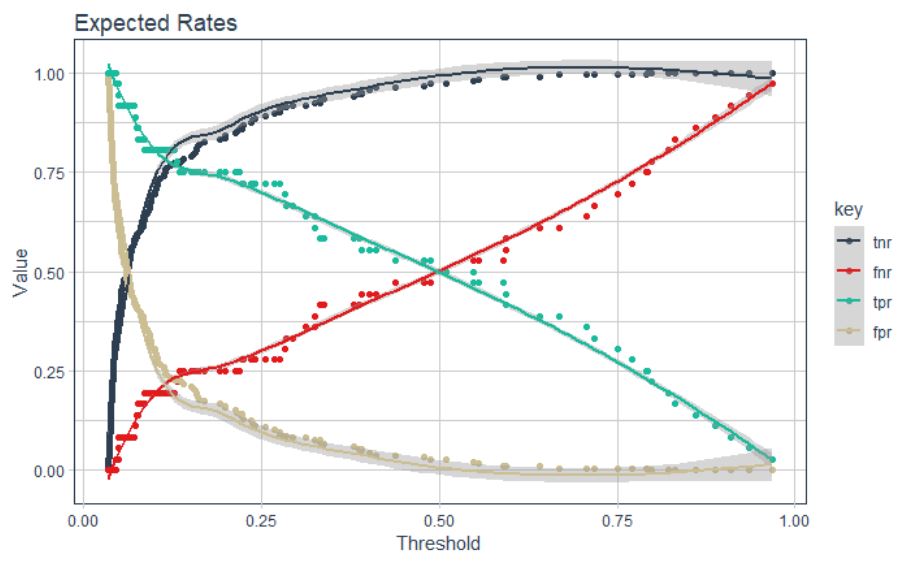
[Plot\_Lime\_Feature\_Explain](#plotlimefeaturetq)



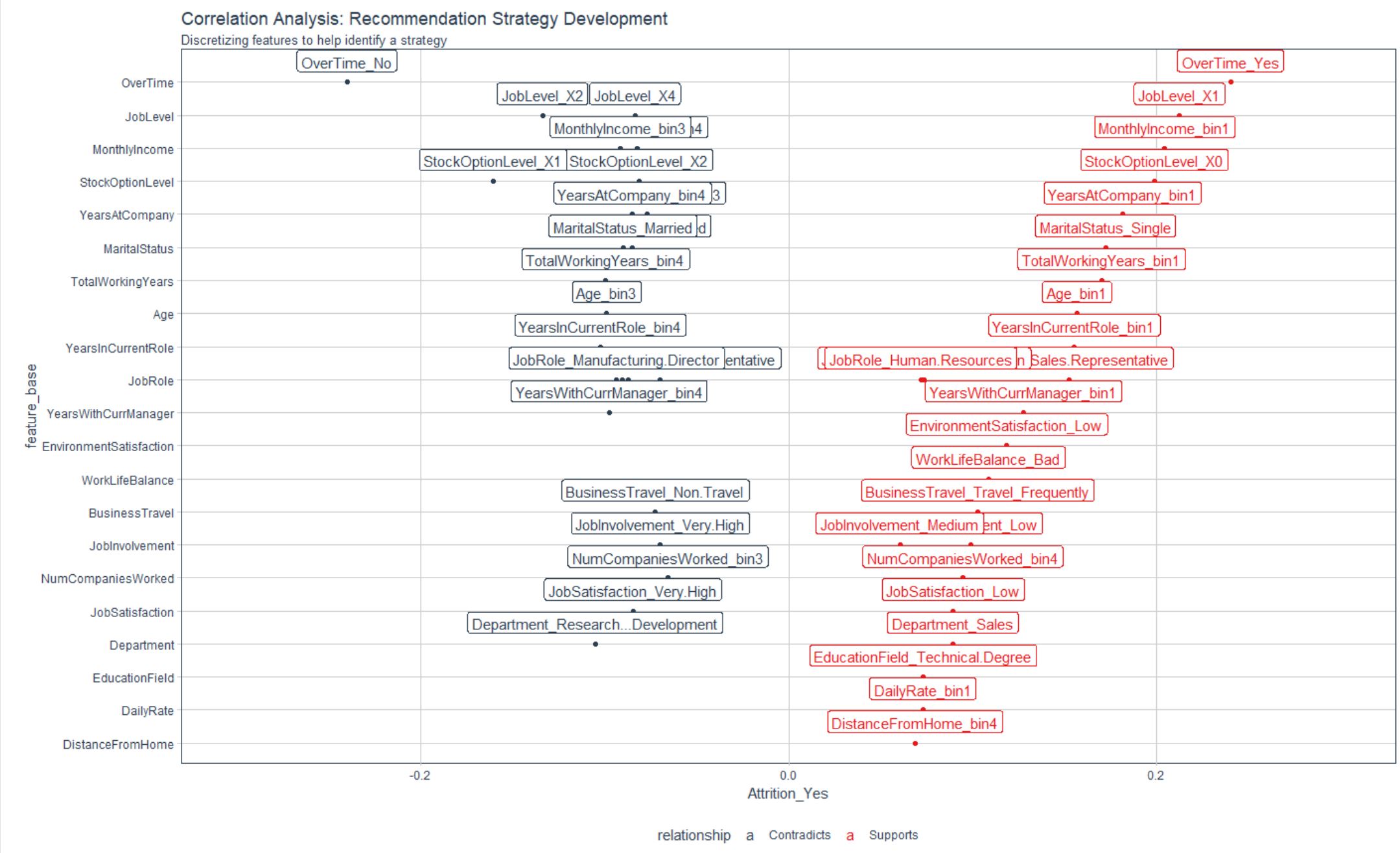
[Plot\_Lime\_Features\_Explanations](#plotlimefeaturestq)



[Confusion\_Matrix\_Plot](#confusionMatrixPlot)

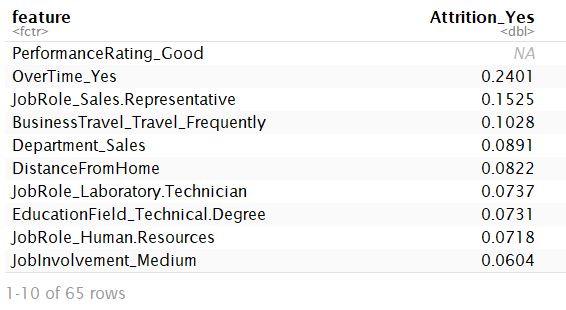


[Correlation\_Analysis](#correlationAnalysis)

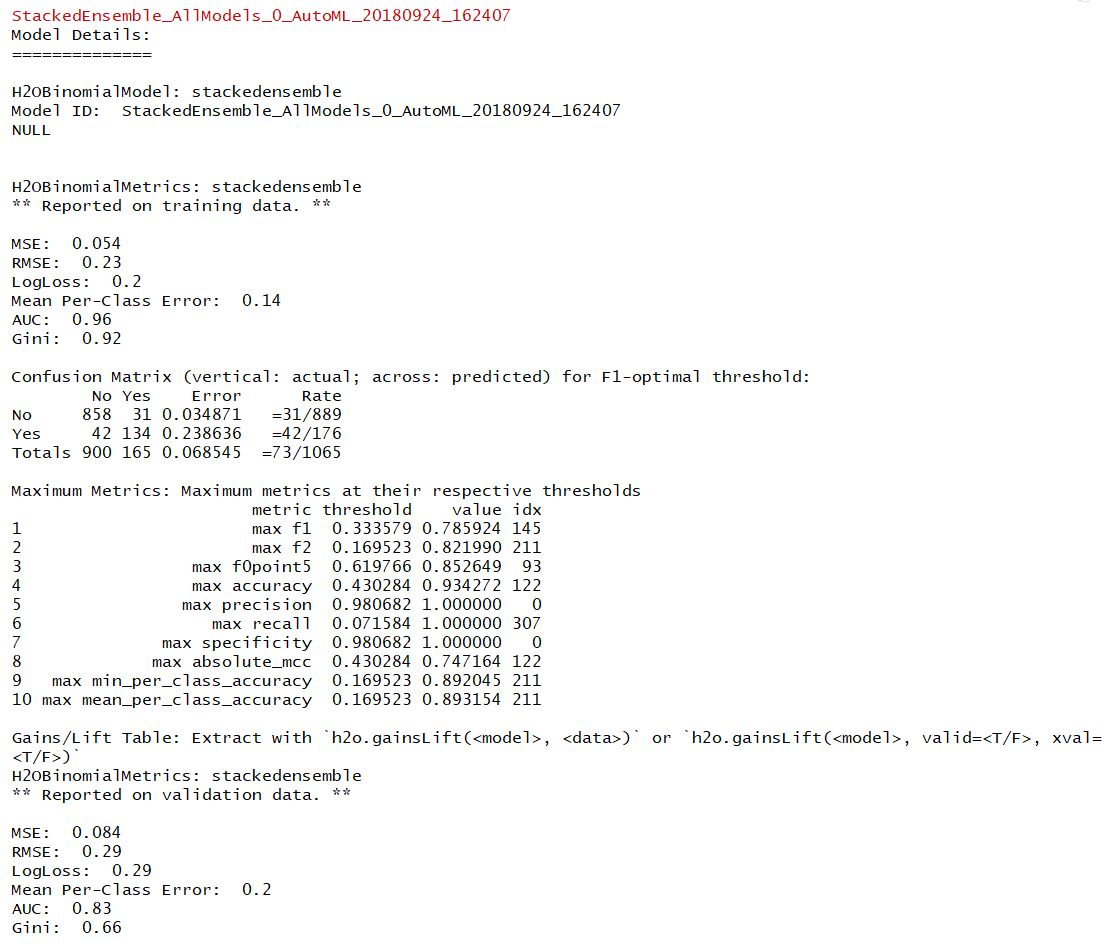


Functions

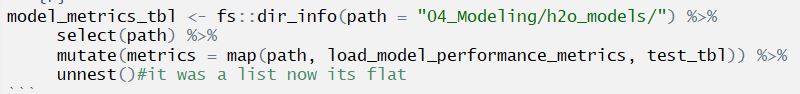
[get\_cor](#get_cor)



[extract\_h2o\_model\_name\_by\_position](#extract_model)



[Model\_Metrics](#modelmetrics)



[calculate\_savings\_by\_thresh\_2](#calculate_savings_by_thresh_2)

Code Snippets

[Tidy\_Eval](#Tidy_Eval) [Tidy\_Eval](#tidy_eval2)

quos, enquo and !!!

[rlang](#rlang)

rlang::sym turns a single character string into an expression. The expression is typically captured in enquo() or quos() to delay evaluation

[anonymous](#anonymous)

If you choose not to give the function a name, you get an [anonymous function](http://adv-r.had.co.nz/Functional-programming.html#anonymous-functions).

[fct\_recode](#fct_recode)

mutate(OverTime = fct\_recode1(OverTime, ‘No’ = ‘Yes’)

[case\_when](#case_when)

mutate(above\_industry\_avg = case\_when(pct > 0.088 ~ ‘Yes’, TRUE ~ ‘No’)

[gather\_example](#gather_example)

Gather takes multiple columns and collapses into key-value pairs

[fct\_reorder2](#fct_reorder2)

fct\_reorder() is useful for 1d displays where the factor is mapped to position; fct\_reorder2() for 2d displays where the factor is mapped to a non-position aesthetic

[add\_column](#add_column)

This is a convenient way to add one or more columns to an existing data frame.

[partial](#partial)

partial allows you to preload a function with arguments that do not change. Part of purrr

[firstMap](#firstMap) [secondMap](#secondMap)

mutate and map are a powerful combination enabling row-wise iteration in a tidy way:  
mutate(metrics = map(model\_id, get\_model\_performance\_metrics, newdata\_tbl))

[expand\_limits](#expand_limits)

Change the range of axis values in ggplot: expand\_limits(x = c(-1, 1))

[scale\_x\_continuous](#scale_x_continuous)

scale\_x\_continuous(label = scales::dollar)

[cross\_df](#cross_df)

cross\_df from purr is really helpful. From a list of variables, it produces all combinations of the list.

[scale\_fill\_gradient2](#scale_fill_gradient2)

scale\_fill\_gradient2 uses 2 gradient transitions by specifying low, mid and high arguments along with a midpoint to specify the middle transition point. There are variations like scale\_fill\_gradient that offers only a low and high point. scale\_fill\_gradient\_n allows for any number of gradients.

[str\_detect](#str_detect)

filter(str\_detect(terms, ‘YearsAtCompany’))

# Employee Turnover: A $15M Problem

Differentiating between productive and non-productive employees is difficult and highly subjective. Some employees are in the top 20% (all stars), some are in the middle 60% (productive), and some are in the bottom 20%. The bottom 20% can be OK or can be poor depending on the overall level of talent in your organization. Sometimes just because they are bottom 20% they are still good employees because the organization is such a high performing organization that everyone they employee is good. However, most organizations do have bad employees.

Now, in terms of defining productivity, if on a 1 to 4 scale with 1 being poor and 4 being the best that managers rate employees 3 to 4, you can usually be OK in viewing the 4’s as productive. They may not always be all stars, especially if 50% of people reviewed get 4’s but in reality only 20% should by the bell curve. However, if a manager rates someone as a 1, there is definitely a problem. These people need to either have immediate corrective actions taken or need to be cut loose. This is good attrition because the employee is a poor job/culture fit.

The analysis we are doing specifically looks at bad attrition, losing people that are 3’s and 4’s. I call them high performers, but in reality, I should be saying productive employees. This group is probably driving 80% of the business results.

Non-profits will need to be handled based on what the business finds important. While maximizing profit may not be the goal, there are still goals. For example, one goal might be to increase high school graduation rates in schools. This could be set up showing that the percentage of student dropouts are much higher in the district of focus and that increasing graduation rates is expected to reduce crime and increase workforce productivity. Crime reduction would have a savings associated with it. Workforce productivity would have a financial benefit associated with it. These are possible ways we can show financial value for a non-profit.

Non-Profit Cost Analysis: <https://www.bridgespan.org/bridgespan/Images/articles/nonprofit-cost-analysis-toolkit/NonprofitCostsAnalysisToolkit.pdf>

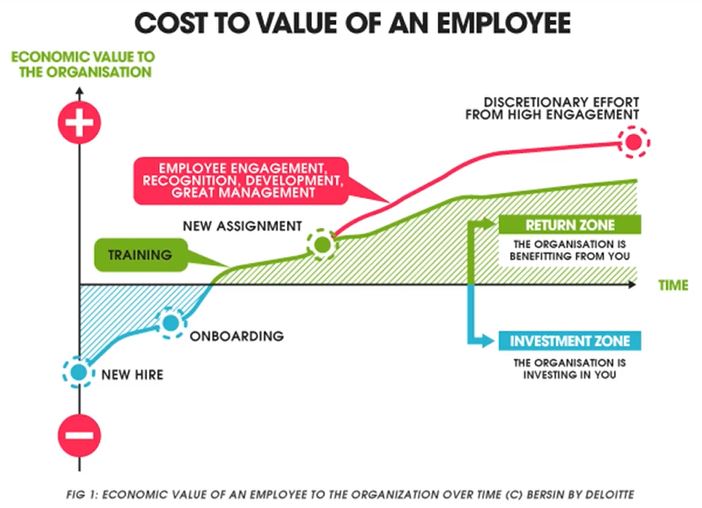
Non-Profit Strategic Planning: <https://knowhownonprofit.org/organisation/strategy>

The tools are quite similar to what you might find in a for-profit scenario. The maximization will be to maximize the financial delta between cost and benefit (to public) versus cost and revenue (for organization).

## What Happens When Good Employees Leave

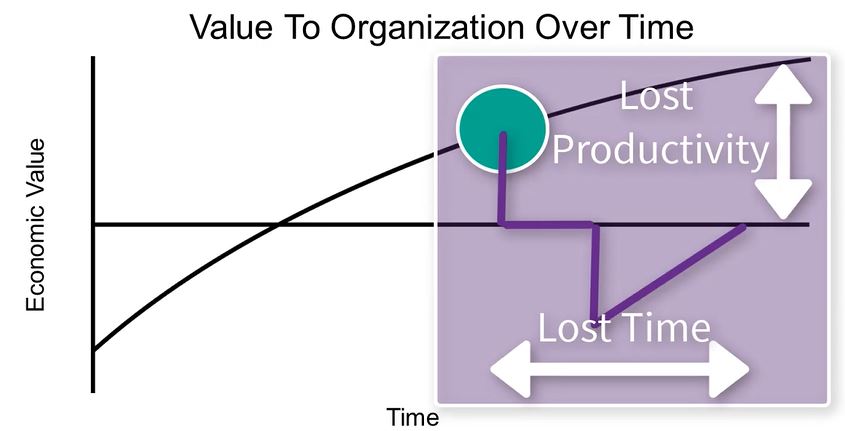
*You takeaway our top 20 employees and we become a mediocre company* –Bill Gates

* All employees turnover, some just faster
* Impact of Attrition
  + Varies based on employee productivity
  + Can be measured
* Bad turnover costs the organization



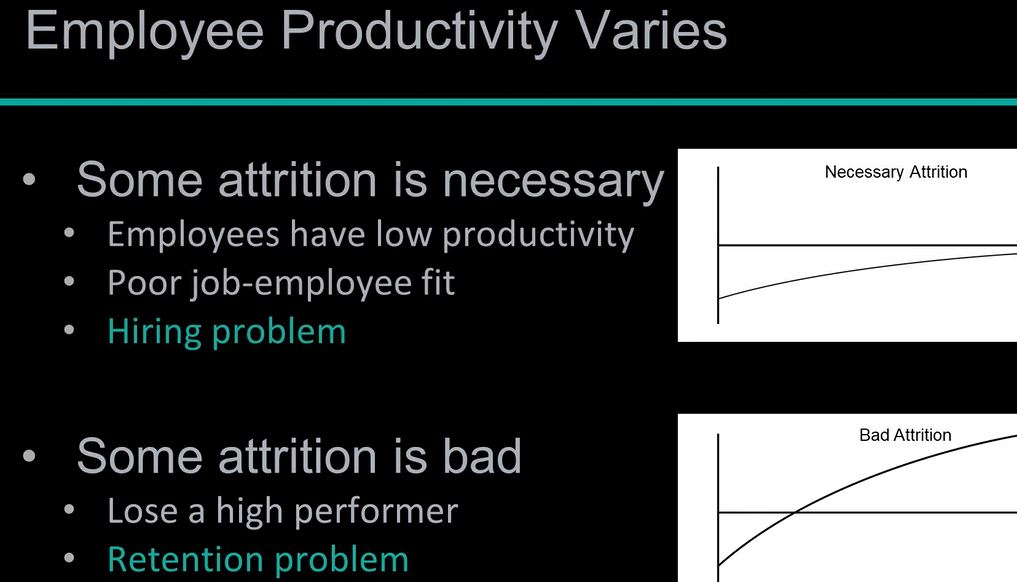
[Employee Retention Now a Big Issue: Why the Tide has Turned](https://www.linkedin.com/pulse/20130816200159-131079-employee-retention-now-a-big-issue-why-the-tide-has-turned/)

When a good employee quits, costs are incurred.



* Quits
* Vacancy during search
* Cost to train new employee

Not all attrition is bad



## Calculating the Cost of Turnover

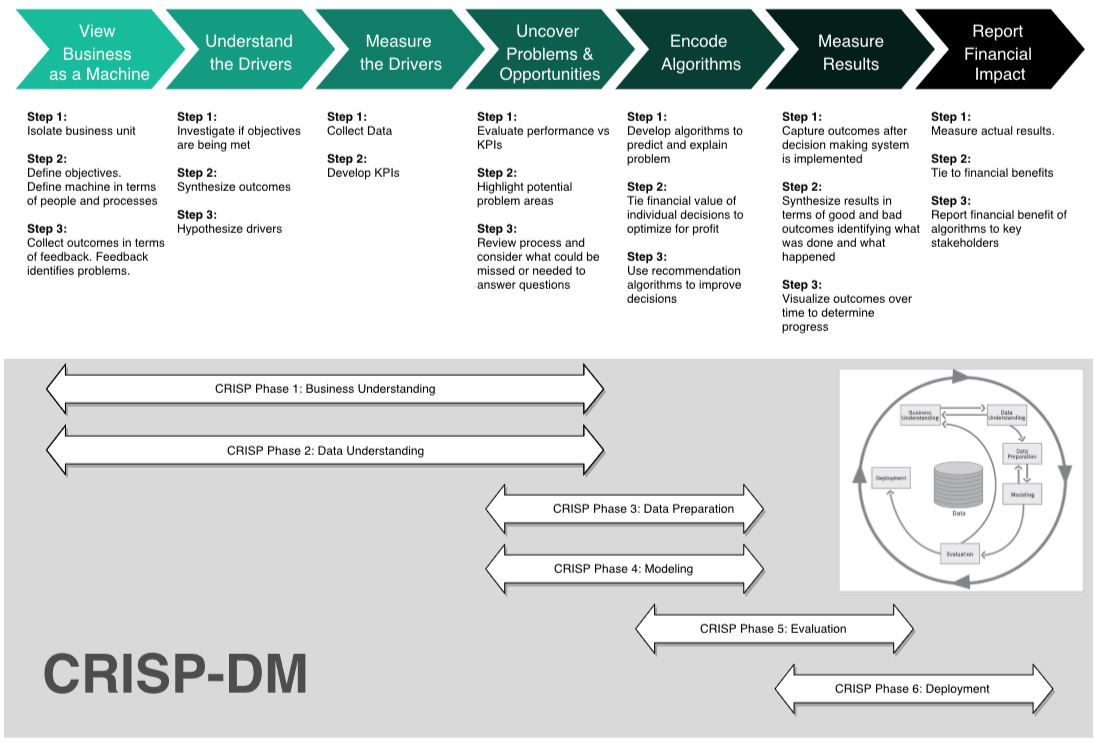
* Direct Costs
  + Average Separation (Exit Interviews, Separation Pay, Admin Costs) ($500)
  + Average Vacancy (Temp Help & OT) ($10,000)
  + Average Acquisition (Ads, Travel, Interviews, Physicals, Drug Testing, Bonuses, Reference Checks, etc) ($4,900)
  + Average Placement (New Supplies, Onboarding, Training) ($3,500)
* Productivity Costs (Hidden Costs - only recognized when the problem becomes large. Much higher than direct costs.)
  + Annual revenue per Employee - less COGS (250,000)
  + Workdays per Year (240)
  + Average Workdays Position Open (40)
  + Average Onboarding/Training Period (60 ~ 12 weeks)
  + Effectiveness During Onboarding/Training (50%)
* Salary & Benefits
  + Average Salary + Benefits (80,000)
  + Workdays per Year (240)
  + Average Workdays Position Open (40)

If there 200 productive employees leave each year, then is the financial cost of turnover.

Just a 10% reduction (using data science) saves the organization $1.6M/year!

# Toolbox Supporting Analysis

## Integrated Data Science Frameworks: BSPF & CRISP-DM

Business Science Problem Framework 

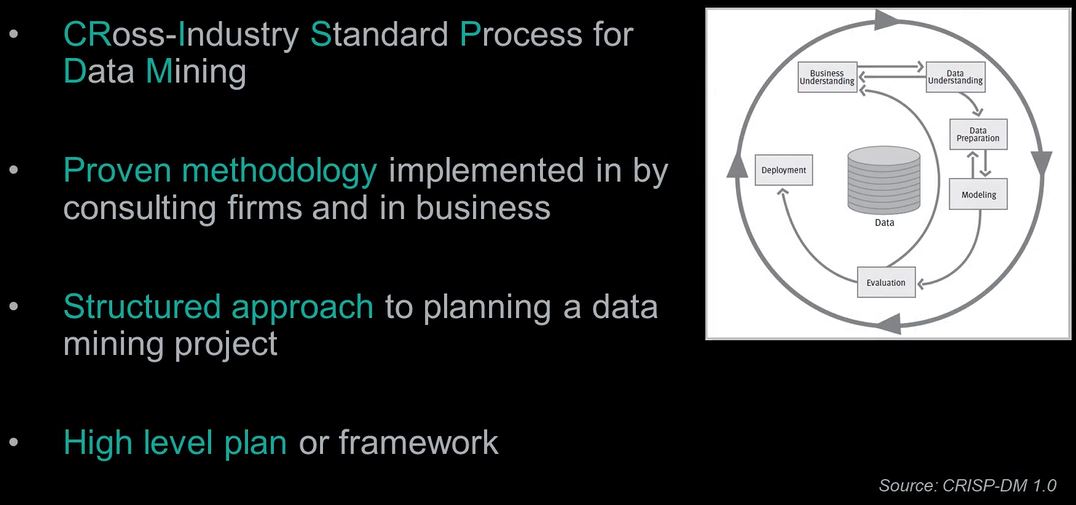
* Framework provide
  + Consistency across projects
  + Critical Step Checklist
  + Clarity to stakeholders
  + Keeps project on track

BSPF is aligned with industry standard CRISP-DM

# Frameworks

## CRISP-DM

[CRISP-DM](https://www.the-modeling-agency.com/crisp-dm.pdf) is a high level industry-standard framework for data mining.



A high level plan that is adaptable but lacks the detail to drive data science projects

CRISP-DM Process Cycle

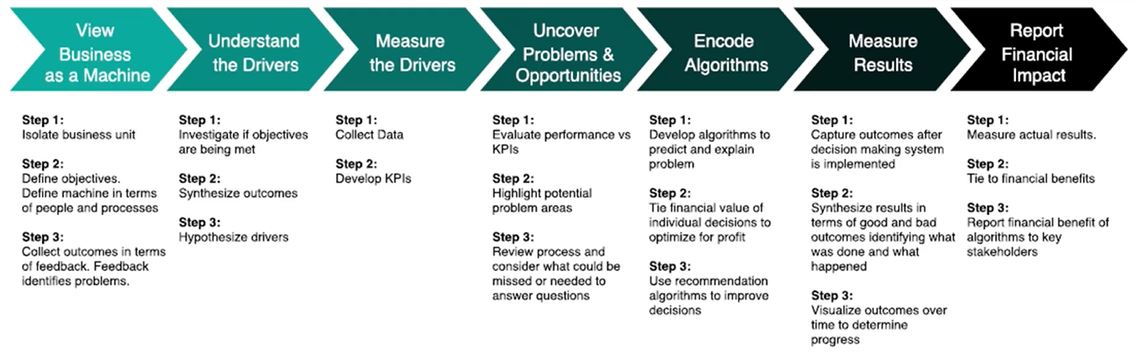
1. Business Understanding
   * Critical - sets stage for all downstream events
   * Problem solving and project scope development with the business
   * Combines discovery and project management
   * Documents current state and the business success metric
2. Data Understanding
   * 20% of project time typically consumed by data collection
3. Data Preparation
   * 60% of your time cleaning and organizing data
4. Modeling
   * information turned into insight
   * Model Understanding
   * Model Assessment
5. Evaluation
   * Assess degree which the data insights meet business objectives (ROI)
6. Deployment
   * REST API
   * Shiny App
   * Markdown file output

## Business Science Problem Framework

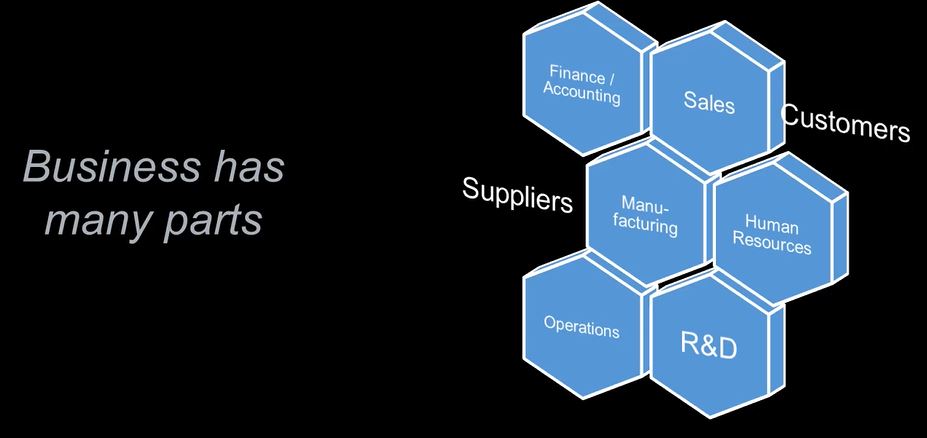
The goal is to make good decisions. Systematic decision making starts with measurement and analysis which leads to improvement.

The DSBF is based on [Principles by Ray Dalio](https://www.principles.com/)

BSPF has 7 steps:



1. Business as a Machine



Each part of the business has Objectives –> Machine –> Outcomes.

* Objectives are the goals that serve a inputs or instructions to the machine
* Machine - performs the work and is made of people and processes
  + If People or process is not right, the machine produces questionable output
  + People have biases
  + Process issues abound in all organizations
  + People/process design fit may be poor
* Outcomes - what the machine produces
  + Provides feedback to determine is the objectives are being met

Our goal is to understand the machine using data

When you have exhausted all possibilities, remember this - you haven’t.

* Thomas Jefferson

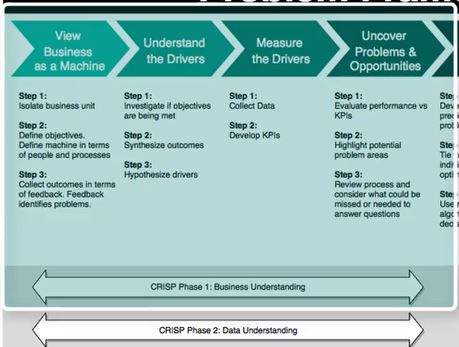
# Business Understanding: BSPF & Code Workflows

## Problem Understanding With BSPF

### Business Understanding

The BSPF includes the following steps of the the CRISP-DM Framework:

1. View the Business as a Machine
2. Understand the Drivers
3. Measure the Drivers
4. Uncover Problems & Opportunities



### View The Business As A Machine

Review business\_understanding.R file in 01\_Business Understanding folder

Business Units: Department and Job Role Define Objectives: Retain high performers Assess Outcomes: TBD - Feedback from the business

*Known* - 16.1% attrition in the current state - is this a bad thing?

## Understand The Drivers: By Dept

Investigate the Objectives: 16% Attrition Synthesize Outcomes: High counts and high percentages (learned after hypothesizing drivers). See below under *Understand the Drivers*. Hypothesize Drivers: Job Role and Departments

## # A tibble: 6 x 4  
## # Groups: Department [3]  
## Department Attrition n pct  
## <chr> <chr> <int> <dbl>  
## 1 Human Resources No 37 0.755  
## 2 Human Resources Yes 12 0.245  
## 3 Research & Development No 721 0.867  
## 4 Research & Development Yes 111 0.133  
## 5 Sales No 291 0.789  
## 6 Sales Yes 78 0.211

Department and Job Roles are common cohorts to evaluate.

HR has 24% attrition, Sales 21% - might be something going on by department.

## Understand The Drivers: By Job Role

dept\_job\_role\_tbl %>% group\_by(Department, JobRole, Attrition) %>% summarise(n = n()) %>% ungroup() %>%   
 group\_by(Department, JobRole) %>% mutate(pct = n/sum(n)) %>% ungroup() %>% filter(Attrition %in% c("Yes"))

## # A tibble: 10 x 5  
## Department JobRole Attrition n pct  
## <chr> <chr> <chr> <int> <dbl>  
## 1 Human Resources Human Resources Yes 12 0.308   
## 2 Research & Development Healthcare Representative Yes 8 0.0762  
## 3 Research & Development Laboratory Technician Yes 49 0.219   
## 4 Research & Development Manager Yes 2 0.0417  
## 5 Research & Development Manufacturing Director Yes 7 0.0569  
## 6 Research & Development Research Director Yes 2 0.0274  
## 7 Research & Development Research Scientist Yes 43 0.166   
## 8 Sales Manager Yes 2 0.0645  
## 9 Sales Sales Executive Yes 50 0.183   
## 10 Sales Sales Representative Yes 26 0.4

# Used Attrition %in% instead of == because the former is more flexible - can add mutiple categories : c("cat1", "cat2", . . .)

Note there are different roles with each department. Perhaps certain roles have greater attrition issues than others?  
Note the HR 30%; Note Sales Executive - only 18.3% but there are 50 leaving; Sales Representative may be a problem too.

## Measure The Drivers: Collect Data

Now in the BSPF - Measure the Drivers Step

This is an iterative task that is ongoing.

You may not have all the data initially. Further, you don’t want all the data since many of the features are a distraction prior to understanding if a problem exists. During the Business Understanding phase we need to isolate the critical features that relate to business outcomes and determine if a problem exists such that when solved it provides real value in terms of financial benefit.

There are 35 features to consider:

names(train\_raw\_tbl)

## [1] "Age" "Attrition"   
## [3] "BusinessTravel" "DailyRate"   
## [5] "Department" "DistanceFromHome"   
## [7] "Education" "EducationField"   
## [9] "EmployeeCount" "EmployeeNumber"   
## [11] "EnvironmentSatisfaction" "Gender"   
## [13] "HourlyRate" "JobInvolvement"   
## [15] "JobLevel" "JobRole"   
## [17] "JobSatisfaction" "MaritalStatus"   
## [19] "MonthlyIncome" "MonthlyRate"   
## [21] "NumCompaniesWorked" "Over18"   
## [23] "OverTime" "PercentSalaryHike"   
## [25] "PerformanceRating" "RelationshipSatisfaction"  
## [27] "StandardHours" "StockOptionLevel"   
## [29] "TotalWorkingYears" "TrainingTimesLastYear"   
## [31] "WorkLifeBalance" "YearsAtCompany"   
## [33] "YearsInCurrentRole" "YearsSinceLastPromotion"   
## [35] "YearsWithCurrManager"

### Group Features

Some features related to one and other.

It is always helpful to breakdown data collection activities in to strategic areas.

* Descriptive Features:
  + Age, DistanceFromHome, Gebder, MaritalStatus, NumComaniesWorked, Over18
* Employment Features:
  + Department, EmployeeCount, EmployeeNumber, JobInvolement, JobLevel, JobRole, JobSatistfaction
* Compensation Features:
  + DailyRate, HourlyRate, MonthlyIncome, MonthlyRate, PercentSalaryHike, StockOptionLevel
* Survey Results:
  + EnvironmentSatisfaction, JobSatisfaction, RelationshipSatisfaction, WorkLifeBalance
* Performance Data:
  + JobInvolvement, PerformanceRating
* Work-Life Balance
  + BusinessTravel, OverTime
* Training and Education
  + Education, EducationField, TrianingTimesLastYear
* Time-Based Features
  + TotalWorkingYears, YearsAtCompany, YeasInCurrentRole, YearsSinceLastPromotion, YearsWithCurrentManager

## Measure The Drivers: Develop KPIs

Often review turnover statistics in your industry. We know our is 16% but we do not know if this is good or bad without [research](<http://www.air-prehire.com/blog/industries-high-employee-turnover-rates/>.  
Perhaps it simply fails an internal KPI.

Via web searching, *8.8% average turnover for utility companies* (useful for this data). (Banking is around 17.2%) {#case\_when}

## # A tibble: 10 x 6  
## Department JobRole Attrition n pct above\_industry\_~  
## <chr> <chr> <chr> <int> <dbl> <chr>   
## 1 Sales Sales Represen~ Yes 26 0.4 Yes   
## 2 Human Resources Human Resources Yes 12 0.308 Yes   
## 3 Research & Dev~ Laboratory Tec~ Yes 49 0.219 Yes   
## 4 Sales Sales Executive Yes 50 0.183 Yes   
## 5 Research & Dev~ Research Scien~ Yes 43 0.166 Yes   
## 6 Research & Dev~ Healthcare Rep~ Yes 8 0.0762 No   
## 7 Sales Manager Yes 2 0.0645 No   
## 8 Research & Dev~ Manufacturing ~ Yes 7 0.0569 No   
## 9 Research & Dev~ Manager Yes 2 0.0417 No   
## 10 Research & Dev~ Research Direc~ Yes 2 0.0274 No

Note the Dept and Job Role with high percentages and counts. To make this more impactful, add cost to this table.

## Uncover Problems & Opportunities

### calculate\_attrition\_cost()

Create a function the mimics the Excel Calculator.

The function includes default values that equal the defaults that were introduced in *Calculating the Cost of Turnover*. So if we call the custom function calculate\_attrition\_cost with the default values, it returns 78483.33. If we change the default value of n = 1 (representing 1 employee) and change it to n = 200 then the function returns 15696666.67.

### Calculating Cost By Job Level

Using code that we have already seen above and modified it slightly to focus on JobRole rather than Department, we can begin this part of the journey reviewing JobRole data. Note the we reuse a custom function - calculate\_attrition\_cost to return the cost\_of\_attrition column below.

dept\_job\_role\_tbl %>% count(JobRole, Attrition) %>% group\_by(JobRole) %>% mutate(pct = n/sum(n)) %>% ungroup() %>%   
 filter(Attrition %in% c("Yes")) %>% arrange(desc(pct)) %>% mutate(above\_industry\_avg = case\_when(pct > 0.088 ~ "Yes", TRUE ~ "No")) %>%   
 mutate(cost\_of\_attrition = calculate\_attrition\_cost(n = n, salary = 80000))

## # A tibble: 9 x 6  
## JobRole Attrition n pct above\_industry\_a~ cost\_of\_attriti~  
## <chr> <chr> <int> <dbl> <chr> <dbl>  
## 1 Sales Represe~ Yes 26 0.4 Yes 2040567.  
## 2 Human Resourc~ Yes 12 0.308 Yes 941800.  
## 3 Laboratory Te~ Yes 49 0.219 Yes 3845683.  
## 4 Sales Executi~ Yes 50 0.183 Yes 3924167.  
## 5 Research Scie~ Yes 43 0.166 Yes 3374783.  
## 6 Healthcare Re~ Yes 8 0.0762 No 627867.  
## 7 Manufacturing~ Yes 7 0.0569 No 549383.  
## 8 Manager Yes 4 0.0449 No 313933.  
## 9 Research Dire~ Yes 2 0.0274 No 156967.

Note the costs associated with Sales Executive and Laboratory Technician.

## Streamlining The Attrition Code Workflow

This section is dedicated to streamlining the code developed above.

### Streamlining The Counts

This is code commonly used:

dept\_job\_role\_tbl %>%   
   
 group\_by(JobRole, Attrition) %>% summarise(n = n())

## # A tibble: 18 x 3  
## # Groups: JobRole [?]  
## JobRole Attrition n  
## <chr> <chr> <int>  
## 1 Healthcare Representative No 97  
## 2 Healthcare Representative Yes 8  
## 3 Human Resources No 27  
## 4 Human Resources Yes 12  
## 5 Laboratory Technician No 175  
## 6 Laboratory Technician Yes 49  
## 7 Manager No 85  
## 8 Manager Yes 4  
## 9 Manufacturing Director No 116  
## 10 Manufacturing Director Yes 7  
## 11 Research Director No 71  
## 12 Research Director Yes 2  
## 13 Research Scientist No 216  
## 14 Research Scientist Yes 43  
## 15 Sales Executive No 223  
## 16 Sales Executive Yes 50  
## 17 Sales Representative No 39  
## 18 Sales Representative Yes 26

The code above is replaced with this simpler code:

dept\_job\_role\_tbl %>% count(JobRole, Attrition)

## # A tibble: 18 x 3  
## JobRole Attrition n  
## <chr> <chr> <int>  
## 1 Healthcare Representative No 97  
## 2 Healthcare Representative Yes 8  
## 3 Human Resources No 27  
## 4 Human Resources Yes 12  
## 5 Laboratory Technician No 175  
## 6 Laboratory Technician Yes 49  
## 7 Manager No 85  
## 8 Manager Yes 4  
## 9 Manufacturing Director No 116  
## 10 Manufacturing Director Yes 7  
## 11 Research Director No 71  
## 12 Research Director Yes 2  
## 13 Research Scientist No 216  
## 14 Research Scientist Yes 43  
## 15 Sales Executive No 223  
## 16 Sales Executive Yes 50  
## 17 Sales Representative No 39  
## 18 Sales Representative Yes 26

count is part of dplyr!

Do not forget about the dplyr::tally function - very useful

*tally() is a convenient wrapper for summarize that will either call n() or sum(n) depending on whether you’re tallying for the first time, or re-tallying. count() is similar but calls group\_by() before and ungroup() after.*

### Streamlining The Count To Percentage Calculation

Pay attention - this is really good code using tidy evaluation!

We have this code so far in streamlining the attrition code:

dept\_job\_role\_tbl %>% count(JobRole, Attrition) %>% group\_by(JobRole) %>% mutate(pct = n/sum(n))

## # A tibble: 18 x 4  
## # Groups: JobRole [9]  
## JobRole Attrition n pct  
## <chr> <chr> <int> <dbl>  
## 1 Healthcare Representative No 97 0.924   
## 2 Healthcare Representative Yes 8 0.0762  
## 3 Human Resources No 27 0.692   
## 4 Human Resources Yes 12 0.308   
## 5 Laboratory Technician No 175 0.781   
## 6 Laboratory Technician Yes 49 0.219   
## 7 Manager No 85 0.955   
## 8 Manager Yes 4 0.0449  
## 9 Manufacturing Director No 116 0.943   
## 10 Manufacturing Director Yes 7 0.0569  
## 11 Research Director No 71 0.973   
## 12 Research Director Yes 2 0.0274  
## 13 Research Scientist No 216 0.834   
## 14 Research Scientist Yes 43 0.166   
## 15 Sales Executive No 223 0.817   
## 16 Sales Executive Yes 50 0.183   
## 17 Sales Representative No 39 0.6   
## 18 Sales Representative Yes 26 0.4

Calculating percentages from counts is a *common programming requirements*. Why not create a function that generalizes this?

count\_to\_pct <- function(data, ..., col = n){  
   
 grouping\_vars\_expr <- quos(...)  
 col\_expr <- enquo(col)  
   
 ret <- data %>% group\_by(!!! grouping\_vars\_expr) %>%   
 mutate(pct = (!! col\_expr)/sum(!! col\_expr)) %>% ungroup()  
   
 return(ret)   
}

* tidy functions always requires the data to take the first position of the function{#Tidy\_Eval}
* ... enables passing multiple, un-named arguments to a function
  + Because the dots are not preselected, the user can flexibly add variables and the function will adapt!
* The 3rd function argument is a column name, in this case n since we are calculating a percentage from the counts in the n column

1. ... must be enclosed on quos or enquo for multiple columns or single columns, respectfully. The saves the column names as unevalutated expressions. They can be called later in the processing. Use quos for multiple columns and enquo for the single column we have called n
2. !!! called bang-bang-bang splices multiple grouping variables and evaluating them. (Use !! for single columns.)

Demo:

dept\_job\_role\_tbl %>% count(Department,JobRole, Attrition) %>% count\_to\_pct(Department, JobRole)

## # A tibble: 21 x 5  
## Department JobRole Attrition n pct  
## <chr> <chr> <chr> <int> <dbl>  
## 1 Human Resources Human Resources No 27 0.692   
## 2 Human Resources Human Resources Yes 12 0.308   
## 3 Human Resources Manager No 10 1   
## 4 Research & Development Healthcare Representative No 97 0.924   
## 5 Research & Development Healthcare Representative Yes 8 0.0762  
## 6 Research & Development Laboratory Technician No 175 0.781   
## 7 Research & Development Laboratory Technician Yes 49 0.219   
## 8 Research & Development Manager No 46 0.958   
## 9 Research & Development Manager Yes 2 0.0417  
## 10 Research & Development Manufacturing Director No 116 0.943   
## # ... with 11 more rows

### Streamlining The Attrition Assessment

Continuing the development of a streamlined function, review where we are so far:

dept\_job\_role\_tbl %>% count(Department, JobRole, Attrition) %>% count\_to\_pct(Department, JobRole) %>%   
 filter(Attrition %in% c("Yes")) %>% arrange(desc(pct)) %>%   
 mutate(above\_industry\_avg = case\_when(pct > 0.088 ~ "Yes", TRUE ~ "No"))

## # A tibble: 10 x 6  
## Department JobRole Attrition n pct above\_industry\_~  
## <chr> <chr> <chr> <int> <dbl> <chr>   
## 1 Sales Sales Represen~ Yes 26 0.4 Yes   
## 2 Human Resources Human Resources Yes 12 0.308 Yes   
## 3 Research & Dev~ Laboratory Tec~ Yes 49 0.219 Yes   
## 4 Sales Sales Executive Yes 50 0.183 Yes   
## 5 Research & Dev~ Research Scien~ Yes 43 0.166 Yes   
## 6 Research & Dev~ Healthcare Rep~ Yes 8 0.0762 No   
## 7 Sales Manager Yes 2 0.0645 No   
## 8 Research & Dev~ Manufacturing ~ Yes 7 0.0569 No   
## 9 Research & Dev~ Manager Yes 2 0.0417 No   
## 10 Research & Dev~ Research Direc~ Yes 2 0.0274 No

Generalize filter - arrange - mutate statements:

assess\_attrition <- function(data, attrition\_col, attrition\_value, baseline\_pct){  
   
 attrition\_col\_expr <- enquo(attrition\_col)  
 data %>% filter((!! attrition\_col\_expr) %in% attrition\_value) %>% arrange(desc(pct)) %>%   
 mutate(above\_industry\_avg = case\_when(pct > baseline\_pct ~ "Yes", TRUE ~ "No"))}

* As always, tidy eval starts with a data (tibble)
* Next argument is the name of the Attrition column - it may not always be called Attrition
* 3rd argument are the values of the attrition\_col - in our case the values are Yes and No.
* baseline\_pct is hopefully self-explanatory!

1. Recall enquo saves a single column name as an expression that can be evaluated later. This way the name of the Attrition column can be whatever the user chooses!
2. pct does not change because that is created consistently in the count\_to\_pct function. (Of course this could be changed as an enhancement!)
3. Simple substitution for 0.088 to baseline\_pct.

Use parenthesis to give Tidy Eval evaluations priority!

Function inputs in text format (attrition\_value = “Yes”) do not require Tidy Eval

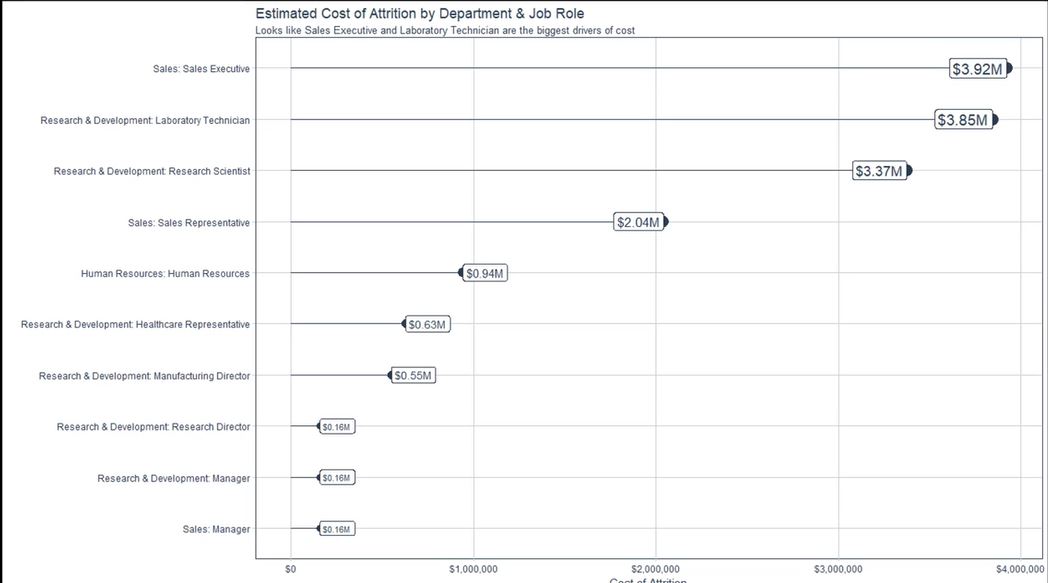
Demo the new code:

dept\_job\_role\_tbl %>%   
   
 count(Department, JobRole, Attrition) %>% count\_to\_pct(Department, JobRole) %>%   
 assess\_attrition(Attrition, attrition\_value = "Yes", baseline\_pct = 0.088) %>% #New code line  
 mutate(cost\_of\_attrition = calculate\_attrition\_cost(n = n, salary = 80000))

## # A tibble: 10 x 7  
## Department JobRole Attrition n pct above\_industry\_~  
## <chr> <chr> <chr> <int> <dbl> <chr>   
## 1 Sales Sales ~ Yes 26 0.4 Yes   
## 2 Human Res~ Human ~ Yes 12 0.308 Yes   
## 3 Research ~ Labora~ Yes 49 0.219 Yes   
## 4 Sales Sales ~ Yes 50 0.183 Yes   
## 5 Research ~ Resear~ Yes 43 0.166 Yes   
## 6 Research ~ Health~ Yes 8 0.0762 No   
## 7 Sales Manager Yes 2 0.0645 No   
## 8 Research ~ Manufa~ Yes 7 0.0569 No   
## 9 Research ~ Manager Yes 2 0.0417 No   
## 10 Research ~ Resear~ Yes 2 0.0274 No   
## # ... with 1 more variable: cost\_of\_attrition <dbl>

## Visualizing Attrition With ggplot2

Create impactful visualization like this:



* The plot above combines Department and JobRole together as labels for the y-axis.
* Need to change the type of the string manipulated in #1 to a factor and order it from high to low based on cost\_of\_attrition to make the plot readable.
* Need to format the text labels to make the plot easy-to-read and professional!

### Data Manipulation For Visualization

Here is the code so far plus some data manipulation so we can produce the plot. {#fct\_reorder}

dept\_job\_role\_tbl %>%   
   
 count(Department, JobRole, Attrition) %>% count\_to\_pct(Department, JobRole) %>%   
 assess\_attrition(Attrition, attrition\_value = "Yes", baseline\_pct = 0.088) %>%   
 mutate(cost\_of\_attrition = calculate\_attrition\_cost(n = n, salary = 80000)) %>%   
  
 # Data Manipulation   
 mutate(name = str\_c(Department, JobRole, sep = ": ") %>% as\_factor()) %>% #LINE 1  
 mutate(name = fct\_reorder(name, cost\_of\_attrition)) %>% # LINE 2  
 mutate(cost\_text = str\_c("$", format(cost\_of\_attrition / 1e6, digits = 2), "M", sep = "")) # LINE 3

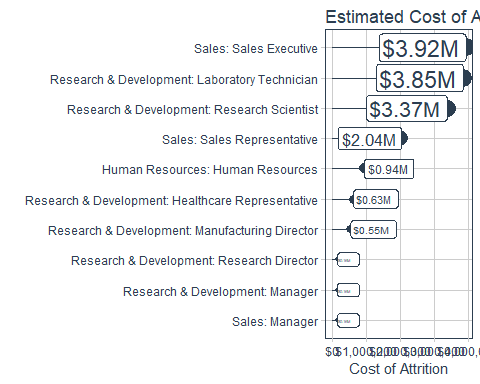
## # A tibble: 10 x 9  
## Department JobRole Attrition n pct above\_industry\_~  
## <chr> <chr> <chr> <int> <dbl> <chr>   
## 1 Sales Sales ~ Yes 26 0.4 Yes   
## 2 Human Res~ Human ~ Yes 12 0.308 Yes   
## 3 Research ~ Labora~ Yes 49 0.219 Yes   
## 4 Sales Sales ~ Yes 50 0.183 Yes   
## 5 Research ~ Resear~ Yes 43 0.166 Yes   
## 6 Research ~ Health~ Yes 8 0.0762 No   
## 7 Sales Manager Yes 2 0.0645 No   
## 8 Research ~ Manufa~ Yes 7 0.0569 No   
## 9 Research ~ Manager Yes 2 0.0417 No   
## 10 Research ~ Resear~ Yes 2 0.0274 No   
## # ... with 3 more variables: cost\_of\_attrition <dbl>, name <fct>,  
## # cost\_text <chr>

* LINE 1: str\_c from stringr concatenates Department and JobRole with a colon in between them.
  + factors are required in ggplot because they are used to order categorical variables (text will not work)
  + as\_factor is part of forcats. This function creates levels in the order in which they appear, which will be the same on every platform. (Base R sorts in the current locale which can vary from place to place.)
* LINE 2: fct\_reorder also from forcats reorders a factors numeric values by the magnitude of a different numeric value, in this case, cost\_of\_attrition.
* LINE 3: str\_c concatenation again - pretty simple - but effective!

### Visualization With ggplot2

OK, the data manipulation to plot the cost or attrition is complete. Now review the ggpolt code to produce the plot shown in the beginning on this section. It is easy - provided you do one step at a time!

dept\_job\_role\_tbl %>%   
   
 count(Department, JobRole, Attrition) %>% count\_to\_pct(Department, JobRole) %>%   
 assess\_attrition(Attrition, attrition\_value = "Yes", baseline\_pct = 0.088) %>%   
 mutate(cost\_of\_attrition = calculate\_attrition\_cost(n = n, salary = 80000)) %>%   
  
 # Data Manipulation   
 mutate(name = str\_c(Department, JobRole, sep = ": ") %>% as\_factor()) %>% #must be factors to order  
 mutate(name = fct\_reorder(name, cost\_of\_attrition)) %>% # orders the names by cost  
 mutate(cost\_text = str\_c("$", format(cost\_of\_attrition / 1e6, digits = 2), "M", sep = "")) %>%   
   
 # Plotting  
 ggplot(aes(x = cost\_of\_attrition, y = name)) +  
 geom\_segment(aes(xend = 0, yend = name), color = palette\_light()[[1]]) + # LINE 1  
 geom\_point(aes(size = cost\_of\_attrition), color = palette\_light()[[1]]) + # LINE 2  
 scale\_x\_continuous(label = scales::dollar) + # LINE 3  
 geom\_label(aes(label = cost\_text, size = cost\_of\_attrition), hjust = "inward", color = palette\_light()[[1]]) + # LINE 4  
 theme\_tq() + scale\_size(name = c(4,5)) + # LINE 5  
 labs(title = "Estimated Cost of Attrition: By Dept and Job Role", y = "", x = "Cost of Attrition") +  
 theme(legend.position = "none")

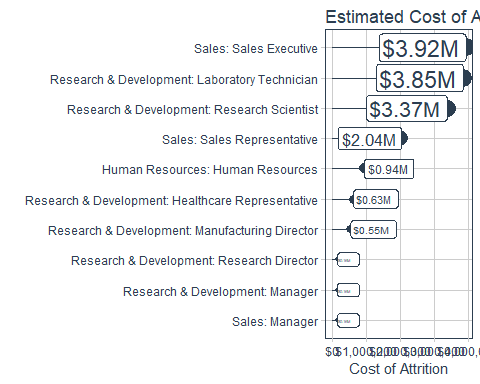


* LINE 1: geom\_segment adds lines to the plot canvas. The first ggplot statement locates the end of the line - at the cost\_of\_attrition. xend defines the start of the line segment at x = 0. yend make sure the line segment aligns with the correct y-axis label. Simple!
  + palette\_light()[[1]] defines the color [#2c3e5](http://www.color-hex.com/color/2c3e50) from the tidyquant package.
  + the other colors in palette\_light are: #2c3e50, #e31a1c, #18BC9C, #CCBE93, #a6cee3, #1f78b4, #b2df8a, #fb9a99, #fdbf6f, #ff7f00, #cab2d6, #6a3d9a
* LINE 2: size = cost\_of\_attrition adjusts the size of the segment endpoint to the attrition cost magnitude (could have included this in the first aes statement)
* LINE 3: Formatting the x-labels with the scales package to get the values in dollar format. Remember the scale\_x\_continuous ggplot function!
* LINE 4: geom\_label simply adds labels to points on the plot using cost\_text` that was created easier in the plot data manipulation.
  + Cool the label size can be dynamically set by the attrition cost.
* LINE 5: theme\_tq from tidyquant is a general theme that adjusts background, faceting palette, legend position and more. Works well for conservative business publications.
  + scale\_size adjusts the max and min size of elements to prevent large/small values from becoming too large/small. The label sizes are now limited to size = 3 making the lower values label text now readable. Cool!

## Making A Custom Plotting Function

The goal is to replace the code below - copied from the last plotting exercise:

dept\_job\_role\_tbl %>%   
   
 count(Department, JobRole, Attrition) %>% count\_to\_pct(Department, JobRole) %>%   
 assess\_attrition(Attrition, attrition\_value = "Yes", baseline\_pct = 0.088) %>%   
 mutate(cost\_of\_attrition = calculate\_attrition\_cost(n = n, salary = 80000)) %>%   
###<b>  
 #Data Manipulation   
 mutate(name = str\_c(Department, JobRole, sep = ": ") %>% as\_factor()) %>%  
 mutate(name = fct\_reorder(name, cost\_of\_attrition)) %>% # orders the names by cost  
 mutate(cost\_text = str\_c("$", format(cost\_of\_attrition / 1e6, digits = 2), "M", sep = "")) %>%   
   
 #Plotting  
 ggplot(aes(x = cost\_of\_attrition, y = name)) +  
 geom\_segment(aes(xend = 0, yend = name), color = palette\_light()[[1]]) +  
 geom\_point(aes(size = cost\_of\_attrition), color = palette\_light()[[1]]) +  
 scale\_x\_continuous(label = scales::dollar) +  
 geom\_label(aes(label = cost\_text, size = cost\_of\_attrition),   
 hjust = "inward", color = palette\_light()[[1]]) +   
 theme\_tq() + scale\_size(name = c(4,5)) +  
 labs(title = "Estimated Cost of Attrition: By Dept and Job Role",  
 y = "", x = "Cost of Attrition") + theme(legend.position = "none")



###</b>

Do no be intimidated - it is not as hard as it might appear initially!

## Developing plot\_attrition() {#plot\_attrition} {#tidy\_eval2}

First, an explanation of quosures and tidy evaluation:

There are four functions: - quo() and enquo(): singular, captures one variable, stored as unevaluated expression - quos() and enquos(): plural, captures multiple variables, stored as list of multiple unevaluated expressions

Explanation: enquo() is singular and used inside a function. It is comparable to quo() which is used outside of functions. Both functions do roughly the same thing, which is quote (meaning capturing the unevaluated variable as an expression before it gets updated). The “quoted” variable is stored for evaluation later. The main difference between quo() and enquo() is quo() should be used outside functions and enquo() should be used inside. The reason why is not important and really complex.

The rlang library now has an enquos() function that is plural (rlang did not always have enquos() which is why I usedquos()was used.. Both capture multiple variables as expressions contained inside a list. This is useful when you don't know how many arguments the user will supply. For example several columns. The difference betweenquos()andenquos()is, again, thatquos()is meant to be used external to functions whereasenquos()is meant to be used internal. However - This is not what the tutorial shows because of an inconsistency with early versions ofrlangthat did not include theenquos()` function.

rlang::sym turns a single character string into an expression. The expression is typically captured in enquo()orquos()to delay evaluation. Thesym()andsyms()functions are only necessary when converting a string to a symbol. Generally the user will supply a symbol, such as a column name. They don’t typically supply a string. Therefore, we only need to usesym/syms` in this special case where we have a string that represents one or more column names.

In summary: - Use quo() / enquo() when capturing one variable. - Use quos() / enquos() when capturing multiple variables. - Use quo() / quos() outside of functions. - Use enquo() / enquos() inside of functions.

plot\_attrition <- function(data, ..., .value, # ... capture the departments - mutate(name = str\_c(Department, JobRole....; .value - cost\_of\_attrition  
 fct\_reorder = TRUE, # order the function by the .value argument  
 fct\_rev = FALSE, # ascending vs descending y lables  
 include\_lbl = TRUE,  
 color = palette\_light()[[1]],  
 units = c("0", "K", "M")) {  
 #sometimes the untis will not be in millions, default is 0 - no units  
# Inputs  
  
 group\_vars\_expr <- quos(...) #perhaps department and JobRoles  
 if(length(group\_vars\_expr) == 0) # Then just use the first column name - these drive the y labels in the plot so there has to be something!  
 group\_vars\_expr <- quos(rlang::sym(colnames(data)[[1]]))  
   
 value\_expr <- enquo(.value)# for data manipulation (its cost\_of\_attrition - used in data manipulation and plotting)  
 value\_name <- quo\_name(value\_expr)# for ggplot - makes a character string in quotes (opposite of rlang::sym()); ggplot is not ready for tidy eval  
   
 units\_val <- switch(units[[1]], "M" = 1e6, "K" = 1e3, "0" = 1) #work like nested if functions  
 if (units[[1]] == "0") units <- "" #Make it empty because it would get appended to labels in the plot  
   
# Data Manipulation  
   
 #Function factory - a function that produces a function, usd is the function factory - try it usd(1000)  
 usd <- scales::dollar\_format(prefix = "$", largest\_with\_cents = 1e3)  
  
 data\_manipulated <- data %>%   
 mutate(name = str\_c(!!! group\_vars\_expr, sep = ": ") %>% as\_factor()) %>%   
 #Above () removed around !!! group\_vars\_expr otherwise str\_c sees it a single input - it is a psuedo bug in str\_c  
 mutate(value\_text = str\_c(usd(!! value\_expr / units\_val), units[[1]], sep = ""))  
   
 if(fct\_reorder){data\_manipulated <- data\_manipulated %>%   
 mutate(name = fct\_reorder(name, (!! value\_expr))) %>%   
 arrange(name)} # arrange() is used because fct\_reorder() does not actually sort the data, arrange() does the sorting  
   
 if(fct\_rev){data\_manipulated <- data\_manipulated %>%   
 mutate(name = fct\_rev(name)) %>%   
 arrange(name)}  
   
# Visualization  
# NOTE: ggplot does not work with tidyeval framework - requires use of aes\_string  
   
 g <- data\_manipulated %>%   
 ggplot(aes\_string(x = value\_name, y = "name")) +  
 geom\_segment(aes(xend = 0, yend = name), color = color) +  
 geom\_point(aes\_string(size = value\_name), color = color) +  
 scale\_x\_continuous(label = scales::dollar) +  
 theme\_tq() +  
 scale\_size(name = c(4,5)) + #c(3,5) should work but I am not satifies with the output.  
 theme(legend.position = "none")  
   
 if(include\_lbl){  
 g <- g + geom\_label(aes\_string(label = "value\_text", size = value\_name), hjust = "inward", color = color)  
 }  
 return(g)  
}

A potential problem with an error when trying to use this function on a grouped data set.

The warning thrown looks like this: mutate\_impl(.data, dots) : binding character and factor vector, coercing into character vector The solution is to add ungroup() before the factoring within the Data Manipulation step.

## More Realistic Attrition Cost

Using estimated average salary and average productivity (revenue) by Job Role and Department, create a more realistic attrition cost estimate.

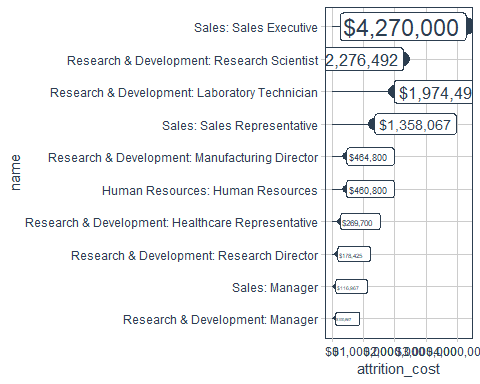
See homework\_1\_assignment.R for some relatively complex dplyr data code.

Here is a peek at the new Excel file that provides the source information.

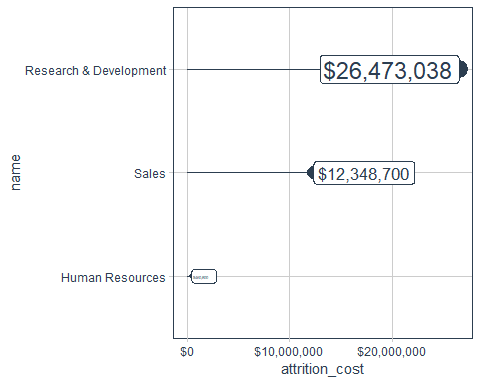
dept\_jobrole\_tbl <- train\_raw\_tbl %>%  
 select(EmployeeNumber, Department, JobRole, PerformanceRating, Attrition)  
kpi\_industry\_turnover\_pct <- 0.088  
# Productivity Cost by Role ----  
productivity\_cost\_by\_role\_tbl <- read\_excel("Challenges/productivity\_cost\_by\_role.xlsx")  
productivity\_cost\_by\_role\_tbl

## # A tibble: 11 x 4  
## Department JobRole Salary\_Average Revenue\_Average  
## <chr> <chr> <dbl> <dbl>  
## 1 Human Resources Human Resources 65000 104000   
## 2 Human Resources Manager 85000 153000   
## 3 Research & Develop~ Healthcare Represen~ 45000 76500   
## 4 Research & Develop~ Laboratory Technici~ 65000 110500   
## 5 Research & Develop~ Manager 88000 158400   
## 6 Research & Develop~ Manufacturing Direc~ 100000 220000.  
## 7 Research & Develop~ Research Director 125000 312500   
## 8 Research & Develop~ Research Scientist 95000 171000   
## 9 Sales Manager 95000 190000   
## 10 Sales Sales Executive 140000 308000   
## 11 Sales Sales Representative 80000 160000

dept\_jobrole\_productivty\_tbl <- dept\_jobrole\_tbl %>%   
 count(Department, JobRole, Attrition) %>%   
 count\_to\_pct(Department, JobRole) %>%   
  
 left\_join(productivity\_cost\_by\_role\_tbl, by = c("Department", "JobRole")) %>%   
   
 assess\_attrition(Attrition,attrition\_value = 'Yes',baseline\_pct = 0.088) %>%  
   
 mutate(attrition\_cost = calculate\_attrition\_cost(n = n, salary = Salary\_Average,  
 net\_revenue\_per\_employee = Revenue\_Average))  
  
dept\_jobrole\_productivty\_tbl %>% plot\_attrition(Department, JobRole, .value = attrition\_cost)

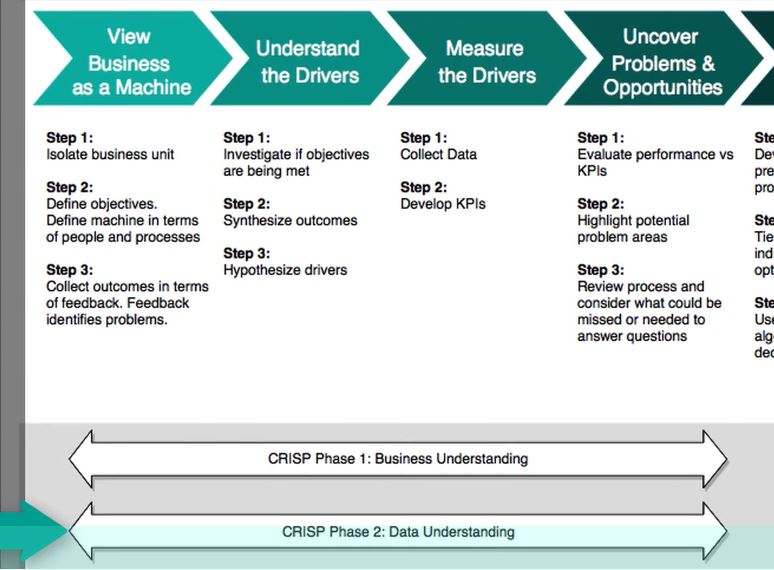


dept\_productivty\_tbl <- dept\_jobrole\_tbl %>%   
 count(Department, JobRole, Attrition) %>%   
 count\_to\_pct(Department, JobRole) %>%   
 assess\_attrition(Attrition, "Yes", kpi\_industry\_turnover\_pct) %>%   
   
 left\_join(productivity\_cost\_by\_role\_tbl, by = c("Department", "JobRole")) %>%   
   
 group\_by(Department) %>%   
 summarise(n = sum(n),  
 Salary\_Average = sum(Salary\_Average), Revenue\_Average = sum(Revenue\_Average)) %>%   
 mutate(attrition\_cost =   
 calculate\_attrition\_cost(n = n, salary = Salary\_Average,   
 net\_revenue\_per\_employee = Revenue\_Average))  
  
dept\_productivty\_tbl %>% plot\_attrition(Department, .value = attrition\_cost)



# Data Understanding

This is the start of the 2nd phase of CRISP-DM - Data Understanding.



1. Collect Data is complete - the Excel files have been provided.
2. EDA to explore the critical properties of the underlying features and potential relationships to attrition.

This section introduces skimr - a tool for feature exploration by data type. GGally is also introduced to visualize feature interaction relationships using ggpairs.

May want to review the feature grouping introduced earlier. [Group Features](#Group%20Features)

## EDA: Exploring Data By Data Type

### Character Data

skimr does provide a good overview of the data automatically separating character and numeric data. For character data, it is simple to see if any features have more than perhaps 9 unique features, it might suggest binning. The numeric data with histograms makes it easy to see which variable might be categorical data very quickly.

skim(train\_raw\_tbl)

## Skim summary statistics  
## n obs: 1250   
## n variables: 35   
##   
## -- Variable type:character ---------------------------------------------------------------------------------------------  
## variable missing complete n min max empty n\_unique  
## Attrition 0 1250 1250 2 3 0 2  
## BusinessTravel 0 1250 1250 10 17 0 3  
## Department 0 1250 1250 5 22 0 3  
## EducationField 0 1250 1250 5 16 0 6  
## Gender 0 1250 1250 4 6 0 2  
## JobRole 0 1250 1250 7 25 0 9  
## MaritalStatus 0 1250 1250 6 8 0 3  
## Over18 0 1250 1250 1 1 0 1  
## OverTime 0 1250 1250 2 3 0 2  
##   
## -- Variable type:numeric -----------------------------------------------------------------------------------------------  
## variable missing complete n mean sd p0  
## Age 0 1250 1250 36.95 9.08 18  
## DailyRate 0 1250 1250 799.74 403.52 102  
## DistanceFromHome 0 1250 1250 9.1 8.1 1  
## Education 0 1250 1250 2.91 1.03 1  
## EmployeeCount 0 1250 1250 1 0 1  
## EmployeeNumber 0 1250 1250 1022.71 600.42 1  
## EnvironmentSatisfaction 0 1250 1250 2.71 1.09 1  
## HourlyRate 0 1250 1250 65.82 20.14 30  
## JobInvolvement 0 1250 1250 2.75 0.71 1  
## JobLevel 0 1250 1250 2.08 1.12 1  
## JobSatisfaction 0 1250 1250 2.73 1.11 1  
## MonthlyIncome 0 1250 1250 6569.4 4777.89 1009  
## MonthlyRate 0 1250 1250 14294.5 7071.61 2097  
## NumCompaniesWorked 0 1250 1250 2.71 2.49 0  
## PercentSalaryHike 0 1250 1250 15.23 3.69 11  
## PerformanceRating 0 1250 1250 3.16 0.36 3  
## RelationshipSatisfaction 0 1250 1250 2.71 1.09 1  
## StandardHours 0 1250 1250 80 0 80  
## StockOptionLevel 0 1250 1250 0.8 0.86 0  
## TotalWorkingYears 0 1250 1250 11.4 7.79 0  
## TrainingTimesLastYear 0 1250 1250 2.81 1.29 0  
## WorkLifeBalance 0 1250 1250 2.76 0.71 1  
## YearsAtCompany 0 1250 1250 7.09 6.21 0  
## YearsInCurrentRole 0 1250 1250 4.26 3.66 0  
## YearsSinceLastPromotion 0 1250 1250 2.21 3.2 0  
## YearsWithCurrManager 0 1250 1250 4.16 3.59 0  
## p25 p50 p75 p100 hist  
## 30 36 43 60 <U+2582><U+2583><U+2586><U+2587><U+2585><U+2583><U+2582><U+2582>  
## 464 798 1156 1498 <U+2587><U+2587><U+2587><U+2587><U+2587><U+2587><U+2587><U+2587>  
## 2 7 13.75 29 <U+2587><U+2585><U+2583><U+2581><U+2581><U+2581><U+2581><U+2581>  
## 2 3 4 5 <U+2582><U+2585><U+2581><U+2587><U+2581><U+2586><U+2581><U+2581>  
## 1 1 1 1 <U+2581><U+2581><U+2581><U+2587><U+2581><U+2581><U+2581><U+2581>  
## 486.25 1020.5 1553.75 2068 <U+2587><U+2587><U+2587><U+2587><U+2587><U+2587><U+2587><U+2587>  
## 2 3 4 4 <U+2585><U+2581><U+2585><U+2581><U+2581><U+2587><U+2581><U+2587>  
## 48 66 83 100 <U+2586><U+2587><U+2587><U+2587><U+2586><U+2587><U+2587><U+2587>  
## 2 3 3 4 <U+2581><U+2581><U+2583><U+2581><U+2581><U+2587><U+2581><U+2582>  
## 1 2 3 5 <U+2587><U+2587><U+2581><U+2583><U+2581><U+2582><U+2581><U+2581>  
## 2 3 4 4 <U+2585><U+2581><U+2585><U+2581><U+2581><U+2587><U+2581><U+2587>  
## 2935.25 4903.5 8395 19999 <U+2587><U+2587><U+2583><U+2582><U+2581><U+2581><U+2581><U+2582>  
## 8191.25 14328 20337.25 26999 <U+2587><U+2587><U+2587><U+2587><U+2587><U+2587><U+2587><U+2586>  
## 1 2 4 9 <U+2587><U+2582><U+2582><U+2582><U+2581><U+2581><U+2581><U+2581>  
## 12 14 18 25 <U+2587><U+2587><U+2583><U+2583><U+2582><U+2582><U+2582><U+2581>  
## 3 3 3 4 <U+2587><U+2581><U+2581><U+2581><U+2581><U+2581><U+2581><U+2582>  
## 2 3 4 4 <U+2585><U+2581><U+2586><U+2581><U+2581><U+2587><U+2581><U+2587>  
## 80 80 80 80 <U+2581><U+2581><U+2581><U+2587><U+2581><U+2581><U+2581><U+2581>  
## 0 1 1 3 <U+2587><U+2581><U+2587><U+2581><U+2581><U+2582><U+2581><U+2581>  
## 6 10 15 40 <U+2583><U+2587><U+2582><U+2582><U+2582><U+2581><U+2581><U+2581>  
## 2 3 3 6 <U+2581><U+2581><U+2587><U+2587><U+2581><U+2582><U+2582><U+2581>  
## 2 3 3 4 <U+2581><U+2581><U+2583><U+2581><U+2581><U+2587><U+2581><U+2582>  
## 3 5 10 40 <U+2587><U+2585><U+2581><U+2581><U+2581><U+2581><U+2581><U+2581>  
## 2 3 7 18 <U+2587><U+2583><U+2581><U+2585><U+2581><U+2581><U+2581><U+2581>  
## 0 1 3 15 <U+2587><U+2582><U+2581><U+2581><U+2581><U+2581><U+2581><U+2581>  
## 2 3 7 17 <U+2587><U+2583><U+2581><U+2583><U+2581><U+2581><U+2581><U+2581>

### Character Data

train\_raw\_tbl %>% select\_if(is.character) %>% glimpse()

## Observations: 1,250  
## Variables: 9  
## $ Attrition <chr> "Yes", "No", "Yes", "No", "No", "No", "No", "No...  
## $ BusinessTravel <chr> "Travel\_Rarely", "Travel\_Frequently", "Travel\_R...  
## $ Department <chr> "Sales", "Research & Development", "Research & ...  
## $ EducationField <chr> "Life Sciences", "Life Sciences", "Other", "Lif...  
## $ Gender <chr> "Female", "Male", "Male", "Female", "Male", "Ma...  
## $ JobRole <chr> "Sales Executive", "Research Scientist", "Labor...  
## $ MaritalStatus <chr> "Single", "Married", "Single", "Married", "Marr...  
## $ Over18 <chr> "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y...  
## $ OverTime <chr> "Yes", "No", "Yes", "Yes", "No", "No", "Yes", "...

Character data does not have levels so use unique instead. map from purrr maps unique across each feature.

train\_raw\_tbl %>% select\_if(is.character) %>% map(unique) %>% head()

## $Attrition  
## [1] "Yes" "No"   
##   
## $BusinessTravel  
## [1] "Travel\_Rarely" "Travel\_Frequently" "Non-Travel"   
##   
## $Department  
## [1] "Sales" "Research & Development"  
## [3] "Human Resources"   
##   
## $EducationField  
## [1] "Life Sciences" "Other" "Medical"   
## [4] "Marketing" "Technical Degree" "Human Resources"   
##   
## $Gender  
## [1] "Female" "Male"   
##   
## $JobRole  
## [1] "Sales Executive" "Research Scientist"   
## [3] "Laboratory Technician" "Manufacturing Director"   
## [5] "Healthcare Representative" "Manager"   
## [7] "Sales Representative" "Research Director"   
## [9] "Human Resources"

This improves upon unique by including the counts for each character feature.

train\_raw\_tbl %>% select\_if(is.character) %>% map(table) %>% head()

## $Attrition  
##   
## No Yes   
## 1049 201   
##   
## $BusinessTravel  
##   
## Non-Travel Travel\_Frequently Travel\_Rarely   
## 124 239 887   
##   
## $Department  
##   
## Human Resources Research & Development Sales   
## 49 832 369   
##   
## $EducationField  
##   
## Human Resources Life Sciences Marketing Medical   
## 23 520 135 392   
## Other Technical Degree   
## 66 114   
##   
## $Gender  
##   
## Female Male   
## 491 759   
##   
## $JobRole  
##   
## Healthcare Representative Human Resources   
## 105 39   
## Laboratory Technician Manager   
## 224 89   
## Manufacturing Director Research Director   
## 123 73   
## Research Scientist Sales Executive   
## 259 273   
## Sales Representative   
## 65

Counts are nice but proportions are better! Below, anonymous function is used. {#anonymous}

The ~ indicates a formula starts

Also understand that train\_raw\_tbl %>% select\_if(is.character) %>% map(~ table(.)) is the sames as train\_raw\_tbl %>% select\_if(is.character) %>% map(table).

# Anonymous function with ~   
train\_raw\_tbl %>% select\_if(is.character) %>% map(~ table(.) %>% prop.table())

## $Attrition  
## .  
## No Yes   
## 0.84 0.16   
##   
## $BusinessTravel  
## .  
## Non-Travel Travel\_Frequently Travel\_Rarely   
## 0.099 0.191 0.710   
##   
## $Department  
## .  
## Human Resources Research & Development Sales   
## 0.039 0.666 0.295   
##   
## $EducationField  
## .  
## Human Resources Life Sciences Marketing Medical   
## 0.018 0.416 0.108 0.314   
## Other Technical Degree   
## 0.053 0.091   
##   
## $Gender  
## .  
## Female Male   
## 0.39 0.61   
##   
## $JobRole  
## .  
## Healthcare Representative Human Resources   
## 0.084 0.031   
## Laboratory Technician Manager   
## 0.179 0.071   
## Manufacturing Director Research Director   
## 0.098 0.058   
## Research Scientist Sales Executive   
## 0.207 0.218   
## Sales Representative   
## 0.052   
##   
## $MaritalStatus  
## .  
## Divorced Married Single   
## 0.22 0.46 0.32   
##   
## $Over18  
## .  
## Y   
## 1   
##   
## $OverTime  
## .  
## No Yes   
## 0.72 0.28

### Numeric Data

map\_df provides a much cleaner output! Outputs a dataframe rather than a list.

train\_raw\_tbl %>% select\_if(is.numeric) %>% map\_df(~ unique(.) %>% length())

## # A tibble: 1 x 26  
## Age DailyRate DistanceFromHome Education EmployeeCount EmployeeNumber  
## <int> <int> <int> <int> <int> <int>  
## 1 43 808 29 5 1 1250  
## # ... with 20 more variables: EnvironmentSatisfaction <int>,  
## # HourlyRate <int>, JobInvolvement <int>, JobLevel <int>,  
## # JobSatisfaction <int>, MonthlyIncome <int>, MonthlyRate <int>,  
## # NumCompaniesWorked <int>, PercentSalaryHike <int>,  
## # PerformanceRating <int>, RelationshipSatisfaction <int>,  
## # StandardHours <int>, StockOptionLevel <int>, TotalWorkingYears <int>,  
## # TrainingTimesLastYear <int>, WorkLifeBalance <int>,  
## # YearsAtCompany <int>, YearsInCurrentRole <int>,  
## # YearsSinceLastPromotion <int>, YearsWithCurrManager <int>

{#gather\_example}

train\_raw\_tbl %>% select\_if(is.numeric) %>% map\_df(~ unique(.) %>% length()) %>%   
 gather() %>% arrange(value) %>% filter(value <= 10) # features with more than 10 might be nonessential or categorical

## # A tibble: 13 x 2  
## key value  
## <chr> <int>  
## 1 EmployeeCount 1  
## 2 StandardHours 1  
## 3 PerformanceRating 2  
## 4 EnvironmentSatisfaction 4  
## 5 JobInvolvement 4  
## 6 JobSatisfaction 4  
## 7 RelationshipSatisfaction 4  
## 8 StockOptionLevel 4  
## 9 WorkLifeBalance 4  
## 10 Education 5  
## 11 JobLevel 5  
## 12 TrainingTimesLastYear 7  
## 13 NumCompaniesWorked 10

Numeric features that have fewer levels are likely to be discrete and numeric variables with more are likely continuous. (Note above the features where value = 1 may indicate non-essential or zero-variance variables.) Above are features that might be converted to factors. The table below suggests the features remain continuous.

train\_raw\_tbl %>% select\_if(is.numeric) %>% map\_df(~ unique(.) %>% length()) %>%   
 gather() %>% arrange(value) %>% filter(value > 10) # features with more than 10 might be nonessential or categorical

## # A tibble: 13 x 2  
## key value  
## <chr> <int>  
## 1 PercentSalaryHike 15  
## 2 YearsSinceLastPromotion 16  
## 3 YearsWithCurrManager 18  
## 4 YearsInCurrentRole 19  
## 5 DistanceFromHome 29  
## 6 YearsAtCompany 37  
## 7 TotalWorkingYears 40  
## 8 Age 43  
## 9 HourlyRate 71  
## 10 DailyRate 808  
## 11 MonthlyIncome 1161  
## 12 MonthlyRate 1217  
## 13 EmployeeNumber 1250

### Visualizing The Feature-Target Interactions

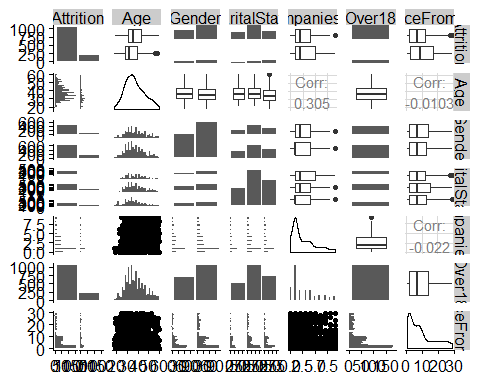
To begin the visualization of the data, select the descriptive variables (serves as an example). There is a mix of character and continuous features.

train\_raw\_tbl %>% select(Attrition, Age, Gender, MaritalStatus, NumCompaniesWorked, Over18, DistanceFromHome)

## # A tibble: 1,250 x 7  
## Attrition Age Gender MaritalStatus NumCompaniesWor~ Over18  
## <chr> <dbl> <chr> <chr> <dbl> <chr>   
## 1 Yes 41 Female Single 8 Y   
## 2 No 49 Male Married 1 Y   
## 3 Yes 37 Male Single 6 Y   
## 4 No 33 Female Married 1 Y   
## 5 No 27 Male Married 9 Y   
## 6 No 32 Male Single 0 Y   
## 7 No 59 Female Married 4 Y   
## 8 No 30 Male Divorced 1 Y   
## 9 No 38 Male Single 0 Y   
## 10 No 36 Male Married 6 Y   
## # ... with 1,240 more rows, and 1 more variable: DistanceFromHome <dbl>

Apply ggpair function:

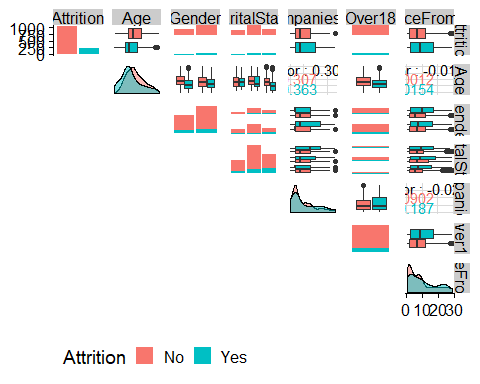
train\_raw\_tbl %>% select(Attrition, Age, Gender, MaritalStatus, NumCompaniesWorked, Over18, DistanceFromHome) %>% ggpairs()



[Customize](http://ggobi.github.io/ggally/#matrix_sections) the output to make it a bit easier to interpret.

* It colors the Attrition to make it stand out
* Removes the lower triangle - it is not quite as helpful or understandable
* customizes what is shown on the diagonal and changes alpha

train\_raw\_tbl %>% select(Attrition, Age, Gender, MaritalStatus, NumCompaniesWorked, Over18, DistanceFromHome) %>%   
 ggpairs(aes(color = Attrition), lower = "blank", legend = 1, diag = list(continuous = wrap("densityDiag", alpha = 0.5))) +  
 theme(legend.position = "bottom")



#ggsave("./images/plot\_ggpairs.jpg")

### Custom Function: plot\_ggpairs

Take the visualization above and create a function.

* color= NULL will output normal grey. However, best to select a feature to highlight (typically the label)
  + rlang::quo\_is\_null simply tests for if color\_expr == NULL. Test by simply typing color\_expr in the console
* Because color\_name is a string after using quo\_name, aes\_string must be used

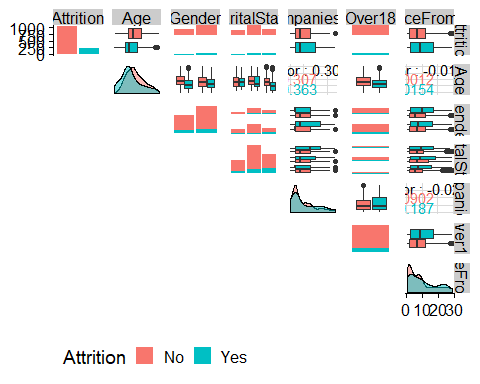
plot\_ggpairs <- function(data, color = NULL, density\_alpha = 0.5){  
 color\_expr <- enquo(color)  
 if(rlang::quo\_is\_null(color\_expr)){gdata %>% ggpairs(lowe = "blank")  
 } else {  
 color\_name <- quo\_name(color\_expr)  
 g <- data %>%  
 ggpairs(mapping = aes\_string(color = color\_name),  
 lower = "blank", legend = 1, diag = list(continuous = wrap("densityDiag",   
 alpha = density\_alpha))) + theme(legend.position = "bottom")}  
 return(g)}

### Evaluate ggpairs Output

Evaluate the data features by category and understand the importance of the features using the new ggpairs function. How great is that?

1. Descriptive Features: age, gender, marital status, etc

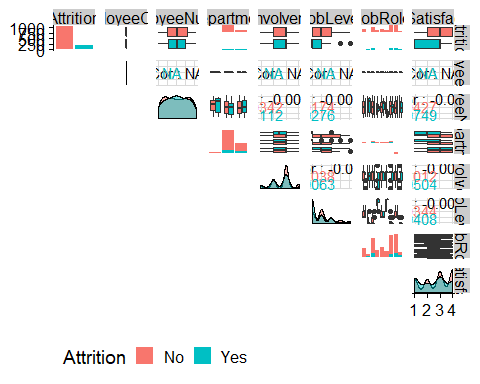
train\_raw\_tbl %>% select(Attrition, Age, Gender, MaritalStatus, NumCompaniesWorked, Over18, DistanceFromHome) %>% plot\_ggpairs(Attrition)

 **Observations**

* AgeYounger people seem to leave more than older workers.
* DistanceFromHome appears skewed - further away leave more frequently
* Num Companies Worked appears more skewed for workers that have had more jobs
* Marital Status interesting: No goes small large medium where Yes goes small medium large (height of histogram bars)

1. Employment Features

train\_raw\_tbl %>% select(Attrition, contains("employee"), contains("department"), contains("job")) %>% plot\_ggpairs(Attrition)



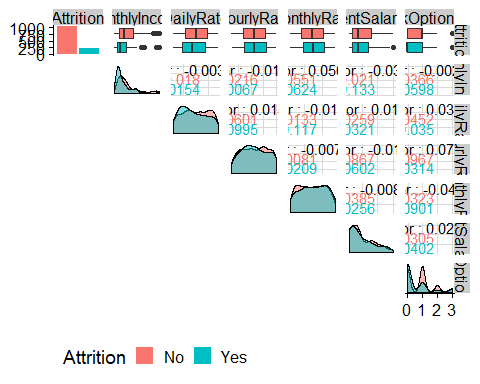
Contains is NOT case sensitive. Also provides end\_with option

**Observations**

* EmployeeCount and EmployeeNumber are not useful - will not be used in model
* Department: Higher proportion evident in Dept-Dept histogram in far right histogram bar
  + Recall there only only 3 departments in this data (HR, Research and Sales)
* JobInvolvement: Density Plot shows higher 3rd spike and 4th to less extent - more likely to stay
* JobLevel: Density plot shows deviation
* JobRoles - some roles appear to have higher attrition rate by evaluating the histogram
* JobStatisfaction: 1 - higher density of attrition than others

1. Compensation Features

train\_raw\_tbl %>% select(Attrition, contains("income"), contains("rate"), contains("salary"),   
 contains("stock")) %>% plot\_ggpairs(Attrition)

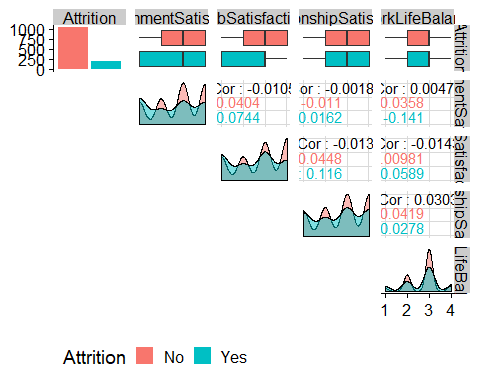


**Observations**

* MonthlyRate: Those that are leaving have a lower Monthly Income
* PercentSalaryHike: It’s difficult to deduce anything based on the visualization
* StockOptionLevel: Those that are staying have a higher stock option level

1. Survey Results

train\_raw\_tbl %>% select(Attrition, contains("satisfaction"), contains("life")) %>% plot\_ggpairs(Attrition)

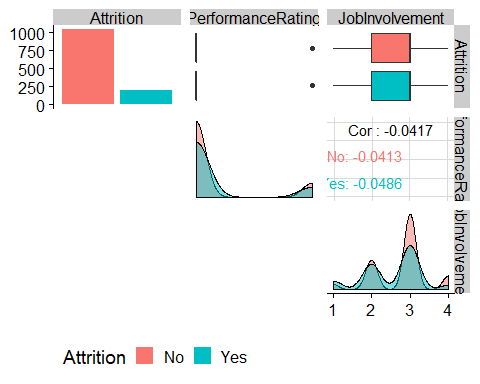


**Observations**

* EnvironmentSatisfaction: A higher proportion of those leaving have a low environment satisfaction level
* WorkLifeBalance: Those that are staying have a higher density of 2’s and 3’s

1. Performance Data

train\_raw\_tbl %>% select(Attrition, contains("performance"), contains("involvement")) %>% plot\_ggpairs(Attrition)



* Observations:
* JobInvolvement: Those that are leaving have a lower density of 3’s and 4’s

1. Work-Life Features

train\_raw\_tbl %>% select(Attrition, contains("overtime"), contains("travel")) %>% plot\_ggpairs(Attrition)

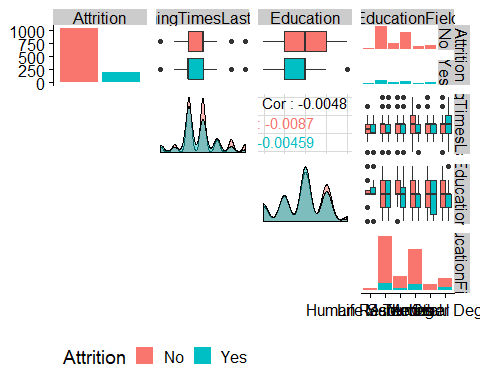


**Observations**

* Overtime: The proportion of those leaving that are working Over Time are high compared to those that are not leaving

1. Training and Education

train\_raw\_tbl %>% select(Attrition, contains("training"), contains("education")) %>% plot\_ggpairs(Attrition)

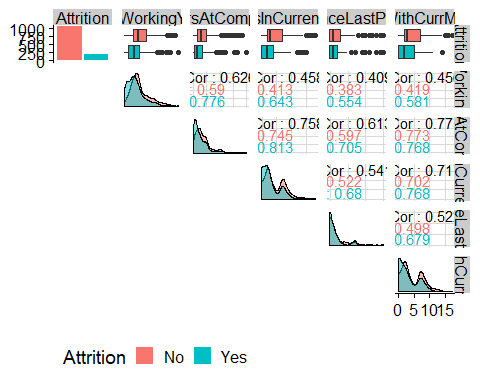


**Observations**

* TrainingTimesLastYear: People that leave tend to have less annual training

1. Time-Based Features

train\_raw\_tbl %>% select(Attrition, contains("years")) %>% plot\_ggpairs(Attrition)



**Observations**

* YearsAtCompany: People that leave tend to have less working years at the company
* YearsSinceLastPromotion: It’s difficult to deduce anything based on the visualization

# Data Preparation

There are to goals for this section:

1. How does the data have to be formatted for people to visualize and understand?

* Prepare a preprocessing data pipeline for people.

1. How does the data have to be formatted for machine learning algorithms to process and for other types of analysis like correlation analysis?

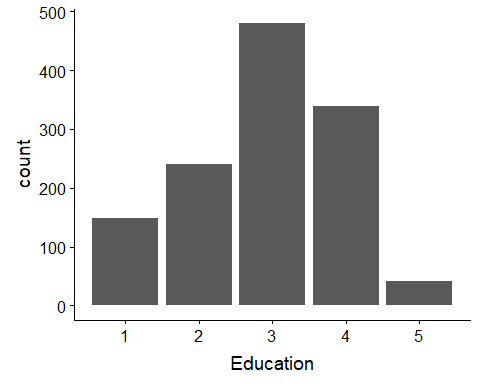
* Use recipies to build machine learning preprocessing templates.

## Data Preparation For People

Must merge the telco-train.xlsx and telco\_definitions.xlsx files to present data in a meaningful way and that is the purpose of the processing pipeline is.

Look at an example:

train\_raw\_tbl %>% ggplot(aes(Education)) + geom\_bar()



The plot above is not very readable.

* The Education numbers on the x-axis do not mean anything to most people.

The human readable pipeline will correct this and other pieces to make the plots more user-friendly.

### Tidying The Data

Explore the data definitions file - note that many of the features are coded - numbers used as values for descriptions for categorical data.

path\_data\_definitions <- "00\_Data/telco\_data\_definitions.xlsx"  
definitions\_raw\_tbl <- read\_excel(path\_data\_definitions, sheet = 1, col\_names = FALSE)  
definitions\_raw\_tbl

## # A tibble: 35 x 2  
## X\_\_1 X\_\_2   
## <chr> <chr>   
## 1 Education 1 'Below College'  
## 2 <NA> 2 'College'   
## 3 <NA> 3 'Bachelor'   
## 4 <NA> 4 'Master'   
## 5 <NA> 5 'Doctor'   
## 6 <NA> <NA>   
## 7 EnvironmentSatisfaction 1 'Low'   
## 8 <NA> 2 'Medium'   
## 9 <NA> 3 'High'   
## 10 <NA> 4 'Very High'   
## # ... with 25 more rows

X\_\_1 = Feature Name; x\_\_2 = Feature Code + Feature Description

Tidying this data will allow us to replace numbers with something more meaningful using tidyverse functions.

* fill replaces missing values (NAs) with the closet entry (previous if .direction = "down" or next if .direction = "up")
  + For example, this will fill NAs in Education under X\_\_1 with Education - very cool!
* Remove NAs under X\_\_2
* Break X\_\_2 into 2 columns: one with the number, the other with the description
  + separate is from tidyr
  + remove simply removes the original X\_\_2 column
  + **Need to start using this!**
* Rename columns to make them more friendly - X\_\_1 does not mean anything!
* Remove the trailing tick mark from the new value column
  + str\_replace replaces a matched pattern in a string with a replacement argument

definitions\_tbl <- definitions\_raw\_tbl %>% fill(X\_\_1, .direction = "down") %>% filter(!is.na(X\_\_2)) %>%   
 separate(X\_\_2, into = c("key", "value"), sep = " '", remove = TRUE) %>%   
 rename(column\_name = X\_\_1) %>% mutate(key = as.numeric(key)) %>%   
 mutate(value = value %>% str\_replace(pattern = "'", replacement = ""))  
definitions\_tbl

## # A tibble: 29 x 3  
## column\_name key value   
## <chr> <dbl> <chr>   
## 1 Education 1 Below College  
## 2 Education 2 College   
## 3 Education 3 Bachelor   
## 4 Education 4 Master   
## 5 Education 5 Doctor   
## 6 EnvironmentSatisfaction 1 Low   
## 7 EnvironmentSatisfaction 2 Medium   
## 8 EnvironmentSatisfaction 3 High   
## 9 EnvironmentSatisfaction 4 Very High   
## 10 JobInvolvement 1 Low   
## # ... with 19 more rows

### Mapping Over Lists

definitions\_tbl has multiple datasets in one dataframe. They will need to be integrated separately into the training and testing datasets. This will make more sense in a bit. This process starts by making definitions\_tbl into a list.

This concept is new - read and study carefully!

* split splits a dataframe into multiple dataframes contained within a list. Supply a column name as a vector (meaning .$column\_name). Therefore, Education, EnvirnmentSatisfaction and JobInvolvement will be separated into individual dataframes. Example:

definitions\_list[[3]]  
# A tibble: 4 x 3  
 column\_name key value   
 <chr> <dbl> <chr>   
1 JobInvolvement 1 Low   
2 JobInvolvement 2 Medium   
3 JobInvolvement 3 High   
4 JobInvolvement 4 Very High

* Use map and an anonymous function (the ~) to remove `column\_name from each of the dataframes stored in the newly created list
  + Example:
* > definitions\_list[[3]]  
  # A tibble: 4 x 2  
   key value   
  <dbl> <chr>   
  1 1 Low   
  2 2 Medium   
  3 3 High   
  4 4 Very High
* Use map again with an anonymous function to each entry in the column value into a factor

definitions\_list <- definitions\_tbl %>%   
 split(.$column\_name) %>% #split df into multiple dfs within a list  
 map(~ select(., -column\_name)) %>% # remove column\_name col leaving key and value  
 map(~ mutate(., value = as\_factor(value)))# Creates factors in the order they appear - not alphabetically like as.factor  
definitions\_list[[3]]

## # A tibble: 4 x 2  
## key value   
## <dbl> <fct>   
## 1 1 Low   
## 2 2 Medium   
## 3 3 High   
## 4 4 Very High

IMPORTANT: **forcats::as\_factor** is different from base::as.factor. The former creates factor in the order in which they appear as opposed to alphabetically with base::as.factor

* Rename the names from key and value to something like Education and Education\_value
  + first get the name of the dataframe in the list and assign it the variable list\_name
  + Then set the column names using list\_name and making the value column into list\_name pasted with \_value - really not that hard!

for(i in seq\_along(definitions\_list)){  
 list\_name <- names(definitions\_list)[i]  
 colnames(definitions\_list[[i]]) <- c(list\_name, paste0(list\_name, "\_value"))}  
definitions\_list[[3]]

## # A tibble: 4 x 2  
## JobInvolvement JobInvolvement\_value  
## <dbl> <fct>   
## 1 1 Low   
## 2 2 Medium   
## 3 3 High   
## 4 4 Very High

This can now be merged with the training dataset!

Iteratively join the dataframes within the definitions list with the main dataframe (training data)

This is new - study and learn!

* Make the train\_raw\_tbl at list and call it HRData - weird, huh!
  + make a list but the original training DF is the only entry called HRData
* append adds elements to a vector or **list**
  + The list grows because the first element is the training DF followed by all the elements in the definitions list
  + everything is in one list - **helpful to use lists to collect objects that need to be iterated over with purrr**
  + recall we are doing this so values are replaced by descriptions in the training data
* Use reduce which iteratively applies a user specified function to successive binary sets of objects. A 3-element vector would have a function applied to the first 2 elements and that output would then have the function applied with the 3rd element
  + here it adds columns like EducationField joined on the Education numerical value - human readable!
* Remove the original training data field names, like Education since we have just joined the more descriptive fields like Education\_value
  + Using negative one\_of to remove them
* Rename the remaining fields like Education\_value to a more useful nomenclature
  + using set\_names from base to do this leverage str\_replace\_all to remove \_value
* Resort the dataframe

data\_merged\_tbl <- list(HRData = train\_raw\_tbl) %>% append(definitions\_list, after = 1) %>%   
 reduce(left\_join) %>%   
 select(-one\_of(names(definitions\_list))) %>%   
 set\_names(str\_replace\_all(names(.), pattern = "\_value", replacement = "")) %>%   
 select(sort(names(.)))  
glimpse(data\_merged\_tbl)

## Observations: 1,250  
## Variables: 35  
## $ Age <dbl> 41, 49, 37, 33, 27, 32, 59, 30, 38, 3...  
## $ Attrition <chr> "Yes", "No", "Yes", "No", "No", "No",...  
## $ BusinessTravel <chr> "Travel\_Rarely", "Travel\_Frequently",...  
## $ DailyRate <dbl> 1102, 279, 1373, 1392, 591, 1005, 132...  
## $ Department <chr> "Sales", "Research & Development", "R...  
## $ DistanceFromHome <dbl> 1, 8, 2, 3, 2, 2, 3, 24, 23, 27, 16, ...  
## $ Education <fct> College, Below College, College, Mast...  
## $ EducationField <chr> "Life Sciences", "Life Sciences", "Ot...  
## $ EmployeeCount <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...  
## $ EmployeeNumber <dbl> 1, 2, 4, 5, 7, 8, 10, 11, 12, 13, 14,...  
## $ EnvironmentSatisfaction <fct> Medium, High, Very High, Very High, L...  
## $ Gender <chr> "Female", "Male", "Male", "Female", "...  
## $ HourlyRate <dbl> 94, 61, 92, 56, 40, 79, 81, 67, 44, 9...  
## $ JobInvolvement <fct> High, Medium, Medium, High, High, Hig...  
## $ JobLevel <dbl> 2, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 2, 1...  
## $ JobRole <chr> "Sales Executive", "Research Scientis...  
## $ JobSatisfaction <fct> Very High, Medium, High, High, Medium...  
## $ MaritalStatus <chr> "Single", "Married", "Single", "Marri...  
## $ MonthlyIncome <dbl> 5993, 5130, 2090, 2909, 3468, 3068, 2...  
## $ MonthlyRate <dbl> 19479, 24907, 2396, 23159, 16632, 118...  
## $ NumCompaniesWorked <dbl> 8, 1, 6, 1, 9, 0, 4, 1, 0, 6, 0, 0, 1...  
## $ Over18 <chr> "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y...  
## $ OverTime <chr> "Yes", "No", "Yes", "Yes", "No", "No"...  
## $ PercentSalaryHike <dbl> 11, 23, 15, 11, 12, 13, 20, 22, 21, 1...  
## $ PerformanceRating <fct> Excellent, Outstanding, Excellent, Ex...  
## $ RelationshipSatisfaction <fct> Low, Very High, Medium, High, Very Hi...  
## $ StandardHours <dbl> 80, 80, 80, 80, 80, 80, 80, 80, 80, 8...  
## $ StockOptionLevel <dbl> 0, 1, 0, 0, 1, 0, 3, 1, 0, 2, 1, 0, 1...  
## $ TotalWorkingYears <dbl> 8, 10, 7, 8, 6, 8, 12, 1, 10, 17, 6, ...  
## $ TrainingTimesLastYear <dbl> 0, 3, 3, 3, 3, 2, 3, 2, 2, 3, 5, 3, 1...  
## $ WorkLifeBalance <fct> Bad, Better, Better, Better, Better, ...  
## $ YearsAtCompany <dbl> 6, 10, 0, 8, 2, 7, 1, 1, 9, 7, 5, 9, ...  
## $ YearsInCurrentRole <dbl> 4, 7, 0, 7, 2, 7, 0, 0, 7, 7, 4, 5, 2...  
## $ YearsSinceLastPromotion <dbl> 0, 1, 0, 3, 2, 3, 0, 0, 1, 7, 0, 0, 4...  
## $ YearsWithCurrManager <dbl> 5, 7, 0, 0, 2, 6, 0, 0, 8, 7, 3, 8, 3...

### Factoring Character Data

There are still some character variables:

data\_merged\_tbl %>% select\_if(is.character) %>% glimpse()

## Observations: 1,250  
## Variables: 9  
## $ Attrition <chr> "Yes", "No", "Yes", "No", "No", "No", "No", "No...  
## $ BusinessTravel <chr> "Travel\_Rarely", "Travel\_Frequently", "Travel\_R...  
## $ Department <chr> "Sales", "Research & Development", "Research & ...  
## $ EducationField <chr> "Life Sciences", "Life Sciences", "Other", "Lif...  
## $ Gender <chr> "Female", "Male", "Male", "Female", "Male", "Ma...  
## $ JobRole <chr> "Sales Executive", "Research Scientist", "Labor...  
## $ MaritalStatus <chr> "Single", "Married", "Single", "Married", "Marr...  
## $ Over18 <chr> "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y...  
## $ OverTime <chr> "Yes", "No", "Yes", "Yes", "No", "No", "Yes", "...

Need to transformed into factors. Evaluate if the features are ordered.

data\_merged\_tbl %>% distinct(BusinessTravel)

## # A tibble: 3 x 1  
## BusinessTravel   
## <chr>   
## 1 Travel\_Rarely   
## 2 Travel\_Frequently  
## 3 Non-Travel

The natural order should be: Non\_Travel –> Travel\_Rarely –> Travel\_Frequently. Change it!

Evaluate the levels of the factors:

# Reorder some factors  
data\_merged\_tbl %>% mutate\_if(is.character, as.factor) %>% select\_if(is.factor) %>% map(levels) %>% head()

## $Attrition  
## [1] "No" "Yes"  
##   
## $BusinessTravel  
## [1] "Non-Travel" "Travel\_Frequently" "Travel\_Rarely"   
##   
## $Department  
## [1] "Human Resources" "Research & Development"  
## [3] "Sales"   
##   
## $Education  
## [1] "Below College" "College" "Bachelor" "Master"   
## [5] "Doctor"   
##   
## $EducationField  
## [1] "Human Resources" "Life Sciences" "Marketing"   
## [4] "Medical" "Other" "Technical Degree"  
##   
## $EnvironmentSatisfaction  
## [1] "Low" "Medium" "High" "Very High"

BusinessTravel and MartitalStatus are the only factor that need to be re-leveled.

data\_processed\_tbl <- data\_merged\_tbl %>% mutate\_if(is.character, as.factor) %>%  
 mutate(BusinessTravel = BusinessTravel %>% fct\_relevel("Non-Travel", "Travel\_Rarely", "Travel\_Frequently"),  
 MaritalStatus = MaritalStatus %>% fct\_relevel("Single", "Married", "Divorced"))  
  
data\_processed\_tbl %>% select(BusinessTravel, MaritalStatus) %>% map(levels)

## $BusinessTravel  
## [1] "Non-Travel" "Travel\_Rarely" "Travel\_Frequently"  
##   
## $MaritalStatus  
## [1] "Single" "Married" "Divorced"

## Making The Processing Pipeline

The pipeline below assumes the source files meet specific formats. It is unlikely that other company data will be different. Modify the code below as needed to fit your specific file format needs.

process\_hr\_data\_readable <- function(data, definitions\_tbl){  
   
 definitions\_list <- definitions\_tbl %>% fill(X\_\_1, .direction = "down") %>%   
 filter(!is.na(X\_\_2)) %>%   
 separate(X\_\_2, into = c("key", "value"), sep = " '", remove = TRUE) %>%   
 rename(column\_name = X\_\_1) %>%   
 mutate(key = as.numeric(key)) %>%   
 mutate(value = value %>% str\_replace(pattern = "'", replacement = "")) %>%   
 split(.$column\_name) %>%   
 map(~ select(., -column\_name)) %>%   
 map(~ mutate(., value = as\_factor(value)))  
   
 for(i in seq\_along(definitions\_list)){  
 list\_name <- names(definitions\_list)[i]  
 colnames(definitions\_list[[i]]) <- c(list\_name, paste0(list\_name, "\_value")) }  
   
 data\_merged\_tbl <- list(HRData = data) %>% append(definitions\_list, after = 1) %>%   
 reduce(left\_join) %>%   
 select(-one\_of(names(definitions\_list))) %>%   
 set\_names(str\_replace\_all(names(.), pattern = "\_value", replacement = "")) %>%   
 select(sort(names(.))) %>%   
 mutate\_if(is.character, as.factor) %>%  
 mutate(BusinessTravel = BusinessTravel %>% fct\_relevel("Non-Travel", "Travel\_Rarely", "Travel\_Frequently"),  
 MaritalStatus = MaritalStatus %>% fct\_relevel("Single", "Married", "Divorced"))  
   
 return(data\_merged\_tbl)  
}

## Data Preparation With Recipes!

This chapter is about preparing data for correlation analysis. Correlation Analysis is a great way to determine if you are getting good features prior to modeling. Always start with a correlation analysis.

### Custom Function: plot\_hist\_facet

To inspect feature distributions and identify transformations needed to make them properly formatted for correlation analysis.

* palette()[[3]] is just a hex code “18BC9C”
* as.factor orders factors alphabetically (as\_factor maintains the order in which they appear)
* using as.factor and as.numeric covers cases for characters and/or factors - noting special here
* gather does wide df to long df
* fact\_rev simply reverses the order of the factors
* facet\_wrap splits data into multiple graphs by a categorical columns - or set of columns -. Use functional format to specify facets (~key)

# Plot Faceted Histogram Function ----  
plot\_hist\_facet <- function(data, bins = 10, ncol = 5, fct\_reorder = FALSE, fct\_rev = FALSE,   
 fill = palette()[[3]], color = "white", scale = "free"){  
   
 data\_factored <- data %>% mutate\_if(is.character, as.factor) %>% mutate\_if(is.factor, as.numeric) %>%   
 gather(key = key, value = value, factor\_key = TRUE)  
   
 if(fct\_reorder){data\_factored <- data\_factored %>% mutate(key = as.character(key) %>% as.factor())}  
   
 if(fct\_rev){data\_factored <- data\_factored %>% mutate(key = fct\_rev(key))}  
   
 g <- data\_factored %>% ggplot(aes(x = value, group = key)) + geom\_histogram(bins = bins,   
 fill = fill, color = color) + facet\_wrap(~ key, ncol = ncol, scale = scale) + theme\_tq()  
 return(g)}

Here is the output as an example:

train\_raw\_tbl %>% select(Attrition, everything()) %>% plot\_hist\_facet(bins = 10, ncol =5, fct\_rev = F)

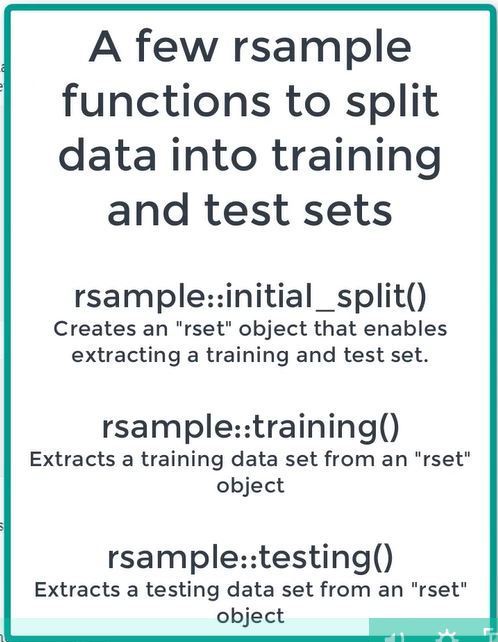


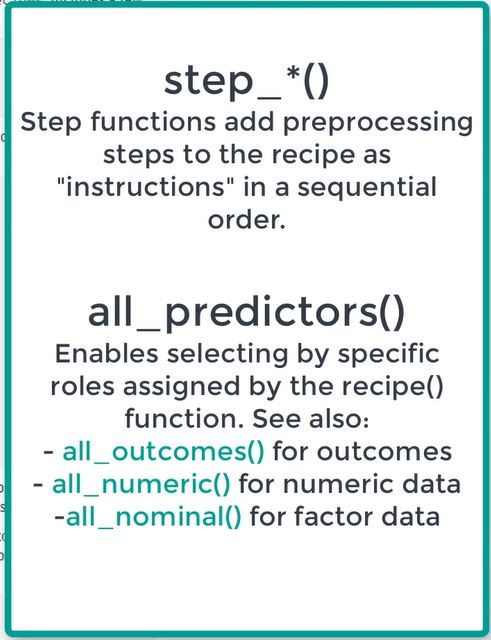
#ggsave("./images/plot\_hist.jpg")

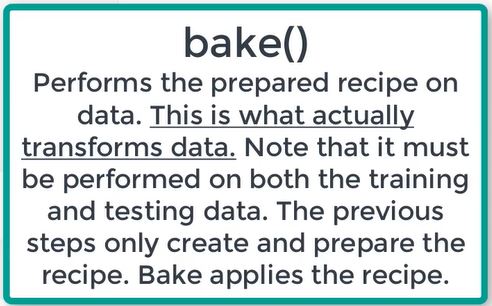
Useful to make the label the first plot

### recipes: Preprocessing Data For Machines

The idea of the [recipes package](https://tidymodels.github.io/recipes/) is to define a recipe or blueprint that can be used to sequentially define the encodings and preprocessing of the data (i.e. *feature engineering*).







In recipes, there are no constraints related to the order in which steps are added to the recipe. However, there are some general suggestions that you should consider:

* If using a Box-Cox transformation, don’t center the data first or do any operations that might make the data non-positive. Alternatively, use the Yeo-Johnson transformation so you don’t have to worry about this.
* Recipes do not automatically create dummy variables (unlike most formula methods). If you want to center, scale, or do any other operations on all of the predictors, run step\_dummy first so that numeric columns are in the data set instead of factors.
* You should make dummy variables before creating the interactions.
* If you are lumping infrequently categories together with step\_other, call step\_other before step\_dummy.

Here is a suggested order of potential steps that should work for most problems:

1. Impute
2. Individual transformations
   * changes to the data to remove skew (e.g. log), stabilize variance (e.g. Box Cox) or make stationary (e.g. difference for time series)
   * Normality is required for linear models that depend on correlation (correlation analysis, linear and logistic regression)
   * Non-linear models like random forest handle non-linear data because they depend on how the data can be segregated
3. Discretize (if needed and if you have no other choice)
   * discretation can hurt correlations. Often best not to discretize unless there is a specific need to do so
     + Example: Explain the difference between cohorts like Millennials vs Generation X
4. Create dummy variables
5. Create interactions
   * When 2 features have a relationship to each other there are said to *interact*. Example: The ratio between height and weight of a person is related to obesity. Height and weight *interact*.
   * Some advanced modeling - deep learning - detect interactions automatically
6. Normalization steps (center, scale, range, etc) **Prefer to make this #4 before dummy variables**
   * Some ML algorithms (PCA, KNN, Deep Learning) require scaled data
7. Multivariate transformation (e.g. PCA, spatial sign, etc)
   * Useful when data is wide and can be susceptible to over-fitting

For our data, here is the recipe plan we will implement:

* Plan: Correlation Analysis
  + Always do correlation analysis before modeling
  + Impute/Zero Variance
    - Imputation: The act of filling in missing values within features.
    - Common methods are:
      * filling in by recency - tidyr: fill
      * filling by similarity - knn impute
  + Transformations
  + Center and scale Dummy Data

We do not have missing values in the current data. However there are features with no variance.

**recipes: Setup**

First create a recipe from the original data and then specify the processing steps. Recipes can be created manually by sequentially adding roles to variables in a data set. If the analysis only required outcomes and predictors, the easiest way to create the initial recipe is to use the standard formula method:

recipe\_obj <- recipe(Attrition ~., data = train\_readable\_tbl)  
recipe\_obj

## Data Recipe  
##   
## Inputs:  
##   
## role #variables  
## outcome 1  
## predictor 34

The data contained in the data argument need not be the training set; this data is only used to catalog the names of the variables and their types (e.g. numeric, etc.).

**recipes: Zero Variance Features**

To add a step to remove zero variance features, it is simple to do:

recipe\_obj %>% step\_zv(all\_predictors())

## Data Recipe  
##   
## Inputs:  
##   
## role #variables  
## outcome 1  
## predictor 34  
##   
## Operations:  
##   
## Zero variance filter on all\_predictors()

prep() prepares the recipe performing preliminary calculations. Useful to see what the step is doing.

* prep() does not transform the data - it just determines is is needed to transform
* bake() performs transformation in new data (even if it is the same data you have already used) - bake takes as its newdata argument the name of the dataset typically used in the setup: recipe\_obj <- recipe(Attrition ~., data = train\_readable\_tbl). Useful to apply the recipe to any data set (commonly test and train)
* Essentially 3 steps:
  1. Create the instructions with the recipe and steps
  2. Prepare the recipe (prep())
  3. bake the recipe (`bake(newdata = testdataname)’)

recipe\_obj %>% step\_zv(all\_predictors()) %>% prep()

## Data Recipe  
##   
## Inputs:  
##   
## role #variables  
## outcome 1  
## predictor 34  
##   
## Training data contained 1250 data points and no missing data.  
##   
## Operations:  
##   
## Zero variance filter removed EmployeeCount, ... [trained]

It is helpful to know 3 features will be removed when the recipe is implemented.

**recipes: Transformations**

Identify which features have high skewness. Could use the facet plot illustrate above or compute using code (helpful when you have many features):

High Value –> fat tail on left | Low Value –> fat tail on right

skewness is a function from PerformanceAnalytics (package is loaded with tidyquant) computes the skewness of an univariate distribution.

factor\_key preserves the order of the variables

train\_readable\_tbl %>% select\_if(is.numeric) %>% map\_df(skewness) %>% gather(factor\_key = TRUE) %>% arrange(desc(value))

## # A tibble: 19 x 2  
## key value  
## <fct> <dbl>  
## 1 YearsSinceLastPromotion 1.94   
## 2 YearsAtCompany 1.76   
## 3 MonthlyIncome 1.35   
## 4 TotalWorkingYears 1.11   
## 5 NumCompaniesWorked 1.02   
## 6 JobLevel 1.02   
## 7 DistanceFromHome 0.985   
## 8 StockOptionLevel 0.968   
## 9 YearsInCurrentRole 0.933   
## 10 PercentSalaryHike 0.802   
## 11 YearsWithCurrManager 0.802   
## 12 TrainingTimesLastYear 0.557   
## 13 Age 0.422   
## 14 EmployeeNumber 0.0121   
## 15 MonthlyRate 0.0100   
## 16 DailyRate 0.00578  
## 17 HourlyRate -0.0263   
## 18 EmployeeCount NaN   
## 19 StandardHours NaN

# NOTE: high value --> fat tail on left, very low --> fat tail on right

Judgment is used to determine a cutoff for what is/not skewed. Evaluate the values from above:

* Appears to be a big difference between YearsWithCurrManager = 0.802 and TrainingTimesLastYear = 0.55

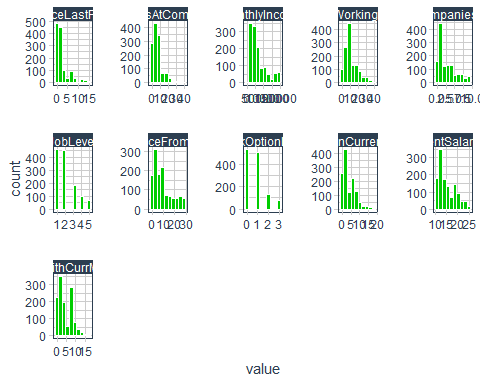
The facet plot indicates this too. as.character is used so we can plot. . . .

skewed\_feature\_names <- train\_readable\_tbl %>% select\_if(is.numeric) %>% map\_df(skewness) %>%   
 gather(factor\_key = TRUE) %>% arrange(desc(value)) %>% filter(value >= 0.8) %>%   
 pull(key) %>% as.character()  
skewed\_feature\_names

## [1] "YearsSinceLastPromotion" "YearsAtCompany"   
## [3] "MonthlyIncome" "TotalWorkingYears"   
## [5] "NumCompaniesWorked" "JobLevel"   
## [7] "DistanceFromHome" "StockOptionLevel"   
## [9] "YearsInCurrentRole" "PercentSalaryHike"   
## [11] "YearsWithCurrManager"

With this information, plot the features:

train\_readable\_tbl %>% select(skewed\_feature\_names) %>% plot\_hist\_facet()



The plot shows JobLevel and StockOptionLevel are factors and therefore are not transformed here (wait for dummying)

factor\_names <- c("JobLevel", "StockOptionLevel")  
  
skewed\_feature\_names <- train\_readable\_tbl %>% select\_if(is.numeric) %>% map\_df(skewness) %>%   
 gather(factor\_key = TRUE) %>% arrange(desc(value)) %>% filter(value >= 0.8) %>%   
 filter(!key %in% factor\_names) %>% pull(key) %>% as.character()

We now have the information we need to continue building our recipe:

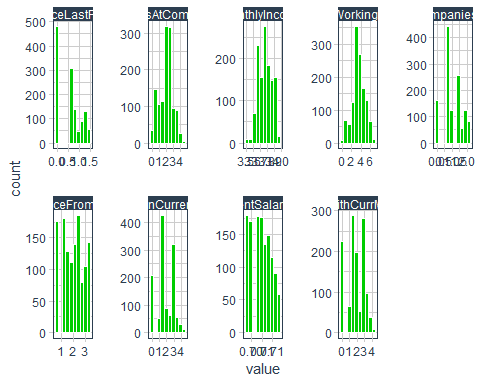
recipe\_obj <- recipe(Attrition ~., data = train\_readable\_tbl) %>% step\_zv(all\_predictors()) %>%   
 step\_YeoJohnson(skewed\_feature\_names) %>%   
 step\_num2factor(factor\_names)  
recipe\_obj %>% prep()

## Data Recipe  
##   
## Inputs:  
##   
## role #variables  
## outcome 1  
## predictor 34  
##   
## Training data contained 1250 data points and no missing data.  
##   
## Operations:  
##   
## Zero variance filter removed EmployeeCount, ... [trained]  
## Yeo-Johnson transformation on 9 items [trained]  
## Factor variables from JobLevel, StockOptionLevel [trained]

step\_YeoJohnson - power transformation. Instead of sqrt, looks for diff roots b/t -5 to 5 to eliminate skew

To make sure the recipe worked, facet plot the numeric features that were the target of the YeoJohnson transformation:

recipe\_obj %>% prep() %>% bake(train\_readable\_tbl) %>% select(skewed\_feature\_names) %>% plot\_hist\_facet()



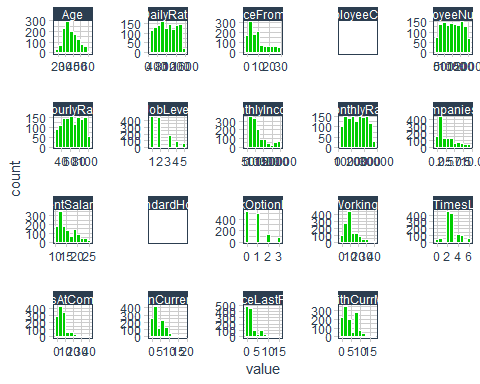
Skewness is significantly reduced!

**recipes: center & scale**

Required for algos that require scaling including kmeans, Deep Learning, PCA, SVMs.

Scaling does not typically hurt algos - so do it

train\_readable\_tbl %>% select\_if(is.numeric) %>% plot\_hist\_facet()



Note the x axis values are different from one and other - different ranges. DailyRate would likely dominate!

Must do in order: Center then Scale (C before S)

Note - H2O does not require centering and scaling - more on this later.

recipe\_obj <- recipe(Attrition ~., data = train\_readable\_tbl) %>% step\_zv(all\_predictors()) %>%   
 step\_YeoJohnson(skewed\_feature\_names) %>% step\_num2factor(factor\_names) %>%   
 step\_center(all\_numeric()) %>%   
 step\_scale(all\_numeric())  
recipe\_obj

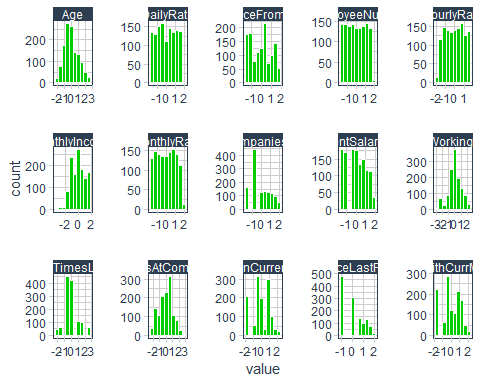
## Data Recipe  
##   
## Inputs:  
##   
## role #variables  
## outcome 1  
## predictor 34  
##   
## Operations:  
##   
## Zero variance filter on all\_predictors()  
## Yeo-Johnson transformation on skewed\_feature\_names  
## Factor variables from factor\_names  
## Centering for all\_numeric()  
## Scaling for all\_numeric()

The output from the recipe object is helpful but we are interested in understanding what it is really doing behind the scenes:

recipe\_obj %>% prep()#Note: recipe output is a list

## Data Recipe  
##   
## Inputs:  
##   
## role #variables  
## outcome 1  
## predictor 34  
##   
## Training data contained 1250 data points and no missing data.  
##   
## Operations:  
##   
## Zero variance filter removed EmployeeCount, ... [trained]  
## Yeo-Johnson transformation on 9 items [trained]  
## Factor variables from JobLevel, StockOptionLevel [trained]  
## Centering for Age, DailyRate, ... [trained]  
## Scaling for Age, DailyRate, ... [trained]

recipe\_obj %>% prep() %>% bake(newdata = train\_readable\_tbl) %>% select\_if(is.numeric) %>% plot\_hist\_facet()



All the x-axis scales are updated for the independent features!

**recipes: dummy variables**

One hot encoding - expands categorical variables into separate columns of 1/0. Important to detect patterns in unordered data.

all\_nominal() selects only categorical/factor data

To see what output we will get, evaluate only JobRole as our example:

dummied\_recipe\_obj <- recipe(Attrition ~., data = train\_readable\_tbl) %>% step\_zv(all\_predictors()) %>% step\_YeoJohnson(skewed\_feature\_names) %>%   
 step\_num2factor(factor\_names) %>% step\_center(all\_numeric()) %>% step\_scale(all\_numeric()) %>%   
 step\_dummy(all\_nominal()) %>% prep() %>% bake(newdata = train\_readable\_tbl) %>% select(contains("JobRole")) %>% plot\_hist\_facet(ncol = 3)

JobRole has been expanded to 8 columns (1 less than the number of unique values in JobRole. Healthcare Representative Manager was left out).

recipe::step\_dummy was updated with a new option - one\_hot: A logical. For C levels, should C dummy variables be created rather than C-1? Very useful for plotting!

recipe\_obj <- recipe(Attrition ~., data = train\_readable\_tbl) %>% step\_zv(all\_predictors()) %>% step\_YeoJohnson(skewed\_feature\_names) %>%   
 step\_num2factor(factor\_names) %>% step\_center(all\_numeric()) %>% step\_scale(all\_numeric()) %>%   
 step\_dummy(all\_nominal())

**recipes: Final Recipe**

Transform the data. . .. first review what the recipe will do . . . .

recipe\_obj <- recipe\_obj %>% prep()  
recipe\_obj

## Data Recipe  
##   
## Inputs:  
##   
## role #variables  
## outcome 1  
## predictor 34  
##   
## Training data contained 1250 data points and no missing data.  
##   
## Operations:  
##   
## Zero variance filter removed EmployeeCount, ... [trained]  
## Yeo-Johnson transformation on 9 items [trained]  
## Factor variables from JobLevel, StockOptionLevel [trained]  
## Centering for Age, DailyRate, ... [trained]  
## Scaling for Age, DailyRate, ... [trained]  
## Dummy variables from BusinessTravel, Department, ... [trained]

train\_tbl <- bake(recipe\_obj, newdata = train\_readable\_tbl)

test\_tbl <- bake(recipe\_obj, newdata = test\_readable\_tbl)

All of this effort is so we can perform a correlation analysis.

Note that the recipe changes our target label:

class(train\_tbl$Attrition\_Yes)

## [1] "numeric"

## Correlation Analysis

Note: Good introduction found [here](http://www.sthda.com/english/wiki/correlation-test-between-two-variables-in-r). May also help expand this to include non-linear relationships.

Correlation analysis only identifies linear relationships - if there is an exponential relationship

(A random forest will find the non-linear relationship.)

If there are no features with relatively strong correlation, it might be wise to collect more data.

Correlation is a good barometer but not necessarily a definitive answer - it is a guide.

### get\_cor

A function to take data and measure correlation.

* pairwise.complete.obs is not the default but it is what you nearly always want to use
  + If you use the default - “everything” - risk of getting missing value errors
* fct\_reorder reorders factor by another column. Changes the levels of a factor but not the DF - use arrange for that
* correlation analysis only works with numeric data
* quo\_name changes the stored value to “Attrition\_Yes” from a quosure
* mutate\_if(is.character, as.factor) %>% mutate\_if(is.factor, as.numeric) prevents function from failing is there is character data
* cor from stats returns a square correlation df correlating every feature against all the others. The return is square meaning the
  + use is an optional character string giving a method for computing covariances in the presence of missing values.
    - This must be (an abbreviation of) one of the strings “everything”, “all.obs”, “complete.obs”, “na.or.complete”, or “pairwise.complete.obs”
    - If use is “everything”, NAs will propagate conceptually, i.e., a resulting value will be NA whenever one of its contributing observations is NA.
    - If use is “all.obs”, then the presence of missing observations will produce an error. If use is “complete.obs” then missing values are handled by casewise deletion (and if there are no complete cases, that gives an error).
    - “na.or.complete” is the same unless there are no complete cases, that gives NA.
    - If use has the value “pairwise.complete.obs” then the correlation or covariance between each pair of variables is computed using all complete pairs of observations on those variables. This can result in covariance or correlation matrices which are not positive semi-definite, as well as NA entries if there are no complete pairs for that pair of variables.
* The (.) in the names function simply represents the data in the function. This creates a new character column called feature. this makes is square.
* select(feature, !! feature\_expr) produces a tibble with 2 columns - the new feature column and the Attrition\_Yes values
* filter(!(feature == feature\_name)) simply removes the value in the feature column that equals Attrition\_Yes - it simply is not needed/redundant
* Recall as\_factor maintains the order in which the values appear - not alphabetized
* fct\_reorder reorders a factor by another column. In this case, reorder by the Attrition\_Yes value. Helpful! Requires arrange() to update the df.
* fct\_rev reverses the ordering from the previous line of code

fct\_reorder and fct\_rev is useful for plotting

get\_cor <- function(data, target, use = "pairwise.complete.obs", fct\_reorder = FALSE, fct\_rev = FALSE){  
   
 feature\_expr <- enquo(target)  
 feature\_name <- quo\_name(feature\_expr)  
   
 data\_cor <- data %>% mutate\_if(is.character, as.factor) %>% mutate\_if(is.factor, as.numeric) %>%  
 cor(use = use) %>% as.tibble() %>% mutate(feature = names(.)) %>%   
 select(feature, !! feature\_expr) %>% filter(!(feature == feature\_name)) %>%  
 mutate\_if(is.character, as\_factor)  
   
 if(fct\_reorder){data\_cor <- data\_cor %>% mutate(feature = fct\_reorder(feature,   
 !! feature\_expr)) %>% arrange(feature)}  
   
 if(fct\_rev){data\_cor <- data\_cor %>% mutate(feature = fct\_rev(feature)) %>% arrange(feature)}  
   
 return(data\_cor)}

Example of get\_cor()

train\_tbl %>% get\_cor(Attrition\_Yes, fct\_reorder = T, fct\_rev = T)

## Warning in stats::cor(x, ...): the standard deviation is zero

## # A tibble: 65 x 2  
## feature Attrition\_Yes  
## <fct> <dbl>  
## 1 PerformanceRating\_Good NA   
## 2 OverTime\_Yes 0.240   
## 3 JobRole\_Sales.Representative 0.153   
## 4 BusinessTravel\_Travel\_Frequently 0.103   
## 5 Department\_Sales 0.0891  
## 6 DistanceFromHome 0.0822  
## 7 JobRole\_Laboratory.Technician 0.0737  
## 8 EducationField\_Technical.Degree 0.0731  
## 9 JobRole\_Human.Resources 0.0718  
## 10 JobInvolvement\_Medium 0.0604  
## # ... with 55 more rows

* Good correlation with real-world data could be +/-0.1 or greater. It just depends on the problem. It’s very often that the data is messy, but if you get one good feature or a few moderately good features (e.g. 0.2 or more) then this could be good.
* The goal with the preprocessing is to help improve the correlation even if the relationship is nonlinear. For example that is why we do YeoJohnson on some of the skewed data. However, there may be other ways. *We could definitely run a random forest and that will pick up any non linearity. We could then try to get feature importance.*

### Note on Multicolinearity

Multicolinearity is when two features in a model are highly correlated because it relates to whether or not the coefficients are uniquely identified. This relates to explanatory power of the model.

The goals are different for machine learning and traditional statistics, which is why multicolinearity is not traditionally checked for in ML but is in statistics.

**Machine Learning**: Assessing multicolinearity is not typically a concern in machine learning where the goal is to get a high performance prediction and more features helps in this pursuit. ML models can be linear or nonlinear and typically have regularization which fights over fitting, a situation that can be caused by many features that are correlated. This enables extreme accuracy, but the feature weights (coefficients) get lost, which reduces the explainability of the model.

In machine learning, we will have hundreds of features, but a few of the features will likely dominate for any given observation. This is where LIME comes in. LIME is a tool that allows us to cut through to the core of the features that are driving the explanation for any given model. It approximates nonlinear models locally, which enables us to detect which features are driving the prediction. It’s not impervious to colinear features but this is why we need to understand what features are going into a model from a business-intuition standpoint and discuss features with Subject Matter Experts. See this resource for more discussion on the LIME topic: <https://github.com/marcotcr/lime/issues/178>

**Traditional Statistics**: In traditional statistics it’s very common to inspect for multicolinearity and remove features. This is because the goal is to build a model using few features that explain the majority of the variance. When removing features, we can assess multicolinearity

**The Debate**: Because of the two sides, it’s debatable and it depends on your goals of high performance vs ability to explain with as few features as possible.

**My Opinion**: When it comes to modeling, my opinion is that we should strive for a high performing model once we know we have good features (features that have some predictive power based on a correlation assessment). When it comes to selecting features, we should meet with Subject Matter Experts and pick reasonable features to build a model around doing our best to not include the same feature twice, which is often where multicolinearity arises from (e.g. including birth year and age in the same model). When it comes to explaining, we should use tools like LIME that have the ability to hone in on the features that are contributing most to the prediction.

For more on this topic: <https://stats.stackexchange.com/questions/168622/why-is-multicollinearity-not-checked-in-modern-statistics-machine-learning>

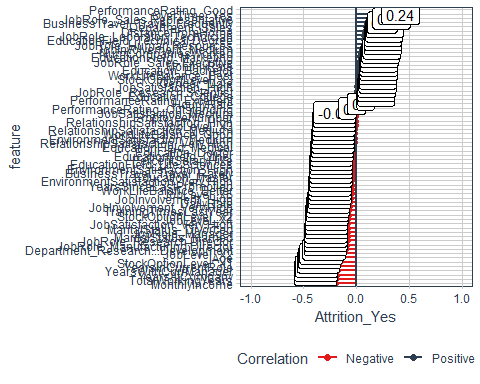
### plot\_cor

The data produced by get\_cor() is terrific, but understanding the data quickly is accomplished by creating a visualization.

The function below produces an informative visualization of the get\_cor() output. Do not be overwhelmed by the function. Many of the parameters are simply ggplot parameters.

* The beginning of the function looks much like get\_cor()
* feature\_name\_text is simply the Attriiotion\_Yes value rounded to the number of digits specified by lbl\_precision
* New feature named Correlation is created based on the value of Attrition\_Yes and outputs a factor = Positive or Negative
* Experiment with the ggplot code to understand how it works. (First set values for the function parameters.)

plot\_cor <- function(data, target, fct\_reorder = FALSE, fct\_rev = FALSE, include\_lbl = TRUE, lbl\_precision = 2, lbl\_position = "outward",  
 size = 2, line\_size = 1, vert\_size = 1, color\_pos = palette\_light()[[1]], color\_neg = palette\_light()[[2]]) {  
   
 feature\_expr <- enquo(target)  
 feature\_name <- quo\_name(feature\_expr)  
   
 data\_cor <- data %>%  
 get\_cor(!! feature\_expr, fct\_reorder = fct\_reorder, fct\_rev = fct\_rev) %>%  
 mutate(feature\_name\_text = round(!! feature\_expr, lbl\_precision)) %>%  
 mutate(Correlation = case\_when((!! feature\_expr) >= 0 ~ "Positive", TRUE ~ "Negative") %>% as.factor())  
   
 g <- data\_cor %>% ggplot(aes\_string(x = feature\_name, y = "feature", group = "feature")) +  
 geom\_point(aes(color = Correlation), size = size) +  
 geom\_segment(aes(xend = 0, yend = feature, color = Correlation), size = line\_size) +  
 geom\_vline(xintercept = 0, color = palette\_light()[[1]], size = vert\_size) +  
 expand\_limits(x = c(-1, 1)) + theme\_tq() + scale\_color\_manual(values = c(color\_neg, color\_pos))   
   
 if (include\_lbl) g <- g + geom\_label(aes(label = feature\_name\_text), hjust = lbl\_position)  
   
 return(g)}  
  
train\_tbl %>% plot\_cor(target = Attrition\_Yes, fct\_reorder = T, fct\_rev = F)



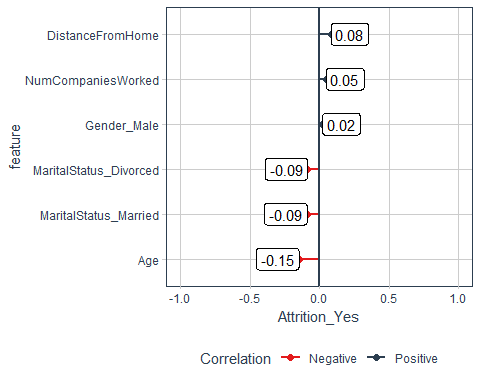
#ggsave("./images/plot\_cor.jpg")

### Evaluating plot\_cor Output

Data have been prepared and we developed a visualization function to easily evaluate feature correlations.

1. Descriptive Features: age, gender, marital status, etc

train\_tbl %>% select(Attrition\_Yes, Age, contains("Gender"), contains("MaritalStatus"), NumCompaniesWorked, contains("Over18"), DistanceFromHome) %>%   
 plot\_cor(target = Attrition\_Yes, fct\_reorder = T, fct\_rev = F)

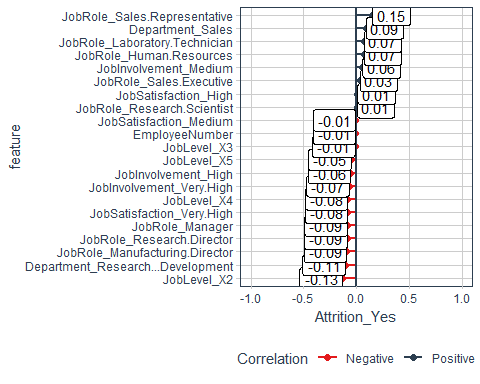


Observations:

* As age increases, less likely to leave company
  + DistanceFromHome has a relationship as dsitance from home supports someone more likely to leaeave the company

1. Employment Features

train\_tbl %>% select(Attrition\_Yes, contains("employee"), contains("department"), contains("job")) %>%   
plot\_cor(target = Attrition\_Yes, fct\_reorder = T, fct\_rev = F)



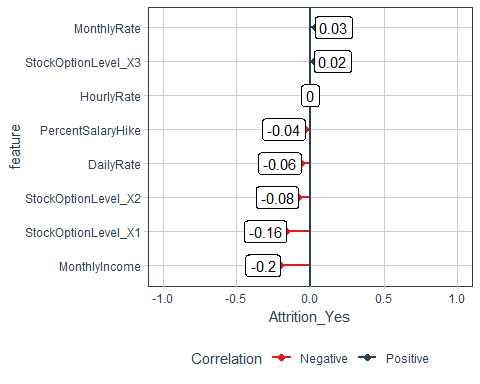
# contains is NOT case sentitive

Observations:

* Certain jobs either support or contradict the lieklihood to leave

1. Compensation Features

train\_tbl %>% select(Attrition\_Yes, contains("income"), contains("rate"), contains("salary"),   
 contains("stock")) %>%   
 plot\_cor(target = Attrition\_Yes, fct\_reorder = T, fct\_rev = F)

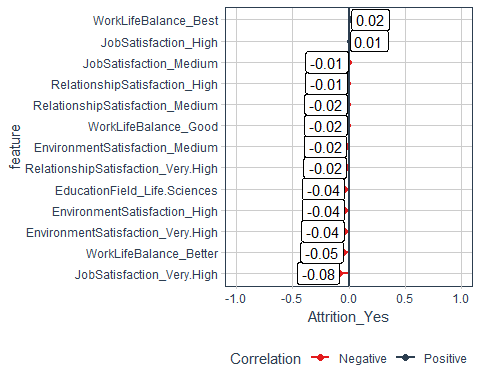


Observations:

-MonhtlyIncome StockOptionLevel are impactful

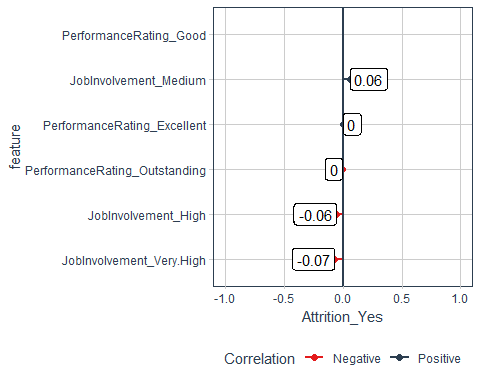
1. Survey Results

train\_tbl %>% select(Attrition\_Yes, contains("satisfaction"), contains("life")) %>%   
 plot\_cor(target = Attrition\_Yes, fct\_reorder = T, fct\_rev = F)



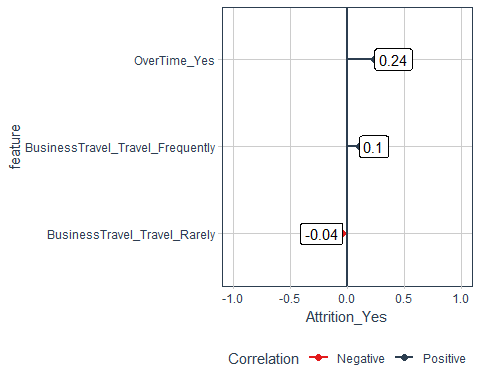
1. Performance Data

train\_tbl %>% select(Attrition\_Yes, contains("performance"), contains("involvement")) %>%   
 plot\_cor(target = Attrition\_Yes, fct\_reorder = T, fct\_rev = F)



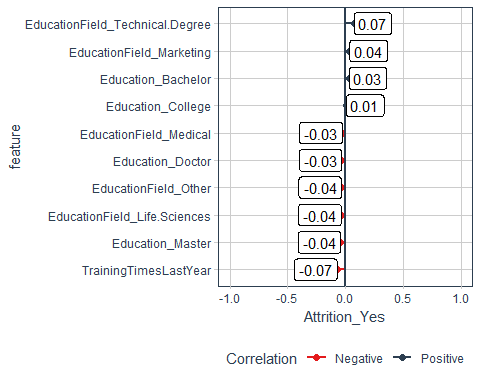
1. Work-Life Features

train\_tbl %>% select(Attrition\_Yes, contains("overtime"), contains("travel")) %>%   
 plot\_cor(target = Attrition\_Yes, fct\_reorder = T, fct\_rev = F)



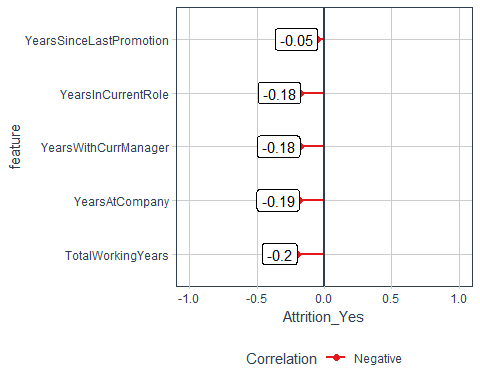
1. Training and Education

train\_tbl %>% select(Attrition\_Yes, contains("training"), contains("education")) %>%   
 plot\_cor(target = Attrition\_Yes, fct\_reorder = T, fct\_rev = F)



1. Time-Based Features

train\_tbl %>% select(Attrition\_Yes, contains("years")) %>% plot\_cor(target = Attrition\_Yes,   
 fct\_reorder = T, fct\_rev = F)

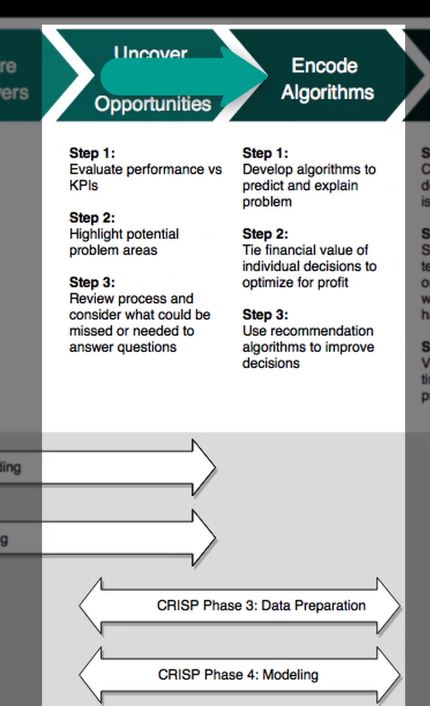


# Automated Machine Learning With H2O

A few random H2O notes:

* H2O automatically centers and scales the data. Most transformations are unnecessary.
* The dummy variables are handled internally by H2O provided that the format supplied is a factor.
* H2O may not complain, but LIME certainly will. My rule on missing data (and outliers) - Don’t ever leave missing data (or outliers) to chance, handle it yourself.
* Recommend doing feature engineering yourself because it’s usually better to apply some business knowledge rather than leave up to an algorithm that supposedly handles feature engineering.
* AutoML performs cross validation and grid search automatically! (We will learn about it anyway because you must know this as a data scientist!)

H2O has great [documentation](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html).



## H2O Modeling

**Update Since Original Modeling** The leaderboard\_frame is a legacy argument that is not necessary and the data is better served being used for the training and/or validation. Therefore, you should only perform one split (training and validation) as opposed to two (training, validation, test - used for leaderboard\_frame). This should increase model performance on the data set. You can try it both ways to see what happens. The leaderboard rankings performed later will be made on the Cross Validation metrics.

The main issue is that it reduces the amount of training data unnecessarily by using the leaderboard. The leaderboard used to be used for creating the Stacked Ensemble models. Now it is a legacy because of using the cross validation as a ranking mechanism. So what happens is that if we split off a set for the leaderboard, then we lose data that could have been used for training. Instead of doing the 70% training, 15% validation (test), and 15% leaderboard (test), we can now combine the leaderboard and training sets to get a higher accuracy model and just ranking by Cross Validation, which is performed on the training set internally within h2o.automl(). The split should be between 85% training and 15% validation. Leave the leaderboard = NULL, which is the preferred approach now.

h2o.init()  
h2o.no\_progress()  
split\_h2o <- h2o.splitFrame(as.h2o(train\_tbl), ratios = c(0.85), seed = 1234)  
  
train\_h2o <- split\_h2o[[1]]; valid\_h2o <- split\_h2o[[2]]  
test\_h2o <- as.h2o(test\_tbl)  
  
y <- "Attrition"; x <- setdiff(names(train\_h2o), y)  
  
automl\_models\_h2o <- h2o.automl(x = x, y = y, training\_frame = train\_h2o, validation\_frame = valid\_h2o,  
 leaderboard\_frame = NULL, max\_runtime\_secs = 30, nfolds = 5)

* max\_runtime\_secs = 30: default is 3600 = 1 hour. Once results look promising, increase run time to get more models with highly tuned parameters
* nfolds = 5: 10 folds more standard but trying to reduce model runtimes for this experiment

### Inspecting The Leaderboard

There is much data produced by the individual models - the leader is shown below:

automl\_models\_h2o@leader

## Model Details:  
## ==============  
##   
## H2OBinomialModel: glm  
## Model ID: GLM\_grid\_0\_AutoML\_20181028\_132826\_model\_0   
## GLM Model: summary  
## family link regularization  
## 1 binomial logit Ridge ( lambda = 0.0042 )  
## lambda\_search  
## 1 nlambda = 30, lambda.max = 6.2463, lambda.min = 0.0042, lambda.1se = 0.02055  
## number\_of\_predictors\_total number\_of\_active\_predictors  
## 1 79 79  
## number\_of\_iterations training\_frame  
## 1 54 \_8df924d3921afbf3d6ff0e6b3b584c0d  
##   
## Coefficients: glm coefficients  
## names coefficients standardized\_coefficients  
## 1 Intercept 0.849792 -1.417281  
## 2 JobRole.Healthcare Representative 0.066056 0.066056  
## 3 JobRole.Human Resources 0.167535 0.167535  
## 4 JobRole.Laboratory Technician 0.294450 0.294450  
## 5 JobRole.Manager -0.188477 -0.188477  
##   
## ---  
## names coefficients standardized\_coefficients  
## 75 TotalWorkingYears -0.042385 -0.330973  
## 76 TrainingTimesLastYear -0.180422 -0.231657  
## 77 YearsAtCompany 0.102775 0.649596  
## 78 YearsInCurrentRole -0.107887 -0.399926  
## 79 YearsSinceLastPromotion 0.098246 0.321247  
## 80 YearsWithCurrManager -0.118166 -0.425507  
##   
## H2OBinomialMetrics: glm  
## \*\* Reported on training data. \*\*  
##   
## MSE: 0.077  
## RMSE: 0.28  
## LogLoss: 0.27  
## Mean Per-Class Error: 0.19  
## AUC: 0.9  
## Gini: 0.8  
## R^2: 0.44  
## Residual Deviance: 573  
## AIC: 733  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 838 51 0.057368 =51/889  
## Yes 58 118 0.329545 =58/176  
## Totals 896 169 0.102347 =109/1065  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.348528 0.684058 128  
## 2 max f2 0.173866 0.739300 220  
## 3 max f0point5 0.517226 0.767857 79  
## 4 max accuracy 0.517226 0.906103 79  
## 5 max precision 0.962484 1.000000 0  
## 6 max recall 0.006173 1.000000 388  
## 7 max specificity 0.962484 1.000000 0  
## 8 max absolute\_mcc 0.348528 0.623206 128  
## 9 max min\_per\_class\_accuracy 0.189074 0.825647 210  
## 10 max mean\_per\_class\_accuracy 0.218153 0.836509 190  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`  
## H2OBinomialMetrics: glm  
## \*\* Reported on validation data. \*\*  
##   
## MSE: 0.086  
## RMSE: 0.29  
## LogLoss: 0.3  
## Mean Per-Class Error: 0.24  
## AUC: 0.81  
## Gini: 0.62  
## R^2: 0.26  
## Residual Deviance: 112  
## AIC: 272  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 148 12 0.075000 =12/160  
## Yes 10 15 0.400000 =10/25  
## Totals 158 27 0.118919 =22/185  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.314235 0.576923 26  
## 2 max f2 0.167744 0.620915 52  
## 3 max f0point5 0.648042 0.612245 5  
## 4 max accuracy 0.648042 0.897297 5  
## 5 max precision 0.933029 1.000000 0  
## 6 max recall 0.010212 1.000000 158  
## 7 max specificity 0.933029 1.000000 0  
## 8 max absolute\_mcc 0.314235 0.508369 26  
## 9 max min\_per\_class\_accuracy 0.167744 0.760000 52  
## 10 max mean\_per\_class\_accuracy 0.167744 0.773750 52  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`  
## H2OBinomialMetrics: glm  
## \*\* Reported on cross-validation data. \*\*  
## \*\* 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) \*\*  
##   
## MSE: 0.094  
## RMSE: 0.31  
## LogLoss: 0.32  
## Mean Per-Class Error: 0.24  
## AUC: 0.85  
## Gini: 0.69  
## R^2: 0.32  
## Residual Deviance: 684  
## AIC: 844  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 799 90 0.101237 =90/889  
## Yes 65 111 0.369318 =65/176  
## Totals 864 201 0.145540 =155/1065  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.303861 0.588859 155  
## 2 max f2 0.192745 0.659068 208  
## 3 max f0point5 0.481045 0.651408 86  
## 4 max accuracy 0.481045 0.881690 86  
## 5 max precision 0.949434 1.000000 0  
## 6 max recall 0.002279 1.000000 396  
## 7 max specificity 0.949434 1.000000 0  
## 8 max absolute\_mcc 0.359402 0.511298 127  
## 9 max min\_per\_class\_accuracy 0.173231 0.772727 219  
## 10 max mean\_per\_class\_accuracy 0.192745 0.781103 208  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`  
## Cross-Validation Metrics Summary:   
## mean sd cv\_1\_valid cv\_2\_valid cv\_3\_valid  
## accuracy 0.87323946 0.02347418 0.84976524 0.82159626 0.90140843  
## auc 0.8425771 0.026760576 0.8296127 0.8300225 0.9127907  
## err 0.12676056 0.02347418 0.15023474 0.17840375 0.09859155  
## err\_count 27.0 5.0 32.0 38.0 21.0  
## f0point5 0.63031274 0.07437051 0.6377551 0.45064378 0.7763975  
## cv\_4\_valid cv\_5\_valid  
## accuracy 0.91079813 0.8826291  
## auc 0.7989362 0.84152335  
## err 0.089201875 0.11737089  
## err\_count 19.0 25.0  
## f0point5 0.6179775 0.6687898  
##   
## ---  
## mean sd cv\_1\_valid cv\_2\_valid cv\_3\_valid  
## precision 0.6580986 0.096957624 0.65789473 0.4117647 0.8333333  
## r2 0.30837095 0.061606336 0.2931284 0.25788376 0.4637525  
## recall 0.5819287 0.06444905 0.5681818 0.7241379 0.6097561  
## residual\_deviance 135.58003 11.968308 165.20996 127.08945 117.54124  
## rmse 0.30468094 0.014051158 0.34037712 0.2954366 0.28870824  
## specificity 0.9306463 0.035521794 0.9230769 0.8369565 0.9709302  
## cv\_4\_valid cv\_5\_valid  
## precision 0.6875 0.7  
## r2 0.20434217 0.32274798  
## recall 0.44 0.5675676  
## residual\_deviance 125.3069 142.75262  
## rmse 0.28709954 0.31178322  
## specificity 0.9734042 0.9488636

H2O even provides a leaderboard - how cool is that?

automl\_models\_h2o@leaderboard

## model\_id auc logloss  
## 1 GLM\_grid\_0\_AutoML\_20181028\_132826\_model\_0 0.85 0.32  
## 2 StackedEnsemble\_AllModels\_0\_AutoML\_20181028\_132826 0.84 0.32  
## 3 StackedEnsemble\_BestOfFamily\_0\_AutoML\_20181028\_132826 0.84 0.32  
## 4 DeepLearning\_0\_AutoML\_20181028\_132826 0.82 0.37  
## 5 GBM\_grid\_0\_AutoML\_20181028\_132826\_model\_4 0.80 0.35  
## 6 GBM\_grid\_0\_AutoML\_20181028\_132826\_model\_12 0.80 0.35  
## mean\_per\_class\_error rmse mse  
## 1 0.24 0.31 0.094  
## 2 0.24 0.31 0.093  
## 3 0.25 0.31 0.093  
## 4 0.26 0.32 0.104  
## 5 0.27 0.32 0.105  
## 6 0.27 0.32 0.103  
##   
## [21 rows x 6 columns]

### Extracting Models From The Leaderboard

Getting the leader is pretty simple:

automl\_models\_h2o@leader

## Model Details:  
## ==============  
##   
## H2OBinomialModel: glm  
## Model ID: GLM\_grid\_0\_AutoML\_20181028\_132826\_model\_0   
## GLM Model: summary  
## family link regularization  
## 1 binomial logit Ridge ( lambda = 0.0042 )  
## lambda\_search  
## 1 nlambda = 30, lambda.max = 6.2463, lambda.min = 0.0042, lambda.1se = 0.02055  
## number\_of\_predictors\_total number\_of\_active\_predictors  
## 1 79 79  
## number\_of\_iterations training\_frame  
## 1 54 \_8df924d3921afbf3d6ff0e6b3b584c0d  
##   
## Coefficients: glm coefficients  
## names coefficients standardized\_coefficients  
## 1 Intercept 0.849792 -1.417281  
## 2 JobRole.Healthcare Representative 0.066056 0.066056  
## 3 JobRole.Human Resources 0.167535 0.167535  
## 4 JobRole.Laboratory Technician 0.294450 0.294450  
## 5 JobRole.Manager -0.188477 -0.188477  
##   
## ---  
## names coefficients standardized\_coefficients  
## 75 TotalWorkingYears -0.042385 -0.330973  
## 76 TrainingTimesLastYear -0.180422 -0.231657  
## 77 YearsAtCompany 0.102775 0.649596  
## 78 YearsInCurrentRole -0.107887 -0.399926  
## 79 YearsSinceLastPromotion 0.098246 0.321247  
## 80 YearsWithCurrManager -0.118166 -0.425507  
##   
## H2OBinomialMetrics: glm  
## \*\* Reported on training data. \*\*  
##   
## MSE: 0.077  
## RMSE: 0.28  
## LogLoss: 0.27  
## Mean Per-Class Error: 0.19  
## AUC: 0.9  
## Gini: 0.8  
## R^2: 0.44  
## Residual Deviance: 573  
## AIC: 733  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 838 51 0.057368 =51/889  
## Yes 58 118 0.329545 =58/176  
## Totals 896 169 0.102347 =109/1065  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.348528 0.684058 128  
## 2 max f2 0.173866 0.739300 220  
## 3 max f0point5 0.517226 0.767857 79  
## 4 max accuracy 0.517226 0.906103 79  
## 5 max precision 0.962484 1.000000 0  
## 6 max recall 0.006173 1.000000 388  
## 7 max specificity 0.962484 1.000000 0  
## 8 max absolute\_mcc 0.348528 0.623206 128  
## 9 max min\_per\_class\_accuracy 0.189074 0.825647 210  
## 10 max mean\_per\_class\_accuracy 0.218153 0.836509 190  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`  
## H2OBinomialMetrics: glm  
## \*\* Reported on validation data. \*\*  
##   
## MSE: 0.086  
## RMSE: 0.29  
## LogLoss: 0.3  
## Mean Per-Class Error: 0.24  
## AUC: 0.81  
## Gini: 0.62  
## R^2: 0.26  
## Residual Deviance: 112  
## AIC: 272  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 148 12 0.075000 =12/160  
## Yes 10 15 0.400000 =10/25  
## Totals 158 27 0.118919 =22/185  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.314235 0.576923 26  
## 2 max f2 0.167744 0.620915 52  
## 3 max f0point5 0.648042 0.612245 5  
## 4 max accuracy 0.648042 0.897297 5  
## 5 max precision 0.933029 1.000000 0  
## 6 max recall 0.010212 1.000000 158  
## 7 max specificity 0.933029 1.000000 0  
## 8 max absolute\_mcc 0.314235 0.508369 26  
## 9 max min\_per\_class\_accuracy 0.167744 0.760000 52  
## 10 max mean\_per\_class\_accuracy 0.167744 0.773750 52  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`  
## H2OBinomialMetrics: glm  
## \*\* Reported on cross-validation data. \*\*  
## \*\* 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) \*\*  
##   
## MSE: 0.094  
## RMSE: 0.31  
## LogLoss: 0.32  
## Mean Per-Class Error: 0.24  
## AUC: 0.85  
## Gini: 0.69  
## R^2: 0.32  
## Residual Deviance: 684  
## AIC: 844  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 799 90 0.101237 =90/889  
## Yes 65 111 0.369318 =65/176  
## Totals 864 201 0.145540 =155/1065  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.303861 0.588859 155  
## 2 max f2 0.192745 0.659068 208  
## 3 max f0point5 0.481045 0.651408 86  
## 4 max accuracy 0.481045 0.881690 86  
## 5 max precision 0.949434 1.000000 0  
## 6 max recall 0.002279 1.000000 396  
## 7 max specificity 0.949434 1.000000 0  
## 8 max absolute\_mcc 0.359402 0.511298 127  
## 9 max min\_per\_class\_accuracy 0.173231 0.772727 219  
## 10 max mean\_per\_class\_accuracy 0.192745 0.781103 208  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`  
## Cross-Validation Metrics Summary:   
## mean sd cv\_1\_valid cv\_2\_valid cv\_3\_valid  
## accuracy 0.87323946 0.02347418 0.84976524 0.82159626 0.90140843  
## auc 0.8425771 0.026760576 0.8296127 0.8300225 0.9127907  
## err 0.12676056 0.02347418 0.15023474 0.17840375 0.09859155  
## err\_count 27.0 5.0 32.0 38.0 21.0  
## f0point5 0.63031274 0.07437051 0.6377551 0.45064378 0.7763975  
## cv\_4\_valid cv\_5\_valid  
## accuracy 0.91079813 0.8826291  
## auc 0.7989362 0.84152335  
## err 0.089201875 0.11737089  
## err\_count 19.0 25.0  
## f0point5 0.6179775 0.6687898  
##   
## ---  
## mean sd cv\_1\_valid cv\_2\_valid cv\_3\_valid  
## precision 0.6580986 0.096957624 0.65789473 0.4117647 0.8333333  
## r2 0.30837095 0.061606336 0.2931284 0.25788376 0.4637525  
## recall 0.5819287 0.06444905 0.5681818 0.7241379 0.6097561  
## residual\_deviance 135.58003 11.968308 165.20996 127.08945 117.54124  
## rmse 0.30468094 0.014051158 0.34037712 0.2954366 0.28870824  
## specificity 0.9306463 0.035521794 0.9230769 0.8369565 0.9709302  
## cv\_4\_valid cv\_5\_valid  
## precision 0.6875 0.7  
## r2 0.20434217 0.32274798  
## recall 0.44 0.5675676  
## residual\_deviance 125.3069 142.75262  
## rmse 0.28709954 0.31178322  
## specificity 0.9734042 0.9488636

To evaluate a model other than the leader, you must use the model\_id in the leaderboard:

#h2o.getModel("StackedEnsemble\_BestOfFamily\_0\_AutoML\_20181028\_112225")

The code above is commented only because when the RMD is knitted, different models will be built and the model\_id will change.

### Custom Function: extract\_h2o\_model\_by\_position

extract\_h2o\_model\_name\_by\_position <- function(h2o\_leaderboard, position = 1, verbose = TRUE){  
   
 model\_name <- h2o\_leaderboard %>% as.tibble() %>% slice(position) %>% pull(model\_id)  
 if(verbose) message(model\_name)  
 return(model\_name)}

# Test new function  
extract\_h2o\_model\_name\_by\_position(automl\_models\_h2o@leaderboard, position = 3, verbose = T) %>% h2o.getModel()

## StackedEnsemble\_BestOfFamily\_0\_AutoML\_20181028\_132826

## Model Details:  
## ==============  
##   
## H2OBinomialModel: stackedensemble  
## Model ID: StackedEnsemble\_BestOfFamily\_0\_AutoML\_20181028\_132826   
## NULL  
##   
##   
## H2OBinomialMetrics: stackedensemble  
## \*\* Reported on training data. \*\*  
##   
## MSE: 0.061  
## RMSE: 0.25  
## LogLoss: 0.22  
## Mean Per-Class Error: 0.14  
## AUC: 0.94  
## Gini: 0.87  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 846 43 0.048369 =43/889  
## Yes 42 134 0.238636 =42/176  
## Totals 888 177 0.079812 =85/1065  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.305771 0.759207 153  
## 2 max f2 0.150512 0.779487 224  
## 3 max f0point5 0.602657 0.824830 89  
## 4 max accuracy 0.402301 0.923944 125  
## 5 max precision 0.979725 1.000000 0  
## 6 max recall 0.048042 1.000000 359  
## 7 max specificity 0.979725 1.000000 0  
## 8 max absolute\_mcc 0.354478 0.714517 137  
## 9 max min\_per\_class\_accuracy 0.150512 0.863636 224  
## 10 max mean\_per\_class\_accuracy 0.150512 0.864889 224  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`  
## H2OBinomialMetrics: stackedensemble  
## \*\* Reported on validation data. \*\*  
##   
## MSE: 0.083  
## RMSE: 0.29  
## LogLoss: 0.29  
## Mean Per-Class Error: 0.22  
## AUC: 0.83  
## Gini: 0.66  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 147 13 0.081250 =13/160  
## Yes 9 16 0.360000 =9/25  
## Totals 156 29 0.118919 =22/185  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.270417 0.592593 28  
## 2 max f2 0.253237 0.639098 32  
## 3 max f0point5 0.791260 0.612245 5  
## 4 max accuracy 0.791260 0.897297 5  
## 5 max precision 0.976481 1.000000 0  
## 6 max recall 0.043502 1.000000 142  
## 7 max specificity 0.976481 1.000000 0  
## 8 max absolute\_mcc 0.270417 0.525395 28  
## 9 max min\_per\_class\_accuracy 0.104461 0.743750 59  
## 10 max mean\_per\_class\_accuracy 0.253237 0.790000 32  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`  
## H2OBinomialMetrics: stackedensemble  
## \*\* Reported on cross-validation data. \*\*  
## \*\* 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) \*\*  
##   
## MSE: 0.093  
## RMSE: 0.31  
## LogLoss: 0.32  
## Mean Per-Class Error: 0.25  
## AUC: 0.84  
## Gini: 0.67  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 833 56 0.062992 =56/889  
## Yes 78 98 0.443182 =78/176  
## Totals 911 154 0.125822 =134/1065  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.336139 0.593939 130  
## 2 max f2 0.116247 0.643330 250  
## 3 max f0point5 0.562036 0.654297 75  
## 4 max accuracy 0.428604 0.882629 101  
## 5 max precision 0.976628 1.000000 0  
## 6 max recall 0.032434 1.000000 395  
## 7 max specificity 0.976628 1.000000 0  
## 8 max absolute\_mcc 0.350408 0.522861 126  
## 9 max min\_per\_class\_accuracy 0.118111 0.761364 247  
## 10 max mean\_per\_class\_accuracy 0.180189 0.773446 196  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

### Saving & Loading H2O Models

Saving models is simple. First use h2o.getModel then save it by running h2o.saveModel. Easy! Below the code is not evaluated because each time the RMD is knitted, the models will change.

h2o.getModel("GLM\_grid\_0\_AutoML\_20181028\_112225\_model\_0") %>%  
 h2o.saveModel(path = "04\_Modeling/h2o\_models/")  
  
h2o.getModel("StackedEnsemble\_BestOfFamily\_0\_AutoML\_20181028\_112225") %>%  
 h2o.saveModel(path = "04\_Modeling/h2o\_models/")

Loaded as saved model is equally simple using h2o.loadModel:

#h2o.loadModel("04\_Modeling/h2o\_models/GLM\_grid\_0\_AutoML\_20181028\_112225\_model\_0")

getModel returns a reference to an existing model in the H2O instance. loadModel loads a saved H2O model from disk.

### Making Predictions

Load a model:

stacked\_ensemble\_h2o <- h2o.loadModel("04\_Modeling/h2o\_models/GLM\_grid\_0\_AutoML\_20181028\_112225\_model\_0")

Make predictions with the model using test\_tbl and h2o.predict - again, simple!

predictions <- h2o.predict(stacked\_ensemble\_h2o, newdata = as.h2o(test\_tbl))  
predictions\_tbl <- predictions %>% as.tibble()   
predictions\_tbl %>% head()

## # A tibble: 6 x 3  
## predict No Yes  
## <fct> <dbl> <dbl>  
## 1 No 0.981 0.0195   
## 2 No 0.803 0.197   
## 3 No 0.999 0.000884  
## 4 No 0.974 0.0260   
## 5 Yes 0.338 0.662   
## 6 No 0.923 0.0766

For H2O Binary Classification, H2O provides 3 columns of data:

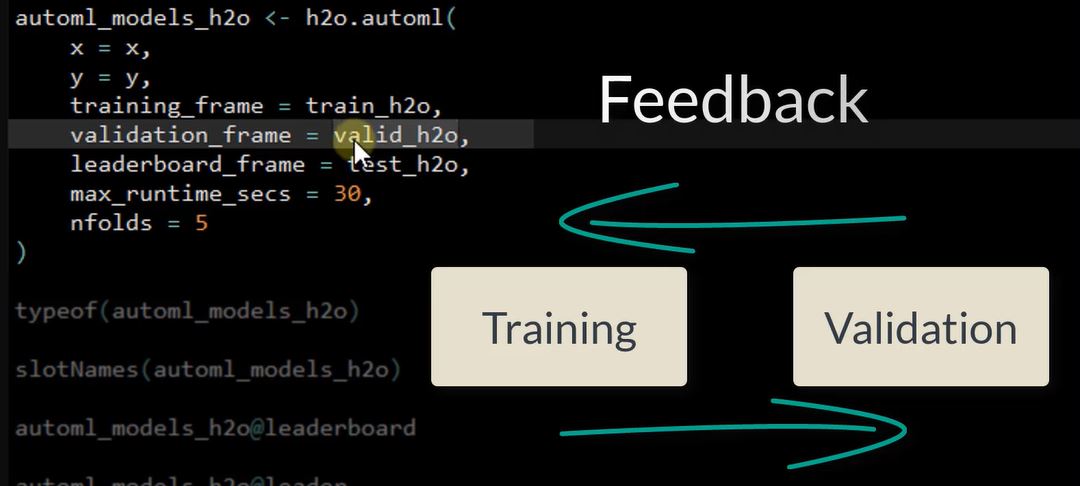
1. Class prediction
2. 1st class probability
3. 2nd class probability

## Advanced Concepts

### Train, Validation, & Leaderboard Frames

The models are trained on the training data and then sends the results to the validation data set and learns from the differences.

**Repeated Note**: The leaderboard\_frame is a legacy argument that is not necessary and the data is better served being used for the training and/or validation. Therefore, you should only perform one split (training and validation) as opposed to two (training, validation, test - used for leaderboard\_frame).



### H2O AutoML Model Parameters

When extracting the model parameters, there sufficient information so the model can be run outside of the H2o framework.H2O automated ML model leaderboard returns a bunch of models. Each model can be extracted and the contents can be opened up to determine the exact model parameters. The only one that will be difficult to recreate is stacked ensemble since this is a combination of models. Others such as GLM, GBM, and Deep Learning can be inspected and recreated. The Bonus Lecture on [Grid Search](#gridSearch).

#### Deep Learning Parameters

Use the @parameters option with the model:

automl\_models\_h2o@leader@parameters

## $model\_id  
## [1] "GLM\_grid\_0\_AutoML\_20181028\_132826\_model\_0"  
##   
## $training\_frame  
## [1] "\_8df924d3921afbf3d6ff0e6b3b584c0d"  
##   
## $validation\_frame  
## [1] "RTMP\_sid\_a61c\_5"  
##   
## $nfolds  
## [1] 5  
##   
## $seed  
## [1] 759443355997646848  
##   
## $keep\_cross\_validation\_predictions  
## [1] TRUE  
##   
## $fold\_assignment  
## [1] "Modulo"  
##   
## $family  
## [1] "binomial"  
##   
## $solver  
## [1] "COORDINATE\_DESCENT"  
##   
## $alpha  
## [1] 0.0 0.2 0.4 0.6 0.8 1.0  
##   
## $lambda  
## [1] 6.2463 4.5467 3.3095 2.4090 1.7535 1.2763 0.9290 0.6762 0.4922 0.3583  
## [11] 0.2608 0.1898 0.1382 0.1006 0.0732 0.0533 0.0388 0.0282 0.0206 0.0150  
## [21] 0.0109 0.0079 0.0058 0.0042 0.0031 0.0022 0.0016  
##   
## $lambda\_search  
## [1] TRUE  
##   
## $nlambdas  
## [1] 30  
##   
## $max\_iterations  
## [1] 300  
##   
## $objective\_epsilon  
## [1] 0.0001  
##   
## $gradient\_epsilon  
## [1] 0.000001  
##   
## $link  
## [1] "logit"  
##   
## $lambda\_min\_ratio  
## [1] 0.0001  
##   
## $max\_active\_predictors  
## [1] 5000  
##   
## $obj\_reg  
## [1] 0.00094  
##   
## $max\_runtime\_secs  
## [1] 4.3  
##   
## $x  
## [1] "JobRole" "EducationField"   
## [3] "JobLevel" "Education"   
## [5] "EnvironmentSatisfaction" "JobInvolvement"   
## [7] "JobSatisfaction" "RelationshipSatisfaction"  
## [9] "StockOptionLevel" "WorkLifeBalance"   
## [11] "Department" "BusinessTravel"   
## [13] "MaritalStatus" "Gender"   
## [15] "OverTime" "PerformanceRating"   
## [17] "Age" "DailyRate"   
## [19] "DistanceFromHome" "EmployeeNumber"   
## [21] "HourlyRate" "MonthlyIncome"   
## [23] "MonthlyRate" "NumCompaniesWorked"   
## [25] "PercentSalaryHike" "TotalWorkingYears"   
## [27] "TrainingTimesLastYear" "YearsAtCompany"   
## [29] "YearsInCurrentRole" "YearsSinceLastPromotion"   
## [31] "YearsWithCurrManager"   
##   
## $y  
## [1] "Attrition"

You can then select a parameter of interest:

automl\_models\_h2o@leader@parameters$nfolds

## [1] 5

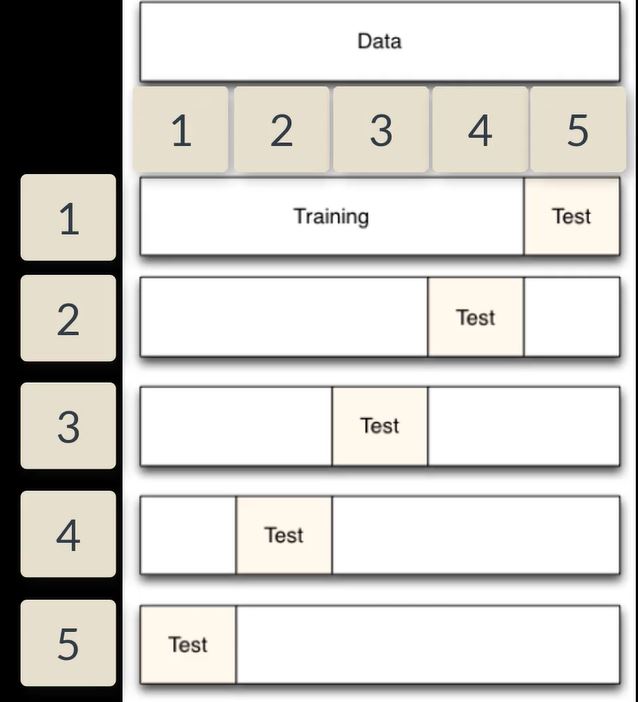
H2O provides complete access to the model parameters so you can recreate the model or tweak model parameters easily.

### Cross Validation (K-Fold CV)

[H2O Documentation](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/cross-validation.html)

nfolds is K-fold validation. K-fold validation is used in model development to determine what parameters affect prediction performance by performing multiple sub-tests rather than just one test. The end result is parameter selections based on model performance stability that generalizes better with new data.

For example, for nfolds=5, 6 models are built. The first 5 models (cross-validation models) are built on 80% of the training data, and a different 20% is held out for each of the 5 models. Then the main model is built on 100% of the training data. This main model is the model you get back from H2O in R, Python and Flow (though the CV models are also stored and available to access later).



All 5 cross-validation models contain training metrics (from the 80% training data) and validation metrics (from their 20% holdout/validation data). To compute their individual validation metrics, each of the 5 cross-validation models had to make predictions on their 20% of of rows of the original training frame, and score against the true labels of the 20% holdout.

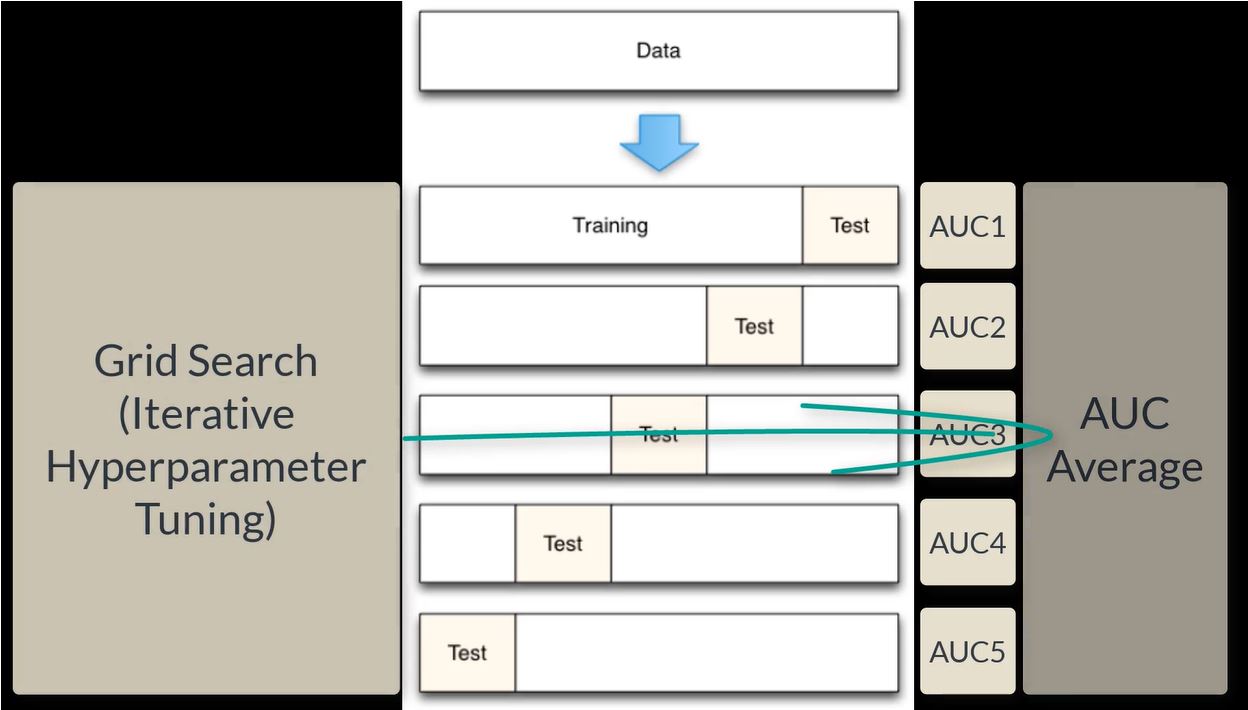
For the main model, this is how the cross-validation metrics are computed: The 5 holdout predictions are combined into one prediction for the full training dataset (i.e., predictions for every row of the training data, but the model making the prediction for a particular row has not seen that row during training). This “holdout prediction” is then scored against the true labels, and the overall cross-validation metrics are computed.

Each cv-model produces a prediction frame pertaining to its fold. It can be saved and probed from the various clients if keep\_cross\_validation\_predictions parameter is set in the model constructor.

These holdout predictions have some interesting properties. First they have names like:

prediction\_GBM\_model\_1452035702801\_1\_cv\_1

and they contain, unsurprisingly, predictions for the data held out in the fold. They also have the same number of rows as the entire input training frame with 0s filled in for all rows that are not in the hold out.



Retrieve the AUC for a classifier with the xval argument to retrieve the average cross validation AUC:

h2o.auc(automl\_models\_h2o@leader, train = T, valid = T, xval = T)# xval-cross-validation AUC

## train valid xval   
## 0.90 0.81 0.85

### Grid/Hyperparameter Search

[H2O Documetnation](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/grid-search.html)

H2O supports two types of grid search – traditional (or “cartesian”) grid search and random grid search. In a cartesian grid search, users specify a set of values for each hyperparameter that they want to search over, and H2O will train a model for every combination of the hyperparameter values. This means that if you have three hyperparameters and you specify 5, 10 and 2 values for each, your grid will contain a total of 5*10*2 = 100 models.

In random grid search, the user specifies the hyperparameter space in the exact same way, except H2O will sample uniformly from the set of all possible hyperparameter value combinations. In random grid search, the user also specifies a stopping criterion, which controls when the random grid search is completed. The user can tell the random grid search to stop by specifying a maximum number of models or the maximum number of seconds allowed for the search. The user may also specify a performance-metric-based stopping criterion, which will stop the random grid search when the performance stops improving by a specified amount.

Once the grid search is complete, the user can query the grid object and sort the models by a particular performance metric (for example, “AUC”). All models are stored in the H2O cluster and are accessible by model id.

**Grid Search**

Note the use of h2o.grid and hyper\_params

# GBM hyperparamters  
gbm\_params1 <- list(learn\_rate = c(0.01, 0.1), max\_depth = c(3, 5, 9), sample\_rate = c(0.8, 1.0),  
 col\_sample\_rate = c(0.2, 0.5, 1.0))  
  
# Train and validate a cartesian grid of GBMs  
gbm\_grid1 <- h2o.grid("gbm", x = x, y = y, grid\_id = "gbm\_grid1", training\_frame = train,   
 validation\_frame = valid, ntrees = 100, seed = 1,  
 hyper\_params = gbm\_params1)  
  
# Get the grid results, sorted by validation AUC  
gbm\_gridperf1 <- h2o.getGrid(grid\_id = "gbm\_grid1", sort\_by = "auc", decreasing = TRUE)  
print(gbm\_gridperf1)  
  
# Grab the top GBM model, chosen by validation AUC  
best\_gbm1 <- h2o.getModel(gbm\_gridperf1@model\_ids[[1]])  
  
# Now let's evaluate the model performance on a test set  
# so we get an honest estimate of top model performance  
best\_gbm\_perf1 <- h2o.performance(model = best\_gbm1,  
 newdata = test)  
h2o.auc(best\_gbm\_perf1) # 0.7781932  
  
# Look at the hyperparamters for the best model  
print(best\_gbm1@model[["model\_summary"]])

**Random Grid Search**

Note the use of seq, search\_criteria: search\_criteria <- list(strategy = "RandomDiscrete", max\_models = 36, seed = 1) and hyper\_parameters

gbm\_params2 <- list(learn\_rate = seq(0.01, 0.1, 0.01), ax\_depth = seq(2, 10, 1),  
 sample\_rate = seq(0.5, 1.0, 0.1), col\_sample\_rate = seq(0.1, 1.0, 0.1))  
search\_criteria <- list(strategy = "RandomDiscrete", max\_models = 36, seed = 1)  
  
# Train and validate a random grid of GBMs  
gbm\_grid2 <- h2o.grid("gbm", x = x, y = y, grid\_id = "gbm\_grid2", training\_frame = train,  
 validation\_frame = valid, ntrees = 100, eed = 1,  
 hyper\_params = gbm\_params2, earch\_criteria = search\_criteria)  
  
gbm\_gridperf2 <- h2o.getGrid(grid\_id = "gbm\_grid2", srt\_by = "auc", decreasing = TRUE)  
print(gbm\_gridperf2)  
  
# Grab the top GBM model, chosen by validation AUC  
best\_gbm2 <- h2o.getModel(gbm\_gridperf2@model\_ids[[1]])  
  
# Now let's evaluate the model performance on a test set  
# so we get an honest estimate of top model performance  
best\_gbm\_perf2 <- h2o.performance(model = best\_gbm2,  
 newdata = test)  
h2o.auc(best\_gbm\_perf2) # 0.7811332  
  
# Look at the hyperparamters for the best model  
print(best\_gbm2@model[["model\_summary"]])

Specify the RandomDiscrete strategy to perform a random search of all the combinations of your hyperparameters. RandomDiscrete should be combined with at least one early stopping criterion, max\_models and/or max\_runtime\_secs

Examples:

list(strategy = "RandomDiscrete", max\_models = 10, seed = 1)  
list(strategy = "RandomDiscrete", max\_runtime\_secs = 3600)  
list(strategy = "RandomDiscrete", max\_models = 42, max\_runtime\_secs = 28800)  
list(strategy = "RandomDiscrete", stopping\_tolerance = 0.001, stopping\_rounds = 10)  
list(strategy = "RandomDiscrete",   
 stopping\_metric = "misclassification", stopping\_tolerance = 0.0005, stopping\_rounds = 5)

To learn everything about this topic, must visit this [tutorial](https://blog.h2o.ai/2016/06/h2o-gbm-tuning-tutorial-for-r/)

H2O deep learning model sets nfolds = 0 by default!

## Visualizing The Leaderboard

Develop code to improve the output of H2O leaderboard:

automl\_models\_h2o@leaderboard

## model\_id auc logloss  
## 1 GLM\_grid\_0\_AutoML\_20181028\_132826\_model\_0 0.85 0.32  
## 2 StackedEnsemble\_AllModels\_0\_AutoML\_20181028\_132826 0.84 0.32  
## 3 StackedEnsemble\_BestOfFamily\_0\_AutoML\_20181028\_132826 0.84 0.32  
## 4 DeepLearning\_0\_AutoML\_20181028\_132826 0.82 0.37  
## 5 GBM\_grid\_0\_AutoML\_20181028\_132826\_model\_4 0.80 0.35  
## 6 GBM\_grid\_0\_AutoML\_20181028\_132826\_model\_12 0.80 0.35  
## mean\_per\_class\_error rmse mse  
## 1 0.24 0.31 0.094  
## 2 0.24 0.31 0.093  
## 3 0.25 0.31 0.093  
## 4 0.26 0.32 0.104  
## 5 0.27 0.32 0.105  
## 6 0.27 0.32 0.103  
##   
## [21 rows x 6 columns]

AUC is a way of measuring the performance of a binary classifier by comparing the false positive rate to the true positive rate. Logloss or logarithmic loss measures the performance of a classifier by comparing the class probability to actual value (1 or 0)

The AUC and Logloss metrics may suggest different models.

### Custom Function: plot\_h2o\_leaderboard

* str\_split simplify makes the output a matrix rather that a list. Makes it easier to extract data [,1]. This allows us to get the type of model from each model ID in the leaderboard.
* rownames\_to\_column adds row names of a data frame to a column. Gives us a column with the row name as a character. In this case it is very help because it identifies the order in which the models appear in the leaderboard.
* Note the use of as\_factor and as.factor. 'as\_factor allows us to order the data by AUC. Recall as.factor would order alphabetically.
* Use gather to take advantage of ggplot color and group aesthetics. Data must be in long format. Long format has each faceting values stacked on top of each other making the data frame long rather than wide. - the key column will have the valuesaucandlogloss. Thevaluecolumn will have the values ofaucandlogloss. gather transposes auc and logloss from two columns to one - key and their associated values will be in the value column - Before:

automl\_models\_h2o@leaderboard %>% as.tibble() %>%   
 mutate(model\_type = str\_split(model\_id, "\_", simplify = T)[,1]) %>% slice(1:10) %>%   
 rownames\_to\_column() %>%   
 mutate(model\_id = as\_factor(model\_id) %>% reorder(auc), model\_type = as.factor(model\_type))

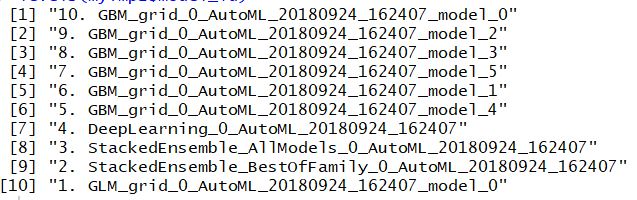
## # A tibble: 10 x 8  
## rowname model\_id auc logloss mean\_per\_class\_~ rmse mse model\_type  
## <chr> <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>   
## 1 1 GLM\_gri~ 0.846 0.321 0.235 0.307 0.0940 GLM   
## 2 2 Stacked~ 0.835 0.318 0.236 0.306 0.0934 StackedEn~  
## 3 3 Stacked~ 0.835 0.318 0.253 0.306 0.0934 StackedEn~  
## 4 4 DeepLea~ 0.815 0.371 0.255 0.322 0.104 DeepLearn~  
## 5 5 GBM\_gri~ 0.805 0.352 0.275 0.324 0.105 GBM   
## 6 6 GBM\_gri~ 0.802 0.347 0.266 0.321 0.103 GBM   
## 7 7 GBM\_gri~ 0.793 0.364 0.293 0.331 0.110 GBM   
## 8 8 GBM\_gri~ 0.791 0.360 0.273 0.329 0.108 GBM   
## 9 9 GBM\_gri~ 0.785 0.370 0.277 0.331 0.110 GBM   
## 10 10 GBM\_gri~ 0.779 0.379 0.275 0.335 0.112 GBM

* After:

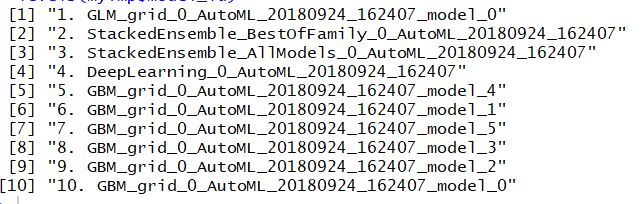
automl\_models\_h2o@leaderboard %>% as.tibble() %>%   
 mutate(model\_type = str\_split(model\_id, "\_", simplify = T)[,1]) %>% slice(1:10) %>%   
 rownames\_to\_column() %>%   
 mutate(model\_id = as\_factor(model\_id) %>% reorder(auc), model\_type = as.factor(model\_type)) %>%   
 gather(key = key, value = value, -c(model\_id, model\_type, rowname), factor\_key = T)

## # A tibble: 50 x 5  
## rowname model\_id model\_type key value  
## <chr> <fct> <fct> <fct> <dbl>  
## 1 1 GLM\_grid\_0\_AutoML\_20181028\_132826\_mo~ GLM auc 0.846  
## 2 2 StackedEnsemble\_AllModels\_0\_AutoML\_2~ StackedEnsem~ auc 0.835  
## 3 3 StackedEnsemble\_BestOfFamily\_0\_AutoM~ StackedEnsem~ auc 0.835  
## 4 4 DeepLearning\_0\_AutoML\_20181028\_132826 DeepLearning auc 0.815  
## 5 5 GBM\_grid\_0\_AutoML\_20181028\_132826\_mo~ GBM auc 0.805  
## 6 6 GBM\_grid\_0\_AutoML\_20181028\_132826\_mo~ GBM auc 0.802  
## 7 7 GBM\_grid\_0\_AutoML\_20181028\_132826\_mo~ GBM auc 0.793  
## 8 8 GBM\_grid\_0\_AutoML\_20181028\_132826\_mo~ GBM auc 0.791  
## 9 9 GBM\_grid\_0\_AutoML\_20181028\_132826\_mo~ GBM auc 0.785  
## 10 10 GBM\_grid\_0\_AutoML\_20181028\_132826\_mo~ GBM auc 0.779  
## # ... with 40 more rows

* factor\_rev() reverses the factor levels.
  + Without fct\_rev:



* With fct\_rev:

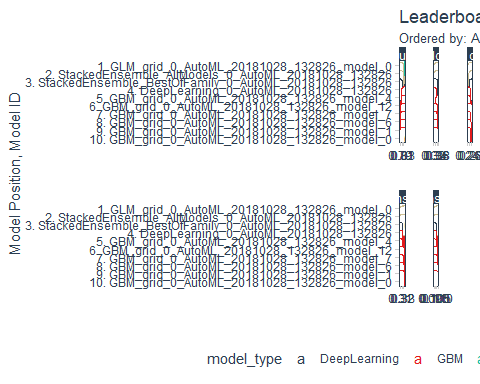


* scales in the ggplot: scales can be fixed (“fixed”, the default), free (“free”), or free in one dimension (“free\_x”, “free\_y”)

The course example used a deep learning model where only auc and logloss were available metrics. Other models may return those plus mean\_per\_class\_error, rmse, and mse.

# Visualize the H2O leaderboard to help with model selection  
plot\_h2o\_leaderboard <- function(h2o\_leaderboard, order\_by = c("auc", "logloss"),   
 n\_max = 20, size = 4, include\_lbl = TRUE) {  
 # Setup inputs  
 order\_by <- tolower(order\_by[[1]])  
 leaderboard\_tbl <- h2o\_leaderboard %>% as.tibble() %>%  
 mutate(model\_type = str\_split(model\_id, "\_", simplify = T) %>% .[,1]) %>%  
 rownames\_to\_column(var = "rowname") %>%  
 mutate(model\_id = paste0(rowname, ". ", as.character(model\_id)) %>% as.factor())  
   
 # Transformation  
 if (order\_by == "auc") {  
 data\_transformed\_tbl <- leaderboard\_tbl %>% slice(1:n\_max) %>%  
 mutate(model\_id = as\_factor(model\_id) %>% reorder(auc),  
 model\_type = as.factor(model\_type)) %>%  
 gather(key = key, value = value,   
 -c(model\_id, model\_type, rowname), factor\_key = T)  
   
 } else if (order\_by == "logloss") {  
 data\_transformed\_tbl <- leaderboard\_tbl %>% slice(1:n\_max) %>%  
 mutate(model\_id = as\_factor(model\_id) %>% reorder(logloss) %>% fct\_rev(),  
 model\_type = as.factor(model\_type)) %>%  
 gather(key = key, value = value, -c(model\_id, model\_type, rowname), factor\_key = T)  
   
 } else {stop(paste0("order\_by = '", order\_by, "' is not a permitted option."))}  
   
 # Visualization  
 g <- data\_transformed\_tbl %>%  
 ggplot(aes(value, model\_id, color = model\_type)) + geom\_point(size = size) +  
 facet\_wrap(~ key, scales = "free\_x") + theme\_tq() + scale\_color\_tq() +  
 labs(title = "Leaderboard Metrics", subtitle = paste0("Ordered by: ", toupper(order\_by)),  
 y = "Model Position, Model ID", x = "")  
   
 if (include\_lbl) g <- g + geom\_label(aes(label = round(value, 2), hjust = "inward"))  
 return(g)}

# Test Model Visualization  
automl\_models\_h2o@leaderboard %>% plot\_h2o\_leaderboard(n\_max = 10, order\_by = "auc")



#ggsave("./images/plotH2oLeaderboard.jpg")

The number of facets changes with the model type. Deep Learning models might provide AUC and Logloss. Other models may return rmse, mse, etc.

# Modeling Churn: Assessing H2O Performance

In this chapter, AUC, Logloss, Precision - Recall Plots, ROC Plots, Gain and Lift Charts.

A consolidated plot will also be created - one that is suitable for executive presentation.

## H2O Performance For Binary Classification

Create an H2O performance object. Could use h2o.loadModel() but it is simpler just to reuse the current leader. Also, lets get a deep learning model too.

myModel\_GLM <- automl\_models\_h2o@leader  
myModel\_Deep <- extract\_h2o\_model\_name\_by\_position(automl\_models\_h2o@leaderboard, position = 4, verbose = F) %>% h2o.getModel()

h2o.performance() makes an H2O performance object by providing a model and new data.

perf\_h2o\_GLM <- h2o.performance(myModel\_GLM, newdata = as.h2o(test\_tbl))  
perf\_h2o\_GLM

## H2OBinomialMetrics: glm  
##   
## MSE: 0.089  
## RMSE: 0.3  
## LogLoss: 0.3  
## Mean Per-Class Error: 0.22  
## AUC: 0.87  
## Gini: 0.74  
## R^2: 0.35  
## Residual Deviance: 131  
## AIC: 291  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 171 13 0.070652 =13/184  
## Yes 13 23 0.361111 =13/36  
## Totals 184 36 0.118182 =26/220  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.361471 0.638889 35  
## 2 max f2 0.305865 0.680628 46  
## 3 max f0point5 0.556700 0.714286 18  
## 4 max accuracy 0.556700 0.895455 18  
## 5 max precision 0.907863 1.000000 0  
## 6 max recall 0.040203 1.000000 145  
## 7 max specificity 0.907863 1.000000 0  
## 8 max absolute\_mcc 0.361471 0.568237 35  
## 9 max min\_per\_class\_accuracy 0.196433 0.777174 68  
## 10 max mean\_per\_class\_accuracy 0.305865 0.804046 46  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

The performance object is verbose!

perf\_h2o\_GLM@metrics

## $model  
## $model$`\_\_meta`  
## $model$`\_\_meta`$schema\_version  
## [1] 3  
##   
## $model$`\_\_meta`$schema\_name  
## [1] "ModelKeyV3"  
##   
## $model$`\_\_meta`$schema\_type  
## [1] "Key<Model>"  
##   
##   
## $model$name  
## [1] "GLM\_grid\_0\_AutoML\_20181028\_132826\_model\_0"  
##   
## $model$type  
## [1] "Key<Model>"  
##   
## $model$URL  
## [1] "/3/Models/GLM\_grid\_0\_AutoML\_20181028\_132826\_model\_0"  
##   
##   
## $model\_checksum  
## [1] 958794843843359360  
##   
## $frame  
## $frame$name  
## [1] "test\_tbl"  
##   
##   
## $frame\_checksum  
## [1] -54403433091472432  
##   
## $description  
## NULL  
##   
## $scoring\_time  
## [1] 1540747749401  
##   
## $predictions  
## NULL  
##   
## $MSE  
## [1] 0.089  
##   
## $RMSE  
## [1] 0.3  
##   
## $nobs  
## [1] 220  
##   
## $custom\_metric\_name  
## NULL  
##   
## $custom\_metric\_value  
## [1] 0  
##   
## $r2  
## [1] 0.35  
##   
## $logloss  
## [1] 0.3  
##   
## $AUC  
## [1] 0.87  
##   
## $pr\_auc  
## [1] 0.66  
##   
## $Gini  
## [1] 0.74  
##   
## $mean\_per\_class\_error  
## [1] 0.22  
##   
## $domain  
## [1] "No" "Yes"  
##   
## $cm  
## $cm$`\_\_meta`  
## $cm$`\_\_meta`$schema\_version  
## [1] 3  
##   
## $cm$`\_\_meta`$schema\_name  
## [1] "ConfusionMatrixV3"  
##   
## $cm$`\_\_meta`$schema\_type  
## [1] "ConfusionMatrix"  
##   
##   
## $cm$table  
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class  
## No Yes Error Rate  
## No 171 13 0.0707 = 13 / 184  
## Yes 13 23 0.3611 = 13 / 36  
## Totals 184 36 0.1182 = 26 / 220  
##   
##   
## $thresholds\_and\_metric\_scores  
## Metrics for Thresholds: Binomial metrics as a function of classification thresholds  
## threshold f1 f2 f0point5 accuracy precision recall  
## 1 0.907863 0.054054 0.034483 0.125000 0.840909 1.000000 0.027778  
## 2 0.815811 0.105263 0.068493 0.227273 0.845455 1.000000 0.055556  
## 3 0.801944 0.153846 0.102041 0.312500 0.850000 1.000000 0.083333  
## 4 0.769743 0.200000 0.135135 0.384615 0.854545 1.000000 0.111111  
## 5 0.720564 0.243902 0.167785 0.446429 0.859091 1.000000 0.138889  
## specificity absolute\_mcc min\_per\_class\_accuracy mean\_per\_class\_accuracy  
## 1 1.000000 0.152769 0.027778 0.513889  
## 2 1.000000 0.216543 0.055556 0.527778  
## 3 1.000000 0.265820 0.083333 0.541667  
## 4 1.000000 0.307653 0.111111 0.555556  
## 5 1.000000 0.344765 0.138889 0.569444  
## tns fns fps tps tnr fnr fpr tpr idx  
## 1 184 35 0 1 1.000000 0.972222 0.000000 0.027778 0  
## 2 184 34 0 2 1.000000 0.944444 0.000000 0.055556 1  
## 3 184 33 0 3 1.000000 0.916667 0.000000 0.083333 2  
## 4 184 32 0 4 1.000000 0.888889 0.000000 0.111111 3  
## 5 184 31 0 5 1.000000 0.861111 0.000000 0.138889 4  
##   
## ---  
## threshold f1 f2 f0point5 accuracy precision recall  
## 215 0.001598 0.286853 0.501393 0.200893 0.186364 0.167442 1.000000  
## 216 0.001287 0.285714 0.500000 0.200000 0.181818 0.166667 1.000000  
## 217 0.001130 0.284585 0.498615 0.199115 0.177273 0.165899 1.000000  
## 218 0.000965 0.283465 0.497238 0.198238 0.172727 0.165138 1.000000  
## 219 0.000884 0.282353 0.495868 0.197368 0.168182 0.164384 1.000000  
## 220 0.000824 0.281250 0.494505 0.196507 0.163636 0.163636 1.000000  
## specificity absolute\_mcc min\_per\_class\_accuracy  
## 215 0.027174 0.067454 0.027174  
## 216 0.021739 0.060193 0.021739  
## 217 0.016304 0.052008 0.016304  
## 218 0.010870 0.042367 0.010870  
## 219 0.005435 0.029890 0.005435  
## 220 0.000000 0.000000 0.000000  
## mean\_per\_class\_accuracy tns fns fps tps tnr fnr fpr  
## 215 0.513587 5 0 179 36 0.027174 0.000000 0.972826  
## 216 0.510870 4 0 180 36 0.021739 0.000000 0.978261  
## 217 0.508152 3 0 181 36 0.016304 0.000000 0.983696  
## 218 0.505435 2 0 182 36 0.010870 0.000000 0.989130  
## 219 0.502717 1 0 183 36 0.005435 0.000000 0.994565  
## 220 0.500000 0 0 184 36 0.000000 0.000000 1.000000  
## tpr idx  
## 215 1.000000 214  
## 216 1.000000 215  
## 217 1.000000 216  
## 218 1.000000 217  
## 219 1.000000 218  
## 220 1.000000 219  
##   
## $max\_criteria\_and\_metric\_scores  
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.361471 0.638889 35  
## 2 max f2 0.305865 0.680628 46  
## 3 max f0point5 0.556700 0.714286 18  
## 4 max accuracy 0.556700 0.895455 18  
## 5 max precision 0.907863 1.000000 0  
## 6 max recall 0.040203 1.000000 145  
## 7 max specificity 0.907863 1.000000 0  
## 8 max absolute\_mcc 0.361471 0.568237 35  
## 9 max min\_per\_class\_accuracy 0.196433 0.777174 68  
## 10 max mean\_per\_class\_accuracy 0.305865 0.804046 46  
##   
## $gains\_lift\_table  
## Gains/Lift Table: Avg response rate: 16.36 %, avg score: 17.06 %  
## group cumulative\_data\_fraction lower\_threshold lift cumulative\_lift  
## 1 1 0.01363636 0.795825 6.111111 6.111111  
## 2 2 0.02272727 0.698406 6.111111 6.111111  
## 3 3 0.03181818 0.657941 6.111111 6.111111  
## 4 4 0.04090909 0.631589 6.111111 6.111111  
## 5 5 0.05000000 0.610115 3.055556 5.555556  
## 6 6 0.10000000 0.493094 3.888889 4.722222  
## 7 7 0.15000000 0.373599 2.222222 3.888889  
## 8 8 0.20000000 0.314707 2.222222 3.472222  
## 9 9 0.30000000 0.200597 0.555556 2.500000  
## 10 10 0.40000000 0.128708 0.833333 2.083333  
## 11 11 0.50000000 0.087737 0.555556 1.777778  
## 12 12 0.60000000 0.055181 0.277778 1.527778  
## 13 13 0.70000000 0.031811 0.833333 1.428571  
## 14 14 0.80000000 0.017333 0.000000 1.250000  
## 15 15 0.90000000 0.008458 0.000000 1.111111  
## 16 16 1.00000000 0.000824 0.000000 1.000000  
## response\_rate score cumulative\_response\_rate cumulative\_score  
## 1 1.000000 0.841873 1.000000 0.841873  
## 2 1.000000 0.745154 1.000000 0.803185  
## 3 1.000000 0.662249 1.000000 0.762918  
## 4 1.000000 0.644891 1.000000 0.736689  
## 5 0.500000 0.625222 0.909091 0.716423  
## 6 0.636364 0.569948 0.772727 0.643185  
## 7 0.363636 0.425287 0.636364 0.570552  
## 8 0.363636 0.343600 0.568182 0.513814  
## 9 0.090909 0.256735 0.409091 0.428121  
## 10 0.136364 0.163885 0.340909 0.362062  
## 11 0.090909 0.103809 0.290909 0.310412  
## 12 0.045455 0.071676 0.250000 0.270622  
## 13 0.136364 0.041599 0.233766 0.237905  
## 14 0.000000 0.024287 0.204545 0.211202  
## 15 0.000000 0.012672 0.181818 0.189144  
## 16 0.000000 0.004134 0.163636 0.170643  
## capture\_rate cumulative\_capture\_rate gain cumulative\_gain  
## 1 0.083333 0.083333 511.111111 511.111111  
## 2 0.055556 0.138889 511.111111 511.111111  
## 3 0.055556 0.194444 511.111111 511.111111  
## 4 0.055556 0.250000 511.111111 511.111111  
## 5 0.027778 0.277778 205.555556 455.555556  
## 6 0.194444 0.472222 288.888889 372.222222  
## 7 0.111111 0.583333 122.222222 288.888889  
## 8 0.111111 0.694444 122.222222 247.222222  
## 9 0.055556 0.750000 -44.444444 150.000000  
## 10 0.083333 0.833333 -16.666667 108.333333  
## 11 0.055556 0.888889 -44.444444 77.777778  
## 12 0.027778 0.916667 -72.222222 52.777778  
## 13 0.083333 1.000000 -16.666667 42.857143  
## 14 0.000000 1.000000 -100.000000 25.000000  
## 15 0.000000 1.000000 -100.000000 11.111111  
## 16 0.000000 1.000000 -100.000000 0.000000  
##   
## $residual\_deviance  
## [1] 131  
##   
## $null\_deviance  
## [1] 196  
##   
## $AIC  
## [1] 291  
##   
## $null\_degrees\_of\_freedom  
## [1] 219  
##   
## $residual\_degrees\_of\_freedom  
## [1] 140

Accessing the information from the performance object is simple!

Some metrics are directly available like auc: **A**rea **U**nder the **C**urve, referring to the **R**eciever **O**perating **C**haracteristics (ROC) plot that measures the true positive rate (TPR) and the false positive rate (FPR)

h2o.auc(perf\_h2o\_GLM)

## [1] 0.87

`h2o.auc() provides options for returning values for train, validation, and cross validation. However, these options are valid only for models, not performance objects.

This works:

h2o.auc(myModel\_GLM,train = T, valid = T, xval = T)

## train valid xval   
## 0.90 0.81 0.85

This does not return multiple values:

h2o.auc(perf\_h2o\_GLM, train = Tt, valid = T, xval = T)

## [1] 0.87

### Gini Coefficient:

|

The Gini Coefficient is a popular metric on Kaggle, especially for imbalanced class values.

The Gini coefficient is one of many to measure divergence, i.e. the difference in the distribution of two variables in a data set. Values range from -1 to 1 to indicate total divergence one direction or the other, and 0 percent representing no divergence.

h2o.giniCoef(perf\_h2o\_GLM)# again gives result based on the test data

## [1] 0.74

h2o.giniCoef(myModel\_GLM, train = T, valid = T, xval = T)

## train valid xval   
## 0.80 0.62 0.69

### Logloss

Logloss is a metric that measures the class probability from the model against the actual value in binary format (0/1). See [this for more information](https://www.r-bloggers.com/making-sense-of-logarithmic-loss/).

Log Loss quantifies the accuracy of a classifier by penalizing false classifications. Minimizing the Log Loss is basically equivalent to maximizing the accuracy of the classifier. A perfect classifier would have a Log Loss of precisely zero. Less ideal classifiers have progressively larger values of Log Loss.

Log Loss heavily penalizes classifiers that are confident about an incorrect classification. For example, if for a particular observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the Log Loss will be very large indeed. Naturally this is going to have a significant impact on the overall Log Loss for the classifier. The bottom line is that it’s better to be somewhat wrong than emphatically wrong. Of course it’s always better to be completely right, but that is seldom achievable in practice! There are at least two approaches to dealing with poor classifications:

1. Examine the problematic observations relative to the full data set. Are they simply outliers? In this case, remove them from the data and re-train the classifier.
2. Consider smoothing the predicted probabilities using, for example, [Laplace Smoothing](https://en.wikipedia.org/wiki/Additive_smoothing). This will result in a less “certain” classifier and might improve the overall Log Loss.

h2o.logloss(perf\_h2o\_GLM)

## [1] 0.3

h2o.logloss(myModel\_GLM, train = T, valid = T, xval = T)

## train valid xval   
## 0.27 0.30 0.32

### Confusion Matrix

h2o.confusionMatrix(perf\_h2o\_GLM)# test data

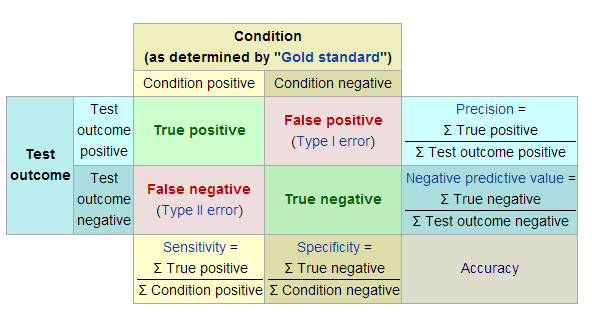
## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.361470825344551:  
## No Yes Error Rate  
## No 171 13 0.070652 =13/184  
## Yes 13 23 0.361111 =13/36  
## Totals 184 36 0.118182 =26/220

h2o.confusionMatrix(myModel\_GLM)#training data

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.348527593514236:  
## No Yes Error Rate  
## No 838 51 0.057368 =51/889  
## Yes 58 118 0.329545 =58/176  
## Totals 896 169 0.102347 =109/1065

Must understand threshold, precision and recall:

* Top Row - Predictions; Left Column - Actual
* Threshold is the value that determines which class probability is a 1 or 0
  + Example: if a 1 is assigned if the probability is above the threshold value - which is a probability
* F1 is the optimal balance between precision and recall. The threshold maximizes F1. However, this is not always the best case. An expected value optimization is required when the costs of false positives and false negatives are known.
  + F1 = ( \* (precision \* recall))/ (precision + recall) - hence balances precision & recall
* PRECISION - Measures false positives (eg, predicted to leave but actually stay) Precision = TP/(TP + FP) : It detects how frequently the algo over-picks the YES class
* RECALL - Measures false negatives (eg, predicted to stay but actually leave)
  + Recall = TP/(TP + FN) - It provides a metric for under-picking YES
* Because there are different costs associated with FN and FP; FN often cost a company more.  
  This is where Expected Value (EV) comes into play - this discussion is coming.



## H2O Metrics: Analyze Confusion Matrix

Review the output from the confusion matrix.

h2o.confusionMatrix(myModel\_GLM)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.348527593514236:  
## No Yes Error Rate  
## No 838 51 0.057368 =51/889  
## Yes 58 118 0.329545 =58/176  
## Totals 896 169 0.102347 =109/1065

How does the threshold max f1 affect the models? Use the h2o.metric. Make the metric easier to read by turning it into a tibble.

performance\_tbl <- perf\_h2o\_GLM %>% h2o.metric() %>% as.tibble()  
performance\_tbl

## # A tibble: 220 x 20  
## threshold f1 f2 f0point5 accuracy precision recall specificity  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.908 0.0541 0.0345 0.125 0.841 1 0.0278 1  
## 2 0.816 0.105 0.0685 0.227 0.845 1 0.0556 1  
## 3 0.802 0.154 0.102 0.312 0.85 1 0.0833 1  
## 4 0.770 0.20 0.135 0.385 0.855 1 0.111 1  
## 5 0.721 0.244 0.168 0.446 0.859 1 0.139 1  
## 6 0.662 0.286 0.200 0.5 0.864 1 0.167 1  
## 7 0.662 0.326 0.232 0.547 0.868 1 0.194 1  
## 8 0.655 0.364 0.263 0.588 0.873 1 0.222 1  
## 9 0.635 0.4 0.294 0.625 0.877 1 0.25 1  
## 10 0.630 0.435 0.325 0.658 0.882 1 0.278 1  
## # ... with 210 more rows, and 12 more variables: absolute\_mcc <dbl>,  
## # min\_per\_class\_accuracy <dbl>, mean\_per\_class\_accuracy <dbl>,  
## # tns <dbl>, fns <dbl>, fps <dbl>, tps <dbl>, tnr <dbl>, fnr <dbl>,  
## # fpr <dbl>, tpr <dbl>, idx <int>

## Precision, Recall, F1 & Effect Of Threshold

**Precision**

In other words, Precision detects how frequently we over-pick the Yes class. In the confusion matrix above, 51 records were over picked - these employees were predicted to leave but did not. (No real harm in this - the company did not lose money.)

118/(51+118)

## [1] 0.7

This is the percent of time the algorithm predicted someone would leave correctly.

**Recall**

Recall provides a metric for under-picking Yes class.

118/(58 + 118)

## [1] 0.67

This measures how often will miss people that leave by predicting they will stay.

Recall is typically more important than Precision because you usually would rather give up some false positives (inadvertently target stayers) to gain false negatives (accurately predict quitters).

**F1**

F1 balances precision and recall.

(2 \* (0.7 \* 0.67))/(0.7 + 0.67)

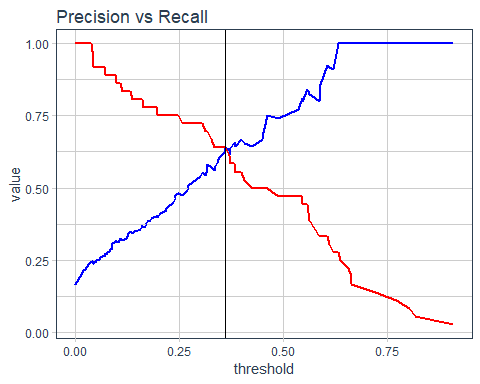
## [1] 0.68

The threshold is the limit that maximizes F1 across all possibilities.

**Threshold**{#precisionrecallplot}

Note the use of a new H~2~O function called h2o.find\_threshold\_by\_max\_metric.

performance\_tbl %>% ggplot(aes(x = threshold)) +   
 geom\_line(aes(y = precision), color = "blue", size = 1) +  
 geom\_line(aes(y = recall), color = "red", size = 1) +  
 geom\_vline(xintercept = h2o.find\_threshold\_by\_max\_metric(perf\_h2o\_GLM, "f1")) +  
 theme\_tq() + labs(title = "Precision vs Recall", y = "value")



## Performance Charts For Data Scientists

To compare models, we need to load a few more models. A Deep Model has already been saved. Lets save one of the Stacked Models.

automl\_models\_h2o@leaderboard

## model\_id auc logloss  
## 1 GLM\_grid\_0\_AutoML\_20181028\_132826\_model\_0 0.85 0.32  
## 2 StackedEnsemble\_AllModels\_0\_AutoML\_20181028\_132826 0.84 0.32  
## 3 StackedEnsemble\_BestOfFamily\_0\_AutoML\_20181028\_132826 0.84 0.32  
## 4 DeepLearning\_0\_AutoML\_20181028\_132826 0.82 0.37  
## 5 GBM\_grid\_0\_AutoML\_20181028\_132826\_model\_4 0.80 0.35  
## 6 GBM\_grid\_0\_AutoML\_20181028\_132826\_model\_12 0.80 0.35  
## mean\_per\_class\_error rmse mse  
## 1 0.24 0.31 0.094  
## 2 0.24 0.31 0.093  
## 3 0.25 0.31 0.093  
## 4 0.26 0.32 0.104  
## 5 0.27 0.32 0.105  
## 6 0.27 0.32 0.103  
##   
## [21 rows x 6 columns]

myModel\_stacked <- extract\_h2o\_model\_name\_by\_position(automl\_models\_h2o@leaderboard, position = 2, verbose = F) %>% h2o.getModel()

Next collect the performance metrics for the Deep Learning and Stacked models. Develop a new functions: load\_model\_performance\_metrics and Model\_Metrics.

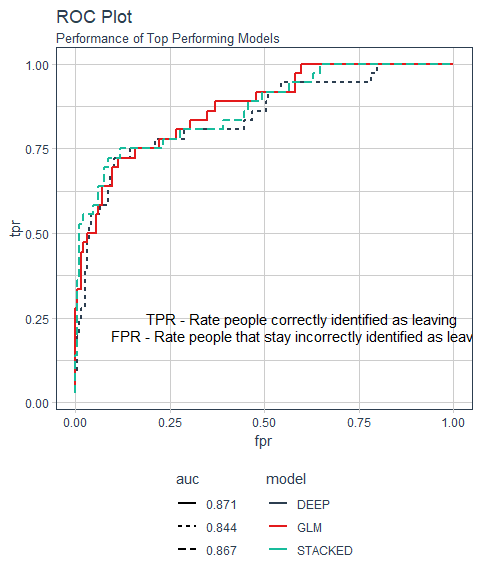
load\_model\_performance\_metrics <- function(path, test\_tbl){  
   
 model\_h2o <- h2o.loadModel(path)  
 perf\_h2o <- h2o.performance(model\_h2o, newdata = as.h2o(test\_tbl))  
 perf\_h2o %>% h2o.metric() %>% as.tibble() %>% mutate(auc = h2o.auc(perf\_h2o)) %>%   
 select(tpr, fpr, auc)}

model\_metrics\_tbl <- fs::dir\_info(path = "04\_Modeling/h2o\_models/") %>%   
 select(path) %>%   
 mutate(metrics = map(path, load\_model\_performance\_metrics, test\_tbl)) %>%   
 unnest()#it was a list now its flat

### ROC Plot

The ROC Plot compares the TPR (y-axis) against the FPR (x-axis). TPR is the rate which people are leaving correctly identified as leaving. FPR is the rate people that stay are incorrectly identified as leaving.

# Data prep for plotting; # recall numeric to factor must first convert to character  
model\_metrics\_tbl %>% mutate(  
 path = str\_split(path, pattern = "/", simplify = T)[, 3] %>% as\_factor(),  
 auc = auc %>% round(3) %>% as.character() %>% as\_factor()) %>%   
   
 ggplot(aes(x = fpr, y = tpr, color = path, linetype = auc)) + geom\_line(size = 1) + theme\_tq() + scale\_color\_tq()+  
 theme(legend.direction = "vertical") + labs(title ="ROC Plot", subtitle = "Performance of Top Performing Models") +  
 annotate(geom = "text", x = 0.6, y = 0.25, label = "TPR - Rate people correctly identified as leaving") +  
 annotate(geom = "text", x = 0.6, y = 0.20, label = "FPR - Rate people that stay incorrectly identified as leaving")

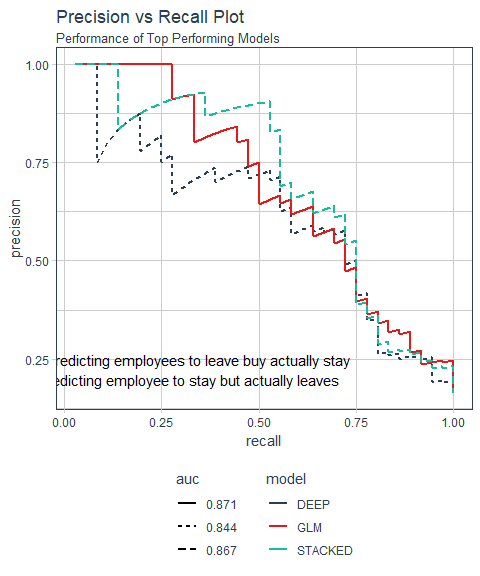


### Precision vs Recall Plot

Turn precision / recall trade off into performance. This is another multiple model evaluation.

load\_model\_performance\_metrics <- function(path, test\_tbl){  
   
 model\_h2o <- h2o.loadModel(path)  
 perf\_h2o <- h2o.performance(model\_h2o, newdata = as.h2o(test\_tbl))  
   
 perf\_h2o %>% h2o.metric() %>% as.tibble() %>% mutate(auc = h2o.auc(perf\_h2o)) %>%   
 select(tpr, fpr, auc, precision, recall)}

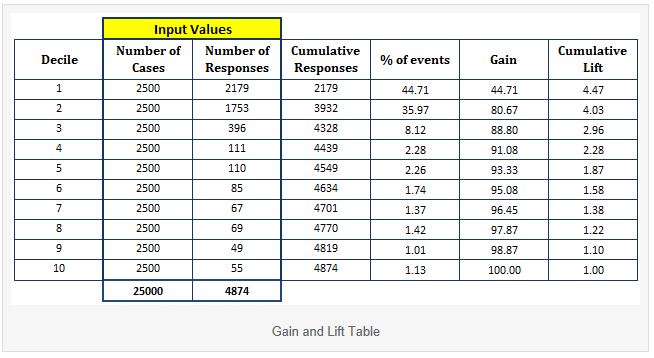
# Data prep for plotting; # recall numeric to factor must first convert to character  
model\_metrics\_tbl %>% mutate(  
 path = str\_split(path, pattern = "/", simplify = T)[, 3] %>% as\_factor(),  
 auc = auc %>% round(3) %>% as.character() %>% as\_factor()) %>%   
   
 ggplot(aes(x = recall, y = precision, color = path, linetype = auc)) + geom\_line(size = 1) + theme\_tq() + scale\_color\_tq()+  
 theme(legend.direction = "vertical") + labs(title ="Precision vs Recall Plot",   
 subtitle = "Performance of Top Performing Models") +  
 annotate(geom = "text", x = 0.6, y = 0.25, label = "Precision - Predicting employees to leave buy actually stay") +  
 annotate(geom = "text", x = 0.6, y = 0.20, label = "Recall - Predicting employee to stay but actually leaves")

 > Goal is to reduce the under-prediction i.e., employees predicted to stay but actually leave (Recall)

## Performance Charts For Business People

Gain and Lift charts are used to evaluate performance of classification model. They measure how much better one can expect to do with the predictive model comparing without a model. It’s a very popular metrics in marketing analytics. It’s not just restricted to marketing analysis. It can be used in other domains as well such as risk modeling, supply chain analytics etc. It also helps to find the best predictive model among multiple challenger models.

See <https://www.listendata.com/2014/08/excel-template-gain-and-lift-charts.html> - seems to follow the same process as the one below. The goal is to make a Gain-Lift Table:



1. Below is a score file in descending order by estimated probability. By ranking by class probability of Yes, we assess the models ability to detect someone that is leaving.

* Note the use of bind\_cols which is part of dplyr

ranked\_predictions\_tbl <- predictions\_tbl %>% bind\_cols(test\_tbl) %>% select(predict:Yes, Attrition) %>%   
 arrange(desc(Yes))  
ranked\_predictions\_tbl

## # A tibble: 220 x 4  
## predict No Yes Attrition  
## <fct> <dbl> <dbl> <fct>   
## 1 Yes 0.0921 0.908 Yes   
## 2 Yes 0.184 0.816 Yes   
## 3 Yes 0.198 0.802 Yes   
## 4 Yes 0.230 0.770 Yes   
## 5 Yes 0.279 0.721 Yes   
## 6 Yes 0.338 0.662 Yes   
## 7 Yes 0.338 0.662 Yes   
## 8 Yes 0.345 0.655 Yes   
## 9 Yes 0.365 0.635 Yes   
## 10 Yes 0.370 0.630 Yes   
## # ... with 210 more rows

### Gain & Lift Tables

Steps:

1. Split the ranked file into 10 sections (deciles). Grouping into cohorts of most likely groups to least likely groups is at the heart of gain/lift. When we do this, we can immediately show that if a candidate has a high probability of leaving, how likely they are to leave.
2. Number of observations in each decile: group = row\_number()
3. Number of actual events in each decile: responses = sum(Attrition == "Yes")

ranked\_predictions\_tbl %>% mutate(ntile = ntile(Yes, n = 10)) %>% group\_by(ntile) %>%   
 summarise(cases = n(), responses = sum(Attrition == "Yes")) %>% arrange(desc(ntile))

## # A tibble: 10 x 3  
## ntile cases responses  
## <int> <int> <int>  
## 1 10 22 17  
## 2 9 22 8  
## 3 8 22 2  
## 4 7 22 3  
## 5 6 22 2  
## 6 5 22 1  
## 7 4 22 3  
## 8 3 22 0  
## 9 2 22 0  
## 10 1 22 0

In ntile = 10, 17 people actually left out of 22 employees! 8 of 22 actually left in the next ntile.

* note the use of ntile also from dplyr. We simply group records into 10 groups. Note the 10th group has the highest values for Yes. Makes the continuous variable Yes into a discrete variable using the new groups called ntile.
* dpplyr::row\_numer is useful for calculations to simply make the row numbers of a tibble into a column

ranked\_predictions\_tbl %>% mutate(ntile = ntile(Yes, n = 10)) %>% group\_by(ntile) %>%   
 summarise(cases = n(), responses = sum(Attrition == "Yes")) %>% arrange(desc(ntile)) %>% mutate(group = row\_number())

## # A tibble: 10 x 4  
## ntile cases responses group  
## <int> <int> <int> <int>  
## 1 10 22 17 1  
## 2 9 22 8 2  
## 3 8 22 2 3  
## 4 7 22 3 4  
## 5 6 22 2 5  
## 6 5 22 1 6  
## 7 4 22 3 7  
## 8 3 22 0 8  
## 9 2 22 0 9  
## 10 1 22 0 10

Now group - the row numbers - now help. The first group has the highest number of employees that actually left. No longer the the ntile column.

Time to calculate the gain and lift.

1. *Cumulative responses*: Number of cumulative actual events in each decile: cumulative\_responses = cumsum(responses)
2. *Percentage responses*: pct\_responses = responses / sum(responses)
3. *Cumulative gain*: Cumulatively sum the pct\_responses. gain = cumsum(pct\_responses)
   * Think of this as what we gained by using the model. If we focused on the first 2 groups, we *gain* the ability to target ~70% of our quitters using the model. Remember, this is measuring employees that actual left and the model is identifying 70% of them!
   * gain is really the cumulative\_capture\_rate
4. A calculation is needed before we can measure lift. cumulative\_pct\_cases = cumsum(cases)/sum(cases). This is a simple calculation. Since we have 10 groups, this calculation will start at 0.100 and end with 1.00 step by 0.100 so: .1, .2, 3. . .
5. Calculate lift. Think lift as a multiplier between what we gained divided by what we expected to gain with no model. For example, if we focused on the first 2 decile groups, we gained the ability to target ~70% of our quitter but we expected to be able to target only 20% (2 deciles (cummulative\_pct\_cases) = 20%)
   * lift is just another way to show the gain
   * Below, lift = 4.7 simply means you are 4.7X more likely to identify quitters using the model

calculated\_gain\_lift\_tbl <- ranked\_predictions\_tbl %>% mutate(ntile = ntile(Yes, n = 10)) %>% group\_by(ntile) %>%   
 summarise(cases = n(), responses = sum(Attrition == "Yes")) %>% arrange(desc(ntile)) %>%   
 mutate(group = row\_number()) %>% select(group, cases, responses) %>%   
 mutate(cumulative\_responses = cumsum(responses), pct\_responses = responses / sum(responses),  
 gain = cumsum(pct\_responses),  
 # we gain ability to target 75% of quitter focused on 1st 3 groups  
 cumulative\_pct\_cases = cumsum(cases)/sum(cases), lift = gain/cumulative\_pct\_cases,  
 gain\_baseline = cumulative\_pct\_cases, lift\_baseline = gain\_baseline / cumulative\_pct\_cases)  
calculated\_gain\_lift\_tbl %>% glimpse()

## Observations: 10  
## Variables: 10  
## $ group <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10  
## $ cases <int> 22, 22, 22, 22, 22, 22, 22, 22, 22, 22  
## $ responses <int> 17, 8, 2, 3, 2, 1, 3, 0, 0, 0  
## $ cumulative\_responses <int> 17, 25, 27, 30, 32, 33, 36, 36, 36, 36  
## $ pct\_responses <dbl> 0.472, 0.222, 0.056, 0.083, 0.056, 0.028,...  
## $ gain <dbl> 0.47, 0.69, 0.75, 0.83, 0.89, 0.92, 1.00,...  
## $ cumulative\_pct\_cases <dbl> 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0...  
## $ lift <dbl> 4.7, 3.5, 2.5, 2.1, 1.8, 1.5, 1.4, 1.2, 1...  
## $ gain\_baseline <dbl> 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0...  
## $ lift\_baseline <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

Above, uncertain why gain\_baseline and lift\_baseline were added. (BTW, `lift\_baseline is always = 1.)

### h2o:gainsLift()

h2o.gainsLift streamline the manual gain-lift code above. h2o.performance() is the starting point for the H2O fain and lift function.

gain\_lift\_tbl <- perf\_h2o\_GLM %>% h2o.gainsLift() %>% as.tibble()  
  
gain\_lift\_tbl %>% glimpse()

## Observations: 16  
## Variables: 13  
## $ group <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...  
## $ cumulative\_data\_fraction <dbl> 0.014, 0.023, 0.032, 0.041, 0.050, 0....  
## $ lower\_threshold <dbl> 0.79583, 0.69841, 0.65794, 0.63159, 0...  
## $ lift <dbl> 6.11, 6.11, 6.11, 6.11, 3.06, 3.89, 2...  
## $ cumulative\_lift <dbl> 6.1, 6.1, 6.1, 6.1, 5.6, 4.7, 3.9, 3....  
## $ response\_rate <dbl> 1.000, 1.000, 1.000, 1.000, 0.500, 0....  
## $ score <dbl> 0.8419, 0.7452, 0.6622, 0.6449, 0.625...  
## $ cumulative\_response\_rate <dbl> 1.00, 1.00, 1.00, 1.00, 0.91, 0.77, 0...  
## $ cumulative\_score <dbl> 0.84, 0.80, 0.76, 0.74, 0.72, 0.64, 0...  
## $ capture\_rate <dbl> 0.083, 0.056, 0.056, 0.056, 0.028, 0....  
## $ cumulative\_capture\_rate <dbl> 0.083, 0.139, 0.194, 0.250, 0.278, 0....  
## $ gain <dbl> 511, 511, 511, 511, 206, 289, 122, 12...  
## $ cumulative\_gain <dbl> 511, 511, 511, 511, 456, 372, 289, 24...

* group = ntiles by 16: 220 leave /16 ntiles = 13.75 employees/group (manual calculation set ntile = 10)

Only a subset of the H2O data is needed:

gain\_lift\_tbl <- perf\_h2o\_GLM %>% h2o.gainsLift() %>% select(group, cumulative\_data\_fraction, cumulative\_capture\_rate, cumulative\_lift) %>% as.tibble()  
gain\_lift\_tbl

## # A tibble: 16 x 4  
## group cumulative\_data\_fraction cumulative\_capture\_rate cumulative\_lift  
## <int> <dbl> <dbl> <dbl>  
## 1 1 0.0136 0.0833 6.11  
## 2 2 0.0227 0.139 6.11  
## 3 3 0.0318 0.194 6.11  
## 4 4 0.0409 0.25 6.11  
## 5 5 0.05 0.278 5.56  
## 6 6 0.1 0.472 4.72  
## 7 7 0.15 0.583 3.89  
## 8 8 0.2 0.694 3.47  
## 9 9 0.3 0.75 2.5   
## 10 10 0.4 0.833 2.08  
## 11 11 0.5 0.889 1.78  
## 12 12 0.6 0.917 1.53  
## 13 13 0.7 1 1.43  
## 14 14 0.8 1 1.25  
## 15 15 0.9 1 1.11  
## 16 16 1 1 1

This is all that is needed to create out gain and lift plot.

### Gain Plot

* Use gather() to stack the data to use ggplot color, group aesthetics or facet\_wrap
* for the gain data, cumumalitive\_lift is not needed (but it is later for lift)

gain\_transformed\_tbl <- gain\_lift\_tbl %>%   
 select(group, cumulative\_data\_fraction, cumulative\_capture\_rate) %>% mutate(baseline = cumulative\_data\_fraction) %>%   
 rename(gain = cumulative\_capture\_rate) %>% gather(key = key, value = value, gain, baseline)  
gain\_transformed\_tbl

## # A tibble: 32 x 4  
## group cumulative\_data\_fraction key value  
## <int> <dbl> <chr> <dbl>  
## 1 1 0.0136 gain 0.0833  
## 2 2 0.0227 gain 0.139   
## 3 3 0.0318 gain 0.194   
## 4 4 0.0409 gain 0.25   
## 5 5 0.05 gain 0.278   
## 6 6 0.1 gain 0.472   
## 7 7 0.15 gain 0.583   
## 8 8 0.2 gain 0.694   
## 9 9 0.3 gain 0.75   
## 10 10 0.4 gain 0.833   
## # ... with 22 more rows

gain\_transformed\_tbl %>% ggplot(aes(x = cumulative\_data\_fraction, y = value, color = key)) +  
 geom\_line(size = 1.5) + theme\_tq() + scale\_color\_tq() +  
 labs(title = "Gain Chart", x = "Cumulative Data Fraction", y = "Gain")



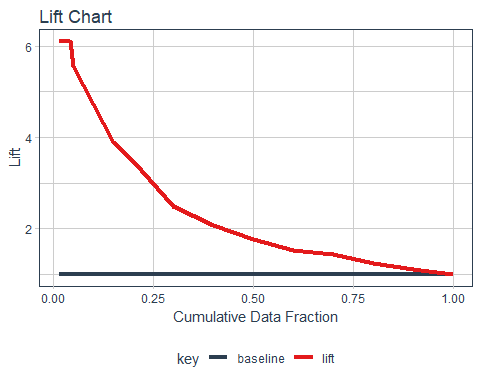
### Lift Chart

* Do not need cumulative\_capture\_rate
* lift-Baseline always = 1

lift\_transformed\_tbl <- gain\_lift\_tbl %>%   
 select(group, cumulative\_data\_fraction, cumulative\_lift) %>% mutate(baseline = 1) %>%   
 rename(lift = cumulative\_lift) %>% gather(key = key, value = value, lift, baseline)  
lift\_transformed\_tbl

## # A tibble: 32 x 4  
## group cumulative\_data\_fraction key value  
## <int> <dbl> <chr> <dbl>  
## 1 1 0.0136 lift 6.11  
## 2 2 0.0227 lift 6.11  
## 3 3 0.0318 lift 6.11  
## 4 4 0.0409 lift 6.11  
## 5 5 0.05 lift 5.56  
## 6 6 0.1 lift 4.72  
## 7 7 0.15 lift 3.89  
## 8 8 0.2 lift 3.47  
## 9 9 0.3 lift 2.5   
## 10 10 0.4 lift 2.08  
## # ... with 22 more rows

lift\_transformed\_tbl %>% ggplot(aes(x = cumulative\_data\_fraction, y = value, color = key)) +  
 geom\_line(size = 1.5) + theme\_tq() + scale\_color\_tq() +  
 labs(title = "Lift Chart", x = "Cumulative Data Fraction", y = "Lift")



### Explain Gain/Lift Plots

NOTE: The plots used for explanation differ from the calculated plots in the RMD slightly.

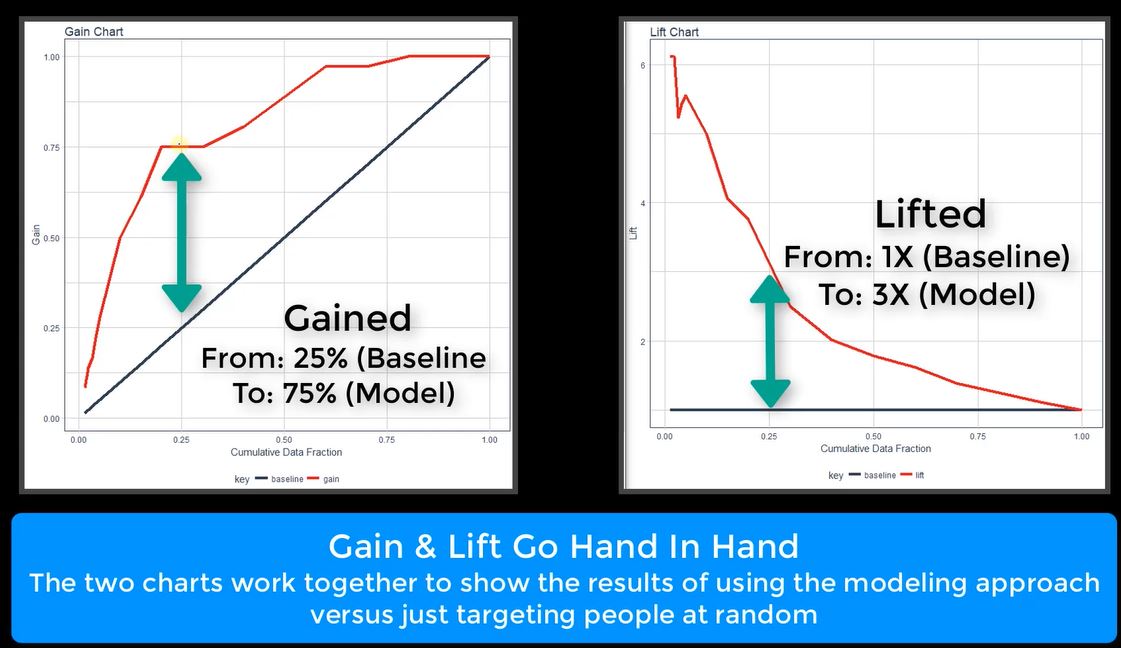
Below, doing nothing - not using the model - 25% of employees would be targeted to leave. With the model, 75% employees would be captured - 3X more!

Strategically targeting 25% of *high probability* customers should yield 75% of potential positive responses!

Lift is a multiplier: How many positive responses would you get at random X Lift = How many would you get with the model? 3X shown below.

Example: How does lift apply to attrition? - Providing stock options strategically to high performers that are at high risk. Could strategically focus on those in the high risk category that are working overtime. Stock options would cost overall less when compared to offering stock options randomly. Reduces cost by 33%!

Always explain in terms of more customers, less churn, better quality, reduced lead times, or anything else executives care about.



## Model Performance Comparison Dashboard

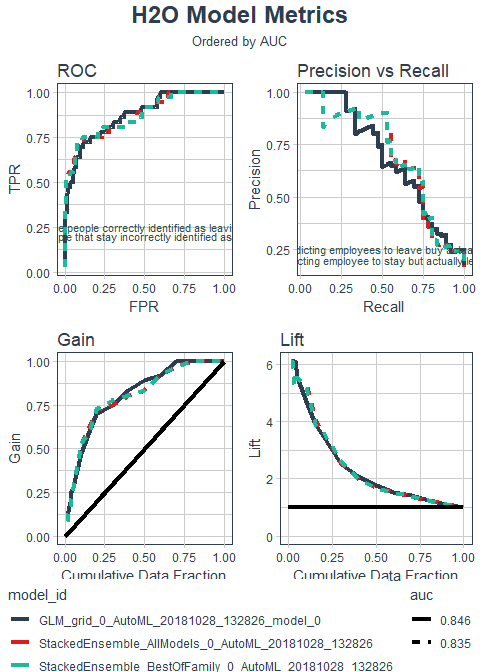
plot\_h2o\_performance brings together the model and business plots into one function: ROC, Precision vs Recall, Gain and Lift plots.

* order\_by <- tolower(order\_by[[1]]) simply selects the first in the list in argument order\_by = c("auc", "logloss"). If user does not select one, auc will be used as default.
* rlang::sym(order\_by) converts a string stored within a variable to a column name (symbol) that is unevaluated so we can use it later using !! (bang bang)
* h2o.getModel gets a model from the current session. loadModel retrieves a model that was previously saved to disk.
* `. *Use this pattern!* {#firstMap}
* fct\_reorder(!! order\_by\_expr, .desc = ifelse(order\_by =="auc", TRUE, FALSE)) is cool. If the selected metric is auc, the code will order highest to lowest. If the metric is logloss, it sorts the data lowest to highest. *Very cool!*
* fct\_reorder requires numeric value
* Combine plots using cowplot
* glue is used - a better method of pasting strings than paste0.
  + Uses {} to mix code with text. Uses only one string rather than many strings - just easier than paste0

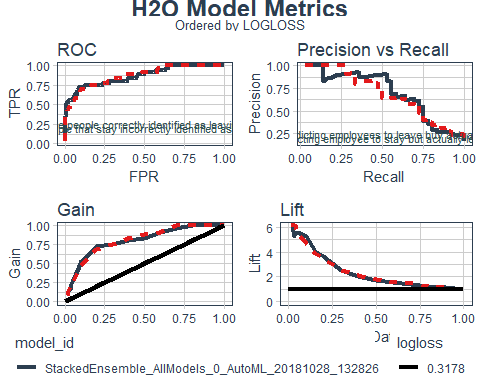
plot\_h2o\_performance <- function(h2o\_leaderboard, newdata, order\_by = c("auc", "logloss"), max\_models = 3, size = 1.5){  
# Inputs  
 leaderboard\_tbl <- h2o\_leaderboard %>% as.tibble() %>% slice(1:max\_models)  
 newdata\_tbl <- newdata %>% as.tibble()  
 order\_by <- tolower(order\_by[[1]])  
 order\_by\_expr <- rlang::sym(order\_by)  
 h2o.no\_progress()  
   
# 1. Model Metrics  
 get\_model\_performance\_metrics <- function(model\_id, test\_tbl){  
 model\_h2o <- h2o.getModel(model\_id)  
 perf\_h2o <- h2o.performance(model\_h2o, newdata = as.h2o(test\_tbl))  
 perf\_h2o %>% h2o.metric() %>% as.tibble() %>% select(threshold, tpr, fpr,  
 precision, recall)}  
   
 model\_metrics\_tbl <- leaderboard\_tbl %>%   
 mutate(metrics = map(model\_id, get\_model\_performance\_metrics, newdata\_tbl)) %>%   
 unnest() %>%   
 mutate(  
 model\_id = as\_factor(model\_id) %>%   
 fct\_reorder(!! order\_by\_expr, .desc = ifelse(order\_by =="auc", TRUE, FALSE)),  
   
 auc = auc %>% round(3) %>% as.character() %>% as\_factor() %>%   
 fct\_reorder(as.numeric(model\_id)),  
   
 logloss = logloss %>% round(4) %>% as.character() %>% as\_factor() %>%   
 fct\_reorder(as.numeric(model\_id)))  
   
 # 1A. ROC Plot  
 p1 <- model\_metrics\_tbl %>% ggplot(aes\_string(x = "fpr",   
 y = "tpr", color = "model\_id", linetype = order\_by)) + geom\_line(size = size) +  
 theme\_tq() + scale\_color\_tq() + labs(title = "ROC", x = "FPR", y = "TPR") +   
 theme(legend.direction = "vertical") +  
 annotate(geom = "text", x = 0.4, y = 0.25, size = 3, color = "DarkSlateGrey",  
 label = "TPR - Rate people correctly identified as leaving") +  
 annotate(geom = "text", x = 0.4, y = 0.20, size = 3, color = "DarkSlateGrey",  
 label = "FPR - Rate people that stay incorrectly identified as leaving")  
   
 # 1B. Precision vs Recall Plot  
 p2 <- model\_metrics\_tbl %>%   
 ggplot(aes\_string(x = "recall", y = "precision", color = "model\_id", linetype = order\_by)) +   
 geom\_line(size = size) +  
 theme\_tq() + scale\_color\_tq() + labs(title = "Precision vs Recall",   
 x = "Recall", y = "Precision") +  
 theme(legend.position = "none") +   
 annotate(geom = "text", x = 0.4, y = 0.25, size = 3, color = "DarkSlateGrey",  
 label = "Precision - Predicting employees to leave buy actually stay") +  
 annotate(geom = "text", x = 0.4, y = 0.20, size = 3, color = "DarkSlateGrey",  
 label = "Recall - Predicting employee to stay but actually leaves")  
  
# 2. Gain / Lift  
 get\_gain\_lift <- function(model\_id, test\_tbl){  
 model\_h2o <- h2o.getModel(model\_id)  
 perf\_h2o <- h2o.performance(model\_h2o, newdata = as.h2o(test\_tbl))  
 perf\_h2o %>% h2o.gainsLift() %>% as.tibble() %>%   
 select(group, cumulative\_data\_fraction, cumulative\_capture\_rate, cumulative\_lift)}   
   
 gain\_lift\_tbl <- leaderboard\_tbl %>%   
 mutate(metrics = map(model\_id, get\_gain\_lift, newdata\_tbl)) %>% unnest() %>%   
 mutate(  
 model\_id = as\_factor(model\_id) %>%   
 fct\_reorder(!! order\_by\_expr, .desc = ifelse(order\_by == "auc", TRUE, FALSE)),  
   
 auc = auc %>% round(3) %>% as.character() %>% as\_factor() %>%  
 fct\_reorder(as.numeric(model\_id)),  
   
 logloss = logloss %>% round(4) %>% as.character() %>% as\_factor() %>%   
 fct\_reorder(as.numeric(model\_id))) %>%   
   
 rename(gain = cumulative\_capture\_rate, lift = cumulative\_lift)  
 # 2A. Gain Plot  
   
 p3 <- gain\_lift\_tbl %>% ggplot(aes\_string(x = "cumulative\_data\_fraction", y = "gain",   
 color= "model\_id", linetype = order\_by)) + geom\_line(size = size) +  
 geom\_segment(x = 0, y = 0, xend = 1, yend = 1, color = "black", size = size) + theme\_tq() + scale\_color\_tq() +  
 expand\_limits(x = c(0, 1), y = c(0, 1)) + labs(title = "Gain", x = "Cumulative Data Fraction", y = "Gain") +  
 theme(legend.position = "none")  
   
 # 2B. Lift Plot  
 p4 <- gain\_lift\_tbl %>% ggplot(aes\_string(x = "cumulative\_data\_fraction", y = "lift",   
 color= "model\_id", linetype = order\_by)) + geom\_line(size = size) +  
 geom\_segment(x = 0, y = 1, xend = 1, yend = 1, color = "black", size = size) + theme\_tq() + scale\_color\_tq() +  
 expand\_limits(x = c(0, 1), y = c(0, 1)) + labs(title = "Lift", x = "Cumulative Data Fraction", y = "Lift") +  
 theme(legend.position = "none")  
# Cow Plot   
 p\_legend <- cowplot::get\_legend(p1)  
 p1 <- p1 + theme(legend.position = "none")  
   
 p <- cowplot::plot\_grid(p1, p2, p3, p4, ncol=2)  
   
 p\_title <- ggdraw() + draw\_label("H2O Model Metrics", size = 18, fontface = "bold",  
 colour = palette\_light()[[1]])#MUST use COLOUR  
 p\_subtitle <- ggdraw() + draw\_label(glue("Ordered by {toupper(order\_by)}"), size = 10,   
 colour = palette\_light()[[1]])  
 ret <- plot\_grid(p\_title, p\_subtitle, p, p\_legend, ncol = 1, rel\_heights = c(0.05, 0.05, 1, 0.05 \* max\_models))  
 h2o.show\_progress()  
 return(ret)}

Test the new function.

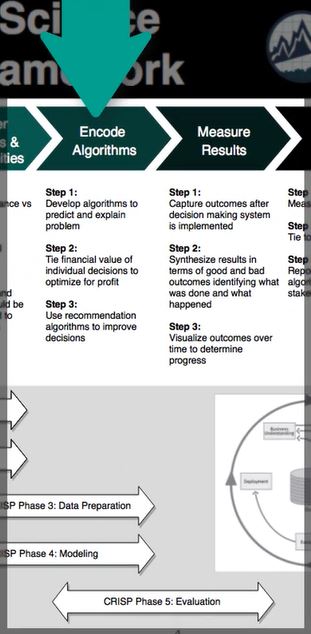
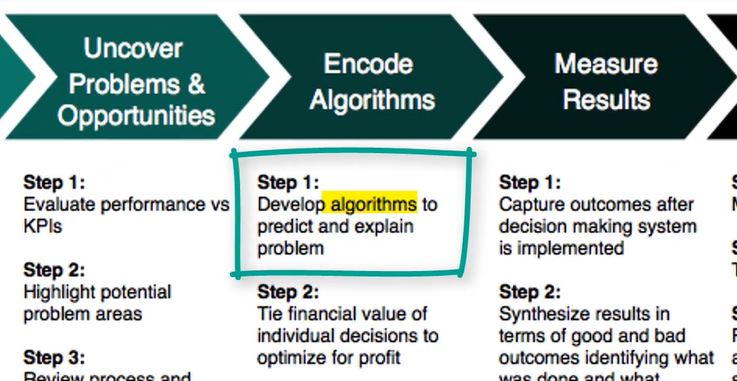
automl\_models\_h2o@leaderboard %>%   
 plot\_h2o\_performance(newdata = test\_tbl, order\_by = "auc", max\_models = 3)



automl\_models\_h2o@leaderboard %>%   
 plot\_h2o\_performance(newdata = test\_tbl, order\_by = "logloss", max\_models = 2)



# Explaining Models With LIME

Take the developed H2O models and show how to explain localized prediction results using a special technique called LIME (Local Interpretable Model-Agnostic Explanations).

* How to make an explainer using the lime() function for single and multiple employees
* How to make an explanation using the explain() function for single and multiple employees
* How to visualize model explanations using plot\_features() (one or a small number of observations) and plot\_explanations() (for many observations)

This section answers the question WHY something happens - at the record level.

## Feature Explanation With LIME

* The primary LIME documentation is found here<http://uc-r.github.io/lime>)
* [Understanding lime Viginette is good](https://cran.r-project.org/web/packages/lime/vignettes/Understanding_lime.html) This includes a good multiclass classification example. Also has a textual example.

### Investigating Predictions

This is a preparatory step before getting into the weeds with lime. This should be familiar and serves as a review.

automl\_leader %>% h2o.predict(newdata = as.h2o(test\_tbl)) %>% as.tibble() %>% head()

## # A tibble: 6 x 3  
## predict No Yes  
## <fct> <dbl> <dbl>  
## 1 No 0.964 0.0359  
## 2 No 0.891 0.109   
## 3 No 0.967 0.0330  
## 4 No 0.953 0.0471  
## 5 Yes 0.210 0.790   
## 6 No 0.937 0.0631

Focus on record 5 since it is predicted to turnover with a high probability. Get more information about this person so we know what actually happened and also get the employees ID so we know who we are talking about.

predictions\_tbl <- automl\_leader %>% h2o.predict(newdata = as.h2o(test\_tbl)) %>% as.tibble() %>%   
 bind\_cols(test\_tbl %>% select(Attrition, EmployeeNumber))  
predictions\_tbl %>% head()

## # A tibble: 6 x 5  
## predict No Yes Attrition EmployeeNumber  
## <fct> <dbl> <dbl> <fct> <dbl>  
## 1 No 0.964 0.0359 No 228  
## 2 No 0.891 0.109 No 1278  
## 3 No 0.967 0.0330 No 1250  
## 4 No 0.953 0.0471 No 2065  
## 5 Yes 0.210 0.790 Yes 1767  
## 6 No 0.937 0.0631 No 1308

We now know the employee actually did turnover.

Now find the detail about employee 1767 that did turnover as predicted.

test\_tbl %>% slice(5) %>% glimpse()

## Observations: 1  
## Variables: 32  
## $ Age <dbl> 43  
## $ Attrition <fct> Yes  
## $ BusinessTravel <fct> Travel\_Frequently  
## $ DailyRate <dbl> 807  
## $ Department <fct> Research & Development  
## $ DistanceFromHome <dbl> 17  
## $ Education <fct> Bachelor  
## $ EducationField <fct> Technical Degree  
## $ EmployeeNumber <dbl> 1767  
## $ EnvironmentSatisfaction <fct> High  
## $ Gender <fct> Male  
## $ HourlyRate <dbl> 38  
## $ JobInvolvement <fct> Medium  
## $ JobLevel <fct> 1  
## $ JobRole <fct> Research Scientist  
## $ JobSatisfaction <fct> High  
## $ MaritalStatus <fct> Married  
## $ MonthlyIncome <dbl> 2437  
## $ MonthlyRate <dbl> 15587  
## $ NumCompaniesWorked <dbl> 9  
## $ OverTime <fct> Yes  
## $ PercentSalaryHike <dbl> 16  
## $ PerformanceRating <fct> Excellent  
## $ RelationshipSatisfaction <fct> Very High  
## $ StockOptionLevel <fct> 1  
## $ TotalWorkingYears <dbl> 6  
## $ TrainingTimesLastYear <dbl> 4  
## $ WorkLifeBalance <fct> Better  
## $ YearsAtCompany <dbl> 1  
## $ YearsInCurrentRole <dbl> 0  
## $ YearsSinceLastPromotion <dbl> 0  
## $ YearsWithCurrManager <dbl> 0

It appears some factors support attrition and some do not. If we want to know why he turned over, use lime.

### Lime For Single Explanation

LIME is used to determine which features contribute to the prediction (and by how much) for a single (local) observation

* 2 steps:  
  1.Build **explainer** with lime()
  1. Create **Explanation** with explain()

h2o, keras and caret have been integrated into lime(). For other packages, use special functions model\_type() and predict\_model. <http://uc-r.github.io/lime>

1. To build an explainer, use the ML model and feature distributions (bins) for the training data.

* drop Attrition for the H2O model
* bin\_continuous makes it easy to detect what causes a continuous feature to have a high feature weight in the lime explanation
* quantile\_bins tells how to distribute observations within the bins. If TRUE, cuts will be evenly distributed within each bin
* N\_bins: 4-5 bins is typically sufficient. If N\_bins = 4,bin\_cuts` will have 5 cuts. The bins are then the observations that fall in between the cuts.

explainer <- train\_tbl %>% select(-Attrition) %>%   
 lime(model = automl\_leader, bin\_continuous = TRUE, N\_bins = 4, quantile\_bins = TRUE)  
#explainer

The output is voluminous and omitted to save pages of printing! It includes how continuous variables are cut. This is important information.

explainer provides much data - be careful before printing.

1. Create explanation with explain

LIME 6 steps for explanation:

1. Given an observation, permute it to create replicated feature data with slight value modifications.
2. Compute similarity distance measure between original observation and permuted observations.
3. Apply selected machine learning model to predict outcomes of permuted data.
4. Select the m number pf features to best describe predicted outcomes.
5. Fit a simple model to the permuted data explaining the complex model outcome with m number features from the permuted data by its similarity to the original observation.
6. Use the resulting feature weights to explain local behavior.

* n\_labels: binary classification therefore just one label
* n\_features: number of important features to return to explain the record prediction
* n\_permutations: the more test/evaluations, the better the explanations (typically). Lime will create 5000 localized linear models for evaluation.
* kernel\_width: Affects the LIME linear model (R^2) and therefore should be tuned to ensure you get the best explanations - IMPORTANT
* Attrition is removed because the data argument must match the format the model requires to predict. Since H2O requires x to be without the target variable, it must be removed below.
* Use lime::explain because explain is a common function used in other packages.

explanation <- test\_tbl %>% slice(5) %>% select(-Attrition) %>%   
 lime::explain(explainer = explainer, n\_labels = 1, n\_features = 8, n\_permutations = 5000,   
 kernel\_width = 0.5)  
explanation

## model\_type case label label\_prob model\_r2 model\_intercept  
## 1 classification 1 Yes 0.79 0.321 0.132  
## 2 classification 1 Yes 0.79 0.321 0.132  
## 3 classification 1 Yes 0.79 0.321 0.132  
## 4 classification 1 Yes 0.79 0.321 0.132  
## 5 classification 1 Yes 0.79 0.321 0.132  
## 6 classification 1 Yes 0.79 0.321 0.132  
## 7 classification 1 Yes 0.79 0.321 0.132  
## 8 classification 1 Yes 0.79 0.321 0.132  
## model\_prediction feature feature\_value feature\_weight  
## 1 0.436 OverTime 2 0.1877  
## 2 0.436 JobLevel 1 0.0917  
## 3 0.436 NumCompaniesWorked 9 0.0949  
## 4 0.436 StockOptionLevel 2 -0.0822  
## 5 0.436 BusinessTravel 3 0.0706  
## 6 0.436 DistanceFromHome 17 0.0703  
## 7 0.436 YearsAtCompany 1 -0.0626  
## 8 0.436 Department 2 -0.0661  
## feature\_desc  
## 1 OverTime = Yes  
## 2 JobLevel = 1  
## 3 4 < NumCompaniesWorked  
## 4 StockOptionLevel = 1  
## 5 BusinessTravel = Travel\_Frequently  
## 6 13.8 < DistanceFromHome  
## 7 YearsAtCompany <= 3  
## 8 Department = Research & Development  
## data  
## 1 43, 3, 807, 2, 17, 3, 6, 1767, 3, 2, 38, 2, 1, 7, 3, 2, 2437, 15587, 9, 2, 16, 3, 4, 2, 6, 4, 3, 1, 0, 0, 0  
## 2 43, 3, 807, 2, 17, 3, 6, 1767, 3, 2, 38, 2, 1, 7, 3, 2, 2437, 15587, 9, 2, 16, 3, 4, 2, 6, 4, 3, 1, 0, 0, 0  
## 3 43, 3, 807, 2, 17, 3, 6, 1767, 3, 2, 38, 2, 1, 7, 3, 2, 2437, 15587, 9, 2, 16, 3, 4, 2, 6, 4, 3, 1, 0, 0, 0  
## 4 43, 3, 807, 2, 17, 3, 6, 1767, 3, 2, 38, 2, 1, 7, 3, 2, 2437, 15587, 9, 2, 16, 3, 4, 2, 6, 4, 3, 1, 0, 0, 0  
## 5 43, 3, 807, 2, 17, 3, 6, 1767, 3, 2, 38, 2, 1, 7, 3, 2, 2437, 15587, 9, 2, 16, 3, 4, 2, 6, 4, 3, 1, 0, 0, 0  
## 6 43, 3, 807, 2, 17, 3, 6, 1767, 3, 2, 38, 2, 1, 7, 3, 2, 2437, 15587, 9, 2, 16, 3, 4, 2, 6, 4, 3, 1, 0, 0, 0  
## 7 43, 3, 807, 2, 17, 3, 6, 1767, 3, 2, 38, 2, 1, 7, 3, 2, 2437, 15587, 9, 2, 16, 3, 4, 2, 6, 4, 3, 1, 0, 0, 0  
## 8 43, 3, 807, 2, 17, 3, 6, 1767, 3, 2, 38, 2, 1, 7, 3, 2, 2437, 15587, 9, 2, 16, 3, 4, 2, 6, 4, 3, 1, 0, 0, 0  
## prediction  
## 1 0.21, 0.79  
## 2 0.21, 0.79  
## 3 0.21, 0.79  
## 4 0.21, 0.79  
## 5 0.21, 0.79  
## 6 0.21, 0.79  
## 7 0.21, 0.79  
## 8 0.21, 0.79

explanation$model\_r2[1]

## [1] 0.321

Want R^2 to be as high as possible - optimize by adjusting kernel\_width

In this case, kernel\_width = 0.5 works relatively well. Trying 0.2 - 1.5 : 0.2 performed best

explanation <- test\_tbl %>% slice(5) %>% select(-Attrition) %>%   
 lime::explain(explainer = explainer, n\_labels = 1, n\_features = 8,   
 n\_permutations = 5000, kernel\_width = 0.2)  
explanation$model\_r2[1]

## [1] 0.31

explanation <- test\_tbl %>% slice(5) %>% select(-Attrition) %>%   
 lime::explain(explainer = explainer, n\_labels = 1, n\_features = 8,   
 n\_permutations = 5000, kernel\_width = 1.0)  
explanation$model\_r2[1]

## [1] 0.304

Explore other tuning parameters

Note we selected 8 features in the explainer so there are 8 records below.

explanation %>% as.tibble() %>% select(feature:prediction)

## # A tibble: 8 x 6  
## feature feature\_value feature\_weight feature\_desc data prediction  
## <chr> <dbl> <dbl> <chr> <lis> <list>   
## 1 OverTime 2 0.182 OverTime = Yes <lis~ <list [2]>  
## 2 NumCompa~ 9 0.0928 4 < NumCompanie~ <lis~ <list [2]>  
## 3 JobLevel 1 0.0793 JobLevel = 1 <lis~ <list [2]>  
## 4 StockOpt~ 2 -0.0755 StockOptionLeve~ <lis~ <list [2]>  
## 5 MonthlyI~ 2437 0.0750 MonthlyIncome <~ <lis~ <list [2]>  
## 6 YearsAtC~ 1 -0.0692 YearsAtCompany ~ <lis~ <list [2]>  
## 7 Distance~ 17 0.0663 13.8 < Distance~ <lis~ <list [2]>  
## 8 Business~ 3 0.0627 BusinessTravel ~ <lis~ <list [2]>

* feature\_value is the actual value used. Fro example, this record indicates 9 companies the employee has worked. There are 17 miles between home and work.
* feature\_weight indicates the importance. +/- indicates support (positively correlated) or contradict.
* Positive weights support the positive binary classification (Yes or attrition in this case)

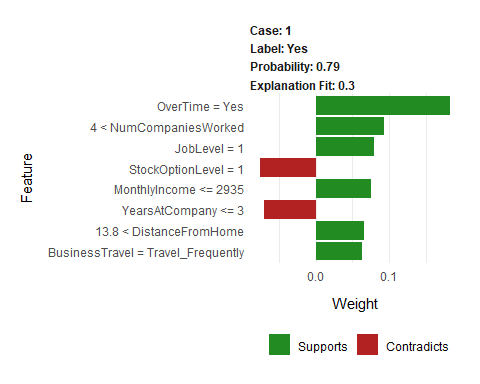
kernel\_width - R^2 values: 0.3: 0.303 0.4:0.294; 1.0:0.309; 1.5:0.294

In this case (values may vary in RMD), kernel\_width = 1 produced the highest R^2 value.

### Visualizing A Single Explanation: plot\_features()

* ncol = 1 since we only have one record to examine - at this time.
* note there are 3 red bars - the same number of negative feature\_weights in the table above.
* Note 4 < NumCompaniesWorked is a label from continuous binning performed earlier
* Red contradict features represent those features that encourage an employee to stay (supports the NO classification)
* GREEN outweigh the RED
* Explanation Fit is the R^2 value

plot\_features(explanation = explanation, ncol = 1)



### Visualizing Multiple Explanations: plot\_explanations()

Modify the slice function for 20 observations by modifying slice.

explanation <- test\_tbl %>% slice(1:20) %>% select(-Attrition) %>%   
 lime::explain(explainer = explainer, n\_labels = 1, n\_features = 8,  
 n\_permutations = 5000, kernel\_width = 0.2)  
  
explanation %>% as.tibble()

## # A tibble: 160 x 13  
## model\_type case label label\_prob model\_r2 model\_intercept  
## <chr> <chr> <chr> <dbl> <dbl> <dbl>  
## 1 classific~ 1 No 0.964 0.226 0.680  
## 2 classific~ 1 No 0.964 0.226 0.680  
## 3 classific~ 1 No 0.964 0.226 0.680  
## 4 classific~ 1 No 0.964 0.226 0.680  
## 5 classific~ 1 No 0.964 0.226 0.680  
## 6 classific~ 1 No 0.964 0.226 0.680  
## 7 classific~ 1 No 0.964 0.226 0.680  
## 8 classific~ 1 No 0.964 0.226 0.680  
## 9 classific~ 2 No 0.891 0.370 0.792  
## 10 classific~ 2 No 0.891 0.370 0.792  
## # ... with 150 more rows, and 7 more variables: model\_prediction <dbl>,  
## # feature <chr>, feature\_value <dbl>, feature\_weight <dbl>,  
## # feature\_desc <chr>, data <list>, prediction <list>

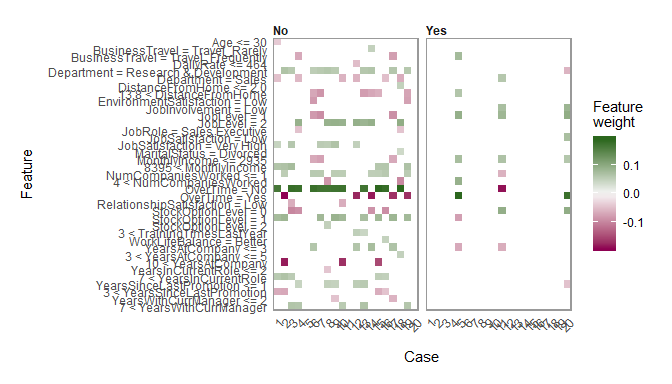
range(explanation$model\_r2)

## [1] 0.226 0.370

#plot\_features(explanation = explanation, ncol = 4)

Too much information. The plot is unreadable. Use another method - plot\_explanations

plot\_explanations(explanation = explanation)

 - The No panel indicates the features that support or contradict attrition. Even when there is support for attrition, other factors balance the employee’s experience. - 5, 11, 20 are likely to leave since they are in the Yes panel. - Overtime = Yes is dark red on the No column contradicting a NO response - If use more than 20 cases, it becomes challenging to decipher. The table view might be easier in those cases. Can always dplyr::summarize on the case number to group individual employees

### Recreating plot\_features() & plot\_explanations()

Extend the Lime plots with `theme\_tq.

Single case:{#plotlimefeaturetq}

### Aside: Glue

Expressions enclosed by braces will be evaluated as R code. Long strings are broken by line and concatenated together. Leading white space and blank lines from the ﬁrst and last lines are automatically trimmed. Example:

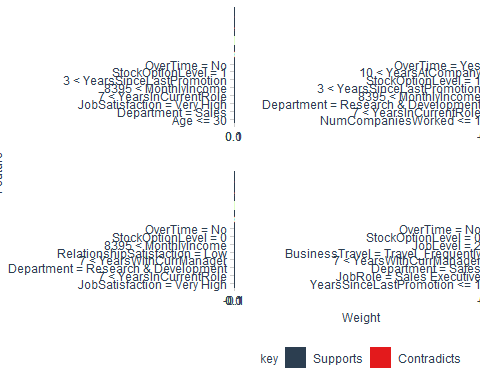
name <- "Fred"; age <- 50; anniversary <- as.Date("1991-10-12")   
glue('My name is {name},', 'my age next year is {age + 1},', 'my anniversary is {format(anniversary, "%A, %B %d, %Y")}.')

### plot\_features\_tq Function

library(glue)# Helps with text  
  
plot\_features\_tq <- function(explanation, ncol) {  
 data\_transformed <- explanation %>% as.tibble() %>%  
 mutate(  
 feature\_desc = as\_factor(feature\_desc) %>%   
 fct\_reorder(abs(feature\_weight), .desc = FALSE),  
 key = ifelse(feature\_weight > 0, "Supports", "Contradicts") %>%   
 fct\_relevel("Supports"),  
 case\_text = glue("Case: {case}"),  
 label\_text = glue("Label: {label}"),  
 prob\_text = glue("Probability: {round(label\_prob, 2)}"),  
 r2\_text = glue("Explanation Fit: {model\_r2 %>% round(2)}")) %>%  
 select(feature\_desc, feature\_weight, key, case\_text:r2\_text)  
   
   
 data\_transformed %>% ggplot(aes(feature\_desc, feature\_weight, fill = key)) +  
 geom\_bar(stat = "identity") + coord\_flip() + theme\_tq() + scale\_fill\_tq() +  
 labs(y = "Weight", x = "Feature") + theme(title = element\_text(size = 9)) +  
 facet\_wrap(~ case\_text + label\_text + prob\_text + r2\_text,  
 ncol = ncol, scales = "free")}

New function example:

explanation %>% filter(case %in% 1:4) %>% plot\_features\_tq(ncol=2)



### Multiple cases:

* mutate(order = order\_1 \* 1000 + order\_2) %>% A trick to get one order that preserves the hierarchy of feature (like Monthly\_Income) and feature\_value (like 9419) in the explanation tibble data.
  + rank simply ranks values from lowest to highest

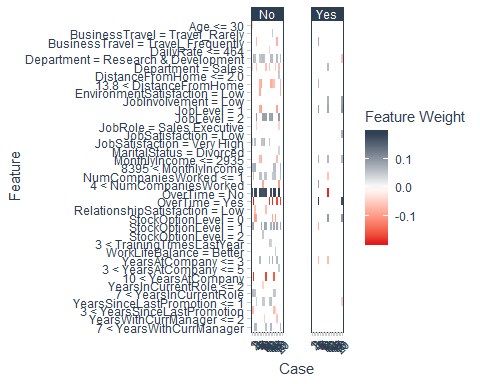
plot\_explanations\_tq <- function(explanation) {  
   
 data\_transformed <- explanation %>% as.tibble() %>%  
 mutate(case = as\_factor(case), order\_1 = rank(feature)) %>%   
 group\_by(feature) %>% mutate(order\_2 = rank(feature\_value)) %>%  
 ungroup() %>% mutate(order = order\_1 \* 1000 + order\_2) %>%   
 mutate(feature\_desc = as.factor(feature\_desc) %>% fct\_reorder(order, .desc = T)) %>%  
 select(case, feature\_desc, feature\_weight, label)  
   
 data\_transformed %>% ggplot(aes(case, feature\_desc)) +  
 geom\_tile(aes(fill = feature\_weight)) + facet\_wrap(~ label) + theme\_tq() +  
 scale\_fill\_gradient2(low = palette\_light()[[2]], mid = "white", high = palette\_light()[[1]]) +  
 theme(panel.grid = element\_blank(), legend.position = "right",  
 axis.text.x = element\_text(angle = 45, hjust = 1, vjust = 1)) +  
 labs(y = "Feature", x = "Case", fill = glue("Feature Weight"))}

Note glue in labs statement above. With the added return, it stacks Feature and Weight label

plot\_explanations(explanation)



plot\_explanations\_tq(explanation)



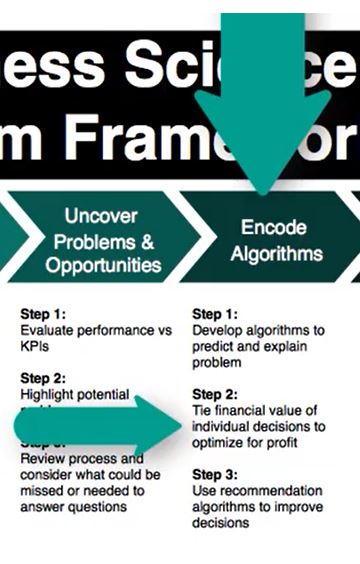
# Calculating Expected ROI Of Policy Change

This teaches you how to add value to organizations - by providing ROI-Driven Data Science! This is split in two:

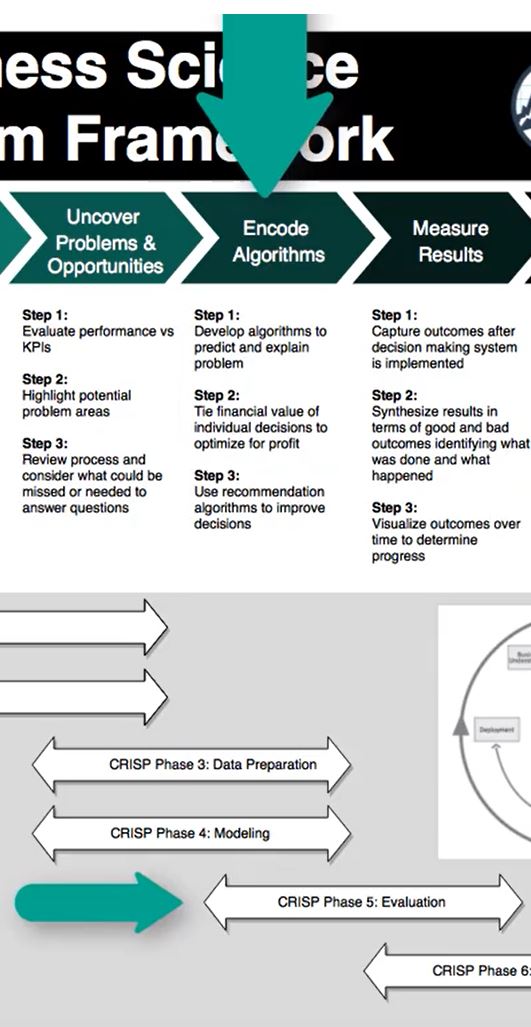
1. Covers calculating expected savings of a policy change (really multiple policy changes)
   * learn how to calculate a simple policy change - Implementing a No Overtime Policy. You will calculate the expected savings comparing to the current policy of allowing overtime. Next, you will develop a more sophisticated policy change - Implementing a Targeted Overtime Policy - wherein you target high-flight risk employees leaving the lower risk employees to work overtime as they please. We’ll teach you everything you need to know about the Expected Value Framework so you can begin to implement this for ANY BINARY CLASSIFICATION MODEL. This includes Customer Churn, Targeted Advertisements (think mailers), and more!
2. Take targeted overtime policy and expands it to cover threshold optimization and sensitivity analysis, two of the most important topics in Data Science for Business!
   * Perform two advanced analyses that are the most critical:
     + Threshold Optimization - A method used to maximize expected saving via iteratively calculating savings at various thresholds
     + Sensitivity Analysis - A method used to investigate how sensitive the expected savings is to various parameter values that were created based on assumptions

## BSPF Update

Starting on Step 2 of the BSPF Encode Algorithms Step:



This marks the beginning of the CRISP-DM Phase 5: Evaluation.



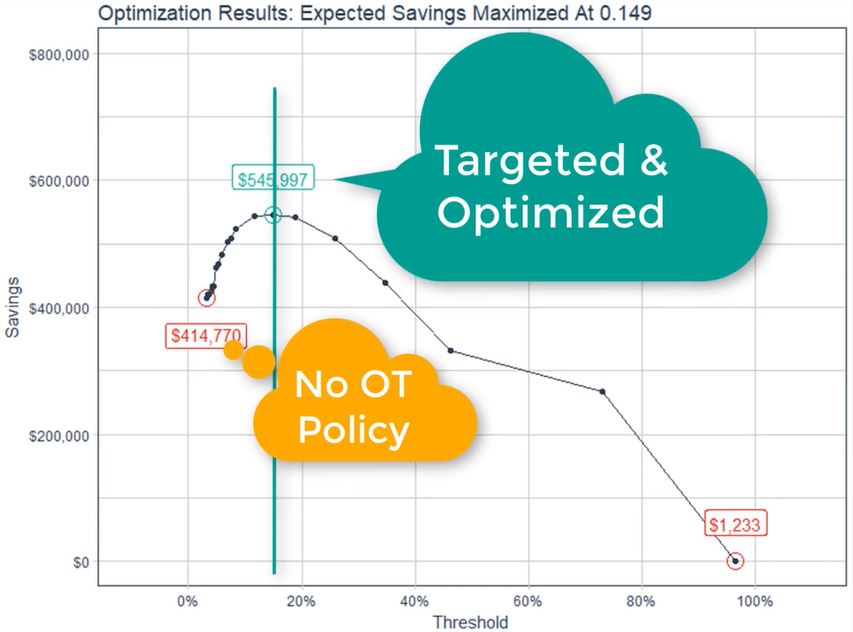
## Expected Value Framework

Using ROI, there will be two solutions are developed:

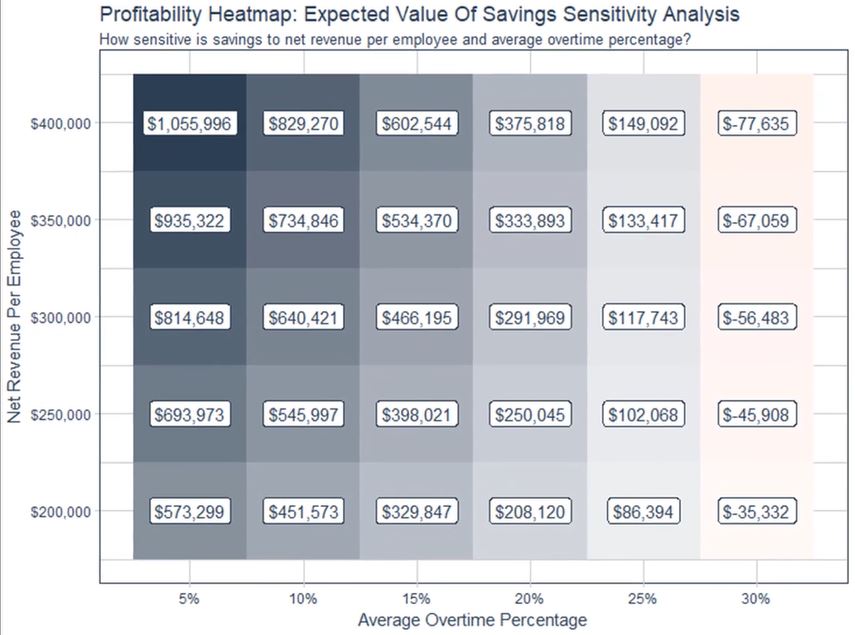
1. A Simple Model where OT is reduced for everyone
2. A more mature model where OT is reduced over time by targeting high risk employees.

Threshold Optimization and Sensitivity Analysis will be developed for optimization.

* **Threshold Optimization** is an iterative approach to expected value. It is critical to maximize profitability (or savings) of a change.
  + **Expected Value** is a way of assigning a value to a decision using both the p*robability of its occurence* and the *potential benefit* of the outcome.
  + Using expected value transforms decisions to be based on probability rather than intuition



* **Sensitivity Analysis** adjusts parameters to illustrate their impact on the model. It is an iterative approach to assess ROI variability.



There are two key terms to understand that support the forthcoming code development:

1. \*Attrition Cost\*\*: The cost resulting primarily from lost productivity quantified as a function of salary and benefits, etc.
2. **Policy Change Cost**: The cost associated with reduction of overtime (it could be another correlated features) given as a percentage of **Attrition Cost**

All costs are annualized

## Expected ROI: No Over Time Policy

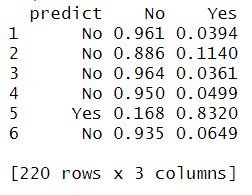
Calculate the expected of changing all employees to no overtime.

3 Steps:

1. Calculate the expected value with overtime as it is now (baseline).
2. Calculate the expected value with no overtime.
3. Calculate the difference and make a decision.

### Expected Value Baseline w OT: Step 1

* Recall assess\_attrition contains several functions:
  + assess\_attrition
  + calculate\_attrition\_cost - This is the one that is needed.
  + count\_to\_pct
  + plot\_attrition
* bind\_cols adds columns from test.tbl to the three columns provided by automl\_leader %>% h2o.predict(newdata = as.h2o(test\_tbl)) %>% as.tibble()
  + Employee Number useful to identify and track employees
  + Monthly Income is used later to estimate the cost of an OT policy change
  + Overtime is needed to determine whether or not the employee is working OT



predictions\_with\_OT\_tbl <- automl\_leader %>% h2o.predict(newdata = as.h2o(test\_tbl)) %>% as.tibble() %>%   
 bind\_cols(test\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime))  
predictions\_with\_OT\_tbl %>% head()

## # A tibble: 6 x 6  
## predict No Yes EmployeeNumber MonthlyIncome OverTime  
## <fct> <dbl> <dbl> <dbl> <dbl> <fct>   
## 1 No 0.964 0.0359 228 9419 No   
## 2 No 0.891 0.109 1278 13577 Yes   
## 3 No 0.967 0.0330 1250 17779 No   
## 4 No 0.953 0.0471 2065 5390 No   
## 5 Yes 0.210 0.790 1767 2437 Yes   
## 6 No 0.937 0.0631 1308 2372 No

Recall calculate\_attrition\_cost:

calculate\_attrition\_cost

## function (n = 1, salary = 80000, separation\_cost = 500, vacancy\_cost = 10000,   
## acquisition\_cost = 4900, placement\_cost = 3500, net\_revenue\_per\_employee = 250000,   
## workdays\_per\_year = 240, workdays\_position\_open = 40, workdays\_onboarding = 60,   
## onboarding\_efficiency = 0.5)   
## {  
## direct\_cost <- sum(separation\_cost, vacancy\_cost, acquisition\_cost,   
## placement\_cost)  
## productivy\_cost <- net\_revenue\_per\_employee/workdays\_per\_year \*   
## (workdays\_position\_open + workdays\_onboarding \* onboarding\_efficiency)  
## salary\_benefit\_reduction <- salary/workdays\_per\_year \* workdays\_position\_open  
## cost\_per\_employee <- direct\_cost + productivy\_cost - salary\_benefit\_reduction  
## total\_cost <- n \* cost\_per\_employee  
## return(total\_cost)  
## }

Some changes we can make:

* annualize the monthly income to calculate the real salary per employee
* capturing net\_revenue\_per\_employee so it can be manipulated
* cost\_of\_policy\_change = 0 because there is no change in the OT policy - keeping everything the same for the baseline
* There is only attrition cost if the employee leaves. The Yes column is the likelihood of leaving (when combined can get an expected attrition cost - coming soon)
* The simplified equation for expected cost:
  + - Remember Yes and No are probabilities - Because cost\_of\_policy\_change is 0,

ev\_with\_OT\_tbl <- predictions\_with\_OT\_tbl %>%   
 mutate(attrition\_cost = calculate\_attrition\_cost(  
 n = 1, salary = MonthlyIncome \* 12, net\_revenue\_per\_employee = 250000)) %>%   
 mutate(cost\_of\_policy\_change = 0) %>%   
 mutate(expected\_attrition\_cost =   
 Yes \* (attrition\_cost + cost\_of\_policy\_change) + No \* (cost\_of\_policy\_change))  
ev\_with\_OT\_tbl

## # A tibble: 220 x 9  
## predict No Yes EmployeeNumber MonthlyIncome OverTime  
## <fct> <dbl> <dbl> <dbl> <dbl> <fct>   
## 1 No 0.964 0.0359 228 9419 No   
## 2 No 0.891 0.109 1278 13577 Yes   
## 3 No 0.967 0.0330 1250 17779 No   
## 4 No 0.953 0.0471 2065 5390 No   
## 5 Yes 0.210 0.790 1767 2437 Yes   
## 6 No 0.937 0.0631 1308 2372 No   
## 7 No 0.956 0.0444 18 2661 No   
## 8 No 0.960 0.0403 460 6347 No   
## 9 No 0.963 0.0370 1369 5363 No   
## 10 No 0.959 0.0414 1040 6347 No   
## # ... with 210 more rows, and 3 more variables: attrition\_cost <dbl>,  
## # cost\_of\_policy\_change <dbl>, expected\_attrition\_cost <dbl>

Now calculate the total attrition cost with OT - the baseline case. Simple to to with summarize.

total\_ev\_with\_OT\_tbl <- ev\_with\_OT\_tbl %>% summarize(total\_expected\_attrition\_cost\_0 = sum(expected\_attrition\_cost))  
total\_ev\_with\_OT\_tbl

## # A tibble: 1 x 1  
## total\_expected\_attrition\_cost\_0  
## <dbl>  
## 1 3078803.

### Expected Value No OT: Step 2

* Modify test\_tbl to change all OT Yes to No. fct\_recode makes this easy!

test\_without\_OT\_tbl <- test\_tbl %>% mutate(OverTime = fct\_recode(OverTime, "No" = "Yes"))  
test\_without\_OT\_tbl$OverTime

## [1] No No No No No No No No No No No No No No No No No No No No No No No  
## [24] No No No No No No No No No No No No No No No No No No No No No No No  
## [47] No No No No No No No No No No No No No No No No No No No No No No No  
## [70] No No No No No No No No No No No No No No No No No No No No No No No  
## [93] No No No No No No No No No No No No No No No No No No No No No No No  
## [116] No No No No No No No No No No No No No No No No No No No No No No No  
## [139] No No No No No No No No No No No No No No No No No No No No No No No  
## [162] No No No No No No No No No No No No No No No No No No No No No No No  
## [185] No No No No No No No No No No No No No No No No No No No No No No No  
## [208] No No No No No No No No No No No No No  
## Levels: No

* Use test\_without\_OT\_tbl to make new predictions using the same code but using a different data file.
* Use rename OverTime so the OT values are side-by-side

predictions\_without\_OT\_tbl <- automl\_leader %>% h2o.predict(newdata = as.h2o(test\_without\_OT\_tbl)) %>%   
 as.tibble() %>%   
 bind\_cols(  
 test\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime),   
 test\_without\_OT\_tbl %>% select(OverTime)) %>%   
 rename(OverTime\_0 = OverTime, OverTime\_1 = OverTime1)  
predictions\_without\_OT\_tbl %>% head()

## # A tibble: 6 x 7  
## predict No Yes EmployeeNumber MonthlyIncome OverTime\_0 OverTime\_1  
## <fct> <dbl> <dbl> <dbl> <dbl> <fct> <fct>   
## 1 No 0.964 0.0359 228 9419 No No   
## 2 No 0.953 0.0465 1278 13577 Yes No   
## 3 No 0.967 0.0330 1250 17779 No No   
## 4 No 0.953 0.0471 2065 5390 No No   
## 5 No 0.793 0.207 1767 2437 Yes No   
## 6 No 0.937 0.0631 1308 2372 No No

Note all the values for Overtime\_1 = No. The new overtime values for Overtime\_1 will have an impact on the expected value calculation.

### Calculate expected value NO OT

* The cost of a policy change will NOT be 0 this time. Instances when OT changes from Yes to No, there is a cost associated with this:
  + $cost\_of\_policy\_change = OverTime\_0 =="Yes" & OverTime\_1 =="No" ~ avg\_overtime\_pct \* attrition\_cost$
  + This cost arises primarily from the lost productivity as a result of not working & producing during OT
    - Add new variable avg\_overtime\_pct. This is a business decision. The goal is to update the calculation to reflect a new policy based on realistic parameters the organization supports.

avg\_overtime\_pct <- 0.10  
  
ev\_without\_OT\_tbl <- predictions\_without\_OT\_tbl %>% mutate(attrition\_cost = calculate\_attrition\_cost(n = 1, salary = MonthlyIncome \* 12,  
 net\_revenue\_per\_employee = 250000)) %>%   
 mutate(cost\_of\_policy\_change = case\_when(  
 OverTime\_0 =="Yes" & OverTime\_1 =="No" ~ avg\_overtime\_pct \* attrition\_cost, TRUE ~ 0) ) %>%   
 mutate(expected\_attrition\_cost = Yes \* (attrition\_cost + cost\_of\_policy\_change) + No \* (cost\_of\_policy\_change))  
ev\_without\_OT\_tbl %>% select(c(MonthlyIncome:expected\_attrition\_cost))

## # A tibble: 220 x 6  
## MonthlyIncome OverTime\_0 OverTime\_1 attrition\_cost cost\_of\_policy\_~  
## <dbl> <fct> <fct> <dbl> <dbl>  
## 1 9419 No No 72979. 0   
## 2 13577 Yes No 64663. 6466.  
## 3 17779 No No 56259. 0   
## 4 5390 No No 81037. 0   
## 5 2437 Yes No 86943. 8694.  
## 6 2372 No No 87073. 0   
## 7 2661 No No 86495. 0   
## 8 6347 No No 79123. 0   
## 9 5363 No No 81091. 0   
## 10 6347 No No 79123. 0   
## # ... with 210 more rows, and 1 more variable:  
## # expected\_attrition\_cost <dbl>

### Calculate total attrition cost

total\_ev\_without\_OT\_tbl <- ev\_without\_OT\_tbl %>% summarise(  
 total\_expected\_attrition\_cost\_1 = sum(expected\_attrition\_cost))  
total\_ev\_without\_OT\_tbl

## # A tibble: 1 x 1  
## total\_expected\_attrition\_cost\_1  
## <dbl>  
## 1 2666825.

Recall the total attrition cost for the baseline (with OT) was 3078802.893 - the cost without OT is lower!

### Calculate the Difference & Make Decision

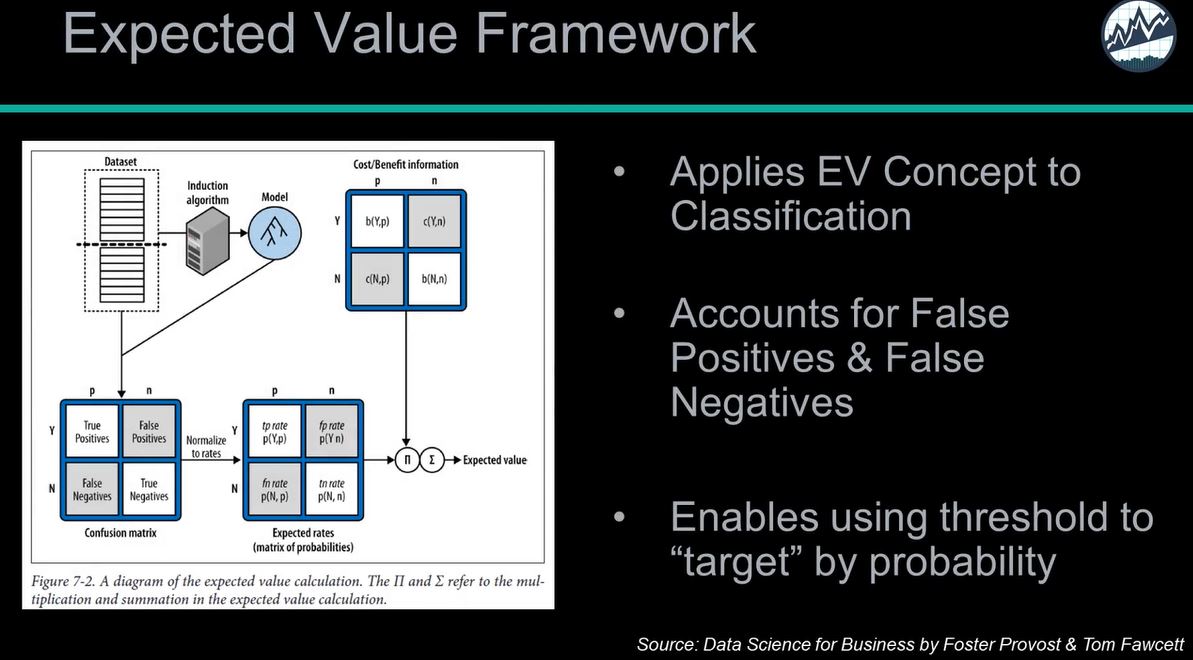
bind\_cols(total\_ev\_with\_OT\_tbl, total\_ev\_without\_OT\_tbl) %>%   
 mutate(  
 savings = total\_expected\_attrition\_cost\_0 - total\_expected\_attrition\_cost\_1,  
 pct\_savings = savings / total\_expected\_attrition\_cost\_0)

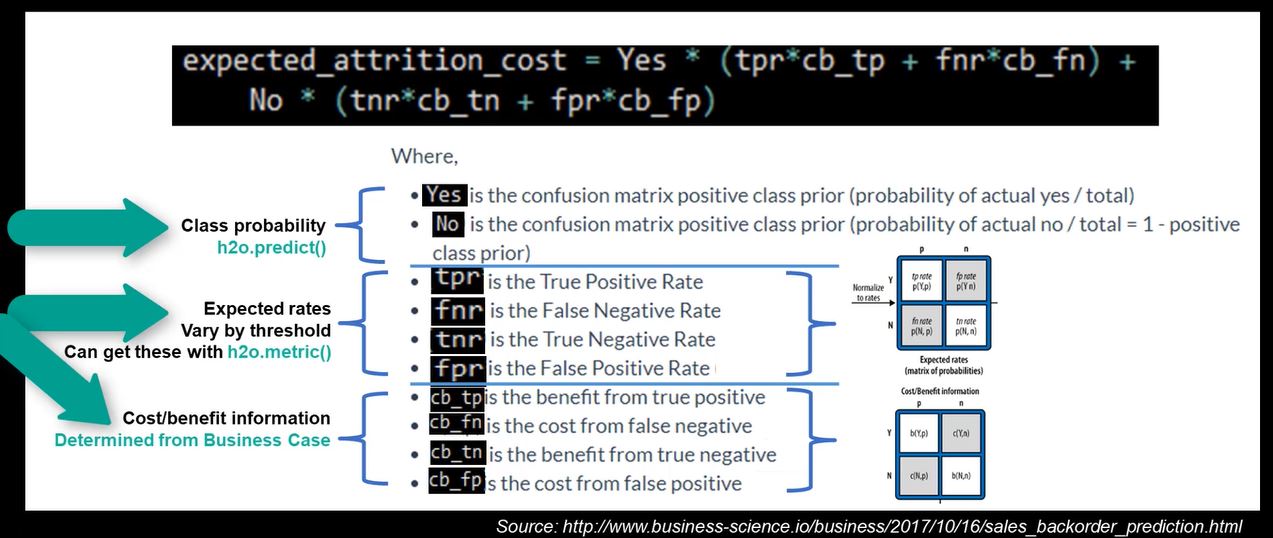
## # A tibble: 1 x 4  
## total\_expected\_attrition\_~ total\_expected\_attrition~ savings pct\_savings  
## <dbl> <dbl> <dbl> <dbl>  
## 1 3078803. 2666825. 411978. 0.134

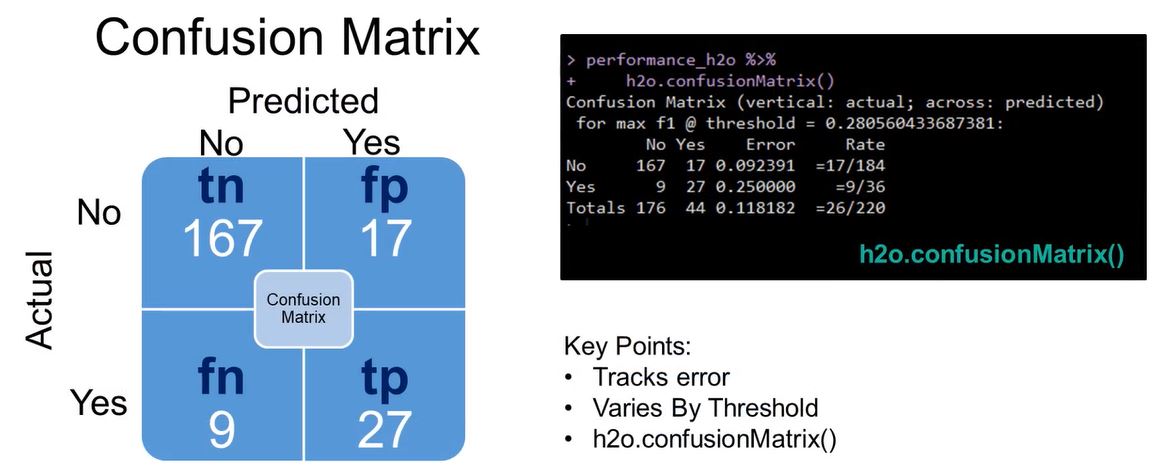
By eliminating OT, there is a potential to save over $400k by eliminating OT. Recall we are examining only 200 records. To annualize this, consider adding the records from train with an additional 1250 employees:

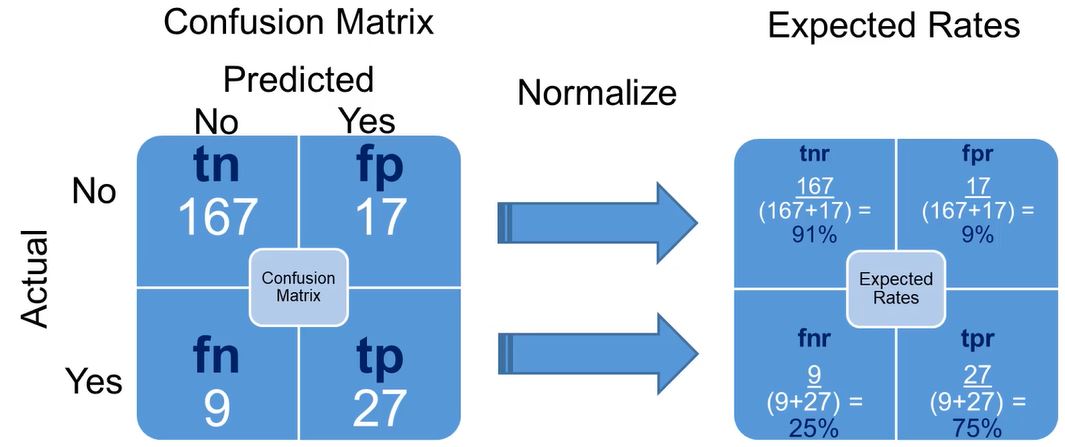
## Targeting OT By Threshold

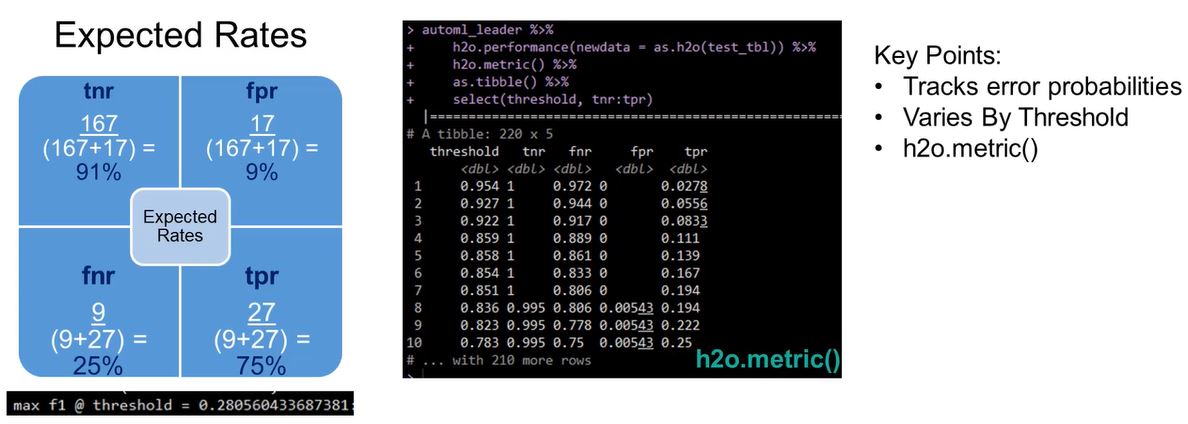
The previous analysis of the baseline compared to a policy of no OT for everyone. If the OT policy was targeted towards those that are likely to leave, then perhaps greater savings might be realized.

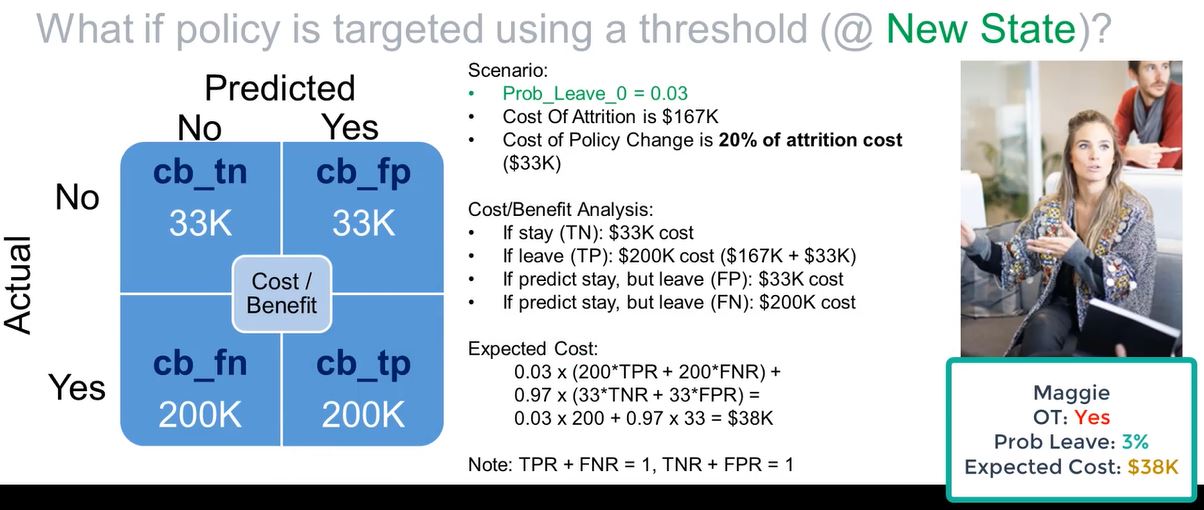












### Threshold Primer: Confusion Matrix

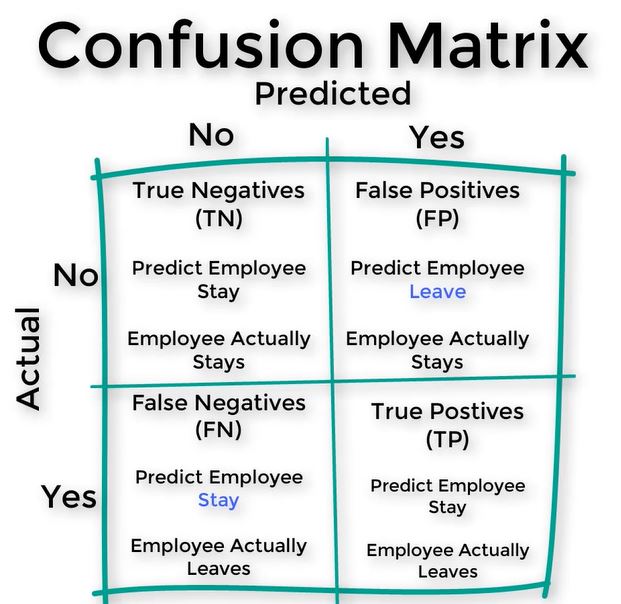
* The rows are actuals No and Yes; the columns are predicted No and Yes
* The False Negatives in Yes row / No column are the most expensive to miss incorrectly - Predicted to STAY but actually LEAVES
* False Positives are not so damaging - predict someone will leave but actually stays - no harm done
* The F1 threshold in the output determines the mix in the confusion matrix
* F1 is the optimal balance between Precision and Recall. However, the threshold @ Max F1 is not typically the optimal value for the business case because FNs are typically more costly then FPs.

performance\_h2o <- automl\_leader %>% h2o.performance(newdata = as.h2o(test\_tbl))

performance\_h2o %>% h2o.confusionMatrix()

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.485216626479869:  
## No Yes Error Rate  
## No 181 3 0.016304 =3/184  
## Yes 16 20 0.444444 =16/36  
## Totals 197 23 0.086364 =19/220

The confusion matrix above is based on a specific F1 threshold



What needs to be evaluated is how the confusion matrix changes over different values of F1. Typically seek to reduce false negatives (because they are often more expensive).

### Threshold Primer: Expected Rates

* h2o.metric returns the classifier performance metrics by threshold.

rates\_by\_threshold\_tbl <- performance\_h2o %>% h2o.metric() %>% as.tibble()  
rates\_by\_threshold\_tbl %>% glimpse()

## Observations: 220  
## Variables: 20  
## $ threshold <dbl> 0.966, 0.899, 0.899, 0.844, 0.839, 0.8...  
## $ f1 <dbl> 0.0541, 0.1053, 0.1538, 0.2000, 0.2439...  
## $ f2 <dbl> 0.0345, 0.0685, 0.1020, 0.1351, 0.1678...  
## $ f0point5 <dbl> 0.125, 0.227, 0.312, 0.385, 0.446, 0.5...  
## $ accuracy <dbl> 0.841, 0.845, 0.850, 0.855, 0.859, 0.8...  
## $ precision <dbl> 1.000, 1.000, 1.000, 1.000, 1.000, 1.0...  
## $ recall <dbl> 0.0278, 0.0556, 0.0833, 0.1111, 0.1389...  
## $ specificity <dbl> 1.000, 1.000, 1.000, 1.000, 1.000, 1.0...  
## $ absolute\_mcc <dbl> 0.153, 0.217, 0.266, 0.308, 0.345, 0.3...  
## $ min\_per\_class\_accuracy <dbl> 0.0278, 0.0556, 0.0833, 0.1111, 0.1389...  
## $ mean\_per\_class\_accuracy <dbl> 0.514, 0.528, 0.542, 0.556, 0.569, 0.5...  
## $ tns <dbl> 184, 184, 184, 184, 184, 184, 184, 184...  
## $ fns <dbl> 35, 34, 33, 32, 31, 30, 29, 28, 27, 27...  
## $ fps <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,...  
## $ tps <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, ...  
## $ tnr <dbl> 1.000, 1.000, 1.000, 1.000, 1.000, 1.0...  
## $ fnr <dbl> 0.972, 0.944, 0.917, 0.889, 0.861, 0.8...  
## $ fpr <dbl> 0.00000, 0.00000, 0.00000, 0.00000, 0....  
## $ tpr <dbl> 0.0278, 0.0556, 0.0833, 0.1111, 0.1389...  
## $ idx <int> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...

Note how the threshold changes other key values.

Only a subset of this data is needed for further analysis.

rates\_by\_threshold\_tbl %>% select(threshold, f1, tnr:tpr)

## # A tibble: 220 x 6  
## threshold f1 tnr fnr fpr tpr  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.966 0.0541 1 0.972 0 0.0278  
## 2 0.899 0.105 1 0.944 0 0.0556  
## 3 0.899 0.154 1 0.917 0 0.0833  
## 4 0.844 0.20 1 0.889 0 0.111   
## 5 0.839 0.244 1 0.861 0 0.139   
## 6 0.816 0.286 1 0.833 0 0.167   
## 7 0.815 0.326 1 0.806 0 0.194   
## 8 0.812 0.364 1 0.778 0 0.222   
## 9 0.790 0.4 1 0.75 0 0.25   
## 10 0.773 0.391 0.995 0.75 0.00543 0.25   
## # ... with 210 more rows

It is easy to get the values for the max F1.

rates\_by\_threshold\_tbl %>% select(threshold, f1, tnr:tpr) %>% filter(f1 == max(f1)) %>% slice(1)

## # A tibble: 1 x 6  
## threshold f1 tnr fnr fpr tpr  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.485 0.678 0.984 0.444 0.0163 0.556

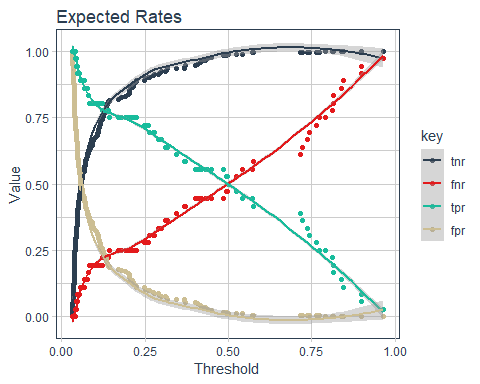
(slice makes sure only 1 F1 record is returned - there can be more than one.)

### Threshold Primer: Visualizing Rates

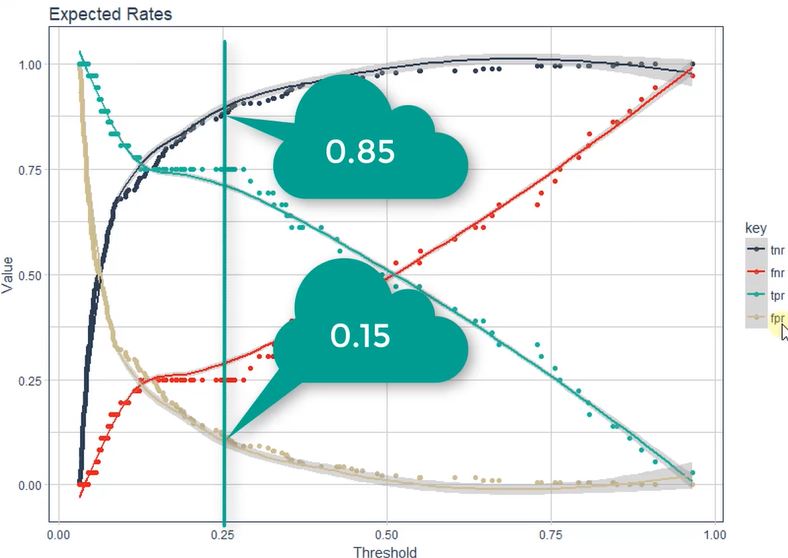
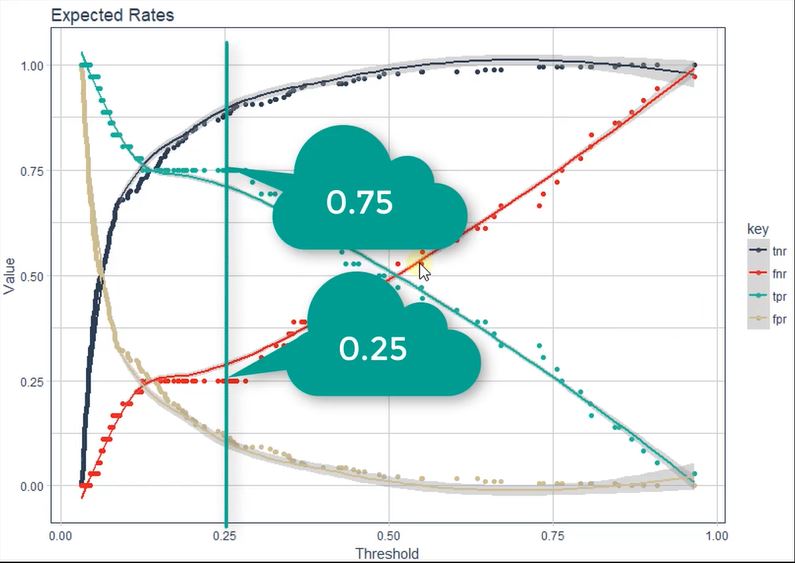
Visualize how the critical confusion matrix values vary by threshold

* Use gather to go from wide to long format - required for ggplot groups
* factor\_key = TRUE keeps the order of tnr:tpr
  + factor\_key: If FALSE, the default, the key values will be stored as a character vector. If TRUE, will be stored as a factor, which preserves the original ordering of the columns.
* fct\_reorder2(key, threshold, value) used when plotting using x and y axis features to control the legend
  + Note the order of the legend matches the order of the values on the right side of the plot
  + the x axis uses Threshold and the y axis uses Value; key controls the legend
  + the key and the threshold have been reordered by key
* and

rates\_by\_threshold\_tbl %>% select(threshold, f1, tnr:tpr) %>%  
 gather(key = "key", value = "value", tnr:tpr, factor\_key = TRUE) %>%  
 mutate(key = fct\_reorder2(key, threshold, value)) %>% ggplot(aes(threshold, value, color = key)) +  
 geom\_point() + geom\_smooth() + theme\_tq() + scale\_color\_tq() +  
 theme(legend.position = "right") + labs(title = "Expected Rates", y = "Value", x = "Threshold")



Note at Threshold = 1, the order of the lines from top to bottom match the order of the legend!

* Group 1: TNR and FPR
  + Probability of correctly identifying a Negative
  + 
* Group 2: TPR and FNR
  + Probability or correctly identifying a positive
  + 

FPs and FNs result in costs

## Targeted Over Time Policy - Expected ROI

### Expected Cost Of Baseline (With OT)

We can copy code developed before in expected\_value\_no\_OT\_policy.R

predictions\_with\_OT\_tbl <- automl\_leader %>% h2o.predict(newdata = as.h2o(test\_tbl)) %>%   
 as.tibble() %>% bind\_cols(test\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime))

ev\_with\_OT\_tbl <- predictions\_with\_OT\_tbl %>%   
 mutate(attrition\_cost = calculate\_attrition\_cost(  
 n = 1, salary = MonthlyIncome \* 12, net\_revenue\_per\_employee = 250000)) %>%   
 mutate(cost\_of\_policy\_change = 0) %>%   
 mutate(expected\_attrition\_cost =   
 Yes \* (attrition\_cost + cost\_of\_policy\_change) +   
 No \* (cost\_of\_policy\_change))  
# Attrition Cost occurs only if the employee leaves. The Yes column is the liklihood of leaving.   
# When combined, expected attrition cost results  
  
total\_ev\_with\_OT\_tbl <- ev\_with\_OT\_tbl %>% summarise(total\_expected\_attrition\_cost\_0 = sum(expected\_attrition\_cost))  
total\_ev\_with\_OT\_tbl

## # A tibble: 1 x 1  
## total\_expected\_attrition\_cost\_0  
## <dbl>  
## 1 3078803.

### Expected Cost Of Targeted OT

Review a previous dataframe, rates\_by\_threshold\_tbl:

rates\_by\_threshold\_tbl

## # A tibble: 220 x 20  
## threshold f1 f2 f0point5 accuracy precision recall specificity  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.966 0.0541 0.0345 0.125 0.841 1 0.0278 1   
## 2 0.899 0.105 0.0685 0.227 0.845 1 0.0556 1   
## 3 0.899 0.154 0.102 0.312 0.85 1 0.0833 1   
## 4 0.844 0.20 0.135 0.385 0.855 1 0.111 1   
## 5 0.839 0.244 0.168 0.446 0.859 1 0.139 1   
## 6 0.816 0.286 0.200 0.5 0.864 1 0.167 1   
## 7 0.815 0.326 0.232 0.547 0.868 1 0.194 1   
## 8 0.812 0.364 0.263 0.588 0.873 1 0.222 1   
## 9 0.790 0.4 0.294 0.625 0.877 1 0.25 1   
## 10 0.773 0.391 0.292 0.592 0.873 0.9 0.25 0.995  
## # ... with 210 more rows, and 12 more variables: absolute\_mcc <dbl>,  
## # min\_per\_class\_accuracy <dbl>, mean\_per\_class\_accuracy <dbl>,  
## # tns <dbl>, fns <dbl>, fps <dbl>, tps <dbl>, tnr <dbl>, fnr <dbl>,  
## # fpr <dbl>, tpr <dbl>, idx <int>

Get the metrics for the max F1

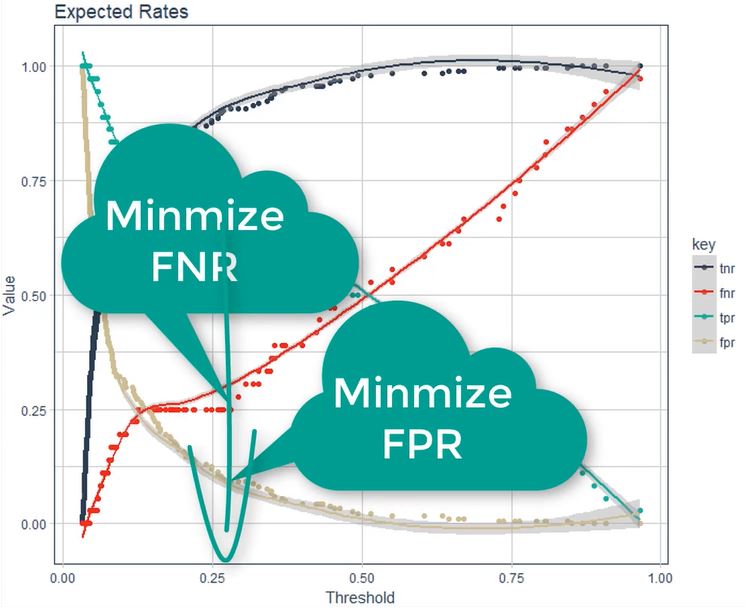
max\_f1\_tbl <- rates\_by\_threshold\_tbl %>% select(threshold, f1, tnr:tpr) %>% filter(f1 == max(f1)) %>% slice(1)  
max\_f1\_tbl

## # A tibble: 1 x 6  
## threshold f1 tnr fnr fpr tpr  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.485 0.678 0.984 0.444 0.0163 0.556

Collect the values for each variable and save to its own variable.

tnr <- max\_f1\_tbl$tnr  
fnr <- max\_f1\_tbl$fnr  
fpr <- max\_f1\_tbl$fpr  
tpr <- max\_f1\_tbl$tpr  
threshold <- max\_f1\_tbl$threshold

The max F1 0.485 is the harmonic mean that minimize the FPR and the FNR:



Any records with a max F1, 0.485, or higher focus on them to change OT to No. To do this, need to modify the test.tbl. Anyone with $Yes >= 0.28 & Overtime = Yes$ then toggle . We are selectively toggling Overtime from Yes to No for employees have a Yes value to No.

* add\_column adds column using a vector of values. From tibble
* Yes >= threshold ~ factor("No", levels = levels(test\_tbl$OverTime))
  + need to make new variable into a factor
  + Use pre-existing levels from test\_tbl

# Anyone with a risk higher than threshold, convert their OT to NO  
test\_targeted\_OT\_tbl <- test\_tbl %>% add\_column(Yes = predictions\_with\_OT\_tbl$Yes) %>%  
 mutate(OverTime = case\_when(  
 Yes >= threshold ~ factor("No", levels = levels(test\_tbl$OverTime)), TRUE ~ OverTime)) %>%  
 select(-Yes)  
test\_targeted\_OT\_tbl

## # A tibble: 220 x 32  
## Age Attrition BusinessTravel DailyRate Department DistanceFromHome  
## <dbl> <fct> <fct> <dbl> <fct> <dbl>  
## 1 30 No Travel\_Rarely 1339 Sales 5  
## 2 55 No Non-Travel 177 Research ~ 8  
## 3 54 No Travel\_Rarely 685 Research ~ 3  
## 4 49 No Travel\_Freque~ 1023 Sales 2  
## 5 43 Yes Travel\_Freque~ 807 Research ~ 17  
## 6 58 No Travel\_Rarely 848 Research ~ 23  
## 7 34 No Travel\_Rarely 1346 Research ~ 19  
## 8 37 No Travel\_Rarely 1192 Research ~ 5  
## 9 35 No Travel\_Rarely 817 Research ~ 1  
## 10 50 No Non-Travel 145 Sales 1  
## # ... with 210 more rows, and 26 more variables: Education <fct>,  
## # EducationField <fct>, EmployeeNumber <dbl>,  
## # EnvironmentSatisfaction <fct>, Gender <fct>, HourlyRate <dbl>,  
## # JobInvolvement <fct>, JobLevel <fct>, JobRole <fct>,  
## # JobSatisfaction <fct>, MaritalStatus <fct>, MonthlyIncome <dbl>,  
## # MonthlyRate <dbl>, NumCompaniesWorked <dbl>, OverTime <fct>,  
## # PercentSalaryHike <dbl>, PerformanceRating <fct>,  
## # RelationshipSatisfaction <fct>, StockOptionLevel <fct>,  
## # TotalWorkingYears <dbl>, TrainingTimesLastYear <dbl>,  
## # WorkLifeBalance <fct>, YearsAtCompany <dbl>, YearsInCurrentRole <dbl>,  
## # YearsSinceLastPromotion <dbl>, YearsWithCurrManager <dbl>

Now the data is prepared, it is time to generate predictions now that we have the test\_targeted\_OT\_tbl.

Running h2o.predict on the new data will output the class probabilities of Attrtition = Yes implementing the policy changes.

predictions\_targeted\_OT\_tbl <- automl\_leader %>%  
 h2o.predict(newdata = as.h2o(test\_targeted\_OT\_tbl)) %>% as.tibble() %>%  
 bind\_cols(  
 test\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime),  
 test\_targeted\_OT\_tbl %>% select(OverTime)) %>%  
 rename(OverTime\_0 = OverTime, OverTime\_1 = OverTime1)

predictions\_targeted\_OT\_tbl

## # A tibble: 220 x 7  
## predict No Yes EmployeeNumber MonthlyIncome OverTime\_0 OverTime\_1  
## <fct> <dbl> <dbl> <dbl> <dbl> <fct> <fct>   
## 1 No 0.964 0.0359 228 9419 No No   
## 2 No 0.891 0.109 1278 13577 Yes Yes   
## 3 No 0.967 0.0330 1250 17779 No No   
## 4 No 0.953 0.0471 2065 5390 No No   
## 5 No 0.793 0.207 1767 2437 Yes No   
## 6 No 0.937 0.0631 1308 2372 No No   
## 7 No 0.956 0.0444 18 2661 No No   
## 8 No 0.960 0.0403 460 6347 No No   
## 9 No 0.963 0.0370 1369 5363 No No   
## 10 No 0.959 0.0414 1040 6347 No No   
## # ... with 210 more rows

**Before**:

## predict No Yes  
## 1 No 0.964 0.0359  
## 2 No 0.891 0.1090  
## 3 No 0.967 0.0330  
## 4 No 0.953 0.0471  
## 5 Yes 0.210 0.7903  
## 6 No 0.937 0.0631  
##   
## [220 rows x 3 columns]

**After**:

## predict No Yes  
## 1 No 0.964 0.0359  
## 2 No 0.891 0.1090  
## 3 No 0.967 0.0330  
## 4 No 0.953 0.0471  
## 5 No 0.793 0.2070  
## 6 No 0.937 0.0631  
##   
## [220 rows x 3 columns]

Record #5 changed from 0.832 to 0.264 and prediction changed from Yes to No.

Predictions have been made and saved in predictions\_targeted\_OT\_tbl. Next step is to calculate the costs of the new OT targeted policy. Some of the code is familiar and copied from previous work.

* calculate attrition cost
* cost of policy change
* Calculating the cost benefits (cb) is the biggest change when working with thresholds.
  + cost benefit are local values that adjust by employee that account for True Positive, True Negative, False Positive and False Negative.
    - cb\_tn = cost of policy change. This is the cost of the employee staying controlled by the cost of policy change logic because they are being targeted.
    - cb\_fp = cost of policy change. This is the cost of predicting an employee to leave but the employee stays. This is controlled by the cost of policy change logic because they were targeted.
    - cb\_tp = cost of policy change + attrition cost. This is controlled by the cost of policy change logic AND attrition costs
    - cb\_fn = cost of policy change + attrition cost. This is the cost when the employee leaves even though he was predicted to stay. This is the most expensive case. This is controlled by the cost of policy change logic AND attrition costs.
* expected\_attrition\_cost calculates the overall expected cost by multiplying the probabilities of the Yes or No by the costs calculated above cb\_\*\*.

Memorize/understand TN and FP go together - they add to 1 - and TP and FN go together because they too sum to 1.

Might be helpful to review the Expected Rates plot above. It makes intuitive sense is you think about it.

avg\_overtime\_pct <- 0.10 # a business decision  
  
ev\_targeted\_OT\_tbl <- predictions\_targeted\_OT\_tbl %>%  
 mutate(attrition\_cost = calculate\_attrition\_cost(n = 1, salary = MonthlyIncome \* 12,  
 net\_revenue\_per\_employee = 250000)) %>%  
 mutate(cost\_of\_policy\_change = case\_when(  
 OverTime\_0 == "Yes" & OverTime\_1 == "No" ~ attrition\_cost \* avg\_overtime\_pct, TRUE ~ 0)) %>%  
 mutate(  
 cb\_tn = cost\_of\_policy\_change, cb\_fp = cost\_of\_policy\_change,  
 cb\_tp = cost\_of\_policy\_change + attrition\_cost, cb\_fn = cost\_of\_policy\_change + attrition\_cost,  
 expected\_attrition\_cost = Yes \* (tpr\*cb\_tp + fnr\*cb\_fn) + No \* (tnr\*cb\_tn + fpr\*cb\_fp))   
  
ev\_targeted\_OT\_tbl

## # A tibble: 220 x 14  
## predict No Yes EmployeeNumber MonthlyIncome OverTime\_0 OverTime\_1  
## <fct> <dbl> <dbl> <dbl> <dbl> <fct> <fct>   
## 1 No 0.964 0.0359 228 9419 No No   
## 2 No 0.891 0.109 1278 13577 Yes Yes   
## 3 No 0.967 0.0330 1250 17779 No No   
## 4 No 0.953 0.0471 2065 5390 No No   
## 5 No 0.793 0.207 1767 2437 Yes No   
## 6 No 0.937 0.0631 1308 2372 No No   
## 7 No 0.956 0.0444 18 2661 No No   
## 8 No 0.960 0.0403 460 6347 No No   
## 9 No 0.963 0.0370 1369 5363 No No   
## 10 No 0.959 0.0414 1040 6347 No No   
## # ... with 210 more rows, and 7 more variables: attrition\_cost <dbl>,  
## # cost\_of\_policy\_change <dbl>, cb\_tn <dbl>, cb\_fp <dbl>, cb\_tp <dbl>,  
## # cb\_fn <dbl>, expected\_attrition\_cost <dbl>

Review the table above with a few examples:

* 1\_st\_ record: 4% Yes, no overtime and therefore nothing to adjust. Because change of policy is not a factor, TN and FP are 0 - they are not leaving. If he does leave, (TP, FN) then these cost benefit features are the same as the attrition costs. The expected attrition cost is low because the chance of leaving is low and they are not targeted.
* The real logic is only leveraged when a record’s threshold - {r threshold} - is exceeded. If someone is working overtime and the threshold is not met, then the overtime stays Yes. Look at record 2 and compare to record 5. If record 2 were to leave, then the full cost would be 64662.667 rather than the expected attrition cost based on probabilities - {r ev\_targeted\_OT\_tbl[2, "expected\_attrition\_cost"]}.

The final step is to calculate the final expected value.

total\_ev\_targeted\_OT\_tbl <- ev\_targeted\_OT\_tbl %>%  
 summarize(total\_expected\_attrition\_cost\_1 = sum(expected\_attrition\_cost))  
  
total\_ev\_targeted\_OT\_tbl

## # A tibble: 1 x 1  
## total\_expected\_attrition\_cost\_1  
## <dbl>  
## 1 2683388.

### Expected Savings - Targeted OT Policy

savings\_tbl <- bind\_cols(total\_ev\_with\_OT\_tbl, total\_ev\_targeted\_OT\_tbl) %>%  
 mutate(savings = total\_expected\_attrition\_cost\_0 - total\_expected\_attrition\_cost\_1,  
 pct\_savings = savings / total\_expected\_attrition\_cost\_0)  
  
savings\_tbl

## # A tibble: 1 x 4  
## total\_expected\_attrition\_~ total\_expected\_attrition~ savings pct\_savings  
## <dbl> <dbl> <dbl> <dbl>  
## 1 3078803. 2683388. 395415. 0.128

Using the targeted OT policy saves ~k500. Recall the targeted OT policy is based on the max F1 threshold. The question next is: *Does the max F1 threshold provide the greatest savings?*

# Threshold Optimization: Maximizing Expected ROI

Is there a better threshold that can provide more savings than the savings using the max F1? We should always check to see this there is a more optimal threshold.

This requires 2 steps:

1. Create a function that can be used on a single threshold
2. Iteratively apply the function to optimize

## Calculate Single Threshold

calculate\_savings\_by\_threshold calculates and returns the savings when the user provides data, h2o model, the threshold and the expected rates (tnr, fpr, fnr, tpr).

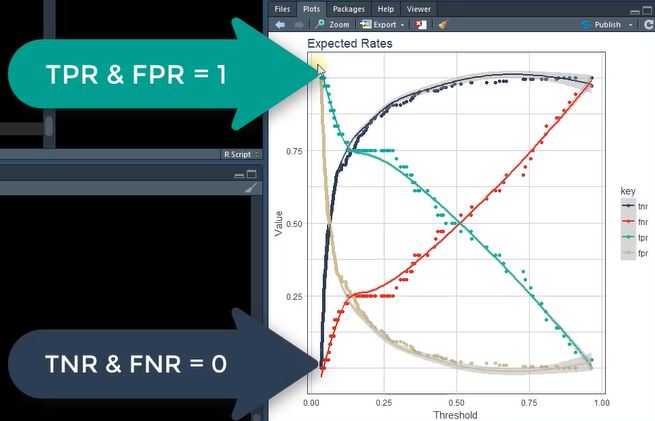
The code below is familiar; this time code is combined into a single function. All that is returned is a single value - the savings.

* The default values set the threshold so low that all OT is eliminated - the same as the no OT use case we completed before.
  + bind\_cols is a dplyr function that does what it says. In this case, the predict, No, and Yes columns returned by h2o\_model %>% h2o.predict(newdata = as.h2o(data\_0\_tbl)) %>% as.tibble() is bound with the notes 3 features from the original data in data/data\_0\_tbl and saved as pred\_0\_tbl.
  + ev\_0\_tbl takes pred\_0\_tbl with the predictions and adds some more features that have been covered before.
  + After dplyr summary code, the expected value is saved in total\_ev\_0\_tbl
* total\_ev\_0\_tbl simply provides the integer for the expected attrition cost.
* data\_1\_tbl get s new new column Yes to overwrite the overtime feature
  + this is the same code explored before.
  + Everyone is switch to no because the threshold is so low.
  + Predictions must be run again.
  + Again we get columns Overtime\_0 and Overtime\_1 again to show who changed Yes to No
  + The rest is code we just reviewed recently.

calculate\_savings\_by\_threshold <- function(data, h2o\_model, threshold = 0,  
 tnr = 0, fpr = 1, fnr = 0, tpr = 1) {  
 data\_0\_tbl <- as.tibble(data)  
   
# Calculating Expected Value With OT   
 pred\_0\_tbl <- h2o\_model %>% h2o.predict(newdata = as.h2o(data\_0\_tbl)) %>% as.tibble() %>%  
 bind\_cols(data\_0\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime))  
   
 ev\_0\_tbl <- pred\_0\_tbl %>%  
 mutate(attrition\_cost = calculate\_attrition\_cost(n = 1,salary = MonthlyIncome \* 12,  
 net\_revenue\_per\_employee = 250000)) %>%  
 mutate(cost\_of\_policy\_change = 0) %>%  
 mutate(expected\_attrition\_cost =   
 Yes \* (attrition\_cost + cost\_of\_policy\_change) + No \* (cost\_of\_policy\_change))  
   
 total\_ev\_0\_tbl <- ev\_0\_tbl %>% summarise(  
 total\_expected\_attrition\_cost\_0 = sum(expected\_attrition\_cost))  
   
#Calculating Expected Value With Targeted OT  
 data\_1\_tbl <- data\_0\_tbl %>% add\_column(Yes = pred\_0\_tbl$Yes) %>%  
 mutate( OverTime = case\_when(  
 Yes >= threshold ~ factor("No", levels = levels(data\_0\_tbl$OverTime)),   
 TRUE ~ OverTime)) %>% select(-Yes)   
   
 pred\_1\_tbl <- h2o\_model %>% h2o.predict(newdata = as.h2o(data\_1\_tbl)) %>% as.tibble() %>%  
 bind\_cols(data\_0\_tbl %>%  
 select(EmployeeNumber, MonthlyIncome, OverTime), data\_1\_tbl %>%  
 select(OverTime)) %>% rename(OverTime\_0 = OverTime, OverTime\_1 = OverTime1)  
   
 avg\_overtime\_pct <- 0.10  
   
 ev\_1\_tbl <- pred\_1\_tbl %>% mutate(attrition\_cost = calculate\_attrition\_cost(n = 1,  
 salary = MonthlyIncome \* 12,  
 net\_revenue\_per\_employee = 250000)) %>%  
 mutate(cost\_of\_policy\_change = case\_when(  
 OverTime\_1 == "No" & OverTime\_0 == "Yes" ~ attrition\_cost \* avg\_overtime\_pct,  
 TRUE ~ 0))%>%  
   
 mutate(cb\_tn = cost\_of\_policy\_change, cb\_fp = cost\_of\_policy\_change,  
 cb\_fn = attrition\_cost + cost\_of\_policy\_change,   
 cb\_tp = attrition\_cost + cost\_of\_policy\_change,  
 expected\_attrition\_cost = Yes \* (tpr\*cb\_tp + fnr\*cb\_fn) + No \* (tnr\*cb\_tn + fpr\*cb\_fp))  
   
 total\_ev\_1\_tbl <- ev\_1\_tbl %>%  
 summarise(total\_expected\_attrition\_cost\_1 = sum(expected\_attrition\_cost))  
   
#Savings Calculation  
 savings\_tbl <- bind\_cols(total\_ev\_0\_tbl, total\_ev\_1\_tbl) %>%  
 mutate(  
 savings = total\_expected\_attrition\_cost\_0 - total\_expected\_attrition\_cost\_1,  
 pct\_savings = savings / total\_expected\_attrition\_cost\_0)  
   
 return(savings\_tbl$savings)}

### Test the new function

Helpful to recall this plot to understand the values from the confusion matrix.



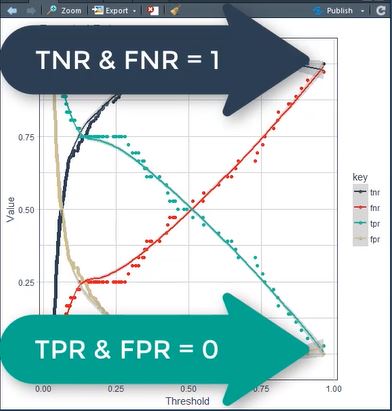
* No OT Policy where
  + Because , everyone get transformed Yes to No

test\_tbl %>% calculate\_savings\_by\_threshold(automl\_leader, threshold = 0, tnr = 0, fnr = 0, tpr = 1, fpr = 1)

## [1] 411978

So the expected savings for a no OT policy is ~$410k

* Do Nothing Policy
  + Threshold is so high no one gets changed from Yes to No.
  + The values for the confusion matrix are reversed and the threshold is set to its max - a value no one can meet.



test\_tbl %>% calculate\_savings\_by\_threshold(automl\_leader, threshold = 1, tnr = 1, fnr = 1, tpr = 0, fpr = 0)

## [1] 0

A savings of $0 makes sense because a threshold = 1 is basically not changing anything. So if there are no changes, there are no savings.

* Threshold @ Max F1

rates\_by\_threshold\_tbl %>% select(threshold, f1, tnr:tpr) %>% filter(f1 == max(f1))  
  
max\_f1\_savings <- calculate\_savings\_by\_threshold(test\_tbl, automl\_leader, threshold = max\_f1\_tbl$threshold,  
 tnr = max\_f1\_tbl$tnr, fpr = max\_f1\_tbl$fpr, fnr = max\_f1\_tbl$fnr, tpr = max\_f1\_tbl$tpr)

max\_f1\_savings

## [1] 395415

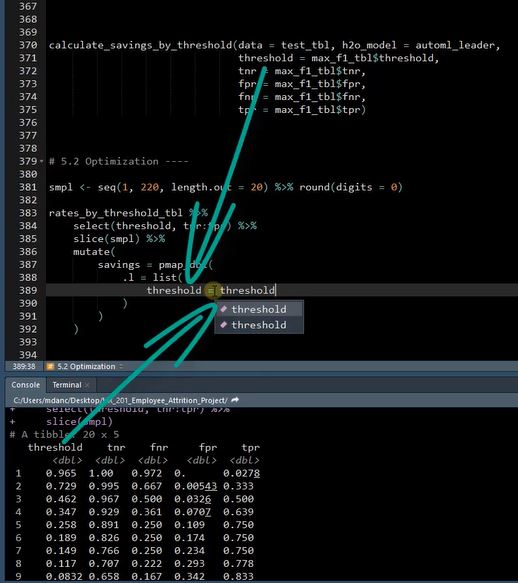
## Optimization with purrr

Time to now iteratively discover the optimal threshold to find the max savings. Will reuse parts of rates\_by\_threshold\_tbl:

head(rates\_by\_threshold\_tbl %>% select(threshold, tnr:tpr))

## # A tibble: 6 x 5  
## threshold tnr fnr fpr tpr  
## <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.966 1 0.972 0 0.0278  
## 2 0.899 1 0.944 0 0.0556  
## 3 0.899 1 0.917 0 0.0833  
## 4 0.844 1 0.889 0 0.111   
## 5 0.839 1 0.861 0 0.139   
## 6 0.816 1 0.833 0 0.167

* seq is used to use a portion of the thresholds.
  + length.out limits the selection to 20 selections from 1 to 200 whole numbers with round(0)
* pmap\_dbl When mutate is used with purrr::map allows for row-wise mapping for iteration {#secondMap}
  + with map alone, column-wise mapping is provided. {#firstMap}
  + One of the pmap functions that use list inputs (.l) arguments, which allow for any number of inputs to be mapped
    - .l is required; it stores a list that map the arguments to the column in the dataframe
  + pmap\_dbl returns a single numerical value as opposed to ’pmap` that returns a list by default
  + purr::map functions are variants of map() iterate over multiple arguments in parallel
  + below, threshold, tnr, fnr, fpr, tpr are all used to to calculate savings
  + recall calculate\_savings\_by\_threshold has the following arguments:
    - function(data, h2o\_model, threshold = 0, tnr = 0, fpr = 1, fnr = 0, tpr = 1)



* .f is a required argument just like .l. - - partial allows you to preload a function with arguments that do not change. {#partial} - in calculate\_savings\_by\_threshold the arguments data and h2o\_model do not change - this is very useful for mapping when arguments like these never change during the iteration

smpl <- seq(1, 220, length.out = 20) %>% round(digits = 0)  
  
rates\_by\_threshold\_optimized\_tbl <- rates\_by\_threshold\_tbl %>%  
 select(threshold, tnr:tpr) %>%  
 slice(smpl) %>%  
 mutate(  
 savings = pmap\_dbl(  
 .l = list(  
 threshold = threshold,  
 tnr = tnr,  
 fnr = fnr,  
 fpr = fpr,  
 tpr = tpr ),  
   
 .f = partial(calculate\_savings\_by\_threshold, data = test\_tbl, h2o\_model = automl\_leader)))

rates\_by\_threshold\_optimized\_tbl

## # A tibble: 20 x 6  
## threshold tnr fnr fpr tpr savings  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.966 1 0.972 0 0.0278 11918.  
## 2 0.737 0.995 0.667 0.00543 0.333 249517.  
## 3 0.450 0.978 0.444 0.0217 0.556 417371.  
## 4 0.315 0.929 0.361 0.0707 0.639 456590.  
## 5 0.225 0.891 0.25 0.109 0.75 504173.  
## 6 0.181 0.826 0.25 0.174 0.75 539867.  
## 7 0.128 0.772 0.222 0.228 0.778 538986.  
## 8 0.113 0.712 0.194 0.288 0.806 529153.  
## 9 0.0930 0.652 0.194 0.348 0.806 517312.  
## 10 0.0737 0.598 0.139 0.402 0.861 513145.  
## 11 0.0648 0.543 0.111 0.457 0.889 490610.  
## 12 0.0570 0.478 0.111 0.522 0.889 477937.  
## 13 0.0513 0.424 0.0833 0.576 0.917 464770.  
## 14 0.0471 0.359 0.0833 0.641 0.917 437500.  
## 15 0.0431 0.310 0.0278 0.690 0.972 429991.  
## 16 0.0403 0.25 0 0.75 1 417023.  
## 17 0.0387 0.190 0 0.810 1 417023.  
## 18 0.0370 0.125 0 0.875 1 417023.  
## 19 0.0345 0.0652 0 0.935 1 411978.  
## 20 0.0312 0 0 1 1 411978.

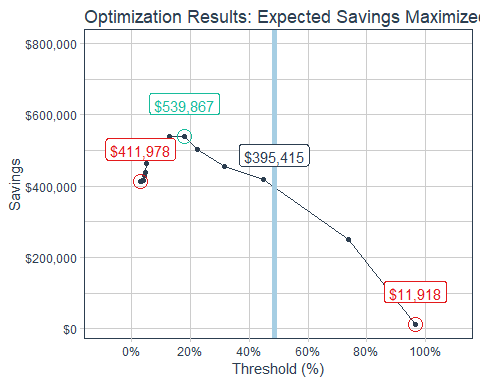
After using the purrr functions, data is available for plotting.{#expand\_limits}{#scale\_x\_continuous}

Max F1 does not provide the maximum savings!

Added this to ensure the title is accurate

myPctMax <- rates\_by\_threshold\_optimized\_tbl %>% filter(savings == max(savings)) %>% select(threshold)  
myPctMax <- scales::percent(as.numeric(myPctMax))

rates\_by\_threshold\_optimized\_tbl %>% ggplot(aes(threshold, savings)) +  
 geom\_line(color = palette\_light()[[1]]) + geom\_point(color = palette\_light()[[1]]) +  
   
 # Optimal Point  
 geom\_point(shape = 21, size = 5, color = palette\_light()[[3]],  
 data = rates\_by\_threshold\_optimized\_tbl %>% filter(savings == max(savings))) +  
 geom\_label(aes(label = scales::dollar(savings)), vjust = -1, color = palette\_light()[[3]],  
 data = rates\_by\_threshold\_optimized\_tbl %>% filter(savings == max(savings))) +  
   
 # F1 Max  
 geom\_vline(xintercept = max\_f1\_tbl$threshold, color = palette\_light()[[5]], size = 2) +  
 annotate(geom = "label", label = scales::dollar(max\_f1\_savings),  
 x = max\_f1\_tbl$threshold, y = max\_f1\_savings, vjust = -1, color = palette\_light()[[1]]) +   
   
 # No OT Policy  
 geom\_point(shape = 21, size = 5, color = palette\_light()[[2]], data = rates\_by\_threshold\_optimized\_tbl %>%  
 filter(threshold == min(threshold))) +   
 geom\_label(aes(label = scales::dollar(savings)), vjust = -1, color = palette\_light()[[2]],  
 data = rates\_by\_threshold\_optimized\_tbl %>% filter(threshold == min(threshold))) +  
   
 # Do Nothing Policy  
 geom\_point(shape = 21, size = 5, color = palette\_light()[[2]], data = rates\_by\_threshold\_optimized\_tbl %>%  
 filter(threshold == max(threshold))) +  
 geom\_label(aes(label = scales::dollar(round(savings, 0))), vjust = -1, color = palette\_light()[[2]],  
 data = rates\_by\_threshold\_optimized\_tbl %>% filter(threshold == max(threshold))) +  
   
 # Aesthestics  
 theme\_tq() + expand\_limits(x = c(-.1, 1.1), y = 8e5) +  
 scale\_x\_continuous(labels = scales::percent, breaks = seq(0, 1, by = 0.2)) +  
 scale\_y\_continuous(labels = scales::dollar) +  
 labs(title = glue("Optimization Results: Expected Savings Maximized At {myPctMax}"),   
 x = "Threshold (%)", y = "Savings")



* Case 1: No OT Policy

rates\_by\_threshold\_optimized\_tbl %>% filter(threshold == min(threshold))

## # A tibble: 1 x 6  
## threshold tnr fnr fpr tpr savings  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.0312 0 0 1 1 411978.

* Threshold = 0
  + Anyone with OT targeted (Yes turns into No)

This policy will reduce turnover, particularly with the higher OT employees.

Recall this is just for a sample of the population - the savings organizationally are much larger

* Case 2: Optimization

rates\_by\_threshold\_optimized\_tbl %>% filter(savings == max(savings))

## # A tibble: 1 x 6  
## threshold tnr fnr fpr tpr savings  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.181 0.826 0.25 0.174 0.75 539867.

* Threshold at max savings
  + Target employees by weighted analysis of cost of FN and cost of FP
  + The optimized value is at a lower threshold than the Max F1. This happens because FNs are more costly the FPs.
    - Precision ~ FPs (predict employee leaves buy stays)
    - Recall ~ FNs (predict employee to stay but leaves)
      * Fail to target and do not reduce OT for 100% of Attrition Cost
      * If the cost of attrition is 100k and the cost to target is, worst case 30% OT, 30k then there is a 70k difference
    - FPs are often a fraction of the cost of a FN

When FNs cost more than FP, the threshold will be lower than the max F1

In this case, reduce the threshold from the Max F1 to 18.053

* Case 3: Max F1

The calculation will be close but not exact. Recall there are only 20 records in rates\_by\_threshold\_optimized\_tbl so the closest record is used for the table below.

#http://r.789695.n4.nabble.com/Find-the-closest-value-in-a-list-or-matrix-td838131.html  
#which(abs(x-your.number)==min(abs(x-your.number)))   
  
x <- rates\_by\_threshold\_optimized\_tbl$threshold  
your.number <- max\_f1\_tbl$threshold  
  
tmp\_row\_index <- which(abs(x-your.number)==min(abs(x-your.number)))   
rates\_by\_threshold\_optimized\_tbl[5,]

## # A tibble: 1 x 6  
## threshold tnr fnr fpr tpr savings  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.225 0.891 0.25 0.109 0.75 504173.

* Threshold = Max F1
  + Target employees balancing FNs and FPs
    - FN = FP
* Case4: Do Nothing - make no changes to the OT policy

rates\_by\_threshold\_optimized\_tbl %>% filter(threshold == max(threshold))

## # A tibble: 1 x 6  
## threshold tnr fnr fpr tpr savings  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.966 1 0.972 0 0.0278 11918.

* Threshold ~1
  + Threshold is so high no one is targeted (no one has a 100% chance of leaving)
  + the actual value should be 0 but we sampled the 20 records, not he full threshold possibilities

## Sensitivity Analysis

Identifying the threshold that maximizes savings is good, but imperfect. Assumptions have been made.

* Assume net revenue per employee = 250k
* Assume average OT = 10%

You will never have perfect information therefore assumptions must be made

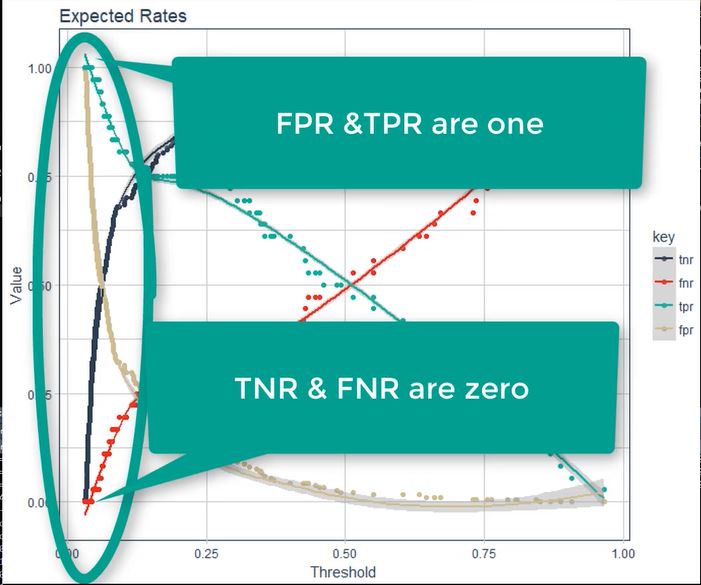
Even though you must make assumptions, you can still try different values to test your assumptions. This is why you do sensitivity analysis.

Next step is to understand the impact these assumptions have on expected savings. Develop a profitability heatmap illustrating the range of values including best, worst and most likely cases and effect on expected savings.

Will create a function then use purrr to iterate through the function.

calculate\_savings\_by\_threshold\_2 is a variant of calculate\_savings\_by\_threshold that has avg\_overtime\_pct and net\_revenue\_per\_employee as inputs. There are only a few changes. . .

* added avg\_overtime\_pct = 0.10 and net\_revenue\_per\_employee = 250000 Note the default values from the confusion matrix:



* net\_revenue\_per\_employee = net\_revenue\_per\_employee is added
* ~ attrition\_cost \* avg\_overtime\_pct is added

###calculate\_savings\_by\_thresh\_2 {#calculate\_savings\_by\_thresh\_2}

calculate\_savings\_by\_threshold\_2 <- function(data, h2o\_model, threshold = 0, tnr = 0, fpr = 1, fnr = 0, tpr = 1,  
 avg\_overtime\_pct = 0.10, net\_revenue\_per\_employee = 250000) {  
 data\_0\_tbl <- as.tibble(data)  
   
 # 4.1 Calculating Expected Value With OT   
 pred\_0\_tbl <- h2o\_model %>% h2o.predict(newdata = as.h2o(data\_0\_tbl)) %>% as.tibble() %>%  
 bind\_cols(data\_0\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime))  
   
 ev\_0\_tbl <- pred\_0\_tbl %>% mutate(attrition\_cost = calculate\_attrition\_cost(n = 1, salary = MonthlyIncome \* 12,  
 # Changed in \_2 ----  
 net\_revenue\_per\_employee = net\_revenue\_per\_employee) ) %>%  
 mutate(cost\_of\_policy\_change = 0) %>%  
 mutate(expected\_attrition\_cost = Yes \* (attrition\_cost + cost\_of\_policy\_change) +  
 No \* (cost\_of\_policy\_change))  
   
 total\_ev\_0\_tbl <- ev\_0\_tbl %>%  
 summarise(total\_expected\_attrition\_cost\_0 = sum(expected\_attrition\_cost))  
   
 # 4.2 Calculating Expected Value With Targeted OT  
 data\_1\_tbl <- data\_0\_tbl %>% add\_column(Yes = pred\_0\_tbl$Yes) %>%  
 mutate(OverTime = case\_when(Yes >= threshold ~ factor("No", levels = levels(data\_0\_tbl$OverTime)),  
 TRUE ~ OverTime)) %>% select(-Yes)   
   
 pred\_1\_tbl <- h2o\_model %>% h2o.predict(newdata = as.h2o(data\_1\_tbl)) %>% as.tibble() %>%  
 bind\_cols(data\_0\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime), data\_1\_tbl %>% select(OverTime)) %>%  
 rename(OverTime\_0 = OverTime, OverTime\_1 = OverTime1)  
   
 ev\_1\_tbl <- pred\_1\_tbl %>% mutate(attrition\_cost = calculate\_attrition\_cost(n = 1, salary = MonthlyIncome \* 12,  
 # Changed in \_2 ----  
 net\_revenue\_per\_employee = net\_revenue\_per\_employee)) %>%  
 mutate(cost\_of\_policy\_change = case\_when(OverTime\_1 == "No" & OverTime\_0 == "Yes"   
 ~ attrition\_cost \* avg\_overtime\_pct, TRUE ~ 0))%>%  
 mutate(cb\_tn = cost\_of\_policy\_change, cb\_fp = cost\_of\_policy\_change,  
 cb\_fn = attrition\_cost + cost\_of\_policy\_change, cb\_tp = attrition\_cost + cost\_of\_policy\_change,  
 expected\_attrition\_cost = Yes \* (tpr\*cb\_tp + fnr\*cb\_fn) + No \* (tnr\*cb\_tn + fpr\*cb\_fp))  
   
 total\_ev\_1\_tbl <- ev\_1\_tbl %>%  
 summarise(total\_expected\_attrition\_cost\_1 = sum(expected\_attrition\_cost))  
   
 # 4.3 Savings Calculation  
   
 savings\_tbl <- bind\_cols(total\_ev\_0\_tbl, total\_ev\_1\_tbl) %>%  
 mutate(savings = total\_expected\_attrition\_cost\_0 - total\_expected\_attrition\_cost\_1,  
 pct\_savings = savings / total\_expected\_attrition\_cost\_0)  
   
 return(savings\_tbl$savings)}

### Sensitivity Analysis

Time to iterate through calculate\_savings\_by\_threshold\_2.

First, use the threshold percentage that optimizes savings for all the iterations. We developed this before. This will be used to set the values in the sensitivity analysis.

max\_savings\_rates\_tbl <- rates\_by\_threshold\_optimized\_tbl %>% filter(savings == max(savings))  
max\_savings\_rates\_tbl

## # A tibble: 1 x 6  
## threshold tnr fnr fpr tpr savings  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.181 0.826 0.25 0.174 0.75 539867.

calculate\_savings\_by\_threshold\_2(  
 data = test\_tbl,  
 h2o\_model = automl\_leader,  
 threshold = max\_savings\_rates\_tbl$threshold,  
 tnr = max\_savings\_rates\_tbl$tnr,  
 fnr = max\_savings\_rates\_tbl$fnr,  
 fpr = max\_savings\_rates\_tbl$fpr,  
 tpr = max\_savings\_rates\_tbl$tpr)

## [1] 539867

Because many of the values will be static, build a partial function using the code above as a template. This is so powerful and yet so simple. Gotta love partial.

calculate\_savings\_by\_threshold\_2\_preloaded <- partial(  
 calculate\_savings\_by\_threshold\_2,  
 # Function Arguments  
 data = test\_tbl,  
 h2o\_model = automl\_leader,  
 threshold = max\_savings\_rates\_tbl$threshold,  
 tnr = max\_savings\_rates\_tbl$tnr,  
 fnr = max\_savings\_rates\_tbl$fnr,  
 fpr = max\_savings\_rates\_tbl$fpr,  
 tpr = max\_savings\_rates\_tbl$tpr)

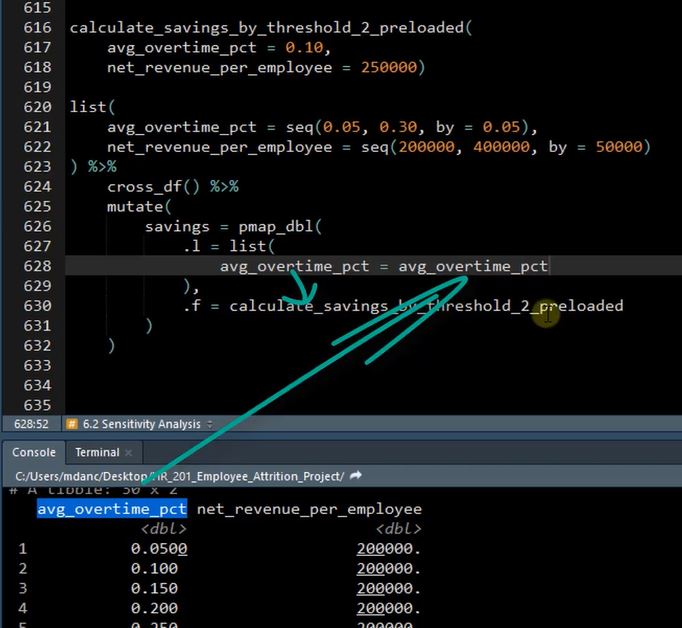
Test the partial function created. This is really helpful - it simplifies our function so we only need to supply the parameters we are iterating on with the sensitivity analysis.

calculate\_savings\_by\_threshold\_2\_preloaded(avg\_overtime\_pct = 0.10, net\_revenue\_per\_employee = 250000)

## [1] 539867

It works - the same value was returned. The goal is to understand how this value changes with revenue per employee and overtime.

* Create a list of the different values to be evaluated in the sensitivity analysis using seq
  + overtime range: 0.05, 0.1, 0.15, 0.2, 0.25, 0.3
  + revenue per employee: 200000, 250000, 300000, 350000, 400000
    - on an income statement,
* cross\_df from purr is really helpful. From a list of variables, it produces all combinations of the list. **Super useful for grid search and sensitivity analysis**. A sample is provided below.{#cross\_df}
  + cross\_df can work with as many variables you want!
* pmap\_dbl is used again to map a function by rows - this time the rows are avg\_overtime\_pct and net\_revenue\_per\_employee
  + returns a numeric value
  + map2 or map2\_dbl would also work however they accept only 2 variable mappings so pmap is more flexible
  + recall pmap takes 2 arguments: .l and .f
    - the .l argument is getting its values from the cross\_df output that has all the combination of values



sample\_n(list(avg\_overtime\_pct = seq(0.05, 0.30, by = 0.05),   
 net\_revenue\_per\_employee = seq(200000, 400000, by = 50000)) %>% cross\_df(), 10) %>%  
 arrange(avg\_overtime\_pct, net\_revenue\_per\_employee)

## # A tibble: 10 x 2  
## avg\_overtime\_pct net\_revenue\_per\_employee  
## <dbl> <dbl>  
## 1 0.05 350000  
## 2 0.1 250000  
## 3 0.1 300000  
## 4 0.1 400000  
## 5 0.2 250000  
## 6 0.2 300000  
## 7 0.25 300000  
## 8 0.25 350000  
## 9 0.25 400000  
## 10 0.3 300000

sensitivity\_tbl <- list(  
 avg\_overtime\_pct = seq(0.05, 0.30, by = 0.05),  
 net\_revenue\_per\_employee = seq(200000, 400000, by = 50000)) %>%  
 cross\_df() %>% mutate(savings = pmap\_dbl(  
 .l = list(avg\_overtime\_pct = avg\_overtime\_pct,  
 net\_revenue\_per\_employee = net\_revenue\_per\_employee),  
 .f = calculate\_savings\_by\_threshold\_2\_preloaded))

head(sensitivity\_tbl)

## # A tibble: 6 x 3  
## avg\_overtime\_pct net\_revenue\_per\_employee savings  
## <dbl> <dbl> <dbl>  
## 1 0.05 200000 552741.  
## 2 0.1 200000 446819.  
## 3 0.15 200000 340896.  
## 4 0.2 200000 234974.  
## 5 0.25 200000 129051.  
## 6 0.3 200000 23128.

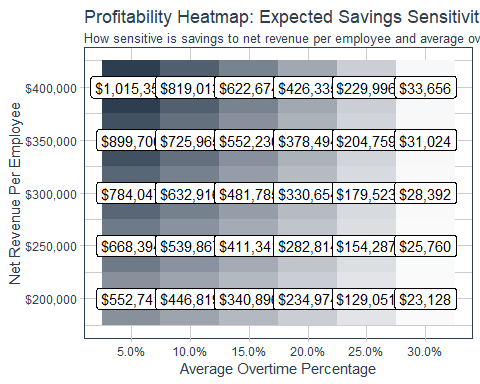
Now the data is available to plot!

## Visualizing Scenarios

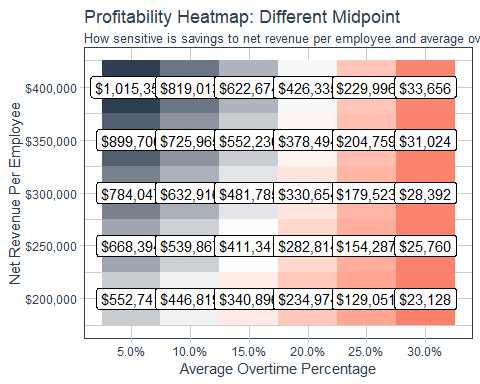
Heat maps are a good visualization to show how 2 variables interact with a 3rd variable (the target)

* scale\_fill\_gradient2 uses 2 gradient transitions by specifying low, mid and high arguments along with a midpoint to specify the middle transition point
  + there are variations like scale\_fill\_gradient that offers only a low and high point.
  + scale\_fill\_gradient\_n allows for any number of gradients.
* scale\_x\_continuous - use the same breaks used earlier: seq(0.05, 0.30, by = 0.05)

sensitivity\_tbl %>%  
 ggplot(aes(avg\_overtime\_pct, net\_revenue\_per\_employee)) + geom\_tile(aes(fill = savings)) +  
 geom\_label(aes(label = savings %>% round(0) %>% scales::dollar())) + theme\_tq() + theme(legend.position = "none") +  
 scale\_fill\_gradient2(low = palette\_light()[[2]], mid = "white", high = palette\_light()[[1]], midpoint = 0) +  
 scale\_x\_continuous(labels = scales::percent, breaks = seq(0.05, 0.30, by = 0.05)) +  
 scale\_y\_continuous(labels = scales::dollar) +  
 labs(title = "Profitability Heatmap: Expected Savings Sensitivity Analysis",  
 subtitle = "How sensitive is savings to net revenue per employee and average overtime percentage?",  
 x = "Average Overtime Percentage", y = "Net Revenue Per Employee")

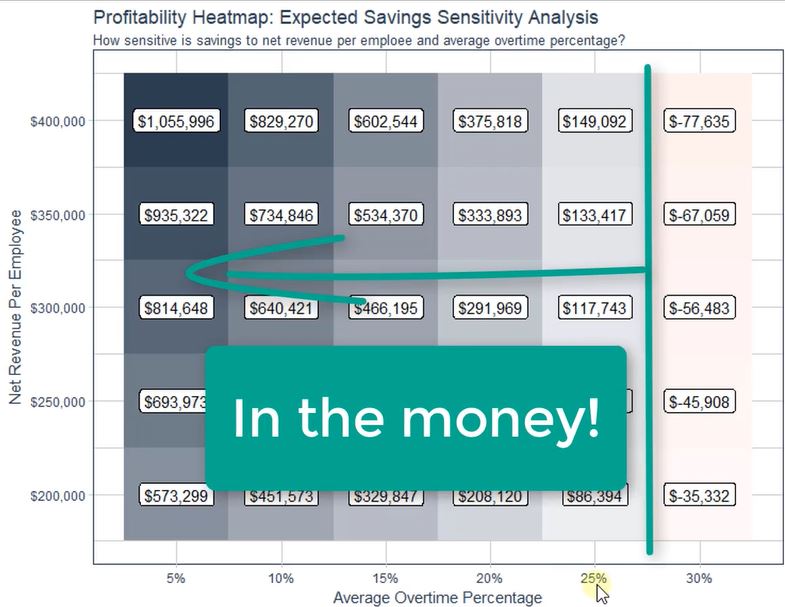


If you want more color variation, change the midpoint value. Try midpoint = mean(sensitivity\_tbl$savings):



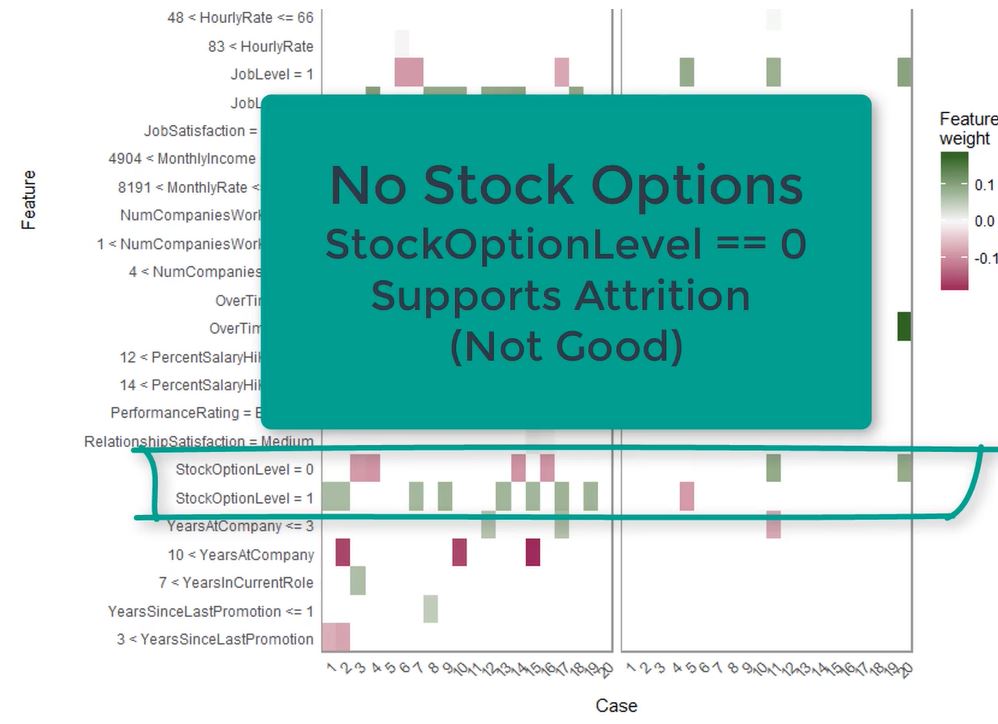
Understanding the profitability heatmap.

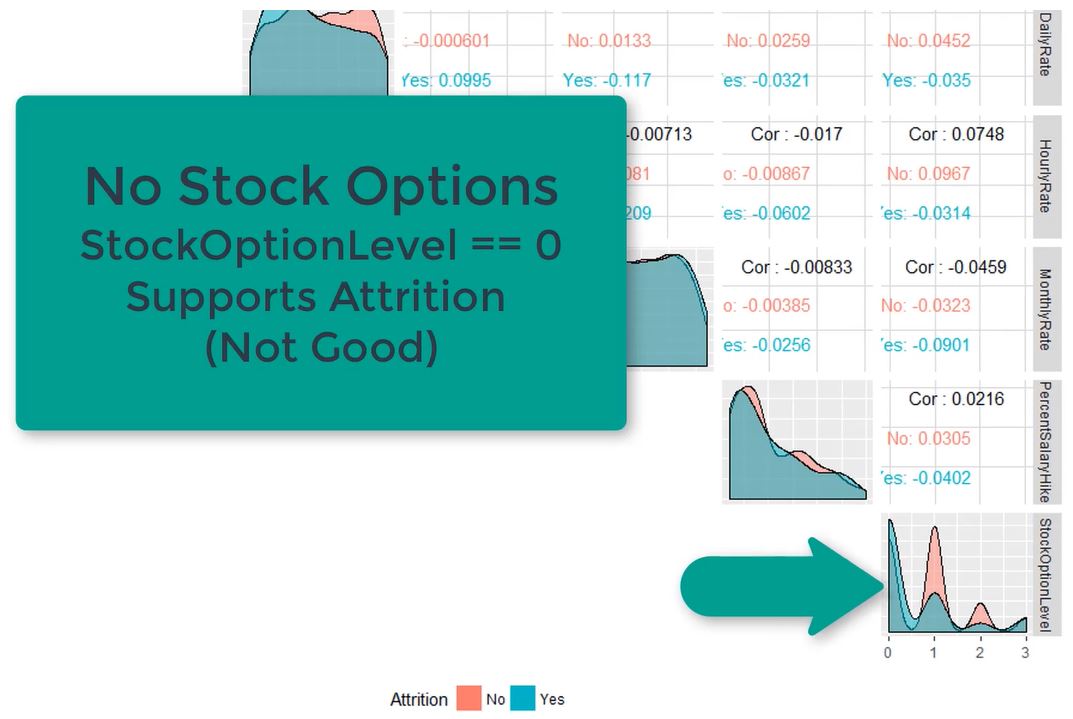
* Provides guidance for unknowns - we do not know the OT, we do not know revenue per employee. There are no perfect assumptions. These can be accounted for by ranges of values of best, worst and most likely cases.
* The heatmap is a great way to show how sensitive features are to ROI with 2 features.
  + For more than 2 features consider developing a solution around a tornado plot. See <https://stackoverflow.com/questions/37059281/tornado-plot-in-r>
* In a different version of the analysis, a break-even point can be identified:



### Optimization For Stock Options

Enhance the sensitivity analysis by adding stock options into the policy change simultaneously with OT reduction.





1. Find the threshold
2. Plot sensitivity

### Find the threshold

Will assume these values:

* avg\_overtime\_pct <- 0.10
* net\_revenue\_per\_employee <- 250000
* stock\_option\_cost <- 5000
  + Targeted employees will have stock option level move from 0 to 1

Changes in the code below to support Stock Options

* data\_0\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime, StockOptionLevel)) to capture the stock info
* mutate(StockOptionLevel = case\_when(Yes >= threshold & StockOptionLevel == 0 ~ factor("1", levels = levels(data\_0\_tbl$StockOptionLevel)), TRUE ~ StockOptionLevel)) to capture when stock option changes from No to Yes
* ind\_cols(data\_0\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime, StockOptionLevel), data\_1\_tbl %>% select(OverTime, StockOptionLevel)) to include stock option in prediction table
* StockOptionLevel\_0 = StockOptionLevel, StockOptionLevel\_1 = StockOptionLevel1 to rename stocks option to be similar to Overtime feature naming
* mutate(cost\_SO = case\_when(StockOptionLevel\_1 == "1" & StockOptionLevel\_0 == "0" ~ stock\_option\_cost, TRUE ~ 0)) to capture stock option cost when a change in options is found

data <- test\_tbl  
h2o\_model <- automl\_leader  
  
threshold <- 0; tnr <- 0; fpr <- 1; fnr <- 0; tpr <- 1  
  
avg\_overtime\_pct <- 0.10  
net\_revenue\_per\_employee <- 250000  
stock\_option\_cost <- 5000  
  
calculate\_savings\_by\_threshold\_3 <- function(data, h2o\_model, threshold = 0, tnr = 0, fpr = 1, fnr = 0, tpr = 1,  
 avg\_overtime\_pct = 0.10, net\_revenue\_per\_employee = 250000,   
 stock\_option\_cost = 5000){  
 data\_0\_tbl <- as.tibble(data)  
   
 #Calculating Expected Value With OT   
 pred\_0\_tbl <- h2o\_model %>% h2o.predict(newdata = as.h2o(data\_0\_tbl)) %>% as.tibble() %>%  
 bind\_cols(  
 # Changed in \_3 Select StockOptionLevel Col----  
 data\_0\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime, StockOptionLevel))  
   
 ev\_0\_tbl <- pred\_0\_tbl %>% mutate(attrition\_cost = calculate\_attrition\_cost(n = 1, salary = MonthlyIncome \* 12,  
 # Changed in \_2 ----  
 net\_revenue\_per\_employee = net\_revenue\_per\_employee)) %>% mutate(cost\_of\_policy\_change = 0) %>%  
 mutate(expected\_attrition\_cost = Yes \* (attrition\_cost + cost\_of\_policy\_change) + No \* (cost\_of\_policy\_change))  
   
 total\_ev\_0\_tbl <- ev\_0\_tbl %>% summarise(total\_expected\_attrition\_cost\_0 = sum(expected\_attrition\_cost))  
   
 #Calculating Expected Value With Targeted OT & Stock Option Policy  
 data\_1\_tbl <- data\_0\_tbl %>% add\_column(Yes = pred\_0\_tbl$Yes) %>%  
 mutate(OverTime = case\_when(Yes >= threshold ~ factor("No", levels = levels(data\_0\_tbl$OverTime)),   
 TRUE ~ OverTime)) %>%  
 # Changed in \_3 Create StockOption Var----  
 mutate(StockOptionLevel = case\_when(Yes >= threshold & StockOptionLevel == 0   
 ~ factor("1", levels = levels(data\_0\_tbl$StockOptionLevel)),   
 TRUE ~ StockOptionLevel)) %>% select(-Yes)   
   
 pred\_1\_tbl <- h2o\_model %>% h2o.predict(newdata = as.h2o(data\_1\_tbl)) %>% as.tibble() %>%  
 # Changed in \_3 Bind StockOption Col----  
 bind\_cols(data\_0\_tbl %>% select(EmployeeNumber, MonthlyIncome, OverTime, StockOptionLevel),  
 data\_1\_tbl %>% select(OverTime, StockOptionLevel)) %>%  
 rename(OverTime\_0 = OverTime, OverTime\_1 = OverTime1,  
 # Changed in \_3 Rename StockOptionLevel----  
 StockOptionLevel\_0 = StockOptionLevel, StockOptionLevel\_1 = StockOptionLevel1)  
   
 ev\_1\_tbl <- pred\_1\_tbl %>% mutate(attrition\_cost = calculate\_attrition\_cost(n = 1, salary = MonthlyIncome \* 12,  
 # Changed in \_2 ----  
 net\_revenue\_per\_employee = net\_revenue\_per\_employee)) %>%  
 # cost\_OT  
 mutate(cost\_OT = case\_when(OverTime\_1 == "No" & OverTime\_0 == "Yes" ~ avg\_overtime\_pct \* MonthlyIncome \* 12, TRUE ~ 0)) %>%  
 # Changed in \_3 Cost Stock----  
 mutate(cost\_SO = case\_when(StockOptionLevel\_1 == "1" & StockOptionLevel\_0 == "0" ~ stock\_option\_cost, TRUE ~ 0)) %>%  
 mutate(cost\_of\_policy\_change = cost\_OT + cost\_SO) %>%  
 mutate(cb\_tn = cost\_of\_policy\_change, cb\_fp = cost\_of\_policy\_change,  
 cb\_fn = attrition\_cost + cost\_of\_policy\_change, cb\_tp = attrition\_cost + cost\_of\_policy\_change,  
 expected\_attrition\_cost = Yes \* (tpr\*cb\_tp + fnr\*cb\_fn) + No \* (tnr\*cb\_tn + fpr\*cb\_fp))   
   
 total\_ev\_1\_tbl <- ev\_1\_tbl %>% summarise(total\_expected\_attrition\_cost\_1 = sum(expected\_attrition\_cost))  
   
 #Savings Calculation  
 savings\_tbl <- bind\_cols(total\_ev\_0\_tbl, total\_ev\_1\_tbl) %>%  
 mutate(savings = total\_expected\_attrition\_cost\_0 - total\_expected\_attrition\_cost\_1,  
 pct\_savings = savings / total\_expected\_attrition\_cost\_0)  
   
 return(savings\_tbl$savings)}

Calculate the savings between doing nothing and max F1:

max\_f1\_tbl <- rates\_by\_threshold\_tbl %>% select(threshold, f1, tnr:tpr) %>% filter(f1 == max(f1))  
  
max\_f1\_savings <- calculate\_savings\_by\_threshold\_3(  
 test\_tbl, automl\_leader, threshold = max\_f1\_tbl$threshold,  
 tnr = max\_f1\_tbl$tnr, fpr = max\_f1\_tbl$fpr, fnr = max\_f1\_tbl$fnr, tpr = max\_f1\_tbl$tpr,  
 avg\_overtime\_pct = 0.10, net\_revenue\_per\_employee = 250000, stock\_option\_cost = 5000)

max\_f1\_savings

## [1] 809316

smpl <- seq(1, 220, length.out = 20) %>% round(digits = 0)  
  
calculate\_savings\_by\_threshold\_3\_preloded <- partial(calculate\_savings\_by\_threshold\_3,   
 data = test\_tbl, h2o\_model = automl\_leader, avg\_overtime\_pct = 0.10,  
 net\_revenue\_per\_employee = 250000, stock\_option\_cost = 5000)  
  
rates\_by\_threshold\_optimized\_tbl\_3 <- rates\_by\_threshold\_tbl %>% select(threshold, tnr:tpr) %>%   
 slice(smpl) %>% mutate(savings = pmap\_dbl(.l = list(threshold = threshold, tnr = tnr, fnr = fnr,   
 fpr = fpr, tpr = tpr),  
 .f = calculate\_savings\_by\_threshold\_3\_preloded))

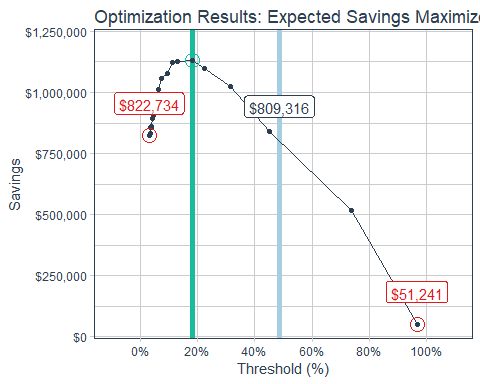
head(rates\_by\_threshold\_optimized\_tbl\_3, 15)

## # A tibble: 15 x 6  
## threshold tnr fnr fpr tpr savings  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.966 1 0.972 0 0.0278 51241.  
## 2 0.737 0.995 0.667 0.00543 0.333 517721.  
## 3 0.450 0.978 0.444 0.0217 0.556 838874.  
## 4 0.315 0.929 0.361 0.0707 0.639 1026488.  
## 5 0.225 0.891 0.25 0.109 0.75 1097668.  
## 6 0.181 0.826 0.25 0.174 0.75 1131973.  
## 7 0.128 0.772 0.222 0.228 0.778 1125033.  
## 8 0.113 0.712 0.194 0.288 0.806 1122143.  
## 9 0.0930 0.652 0.194 0.348 0.806 1079308.  
## 10 0.0737 0.598 0.139 0.402 0.861 1056028.  
## 11 0.0648 0.543 0.111 0.457 0.889 1014063.  
## 12 0.0570 0.478 0.111 0.522 0.889 991530.  
## 13 0.0513 0.424 0.0833 0.576 0.917 963847.  
## 14 0.0471 0.359 0.0833 0.641 0.917 905913.  
## 15 0.0431 0.310 0.0278 0.690 0.972 892787.

rates\_by\_threshold\_optimized\_tbl\_3 %>% filter(savings == max(savings))

## # A tibble: 1 x 6  
## threshold tnr fnr fpr tpr savings  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.181 0.826 0.25 0.174 0.75 1131973.

rates\_by\_threshold\_optimized\_tbl\_3 %>% ggplot(aes(threshold, savings)) +  
 # Vlines  
 geom\_vline(xintercept = max\_f1\_tbl$threshold, color = palette\_light()[[5]], size = 2) +  
 geom\_vline(aes(xintercept = threshold), color = palette\_light()[[3]], size = 2,   
 data = rates\_by\_threshold\_optimized\_tbl\_3 %>% filter(savings == max(savings))) +  
   
 # Points  
 geom\_line(color = palette\_light()[[1]]) + geom\_point(color = palette\_light()[[1]]) +  
   
 # F1 Max  
 annotate(geom = "label", label = scales::dollar(max\_f1\_savings),  
 x = max\_f1\_tbl$threshold, y = max\_f1\_savings, vjust = -1, color = palette\_light()[[1]]) +   
   
 # Optimal Point  
 geom\_point(shape = 21, size = 5, color = palette\_light()[[3]], data = rates\_by\_threshold\_optimized\_tbl\_3 %>%  
 filter(savings == max(savings))) +  
 geom\_label(aes(label = scales::dollar(savings)), vjust = -2, color = palette\_light()[[3]],   
 data = rates\_by\_threshold\_optimized\_tbl\_3 %>% filter(savings == max(savings))) +  
   
 # No OT Policy  
 geom\_point(shape = 21, size = 5, color = palette\_light()[[2]], data = rates\_by\_threshold\_optimized\_tbl\_3 %>%  
 filter(threshold == min(threshold))) +  
 geom\_label(aes(label = scales::dollar(savings)), vjust = -1, color = palette\_light()[[2]],   
 data = rates\_by\_threshold\_optimized\_tbl\_3 %>% filter(threshold == min(threshold))) +  
   
 # Do Nothing Policy  
 geom\_point(shape = 21, size = 5, color = palette\_light()[[2]], data = rates\_by\_threshold\_optimized\_tbl\_3 %>%  
 filter(threshold == max(threshold))) +  
 geom\_label(aes(label = scales::dollar(round(savings, 0))), vjust = -1, color = palette\_light()[[2]],   
 data = rates\_by\_threshold\_optimized\_tbl\_3 %>% filter(threshold == max(threshold))) +  
   
 # Aesthestics  
 theme\_tq() + expand\_limits(x = c(-.1, 1.1), y = 12e5) +   
 scale\_x\_continuous(labels = scales::percent, breaks = seq(0, 1, by = 0.2)) +  
 scale\_y\_continuous(labels = scales::dollar) +  
 labs(title = glue::glue("Optimization Results: Expected Savings Maximized at {max\_f1\_tbl$threshold \* 100}"),  
 x = "Threshold (%)", y = "Savings")



### Plot Sensitivity

#Perform sensitivity analysis at optimal threshold ----  
  
net\_revenue\_per\_employee <- 250000  
avg\_overtime\_pct <- seq(0.05, 0.30, by = 0.05)  
stock\_option\_cost <- seq(5000, 25000, by = 5000)  
  
max\_savings\_rates\_tbl\_3 <- rates\_by\_threshold\_optimized\_tbl\_3 %>% filter(savings == max(savings))  
  
calculate\_savings\_by\_threshold\_3\_preloaded <- partial(calculate\_savings\_by\_threshold\_3,data = test\_tbl,  
 h2o\_model = automl\_leader,  
 threshold = max\_savings\_rates\_tbl\_3$threshold,  
 tnr = max\_savings\_rates\_tbl\_3$tnr,   
 fnr = max\_savings\_rates\_tbl\_3$fnr,  
 fpr = max\_savings\_rates\_tbl\_3$fpr,   
 tpr = max\_savings\_rates\_tbl\_3$tpr)  
  
calculate\_savings\_by\_threshold\_3\_preloaded(avg\_overtime\_pct = 0.10,   
 net\_revenue\_per\_employee = 250000, stock\_option\_cost = 5000)  
  
sensitivity\_tbl\_3 <- list(avg\_overtime\_pct = seq(0.05, 0.30, by = 0.05), net\_revenue\_per\_employee = 250000,  
 stock\_option\_cost = seq(5000, 25000, by = 5000)) %>% cross\_df() %>%  
 mutate(savings = pmap\_dbl(.l = list(avg\_overtime\_pct = avg\_overtime\_pct,   
 net\_revenue\_per\_employee = net\_revenue\_per\_employee,  
 stock\_option\_cost = stock\_option\_cost),  
 .f = calculate\_savings\_by\_threshold\_3\_preloaded))

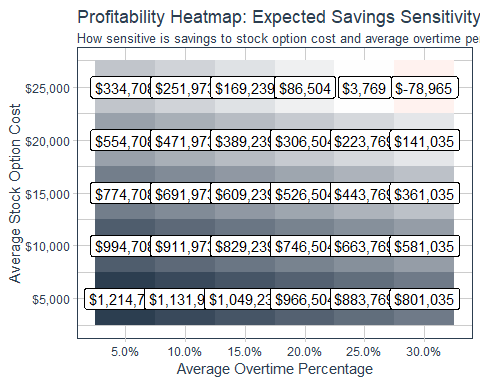
sample\_n(sensitivity\_tbl\_3, 15) %>% arrange(avg\_overtime\_pct, stock\_option\_cost)

## # A tibble: 15 x 4  
## avg\_overtime\_pct net\_revenue\_per\_employee stock\_option\_cost savings  
## <dbl> <dbl> <dbl> <dbl>  
## 1 0.05 250000 10000 994708.  
## 2 0.05 250000 20000 554708.  
## 3 0.1 250000 10000 911973.  
## 4 0.1 250000 15000 691973.  
## 5 0.1 250000 20000 471973.  
## 6 0.1 250000 25000 251973.  
## 7 0.15 250000 5000 1049239.  
## 8 0.15 250000 10000 829239.  
## 9 0.15 250000 20000 389239.  
## 10 0.2 250000 5000 966504.  
## 11 0.2 250000 15000 526504.  
## 12 0.2 250000 20000 306504.  
## 13 0.25 250000 5000 883769.  
## 14 0.25 250000 15000 443769.  
## 15 0.3 250000 20000 141035.

If an employee likelihood to leave > threshold. Recall Stock Option Cost is the value required to move an employee from 0 (no stock options) to 1.

Breakeven is where the row and column transition from negative to positive.

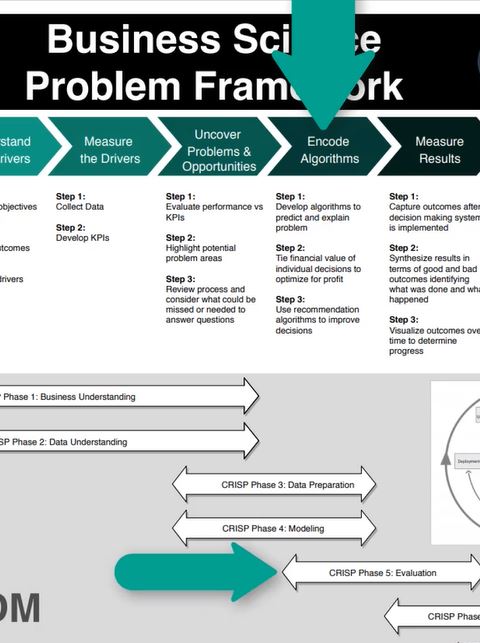
sensitivity\_tbl\_3 %>% ggplot(aes(avg\_overtime\_pct, stock\_option\_cost)) + geom\_tile(aes(fill = savings)) +  
 geom\_label(aes(label = savings %>% round(0) %>% scales::dollar())) + theme\_tq() +  
 theme(legend.position = "none") +  
 scale\_fill\_gradient2(low = palette\_light()[[2]], mid = "white",   
 high = palette\_light()[[1]], midpoint = 0) +  
 scale\_x\_continuous(labels = scales::percent, breaks = seq(0.05, 0.30, by = 0.05)) +  
 scale\_y\_continuous(labels = scales::dollar, breaks = seq(5000, 25000, by = 5000)) +  
 labs(title = "Profitability Heatmap: Expected Savings Sensitivity Analysis",  
 subtitle = "How sensitive is savings to stock option cost and average overtime percentage?",  
 x = "Average Overtime Percentage", y = "Average Stock Option Cost")



# Creating A Recommendation Algorithm

3 step process:

1. Creating a “Correlation Funnel” - This is a discretized correlation plot that enables us to visualize cohort relationships to the target variable, attrition.
2. Using A Recommendation Algorithm Worksheet To Develop Strategies - This is key to having a structured approach to developing data-driven recommendations. It’s also a communication tool for executives & process stakeholders to showcase the logic you used and to incorporate their input into the strategies.
3. Implementing Strategies Into R Code - develop a function called recommend\_strategies() that outputs strategies by employee.



The recommendation algorithm supports Step 3 under Encode Algorithms.

## Recipes For Feature Discretization

<https://tidymodels.github.io/recipes/articles/Ordering.html>

Recall recipes was used in *Data Preparation* when data\_preparation\_part\_2\_machine\_readable.R was built. (#startRecipes) While your project’s needs may vary, here is a suggested order of potential steps that should work for most problems:

1. Impute
2. Individual transformations for skewness and other issues
3. Discretize (if needed and if you have no other choice)
4. Create dummy variables
5. Create interactions
6. Normalization steps (center, scale, range, etc)
7. Multivariate transformation (e.g. PCA, spatial sign, etc)

* Numeric features need to be binned to compare cohorts within the population. Discretation is the process of converting numeric data to factors via binning
* recipes::one\_hot provides a column for every category when dummying variables versus the default of one less column than the number of variables This is very beneficial for correlation analysis interpretability (It does give some warnings - OK in this case b/c there is not enough data for breaks)

Recall what the training data looks like - it has been a while!

train\_readable\_tbl %>% glimpse()

## Observations: 1,250  
## Variables: 35  
## $ Age <dbl> 41, 49, 37, 33, 27, 32, 59, 30, 38, 3...  
## $ Attrition <fct> Yes, No, Yes, No, No, No, No, No, No,...  
## $ BusinessTravel <fct> Travel\_Rarely, Travel\_Frequently, Tra...  
## $ DailyRate <dbl> 1102, 279, 1373, 1392, 591, 1005, 132...  
## $ Department <fct> Sales, Research & Development, Resear...  
## $ DistanceFromHome <dbl> 1, 8, 2, 3, 2, 2, 3, 24, 23, 27, 16, ...  
## $ Education <fct> College, Below College, College, Mast...  
## $ EducationField <fct> Life Sciences, Life Sciences, Other, ...  
## $ EmployeeCount <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...  
## $ EmployeeNumber <dbl> 1, 2, 4, 5, 7, 8, 10, 11, 12, 13, 14,...  
## $ EnvironmentSatisfaction <fct> Medium, High, Very High, Very High, L...  
## $ Gender <fct> Female, Male, Male, Female, Male, Mal...  
## $ HourlyRate <dbl> 94, 61, 92, 56, 40, 79, 81, 67, 44, 9...  
## $ JobInvolvement <fct> High, Medium, Medium, High, High, Hig...  
## $ JobLevel <dbl> 2, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 2, 1...  
## $ JobRole <fct> Sales Executive, Research Scientist, ...  
## $ JobSatisfaction <fct> Very High, Medium, High, High, Medium...  
## $ MaritalStatus <fct> Single, Married, Single, Married, Mar...  
## $ MonthlyIncome <dbl> 5993, 5130, 2090, 2909, 3468, 3068, 2...  
## $ MonthlyRate <dbl> 19479, 24907, 2396, 23159, 16632, 118...  
## $ NumCompaniesWorked <dbl> 8, 1, 6, 1, 9, 0, 4, 1, 0, 6, 0, 0, 1...  
## $ Over18 <fct> Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y...  
## $ OverTime <fct> Yes, No, Yes, Yes, No, No, Yes, No, N...  
## $ PercentSalaryHike <dbl> 11, 23, 15, 11, 12, 13, 20, 22, 21, 1...  
## $ PerformanceRating <fct> Excellent, Outstanding, Excellent, Ex...  
## $ RelationshipSatisfaction <fct> Low, Very High, Medium, High, Very Hi...  
## $ StandardHours <dbl> 80, 80, 80, 80, 80, 80, 80, 80, 80, 8...  
## $ StockOptionLevel <dbl> 0, 1, 0, 0, 1, 0, 3, 1, 0, 2, 1, 0, 1...  
## $ TotalWorkingYears <dbl> 8, 10, 7, 8, 6, 8, 12, 1, 10, 17, 6, ...  
## $ TrainingTimesLastYear <dbl> 0, 3, 3, 3, 3, 2, 3, 2, 2, 3, 5, 3, 1...  
## $ WorkLifeBalance <fct> Bad, Better, Better, Better, Better, ...  
## $ YearsAtCompany <dbl> 6, 10, 0, 8, 2, 7, 1, 1, 9, 7, 5, 9, ...  
## $ YearsInCurrentRole <dbl> 4, 7, 0, 7, 2, 7, 0, 0, 7, 7, 4, 5, 2...  
## $ YearsSinceLastPromotion <dbl> 0, 1, 0, 3, 2, 3, 0, 0, 1, 7, 0, 0, 4...  
## $ YearsWithCurrManager <dbl> 5, 7, 0, 0, 2, 6, 0, 0, 8, 7, 3, 8, 3...

* JobLevel and StockOptionLevel are really factors

factor\_names <- c("JobLevel", "StockOptionLevel")

Build the recipe for the training data and glimpse the result.

* recall thisstep\_dummy(all\_nominal() dummies all categorical data
* previously we did not use this option: step\_discretize(all\_numeric(), options = list(min\_unique = 1))
* Use the bake() to apply the recipe to the data.

recipe\_obj <- recipe(Attrition ~ ., data = train\_readable\_tbl) %>%   
 step\_zv(all\_predictors()) %>%   
 step\_num2factor(factor\_names) %>%   
 step\_discretize(all\_numeric(), options = list(min\_unique = 1)) %>%   
 step\_dummy(all\_nominal(), one\_hot = TRUE) %>%   
 prep()  
  
train\_corr\_tbl <- bake(recipe\_obj, new = train\_readable\_tbl)  
train\_corr\_tbl %>% glimpse()

## Observations: 1,250  
## Variables: 141  
## $ Age\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ Age\_bin1 <dbl> 0, 0, 0, 0, 1, 0, 0, 1, 0,...  
## $ Age\_bin2 <dbl> 0, 0, 0, 1, 0, 1, 0, 0, 0,...  
## $ Age\_bin3 <dbl> 1, 0, 1, 0, 0, 0, 0, 0, 1,...  
## $ Age\_bin4 <dbl> 0, 1, 0, 0, 0, 0, 1, 0, 0,...  
## $ BusinessTravel\_Non.Travel <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ BusinessTravel\_Travel\_Rarely <dbl> 1, 0, 1, 0, 1, 0, 1, 1, 0,...  
## $ BusinessTravel\_Travel\_Frequently <dbl> 0, 1, 0, 1, 0, 1, 0, 0, 1,...  
## $ DailyRate\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ DailyRate\_bin1 <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 1,...  
## $ DailyRate\_bin2 <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0,...  
## $ DailyRate\_bin3 <dbl> 1, 0, 0, 0, 0, 1, 0, 0, 0,...  
## $ DailyRate\_bin4 <dbl> 0, 0, 1, 1, 0, 0, 1, 1, 0,...  
## $ Department\_Human.Resources <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ Department\_Research...Development <dbl> 0, 1, 1, 1, 1, 1, 1, 1, 1,...  
## $ Department\_Sales <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ DistanceFromHome\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ DistanceFromHome\_bin1 <dbl> 1, 0, 1, 0, 1, 1, 0, 0, 0,...  
## $ DistanceFromHome\_bin2 <dbl> 0, 0, 0, 1, 0, 0, 1, 0, 0,...  
## $ DistanceFromHome\_bin3 <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0,...  
## $ DistanceFromHome\_bin4 <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 1,...  
## $ Education\_Below.College <dbl> 0, 1, 0, 0, 1, 0, 0, 1, 0,...  
## $ Education\_College <dbl> 1, 0, 1, 0, 0, 1, 0, 0, 0,...  
## $ Education\_Bachelor <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 1,...  
## $ Education\_Master <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0,...  
## $ Education\_Doctor <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ EducationField\_Human.Resources <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ EducationField\_Life.Sciences <dbl> 1, 1, 0, 1, 0, 1, 0, 1, 1,...  
## $ EducationField\_Marketing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ EducationField\_Medical <dbl> 0, 0, 0, 0, 1, 0, 1, 0, 0,...  
## $ EducationField\_Other <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0,...  
## $ EducationField\_Technical.Degree <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ EmployeeNumber\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ EmployeeNumber\_bin1 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1,...  
## $ EmployeeNumber\_bin2 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ EmployeeNumber\_bin3 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ EmployeeNumber\_bin4 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ EnvironmentSatisfaction\_Low <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0,...  
## $ EnvironmentSatisfaction\_Medium <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ EnvironmentSatisfaction\_High <dbl> 0, 1, 0, 0, 0, 0, 1, 0, 0,...  
## $ EnvironmentSatisfaction\_Very.High <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1,...  
## $ Gender\_Female <dbl> 1, 0, 0, 1, 0, 0, 1, 0, 0,...  
## $ Gender\_Male <dbl> 0, 1, 1, 0, 1, 1, 0, 1, 1,...  
## $ HourlyRate\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ HourlyRate\_bin1 <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 1,...  
## $ HourlyRate\_bin2 <dbl> 0, 1, 0, 1, 0, 0, 0, 0, 0,...  
## $ HourlyRate\_bin3 <dbl> 0, 0, 0, 0, 0, 1, 1, 1, 0,...  
## $ HourlyRate\_bin4 <dbl> 1, 0, 1, 0, 0, 0, 0, 0, 0,...  
## $ JobInvolvement\_Low <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ JobInvolvement\_Medium <dbl> 0, 1, 1, 0, 0, 0, 0, 0, 1,...  
## $ JobInvolvement\_High <dbl> 1, 0, 0, 1, 1, 1, 0, 1, 0,...  
## $ JobInvolvement\_Very.High <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0,...  
## $ JobLevel\_X1 <dbl> 0, 0, 1, 1, 1, 1, 1, 1, 0,...  
## $ JobLevel\_X2 <dbl> 1, 1, 0, 0, 0, 0, 0, 0, 0,...  
## $ JobLevel\_X3 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1,...  
## $ JobLevel\_X4 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ JobLevel\_X5 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ JobRole\_Healthcare.Representative <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ JobRole\_Human.Resources <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ JobRole\_Laboratory.Technician <dbl> 0, 0, 1, 0, 1, 1, 1, 1, 0,...  
## $ JobRole\_Manager <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ JobRole\_Manufacturing.Director <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1,...  
## $ JobRole\_Research.Director <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ JobRole\_Research.Scientist <dbl> 0, 1, 0, 1, 0, 0, 0, 0, 0,...  
## $ JobRole\_Sales.Executive <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ JobRole\_Sales.Representative <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ JobSatisfaction\_Low <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0,...  
## $ JobSatisfaction\_Medium <dbl> 0, 1, 0, 0, 1, 0, 0, 0, 0,...  
## $ JobSatisfaction\_High <dbl> 0, 0, 1, 1, 0, 0, 0, 1, 1,...  
## $ JobSatisfaction\_Very.High <dbl> 1, 0, 0, 0, 0, 1, 0, 0, 0,...  
## $ MaritalStatus\_Single <dbl> 1, 0, 1, 0, 0, 1, 0, 0, 1,...  
## $ MaritalStatus\_Married <dbl> 0, 1, 0, 1, 1, 0, 1, 0, 0,...  
## $ MaritalStatus\_Divorced <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0,...  
## $ MonthlyIncome\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ MonthlyIncome\_bin1 <dbl> 0, 0, 1, 1, 0, 0, 1, 1, 0,...  
## $ MonthlyIncome\_bin2 <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 0,...  
## $ MonthlyIncome\_bin3 <dbl> 1, 1, 0, 0, 0, 0, 0, 0, 0,...  
## $ MonthlyIncome\_bin4 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1,...  
## $ MonthlyRate\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ MonthlyRate\_bin1 <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0,...  
## $ MonthlyRate\_bin2 <dbl> 0, 0, 0, 0, 0, 1, 1, 1, 1,...  
## $ MonthlyRate\_bin3 <dbl> 1, 0, 0, 0, 1, 0, 0, 0, 0,...  
## $ MonthlyRate\_bin4 <dbl> 0, 1, 0, 1, 0, 0, 0, 0, 0,...  
## $ NumCompaniesWorked\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ NumCompaniesWorked\_bin1 <dbl> 0, 1, 0, 1, 0, 1, 0, 1, 1,...  
## $ NumCompaniesWorked\_bin2 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ NumCompaniesWorked\_bin3 <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0,...  
## $ NumCompaniesWorked\_bin4 <dbl> 1, 0, 1, 0, 1, 0, 0, 0, 0,...  
## $ OverTime\_No <dbl> 0, 1, 0, 0, 1, 1, 0, 1, 1,...  
## $ OverTime\_Yes <dbl> 1, 0, 1, 1, 0, 0, 1, 0, 0,...  
## $ PercentSalaryHike\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ PercentSalaryHike\_bin1 <dbl> 1, 0, 0, 1, 1, 0, 0, 0, 0,...  
## $ PercentSalaryHike\_bin2 <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0,...  
## $ PercentSalaryHike\_bin3 <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0,...  
## $ PercentSalaryHike\_bin4 <dbl> 0, 1, 0, 0, 0, 0, 1, 1, 1,...  
## $ PerformanceRating\_Low <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ PerformanceRating\_Good <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ PerformanceRating\_Excellent <dbl> 1, 0, 1, 1, 1, 1, 0, 0, 0,...  
## $ PerformanceRating\_Outstanding <dbl> 0, 1, 0, 0, 0, 0, 1, 1, 1,...  
## $ RelationshipSatisfaction\_Low <dbl> 1, 0, 0, 0, 0, 0, 1, 0, 0,...  
## $ RelationshipSatisfaction\_Medium <dbl> 0, 0, 1, 0, 0, 0, 0, 1, 1,...  
## $ RelationshipSatisfaction\_High <dbl> 0, 0, 0, 1, 0, 1, 0, 0, 0,...  
## $ RelationshipSatisfaction\_Very.High <dbl> 0, 1, 0, 0, 1, 0, 0, 0, 0,...  
## $ StockOptionLevel\_X0 <dbl> 1, 0, 1, 1, 0, 1, 0, 0, 1,...  
## $ StockOptionLevel\_X1 <dbl> 0, 1, 0, 0, 1, 0, 0, 1, 0,...  
## $ StockOptionLevel\_X2 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ StockOptionLevel\_X3 <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0,...  
## $ TotalWorkingYears\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ TotalWorkingYears\_bin1 <dbl> 0, 0, 0, 0, 1, 0, 0, 1, 0,...  
## $ TotalWorkingYears\_bin2 <dbl> 1, 1, 1, 1, 0, 1, 0, 0, 1,...  
## $ TotalWorkingYears\_bin3 <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0,...  
## $ TotalWorkingYears\_bin4 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ TrainingTimesLastYear\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ TrainingTimesLastYear\_bin1 <dbl> 1, 0, 0, 0, 0, 1, 0, 1, 1,...  
## $ TrainingTimesLastYear\_bin2 <dbl> 0, 1, 1, 1, 1, 0, 1, 0, 0,...  
## $ TrainingTimesLastYear\_bin3 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ WorkLifeBalance\_Bad <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ WorkLifeBalance\_Good <dbl> 0, 0, 0, 0, 0, 1, 1, 0, 0,...  
## $ WorkLifeBalance\_Better <dbl> 0, 1, 1, 1, 1, 0, 0, 1, 1,...  
## $ WorkLifeBalance\_Best <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsAtCompany\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsAtCompany\_bin1 <dbl> 0, 0, 1, 0, 1, 0, 1, 1, 0,...  
## $ YearsAtCompany\_bin2 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsAtCompany\_bin3 <dbl> 1, 1, 0, 1, 0, 1, 0, 0, 1,...  
## $ YearsAtCompany\_bin4 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsInCurrentRole\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsInCurrentRole\_bin1 <dbl> 0, 0, 1, 0, 1, 0, 1, 1, 0,...  
## $ YearsInCurrentRole\_bin2 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsInCurrentRole\_bin3 <dbl> 1, 1, 0, 1, 0, 1, 0, 0, 1,...  
## $ YearsInCurrentRole\_bin4 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsSinceLastPromotion\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsSinceLastPromotion\_bin1 <dbl> 1, 1, 1, 0, 0, 0, 1, 1, 1,...  
## $ YearsSinceLastPromotion\_bin2 <dbl> 0, 0, 0, 1, 1, 1, 0, 0, 0,...  
## $ YearsSinceLastPromotion\_bin3 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsWithCurrManager\_bin\_missing <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsWithCurrManager\_bin1 <dbl> 0, 0, 1, 1, 1, 0, 1, 1, 0,...  
## $ YearsWithCurrManager\_bin2 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0,...  
## $ YearsWithCurrManager\_bin3 <dbl> 1, 1, 0, 0, 0, 1, 0, 0, 0,...  
## $ YearsWithCurrManager\_bin4 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1,...  
## $ Attrition\_No <dbl> 0, 1, 0, 1, 1, 1, 1, 1, 1,...  
## $ Attrition\_Yes <dbl> 1, 0, 1, 0, 0, 0, 0, 0, 0,...

* Note all values are binary - 1s and 0s. Data has 141 variables.
* tidy() is a context-specific function that converts non-data-frame objects to data frames (tibbles). From broom package.
  + recipes has a special implementation of tidy()

Below provides the steps that were taken

tidy(recipe\_obj)

## # A tibble: 4 x 5  
## number operation type trained skip   
## <int> <chr> <chr> <lgl> <lgl>  
## 1 1 step zv TRUE FALSE  
## 2 2 step num2factor TRUE FALSE  
## 3 3 step discretize TRUE FALSE  
## 4 4 step dummy TRUE FALSE

It is easy to return the details of the operation in the recipe hierarchy provided by tidy(recipe\_obj). This s very helpful to understand the binning strategy the recipe function provided.

tidy(recipe\_obj, 3)

## # A tibble: 73 x 2  
## terms value  
## <chr> <dbl>  
## 1 Age -Inf  
## 2 Age 30  
## 3 Age 36  
## 4 Age 43  
## 5 Age Inf  
## 6 DailyRate -Inf  
## 7 DailyRate 464  
## 8 DailyRate 798  
## 9 DailyRate 1156  
## 10 DailyRate Inf  
## # ... with 63 more rows

* recall as.factor converts in the order the data appears in the dataset

## Data Manipulation

Goal is to get the correlation Yes information from the train\_corr\_level. First recall what the function get\_cor does. [get\_cor](#get_cor) Review [tidy eval](#tidy_eval) if needed.

get\_cor

## function (data, target, use = "pairwise.complete.obs", fct\_reorder = FALSE,   
## fct\_rev = FALSE)   
## {  
## feature\_expr <- enquo(target)  
## feature\_name <- quo\_name(feature\_expr)  
## data\_cor <- data %>% mutate\_if(is.character, as.factor) %>%   
## mutate\_if(is.factor, as.numeric) %>% cor(use = use) %>%   
## as.tibble() %>% mutate(feature = names(.)) %>% select(feature,   
## !(!feature\_expr)) %>% filter(!(feature == feature\_name)) %>%   
## mutate\_if(is.character, as\_factor)  
## if (fct\_reorder) {  
## data\_cor <- data\_cor %>% mutate(feature = fct\_reorder(feature,   
## !(!feature\_expr))) %>% arrange(feature)  
## }  
## if (fct\_rev) {  
## data\_cor <- data\_cor %>% mutate(feature = fct\_rev(feature)) %>%   
## arrange(feature)  
## }  
## return(data\_cor)  
## }

* Attrion\_No is not needed - only interested in employees that leave
* Filter the records to only evaluate features that have some moderately strong correlation
* After running get\_cor, the table looks like this:

## Warning in stats::cor(x, ...): the standard deviation is zero

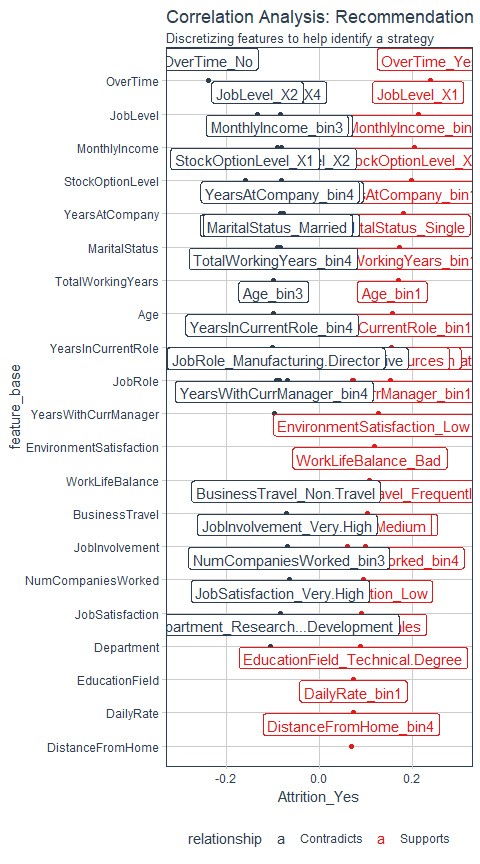
## # A tibble: 6 x 2  
## feature Attrition\_Yes  
## <fct> <dbl>  
## 1 OverTime\_No -0.240   
## 2 StockOptionLevel\_X1 -0.161   
## 3 JobLevel\_X2 -0.134   
## 4 Department\_Research...Development -0.105   
## 5 YearsInCurrentRole\_bin4 -0.102   
## 6 TotalWorkingYears\_bin4 -0.0994

* create a new feature relationship turning the factors in feature to a new column with type character so it can be manipulated easily
* tidyr::separate splits columns of character data into multiple columns using a separator
  + Want to keep only the text before the "\_"
  + extra: If sep is a character vector, this controls what happens when there are too many pieces. There are three valid options:
    1. “warn” (the default): emit a warning and drop extra values.
    2. “drop”: drop any extra values without a warning. (In this case. drop text after the "\_)
    3. “merge”: only splits at most length(into) times
  + This returns the values we will group by
* mutate(feature\_base = as\_factor(feature\_base) %>% fct\_rev())
  + as\_factor assigns factor levels in the order they appear. Therefore, Overtime would be assigned 1, JobLevel would be assigned 2. . . .
  + fct\_rev() changes this so Overtime level would become the highest level number, JobLevel assigned one less and so on. . .
  + This code is needed so the resulting plot a bit later displays correctly (like a funnel). The higher levels are plotted at the top of the plot with each successive level ordered in a descending manner.

corr\_level <- 0.06  
  
correlation\_results\_tbl <- train\_corr\_tbl %>% select(-Attrition\_No) %>%  
 get\_cor(Attrition\_Yes, fct\_reorder = T, fct\_rev = T) %>%  
 filter(abs(Attrition\_Yes) >= corr\_level) %>%  
 mutate(relationship = case\_when(Attrition\_Yes > 0 ~ "Supports", TRUE ~ "Contradicts")) %>%  
 mutate(feature\_text = as.character(feature)) %>%  
 separate(feature\_text, into = "feature\_base", sep = "\_", extra = "drop") %>%  
 mutate(feature\_base = as\_factor(feature\_base) %>% fct\_rev())

## Create Visualization

length\_unique\_groups <- correlation\_results\_tbl %>% pull(feature\_base) %>% unique() %>% length()  
  
correlation\_results\_tbl %>%   
 ggplot(aes(Attrition\_Yes, feature\_base, color = relationship)) + geom\_point() +  
 geom\_label(aes(label = feature), vjust = -0.5) +   
 expand\_limits(x = c(-0.3, 0.3),y = c(1, length\_unique\_groups + 1)) + theme\_tq() +   
 scale\_color\_tq() + labs(title = "Correlation Analysis: Recommendation Strategy Development",  
 subtitle = "Discretizing features to help identify a strategy")

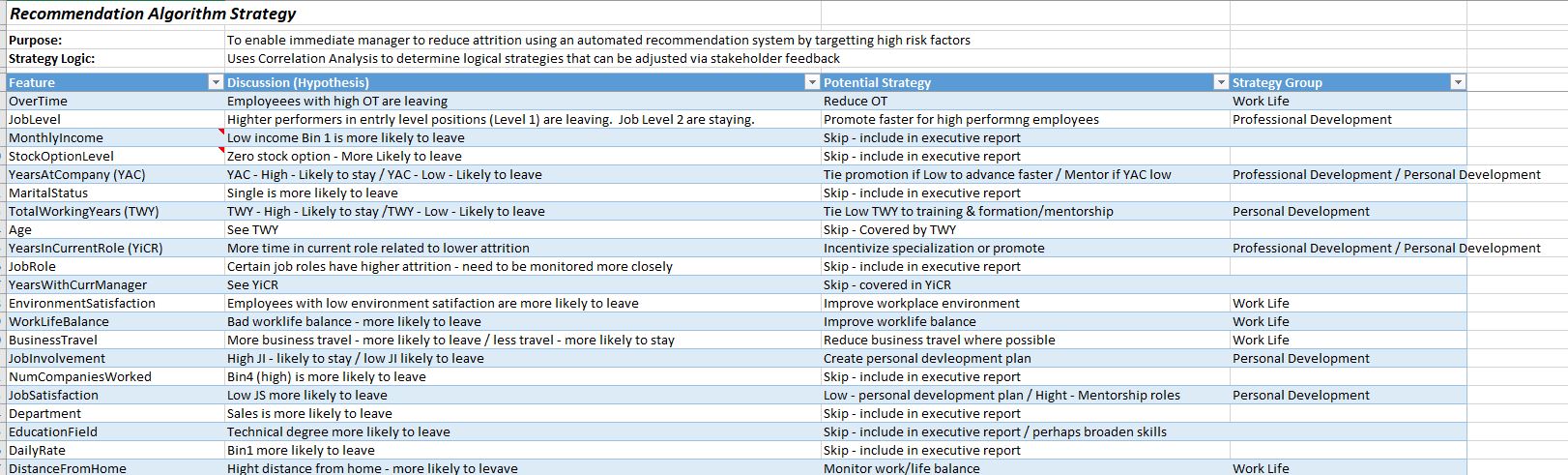


### Explain Correlation Analysis:

* Overtime: Overtime greatly influences people leave.
* JobLevel: Potential strategy - move resources from Level 1 to Level 2 to significantly risk them leaving

## Recommendation Strategy Development Worksheet

The worksheet takes the information from the correlation analysis to develop strategies to implement a recommendation engine. The intent is to provide managers targeted strategies based on high risk factors.

 Note in the worksheet the strategy group options provide a way to group features that appear to have affinity with one and other. These groups will be used in the Shiny App to be developed later.

Skip is selected under Potential Strategy because some features are not open for change/manipulation.

Strategy Group assignment is performed after the other columns are completed

This is where data scientists are important. They apply critical thinking to develop a coded algorithm combining data (data-driven) and knowledge of a system (business-driven). This recommendation algorithm becomes valuable to the business because once deployed, it drives decision making from process owners (in this case Managers)

### Personal Development (Mentorship, Education)

**The information below is copied from the Excel Spreadsheet**

* YearsAtCompany
  + YAC - High - Likely to stay / YAC - Low - Likely to leave
  + Tie promotion if Low to advance faster / Mentor if YAC low
* TotalWorkingYears
  + TWY - High - Likely to stay /TWY - Low - Likely to leave
  + Tie Low TWY to training & formation/mentorship
* YearsInCurrentROle
  + More time in current role related to lower attrition
  + Incentivize specialization or promote
    - more professional than personal - its a mix so modified below ( Incentivize specialization or promote / Mentorship Role)
* JobInvolvement
  + High JI - likely to stay / low JI likely to leave
  + Create personal development plan if low / High seek leadership role
* JobSatisfaction
  + Low JS more likely to leave
  + Low - personal development plan / High - Mentorship roles
* PerformanceRating - ADDED
  + If low, target for personal development plan / High possible mentor or leader
  + If you know features should be included based of your knowledge of the business, add them

### Good-Better-Best Approach

Using the information from the spreadsheet, develop logic that can be turned into code.

Good-Better-Best is just one type of approach.

*Alternatively*, consider for Personal Development, YearsAtCompany has the greatest separation in the Correlation Analysis Plot. Wouldn’t it’s position on the plot suggest it is the feature that needs the most attention and therefore the worst case? (I get that there is little you can really do as a manager for YAC. Just an example to try to understand the Best/Worse Case logic.) *Response*: This is a reasonable assessment. In reality, you’d come up with your logic, which may be imperfect. You’d present this to your management & stakeholders. They would recommend changes as necessary to tweak to their experience and the data, just as you are doing with me. This is the iterative process and communication. The goal is to get a story to tell, and they will suggest changes. This is good - because when you incorporate their changes, you gain buy-in!

* Worst Case -Create Personal Development plan - Job Satisfaction - Job Involvement - ADDED PerformanceRating if low (because it is a logical thing to do)
* Better Case
  + Promote training and formation
    - YearsAtCompany
    - TotalWorkingYears (Not using YearsInCurrentRole is part of the 2 selected above)
* Best Case 1
  + Seek Mentorship Role
    - YearsInCurrentRole
    - YearsAtCompany
    - ADDED PerformanceRating if high, JobSatisfaction (Some people make great mentors, others are good leaders) Mentors tend to be knowledgeable, experienced and have good satisfaction metrics. Typically someone with 3-5 years has sufficient knowledge to become a mentor - this drive the comparison value

Best Case 2 - Seek Leadership Role - JobInvolvement - JobSatisfaction - ADDED PerformanceRating Leaders tend to be very involved and have good performance metrics Want to identify leaders early. The main difference between mentors and leaders is leaders do not have a time component Leaders have high JobInvolvement which is a measure of engagement.

Using the information from the above strategy, code the logic into R code.

Before coding, understand how to discover bins were divided in the recipe. This is easy to do using tidy again. Here is an example using YearsAtCompany:

tidy(recipe\_obj)

## # A tibble: 4 x 5  
## number operation type trained skip   
## <int> <chr> <chr> <lgl> <lgl>  
## 1 1 step zv TRUE FALSE  
## 2 2 step num2factor TRUE FALSE  
## 3 3 step discretize TRUE FALSE  
## 4 4 step dummy TRUE FALSE

Note the *discretize* type above. This tells use the binning occurs in step 3.

tidy(recipe\_obj, number = 3)

## # A tibble: 73 x 2  
## terms value  
## <chr> <dbl>  
## 1 Age -Inf  
## 2 Age 30  
## 3 Age 36  
## 4 Age 43  
## 5 Age Inf  
## 6 DailyRate -Inf  
## 7 DailyRate 464  
## 8 DailyRate 798  
## 9 DailyRate 1156  
## 10 DailyRate Inf  
## # ... with 63 more rows

This provides all the discretized steps. In this example. only interested in YearsAtCompany:{#str\_detect}

tidy(recipe\_obj, number = 3) %>% filter(str\_detect(terms, "YearsAtCompany"))

## # A tibble: 5 x 2  
## terms value  
## <chr> <dbl>  
## 1 YearsAtCompany -Inf  
## 2 YearsAtCompany 3  
## 3 YearsAtCompany 5  
## 4 YearsAtCompany 10  
## 5 YearsAtCompany Inf

This defines the bins. Bin1 = <3, Bin2 = 3 - 4, Bin 3 = 5 - 9, Bin4 > 9. Knowing the definition of the bins is needed below. In practice, perform this function for each binned variable.

tidy(recipe\_obj, number = 3) %>% filter(str\_detect(terms, "TotalWorkingYears"))  
tidy(recipe\_obj, number = 3) %>% filter(str\_detect(terms, "YearsInCurrent"))  
tidy(recipe\_obj, number = 3) %>% filter(str\_detect(terms, "YearsAtCompany"))

It is also important to know factor levels:

# Relate factor label to underlying numeric value  
train\_readable\_tbl %>% pull(JobInvolvement) %>% levels()

## [1] "Low" "Medium" "High" "Very High"

train\_readable\_tbl %>% pull(OverTime) %>% levels()  
train\_readable\_tbl %>% pull(WorkLifeBalance) %>% levels()  
train\_readable\_tbl %>% pull(BusinessTravel) %>% levels()

The next step easy to do. Create an outline that provides sections for Worst Case, Better Case and Best Case. Then collect the measures that are part of each of these sections. Coding then becomes pretty easy! Just leveraging case\_when with the logic that has been defined.

### Recommendation: Personal Development Plan

To illustrate, the worst case is presented below and finished with extra code to see the results. This is importunate as you translate logic into code. While the logic may seem sound at first, if the resulting spread does not make intuitive sense. the metrics assigned may need to be tweaked so the balance of the results make sense and are balanced is such a way that action can be taken effectively.

train\_readable\_tbl %>%   
 select(YearsAtCompany, TotalWorkingYears, YearsInCurrentRole, JobInvolvement, JobSatisfaction, PerformanceRating) %>%   
 mutate\_if(is.factor, as.numeric) %>%   
 mutate(personal\_development\_strategy = case\_when(  
 #Worst Case  
 PerformanceRating == 1 | JobSatisfaction == 1 |   
 JobInvolvement <= 2 ~ "Create Personal Development Plan",   
 TRUE ~ "Retain and Maintain")) %>% pull(personal\_development\_strategy) %>% table()

## .  
## Create Personal Development Plan Retain and Maintain   
## 553 697

About 1/3 of the employees are being targeted to *Create Personal Development Plan*. That seems a bit much so change Job\_Involvment from 12 to 1. This will reduce the number that appears to be more reasonable but the data shows thatJob\_Involvement is an important measure so leave it at 2 for now.

The important thing to do is to always evaluate the outcome from a business perspective at each step.

* Many assigned to Personal Development Plan - may be too much to manage. Use business domain knowledge to change case\_when logic to reduce if appropriate (~308 if JobInvolvement == 1)
* Best Case 2: Seek Leadership Role: When JI, JS and PR all >=4, only one record returned - too aggressive - vary to achieve results that satisfy business domain knowledge

train\_readable\_tbl %>%   
 select(YearsAtCompany, TotalWorkingYears, YearsInCurrentRole, JobInvolvement, JobSatisfaction, PerformanceRating) %>%   
 mutate\_if(is.factor, as.numeric) %>%   
 mutate(personal\_development\_strategy = case\_when(  
 #Worst Case  
 PerformanceRating == 1 | JobSatisfaction == 1 |   
 JobInvolvement <= 2 ~ "Create Personal Development Plan",  
 #Better Case  
 YearsAtCompany < 3 | TotalWorkingYears < 6 ~ "Promote Training and Formation",  
 #Best Case 1: Seek mentorship role  
 (YearsInCurrentRole > 3 | YearsAtCompany >= 5) & PerformanceRating >=3 & JobSatisfaction == 4  
 ~ "Seek Mentorship Role",  
 #Best Case 2: Seek Leadership Role:  
 JobSatisfaction >= 3 & PerformanceRating >= 3 ~ "Seek Leadership Role",  
 # Catch All  
 TRUE ~ "Retain and Maintain")) %>% pull(personal\_development\_strategy) %>% table()

## .  
## Create Personal Development Plan Promote Training and Formation   
## 553 216   
## Retain and Maintain Seek Leadership Role   
## 111 195   
## Seek Mentorship Role   
## 175

Progressive Targeting: The logic based system uses an if-style evaluation progressively isolating the most concerning groups to the least concerning groups.

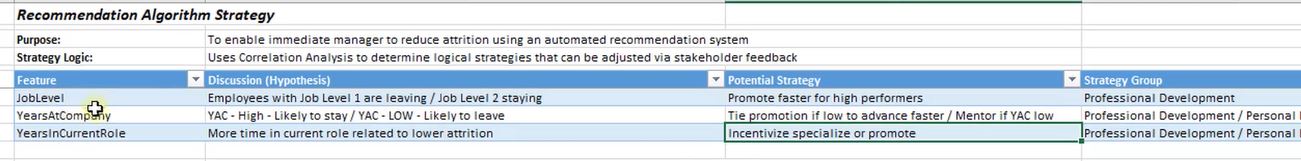
Glimpse at the results:

## Observations: 1,250  
## Variables: 7  
## $ YearsAtCompany <dbl> 6, 10, 0, 8, 2, 7, 1, 1, 9, 7, 5...  
## $ TotalWorkingYears <dbl> 8, 10, 7, 8, 6, 8, 12, 1, 10, 17...  
## $ YearsInCurrentRole <dbl> 4, 7, 0, 7, 2, 7, 0, 0, 7, 7, 4,...  
## $ JobInvolvement <dbl> 3, 2, 2, 3, 3, 3, 4, 3, 2, 3, 4,...  
## $ JobSatisfaction <dbl> 4, 2, 3, 3, 2, 4, 1, 3, 3, 3, 2,...  
## $ PerformanceRating <dbl> 3, 4, 3, 3, 3, 3, 4, 4, 4, 3, 3,...  
## $ personal\_development\_strategy <chr> "Seek Mentorship Role", "Create ...

The goal is not perfection. This will develop over iterations and decision-maker feedback before and during deployment.

### Recommendation: Professional Development

Focused on promotion, rotation and specialization in their job. Rotation keeps people fresh, excited and helps understand multiple aspects of the business. Specialization focuses on becoming the best specialist. Often for highly technical people that love their job (high satisfaction) but are not interested in leadership.

 Also use the Correlation Analysis Plot for guidance.

Progressively make it more difficult to become promoted. This is a good thing because it will incentivize the attribute of performance metrics that are important to the business. These key characteristics should be communicated to employees during performance reviews.

Focused on promotion, rotation and specialization in their job Rotation keeps people fresh, excited and helps understand multiple aspects of the business Specialization focuses on becoming the best specialist. Often for highly technical people that love their job (high satisfaction) but are not interested in leadership

JobLevel - Higher performers in entry level positions (Level 1) are leaving. Job Level 2 are staying.  
- Promote faster for high performing employees

YearsAtCompany (YAC) - YAC - High - Likely to stay / YAC - Low - Likely to leave - Tie promotion if Low to advance faster / Mentor if YAC low

YearsInCurrentRole (YiCR)  
- More time in current role related to lower attrition  
- Incentivize specialization or promote - Preferred as main drive over YAC

Additional Features - JobInvolvement - important for promotion readiness, incentivizes involvement for leaders and early promotion - JobSatisfaction - Important for specialization, incentivizes satisfaction for mentors - PerformanceRating - Important for any promotion

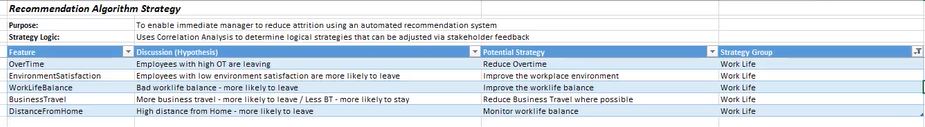
Good Better Best Approach - Typical Career Paths: 1 - Leadership (management); 2- Specialist (Technical) - Both should have promotion schedules based on clear metrics

Ready for rotation: YearsInCurrentRole, JobInvovement, PerformanceRating (LOW) Ready for Promotion Level 2: JobLevel, YearsInCurrentRole, JobInvovement, PerformanceRating Ready for Promotion Level 3: JobLevel, YearsInCurrentRole, JobInvovement, PerformanceRating Ready for Promotion Level 4: JobLevel, YearsInCurrentRole, JobInvovement, PerformanceRating Ready for Promotion Level 5: JobLevel, YearsInCurrentRole, JobInvovement, PerformanceRating Incentives Specialization: YearsInCurrentRole, JobInvovement, PerformanceRating

train\_readable\_tbl %>%   
 select(JobLevel, YearsInCurrentRole, JobInvolvement, JobSatisfaction, PerformanceRating) %>%   
 mutate\_if(is.factor, as.numeric) %>%   
 mutate(  
 professional\_development\_strategy = case\_when(  
 # Ready for rotation: YearsInCurrentRole, JobInvovement, PerformanceRating (LOW)  
 YearsInCurrentRole >= 2 & JobSatisfaction <=2 ~ "Ready for Rotation",  
 # Ready for Promotion Level 2: JobLevel, YearsInCurrentRole, JobInvovement, PerformanceRating  
 JobLevel == 1 & YearsInCurrentRole >= 2 & JobInvolvement >= 3 &   
 PerformanceRating >= 3 ~ "Ready for Promotion",  
 # Ready for Promotion Level 3: JobLevel, YearsInCurrentRole, JobInvovement, PerformanceRating  
 JobLevel == 2 & YearsInCurrentRole >= 2 & JobInvolvement >= 4 &   
 PerformanceRating >= 3 ~ "Ready for Promotion",  
 # Ready for Promotion Level 4: JobLevel, YearsInCurrentRole, JobInvovement, PerformanceRating  
 JobLevel == 3 & YearsInCurrentRole >= 3 & JobInvolvement >= 4 &   
 PerformanceRating >= 3 ~ "Ready for Promotion",  
 # Only a few added at this point but that is probaly OK, pretty high up in the company  
 # Ready for Promotion Level 5: JobLevel, YearsInCurrentRole, JobInvovement, PerformanceRating  
 JobLevel == 4 & YearsInCurrentRole >= 4 & JobInvolvement >= 4 &   
 PerformanceRating >= 3 ~ "Ready for Promotion",  
 # Only a few added at this point but that is probaly OK, pretty high up in the company  
 # Incentives Specialization: YearsInCurrentRole, JobInvovement, PerformanceRating   
 YearsInCurrentRole >= 4 & JobSatisfaction >= 4 &   
 PerformanceRating >= 3 ~ "Incentivizes Specialization",  
 # Catch All  
 TRUE ~ "Retain and Maintain")) %>% pull(professional\_development\_strategy) %>% table()

## .  
## Incentivizes Specialization Ready for Promotion   
## 138 182   
## Ready for Rotation Retain and Maintain   
## 384 546

### Recommendation: Work Environment Strategy



*Good Better Best Approach*

* Improve work life: Overtime, WorklifeBalance
* Monitor Business Travel: BusinessTravel, DistanceFromHome, WorkLifeBalance
* Review Job Assignment: EnvironmentSatisfaction, YearsInCurrentRole
* Promote Job Engagement: JobInvolvement - just a rational addition

train\_readable\_tbl %>%   
 select(OverTime, EnvironmentSatisfaction, WorkLifeBalance, BusinessTravel, DistanceFromHome, YearsInCurrentRole, JobInvolvement) %>%   
 mutate\_if(is.factor, as.numeric) %>%   
 mutate(  
 work\_environment\_strategy = case\_when(  
 # Improve work life: Overtime, WorklifeBalance  
 OverTime == 2 | WorkLifeBalance == 1 ~ "Improve Work-Life Balance",   
 #Recall Overtime 2 == YES: train\_readable\_tbl %>% pull(OverTime) %>% levels()  
 # Monitor Business Travel: BusinessTravel, DistanceFromHome, WorkLifeBalance  
 (BusinessTravel == 3 | DistanceFromHome >= 10) & WorkLifeBalance ==2 ~ "Monitor Business Travel",  
 # WorkLifeBalance = 2 to differentiate from the case\_when above  
 # Review Job Assignment: EnvironmentSatisfaction, YearsInCurrentRole  
 EnvironmentSatisfaction == 1 & YearsInCurrentRole >=2 ~ "Review Job Assignment",  
 # Give the new employee enough time to adjust therefore the 2 years  
 # Promote Job Engagement: JobInvolvement - just a rational addition  
 JobInvolvement <=2 ~ "Promote Job Involvement",  
 TRUE ~ "Retain and Maintain")) %>% count(work\_environment\_strategy)

## # A tibble: 5 x 2  
## work\_environment\_strategy n  
## <chr> <int>  
## 1 Improve Work-Life Balance 407  
## 2 Monitor Business Travel 105  
## 3 Promote Job Involvement 182  
## 4 Retain and Maintain 423  
## 5 Review Job Assignment 133

Review all the work performed so far:

glimpse(train\_readable\_tbl)

## Observations: 1,250  
## Variables: 35  
## $ Age <dbl> 41, 49, 37, 33, 27, 32, 59, 30, 38, 3...  
## $ Attrition <fct> Yes, No, Yes, No, No, No, No, No, No,...  
## $ BusinessTravel <fct> Travel\_Rarely, Travel\_Frequently, Tra...  
## $ DailyRate <dbl> 1102, 279, 1373, 1392, 591, 1005, 132...  
## $ Department <fct> Sales, Research & Development, Resear...  
## $ DistanceFromHome <dbl> 1, 8, 2, 3, 2, 2, 3, 24, 23, 27, 16, ...  
## $ Education <fct> College, Below College, College, Mast...  
## $ EducationField <fct> Life Sciences, Life Sciences, Other, ...  
## $ EmployeeCount <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...  
## $ EmployeeNumber <dbl> 1, 2, 4, 5, 7, 8, 10, 11, 12, 13, 14,...  
## $ EnvironmentSatisfaction <fct> Medium, High, Very High, Very High, L...  
## $ Gender <fct> Female, Male, Male, Female, Male, Mal...  
## $ HourlyRate <dbl> 94, 61, 92, 56, 40, 79, 81, 67, 44, 9...  
## $ JobInvolvement <fct> High, Medium, Medium, High, High, Hig...  
## $ JobLevel <dbl> 2, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 2, 1...  
## $ JobRole <fct> Sales Executive, Research Scientist, ...  
## $ JobSatisfaction <fct> Very High, Medium, High, High, Medium...  
## $ MaritalStatus <fct> Single, Married, Single, Married, Mar...  
## $ MonthlyIncome <dbl> 5993, 5130, 2090, 2909, 3468, 3068, 2...  
## $ MonthlyRate <dbl> 19479, 24907, 2396, 23159, 16632, 118...  
## $ NumCompaniesWorked <dbl> 8, 1, 6, 1, 9, 0, 4, 1, 0, 6, 0, 0, 1...  
## $ Over18 <fct> Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y...  
## $ OverTime <fct> Yes, No, Yes, Yes, No, No, Yes, No, N...  
## $ PercentSalaryHike <dbl> 11, 23, 15, 11, 12, 13, 20, 22, 21, 1...  
## $ PerformanceRating <fct> Excellent, Outstanding, Excellent, Ex...  
## $ RelationshipSatisfaction <fct> Low, Very High, Medium, High, Very Hi...  
## $ StandardHours <dbl> 80, 80, 80, 80, 80, 80, 80, 80, 80, 8...  
## $ StockOptionLevel <dbl> 0, 1, 0, 0, 1, 0, 3, 1, 0, 2, 1, 0, 1...  
## $ TotalWorkingYears <dbl> 8, 10, 7, 8, 6, 8, 12, 1, 10, 17, 6, ...  
## $ TrainingTimesLastYear <dbl> 0, 3, 3, 3, 3, 2, 3, 2, 2, 3, 5, 3, 1...  
## $ WorkLifeBalance <fct> Bad, Better, Better, Better, Better, ...  
## $ YearsAtCompany <dbl> 6, 10, 0, 8, 2, 7, 1, 1, 9, 7, 5, 9, ...  
## $ YearsInCurrentRole <dbl> 4, 7, 0, 7, 2, 7, 0, 0, 7, 7, 4, 5, 2...  
## $ YearsSinceLastPromotion <dbl> 0, 1, 0, 3, 2, 3, 0, 0, 1, 7, 0, 0, 4...  
## $ YearsWithCurrManager <dbl> 5, 7, 0, 0, 2, 6, 0, 0, 8, 7, 3, 8, 3...

# Recommendation Function

data <- train\_readable\_tbl  
employee\_number <- 19  
  
recommend\_function <- function(data, employee\_number){  
 data %>% filter(EmployeeNumber == employee\_number) %>% mutate\_if(is.factor, as.numeric) %>%  
   
 #Personal Development Strategy  
 mutate(  
 personal\_development\_strategy = case\_when(  
 PerformanceRating == 1 | JobSatisfaction == 1 |   
 JobInvolvement <= 2 ~ "Create Personal Development Plan",  
 YearsAtCompany < 3 | TotalWorkingYears < 6  
 ~ "Promote Training and Formation",  
 (YearsInCurrentRole > 3 | YearsAtCompany >= 5) & PerformanceRating >=3 &   
 JobSatisfaction ==4 ~ "Seek Mentorship Role",  
 JobInvolvement >= 3 & JobSatisfaction >= 3 &   
 PerformanceRating >= 3 ~ "Seek Leadership Role", TRUE ~ "Retain and Maintain")) %>%   
 # Professional Development Strategy  
 mutate(  
 professional\_development\_strategy = case\_when(  
 YearsInCurrentRole >= 2 & JobSatisfaction <=2 ~ "Ready for Rotation",  
 JobLevel == 1 & YearsInCurrentRole >= 2 & JobInvolvement >= 3 &   
 PerformanceRating >= 3 ~ "Ready for Promotion",  
 JobLevel == 2 & YearsInCurrentRole >= 2 & JobInvolvement >= 4 &   
 PerformanceRating >= 3 ~ "Ready for Promotion",  
 JobLevel == 3 & YearsInCurrentRole >= 3 & JobInvolvement >= 4 &   
 PerformanceRating >= 3 ~ "Ready for Promotion",  
 JobLevel == 4 & YearsInCurrentRole >= 4 & JobInvolvement >= 4 &   
 PerformanceRating >= 3 ~ "Ready for Promotion",  
 YearsInCurrentRole >= 4 & JobSatisfaction >= 4 &   
 PerformanceRating >= 3 ~ "Incentivizes Specialization",  
 TRUE ~ "Retain and Maintain")) %>%   
 # Work Environment Strategy  
 mutate(  
 work\_environment\_strategy = case\_when(  
 OverTime ==2 | WorkLifeBalance ==1 ~ "Improve Work-Life Balance",   
 (BusinessTravel ==3 | DistanceFromHome >= 10) &   
 WorkLifeBalance ==2 ~ "Monitor Business Travel",  
 # WorkLifeBalance = 2 to differentiate from the case\_when above  
 EnvironmentSatisfaction ==1 & YearsInCurrentRole >=2 ~ "Review Job Assignment",  
 JobInvolvement <=2 ~ "Promote Job Involvement", TRUE ~ "Retain and Maintain"))%>%   
 select(EmployeeNumber, personal\_development\_strategy,   
 professional\_development\_strategy, work\_environment\_strategy)}

Test the recommendation algorithm.

train\_readable\_tbl %>% select(EmployeeNumber)

## # A tibble: 1,250 x 1  
## EmployeeNumber  
## <dbl>  
## 1 1  
## 2 2  
## 3 4  
## 4 5  
## 5 7  
## 6 8  
## 7 10  
## 8 11  
## 9 12  
## 10 13  
## # ... with 1,240 more rows

train\_readable\_tbl %>% recommend\_function(1)

## # A tibble: 1 x 4  
## EmployeeNumber personal\_developme~ professional\_devel~ work\_environment~  
## <dbl> <chr> <chr> <chr>   
## 1 1 Seek Mentorship Ro~ Incentivizes Speci~ Improve Work-Lif~

train\_readable\_tbl %>% recommend\_function(2)

## # A tibble: 1 x 4  
## EmployeeNumber personal\_developme~ professional\_devel~ work\_environment~  
## <dbl> <chr> <chr> <chr>   
## 1 2 Create Personal De~ Ready for Rotation Promote Job Invo~

train\_readable\_tbl %>% recommend\_function(4)

## # A tibble: 1 x 4  
## EmployeeNumber personal\_developme~ professional\_devel~ work\_environment~  
## <dbl> <chr> <chr> <chr>   
## 1 4 Create Personal De~ Retain and Maintain Improve Work-Lif~

train\_readable\_tbl %>% recommend\_function(3)# There is no employee3 - need to provide error capture

## # A tibble: 0 x 4  
## # ... with 4 variables: EmployeeNumber <dbl>,  
## # personal\_development\_strategy <chr>,  
## # professional\_development\_strategy <chr>,  
## # work\_environment\_strategy <chr>

test\_readable\_tbl %>% select(EmployeeNumber)

## # A tibble: 220 x 1  
## EmployeeNumber  
## <dbl>  
## 1 228  
## 2 1278  
## 3 1250  
## 4 2065  
## 5 1767  
## 6 1308  
## 7 18  
## 8 460  
## 9 1369  
## 10 1040  
## # ... with 210 more rows

test\_readable\_tbl %>% recommend\_function(1767)

## # A tibble: 1 x 4  
## EmployeeNumber personal\_developme~ professional\_devel~ work\_environment~  
## <dbl> <chr> <chr> <chr>   
## 1 1767 Create Personal De~ Retain and Maintain Improve Work-Lif~