Navigating Insurance Claim Data Through Tidymodels Universe

Future of Insurance

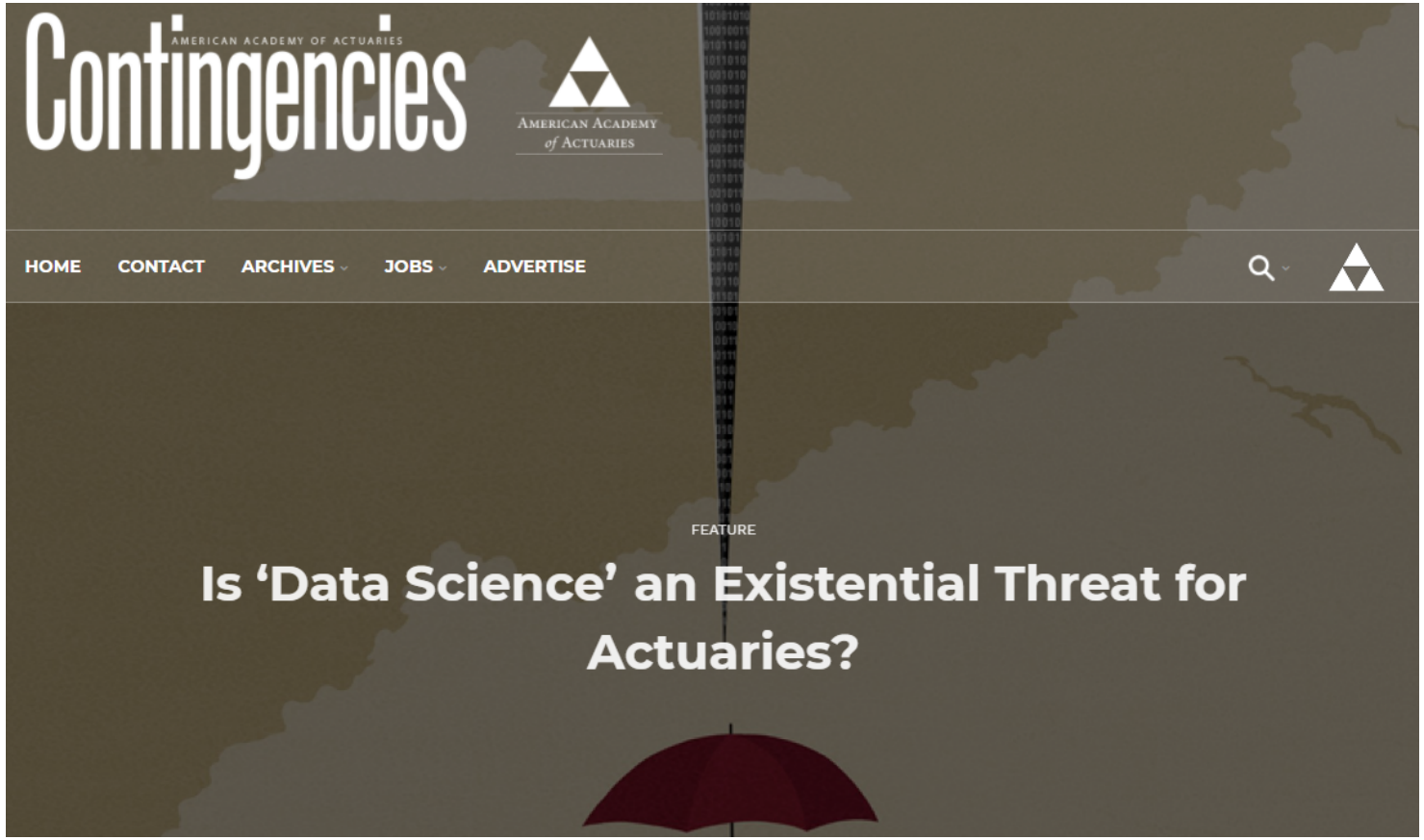
Jasper LOK

Singapore Actuarial Society

26 Aug 2021


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Are data scientists taking over actuaries?



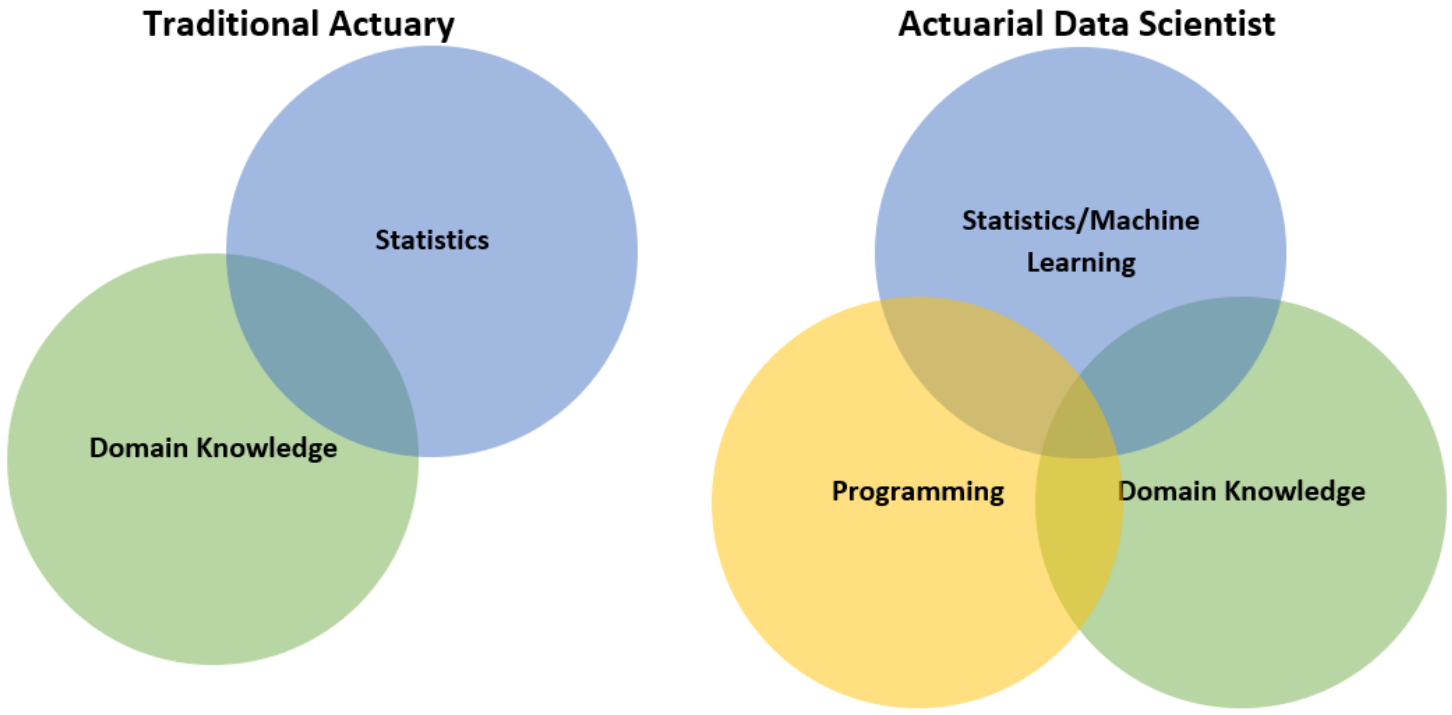
Source: Contingencies Volume 32 No 1

Should actuaries start looking for other jobs in other industries?

Relax! Most articles agree that data scientists are not likely to take over actuaries due to the complexity of actuarial science.

In general, these articles also pointed out there are much more actuaries could learn from data scientists to assist us in our actuarial analysis.

Additional Skillset for Actuaries to acquire



Modified the graph by Quantee

It is not just about learning the programming language, but also what are the best practice of performing machine learning tasks (eg. what is the suitable package to use to perform the necessary analysis).

SMU
SINGAPORE MANAGEMENT UNIVERSITY

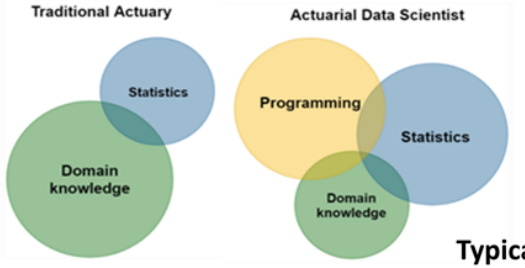
When 1 + 1 > 2

How Modern Data Science Could Complement Actuarial Science in Claim Cost Estimation

Jasper LOK Jun Haur **Supervisor: Prof. KAM Tin Seong**

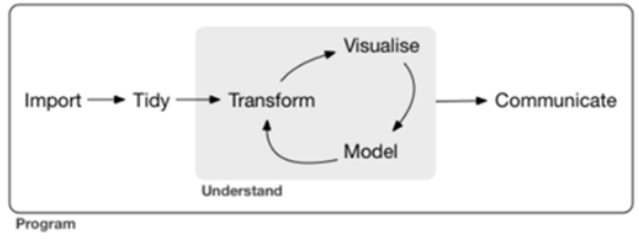
INTRODUCTION

The increasing ability to store and analyze the data due to the advancement in technology has provided actuaries opportunities in optimizing capital held by insurance companies. The modern data science packages also allows the actuaries harass the power of data science to mine the "gold" in unstructured data. This would help the companies in gaining competitive advantage in the market.



Analysis tools used:

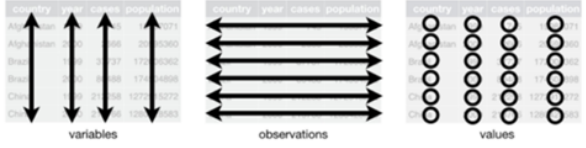
Typical Data Science Workflow:



MOTIVATION

In this capstone project, various modern data science techniques are used to demonstrate how they provides the actuaries another toolkit to sharpen the conventional actuarial analysis. **tidyverse** and **tidymodels** packages are being selected for the demonstration.

TIDY DATA CONCEPT

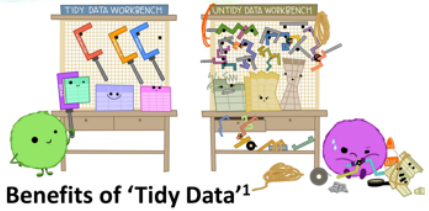


Following are the definition of tidy data:

- Each variable must have its column
- Each observation must have its row
- Each value must have its cell

EXPLANATORY DATA ANALYSIS (EDA)

Various R packages from **tidyverse** is used to import & clean data, check data quality and perform statistical test. The main benefit of using these packages is that they conform to tidy data concept, allowing the users effectively perform necessary data analysis, without spending too much time on data transformation.

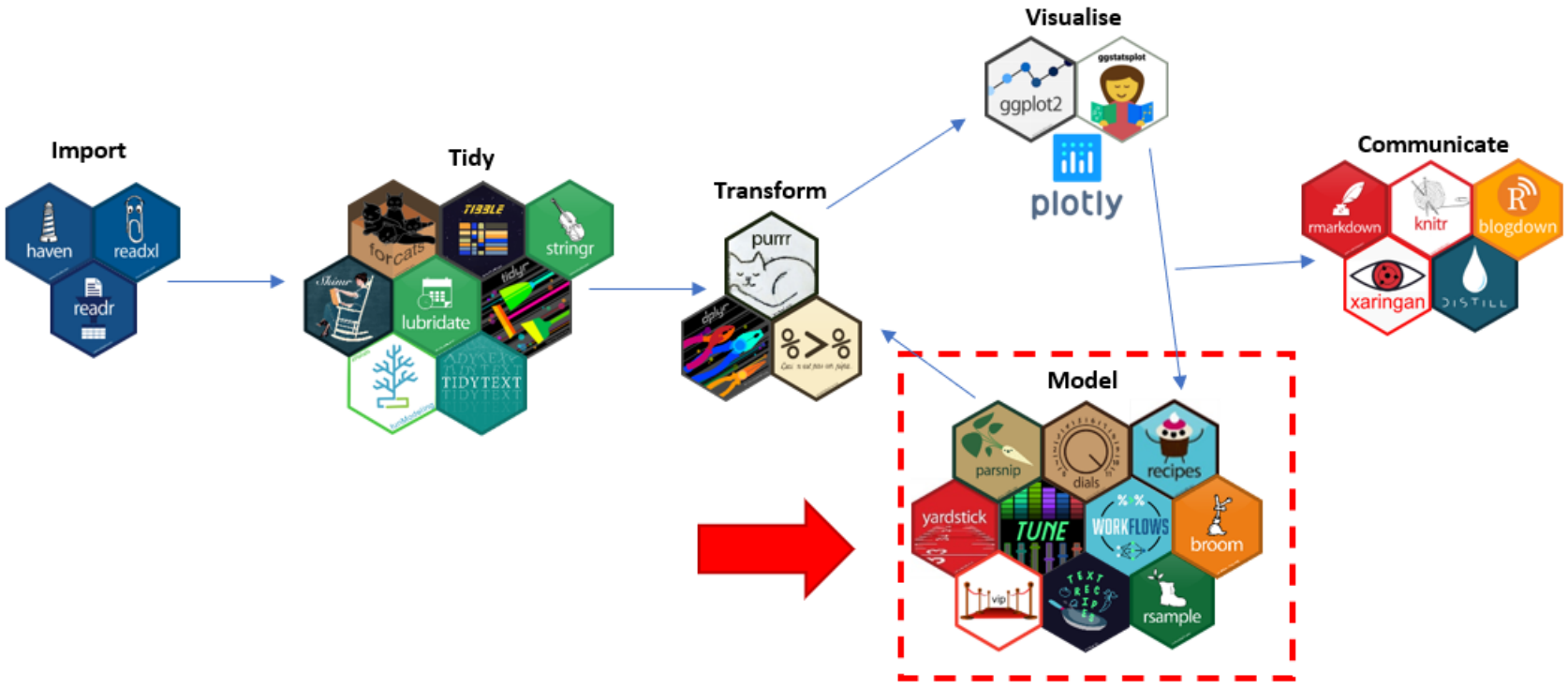


Benefits of 'Tidy Data'[1]

MODEL BUILDING

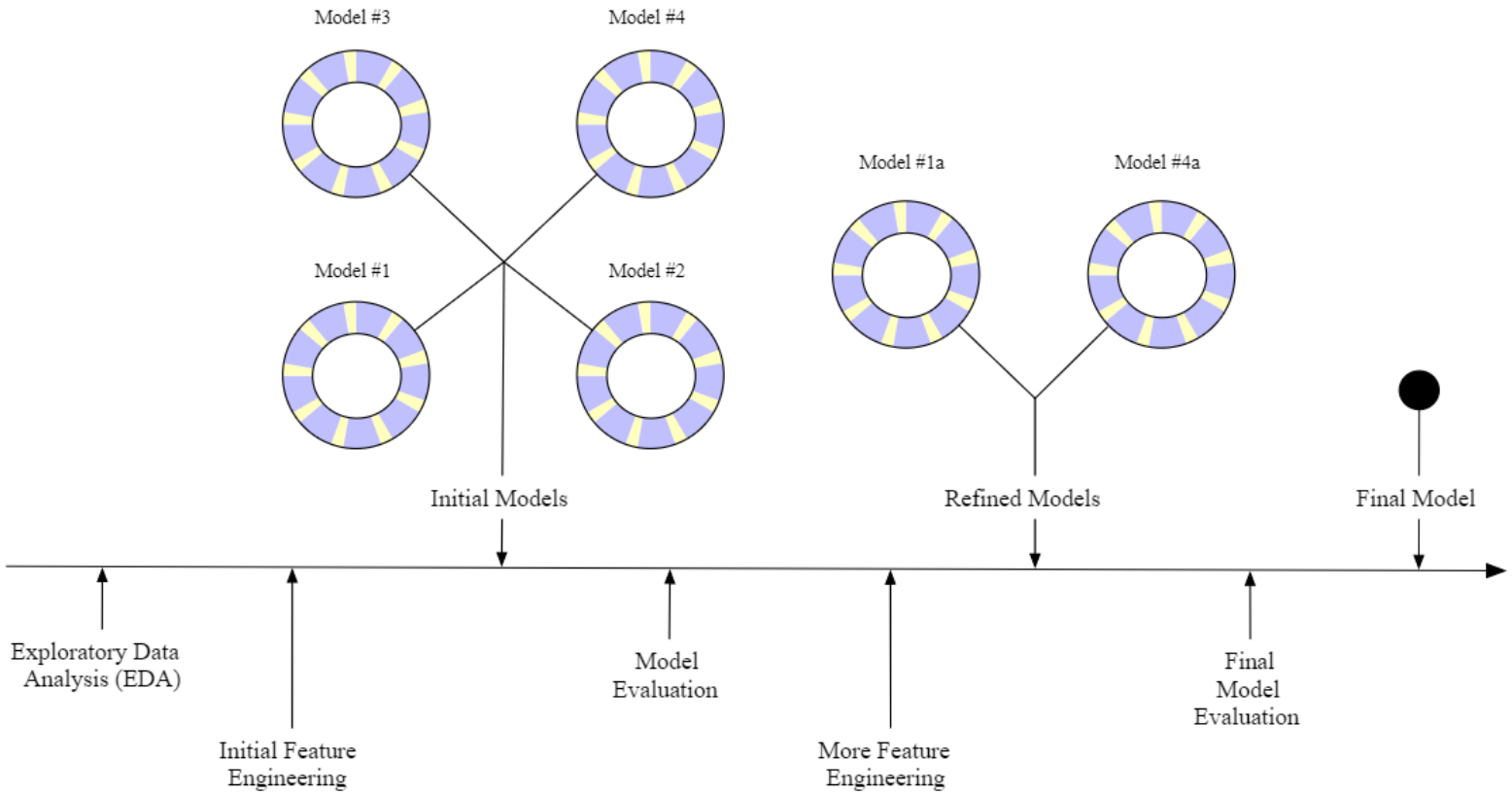
Screen shot of the poster from the research project during my Master program

Typical Data Science Project



The packages shown above are designed to work together, instead of some loosely designed packages.

Typical Modeling Process



Source: *Chapter 3.3 of Tidy Modeling with R*

Issue with open-source software

Function	Package	Code
`lda`	MASS	`predict(object)`
`glm`	stats	`predict(object, type = "response")`
`gbm`	gbm	`predict(object, type = "response", n.trees)`
`mda`	mda	`predict(object, type = "posterior")`
`rpart`	rpart	`predict(object, type = "prob")`
various	RWeka	`predict(object, type = "probability")`
`logitboost`	LogitBoost	`predict(object, type = "raw", nIter)`
`pamr.train`	pamr	`pamr.predict(object, type = "posterior")`

Source: Chapter 3.3 of Tidy Modeling with R

This can be a stumbling block for users to use R to perform machine learning analysis.

Sometimes the issues can happen within the same package as well 

Over here, we will take a look at this **glmnet** example shared by Max Kuhn during his **tidymodels** sharing at Cleveland R User Group.

glmnet is a package that allows one to fit a regularized generalized linear models. The prediction output from this package can come in various forms.

glmnet Class Predictions

```
predict(two_class_mod, newx = new_x, type = "class")
```

```
##          s0  s1  s2
## sample_1 "a" "b" "b"
## sample_2 "a" "b" "b"
```

glmnet Class Probabilities (Two Classes)

```
predict(two_class_mod, newx = new_x, type = "response")
```

```
##            s0  s1        s2
## sample_1 0.5 0.5 0.5059110
## sample_2 0.5 0.5 0.5261249
```

Now, the **predict** function returns a matrix of probability for the second level of outcome factor.

glmnet Class Probabilities (Three Classes)

```
predict(three_class_mod, newx = new_x,
        type = "response")
```

```
## , , s0
##
##                   a         b         c
## sample_1 0.3333333 0.3333333 0.3333333
## sample_2 0.3333333 0.3333333 0.3333333
##
## , , s1
##
##                   a         b         c
## sample_1 0.3333333 0.3333333 0.3333333
## sample_2 0.3333333 0.3333333 0.3333333
##
## , , s2
##
##                   a         b         c
## sample_1 0.3727891 0.2441238 0.3830872
## sample_2 0.3269560 0.3388499 0.3341941
```

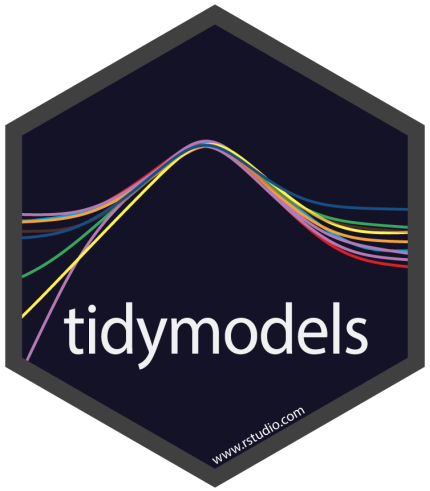
The output is no longer in matrix format. It is a 3D array format. Fainted. Often, the users would spend quite a fair bit of time in transforming the output into the required format of the next function.

Perhaps illustrating the output in such format would be better?

```
## # A tibble: 6 x 4
##       a     b     c lambda
##   <dbl> <dbl> <dbl>  <dbl>
## 1 0.333 0.333 0.333   1
## 2 0.333 0.333 0.333   1
## 3 0.333 0.333 0.333   0.1
## 4 0.333 0.333 0.333   0.1
## 5 0.373 0.244 0.383   0.01
## 6 0.327 0.339 0.334   0.01
```

Tidymodels to the rescue!

The author of **caret** package (ie. Max Kuhn) felt that there should be a better approach to perform machine learning tasks.



Tidymodels is a collection of various machine learning packages, from data pre-processing to model building & model comparison. The package provides an unified interface to perform machine learning tasks.

Instead of reinventing the "wheels", the package functions as a wrapper to wrap around existing package.

Pre-Process → Train → Validate

Source: A Gentle Introduction to tidymodels

Above are the various key packages that assist us in our various machine learning activities.

These packages are recent project by RStudio to look into how the packages could better support users in the machine learning analysis.

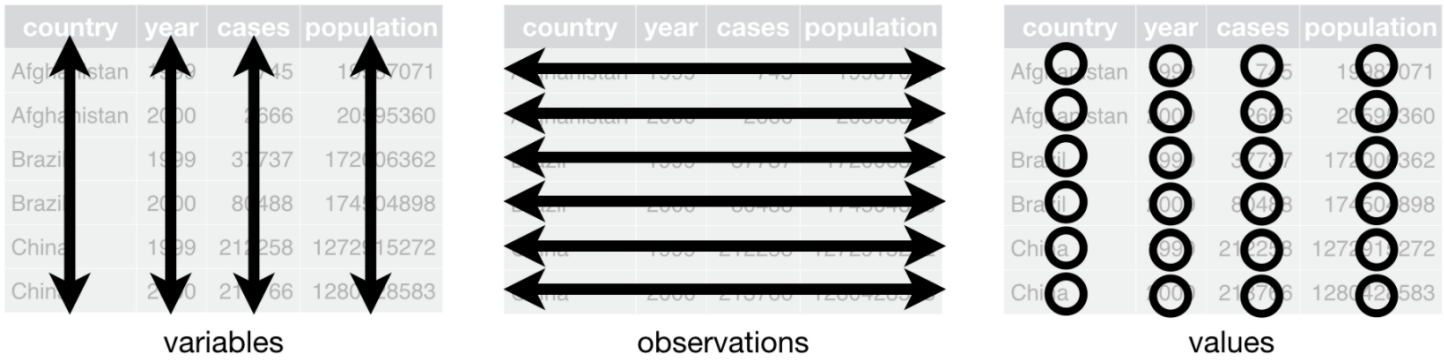
The cool thing about these packages are following *tidy data* concepts.

What do you mean by tidy data?

A concept introduced by Hadley Wickham, Chief Scientist at R Studio.

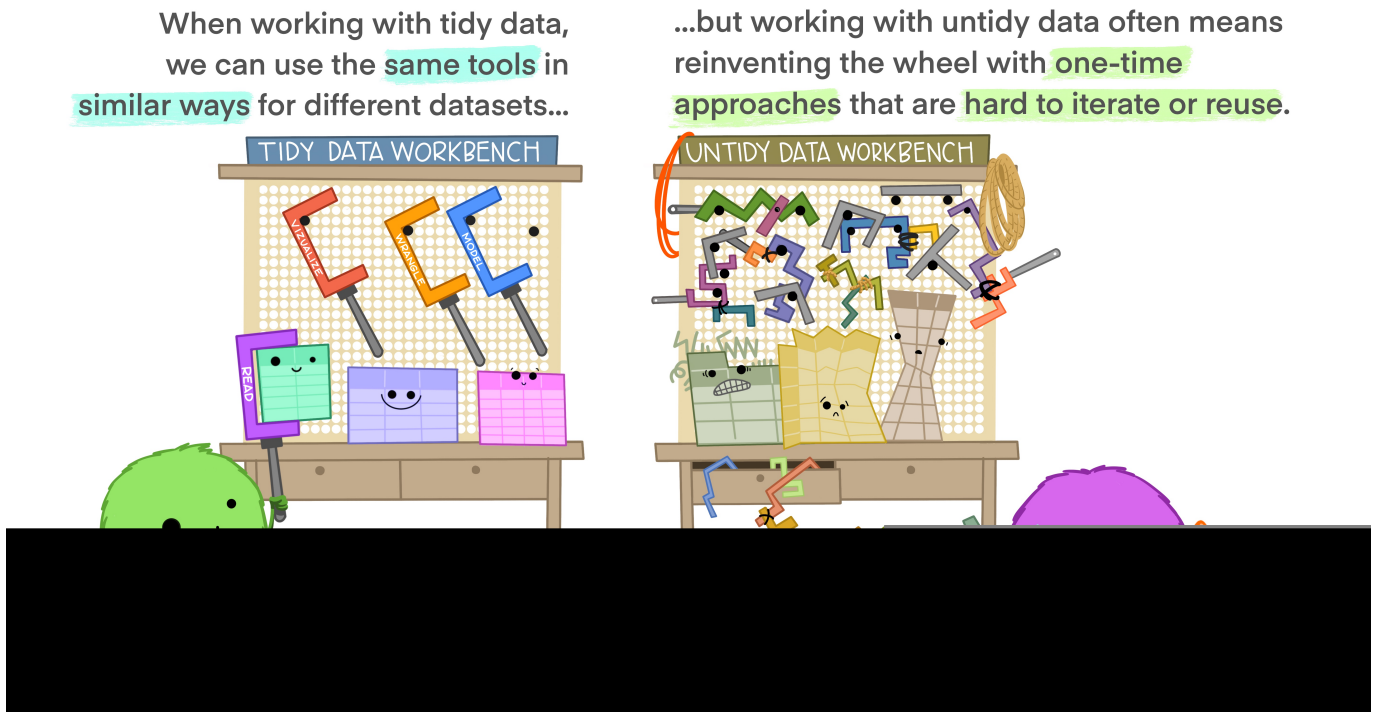
Below are the definition of tidy data:

- Each variable must have its column
- Each observation must have its row
- Each value must have its cell



Source: *Chapter 12.2 of R for Data Science*

Hmmm, but why is this important?



Source: stats-illustrations by Allison Horst

Setting Context for the Actuarial Use Case

For the demonstration, I will be using the worker compensation insurance claim dataset from Kaggle https://www.kaggle.com/c/actuarial-loss-estimation/.

Below is the snapshot of the dataset:

Show 3 entries Search:

	ClaimNumber	DateTimeOfAccident	DateReported	Age	Gender	MaritalStatus	DependentChildren	DependentsOther
1	WC8285054	2002-04-09T07:00:00Z	2002-07-05T00:00:00Z	48	M	M	0	0
2	WC6982224	1999-01-07T11:00:00Z	1999-01-20T00:00:00Z	43	F	M	0	0
3	WC5481426	1996-03-25T00:00:00Z	1996-04-14T00:00:00Z	30	M	U	0	0

Showing 1 to 3 of 100 entries Previous 1 2 3 4 5 … 34 Next

Data Splitting

One of the initial steps for any machine learning analysis is to split the dataset into training & testing dataset.

First, read in the clean dataset.

```
df <- read_csv("data/data_eda_actLoss_3.csv") %>%
  drop_na()
```

Next, use **initial_split** function to create a binary split of the data into training and testing set.

```
df_split <- initial_split(df,
                          prop = 0.6,
                          strata = init_ult_diff)
```

training function & **testing** function are used to extract the relevant data.

```
df_train <- training(df_split)
df_test <- testing(df_split)
```

Data Pre-processing

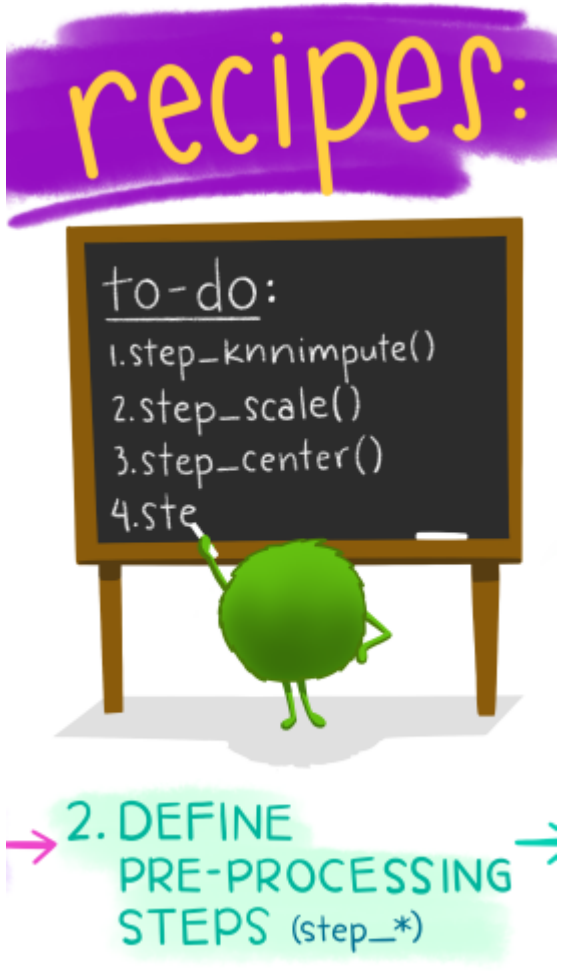
Source: stats-illustrations by Allison Horst

Following is the example of how the **recipe** looks like in **tidymodels**:

In line 1, I have specified the formula & dataset.

```
gen_recipe <- recipe(init_ult_diff ~ .,
                     data = df_train)
```

Following is the example of how the **recipe** looks like in **tidymodels**:



Step_date function allows users to extract year, month and day of the week from the date variable.

```
gen_recipe <- recipe(init_ult_diff ~ .,
                     data = df_train) %>%
  step_date(c(DateTimeOfAccident,
              DateReported))
```

One also could perform data wrangling by using **step_mutate** function as shown under line 3 & line 4

```
gen_recipe <- recipe(init_ult_diff ~ .,
                     data = df_train) %>%
  step_date(c(DateTimeOfAccident,
              DateReported)) %>%
  step_mutate(DateTimeOfAccident_hr =
                hour(DateTimeOfAccident),
              DateTimeOfAccident_hr =
                factor(DateTimeOfAccident_hr,
                       order = TRUE))
```

Following is the example of how the **recipe** looks like in **tidymodels**:

Indicating the correct data type is also very important as incorrect data type would affect the model performance.

```
gen_recipe <- recipe(init_ult_diff ~ .,
                     data = df_train) %>%
  step_date(c(DateTimeOfAccident,
              DateReported)) %>%
  step_mutate(DateTimeOfAccident_hr =
                hour(DateTimeOfAccident),
              DateTimeOfAccident_hr =
                factor(DateTimeOfAccident_hr,
                       order = TRUE)) %>%
  update_role(c(DateTimeOfAccident,
                DateReported),
              new_role = "id") %>%
  prep()
```

This modeling approach has effectively allowed to modularize the different functions and chain them together.

A Peek into the Created Recipe

The code will show us the different steps we have specified under recipe step by calling the recipe object we have created.

```
gen_recipe
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##         id          2
##    outcome          1
##  predictor         16
##
## Training data contained 27892 data points and no missing data.
##
## Operations:
##
## Date features from DateTimeOfAccident, DateReported [trained]
## Variable mutation for DateTimeOfAccident_hr, DateTimeOfAccident_hr, DateTimeOfAccident_year, DateReported_year [trained]
```

Checking the data type is also a very crucial step before the modeling as inappropriate variable types might affect the performance of the models.

```
gen_recipe %>%
  summary()
```

```
## # A tibble: 26 x 4
##    variable           type    role      source
##    <chr>              <chr>   <chr>     <chr>
##  1 DateTimeOfAccident date    id        original
##  2 DateReported       date    id        original
##  3 Age                numeric predictor original
##  4 Gender             nominal predictor original
##  5 MaritalStatus      nominal predictor original
##  6 DependentChildren  numeric predictor original
##  7 DependentsOther    numeric predictor original
##  8 WeeklyWages        numeric predictor original
##  9 PartTimeFullTime   nominal predictor original
## 10 HoursWorkedPerWeek numeric predictor original
## # ... with 16 more rows
```

The **summary** function provides the users a quick overview of the data type.

The modularise structure allows us to effectively reuse the model components. For example, GLM model is unable to use categorical variables to fit the model.

So, to resolve this, we can add on additional data preprocessing step.

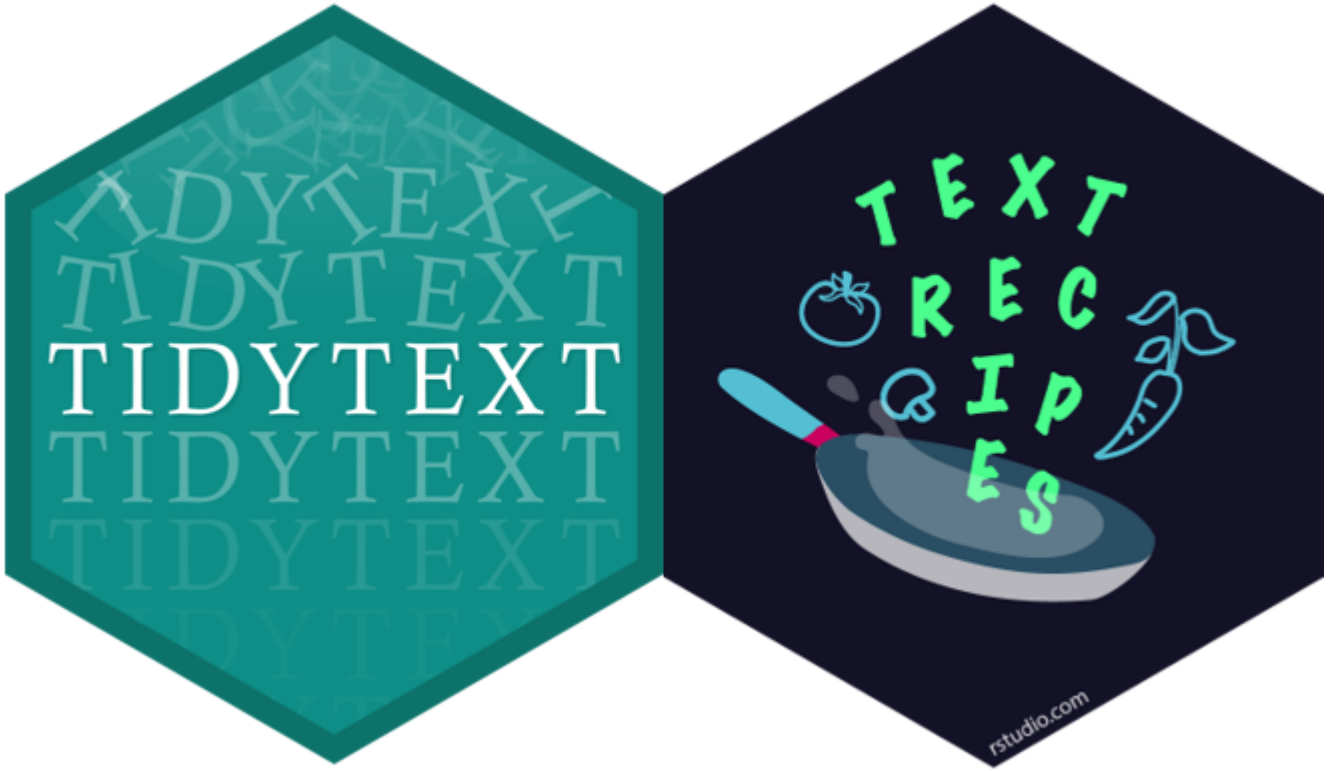
```
glmnet_recipe <- gen_recipe %>%
  step_dummy(all_nominal())
```

The code above indicates that one-hot ecoding method to be applied on all the categorical variables .

Wait! Don't throw away the text fields

Remember there is a claim description field within the data?

This is where **tidytext** package and **textrecipe** comes very handy.



tidytext

To extract the text, I will first split the words into tokens by using **unnest_token**.

```
tidy_clm_unigram <- data_1 %>%
  unnest_tokens(word, ClaimDescription,
                token = "ngrams",
                n = 1)
```

I will also remove the stopwords (eg. above, onto, and) from the tokens since they are not meaningful in the analysis.

```
tidy_clm_unigram <- data_1 %>%
  unnest_tokens(word, ClaimDescription,
                token = "ngrams",
                n = 1) %>%
  anti_join(get_stopwords())
```

Once the words are tokenized, we can perform frequency count on the words to understand which are the words tend to appear more frequent than the rest.

```
tidy_clm_unigram %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 3,449 x 2
##    word        n
##    <chr>   <int>
##  1 right   19839
##  2 left    18470
##  3 back    13438
##  4 strain  12124
##  5 lower    8170
##  6 finger   7884
##  7 hand     7145
##  8 struck   6868
##  9 lifting  6774
## 10 eye      5155
## # ... with 3,439 more rows
```

After that, we can create indicators for the more frequently appeared words to see whether the model performance would improve by including these features.

Once they are extracted, we can visualize the extracted texts by using word cloud.

```
cleaned_clm_unigram %>%
  filter(n > 2000) %>%
  ggplot(aes(label = word,
             size = n,
             color = n)) +
  geom_text_wordcloud() +
  scale_size_area(max_size = 20) +
  theme_minimal()
```

textrecipe

Alternatively, we can perform text mining through specifying the steps in the recipe.

Same as the previous recipe created, after specifying the 'recipe' for the machine learning model, we will specify what are the pre-processing steps the model should perform.

```
ranger_recipe_clmdesc <-
  recipe(formula = init_ult_diff ~ .,
         data = df_wClmDesc_train) %>%
  step_tokenize(ClaimDescription) %>%
  step_stopwords(ClaimDescription) %>%
  step_tokenfilter(ClaimDescription,
                   max_tokens = 20) %>%
  step_tfidf(ClaimDescription)
```

- This approach is consistent with recipe to "prepare" the text data for modeling
- It is easier to understand and clear at first glance what are we "preparing"

More explanation on the different text mining techniques and how they work, do check this book or this package.

Model Selection - Model X, I choose you!

Source: stats-illustrations by Allison Horst

Back to Actuaries' Most Beloved Model - GLM

Convention Approach

```
x <- df_train %>%
  dplyr::select(-init_ult_diff) %>%
  data.matrix()

y <- df_train$init_ult_diff


glmnet(x,
       y,
       family = "gaussian",
       nlambda = 10,
       alpha = 0.5)
```

Tidymodels Approach

```
glmnet_spec <-
  linear_reg(penalty = tune(),
             mixture = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet", family = "gaussian")
```

parsnip package provides users a more unified model interface without sacrificing the flexibility.

parsnip package do support a wide range of machine learning models.

Random Forest

```
ranger_spec <-
  rand_forest(mtry = tune(),
              min_n = tune(),
              trees = 50) %>%
  set_mode("regression") %>%
  set_engine("ranger",
             importance = "impurity")
```

XGBoost

```
xgboost_spec <-
  boost_tree(trees = tune(),
             min_n = tune(),
             tree_depth = tune(),
             learn_rate = tune(),
             loss_reduction = tune(),
             sample_size = tune()) %>%
  set_mode("regression") %>%
  set_engine("xgboost")
```

Linear Regression

```
lm_spec <- linear_reg() %>%
  set_mode("regression") %>%
  set_engine("lm")
```

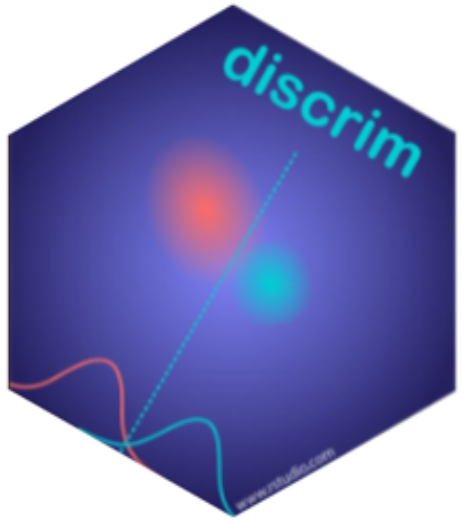
MARS

```
earth_spec <-
  mars(num_terms = tune(),
       prod_degree = tune(),
       prune_method = "none") %>%
  set_mode("regression") %>%
  set_engine("earth")
```

KNN

```
kknn_spec <-
  nearest_neighbor(neighbors = tune(),
                   weight_func = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn")
```

There are other tidymodels packages support different machine learning models...

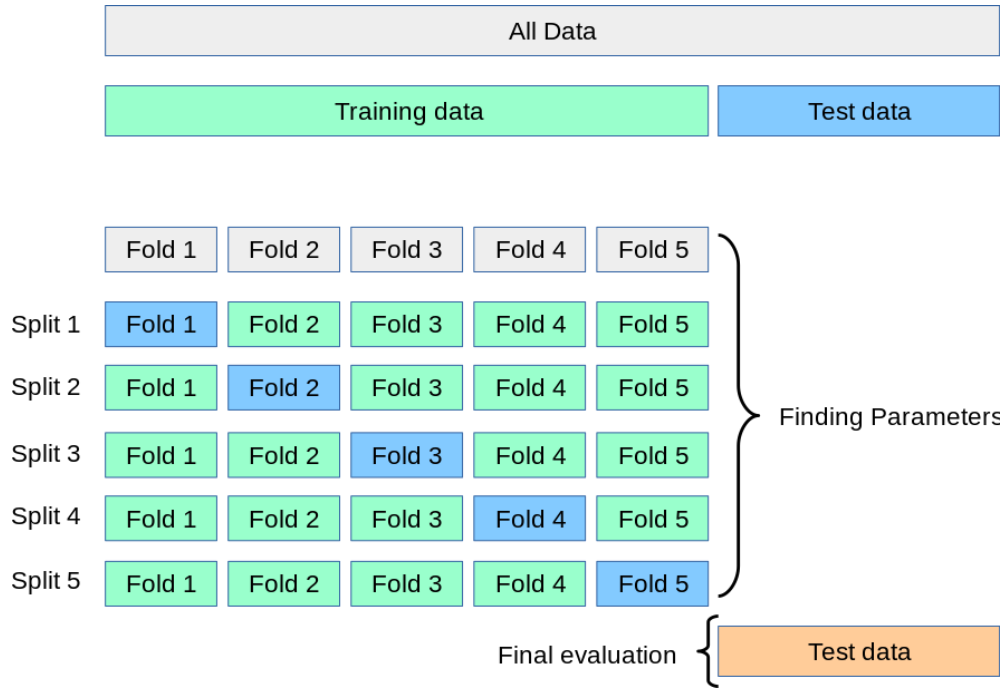


Model Tuning

Cross validation is commonly used to:

- Find the best set of parameters that would give us best model performance
- Prevent models from overfitting

Over here, I will be using k-fold validation to perform model tuning.



Source: Section 3.1 of Scikit-learn

To do so, we will create the dataset for k-fold cross validation by using **vfold_cv** function as shown below.

```
df_folds <- vfold_cv(df_train, strata = init_ult_diff)
df_folds
```

And yes we are almost there!

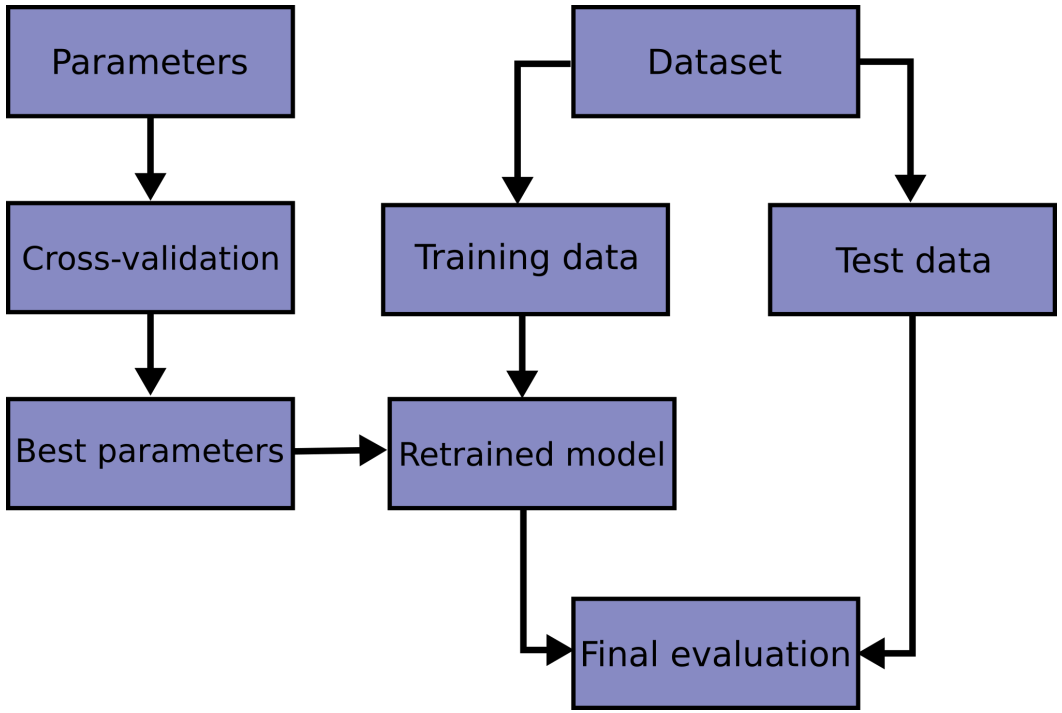
Over here, **grid search** is used to find the best parameters fit a given model.

```
ranger_tune <-
  tune_grid(ranger_workflow,
            resamples = df_folds,
            grid = 5)
```

```
ranger_fit <- ranger_workflow %>%
  finalize_workflow(select_best(ranger_tune)) %>%
  last_fit(df_split)
```

Once the model is tuned, **select_best** function is used to select the best parameters. Then, finalise the workflow & run the fitted model on testing data by using **last_fit** function.

If we were to visualize the flow, the modeling steps mentioned above would look something as following....



Source: Section 3.1 of Scikit-learn

Now, we can "chain" the different components as a workflow

Once the necessary parameters are setup, we will join the different steps together to form a workflow.

Conventional Approach

```
x <- df_train %>%
  dplyr::select(-init_ult_diff) %>%
  data.matrix()

y <- df_train$init_ult_diff


glmnet(x,
       y,
       family = "gaussian",
       nlambda = 10,
       alpha = 0.5)
```

tidymodels Approach

```
glmnet_workflow <-
  workflow() %>%
  add_recipe(glmnet_recipe) %>%
  add_model(glmnet_spec)
```

Need another workflow? Not an issue, sir

```
glmnet_workflow_ult <-
  glmnet_workflow %>%
  update_formula(UltimateIncurredClaimCost ~ .)
```

The created workflow can only be visualised by calling the workflow item.

```
glmnet_workflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
## * step_date()
## * step_mutate()
## * step_dummy()
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Main Arguments:
##   penalty = tune()
##   mixture = tune()
##
## Computational engine: glmnet
```

Such modularized modeling method allows us to reuse the different machine learning parts through the analysis.

Confused?? Worry not

usemodels package can be used to generate all the necessary modeling templates for the users.

```
use_ranger(formula, data)
```

```
ranger_recipe <-
  recipe(formula = init_ult_diff ~ ., data = df_train) %>%
  step_string2factor(one_of(MaritalStatus, PartTimeFullTime, injury_body, injury_side,
    injury_item, injury_cause, injury_type))

ranger_spec <-
  rand_forest(mtry = tune(), min_n = tune(), trees = 1000) %>%
  set_mode("regression") %>%
  set_engine("ranger")

ranger_workflow <-
  workflow() %>%
  add_recipe(ranger_recipe) %>%
  add_model(ranger_spec)

set.seed(93447)
ranger_tune <-
  tune_grid(ranger_workflow, resamples = stop("add your rsample object"), grid = stop("add number of candidate points"))
```

Model Comparison

To facilitate the model comparison, **yardstick** is being used to compare the different models.

Same as other packages under **tidymodels**, various model performance metrics under **yardstick** also have same interface, which allows us to loop through the various syntax to calculate the model performance.

First, I have defined the different measurements I need.

I will be using *root mean square error*, *R squared* and *mean absolute scaled error*.

Root Mean Square Error

```
rmse(data, truth, estimate, na_rm = TRUE, ...)
```

R Squared

```
rsq(data, truth, estimate, na_rm = TRUE, ...)
```

Mean Absolute Scaled Error

```
mase(data, truth, estimate, m = 1L, mae_train = NULL, na_rm = TRUE, ...)
```

As shown above, the metric interfaces look very similar. This allows to calculate the model performance by looping through the different metrics.

```
model_metrics <- metric_set(rmse, rsq, mase)
```

```
ranger_pred <- ranger_fit %>%
  collect_predictions()

ranger_metric <- model_metrics(ranger_pred,
                               truth = init_ult_diff,
                               estimate = .pred) %>%
  mutate(model = "ranger") %>%
  pivot_wider(names_from = .metric,
              values_from = .estimate)
```

Next, I will join these model results into a tibble table so that I can compare how does the different models perform under different scenarios.

Showdown by Different Models

Following are the performance metrics of ddifferent models:

```
## # A tibble: 6 x 5
##   .estimator model    rmse   rsq  mase
##   <chr>      <chr>   <dbl> <dbl> <dbl>
## 1 standard   ranger  1537. 0.457 0.490
## 2 standard   xgboost 1577. 0.441 0.511
## 3 standard   glmnet  1656. 0.368 0.564
## 4 standard   lm      1695. 0.338 0.569
## 5 standard   earth   1708. 0.328 0.593
## 6 standard   kknn    1758. 0.288 0.581
```

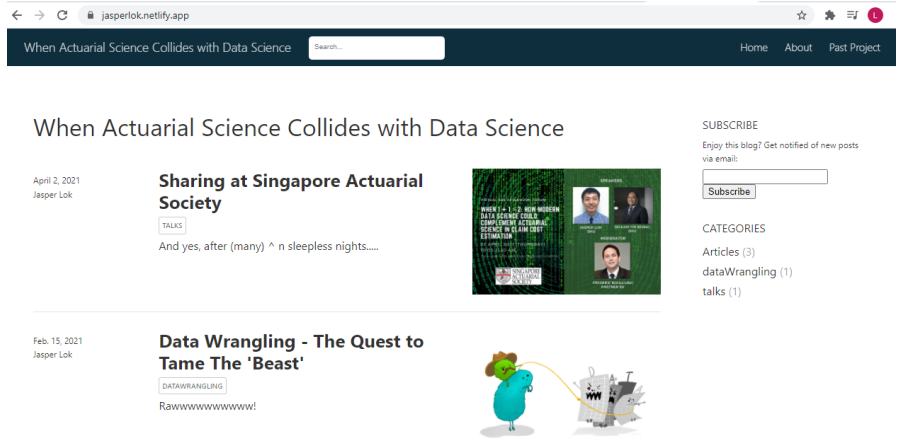
In general, the different model performance results are quite consistent with one another.

Out of all the models, random forest has the highest accuracy.

This is just a very small subset of the entire research

The full research paper and R code are published on my Github https://github.com/jasperlok/SMU_Capstone.

This deck of interactive slides will be shared on my data science blog, When Actuarial Science Collides with Data Science.



FAQ

Thank you!

Jasper LOK

Email: junhaur.lok.2019@mitb.smu.edu.sg
Profile: linkedin.com/in/jasper-l-13426232/
Blog: https://jasperlok.netlify.app/

Appendix

Link to the relevant R packages

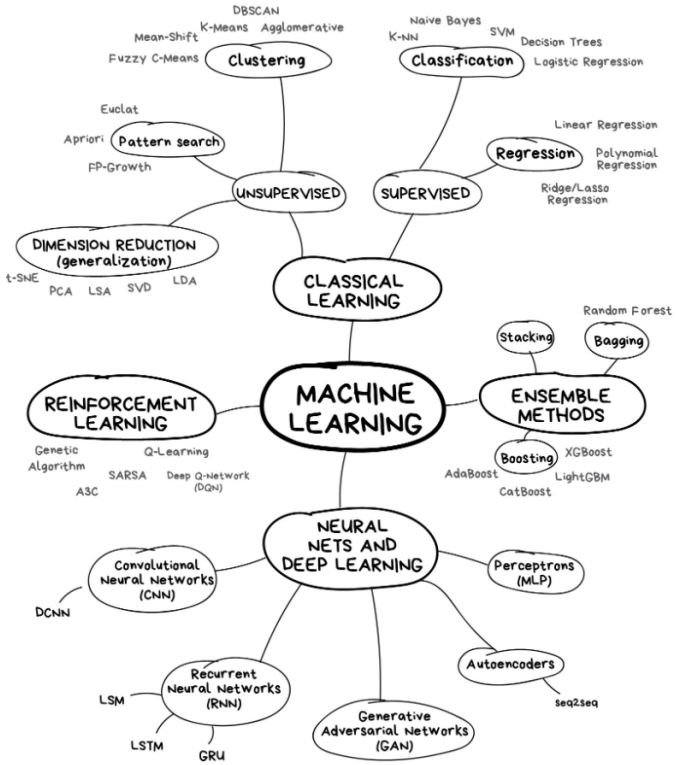
Tidyverse https://www.tidyverse.org/

Tidymodels https://www.tidymodels.org/

Variable Importance Plot https://koalaverse.github.io/vip/index.html

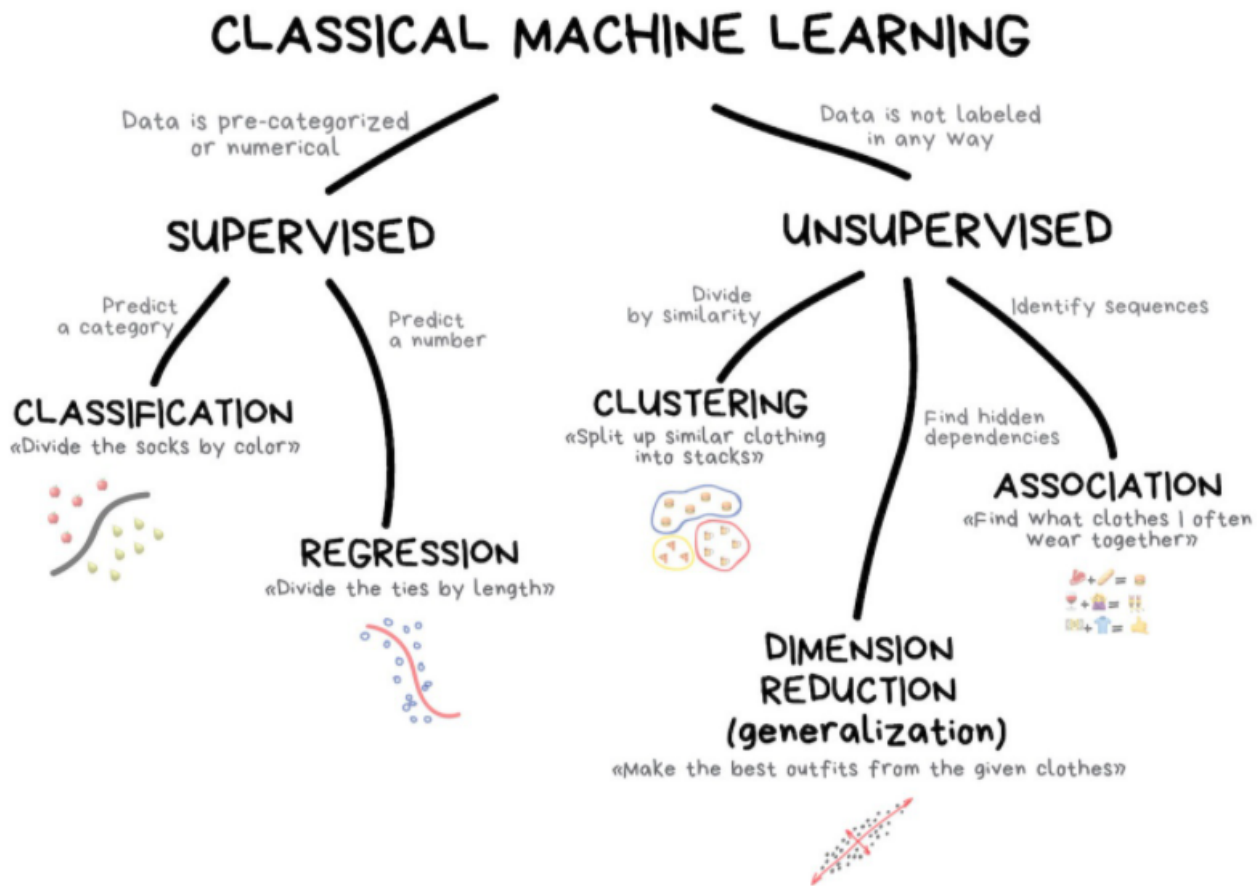
Partial Dependence Plot https://bgreenwell.github.io/pdp/index.html

Overview of Machine Learning



Source: xkcd blog

Overview of Classical Machine Learning



Source: xkcd blog

Further reading

R for Data Science

https://r4ds.had.co.nz/

Tidymodeling with R

https://www.tmwr.org/

Feature Engineering

https://bookdown.org/max/FES/

Text Mining

https://www.tidytextmining.com/

Interpretable Machine Learning

https://christophm.github.io/interpretable-ml-book/