

Sharing at Singapore Actuarial Society

When $1 + 1 > 2$: How Modern Data Science could
Complement Actuarial Science in Claim Cost
Estimation

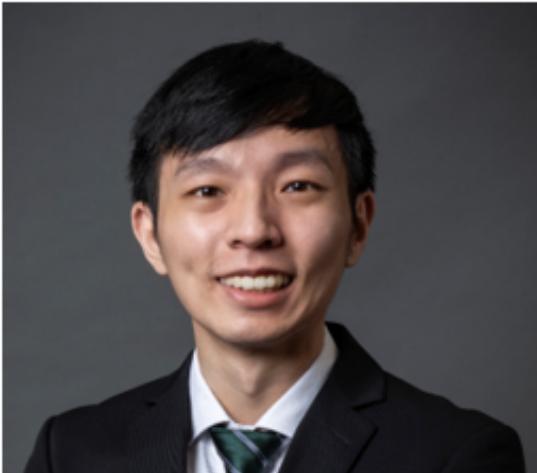
Jasper LOK

Professor KAM Tin Seong

Singapore Management University

22 Apr 2021

Speakers' Bio



Jasper LOK

- Worked in both life + GI insurance companies for more than 8 years
- Final year student in SMU Master of IT in Business (Analytics)
- In the pipeline to become a junior data scientist



Professor KAM Tin Seong

- Professor in SMU
- More than 20 years of experience in both the analytics industry & academia
- His research interests include customer analytics, visual analytics, and geospatial analytics

THIS IS YOUR MACHINE LEARNING SYSTEM?

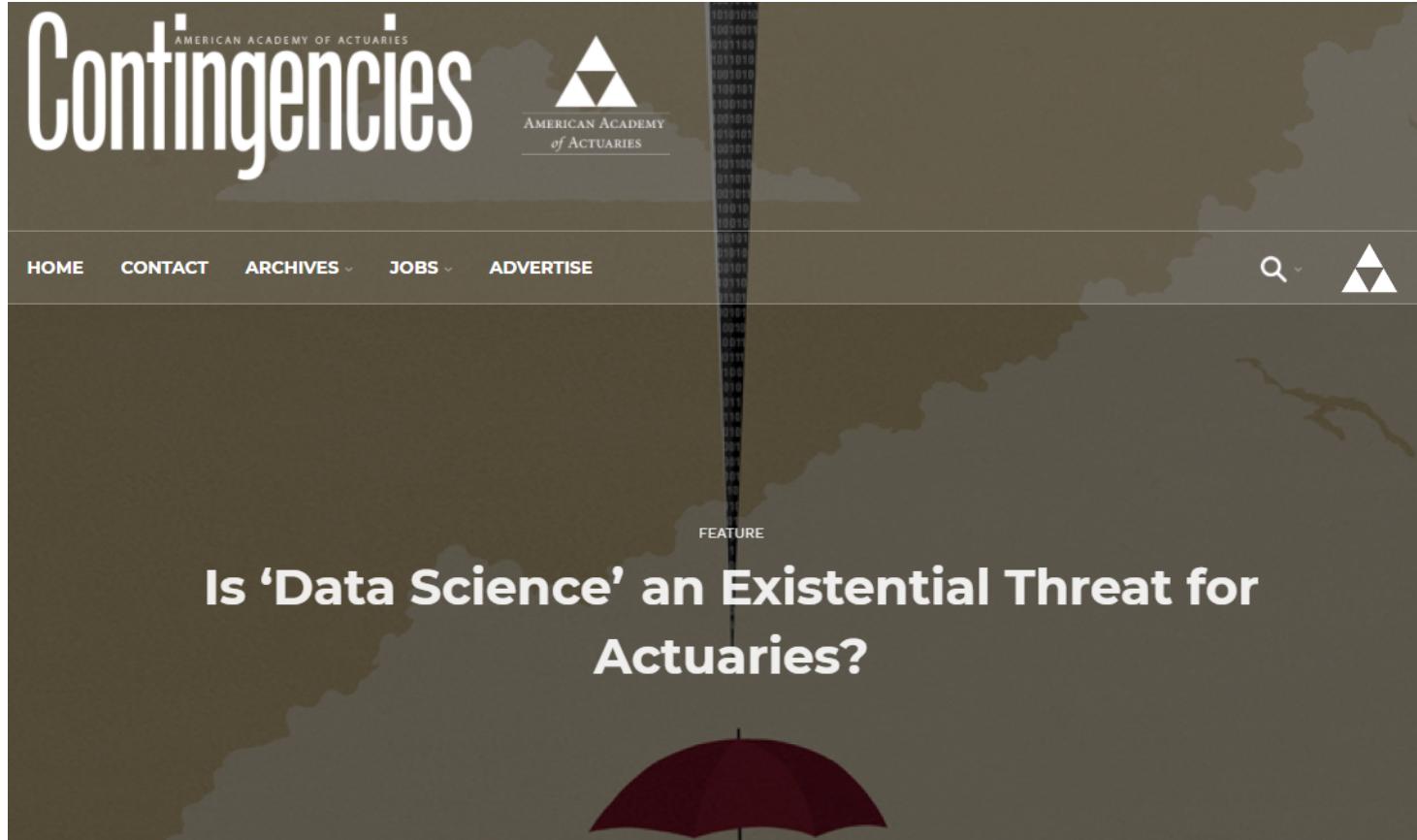
YUP! YOU POUR THE DATA INTO THIS BIG
PILE OF LINEAR ALGEBRA, THEN COLLECT
THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL
THEY START LOOKING RIGHT.



Are data scientists taking over actuaries?



Source: *Contingencies* Volume 32 No 1

Should actuaries start looking for other jobs in other industries?

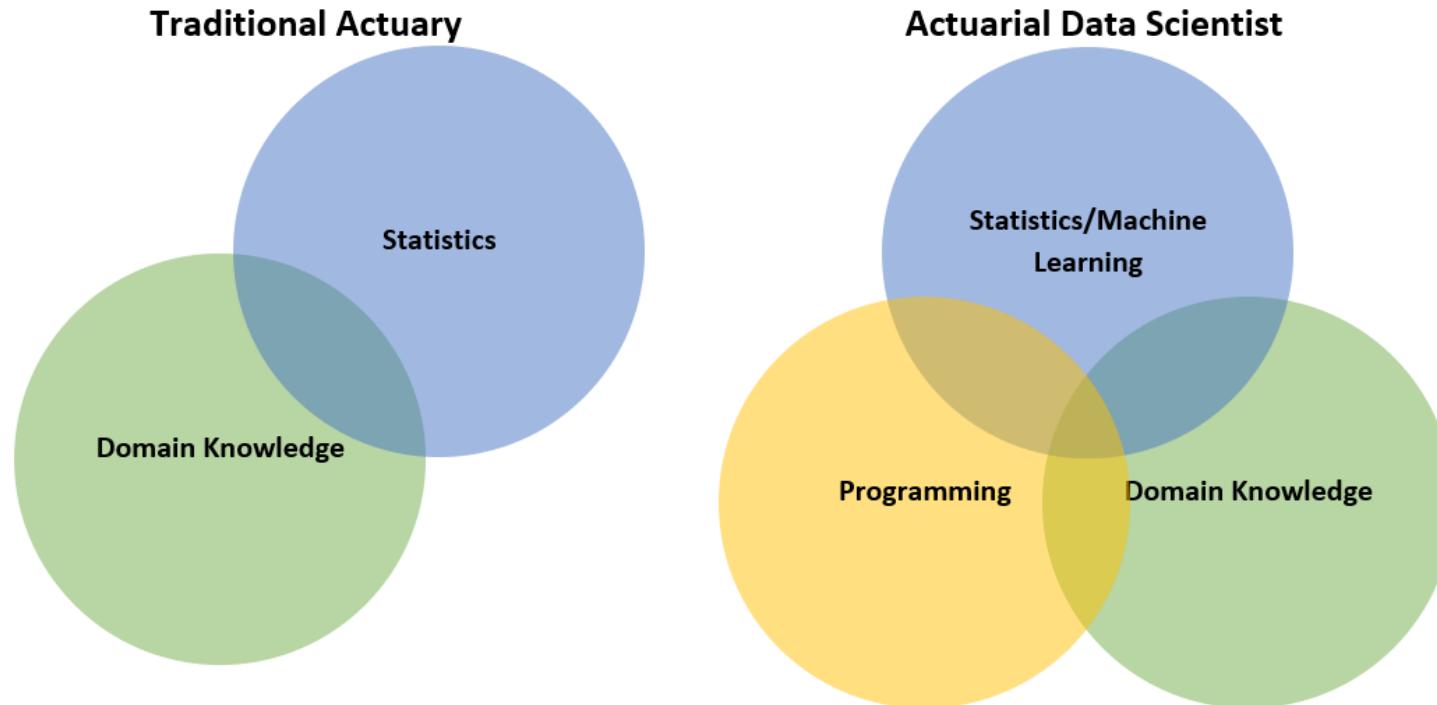
Relax! Most articles agree that data scientists are not likely to take over actuaries due to the complexity of actuarial science.

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

List of benefits data science could bring to actuarial task

Benefits	Descriptions
Improved data quality	Machine learning is a key driver for companies to improve data capture and storage
New data sources	Machine learning potentially opens up opportunities for actuaries to explore alternative data sources
Speed of analysis	Machine learning models can generally be fitted and validated in a short space of time
New modeling techniques	Utilising alternative modeling approaches allows different perspectives to be gained on data
New Approaches to Problems	Produce a wider variety of models quickly - better ability to select the appropriate modeling approach for a given problem
Improved Data Visualisations	Increasing power to produce stunning visualisations of data which can itself provide new perspectives on a task

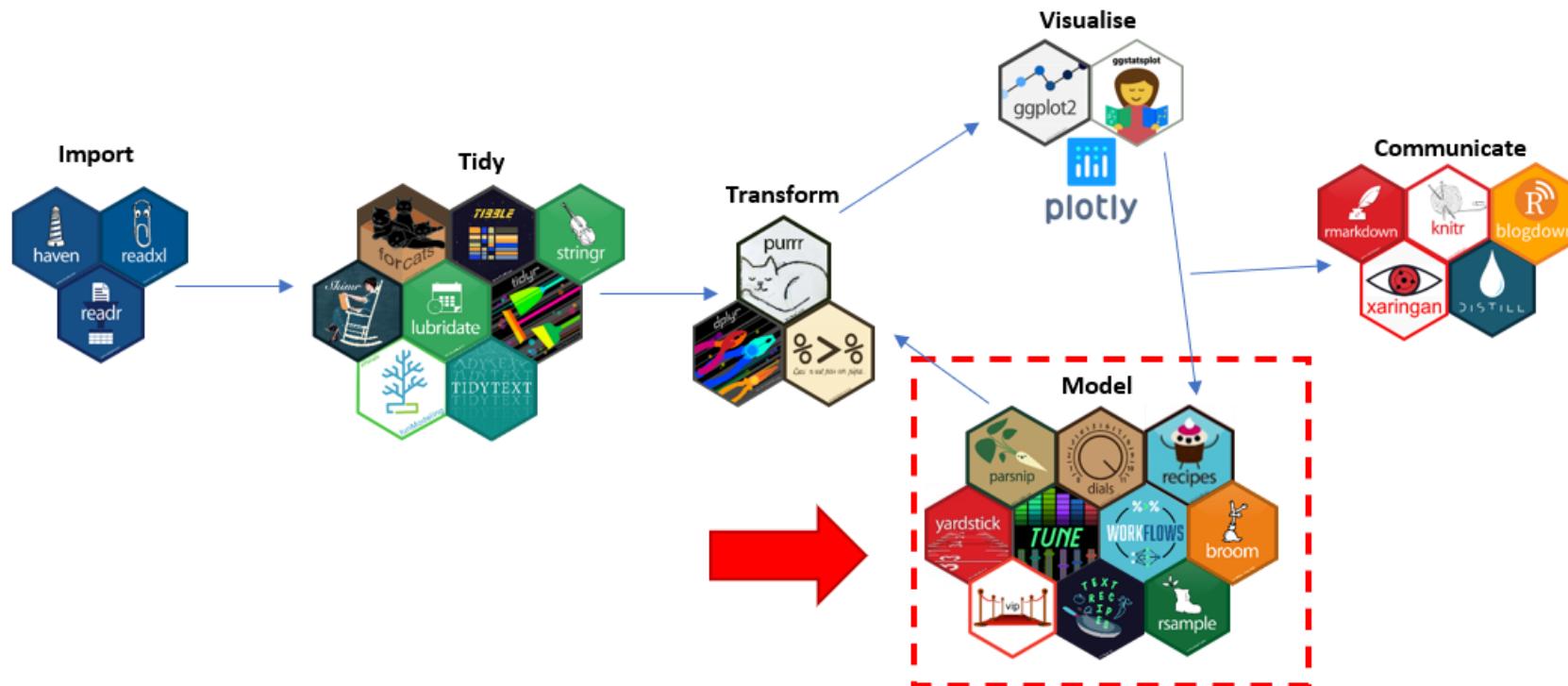
Additional Skillset for Actuaries to acquire



Modified the graph by *Quantee*

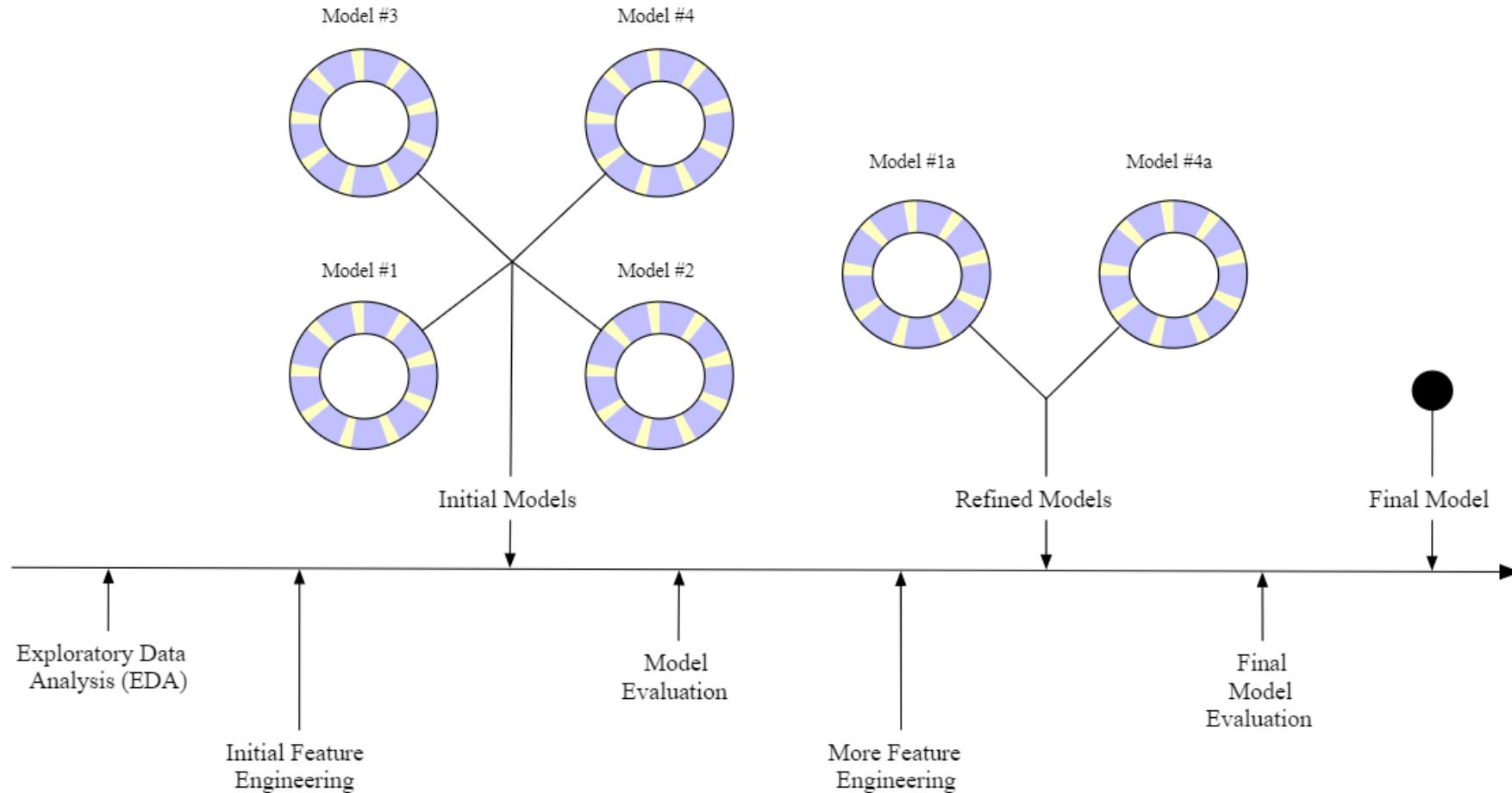
It is not just about learning the programming language, but also what are the best practice of performing machine learning tasks (eg. what is the suitable package to use to perform the necessary analysis).

Typical Data Science Project



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

Typical Modeling Process



Issue with open-source software

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

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
```

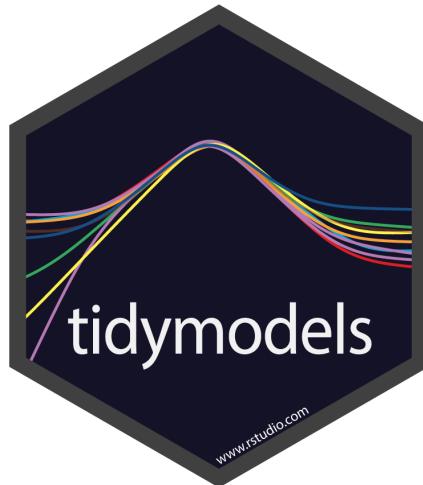
The output is no longer in matrix format. It is a 3D array format. Fainted. Often, the users would spend quite a fair bit of time in transforming the output into the required format of the next function.

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 projects 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.

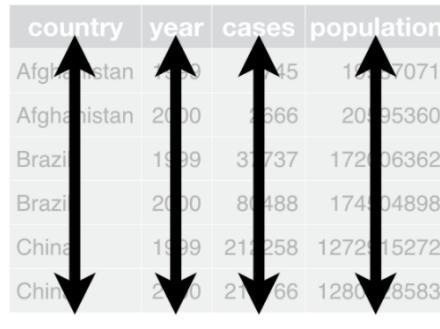
What do you mean by tidy data?

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

Below are the definition of tidy data:

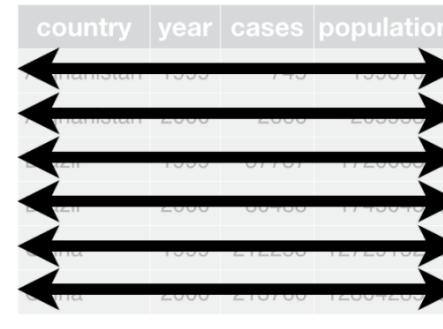
- Each variable must have its column
- Each observation must have its row
- Each value must have its cell

country	year	cases	population
Afghanistan	1990	745	1908071
Afghanistan	2000	2666	2059360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272015272
China	2000	21666	128042583



variables

country	year	cases	population
Afghanistan	1990	745	1908071
Afghanistan	2000	2666	2059360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272015272
China	2000	21666	128042583



observations

country	year	cases	population
Afghanistan	1990	745	1908071
Afghanistan	2000	2666	2059360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272015272
China	2000	21666	128042583

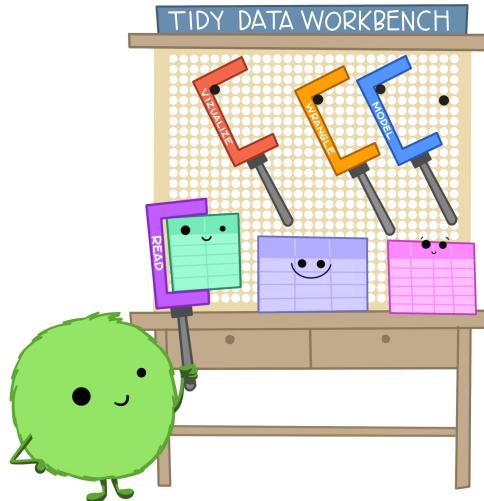


values

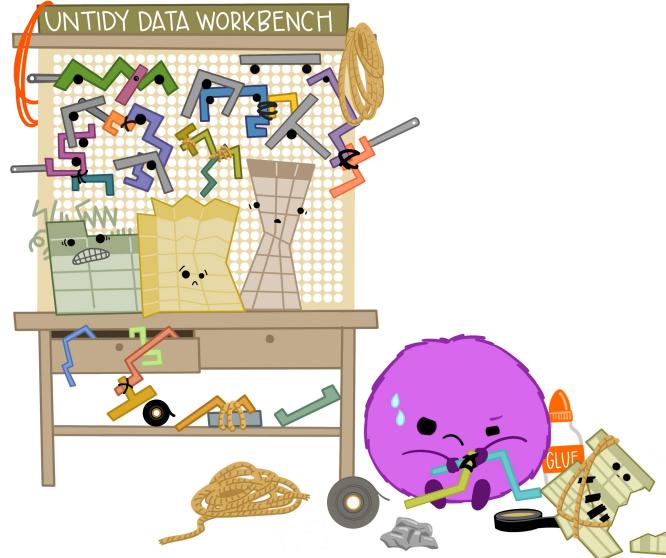
Source: *Chapter 12.2 of R for Data Science*

Hmmm, but why is this important?

When working with tidy data,
we can use the **same tools** in
similar ways for different datasets...



...but working with untidy data often means
reinventing the wheel with **one-time**
approaches that are **hard to iterate or reuse**.



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:

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

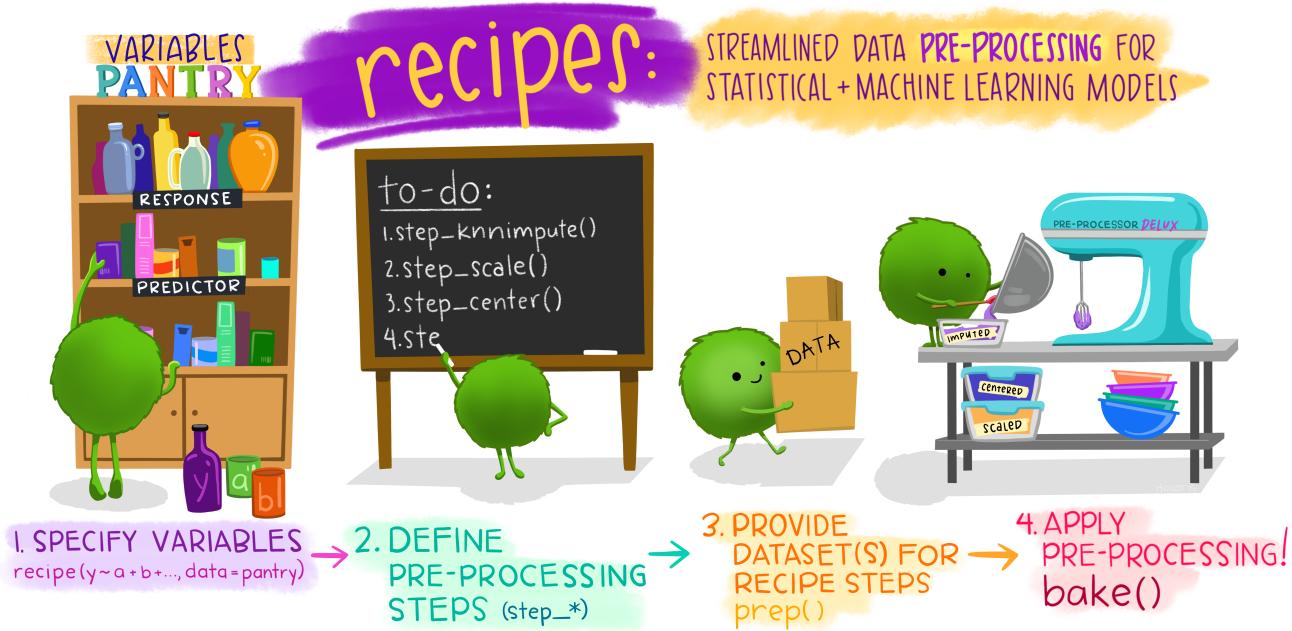
Next, use **initial_split** function to create a binary split of the data into training and testing set.

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

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

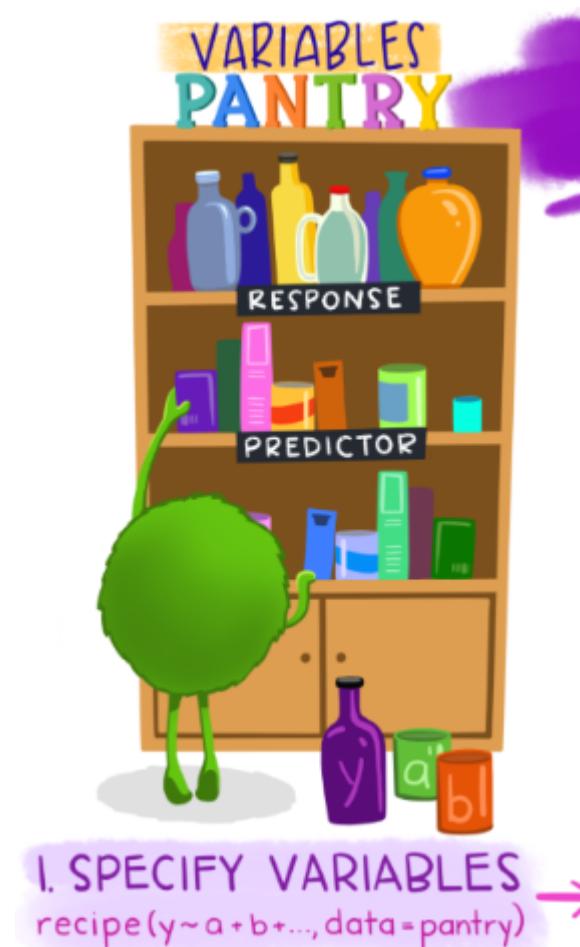
```
df_train <- training(df_split)  
df_test <- testing(df_split)
```

Data Pre-processing



Source: *stats-illustrations* by Allison Horst

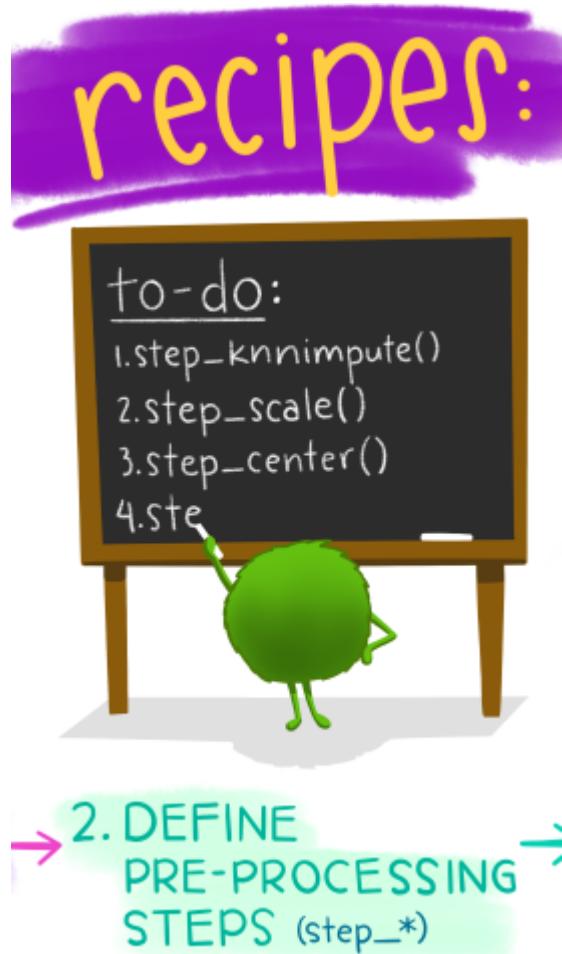
Following is the example of how the **recipe** looks like in **tidymodels**:



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**:



- **3. PROVIDE DATASET(S) FOR RECIPE STEPS**
`prep()`

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.

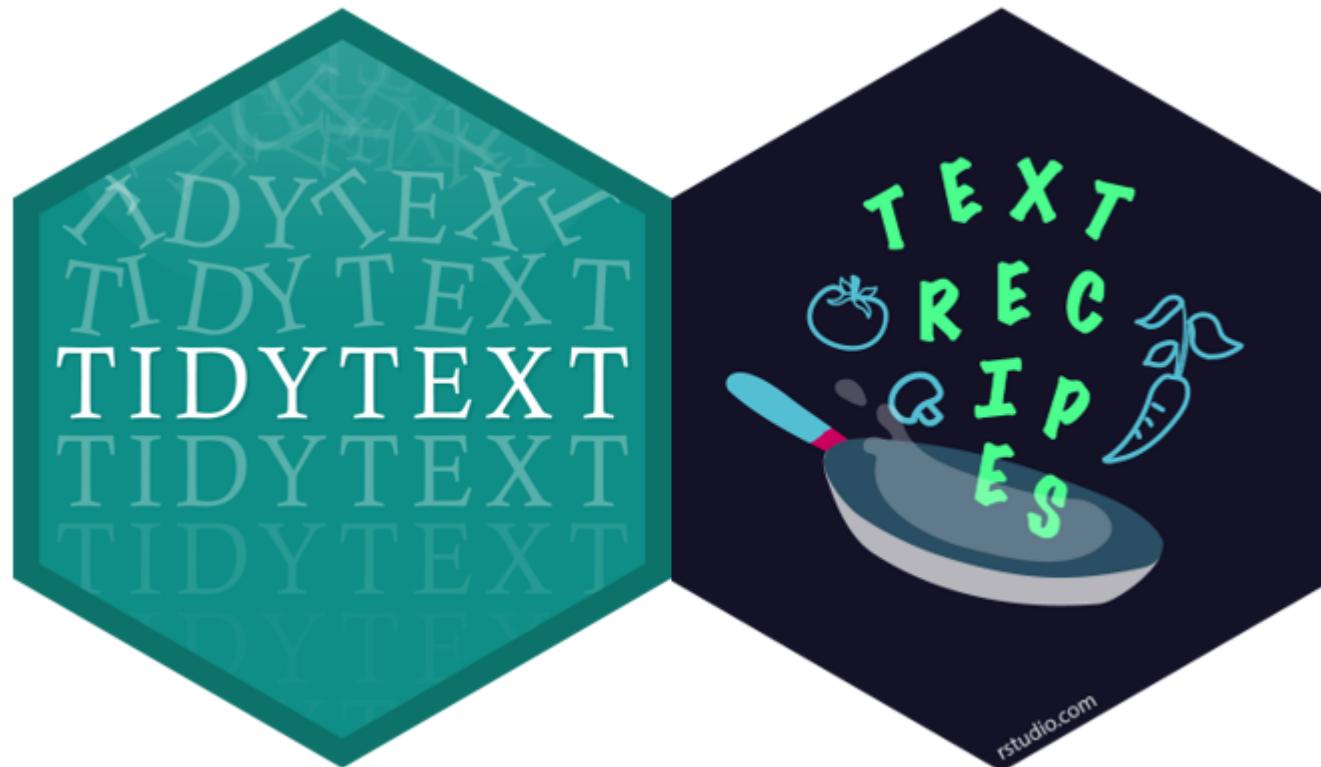
```
glmnet_recipe <- gen_recipe %>%  
  step_dummy(all_nominal())
```

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 [website](#).

Model Selection - Model X, I choose you!



a standardized interface for fitting models
and making predictions.

Horst 2019

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

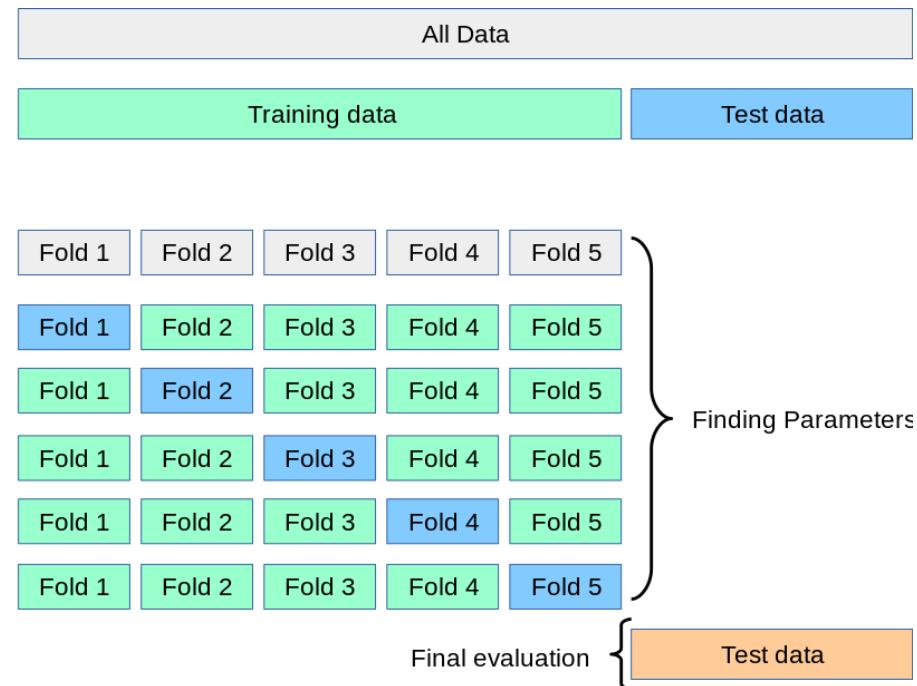
```
kknn_spec <-
  nearest_neighbor(neighbors = tune(),
                  weight_func = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn")
```

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!

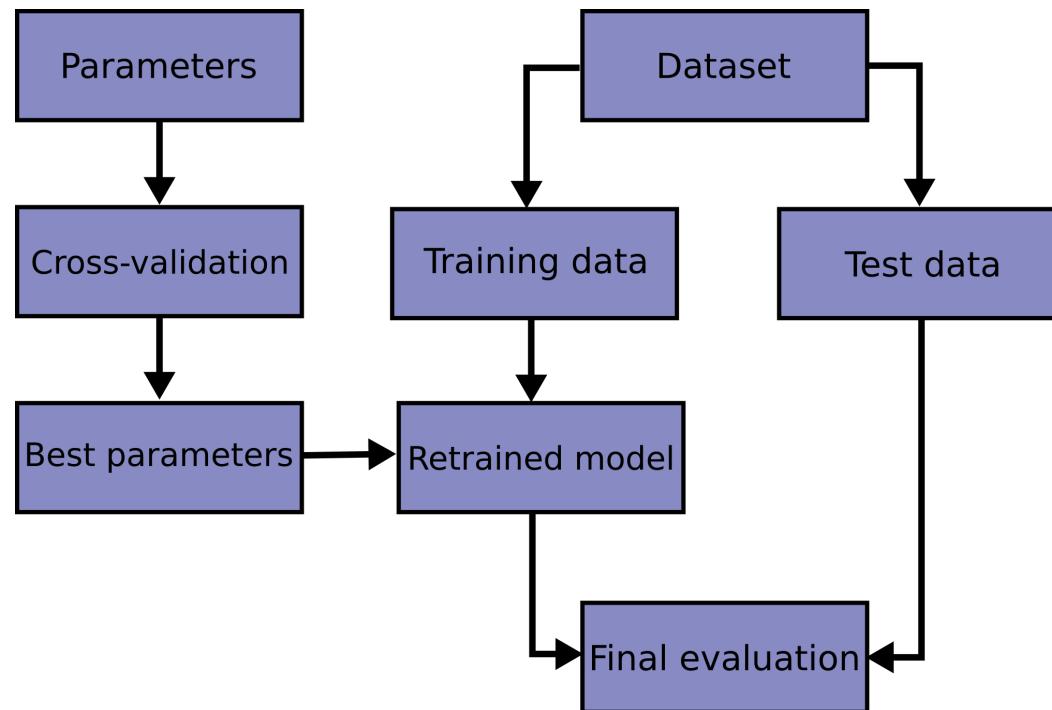
Over here, **grid search** is used to find the best parameters fit a given model.

```
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 different 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.

Is Performing Data Cleaning Waste Time?

First, let's look at what if we build two models, one *without* data cleaning (ie. ranger_org_metric) & the other *with* data cleaning (ie. ranger_ult_metric).

```
## # A tibble: 2 x 5
##   .estimator model      rmse     rsq    mase
##   <chr>       <chr>     <dbl>   <dbl>   <dbl>
## 1 standard    ranger_org 35532. 0.0405  0.680
## 2 standard    ranger_ult  1604. 0.405   0.521
```

Overall, it seems to indicate that the model performance improves after data cleaning is performed

There are unreasonable values & outliers within the dataset. Below is one example of the unreasonable value:

```
-- Variable type: numeric --
# A tibble: 1 x 11
  skim_variable n_missing complete_rate  mean     sd     p0     p25     p50     p75     p100 hist
* <chr>          <int>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
1 HoursworkedPerWeek      0         1  37.7  12.6     0    38     38    40    640 █_____
```

```
## [1] "One week only has max 168 hours."
```

Different Way of Looking at The Problem

In this dataset, we are interested in finding out what is the ultimate claim cost to us.

Another way to look at this problem is how far off is the initial estimate from the actual claim amount.

So, with that, I have built two models over here.

- *ranger_ult* is the model directly predicting on ultimate claim cost
- *ranger* is predicting how far off is the initial estimate from the actual claim amount.

```
## # A tibble: 2 x 5
##   .estimator model      rmse    rsq    mase
##   <chr>     <chr>     <dbl> <dbl> <dbl>
## 1 standard  ranger_ult 1604.  0.405  0.521
## 2 standard  ranger     1537.  0.457  0.490
```

The model performance actually improves when I have modeled on the claim difference.

Important Note: This does not imply that we should always modeled on the claim differences. Just use different methods to predict the claim costs and see which methods give the best results.

Is simpler model better?

```
## # A tibble: 2 x 5
##   .estimator model      rmse    rsq   mase
##   <chr>       <chr>     <dbl> <dbl> <dbl>
## 1 standard    ranger    1537.  0.457  0.490
## 2 standard    ranger_vip 1541.  0.454  0.495
```

The simpler model can perform almost on par as the full model although the simpler model is only using 10 variables (based on variable importance) to fit the model (where the full model is using 19 variables).

In other words, the simpler model requires less inputs to have similar level of model accuracy.

Model Performance under Different Text Mining Approach

```
## # A tibble: 2 x 5
##   .estimator model      rmse    rsq    mase
##   <chr>     <chr>      <dbl> <dbl> <dbl>
## 1 standard  ranger     1537.  0.457  0.490
## 2 standard  ranger_clmdesc 1492.  0.485  0.463
```

The model that use **textrecipe** has a better model performance than the one uses **tidytext**. This is probably due to the penalty function that I used (ie. tf-idf) under **textrecipe**.

```
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)
```

Model Explainability - The Journey Uncover the Black Box

There is an increasing recognition on model explainability.

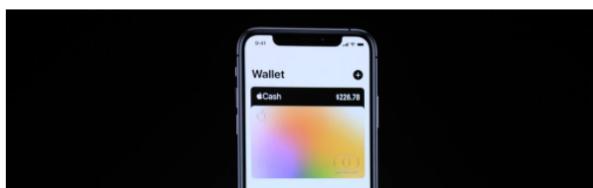
Without understanding why the model predicts the way it did, it can be dangerous.

Below is one of the famous case that the machine learning "went wrong":

The New York Times

Apple Card Investigated After Gender Discrimination Complaints

A prominent software developer said on Twitter that the credit card was “sexist” against women applying for credit.



Source: *The New York Times*

Variable Importance

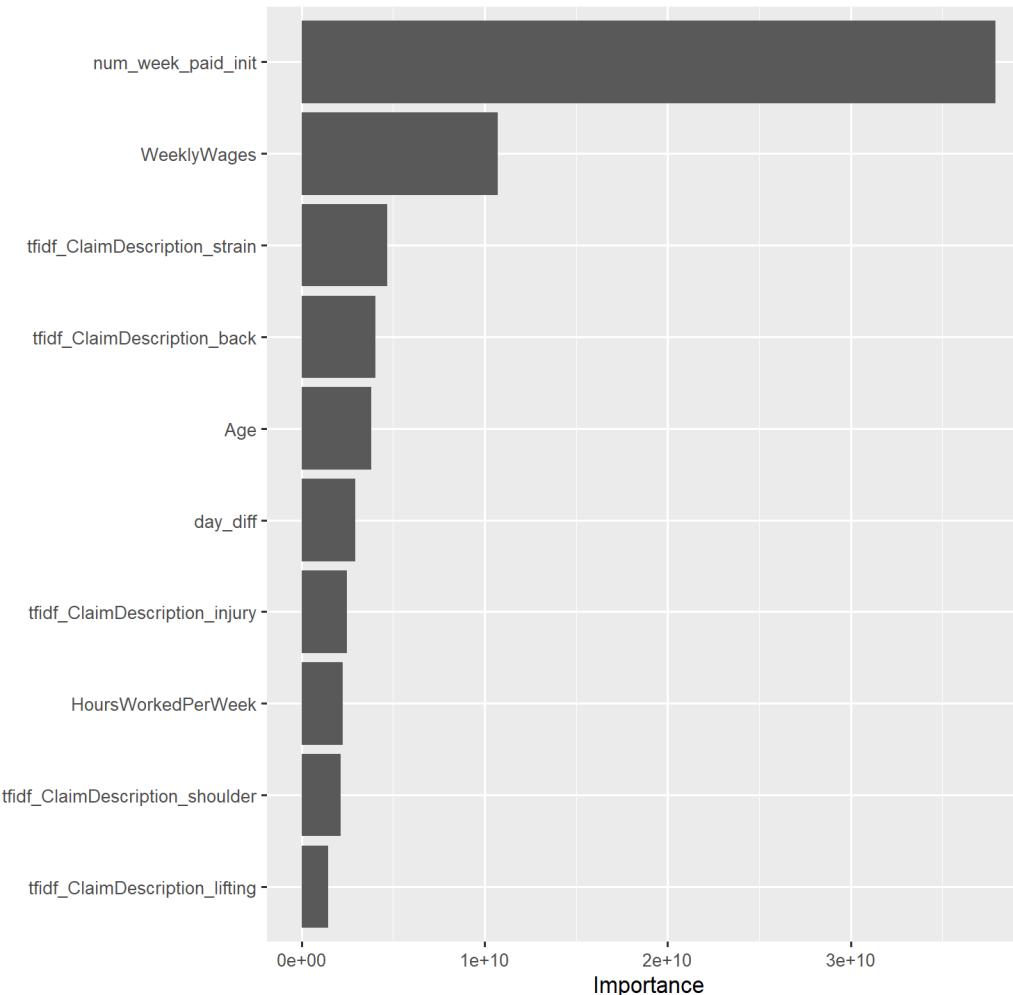
```
ranger_vip_clmdesc <- pull_workflow_fit(ranger_fit_clmdesc$.workflow[[1]]) %>%  
  vi()  
  
ranger_vip_graph_clmdesc <- ranger_vip_clmdesc %>%  
  slice_max(abs(Importance), n = 10) %>%  
  ungroup() %>%  
  mutate(  
    Importance = abs(Importance),  
    Variable = fct_reorder(Variable, Importance),  
  ) %>%  
  ggplot(aes(Importance, Variable)) +  
  geom_col(show.legend = FALSE) +  
  labs(y = NULL, title = "Random Forest Model with TidyText")
```

First, pull the info and pass to variable importance function

Next, plot out the variable importance by using **ggplot** function.

The benefit of using such approach is it still provides the users some flexibility on the visualization of the results.

ranger - Model directly on claim descriptions



It is easier to illustrate in graph format so that its clear on how the "importance" of each variable differs by one another.

Partial Regression Plot

```
ranger_fit_pdp <- fit(ranger_final_wf, df_train)

ranger_pdp <-
  recipe(init_ult_diff ~ ., data = df_train) %>%
  step_profile(all_predictors(), -num_week_paid_init, profile = vars(num_week_paid_init)) %>%
  prep() %>%
  juice()
```

- Use **step_profile** function from **recipe** package to create the dataset with permutation over number of weeks paid initially estimated (ie. num_week_paid_init)
- **juice** function allows us to extract the dataset from the recipe

Dataset we have created for the partial regression plot:

Dataset we have created for the partial regression plot:													
ClaimNumber		DateTimeOfAccident		DateReported		Age		Gender		MaritalStatus	DependentChildren	DependentsOther	W
1	WC1593727	1996-06-18T10:30:00Z		1996-07-23T00:00:00Z		31	F		M		0	0	
2	WC1593727	1996-06-18T10:30:00Z		1996-07-23T00:00:00Z		31	F		M		0	0	
3	WC1593727	1996-06-18T10:30:00Z		1996-07-23T00:00:00Z		31	F		M		0	0	

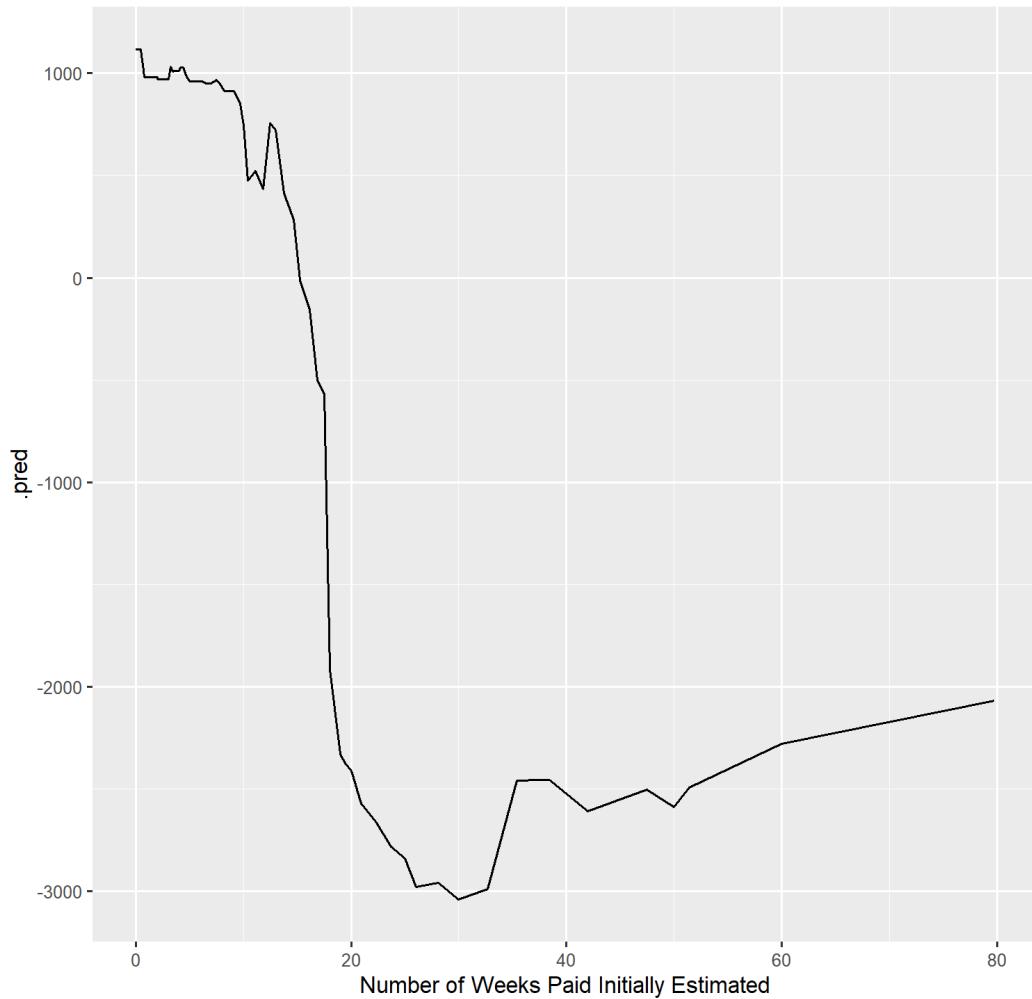
Showing 1 to 3 of 10 entries

Previous 1 2 3 4 Next

Once the dataset is created, we will pass the information into the **predict** function to perform the prediction and visualize the partial regression plot.

```
predict(ranger_fit_pdp, ranger_pdp) %>%
  bind_cols(ranger_pdp) %>%
  ggplot(aes(x = num_week_paid_init, y = .pred)) +
  xlab(sym(i)) +
  labs(title = "Partial Regression Plot on Number of Weeks Paid Initially Estimated") +
  geom_path()
```

Partial Regression Plot on Number of Weeks Paid Initially Estimated



Overall, the initial estimated claim seems to be overstated if the number of weeks paid initially estimated is longer than 15 weeks

Partial Dependent Plot

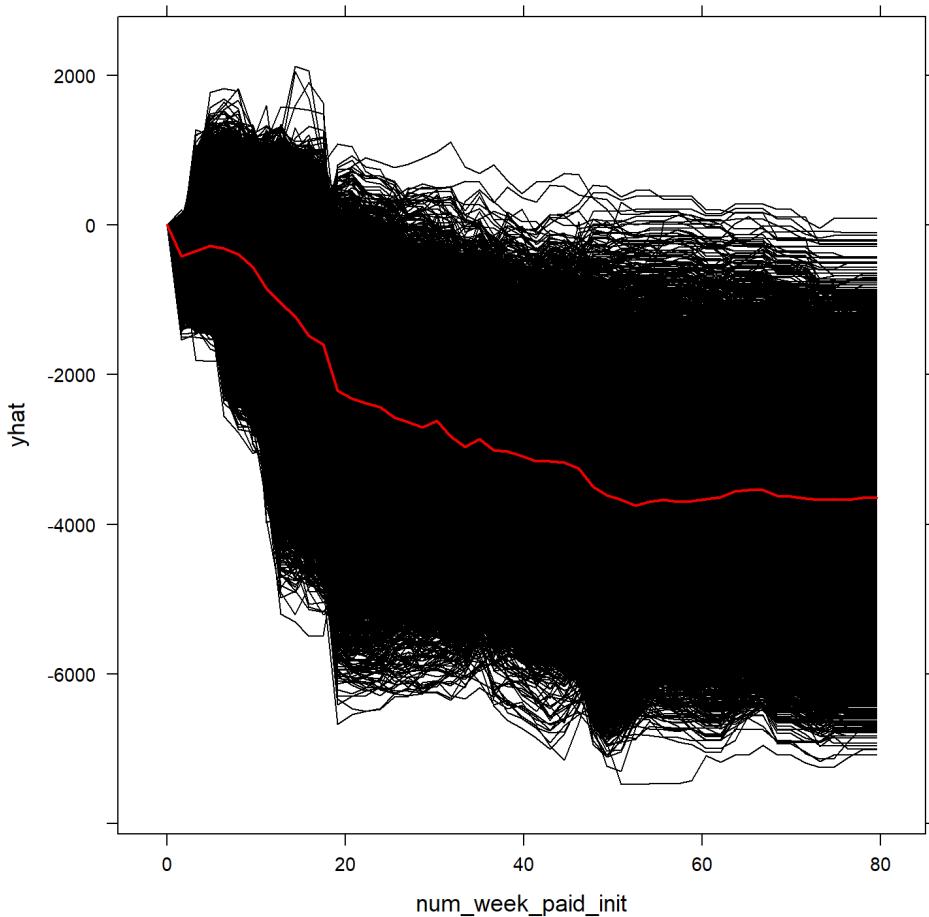
```
pdp_pred_fun <- function(object, newdata) {  
  predict(object, newdata, type = "numeric")$.pred  
}  
  
workflow_partial_num_week_paid_init <-  
  pdp::partial(ranger_fit_pdp,  
    pred.var = "num_week_paid_init",  
    ice = TRUE,  
    center = TRUE,  
    plot.engine = "ggplot2",  
    pred.fun = pdp_pred_fun,  
    train = df_train %>% dplyr::select(-init_ult_diff))  
  
plotPartial(workflow_partial_num_week_paid_init)
```

The key idea of partial dependent plot is similar to partial regression plot.

The main difference is the partial dependent plot permutes over many combinations, the result from this function is less bias.

The red line represents the average effect of the selected variables.

On average, we would expect the total claim amount to be overestimated when the number of weeks paid initially estimated increases.



This is just a very small subset of the entire research



The full research paper and R code will be posted on SAS website <https://www.actuaries.org.sg/>.

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

A screenshot of a blog homepage titled "When Actuarial Science Collides with Data Science". The header includes a navigation bar with links for Home, About, and Past Project. Below the header, there are two main posts: "Sharing at Singapore Actuarial Society" (published April 2, 2021) and "Data Wrangling - The Quest to Tame The 'Beast'" (published Feb. 15, 2021). Each post has a thumbnail image, a brief description, and a "Read more" link. To the right of the posts, there is a sidebar with a "SUBSCRIBE" section for email notifications and a "CATEGORIES" section listing Articles (3), dataWrangling (1), and talks (1). A cartoon illustration of a green character interacting with a machine is positioned at the bottom right.

FAQ

Thank you!



Speaker details:

Jasper LOK

Email: junhaur.lok.2019@mitb.smu.edu.sg

Profile: linkedin.com/in/jasper-l-13426232/

Blog: <https://jasperlok.netlify.app/>

Professor KAM Tin Seong

Email: tskam@smu.edu.sg

Profile: www.smu.edu.sg/faculty/profile/9618/KAM-Tin-Seong

Appendix

Link to the relevant R packages

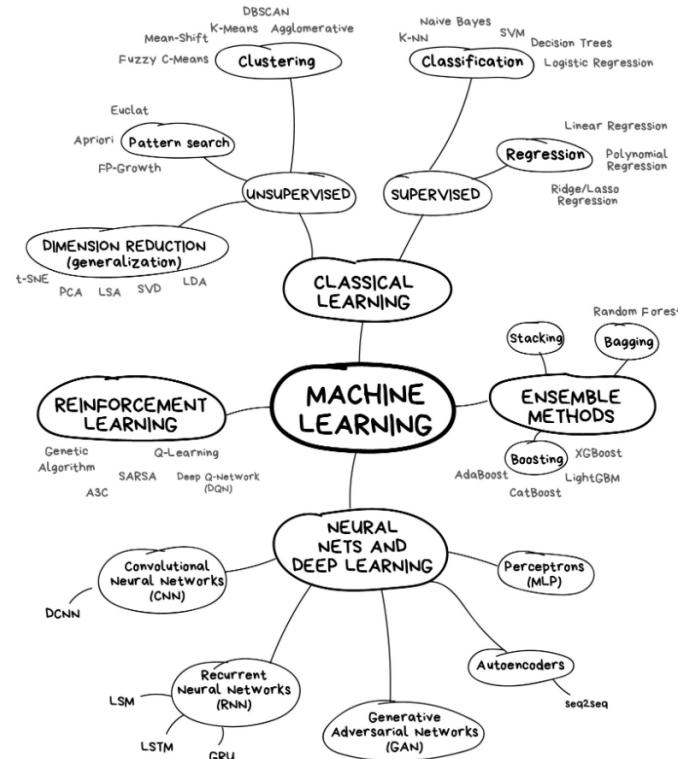
Tidyverse <https://www.tidyverse.org/>

Tidymodels <https://www.tidymodels.org/>

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

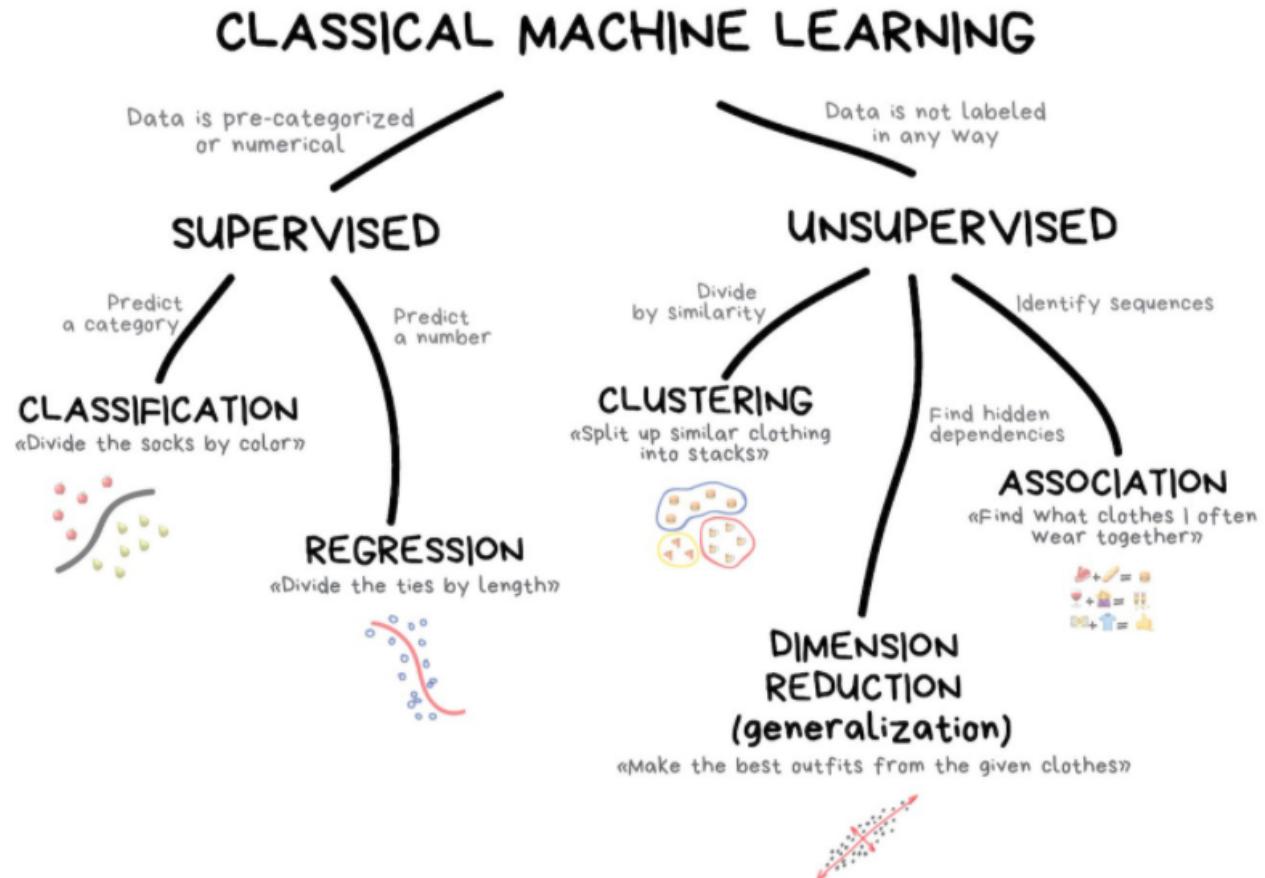
Partial Depedence Plot <https://bgreenwell.github.io/pdp/index.html>

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/>

Text Mining

<https://www.tidytextmining.com/>

Interpretable Machine Learning

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