When 1 + 1 > 2
How Modern Data Science Could Complement Actuarial Science in Claim Cost
Estimation

Final Report

by

Jasper LOK Jun Haur

In Partial Fulfillment

of the Requirements for the Capstone Project Module in the

Master in IT in Business (MITB) Programme at the

Singapore Management University

Supervisor: Professor KAM Tin Seong

Apr 2021

Abstract

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. Often, the ability to optimize the capital would lower the cost of capital for companies. This could translate into an increase in profit from the lower cost incurred or an increase in competitiveness through lowering the premiums companies charge for their insurance plans.

In this analysis, **tidyverse** and **tidymodels** packages are used to demonstrate how the modern data science R packages could assist the actuaries in predicting the ultimate claim cost once the claims are reported. The conformity with tidy data concepts by these R packages has flattened the learning curve to use different machine learning techniques to complement the conventional actuarial analysis. This has effectively allowed actuaries in building various machine learning models in a more tidy and efficient manner. The packages also enable users to harass on the power of data science to mine the "gold" in unstructured data, such as claim descriptions, item descriptions, and so on. Nevertheless, these would enable the companies to hold less reserve through a more accurate claim estimation while not compromising the solvency of the companies, allowing the capital to be re-deployed for other purposes.

Table of Contents

1.0   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. Often, the ability to optimize the capital would lower the cost of capital for companies. This could translate into an increase in profit from the lower cost incurred or an increase in competitiveness through lowering the premiums companies charge for their insurance plans. In the China Insurtech industry report by Oliver Wyman, the authors discussed how data science contributes to the different key success factors for the insurtech in China (Sheng et al. 2016).

As data science can be applied to many areas of actuarial analysis, claim cost (also known as burning cost in actuarial context) estimation is selected to be the use case for this capstone project. Generalized linear model (GLM) has been widely adopted by the industry to determine the premium (also known as 'ratemaking' in the insurance context) due to benefits this model offers, such as model explainability and so on. However, GLM suffers a few drawbacks and these drawbacks could potentially result in a loss to insurers due to the insufficient premium. Fortunately, these drawbacks can be addressed by using some of the modern data science techniques, potentially allowing for a more accurate actuarial estimation.

By combining the domain knowledge, programming skills, and advancement in technology, these would allow the insurers to harness the power of data science such as exploring unstructured data with modern data science techniques, venturing into new business models by leveraging on big data, different machine learning algorithm might better in reflecting the policy price and so on. Undoubtedly, this would help companies in better running, growing, and transforming the business, gaining a competitive edge in the financial markets.

R is chosen to be the programming language to demonstrate how modern data science techniques can be applied in actuarial science context. This is because R has a wide range of packages that enable the users in performing machine learning tasks. **caret** package is one of the widely used R package to perform machine learning tasks. This package functions as a wrapper to provide the users a more unified interface to perform machine learning tasks. However, the author recognized the limitation on how **caret** package was designed, making it hard to extend to other users to tap into this framework (RStudio 2018).

Another popular machine learning R package is **mlr3**. Same as **caret** package, this package also functions as a wrapper. It has gained popularity due to the unified interface and wide range of models supported under the package. However, one is unable to pass the output from **mlr3** package directly to the packages under **tidyverse** as the output data does not follow tidy data concepts. The authors have get around on this issue by writing the wrapper function around some of the more popular **tidyverse** functions. For instance, to visualize the results, the **autoplot** function from **mlr3viz** package passes the necessary inputs into the **ggplot2** package to plot the necessary graphs. Meanwhile, based on the documentation (Lang et al. 2021), the function from the **mlr3viz** package does have some limitations on the customization one could have on the graph due to the design of the function and package.

Therefore, **tidyverse** and **tidymodels** packages are used to demonstrate how the modern data science R packages could assist the actuaries in predicting the ultimate claim cost once the claims are reported. The conformity with tidy data concepts by these R packages has flattened the learning curve to use different machine learning techniques to complement the conventional actuarial analysis. This has effectively allowed actuaries in building various machine learning models in a more tidy and efficient manner. Besides, the functions under these packages allow the users to pass the outputs to another function within the same ecosystem without requiring much data transformation. This has effectively shortened the preparation and modeling time, allowing actuaries to have more time in explaining or further analyzing the results. These could enable the companies to sharpen their claim cost estimation, resulting in less required reserve while not compromising the solvency of the companies. This would allow the capital to be re-deployed for other purposes.

Section 1 provides an overview of this report. Section 2 will cover the motivation behind this research. Literature review can be found under Section 3. Section 4 summarizes the selection criteria of the dataset and descriptions of the selected dataset. Next, an overview of the data science project is described under Section 5. This will be followed by the explanatory data analysis (EDA) in Section 6. Section 7 first provides an overview of machine learning before diving into the machine learning analysis. Demonstrations on the model building, including data splitting, data pre-processing, model fitting, model evaluation, and so on, will be covered under Section 8. Communication (ie. the last step of a typical data science project) will be covered under Section 9. The report will end with a conclusion from this research project.

2.0   Motivation

This capstone project was motivated by how the modern data science approach could complement conventional actuarial analysis, where claim cost estimation will be used as a demonstration. This report also aims to demonstrate how one could build a more seamless process in data science analysis workflow.

As such, this capstone project attempts to cover the following requirements to address the issues mentioned in Section 1:

- To demonstrate some of the best practices when performing data science analysis and the benefits of using the modern data science packages to perform machine learning analysis

- To illustrate how actuaries could perform feature engineering on unstructured data to draw additional insights from the dataset

- To demonstrate how R Markdown could allow users to create a reproducible analysis

Also, this project also aims to exhibit how **tidyverse** package and **tidymodels** package can work seamlessly together to assist one in their data science analysis.

3.0 Literature Review

3.1 Actuarial Science vs Data Science

Some existing articles and posts discussed the differences between actuarial science and data science and how data science might affect the future of actuaries. Some of these articles even suggested that the demand for actuaries might decrease in the future due to the increasing adoption of data science by companies. By leveraging on the skillsets of data scientists, companies could achieve the following, resulting in lower demand for actuaries (Franklin 2016):

- Automating claim approval by leveraging on various machine learning models, resulted in expense saving

- Automating existing processes, resulting in a shorter turnaround time required

- Exploring different machine learning algorithms to perform the conventional actuarial analysis (eg. claim cost estimations), providing a potential more accurate result

These benefits could well translate into comparative advantages for companies in this increasingly competitive market.

Interestingly enough, the Institute and Faculty of Actuaries suggested in one of their webpage that actuarial science and data science are rather much in common (Institute and Faculty of Actuaries, n.d.). The primary commonality between data science and actuarial science is their respective appetites for data. The more data they have, the better their analyses turn out. Despite the similarity, most articles agree that it is unlikely data scientists could completely take over actuaries due to the unique skillsets, apprenticeship, and understanding of regulations actuaries have built up over the years (McKinsey Company 2020), (Smith Hanley Associate LLC 2018).

However, these articles also agree that there are new skill sets actuaries should consider acquiring to venture into this new area as shown under Figure 1 (Quantee, n.d.). Note that the bubble sizes in the graph have been modified to empathize the equal importance of each component.
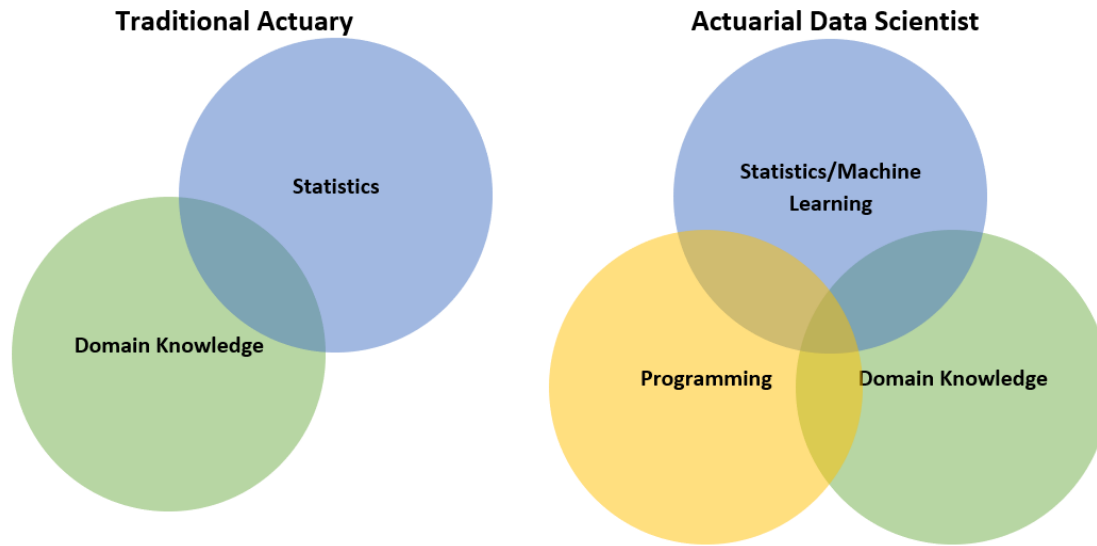
Figure 1: Comparison of the traditional actuary and actuarial data scientist

Acquiring the relevant data science skill sets could add the following values to actuarial tasks (Perkins and Preez 2018):

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	Utilizing alternative modeling approaches allows different perspectives to be gained on data
New Approaches to Problems	Produce a wider variety of models quickly, allowing the insurers to have better ability to select the appropriate modeling approach for a given problem
Improved Data Visualisations	Increasing power to produce stunning visualizations of data which can itself provide new perspectives on a task

Figure 2: List of benefits data science could bring to the actuarial task

As for this capstone, the focus is to demonstrate how one could use various modern data science packages to perform the necessary analysis once the dataset has been identified and extracted. Hence, data capturing and storing are out of scope for this capstone project.

Apart from that, various actuarial professional bodies (eg. Society of Actuaries and Actuaries Institute) have incorporated or will be incorporating data science to be part of the education over the years, showing the actuarial community has recognized the importance of learning data science to stay relevant in the financial industry.

Figure 3: Machine Learning

However, there is a lack of discussions and demonstrations on how actuaries could embark on this data science journey. More importantly, there is also a lack of discussions on how one could use the different packages effectively and efficiently to perform necessary analysis. In other words, knowing programming language or using off-the-shelf software does not guarantee the users the ability to perform a good data science analysis (Figure 3). It is valuable when one could combine data science skills with domain expertise, programming, software tools, and knowledge of mathematics and statistics, extract desired insights from data – insights that can be translated into tangible and quantifiable business value, such as market intelligence, risk assessment, and executive decision-support (Institute and Faculty of Actuaries, n.d.).

Besides, most of the data science discussions focus on how we could use the various machine learning algorithms to improve the accuracy of the models. However, selecting the various machine learning algorithms is a very small subset of the entire modeling steps. Often, the approaches used to perform modeling (eg. do we split the data into small subsets and build different models on each dataset?) are equally important. Also, conventionally different individual machine learning packages are used to perform machine learning analysis. As there is no widely accepted standard on how the packages should be designed and structured, hence users often spend a fair amount of time "joining" the functions together so that they could work together.

Hence this capstone project aims to provide the users with an overview of a modern data science project and the relevant techniques. I will be using different modeling approaches on the same type of machine learning model by leveraging on **tidymodels** to attempt to improve the model accuracy. As the dataset used is open-source worker compensation claim

data, this project aims to benefit the actuarial community by providing the readers with a use case on how modern data science can be implemented in an actuarial context.

3.2 Claim Cost Estimation

Claim cost estimation is one of the important tasks in actuarial science. The ability to estimate claim cost accurately could translate into a lower cost of capital or a lower premium for the customers.

In general, cash flow can be categorized into 4 types depending on the timing of the cash flow and whether the amount of the cash flow is known upfront. They are as shown following:

	Timing of Cashflow	Cashflow Amount
Type I	Deterministic	Deterministic
Type II	Deterministic	Stochastic
Type III	Stochastic	Deterministic
Type IV	Stochastic	Stochastic

Figure 4: Different Types of Liability Cashflow

Often both claim timing and claim payout in general insurance are stochastic. For example, the policyholders may claim any time during their policy coverage and the claim size may depend on other factors such as hospital bill size, cost of repairing the car, and so on. In other words, the claim size depends on what is being covered under the policy. The claim size is usually not known or fixed at the point of the inception of the policy.

Therefore, insurers will typically attempt to estimate the amount of money to set aside to meet the potential claim payout. Incorrect assumptions could hurt the business, even resulting in the insurers being insolvent under the extreme scenario.

Besides, with the advancement of the internet and technology, there are more and more third-party aggregators (eg. CompareFirst, MoneySmart, and so on) that consolidate the premiums charged by the different insurers. This has created more transparency for the customers to compare the premiums insurers are charging for different plans. This has indirectly created a price war among the different insurers, resulting in a shrinking profit. Hence, an inappropriate premium setting might leave the insurers with a pool of "bad" risks.

As the competition in the market becomes increasingly intense, insurers are also looking at different ways to reduce or lower the cost while protecting the profit margin of the business. Hence, sharpening assumptions become increasingly important to ensure the assumptions are not overly prudent. Updating the assumptions to be more aligned with the underlying risks could potentially allow the insurers to release the capital and use it for other purposes (eg. expanding the business into other markets, invest in the infrastructure, and so on).

3.3 Conventional Modeling Approach

3.3.1 Modeling Approach

Conventionally, while performing data science analysis or building machine learning models, various R packages are being used. For example, one might use **lm** package to build a linear regression model and **ranger** package to build a random forest model. However, one of the issues is the heterogeneity of the model interface for the various machine learning models. This could become a stumbling block for one to start building machine learning models.

Apart from that, most of the output from the various machine learning models also comes in different forms or data types. Besides, (Cleveland R User Group 2021) also demonstrated in one of his recent data science sharing that even with the same package, the output from the model function could be different depending on the arguments specified in the model function. Often, it becomes an issue when we try to use different packages in various parts of data analysis. Data transformation is often required to transform the output data type into the required format by the next function.

Besides, the heterogeneity of the model interface and output also resulted in less reusability of the various model components. Much of the time might be spent in transforming the data into the required format, instead of uncovering the insights from the dataset.
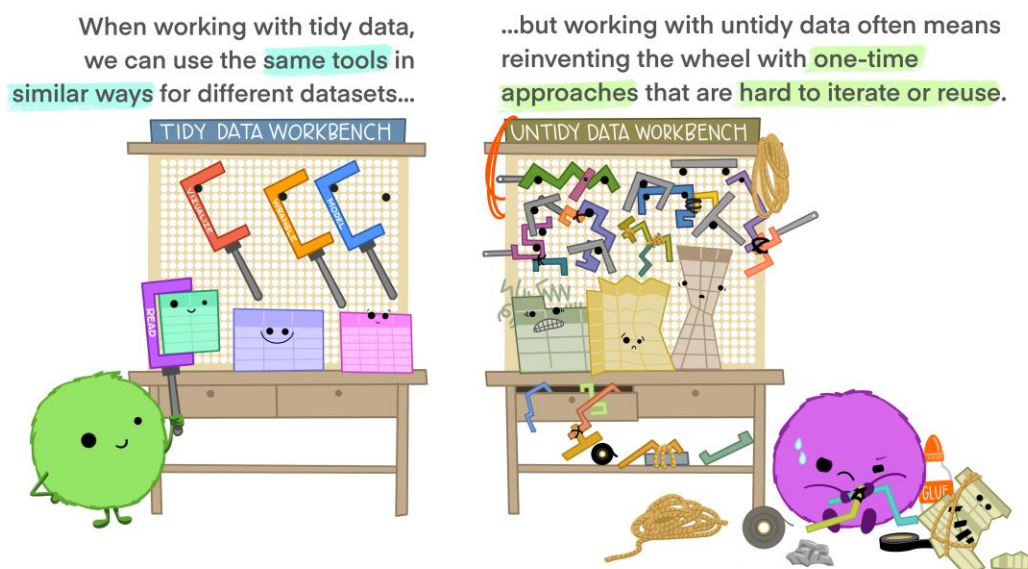

Figure 5: Tidy Data Workbench(Horst, n.d.)

As such, the author of **caret** package worked with several people in RStudio to create **tidymodels**, where it consists of a series of modeling packages that follows principles under tidy API.

Below are the principles of tidy API (Wickham 2020):

- Reuse existing data structures

- Compose simple functions with the pipe

- Embrace functional programming

- Design for humans

While knowing how to run the different packages is important, most importantly knowing the best practice in using the various packages to work together in performing machine learning models is also crucial. This would allow us in achieving efficiency and minimize the potential errors in the analysis.

Therefore, one of the key aims of this research report is to demonstrate how **tidyverse** package and **tidymodels** package can work together to provide the users a more seamless experience in their data analytics tasks. **tidyverse** is a collection of R packages designed for data science. These packages share an underlying design philosophy, grammar, and data structures (RStudio, n.d.). This would provide users a more seamless experience while performing the data science task. This also allows the learning curves for users with little or no programming experience to perform data science not as steep as many other languages. **tidymodels** is collections of R packages that one might need for machine learning purposes. As **tidymodels** also conform to tidy data concepts, it would enable the users to spend less time on the data transformation, but spend more time in performing the analysis itself to uncover the hidden insights in the dataset.

3.3.2 Conventional Machine Learning Model

GLM is traditionally used to estimate claim cost due to the benefits it brings. However, GLM might not be appropriate in some of the circumstances. In the SOA report, the authors have listed some of the drawbacks the GLM model suffers (Diana et al. 2019). One of the key assumptions made by using the GLM model is the relationship between features and the response is linear after the transformation of the effect through a link function. This poses a model risk as the estimation will be inaccurate if the inappropriate link function is used in GLM.

Several articles also pointed out that the conventional way to approach GLM does not provide a systematic way to find the interaction terms (Zhou and Deng 2019). Users would need to trial and error to find the significant interaction terms to be included in the claim cost prediction. With the increasing number of parameters, this approach is time-consuming and impractical. Hence, I will be using various machine learning models to attempt to account for the interaction terms effects.

4.0   Data Source

To demonstrate how different modern data science R packages can help in the convention actuarial analysis, below are criteria set in selecting the dataset:

- Does the dataset contain necessary policyholders' information (eg. age, gender) for us to perform the necessary analysis?

- Does the dataset contain unstructured data fields so that we could demonstrate how various R packages can be used in building the machine learning models?

After comparing various datasets, the worker compensation insurance claim dataset from the Kaggle website is being chosen for this capstone project. Worker compensation insurance is a form of insurance that provides wage replacement and medical benefits for the employees if they are injured.

In this analysis, I will be predicting the estimated ultimate claim cost after the claims have been reported based on the information provided in the dataset.

Variable Name	Descriptions
ClaimNumber	Unique policy identifier
DateTimeOfAccident	Date and time of the accident
DateReported	Date that accident was reported
Age	Age of worker
Gender	Gender of worker
MaritalStatus	Martial status of the worker. (M)arried, (S)ingle, (U)nknown
DependentChildren	The number of dependent children
DependentsOther	The number of dependants excluding children
WeeklyWages	Total weekly wage
PartTimeFullTime	Binary (P) or (F)
HoursWorkedPerWeek	Total hours worked per week
DaysWorkedPerWeek	Number of days worked per week
ClaimDescription	Free text description of the claim
InitialIncurredClaimCost	Initial estimate by the insurer of the claim cost
UltimateIncurredClaimCost	Total claims payments by the insurance company

Figure 6: Table of Variables Name

Following are the definition of additional variables I have derived based on the information within the existing dataset:

Derived Variable	Descriptions
init_ult_diff	Difference between initial and ultimate claim cost
num_weeks_paid_init	Number of weeks initially estimated to be paid
acc_hr	The hour that the accident happened
day_diff	Number of days apart between Accident Date and Claim Reported Date

Figure 7: List of Derived Variables

Note that the definition of those time variables I derived from the time date variable later in the section is not included in the table since they are intuitive.

Further data pre-processing on the dataset is performed on the dataset as some of the variables (eg. DateTimeOfAccident) are inappropriate to be used directly to fit models. The details on the data pre-processing will be discussed in the later part of the report.

5.0 Overview of Data Science Project Workflow

A typical data science project workflow would look as following (Grolemund and Wickham 2016):
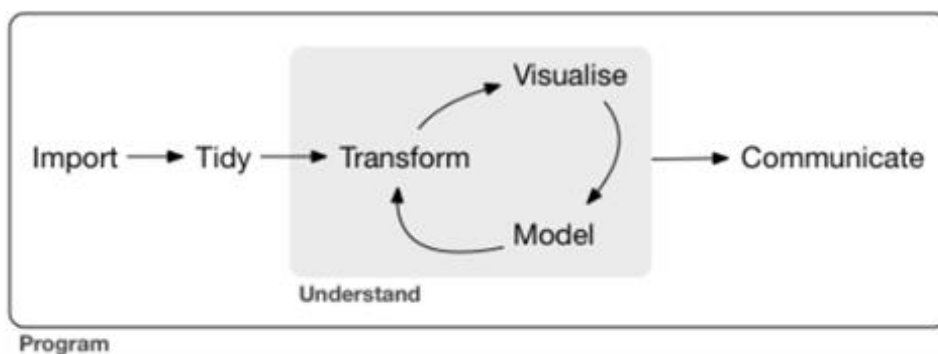
Figure 8: Typical Data Science Process

Following is a simple illustration of how the different R packages cover various parts in data science project workflow:
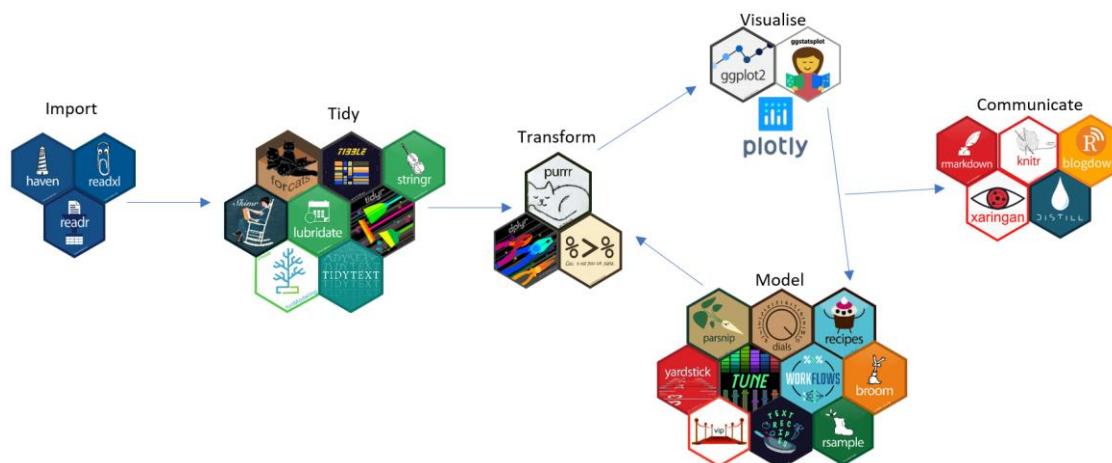

Figure 9: Illustration of the data science project workflow that different R packages cover

The beauty of most of these packages is they are designed to work "tidy data". To be considered as tidy data, the data must fulfill the following three rules (Grolemund and Wickham 2016):

- Each variable must have its column

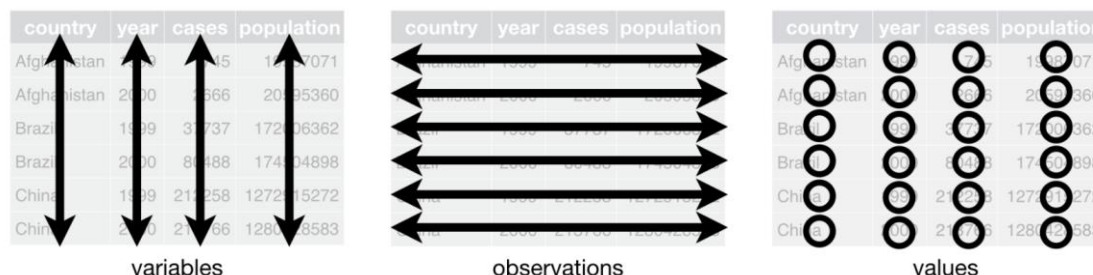- Each observation must have its row

- Each value must have its cell

Figure 10: Definition of tidy data

By following this "tidy data" concept, it would be easier for the users to learn and perform analysis due to the consistency in how data is stored. Also, the packages mentioned under Figure 9 are designed to work together, instead of individual packages designed separately.

Some of the recommended R packages for each step and brief descriptions of the respective packages are included below. Note that the list of packages used in this report is meant to provide users with recommended packages for the various data science tasks. It is not meant to take the recommended packages as the only packages as there are still a lot of developments within the R community. There will likely be even better R packages released in the future to support the users in performing data analysis.

The process will start with importing data into the necessary tools before one could perform any analysis. Different packages such as **readr**, **readxl**, **haven**, and so on, allow the users to import different types of data into the environment.

Package	Descriptions
readr	Read rectangular data (eg. csv, tsv, and fwf)
readxl	Read Excel file (eg. xls, xlsx)
haven	Read SAS, SPSS, Stata data file

Figure 11: List of Importing Data Packages

Once the data is imported, it is a good practice to check the quality of the data.

Package	Descriptions
funModeling	Tools to perform EDA, data preparation, and model performance
skimr	Provide summary statistics in a very neat format

Figure 12: List of Checking Data Quality Packages

Data wrangling will be then performed, which involves tidying and transforming the data.

Package	Descriptions
tidyr	Responsible for creating tidy data
stringr	Tools to transform string
lubridate	Transform date and time-related fields
tibble	Create/transform the data into tibble format
forcats	Provides tools to transform factors (eg. rearrange the factor levels)

Figure 13: List of Tidying Data Packages

Package	Descriptions
dplyr	Provide a selection of tools for data manipulation
purrr	Enhance R functional programming
magrittr	Allow users to use the "piping" method to join the different codes

Figure 14: List of Data Transformation Packages

Different packages focus on the data transformation for different types of data as shown in both tables above. This step would help the users in understanding the underlying data and most importantly whether the dataset could help in addressing the data science problem.

Feature selection is one of the important steps before building the various machine learning models. This is because more complicated models do not guarantee better model performance. R has various packages that could help the users in understanding which variables are important in explaining the target variable.

Package	Descriptions
infer	Allow the users to conduct statistical inference and visualize the results
corrplot	Visualize correlation in graph format
funModeling	Various functions to illustrate the relationship in graphic format
ggstatsplot	Visualize statistical tests in graphical format

Figure 15: List of Feature Selection Packages

Visualization is also a crucial part of the process. The aims of visualization are to detect the expected and to discover the unexpected (Kam 2020a). Hence, often a good visualization could show us things that one might not expect or anything one might want to investigate further. Besides, visualization will also convey other useful information, such as which variables will be useful in the machine learning tasks, how one could split the dataset before building machine learning models, and so on.

Package	Descriptions
ggplot	Allow the users to plot different types of charts and graphs
plotly	Allow the users to plot interactive graphs

Figure 16: List of Graphs Plotting Packages

As discussed under the Introduction section, **tidymodels** package is selected to build the various machine learning models. The advantage of using packages under **tidymodels** is these packages also follow the tidy data concepts mentioned in Figure 10. This has allowed the users to use functions from **tidyverse** package as well. Besides, this package allows users to modularize the model components, allowing the created components to be reused throughout the analysis when necessary.

Package	Descriptions
rsample	Data splitting and resampling
parnsip	Unified the model interface for different machine learning models
recipe	Data pre-processing
workflows	Chain the different steps (eg. data pre-processing, modeling, model assessment) in the machine learning process together
tune	Optimize the hyperparameters of the models and data pre-processing steps
yardstick	Measure the performance of the machine learning models
broom	Tidy the information in tibble format
dials	Create and manage the tuning parameters

Figure 17: List of Machine Learning Packages for Tidymodels

Lastly, one would build a machine learning model, attempting to answer the initial data science questions one tries to address by using data. Note that the process between transform, visualize and model is often iterative as often one might go back to some of the previous steps due to various reasons, such as calibrating the model on the fly by putting the visualization and parameters side by side and so on. Communication is the last step in any data science project. Without effective communication, it does not matter how well the earlier data science tasks perform.

Currently, R supports a wide range of R packages that allow the users effectively to communicate the results via various formats while enabling the results are reproducible:

Package	Descriptions
rmarkdown	Allow the users to convert markdown documents into different formats, such as HTML file, word documents, PDF, and so on
knitr	General-purpose literate programming engine
xaringan	Render the Rmarkdown document into presentation slides
blogdown	Allow users to build a blog by rendering on Rmarkdown document
distill	Alternative R package allows the users to build a blog

Figure 18: List of Packages for Communications

RMarkdown is chosen to demonstrate how they could use it to communicate the results while maintaining reproducibility. Research is considered to be reproducible when the exact results can be reproduced if given access to the original data, software, or code. Reproducible

research is sometimes known as reproducibility, reproducible statistical analysis, reproducible data analysis, reproducible reporting, and literate programming (Bock, n.d.). Typically, the results, including tables, charts, graphs, values, and so on would need to be reproducible in the analysis.

However, according to a survey conducted by Nature, more than 70% of the researchers are unable to reproduce the results of other scientists and more than half have failed to reproduce their experiments (Baker 2016). This is concerning as actuarial works are strictly regulated and heavily audited. Without reproducibility, this may make it harder to verify the analysis or even replicate the results in the future. Apart from that, reproducibility allows one to extend the analysis to other areas, allowing for greater efficiency and providing an advantage for the insurers.
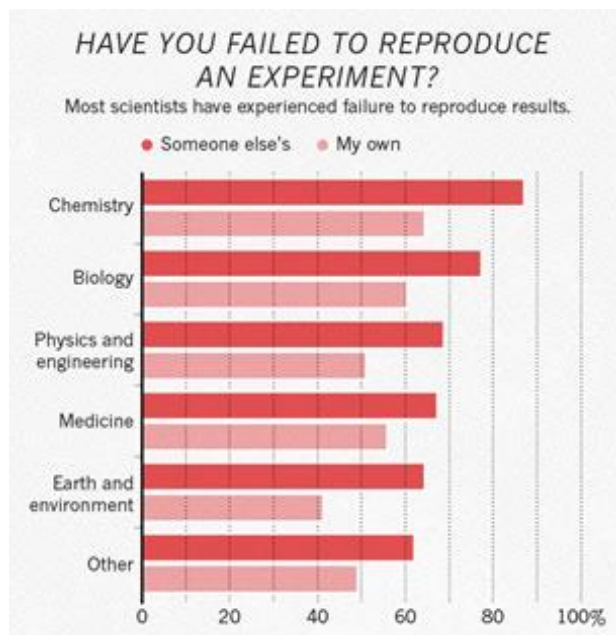

Figure 19: Reproducibility survey conducted by Nature

In the Rmarkdown for Scientists (Tierney 2020), the author illustrated with an example on RMarkdown enables literate programming, which resolved the reproducibility problem. The conventional way of producing the report for analysis is to run the results and copy the necessary into Excel or Word. However, the analysis and report are separated, exposing the users to the risk that the results in the report are not consistent with the analysis since this method relies on human copy and paste. Besides, this makes it harder for the users to understand the analysis without having the report. Also, this process is not exactly sharable as the audiences will not know how exactly the analysis was done. Moreover, this process is more prone to human error. One might miss out on updating the results in the report after running the analysis.

Besides, **Rmarkdown** can be rendered into different formats such as HTML, word documents, presentation slides, and so on, allowing the users to communicate the results without needing to cut and copy the results into other documents. This has eliminated the need to perform reconciliation between different documents. This method also enables

literate programming, making it easier for the users to understand how the analysis is performed. Nevertheless, almost 90% of the entire project (including presentation slides and research report) are written in **RMarkdown**.

Often these data science tasks are performed with programs. The analytics program can be broadly split into two main groups: open source (eg. R, Python, etc) and off-the-shelf software (eg. SAS, Tableau, Alteryx). Although off-the-shelf software allows the users to embark on the data science journey without needing to learn to code, a license is often required to use off-the-shelf software and the cost of acquiring the license can be a hurdle. The off the shelf software are usually standardized and may not fit into some of the contexts. For example, one may want some results that are currently not included in the software output. Often it costs a lot of money to customize the software to handle such scenarios, where open-source software does provide the users the flexibility to customize the code and interface based on the business needs.

Also, often how successful the data science project also depends on understanding on data science project workflow and the considerations under each step such as the suitability of the data, how well the data is being prepared, and so on. In other words, having this off-the-shelf software does not guarantee the success of the data science project.

5.1   Other Considerations in Data Science Workflow

Loop Function to Attach Packages

The common approach to call the library into the environment is directly using **library** function in R (Figure 20). However, this poses an issue if the package is not installed in the machine. The environment will show an error that the relevant package is not installed.

```r
library(tidyverse)
library(tidymodels)
```

Figure 20: Conventional method to call libraries

The better approach is to write a loop function to check whether the availability of the packages in the machine (Kam 2020b). If the packages are not installed for the machine, the function will download the relevant packages from R CRAN. Once the packages are installed or already installed, the function will call the relevant packages. This would ensure that even if the codes are passed to a beginner or anyone with no R coding experience, they would be able to still run the codes. This approach also allows for neater coding and prone to fewer errors when making necessary changes to the coding.

```r
packages <- c('tidyverse', 'funModeling', 'tidymodels', 'doParallel', 'lubridate', 'usemodels', 'ranger', 'vip', 'pdp', 'skimr', 'plotly', 'ggpubr', 'stacks', 'ggExtra', 'textrecipes', 'xrf', 'tictoc')

for (p in packages){
  if(!require (p, character.only = T)){
    install.packages(p)
  }
```

```
    library(p, character.only = T)
}
```

Figure 21: Recommended approach to call libraries

Data Science Project Folder Structure

The conventional approach to read the data file is to specify the full location of where the data is located as shown below.

```{r}
add_info <- read_csv("C:/Users/Jasper Lok/Documents/1_MITB/Term
99_Capstone/4_Sandbox/R_workspace/data/Additional_Info.csv")
```

Figure 22: Conventional method to store data

However, this poses an issue when this project is shared with others. The link on the data location would need to be changed. An easier approach is to have a data folder within the R project folder (Figure 23). This method would not require the users to specify the location of the data. Hence, when there is a need to share the coding and information, one could just share the entire project folder and the recipients would be able to run the coding without needing to update the location of the data file (Kam 2020b).
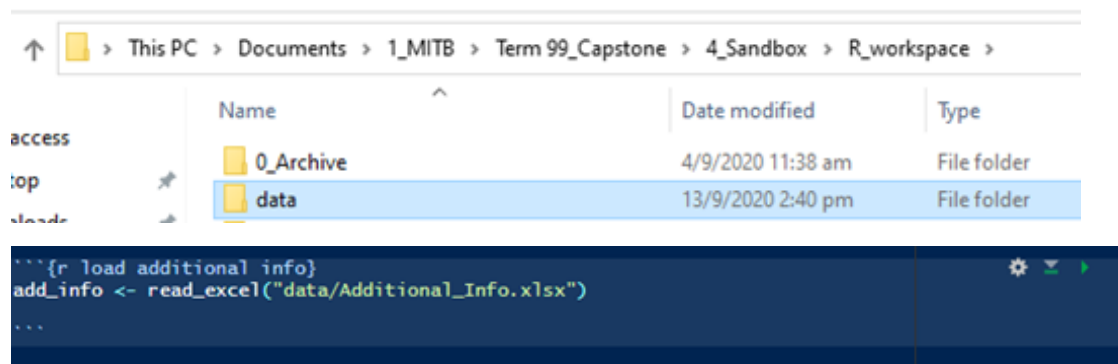
↑ ▸ This PC ▸ Documents ▸ 1_MITB ▸ Term 99_Capstone ▸ 4_Sandbox ▸ R_workspace ▸

Name	Date modified	Type
0_Archive	4/9/2020 11:38 am	File folder
data	13/9/2020 2:40 pm	File folder

```{r load additional info}
add_info <- read_excel("data/Additional_Info.xlsx")
```

Figure 23: Recommended approach in structuring folder

Support level from the packages

One of the other important factors to consider while performing a data science project is to find out the support level from the packages or software. This is particularly crucial for open-source software. Figure 24 is an example of Github where all the current and past issues with packages are being shown. There are circumstances that the details on the support contact were outdated as packages were developed by others during their PhD. Fortunately, RStudio has a professional team that maintains a list of R packages. RStudio team has a periodic release on new features or fixes on some of the packages as shown in Figure 25.
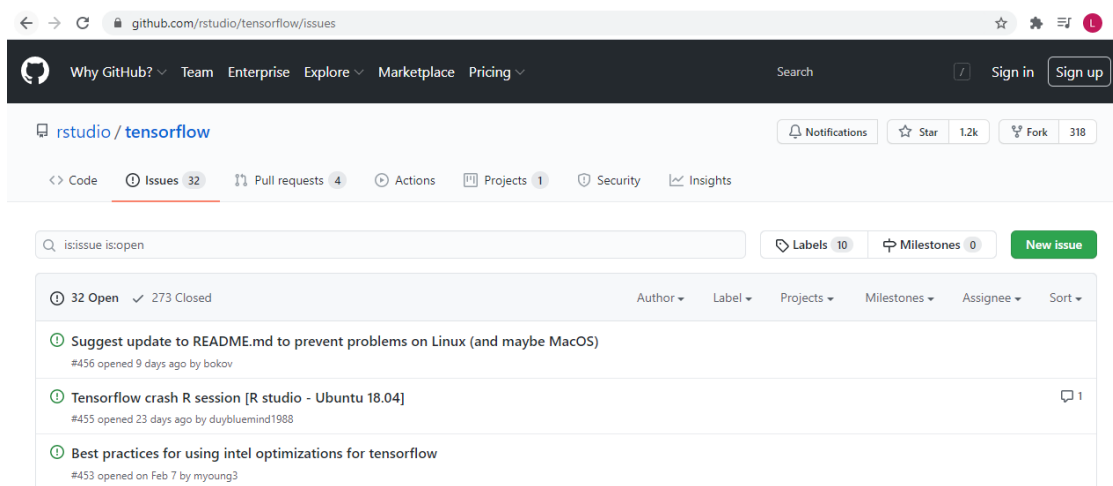
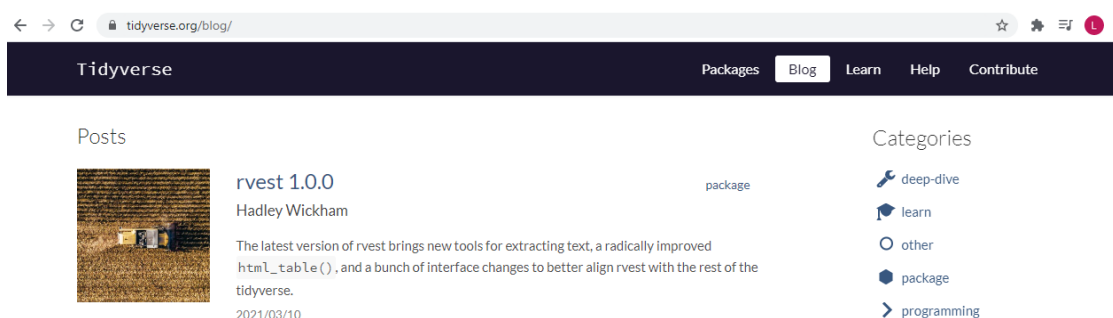Figure 24: Example of Github issue page for a particular package

Figure 25: Screen shot of Tidyverse Blog

Therefore, as mentioned earlier in the report, R is selected to demonstrate how the analytics project files can share across users without needing to acquire any license. **tidyverse** and **tidymodels** selected for this capstone project to demonstrate how this recommended workflow can be implemented for the actuarial science context.

Nevertheless, the modern data science concepts shared in this report can be applied to other programming languages (eg. Python, Julia, etc) as well.

6.0   Exploratory Data Analysis (EDA)

Tukey described the data analysis as procedures for analyzing data, techniques for interpreting the results of such procedures, ways of planning the gathering of data to make its analysis easier, more precise, or more accurate, and all machinery and results of (mathematical) statistics which apply to analyzing data (Tukey 1961). In other words, exploratory data analysis is a process that enables users to gain a better understanding of the underlying data structure and information contained in the dataset.

In one of the paper by the Casualty Actuarial Society, the authors highlighted that this step is crucial for determining the success or failure of the entire analysis since most of the machine

learning techniques are sensitive to the format and scale of the variables (Spedicato, Dutang, and Petrini 2018). Besides, some of the models are not appropriate if the data violates the assumptions of the model. For example, highly skewed data is not suitable for linear regression since linear regression assumes the relationship between target and response variables is linear.

Similarly, (Grolemund and Wickham 2016) also suggested in their book, R for Data Science, the best practice for the data science process should start with EDA, which involves data exploration, feature engineering, feature selection, and data cleaning.

Through EDA, the users would be able to identify the following:

- Are there any existing issues in the dataset?

- What kind of data cleaning is required for this dataset?

- Does the current dataset contain all the necessary information for analysis? If not, what information should the users gather?

- Is there any need to split data before building machine learning models to improve the overall accuracy in the prediction?

In this section, I will attempt to explain how different modern data science R functions are used to assist in explanatory data analysis. The detailed steps can be found in the HTML results attached under Appendix.

6.1   Import Data

The modern data science approach in importing data is by using **read_csv** function from readr package. **readr** is a R package that allows the users to import rectangular data (Wickham and Hester, n.d.). This package allows the users to read any delimiter files (including csv), fixed-width files, and Apache-style log files.

```
data_1 <- read_csv("data/actuarial_loss_train.csv")
```

Figure 26: Code chunk on read_csv function

If the data file is in excel format, one could use **readxl** package to import excel in the R environment. **readxl** allows the users to read in various types of excel files. Code hunk in Figure 27 shows how one could import the excel file into the environment by using **read_excel** function in **readxl** package. One could even select which excel sheet and how many rows to be imported by indicating the parameter in the function.

```
data <- read_excel("data/actuarial_loss_train.xlsx")
```

Figure 27: Code chunk on read_excel function

Note that base R also has **read.csv** function, allowing users to import csv files. However, **read_csv** is more efficient than **read.csv** as it takes less time to read in the dataset as shown

in Figure 28. This function also imports the data in tibble format, which provides the users with a tidy format of the data.

```
tic("Read data by using read_csv")
data_1 <- read_csv("data/actuarial_loss_train.csv")
toc()

## Read data by using read_csv: 0.61 sec elapsed

tic("Read data by using read.csv")
data_2 <- read.csv("data/actuarial_loss_train.csv")
toc()

## Read data by using read.csv: 1.12 sec elapsed
```

Figure 28: Time spent on importing data by using read.csv and read_csv function

Besides, **readr** is part of tidyverse package, which allows the users a more seamless experience in exploring and transforming the data. Most importantly, **read_csv** allows for reproducibility as the base R functions inherit some behavior from the user's operating system and environment variables. Hence, even the coding work on one computer does not guarantee it would work on another computer (Grolemund and Wickham 2016).

However, note that **read_csv** function does not convert characters into factors upon importing the data. Further data transformation may be required if the string needs to be in factor format for some of the machine learning models.

6.2 Data Quality

Conventionally, **summary** function from the base R function is used to summarize the characteristics of the different variables. This function would return generic information, which is the frequency count for categorical variables and frequency count on the different quantiles for continuous variables (Figure 29).

```
summary(data_1)

##  ClaimNumber       DateTimeOfAccident              DateReported
##  Length:54000     Min.   :1988-01-01 09:00:00   Min.   :1988-01-08 00:00:00
##  Class :character  1st Qu.:1992-06-30 07:00:00   1st Qu.:1992-08-04 00:00:00
##  Mode  :character  Median :1997-01-07 10:00:00   Median :1997-02-16 00:00:00
##                    Mean   :1997-01-03 05:08:31   Mean   :1997-02-11 01:00:17
##                    3rd Qu.:2001-07-09 11:00:00   3rd Qu.:2001-08-25 00:00:00
##                    Max.   :2005-12-31 10:00:00   Max.   :2006-09-23 00:00:00
##       Age            Gender           MaritalStatus     DependentChildren
##  Min.   :13.00   Length:54000      Length:54000      Min.   :0.0000
##  1st Qu.:23.00   Class :character  Class :character  1st Qu.:0.0000
##  Median :32.00   Mode  :character  Mode  :character  Median :0.0000
##  Mean   :33.84                                       Mean   :0.1192
##  3rd Qu.:43.00                                       3rd Qu.:0.0000
##  Max.   :81.00                                       Max.   :9.0000
##  DependentsOther    WeeklyWages      PartTimeFullTime   HoursWorkedPerWeek
##  Min.   :0.000000  Min.   :   1.0   Length:54000       Min.   :  0.00
```

```
## 1st Qu.:0.000000   1st Qu.: 200.0   Class :character   1st Qu.: 38.00
## Median :0.000000   Median : 392.2   Mode  :character   Median : 38.00
## Mean   :0.009944   Mean   : 416.4                      Mean   : 37.74
## 3rd Qu.:0.000000   3rd Qu.: 500.0                      3rd Qu.: 40.00
## Max.   :5.000000   Max.   :7497.0                      Max.   :640.00
## DaysWorkedPerWeek ClaimDescription   InitialIncurredCalimsCost
## Min.   :1.000      Length:54000      Min.   :       1
## 1st Qu.:5.000      Class :character   1st Qu.:     700
## Median :5.000      Mode  :character   Median :    2000
## Mean   :4.906                         Mean   :    7841
## 3rd Qu.:5.000                         3rd Qu.:    9500
## Max.   :7.000                         Max.   :2000000
## UltimateIncurredClaimCost
## Min.   :     122
## 1st Qu.:     926
## Median :    3371
## Mean   :   11003
## 3rd Qu.:    8197
## Max.   :4027136
```

Figure 29: Output from summary function

Often, this is not sufficient for data profiling purpose as profiling is the very first step in a series of iterative stages in the pursuit of finding what the data want to tell the users (Casas 2019). The author also mentioned that data has its dirtiness in practice and the users would need to sculpt it to expose its information to find answers and potentially new questions.

Therefore, good data profiling should cover the following items:

- Dataset health status (eg. total rows & columns, is any missing value or zero, and so on)

- Univariate analysis in categorical variables (eg. frequency count)

- Univariate analysis in numerical variables (eg. mean, quantiles)

This is crucial as it would guide the users to determine whether any data cleaning or transformation is required. Meanwhile, it could highlight to the users the existing issues within the dataset.

Over the years, many other data quality checking functions are being developed and introduced. These functions do provide users with richer information on the dataset. Therefore, **status** and **profiling_num** function from **funModeling** package, and **skim** function from **skimr** package are used to illustrate the benefits of using these modern data science functions.

Following are some descriptions of the relevant packages in this section:

- **funModeling** package is a very useful package that contains a wide range of different R functions to perform exploratory data analysis and data preparation.

- **skimr** package mainly focuses on summarizing the statistics of the data. The output of the **skim** function is in a professional compact horizontal format, which makes it easier for the users to read the results

- **status** function from **funModeling** offers the users to have a glance at the data quality (eg. is there any missing values, is there any excessive categories and so on). For example, Figure 30 shows there is an issue of excessive categories under ClaimDescription variable. There are more than 28,000 unique categories under Claim Description variable since ClaimDescription is likely to be a free text field for the claim officers to input the descriptions for a particular claim. This could adversely impact the performance of the machine learning model as the model will try to fit all unique categories, resulting in lower accuracy. Also, the granularity in categories does not seem to be representative of the underlying risks. In contrast, it might cause even more noise, resulting in a decrease in model performance. Hence, data cleaning and transformation on these variables are required before they can be used for machine learning.

```
status(data_1)
##                       variable q_zeros      p_zeros q_na          p_na q_inf p_inf
## 1               ClaimNumber        0 0.000000000     0 0.000000000     0     0
## 2          DateTimeOfAccident       0 0.000000000     0 0.000000000     0     0
## 3               DateReported       0 0.000000000     0 0.000000000     0     0
## 4                        Age       0 0.000000000     0 0.000000000     0     0
## 5                     Gender       0 0.000000000     0 0.000000000     0     0
## 6              MaritalStatus       0 0.000000000    29 0.000537037     0     0
## 7           DependentChildren   50639 0.937759259     0 0.000000000     0     0
## 8             DependentsOther   53506 0.990851852     0 0.000000000     0     0
## 9                 WeeklyWages       0 0.000000000     0 0.000000000     0     0
## 10            PartTimeFullTime       0 0.000000000     0 0.000000000     0     0
## 11          HoursWorkedPerWeek      29 0.000537037     0 0.000000000     0     0
## 12           DaysWorkedPerWeek       0 0.000000000     0 0.000000000     0     0
## 13            ClaimDescription       0 0.000000000     0 0.000000000     0     0
## 14 InitialIncurredCalimsCost       0 0.000000000     0 0.000000000     0     0
## 15 UltimateIncurredClaimCost       0 0.000000000     0 0.000000000     0     0
##              type unique
## 1       character  54000
## 2   POSIXct/POSIXt  36673
## 3   POSIXct/POSIXt   6653
## 4         numeric     68
## 5       character      3
## 6       character      3
## 7         numeric      9
## 8         numeric      5
## 9         numeric  13211
## 10      character      2
## 11        numeric    424
## 12        numeric      7
## 13      character  28114
## 14        numeric   1989
## 15        numeric  53999
```

Figure 30: Data quality report from status function

Besides, Figure 30 shows that there are missing values in the dataset. The common approach to overcome this issue is to impute the missing values through several methods, such as imputing missing values by using average values and so on. However, performing imputations without a good understanding of the data and business might introduce more noise and bias into the dataset, resulting in a drop in the accuracy of the model. Given that we do not have the context of this dataset to deduce the missing values and some of the machine learning models (eg. Statistical learning models) are unable to fit models with some missing values in the variables, model points with missing values are being dropped from the analysis. Even we remove the missing values from the analysis, it is unlikely this would have a significant impact on the model performance since the proportion of missing values in the dataset is less than 0.1%.

Another drawback of this function is it does not output other information, such as the values at different quantiles, which would often indicate whether there are existing issues (eg. outliers) in the data. This is where **profiling_num** function could complement the information not provided by **status** function. Results from **profiling_num** do include such additional information in the output. The results contain information such as mean, standard deviation, variation coefficient, different quantile values, skewness, kurtosis, and so on. This would come very handy when we want to investigate whether there are any signs of outliers, which helps us in determining the "trimming strategies" later in the analysis. Also, some of the machine learning algorithms are sensitive to the range of the variables. Without a proper transformation on the variables with ranges that vary widely is unlikely to produce a more accurate model. Again, from the results, we noted that all the claim-related variables are skewed. For example, the density for InitialIncurredClaimCost and UltimateIncurredClaimCost are very skewed.

```
profiling_num(data_1)
```

```
##                        variable          mean      std_dev variation_coef      p_01
## 1                            Age 3.384237e+01 1.212216e+01      0.3581949    16.000
## 2              DependentChildren 1.191852e-01 5.177800e-01      4.3443321     0.000
## 3                 DependentsOther 9.944444e-03 1.093475e-01     10.9958422     0.000
## 4                     WeeklyWages 4.163648e+02 2.486387e+02      0.5971654    42.000
## 5             HoursWorkedPerWeek 3.773508e+01 1.256870e+01      0.3330774    10.000
## 6              DaysWorkedPerWeek 4.905759e+00 5.521291e-01      0.1125471     2.000
## 7 InitialIncurredCalimsCost 7.841146e+03 2.058408e+04      2.6251360   160.000
## 8 UltimateIncurredClaimCost 1.100337e+04 3.339099e+04      3.0346152   207.911
##         p_05      p_25      p_50      p_75        p_95       p_99   skewness
## 1   18.0000   23.0000    32.000    43.000    56.0000      63.00  0.5363262
## 2    0.0000    0.0000     0.000     0.000     1.0000       3.00  5.1122373
## 3    0.0000    0.0000     0.000     0.000     0.0000       0.00 13.7064537
## 4  200.0000  200.0000   392.200   500.000   817.0055    1237.52  4.1226523
## 5   22.3815   38.0000    38.000    40.000    40.0000      60.00 24.1323038
## 6    4.0000    5.0000     5.000     5.000     5.0000       6.00 -3.3403751
## 7  315.0000  700.0000  2000.000  9500.000 30000.0000   75000.00 26.8529115
## 8  306.5569  926.3384  3371.242  8197.249 45224.1844  139024.97 37.5514607
##     kurtosis       iqr                 range_98
## 1    2.39387     20.00                 [16, 63]
## 2   33.00333      0.00                  [0, 3]
## 3  267.37575      0.00                  [0, 0]
## 4   71.01694    300.00           [42, 1237.52]
```

```
## 5  913.12756    2.00                          [10, 60]
## 6   21.23888    0.00                           [2, 6]
## 7 1891.10207 8800.00                       [160, 75000]
## 8 3943.49883 7270.91 [207.910970483, 139024.967362]
##                        range_80
## 1              [19, 52]
## 2               [0, 0]
## 3               [0, 0]
## 4          [200, 681.451]
## 5           [34.5, 40]
## 6               [5, 5]
## 7           [500, 18500]
## 8 [423.89078703, 24005.783949]
```

Figure 31: Data quality report from profiling function

However, one drawback is this function only shows the value at 99% percentile, instead of the maximum value. In the dataset, the value at 99 percentile for UltimateIncurredClaimCost is about 139,000 and the max value of the variable is more than 4,000,000, which is about 29 times the value at 99 percentile. Hence, extreme value/outliers might be missed out if the maximum value is not checked. Also, **profiling_num** function only shows the data quality for the numeric variables.

Results generated from **skim** function combine the benefits of both **status** and **profiling_num**. For example, **profiling_num** and **skim** do provide the figures under different percentiles for the numeric figures. **skim** also include a small histogram under each continuous variable to give the users a flavor of the distribution. Apart from that, the **skim** function also shows the summary statistics for categorical variables, like what is shown in the output under **status** function. **skim** function also includes frequency count for the top 4 variables under each factor variable.

```
skim(data_1)
```

Data summary

Name	data_1
Number of rows	54000
Number of columns	15

_____

Column type frequency:

character	5
numeric	8
POSIXct	2

_____

Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
ClaimNumber	0	1	9	9	0	54000	0
Gender	0	1	1	1	0	3	0
MaritalStatus	29	1	1	1	0	3	0
PartTimeFullTime	0	1	1	1	0	2	0
ClaimDescription	0	1	3	94	0	28114	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Age	0	1	33.84	12.12	13.00	23.00	32.00	43.00	81	▆▆▂▁▁
DependentChildren	0	1	0.12	0.52	0.00	0.00	0.00	0.00	9	▇▁▁▁▁
DependentsOther	0	1	0.01	0.11	0.00	0.00	0.00	0.00	5	▇▁▁▁▁
WeeklyWages	0	1	416.36	248.64	1.00	200.00	392.20	500.00	7497	▇▁▁▁▁
HoursWorkedPerWeek	0	1	37.74	12.57	0.00	38.00	38.00	40.00	640	▇▁▁▁▁
DaysWorkedPerWeek	0	1	4.91	0.55	1.00	5.00	5.00	5.00	7	▁▁▁▇▁
InitialIncurredClaimsCost	0	1	7841.15	20584.08	1.00	700.00	2000.00	9500.00	2000000	▇▁▁▁▁
UltimateIncurredClaimCost	0	1	11003.37	33390.99	121.89	926.34	3371.24	8197.25	4027136	▇▁▁▁▁

Variable type: POSIXct

skim_variable	n_missing	complete_rate	min	max	median	n_unique
DateTimeOfAccident	0	1	1988-01-01 09:00:00	2005-12-31 10:00:00	1997-01-07 10:00:00	36673
DateReported	0	1	1988-01-08 00:00:00	2006-09-23 00:00:00	1997-02-16 00:00:00	6653

Figure 32: Data quality report from skim function

It does show that there are some data issues based on the data quality report from the **skim** function. For example, the maximum value for HoursWorkedPerWeek is about 640 hours. This is not reasonable as one week only contains about 168 hours. Hence, HoursWorkedPerWeek variable does seem to contain invalid data. Also, the maximum age of the workers does seem odd as typically the worker compensation insurance is designed

for blue-collar workers due to exposure to the risk of being injured at the workplace. So, the maximum age in the dataset does seem a bit too old for a blue-collar worker.

Furthermore, the **skim** outputs the results in a tidier format. I would recommend using this function to generate the output so that it is easier for the audiences to follow through with the result file. Nevertheless, the output context from these functions differs, hence it is probably worthwhile to explore the data with these functions to check the data quality during the data exploratory stage.

Next, the **tukey_outlier** function from the **funModeling** package is being used to check the cutoff value before the data point is considered an outlier. This is crucial as outliers can impact the performance of the machine learning models by introducing a global bias (Alfredo, Dutang, and Petrini 2018).

To demonstrate this, a few of the variables that are feature engineered from the existing dataset are passed into the **tukey_outlier** function to compute the necessary cutoff for outliers.

Following are the definitions for the derived variables:

Derived Variable	Descriptions
init_ult_diff	Difference between Initial and Ultimate Claim Amount
num_week_paid_init	Number of Weeks Paid Initially Estimated
num_week_paid_ult	Number of Actual Weeks Paid
day_diff	Number of Day Difference between Accident Date and Reported Date

Figure 33: Definition of the Derived Variables

Next, the derived variables are being passed into the **tukey_outlier** function. Note that day_diff is not checked as it is common that there is a delay between the accident date and reported date. There are scenarios where it takes a longer time to settle claims due to the complexity of the claim situation. Hence, it is not reasonable to remove these valid claims.

```
tukey_outlier(data_1$init_ult_diff)

## bottom_threshold     top_threshold
##       -6393.157          7274.613

tukey_outlier(data_1$num_week_paid_init)

## bottom_threshold     top_threshold
##        -54.92258          78.13011

tukey_outlier(data_1$num_week_paid_ult)

## bottom_threshold     top_threshold
##        -58.69593          84.52272
```

Figure 34: Thresholds of the Derived Variables

prep_outliers is the function to treat outliers in the **funModeling** package. However, the way how the outliers are treated by using the **prep_outliers** function does not align with the intentions. The function set the threshold as the values of the outliers. For example, the maximum value of init_ult_diff is about 3,980,000 and the **prep_outliers** function will set the value of this data point to be 7,274. This does not look reasonable, hence I have manually removed the outliers by using the **filter** function.

Following are criteria selected to remove the outliers:

- Remove any data points with HoursWorkedPerWeek as 168 or more

- Remove any data points beyond the range of the threshold specified by the outlier functions

```
data_2 <- data_1 %>%
  filter(HoursWorkedPerWeek < 168,
         init_ult_diff > -6500,
         init_ult_diff < 7300,
         num_week_paid_init < 80)
```

Figure 35: Outliers Removal

Nevertheless, it is also recommended to check with the business units on the outliers so that we know these outliers are valid values or a systematic error in the claim process. This highlights the importance of understanding the business context, instead of just blindly following what the models suggested. Without a good understanding of the business, the users might exclude the data points incorrectly, resulting in an understated estimated claim cost.

6.3   Feature Selection

One of the key steps in the machine learning process is to perform feature selection. Feature selection is the process of reducing the number of input variables when developing a predictive model (Brownlee 2020). The author also mentioned that it is desirable to reduce the number of input variables to both reduce the computation cost of modeling and, in some cases, to improve the performance of the model.

The author also summarized the various feature selection techniques into different categories as following:

Derived Variable	Descriptions
Unsupervised Method	Do not use the target variable (e.g. remove redundant variables)
Wrapper	Search for well-performing subsets of features
Filter	Select subsets of features based on their relationship with the target (eg. based on the statistical method or feature importance method

| Intrinsic | Algorithms that perform automatic feature selection during training |
| Dimensionality Reduction | Project input data into a lower-dimensional feature space |

Figure 36: List of Feature Selections

Below is the summary of how one could perform feature selection on the different type of variables (Brownlee 2020):
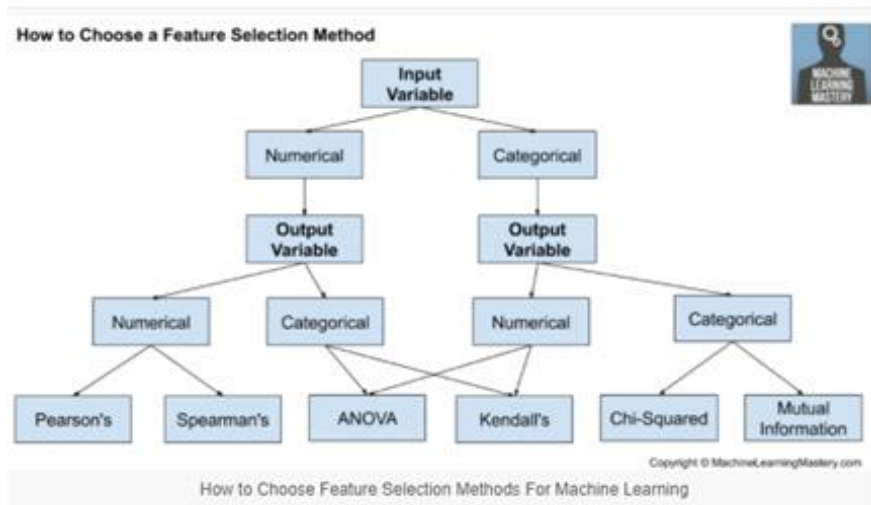

Figure 37: Feature selection techniques

The purpose of the feature selection analysis is to help in understanding the relative strength of each variable in predicting the outcome. In this capstone, we will be explored filter-based and intrinsic feature selection methods. Demonstrations will be included to show how to implement these methods in R code.

Filter-based Feature Selection

Currently, various R packages support users in performing filter-based feature selection:

Tasks	R Package
Frequency Count Plot	ggplot2
Relationship between categorical target variable and categorical input variables	funModeling, ggmosaic, infer
Relationship between categorical target variable and continuous input variables	funModeling
Relationship between continuous target variable and categorical input variables	ggstatsplot
Relationship between continuous target variable and continuous input variables	corrplot

Figure 38: R Packages Support Feature Selection

Note that as the target variable is not a categorical input, hence I have not performed any analysis on understanding the relationships between categorical target and predictors for this dataset. As the various methods were being explored to understand how the different R packages can explain the relationships between categorical target variables and predictors during the experiment stage, hence the discussions on those methods can be found under Appendix.

Intrinsic Feature Selection

(Kuhn and Johnson 2013) explained in their book that some machine models, such as tree- and rule-based models, multivariate adaptive regression splines (MARS), and LASSO intrinsically conduct feature selection. This is because the algorithms for these machine learning models will choose the variables that give the highest model performance at each step/split. Therefore, these models have indirectly performed feature selection.

To demonstrate this, tree-based models and MARS will be built to analyze whether there is any improvement in the model accuracy by using such models.

6.3.1 Frequency Count Plot

Figure 8 has discussed the purpose of visualization. Currently, base R also provides the basic plotting function. However, the function only able to produce static graphs, and the choices of the graph types are very limited with the base R function.

Below are the graphs of WeeklyWages vs init_ult_diff by using different functions from base R. There are some data points cluttered at around init_ult_diff = 0. However, we are unable to zoom into the data points that are cluttered together by using functions from Base R. Also, the base R plot function does have limitations of the different formatting on the graph.

```r
plot(data_1$init_ult_diff, data_1$WeeklyWages, type = "p", main = "WeeklyWage
s vs init_ult_diff", xlab = "init_ult_diff", ylab = "WeeklyWages")
```
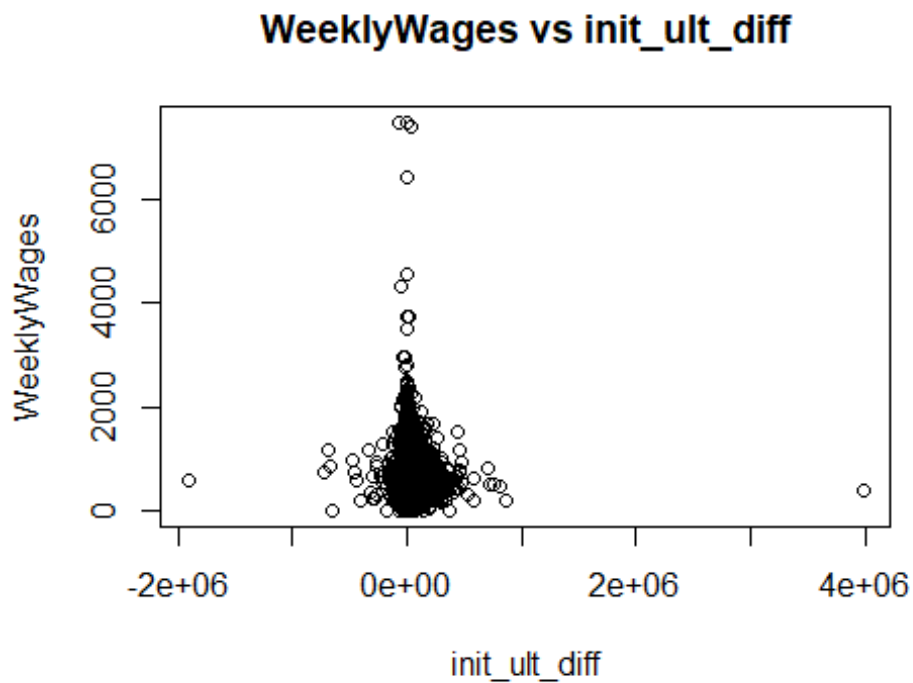
WeeklyWages vs init_ult_diff

Figure 39: Graph of WeeklyWages vs init_ult_diff (using the plot function from Base R)

```
stripchart(init_ult_diff ~ WeeklyWages, data = data_1, frame = FALSE, method
= "jitter", pch = 19, main = "WeeklyWages vs init_ult_diff", xlab = "init_ult
_diff", ylab = "WeeklyWages")
```
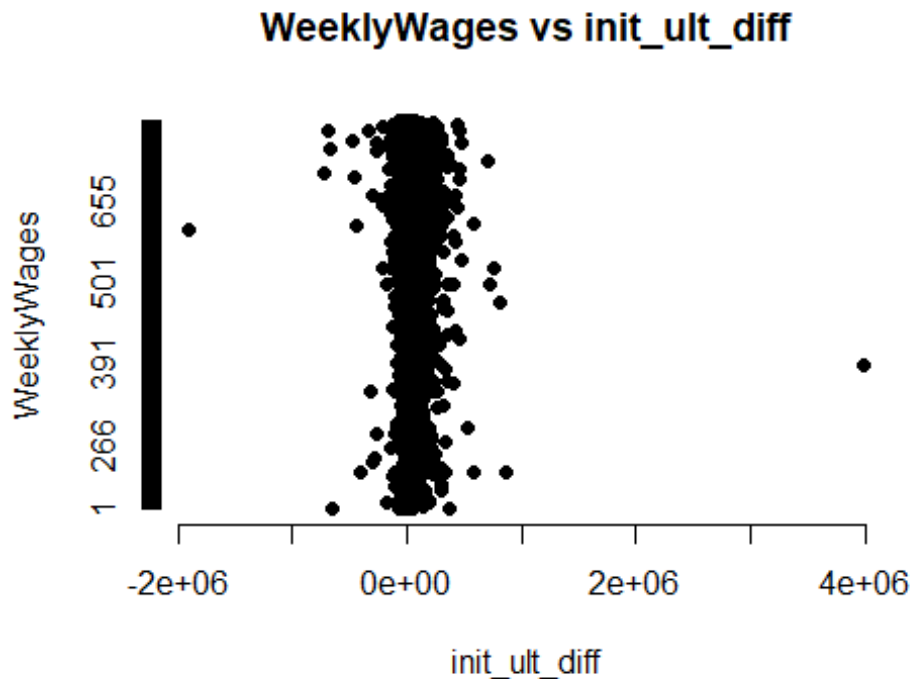
WeeklyWages vs init_ult_diff

Figure 40: Graph of WeeklyWages vs init_ult_diff (using the stripchart function from Base R)

To overcome the issues mentioned above, **ggplot2** is being selected to demonstrate how this package can be used to help the users to visualize by using different graph types it is currently supported. It is R package to plot various types of graphs. **ggplot2** follows the grammar of graphics, offering the users a wide range of flexibility to make the changes to the details of the graph (eg. axis title, axis range, the color of the bars, etc). **ggplot2** is also part of the **tidyverse** family, making it easier to plot graphs after cleaning the data by using the relevant R packages in **tidyverse**.

Figure 41: Grammar of graphics

Furthermore, the **ggplotly** function from the **plotly** is being used to include interactivity to the visualization prepared by **ggplot**. **plotly** is a package powered by the JavaScript library

plotly.js (Sievert 2019). It allows users to build an interactive graph without requiring the users to code on javascript. Same as **ggplot2**, this package also adopts the principles of the grammar of graphics, making it easier for the users to make modifications to the graphs. Besides that, **plotly** also adopted the principle of *'Overview first, zoom and filter, then details on demands'*, where this principle was introduced by Ben Shneiderman in his article 'The eyes have it: A task by data type taxonomy for information visualization (Shneiderman 2005). Such interactivity would allow users to interact with the graphs to inspect the graph and draw more insights from the dataset.

Currently, **plotly** allows users to build interactive graphs through two methods:

- *Method 1:* Directly pass the information (eg. data, variables) to the **plot_ly** function in the **plotly** package

- *Method 2:* Build the graph by using various functions under the **ggplot2** package, then wrap the graph object by using the **ggplotly** function to convert the graph to an interactive graph

The second method is selected to demonstrate how one could include interactivity into the graph. By adding just one more line to the code to wrap the graph object by using the **ggplotly**, we can effectively convert the graph into an interactive graph. This also highlights the benefits of the modern data science R package.

Figure 42 is the code to plot the static graph by using the **ggplot** function. As I would like to color the data points depending on whether their value is positive or negative so that it is easier to visually inspect later. To do so, I have used **mutate** function to create a new variable (ie. sign_ind) and **ifelse** function is used to set the variable to be "pos" or "neg" depending on whether the init_ult_diff is positive or negative.

After that, the dataset will be passed to **ggplot** function. init_ult_diff is set to be the x-axis and WeeklyWages is set to be the y-axis. I also set the data points to be colored based on the sign_ind. Next, I choose the scatter point to be the graph type, hence **geom_point** is called. As that are many data points are cluttered together as shown in the graph earlier on, I have set the alpha to be 0.3 so that the points have low opacity and it would be easier for us to identify the density of scatter points on the graph.

```
graph_wages_clmdiff <- data_1 %>%
  mutate(sign_ind = ifelse(init_ult_diff > 0, "pos", "neg")) %>%
  ggplot(aes(init_ult_diff, WeeklyWages, color = sign_ind)) +
  geom_point(alpha = 0.3)
```

Figure 42: Scatter Plot (WeeklyWages vs init_ult_diff)

To convert the static graph into an interactive graph, we just need to pass the graph object into **ggplotly** function.

```
ggplotly(graph_wages_clmdiff)
```
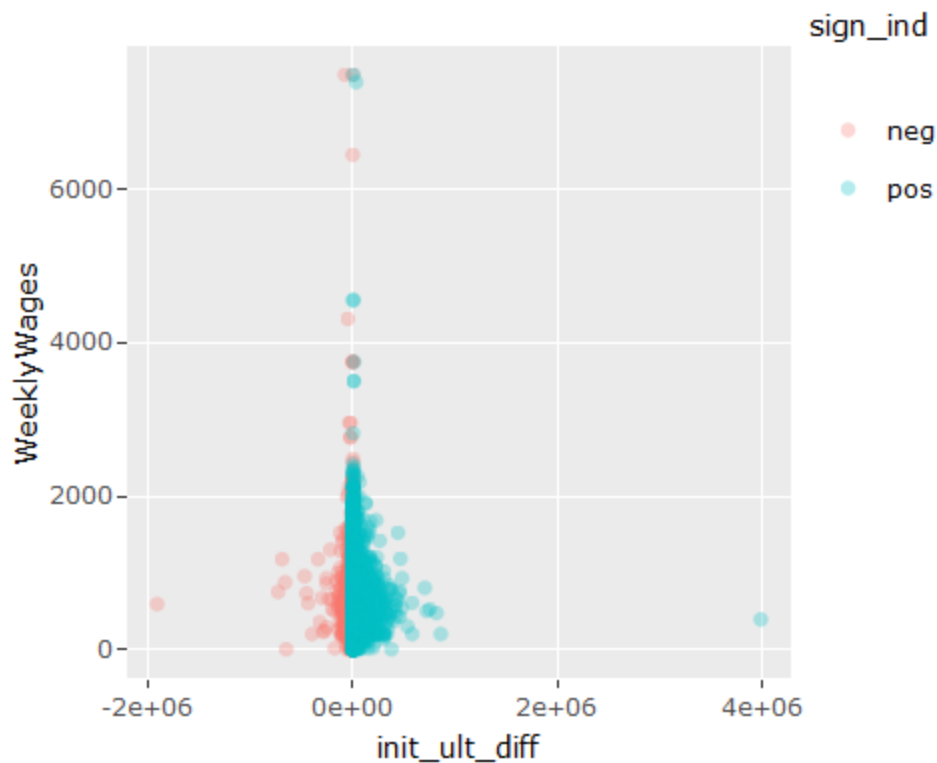
Figure 43: Interactive Scatter Plot (WeeklyWages vs init_ult_diff)

By including the interactivity, it allows the users to investigate the graph to uncover the underlying relationships between the variables. Following is the screen shot on how the graph looks like if we were to zoom to focus on one part of the graph:
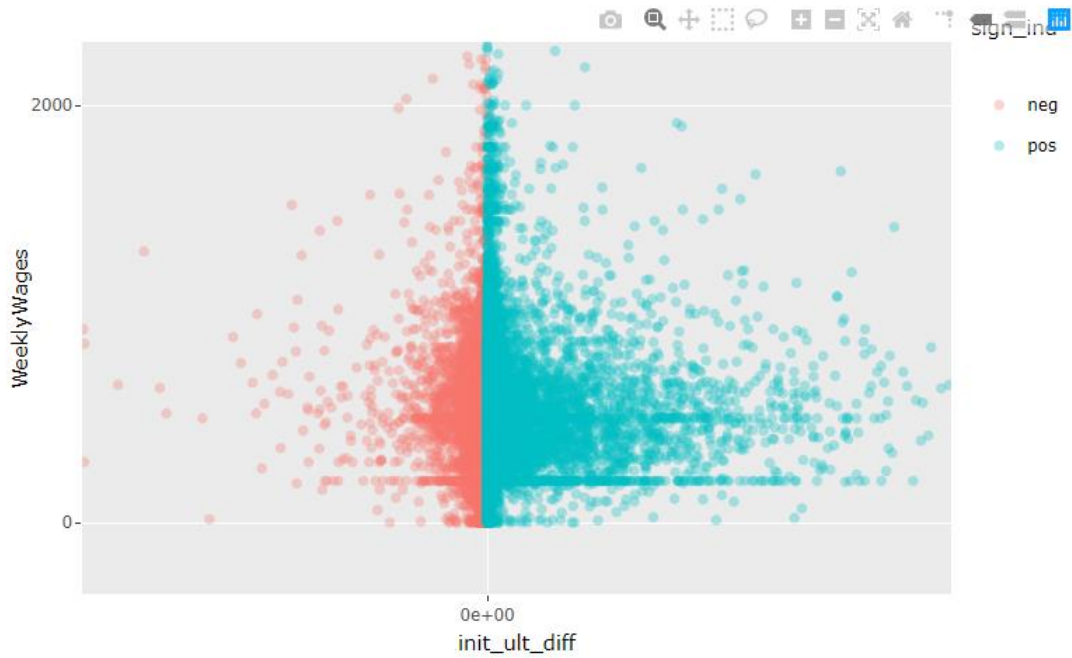
Figure 44: Zoom into Scatter Plot

6.3.2 Relationship between continuous target variable and categorical input variables

Conventionally users would perform an analysis of variance (ANOVA) test to check the strength of the relationship by using **aov** function from **stat** package.
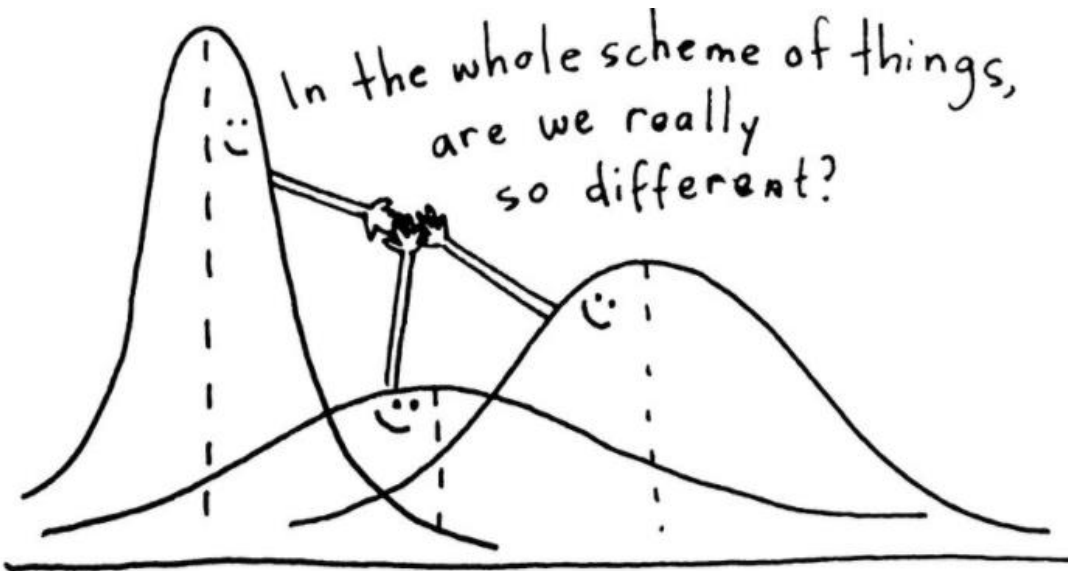
Figure 45: Illustration on ANOVA

ANOVA allows us to check whether the groups are significantly different from one another. Below is the null and alternative hypothesis under ANOVA:

$H_0: \mu_1 = \mu_2 = \ldots = \mu_n$

$H_a: \mu_1 \neq \mu_2 \neq \ldots \neq \mu_n$

```
aov_clmdiff_injside <- aov(init_ult_diff ~ MaritalStatus, data = data_3)
summary(aov_clmdiff_injside)

##                   Df    Sum Sq   Mean Sq F value Pr(>F)
## MaritalStatus      2 7.176e+08 358788633   83.27 <2e-16 ***
## Residuals      46483 2.003e+11   4308778
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 27 observations deleted due to missingness
```

Figure 46: ANOVA Test(Singh 2018)

However, this method is unable to show which groups are significant from the rest. The ANOVA test above shows that there is statistical evidence that injury_side is not independent of init_ult_diff. However, we are unable to see which categories are different from the rest.

As such, the **ggbetweenstats** R function from the **ggstatplot** package is selected to demonstrate how we might plot out the claim count distribution by driver age, while not losing the statistical results. **ggstatplot** is an extension of the **ggplot2** package for creating graphics with details from statistical tests included in the information-rich plot themselves (Patil, n.d.). **ggstatplot** supports different types of analysis as shown under Figure 47. The function has effectively combined visualization and statistics into one single graph.

The table below summarizes all the different types of analyses currently supported in this package-

Functions	Description	Parametric	Non-parametric	Robust	Bayes Factor
ggbetweenstats	Between group/condition comparisons	Yes	Yes	Yes	Yes
ggwithinstats	Within group/condition comparisons	Yes	Yes	Yes	Yes
gghistostats, ggdotplotstats	Distribution of a numeric variable	Yes	Yes	Yes	Yes
ggcorrmat	Correlation matrix	Yes	Yes	Yes	Yes
ggscatterstats	Correlation between two variables	Yes	Yes	Yes	Yes
ggpiestats, ggbarstats	Association between categorical variables	Yes	NA	NA	Yes
ggpiestats, ggbarstats	Equal proportions for categorical variable levels	Yes	NA	NA	Yes
ggcoefstats	Regression model coefficients	Yes	Yes	Yes	Yes
ggcoefstats	Random-effects meta-analysis	Yes	NA	Yes	Yes

Figure 47: Various Statistical Tests supported under ggstatsplot

Following are the explanations of the different figures on the visualization of **ggbetweenstats** function:

Template for Frequentist analysis

test   parameter   statistic   significance   effect size type + estimate + confidence intervals   number of observations

$$t_{\text{Welch}}(281.95) = -10.75, p = 8.31e\text{-}23, \hat{g}_{\text{Hedges}} = -1.27, \text{CI}_{99\%}[-1.61, -0.94], n_{\text{obs}} = 284$$

Template for Bayesian analysis

evidence in favor of null over alternative hypothesis   natural logarithm of Bayes Factor   posterior type + estimate + credible intervals   prior type and value

$$\log_e(\text{BF}_{01}) = -6.20, \delta_{\text{difference}}^{\text{posterior}} = -5.06, \text{CI}_{95\%}^{\text{HDI}}[-6.75, -3.53], r_{\text{cauchy}}^{\text{JZS}} = 0.71$$

Figure 48: Explanations on the Figures on ggbetweenstats function

In Figure 46, the ANOVA test indicates that the results from the different MaritalStatus are statistically significant from the rest. **ggbetweenstats** function is used to further understand which MaritalStatus are different from the rest.

```
ggbetweenstats(data_3,
               x = MaritalStatus,
               y = init_ult_diff,
               pairwise.comparisons = TRUE,
               title = "ANOVA Test on MaritalStatus vs init_ult_diff",
               ggtheme = ggplot2::theme(axis.text.x = element_text(angle = 90)
),
               package = "RColorBrewer",
               palette = "Set3")
```
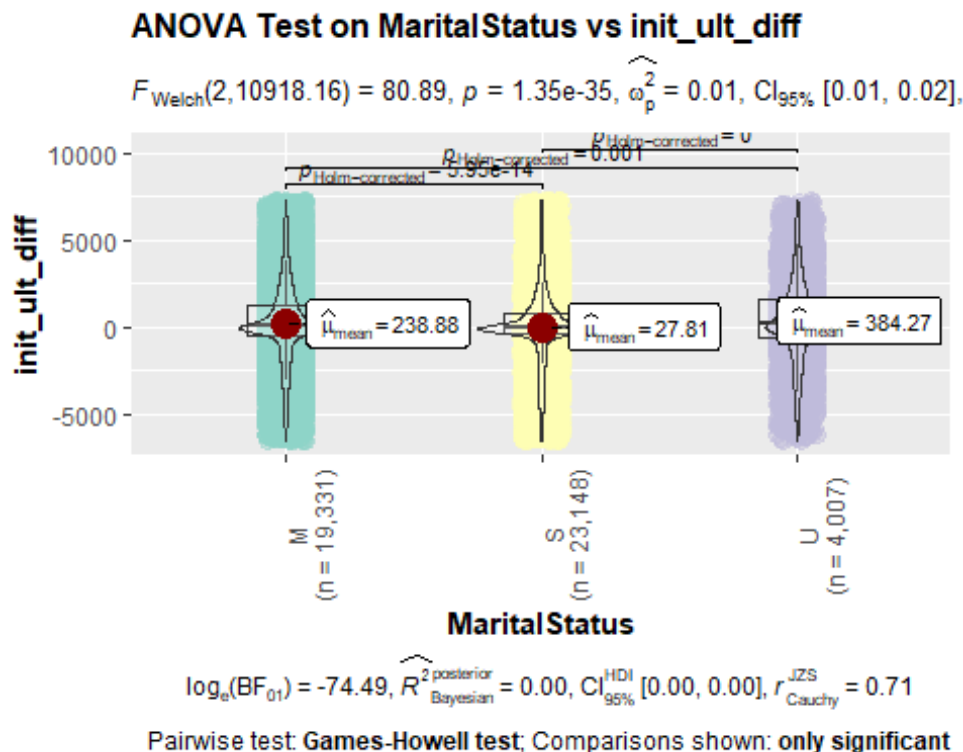

Figure 49: ggbetweenstats for MaritalStatus

In the code chunk above, aside from indicating the dataset, x, and y variables, I have indicated to show which pairs are statistically different by indicating *TRUE* for **pairwise.comparisons** argument. As the **ggstatsplot** package is an extension of the **ggplot2** package, the function allows us to pass additional **ggplot2** arguments into the function. In the code above, I have indicated the label on the x-axis to be rotated by 90 degrees so that they would not overlap one another in the graph.

Also, as the number of default colors in the **ggstatsplot** package is very limited, hence **Set3** color palette from **RColorBrewer** package is chosen as shown under Figure 49.

The line above various pairs shown in the graph indicates the pairs are statistically different from one another. The graph indicates that the average init_ult_diff under each category in MaritalStatus are indeed different from one another.

There will no line on top of the pairs if the pairs are not statistically different. The graph below shows that the average init_ult_diff under PartTime and FullTime is not statistically different. This implies that this variable is unlikely to have high predictive power in the machine learning model.

ANOVA Test on PartTimeFullTime vs init_ult_diff

$t_{\text{Welch}}(4451.91) = 0.81$, $p = 0.416$, $\widehat{g}_{\text{Hedges}} = 0.01$, $CI_{95\%}$ [-0.02, 0.05], n_{ot}

$\log_e(BF_{01}) = 3.60$, $\widehat{\delta}_{\text{difference}}^{\text{posterior}} = -27.12$, $CI_{95\%}^{\text{HDI}}$ [-96.03, 38.15], $r_{\text{Cauchy}}^{\text{JZS}} = 0.71$
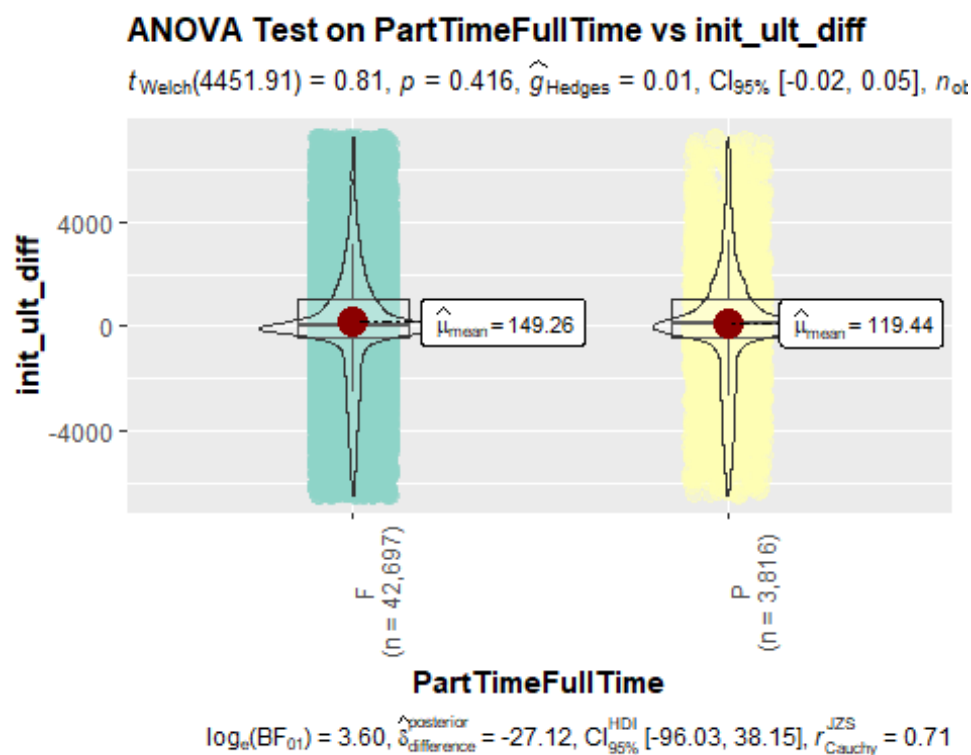
Figure 50: ggbetweenstats for PartTimeFullTime

This again demonstrates that the modern data science R package skillfully presents the information to understand while including the relevant information in the graphics. This has enabled users to perform data analysis more efficiently.

6.3.3 Relationship between continuous target variable and continuous input variables

It is also important to check the relationship between the continuous variables. Conventionally, the **pairs** function is used to generate the correlation matrix. However, the output from this function is not ideal, especially when the number of data points is rather large (i.e. more than 500 observations) (Kam 2019).

To contrast the difference between the correlation plotting function by using **base R** and **corrplot** function, both methods are used to plot the corrlation graph by using the same set of data. The time taken each method takes to plot out the correlation is also measured and compared as shown below.

Conventional Correlation Plotting in **Base R**

```
data_1_num <- data_1 %>%
  select_if(is.numeric)


tic("Conventional Method for Correlation Matrix")
panel.cor <- function(x, y, digits=2, prefix="", cex.cor, ...) {
  usr <- par("usr")
  on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- abs(cor(x, y, use="complete.obs"))
  txt <- format(c(r, 0.123456789), digits=digits)[1]
  txt <- paste(prefix, txt, sep="")
  if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
  text(0.5, 0.5, txt, cex = cex.cor * (1 + r) / 2)
}


pairs(data_1_num, upper.panel = panel.cor)
```
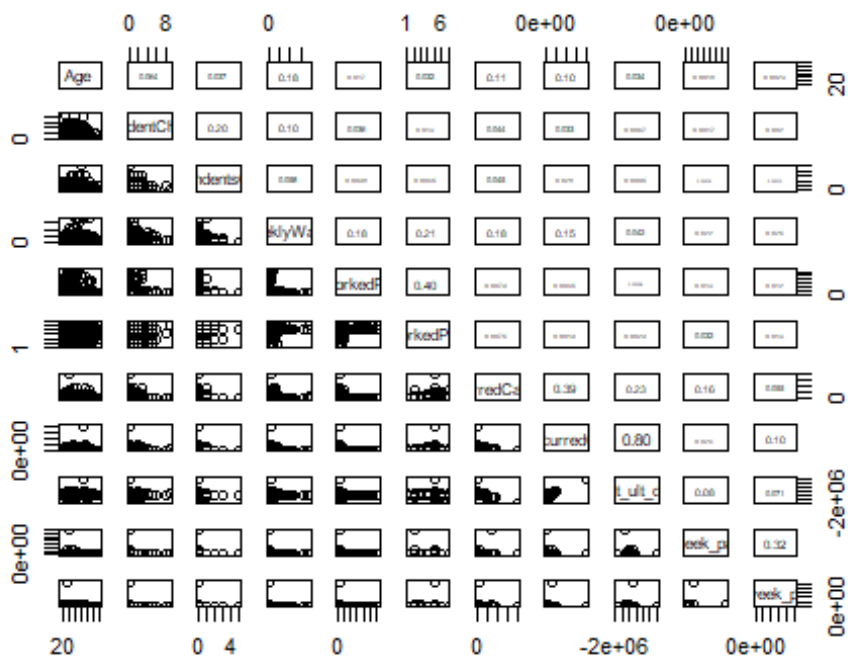


```
toc()
```

```
## Conventional Method for Correlation Matrix: 245.13 sec elapsed
```

Figure 51: Correlation Plot from Base R Function

One of the issues with the base R correlation plot function is that the data points tend to cluster together when the dataset size grows. For example, we are unable to tell how the data

points are distributed based on the graph of Age vs DayWorkedPerWeek (ie. the graph on the first column and sixth row). The box size is too small for the function to visualize the underlying relationship between Age and DayWorkedPerWeek. Also, color is often a better visualization choice when we need to visualize the relativity between the figures.

Therefore, the **corr_plot** function from the **corrplot** package is selected to demonstrate how one could better visualize the correlation matrix. **corrplot** package allows users to display the correlation matrix in graphical format.

Correlation Plotting by using **corrplot** *package*

```
tic("corrplot")
corrplot(cor(data_1_num, use="pairwise.complete.obs"),
         method = "number",
         type = "upper",
         tl.cex = 0.65,
         number.cex = 0.65,
         diag = FALSE)
```
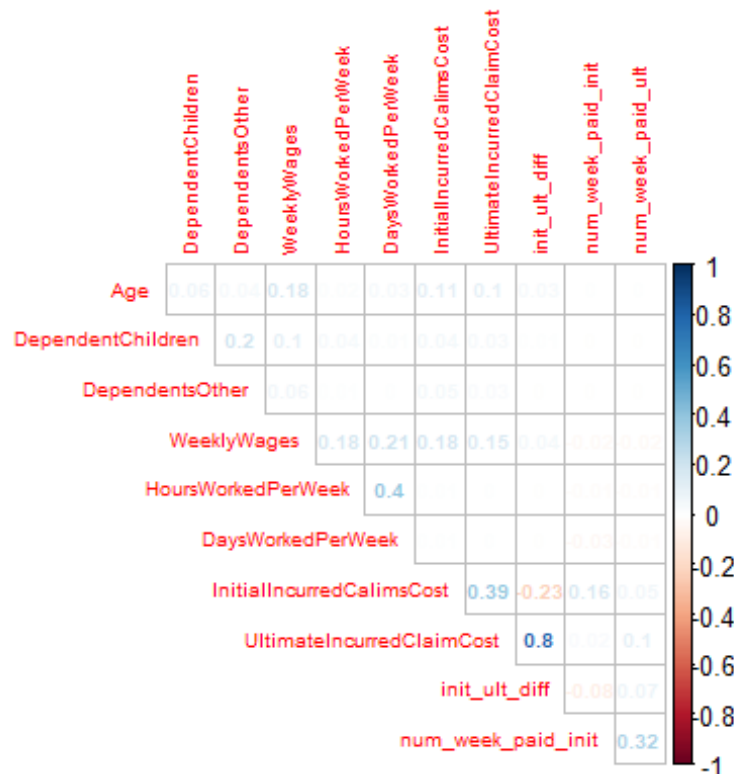


```
toc()
```

```
## corrplot: 0.47 sec elapsed
```

Figure 52: Correlation Plot from corr_plot Function

Besides that, to generate the same output, the conventional **pairs** function takes a much longer time to output the necessary results. With about 54,000 data points and 11 variables,

the **pairs** function took more than 1 min to generate the correlation output, where the **corr_plot** function only took less than 20 sec.

Also, the **corr_plot** function does provide the users the flexibility to illustrate the correlation in different formats such as color, ellipse, and so on. This can be quite handy especially when there are a lot of variables in the dataset and other methods such as color would be a better alternative to illustrate the correlation, instead of numbers. Again, these demonstrate the superior of these modern data science R packages.

```
corrplot(cor(data_1_num, use="pairwise.complete.obs"),
         method = "color",
         type = "upper",
         tl.cex = 0.65)
```
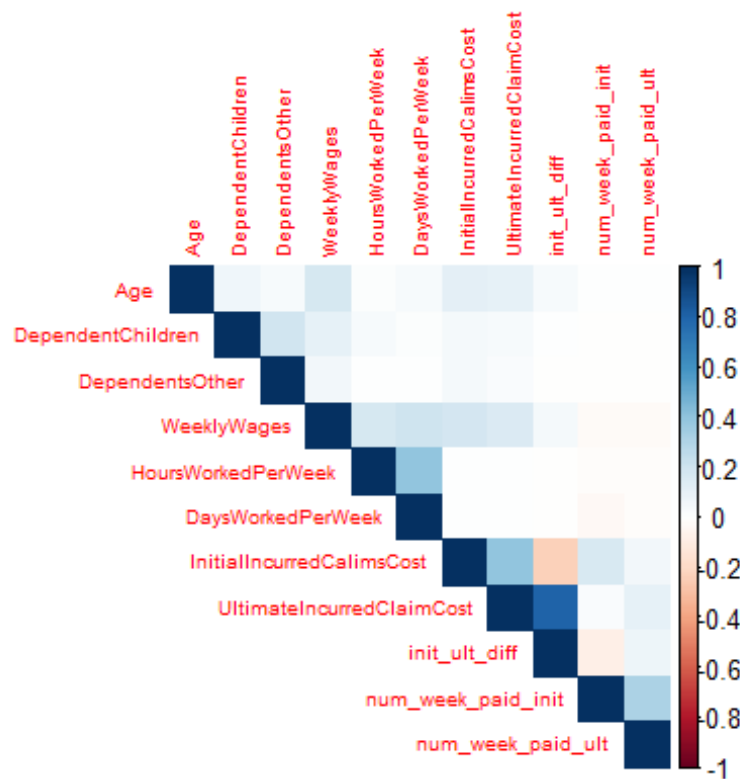

Figure 53: Correlation Plot from corr_plot Function (illustrated by using colors)

To visualize the relationship between the variables, it would be better to visualize them separately, instead of trying to visualize them within the small box.

```
data_1 %>%
  mutate(sign_ind = ifelse(init_ult_diff > 0, "pos", "neg")) %>%
  ggplot(aes(init_ult_diff, Age, color = sign_ind)) +
  geom_point(alpha = 0.3)
```
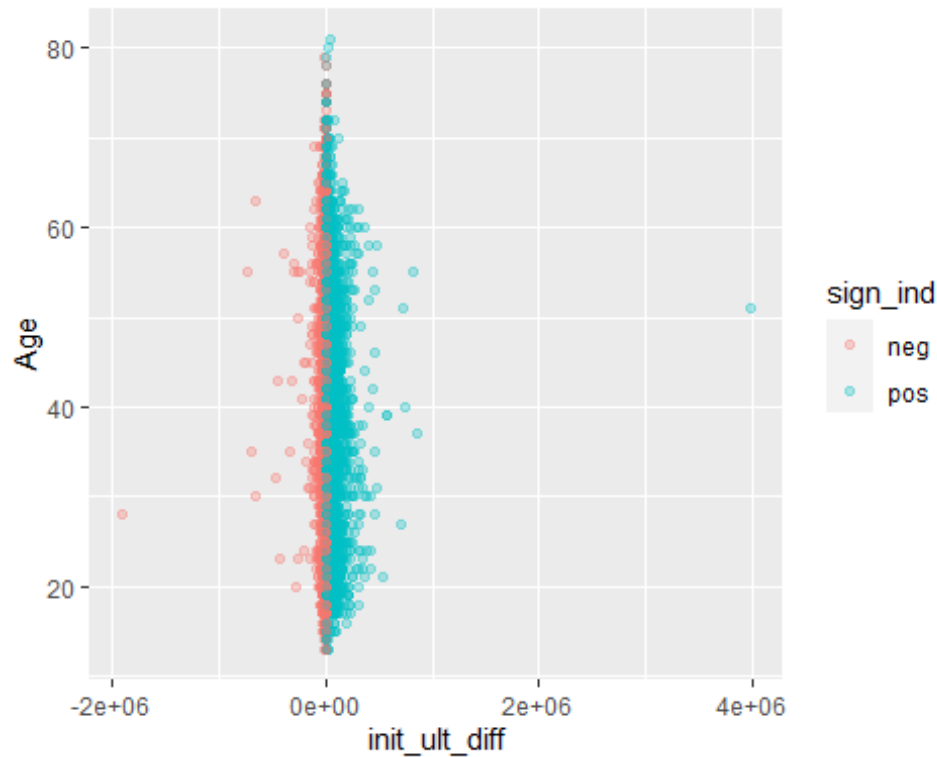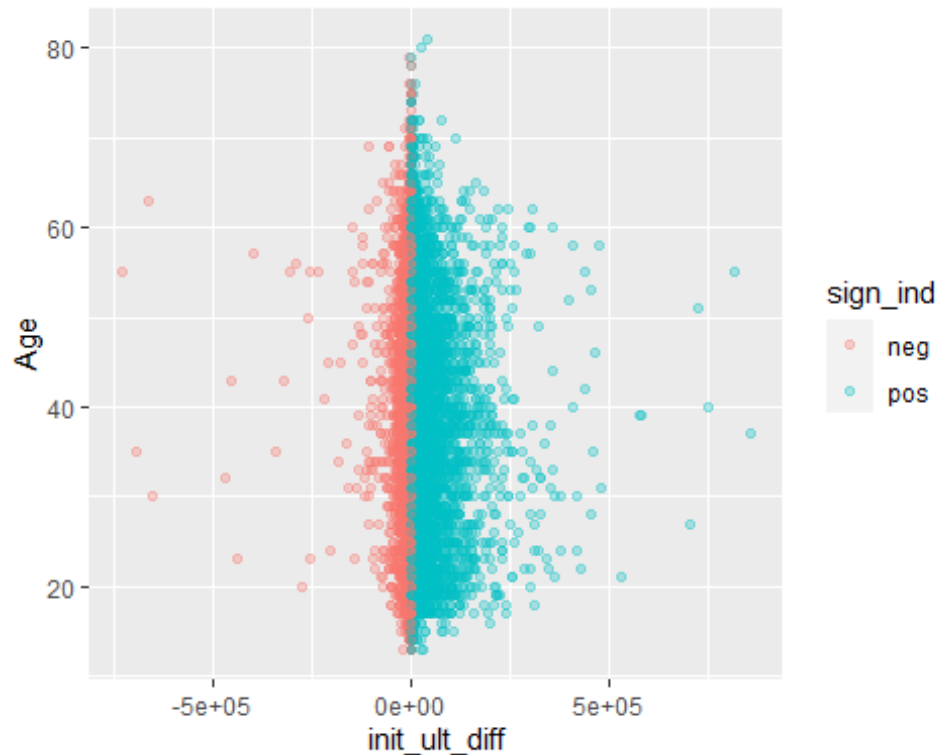
Figure 54: Scatter Plot (Age vs init_ult_diff)

There are two extreme values in the graph, resulting in most of the points cluttered around init_ult_diff = 0. So to better visualize how the points are scattered around init_ult_diff = 0, the scatter plot is re-plot with the removal of the two extreme values.

```
data_1 %>%
  mutate(sign_ind = ifelse(init_ult_diff > 0, "pos", "neg")) %>%
  filter(init_ult_diff < 1000000 & init_ult_diff > -1000000) %>%
  ggplot(aes(init_ult_diff, Age, color = sign_ind)) +
  geom_point(alpha = 0.3)
```

Figure 55: Scatter Plot (Age vs init_ult_diff) after Removing Extreme Values

From the graph, it seems like more claim amounts are being underestimated how much it actually would be. This could be an issue for the insurance companies as the reserve set aside may not be sufficient to meet the claim outgo. In the extreme scenario, the insurers might need to fire sell the assets to meet the claim payout.

Alternatively, one could use the **corrr** package, ie. the correlation package under **tidymodels** framework to visualize the correlation matrix in graph format. This package is not used as the visualization under the **corrplot** package is better than **corrr** package.

7.0   Overview of Machine Learning

In general, machine learning models can be classified into four main broad categories (i.e. classical learning, reinforcement learning, ensemble methods, and neural nets & deep learning) as shown under Figure 56. Some examples of the models are stated beside the different categories.
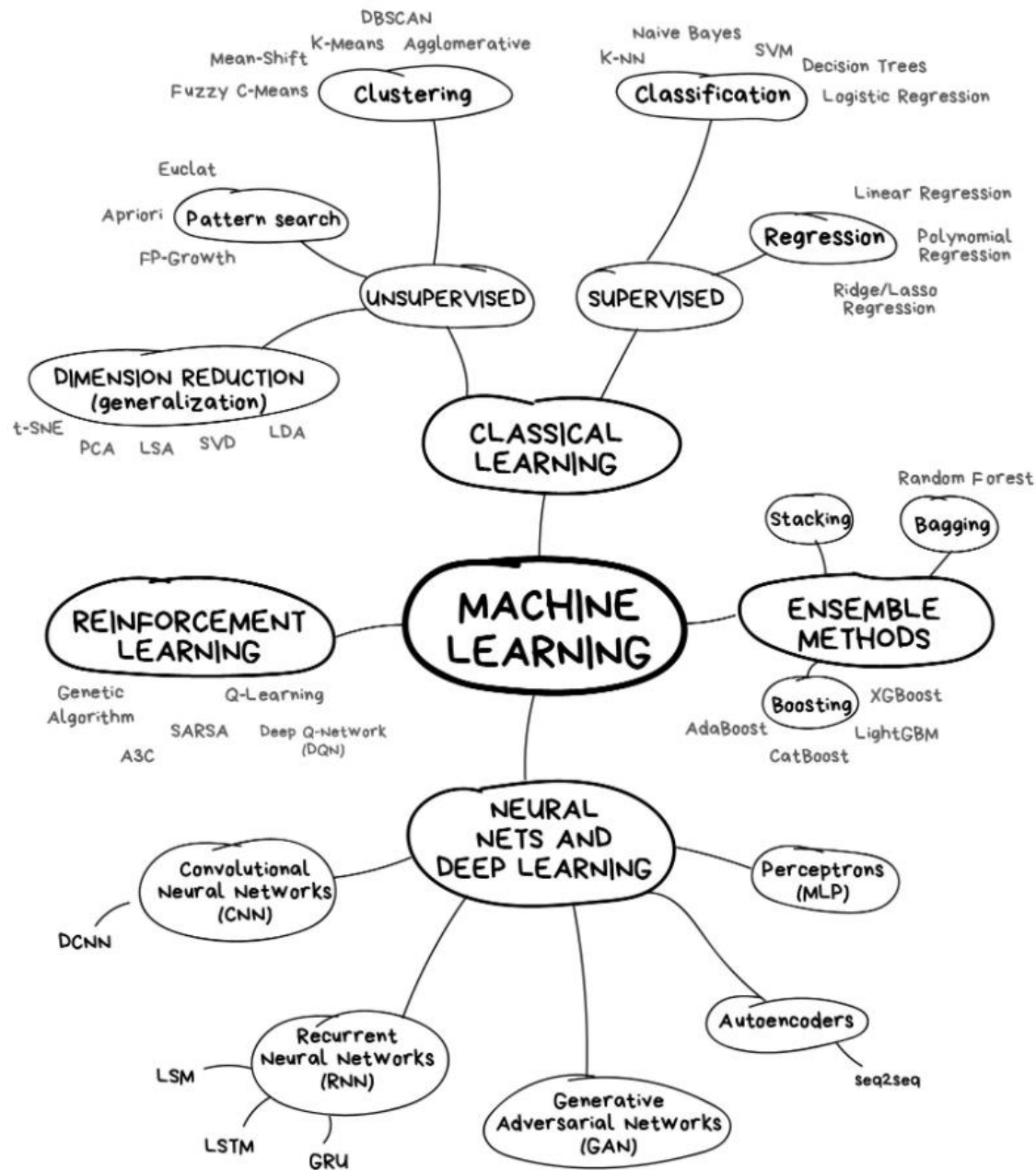
Figure 56: Different types of machine learning models

Classical Learning (a.k.a. classical machine learning)

This is one of the most common machine learning techniques we encounter in our daily lives. For example, companies might use regression models to estimate the profits the customers would bring in a given period. One could also use classification models to determine whether the tumor is malignant.

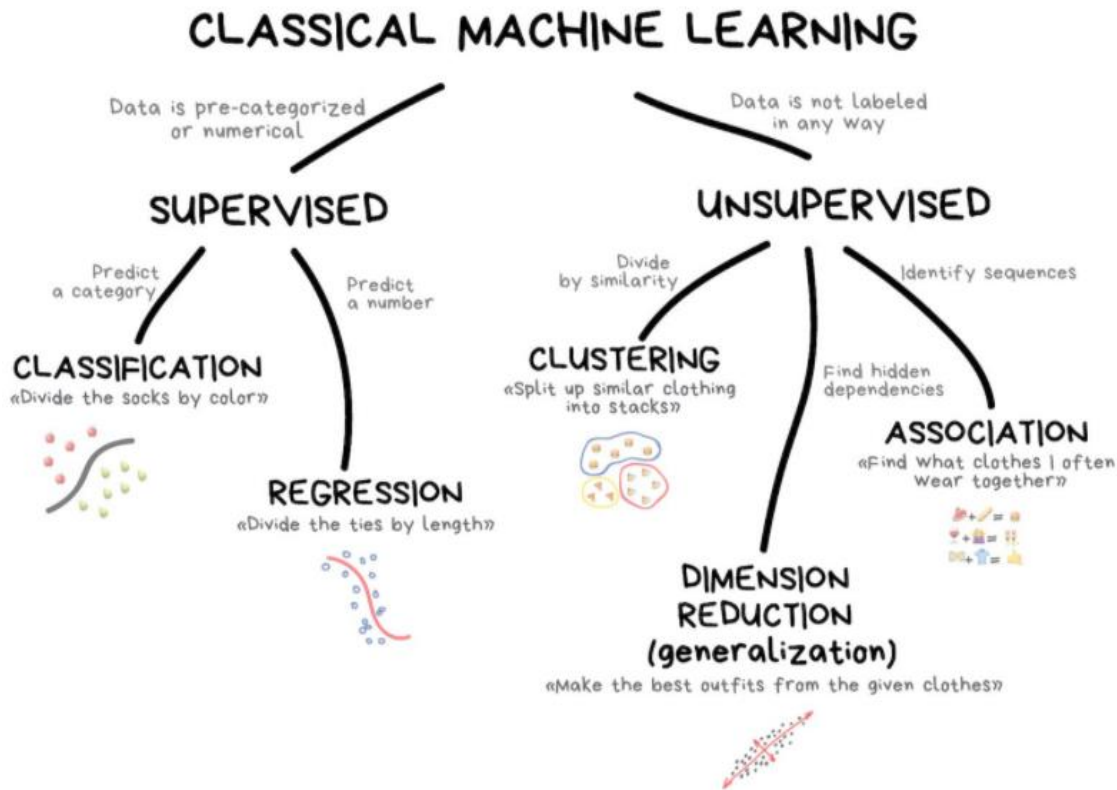Following are an overview of the different classical learning methods:

Figure 57: Classical Machine Learning

The classical machine learning models can be further split into two broad categories, which are supervised and unsupervised learning methods. As the name suggested, the supervised learning method has a clear target variable (a.k.a. response variable or dependent variable) for one to train the models, where the unsupervised learning method allows the algorithm to uncover hidden relationships within the dataset.

Unsupervised machine learning

Unsupervised learning method	Details
Clustering	Clustering is one of the basic algorithms any beginners would learn when they started their data science journey. The clustering algorithms include k-means, hierarchical clustering, latent class analysis, and so on. It is commonly used in many applications as shown following: - Find the different customer segments so that one could derive a more targeted strategy to boost the sales - Group the customers into different groups based on their behaviors
Association	Apriori is one of the common algorithms used to mine the association rules between the different items. In other words, this method allows one to discover how the items are "associated" with one another. Following are some of the common examples of how the association rules are used: - Product bundling by bundling products with higher

	profit margin and closely associated to boost the sales - Perform cross-selling and upselling products - Arranging the closely associated products together to increase the sales The more advanced technique includes taking time into considerations while performing association algorithms. This would allow one to understand whether there is a change in customer purchasing behaviors.
Dimension reduction	As the data increases, often one would face the "curve of dimensionality". This is an issue while building machine learning models as the data points in high-dimensional space are sparse, resulting in a less desirable machine learning model. Therefore, the common way to resolve this is to perform principal component analysis (PCA). The key idea of PCA is the algorithm attempts to find a low-dimensional representation of a dataset that contains as much as possible of the variation (James, Witten, Hastie, & Tibshirani, 2013). However, as the principal components derived from the algorithm are not directly observable from the dataset, hence it is often more challenging to explain or understand the results.

Figure 58: Unsupervised Learning Method

Supervised machine learning

Supervised learning method	Details
Classification models	The classification model can be used to predict categories of the data. Following are some of the common use cases for classification models: - Predict whether the customers would churn - Predict whether the new tumor found is malignant - Predict whether this new claim is a fraud or a genuine claim
Regression models	The regression model can use estimate the amount of the interested outcome variables, eg. house prices, insurance claim amount, and so on. Following are some of the common use cases for regression models: - Estimate the claim amount of new insurance policies - Estimate the amount the customers will spend in the upcoming marketing campaigns

Figure 59: Supervised Learning Method

Aside from that, there is another also type of machine learning method that falls in between supervised and unsupervised learning methods, which is not covered under Figure 57. This method is known as the "semi-supervised learning method". Unlike the supervised or unsupervised learning method, this method allows us to use both labeled and unlabeled data to fit the model. This can come quite handy especially sometimes the data might contain a missing target variable. One example of such a method is network classification. Nevertheless, the semi-supervised learning method is beyond the scope of this capstone project.

Reinforcement Learning

This machine learning uses rewards and penalties to train the machines. The goal is to find a suitable action model that would maximize the total cumulative reward of the agent (Bhatt, n.d.).

Ensemble Methods

The various ensemble methods (eg. bagging, boosting, and so on) have gained popularity in Kaggle competitions in recent years due to high accuracy in model predictions. One of the most common ensemble methods is random forest, which is applying bagging techniques on decision trees.

Neural nets and deep learning

The way how this type of model works is similar to the human brain. The model is essentially made up of a bunch of neurons and connections. One common application for this type of model is image recognition. For example, one might train a neural net model to recognize a cat by passing through a bunch of cat photos into the models. Besides that, this technique could be used to perform the conventional actuarial analysis as well. Swiss Association of Actuaries published a research paper on how one could nest an actuarial model into a neural network (Schelldorfer and Wuthrich 2019).

For this capstone, various classical machine learning models and ensemble methods will be used to show how they can use in claim cost prediction.

8.0   Modeling Building

In the Tidy Modeling with R (Kuhn and Silge 2021), the authors mentioned that the typical data analysis process requires multiple iterations. Often, after building the models, the users might go back to previous steps to further clean or split data to build the necessary machine learning models. Nevertheless, the process typically starts with data pre-processing, model fitting, model tuning, and model evaluation as shown under Figure 60.
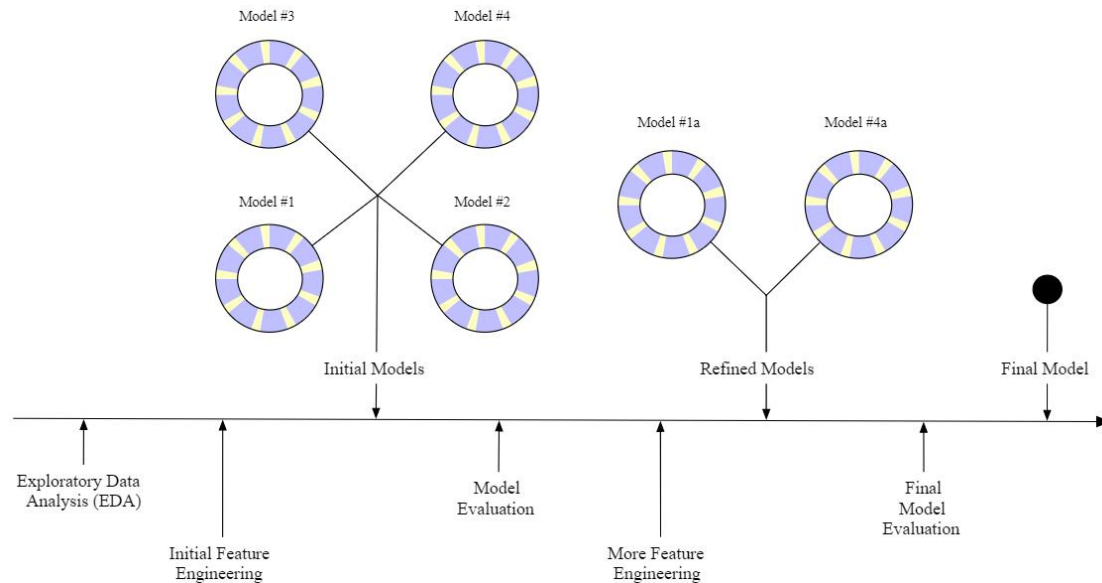
Figure 60: Modeling Process

Note that the modeling process is rather iterative. After building the model, the users might go back to previous steps to further tune the dataset (eg. perform more data pre-processing or/and feature engineering) or model parameters, seeking to improve model performance.

8.1 Data Preprocessing

As mentioned above, often the algorithm requires the data in certain formats/types before the program could fit the models. For example, statistical learning models are unable to handle missing data, the XGBoost algorithm is unable to handle categorical data, and so on.

Besides, there are circumstances the data should not be used directly due to various reasons, such as invalid values within the dataset, there are excessive categories in the dataset, and so on. For example, in the earlier section, we have shown that there are invalid values within the dataset (eg. number of hours worked per week). Without 'fixing' such issues in the data before fitting the machine learning models, the performance of the models could be affected.

Various functions from the **recipe** package will be used. This R package helps the users to create the blueprints of the different required data pre-processing and chain them together as a model workflow from the **workflows** package.

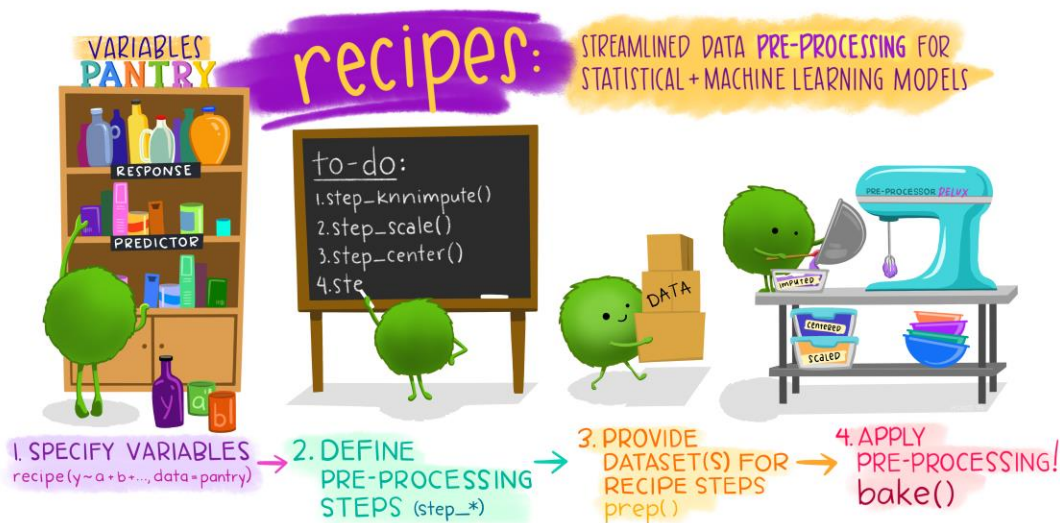Below is an illustration of how the **recipe** packages work (Horst, n.d.):

Figure 61: Recipe Package

Next, I will demonstrate how the **recipe** package and **workflows** package could work together in the code chunks below:

First, cleaned data is imported into the environment and the data is being split into training and testing datasets by using **initial_split** function. Note that I have specified the proportion between the training and testing dataset to be 60/40 in the **prop** argument. Then, the **training** and **testing** function is used to carve out training and testing datasets.

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

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

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

Figure 62: Import & Split the Dataset

Next, the formula and necessary data pre-processing to be run are defined by using various functions from the **recipe** package.

```
gen_recipe <- recipe(init_ult_diff ~ ., data = df_train) %>%
  step_date(c(DateTimeOfAccident, DateReported)) %>%
  step_mutate(DateTimeOfAccident_hr = lubridate::hour(DateTimeOfAccident),
              DateTimeOfAccident_hr = factor(DateTimeOfAccident_hr, order = T
RUE)) %>%
  update_role(c(DateTimeOfAccident, DateReported, ClaimNumber), new_role = "i
```

```
d") %>%
  prep()
```

Figure 63: Defining the recipe and prepare the recipe

The first line indicates what is the "recipe" for machine learning, which comprises the formula and the name of the dataset. Next, I have indicated that I would like to extract the time information from the date-time variable through **step_date** function. As I did not indicate the required features to be extracted in the function, the default time date features will be extracted (ie. day of the week, month, and year).

As the **step_date** function currently can only extract certain date-time features as mentioned above, hence I have further used **step_mutate** function to extract the hour from the date-time variable. This function has effectively allowed one to perform similar **mutate** function under **dplyr** package. I have also converted the extracted hour into an ordinal variable instead.

As not all the variables should be used to train and fit the model, hence the roles for some of these variables have been updated to 'id' through the **update_function** function. This is to ensure the model does not use these variables to train the model.

Lastly, I have used the **prep** function to train a data recipe. Note that this step is not necessarily required. It is required when we would like to see the list of variables, especially after we have derived new variables based on existing variables or we intend to apply the same recipe on other datasets.

Next, Figure 64 shows the summary information of the recipe created under Figure 63.

```
gen_recipe

## Data Recipe
##
## Inputs:
##
##       role #variables
##         id          3
##    outcome          1
##  predictor         29
##
## Training data contained 27894 data points and no missing data.
##
## Operations:
##
## Date features from DateTimeOfAccident, DateReported [trained]
## Variable mutation for DateTimeOfAccident_hr, DateTimeOfAccident_hr [traine
d]
```

Figure 64: Summary of the created recipe

We could call the defined recipe to check the high-level details (eg. the list of data pre-processing will be performed) before proceeding to build models.

After defining the recipe, we could also use the **summary** function within the **recipe** package to summarize the information of the variables. Note that the derived variables would not appear in the list below if the **prep** function is not run on the created recipe.

This flexibility allows the users to check through the data type of each variable to ensure they are in the right data type before fitting machine learning.

```
gen_recipe %>%
  summary()

## # A tibble: 40 x 4
##    variable          type    role     source
##    <chr>             <chr>   <chr>    <chr>
##  1 ClaimNumber       nominal id       original
##  2 DateTimeOfAccident date    id       original
##  3 DateReported       date    id       original
##  4 Age               numeric predictor original
##  5 Gender            nominal predictor original
##  6 MaritalStatus     nominal predictor original
##  7 DependentChildren numeric predictor original
##  8 DependentsOther   numeric predictor original
##  9 WeeklyWages       numeric predictor original
## 10 PartTimeFullTime  nominal predictor original
## # ... with 30 more rows
```

Figure 65: Summary of the variables in the created recipe

Alternatively, the users can still use the various functions under **tidyverse** to perform the necessary pre-processing before fitting the model. Nevertheless, this **recipe** R package provides the users another option to perform data pre-processing before building machine learning.

Note that **tidymodels** allow us to modularize the different model components, allowing the users to reuse some of the model components throughout the analysis. For example, unlike random forest, GLM is unable to handle categorical variables due to its algorithm. As such, one of the common methods to handle such a situation is to perform one-hot encoding on the categorical variables.

Instead of recreating another recipe variable, we could reuse the recipe created for random forest earlier and add additional steps to indicate one-hot encoding will be performed on all categorical variables as shown below.

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

Figure 66: Updated Recipe Object

8.1.1 Feature Engineering

Apart from that, feature engineering is one of the steps under data pre-processing. It is a process where new features are created from the existing ones or new variables are being

"derived" from the list of the current dataset (Sethi 2020). The goal of this activity is to reformat the predictor values to make them easier for a model to use effectively (Kuhn and Silge 2021). Data scientists use various techniques, such as binning, imputation, log transformation, and so on to extract the "gold" from the very noisy raw data. This is crucial as often a good feature engineering would help to improve the accuracy of the models.

The claim description is a classic example of a variable that cannot be used directly for modeling purposes. There is an excessive number of unique categories within claim descriptions. Using the variable directly in building the model would create more noise to the model, resulting in lower model accuracy. Neither we should just discard the variables before trying to incorporate this variable while building the model. One of the ways to incorporate this variable in model building is to extract the keywords from the claim description. In this analysis, two different pre-processing text data methods are used and separate models are built to compare their results.

Method 1 - Use **tidytext** *to perform text mining*

The claim description fields are first tokenized by using the **unnest_tokens** function. The argument n is set as 1, so the n-gram will be used. To remove the stopwords from the list, I have performed an anti join (ie. the function will drop the word if it matches with any of the stopwords) with the stopwords.

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

Figure 67: Tokenize the Relevant Words and Remove all the Relevant Stopwords

Then, I count the number of words and sort the list accordingly.

```
cleaned_clm_unigram <- tidy_clm_unigram %>%
  count(word, sort = TRUE) %>%
  mutate(cum_count = cumsum(n),
         cum_perc = cum_count/sum(n))
```

Figure 68: Count the Tokens and Sort Them Descending

One of the common visualization methods for text is to use a word cloud. First, I have filtered out those words with a relatively lower count by using the **filter** function. Next, I will pass the dataset to the **ggplot** function and I specify the graph type by calling the **geom_text_wordcloud** function.

```
cleaned_clm_unigram %>%
  filter(n > 1000) %>%
  ggplot(aes(label = word, size = n, color = n)) +
  geom_text_wordcloud() +
  scale_size_area(max_size = 15) +
  theme_minimal()
```

Figure 69: Word Cloud

By visualizing with the word cloud, it is quite clear the words tend to appear more frequently in the claim descriptions than the rest. The bigger the word size in the graph, the more frequently the word appears. It is quite clear that some of the words appear more frequently than others and they might be explained the difference in the claim amount. For example, it could be the case where the length of the recovery period of a certain injury is relatively more uncertain due to the potential complications in treatments.

Therefore, I will convert the keywords into indicators to improve the model performance. Below are the 5 variables created in this process:

- injury_body: Body part(s) that were injured (eg. leg, hand, head)

- injury_item: Item(s) that caused the injury (eg. knife, metal, ladders)

- injury_cause: Cause(s) of the injury (eg. struck, lifting, slip, accident)

- injury_type: Type(s) of injury (eg. strain, sprain, bruise)

- injury_side: Side of the body parts that were injured (eg. left, right, middle)

Figure 70 is an example of how the indicators are extracted as an indicator. As I would like to extract the different body parts from the text fields, so I have used for loop to complete this task. The list of body parts to be extracted from claim descriptions is first defined in a list. Then, I will loop through the extraction through the created list. Within the loop function, I will first define the new column name, then call this new column in the **mutate** function.

Note that the **sym** function will be wrapped around the new column name to indicate this new column name is a symbol, instead of merely text. Lastly, the **case_when** function is used to set the indicator to be 1 if the word matches the current word is being looped within the function.

```r
body_part_list <- c("back", "finger", "hand", "shoulder", "eye", "knee", "wrist", "thumb", "neck", "ankle", "arm", "foot", "leg", "forearm", "elbow", "head")

for (i in body_part_list){
  new_col <- paste0("body_", i)

  tidy_clm_unigram <- tidy_clm_unigram %>%
    mutate(!!sym(new_col) := case_when(word == i ~ 1,
                                       TRUE ~ 0))
}
```

Figure 70: Create indicator from Claim Description Field

Once the indicators are created, the dataset will be group by their claim number. This is because when the words are being tokenized, each data record is being duplicated multiple times, depending on the length of the claim descriptions. When the data is being grouped, I have set the new body part indicator to be one if the body part is being mentioned in the claim descriptions. Lastly, I have joined the indicator back to the original dataset for later modeling purposes.

```r
for (i in body_part_list){
  new_col <- paste0("body_", i)

  temp <- tidy_clm_unigram %>%
    group_by(ClaimNumber) %>%
    summarise(!!sym(new_col) := case_when(sum(get(paste0("body_", i))) > 0 ~
1,
                                          TRUE ~ 0))
  data_2 <- data_2 %>%
    left_join(temp, by = c("ClaimNumber"))

  rm(temp)
}
```

Figure 71: Consolidate the Indicators and Join Back to Original Dataset

Instead of modeling the injury body as dummy variables, I will consolidate the different injury body part indicators into one single column. First, I perform a row sum over all the indicators that start with "body_" in the variable name. Then, the **gather** function is used to pivot the columns into a single column. The **filter** function is then used to filter the claim that mentioned only one body part so that the relevant body part that is injured can be tagged accordingly. After that, I will drop the flag variable and replace the 'body_' with an empty string in 'injury_body' by using the **str_replace** function.

```
data_3_singleInjury <- data_2 %>%
  mutate(sum_injury = rowSums(across(contains("body_")))) %>%
  gather(injury_body, flag, body_back:body_head) %>%
  filter(flag == 1 & sum_injury == 1) %>%
  dplyr::select(-flag) %>%
  mutate(injury_body = str_replace(injury_body, "body_", ""))
```

Figure 72: Extract Single Body Injury

Next, I have performed similar steps as above to indicate those claims with multiple injured body parts and no body part was injured.

```
# the code below gathers the remaining types
data_3_multipleInjury <- data_2 %>%
  mutate(sum_injury = rowSums(across(contains("body_")))) %>%
  filter(sum_injury != 1) %>%
  mutate(injury_body = case_when(sum_injury > 1 ~ "multiple",
                                 TRUE ~ "others")) %>%
  dplyr::select(-contains("body_"))
```

Figure 73: Extract Multiple Body Injury

Eventually, I will bind the two datasets together for later modeling purposes as shown below. After the different injury indicators are being extracted, the dataset will then be used to fit a model. The comparison of model performance can be found below.

```
# bind the rows
data_3 <- bind_rows(data_3_singleInjury, data_3_multipleInjury)
```

Figure 74: Combined Dataset once Indicators are Extracted

*Method 2 - Use **tidyrecipe** to perform text mining*

Alternatively, **tidyrecipe** package can be used to perform text mining. With just a few simple lines of codes, the functions could extract the necessary information as features for the model building purpose later. This method is more consistent with how the **recipe** is being **prepared** when we build machine learning models.

First, the claim descriptions are being tokenized by using **step_tokenize** function. Next, the stopwords are removed by using **step_stopwords** function. Then, only the top 20 words (based on word count) are selected in building the model.

Lastly, **step_tdidf** function is used to convert the words into term frequency-inverse document frequency (tf-idf) of the token. The inverse document frequency function within td-idf would penalize how frequently the words appear in the claim description fields. The idea is that the more common the words appear (eg. stopwords), the less important the words are.

```
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)
```

Figure 75: Tidytext Recipe

8.1.2 Other data pre-processing considerations for insurance claim data

Below are other considerations worthwhile to take note of when handling insurance data:

- Claim data usually suffer from an imbalanced data issue. In other words, the proportion of the categories (eg. indicator whether this policy has made a claim before) is highly imbalanced. This is a common issue in the insurance claim data as the claim made is only coming from a handful of policies within all the policies the company has written. To resolve this, one could use various resampling techniques under **themis** package to fix the imbalanced categories within the data.

- Alternatively, one could leverage on **probably** package to find the optimal cutoff for the classification models, instead of using the default cutoff (ie. 0.5). As the data used in this project is the claim data, hence it does not suffer from imbalanced data issues. Hence such data pre-processing steps would not be performed. Nevertheless, refer to recipe package documentation page for more data pre-processing that are currently available under the package.

- Claim data is highly skewed, where this usually does not work well with statistical learning models. Hence, data transformation is often performed to convert the distribution into a normal distribution. Alternatively, hierarchical modeling can be another possible alternative to further improve the modeling results.

8.2 Model Selection

Model selection is also one of the key modeling steps. One would need to determine whether a classification or regression model should be used based on the data science problem. This is because some machine learning models can solve one type of problem (ie. either classification or regression problem). For example, logistic regression can only solve classification problems, making this model inappropriate when we want to solve a regression problem.

Besides, different types of machine learning models might be built and compared to see which machine learning models have a superior result. (Burns et al. 2019) provides a list of considerations while selecting the models to complement actuarial analysis:

- *Computation Resource*: While some models could provide higher accuracy in the prediction, they might be more computationally intensive (eg. deep learning).

- *Interpretability*: How easy is it for the users to interpret the results? Do the users know why the models make certain recommendations?

- *Time*: Do we have time to perform the necessary research on the algorithm? If not, (Burns et al. 2019) recommended using the current standards.

- *Implementability*: The goal of using analytics is to complement the conventional actuarial analysis. However, some of the machine learning models may not be easier in incorporating into

As the focus of this capstone project is to demonstrate how one could build an end-to-end machine learning process by using these modern data science packages, hence the discussions on the considerations above would be out of the scope of this capstone project. Nonetheless, these considerations provide the users a more objective approach in selecting the algorithms.

One of the common issues with the machine learning functions from the open-source software is heterogeneity in the arguments. This is often because the codes were focused on the needs of the person developing the code, instead of the needs of the person using the code (Kuhn and Silge 2021). This can be a stumbling roadblock for users in their data science projects. In Figure 76, it shows the different **predict** function under different machine learning package.

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")

Figure 76: Different machine learning functions

Also, another common issue is the output from the different machine learning models is in different formats. This would require the users to perform data transformation before passing the output to another machine learning function, which slows down the analysis process.

As mentioned in Figure 17, instead of reinventing the "wheels", the RStudio team has developed a wrapper package to wrap over the existing R package and provide the users a more consistent model interface. These functions also conform to tidy data concepts, which allows the users to pass the output to other functions without requiring the users to transform the data. As such, the **parsnip** package will be used to select the necessary

machine learning model. This package provides users a more unified interface to different machine learning models, providing a more seamless experience in performing data science analysis.

```r
ranger_spec <-
  rand_forest() %>%
  set_mode("regression") %>%
  set_engine("ranger")
```

Figure 77: Model Specification for Random Forest

As shown above, the standard model specs under **tidymodels** package starts with selecting what type of models we are building (eg. random forest, linear regression etc). We will indicate the type of data science problem to be solved (ie. regression or classification problem). As the target variable is a continuous variable, regression is being indicated under **set_mode** function. Lastly, the R package to be wrapped is indicated under the **set_engine** function.

With this standardized structure, it allows the users to focus on other modeling matters, such as understanding how the models work, parameters to be tuned, and so on without need to worry how to write the codes for the various machine learning models.

To check the model specification I have just specified earlier, the **translate** function can be used to call out the model specification. Note that there is a conflict in the **translate** function between different packages. Hence to ensure the right function is being used for the analysis in the code chunk below, I have specified which package the code should refer to when using the **translate** function as shown below.

```r
ranger_spec %>%
  parsnip::translate()

## Random Forest Model Specification (regression)
##
## Computational engine: ranger
##
## Model fit template:
## ranger::ranger(x = missing_arg(), y = missing_arg(), case.weights = missin
g_arg(),
##      num.threads = 1, verbose = FALSE, seed = sample.int(10^5,
##         1))
```

Figure 78: Translating Model Specification

While the model interfaces are unified, it still provides the users to specify additional arguments within the functions. Figure 79 is the demonstration of how one could fit a GLM model with a different family. By default, the family is set as gaussian if the family is not specified. Therefore, one can indicate the family by calling the **family** argument in the **set_engine** function.

```
glm_spec <-
  linear_reg()) %>%
  set_mode("regression") %>%
  set_engine("glmnet", family = "binomial")
```

Figure 79: Example of how to pass additional arguments to set_engine function

Following are some of the models supported by **tidymodels** at the point of analysis:

Figure 80: Different Parsnip Models

Refer to the 'Search Parsnip Models' page on the various models that are currently supported under **tidymodels** framework.

8.2.1 Linear regression

Linear regression is one of the most basic machine learning models anyone would learn when they embark on their data science journey. The target variable is assumed to be linear related to the independent variables. It can be expressed in the following form:

$$y_i = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \ldots + \beta_n * x_n + \epsilon_i$$

where y_i is the response variable (also known as the dependent variable or target variable), β_0 is the intercept term estimated from the model, β_i is the estimated coefficient for x_i and ϵ_i is the random error that is unexplained by the model.

The objective of ordinary least square linear regression is to find the plane that minimizes the sum of squared errors (SSE) between the observed and response variables (Kuhn and Johnson 2013).

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where y_i is the actual value of variable i and $\hat{y}_i$ is the fitted value of variable i.

One can fit a linear regression by setting the mode to be "lm" in **linear_reg** function to fit a linear regression model under tidymodels as shown in Figure 81.

```
linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression") %>%
  parsnip::translate()

## Linear Regression Model Specification (regression)
##
## Computational engine: lm
##
## Model fit template:
## stats::lm(formula = missing_arg(), data = missing_arg(), weights = missing
_arg())
```

Figure 81: Linear Regression Example is taken from documentation page of Parsnip package

8.2.2 Generalised linear regression (GLM)

One of the key criticisms of linear regression is the model assumes the underlying relationship between independent and response variables. This assumption is unlikely going to be appropriate for most of the insurance context. Often, the response variable the insurers are measuring (eg. claim count, claim size) is unlikely to follow a normal distribution.

Another key assumption under linear regression is constant variance. Under this assumption, the residual (also known as the error term) does not vary with the independent variables. Linear regression may not be appropriate if the plot of residual versus predicted values exhibits some patterns, instead of scattering around zero.

GLMs, one of the more popular models used by actuaries, relax the normality and constant variance assumptions. It was originally introduced in actuarial analysis to improve the accuracy of motor insurance pricing (Michel, Donatien, and Julien 2019). This model can gain popularity to use in other insurance areas such as claim cost estimation, elasticity modeling, competitive analysis, and so on. (Alfredo, Dutang, and Petrini 2018) published a paper on

how one could use GLM to perform pricing optimization and how the GLM model performed compared to other machine learning models.

GLM consists of the following components(Michel, Donatien, and Julien 2019):

- Response distribution (random component) selected from the exponential dispersion family

- Linear score (systematic component)

- Link function relating the mean response to the score involving the available features

Below are some of the common link functions under GLM(Michel, Donatien, and Julien 2019):

Link function	$s_i = g(\mu_i)$	$\mu_i = g^{-1}(s_i)$
Identity	μ_i	s_i
Log	$\ln \mu_i$	$\exp(s_i)$
Inverse	μ_i^{-1}	s_i^{-1}
Inverse-square	μ_i^{-2}	$s_i^{-1/2}$
Square-root	$\sqrt{\mu_i}$	s_i^2
Logit	$\ln \frac{\mu_i}{1-\mu_i}$	$\frac{\exp(s_i)}{1+\exp(s_i)}$
Probit	$\Phi^{-1}(\mu_i)$	$\Phi(s_i)$
Log-log	$-\ln(-\ln \mu_i)$	$\exp(-\exp(s_i))$
Complementary log-log	$-\ln(-\ln(1-\mu_i))$	$1 - \exp(-\exp(s_i))$

Figure 82: Link function (extraction from Effective Statistical Learning Methods for Actuaries I)

While GLM is one of the popular machine learning model choices among the actuarial community, GLM does suffer a few key issues. In the Machine Learning Methods for Insurance Application - A Survey by SOA (Diana et al. 2019), the authors summarized some of the criticisms of using the GLM model:

- Either zero or full credibility is given to the data, and there is no way to do blending

- Prediction of risk depends on data in other, potentially different segments

- Model predictions depend on the mixture of rating factors in the data

- Maximum-likelihood estimate of prediction is lower than the mean of the prediction distribution

- Link function could bias the model prediction and significantly change the lower and upper bound of the prediction

- Model diagnostics are relevant only in the segments where the model is used

Nevertheless, one could also fit a GLM model under the **parsnip** package. There are several packages supported by the **linear_reg** function that allow users in fitting a generalized linear model.

```
linear_reg() %>%
  set_engine("glmnet", family = "poisson") %>%
  set_mode("regression") %>%
  parsnip::translate()

## Linear Regression Model Specification (regression)
##
## Engine-Specific Arguments:
##    family = poisson
##
## Computational engine: glmnet
##
## Model fit template:
## glmnet::glmnet(x = missing_arg(), y = missing_arg(), weights = missing_arg(),
##     family = "poisson")
```

Figure 83: Generalised Linear Regression Example with Poisson Distribution as the family

8.2.3 Multivariate Adaptive Regression Splines (MARS)

In a survey published by the Society of Actuaries (Diana et al. 2019), the authors explained that MARS is a nonparametric regression technique that splits the regression line into shorter segments:

$$f(x) = \beta_0 + \sum_{i=1}^{m} \beta_i \, h(x_i)$$

where $h(x_i)$ (also known as the basis function), is the form $\max(x_i - c, 0)$, $\max(c - x_i, 0)$ or a product of these expressions. The authors pointed out that MARS is useful in multivariate context since it explores the interaction effects of the variables.

Currently, only the **earth** package is supported by the **mars** function in the **parsnip** package. Following are the common parameters in **mars** function (Kuhn and Vaughan, n.d.d):

- Number of features (num_terms): The number of features that will be retained in the final model.

- Number of interaction degrees between features (prod_degree): The highest possible degree of interaction between features. A value of 1 indicates an additive model while a value of 2 allows, but does not guarantee, two-way interactions between features.

- Method of pruning (prune_method): The type of pruning

```
mars() %>%
  set_mode("regression") %>%
```

```
  set_engine("earth") %>%
  parsnip::translate()

## MARS Model Specification (regression)
##
## Computational engine: earth
##
## Model fit template:
## earth::earth(formula = missing_arg(), data = missing_arg(), weights = miss
ing_arg(),
##     keepxy = TRUE)
```

Figure 84: MARS Example

8.2.4 Decision Tree

Decision tree is one of frequent choice of machine learning model as it resembles how human makes a decision.

The key idea of a decision tree is the algorithm will divide the predictor space to minimize the residual sum of square (RSS) as shown below (James et al. 2013):

$$RSS = e_1^2 + e_2^2 + \ldots + e_J^2 = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \widehat{y_{R_j}})^2$$

Where e_j is the residual of variable j (i.e. $e_j = y_j - \hat{y}_j$), y_i is the actual value of variable i and $\hat{y}_{R_j}$ is the mean response for the training dataset within jth box.

Following are the common parameters under the decision tree (Kuhn and Vaughan, n.d.b):

- Parameter for cost of complexity (cost_complexity): The cost/complexity parameter (a.k.a. Cp) used by CART models (rpart only)

- Depth of the tree (tree_depth): The maximum depth of a tree (rpart and spark only)

- Minimum number of nodes (min_n): The minimum number of data points in a node that is required for the node to be split further

```
decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("regression") %>%
  parsnip::translate()

## Decision Tree Model Specification (regression)
##
## Computational engine: rpart
##
## Model fit template:
## rpart::rpart(formula = missing_arg(), data = missing_arg(), weights = miss
ing_arg())
```

Figure 85: Decision Tree Example

8.2.5 Random Forest

The algorithm of the decision tree is based on the top-down greedy approach in splitting this model, hence the algorithm will keep on finding the best split at the particular step. Therefore, this model typically suffers from overfitting, ie. low bias but high variance. So to overcome this issue, the algorithm fits a collection of decision trees and the predicted value is just the average value from all the decision trees. This is how random forest builds the models. Often, the random forest model would give us a better model performance, compared to the decision tree. Nevertheless, it is still important to check whether there are any signs of overfitting after fitting the model.

Following are the common parameters for random forest (Kuhn and Vaughan, n.d.e):

- Number of predictors (mtry): The number of predictors that will be randomly sampled at each split when creating the tree models

- Number of trees (trees): The number of trees contained in the ensemble

- Minimum number of nodes (min_n): The minimum number of data points in a node that is required for the node to be split further.

Currently, **tidymodels** supports several random forest packages in R, including **ranger**, **randomForest**, and **spark** as this paper was written.

```
rand_forest() %>%
  set_engine("ranger") %>%
  set_mode("regression") %>%
  parsnip::translate()

## Random Forest Model Specification (regression)
##
## Computational engine: ranger
##
## Model fit template:
## ranger::ranger(x = missing_arg(), y = missing_arg(), case.weights = missin
g_arg(),
##     num.threads = 1, verbose = FALSE, seed = sample.int(10^5,
##         1))
```

Figure 86: Random Forest Example

8.2.6 XGBoost

XGBoost, also known as eXtreme Gradient Boosting, is one of the more popular machine learning models within the data science community due to its superior performance. This model leverages "boosting" (i.e. one of the ensemble learning methods) to improve its performance.

In boosting, the trees are being built sequentially such that each subsequent tree aims to reduce the errors of the previous tree (Sundaram 2018). In other words, the idea of boosting is that a strong learner can be produced by effectively combining the weak learners.

parsnip package supports a few different R XGBoost packages. Following are the parameters for the XGBoost model (Kuhn and Vaughan, n.d.a):

- Number of predictors ("mtry"): A number for the number (or proportion) of predictors that will be randomly sampled at each split when creating the tree models (xgboost only).

- Number of trees ("trees"): An integer for the number of trees contained in the ensemble.

- Minimum number of data point ("min_n"): An integer for the minimum number of data points in a node that is required for the node to be split further.

- Number of splits ("tree_depth"): An integer for the maximum depth of the tree (i.e. number of splits) (xgboost only).

- Learning rate ("learn_rate"): A number for the rate at which the boosting algorithm adapts from iteration-to-iteration (xgboost only). A lower learning rate will take a longer time to reach the convergence, but the algorithm might be able to find a better set of parameters that would give a better performing model.

- Parameter for loss function ("loss_reduction"): A number for the reduction in the loss function required to split further (xgboost only).

- Sample size ("sample_size"): A number for the number (or proportion) of data that is exposed to the fitting routine. For xgboost, the sampling is done at each iteration while C5.0 samples once during training.

- Number of iterations ("stop_iter"): The number of iterations without improvement before stopping (xgboost only).

```
boost_tree() %>%
  set_mode("regression") %>%
  set_engine("xgboost") %>%
  parsnip::translate()

## Boosted Tree Model Specification (regression)
##
## Computational engine: xgboost
##
## Model fit template:
## parsnip::xgb_train(x = missing_arg(), y = missing_arg(), nthread = 1,
##     verbose = 0)
```

Figure 87: XGBoost Example

8.2.7 K nearest neighbor (KNN)

As the name suggested, K nearest neighbor predicts the value of the particular response variable by referencing the K nearest neighbors. This model can be powerful if there is some relationships between local predictor structure and response variables. However, as the model uses the nearest neighbors to make the necessary predictions, hence it is important to remove or exclude any irrelevant variables to remove the potential noises created by these variables. In other words, putting more variables into the model does not guarantee a better K nearest neighbor model.

kknn package is the only K nearest neighbor R package supported by parsnip package as this analysis is conducted. Following are the main arguments for this model extracted from **parsnip** package website (Kuhn and Vaughan, n.d.c):

- Number of neighbors ("neighbors"): A single integer for the number of neighbors to consider (often called k). For kknn, a value of 5 is used if neighbors are not specified.

- Type of kernel function ("weight_func"): A single character for the type of kernel function used to weight distances between samples. Valid choices are: "rectangular", "triangular", "epanechnikov", "biweight", "triweight", "cos", "inv", "gaussian", "rank", or "optimal".

- Paramater of Minkowski distance ("dist_power"): A single number for the parameter used in calculating Minkowski distance.

```
nearest_neighbor() %>%
  set_engine("kknn") %>%
  set_mode("regression")

## K-Nearest Neighbor Model Specification (regression)
##
## Computational engine: kknn
```

Figure 88: K Nearest Neighbour Example

8.2.8 UseModels R package

Alternatively, the RStudio team has developed another R package, i.e. **usemodels** package that could generate the model template (including recommended data pre-processing steps, model specification, and so on).

Below are various machine models supported under the **usemodels** package:

R functions	Remarks
use_glmnet	Model template for Generalised Linear Model
use_ranger	Model template for Random Forest
use_xgboost	Model template for XGBoost
use_earth	Model template for Multivariate Adaptive Regression Splines

use_cubist   Model template for Cubist Rule-based Regression Model

use_kknn   Model template for K Nearest Neighbours

Figure 89: Model Template from usemodels package

To use such functions, the users would just need to pass the formula & data as the arguments. The package will generate the necessary model template as shown below.

```
use_ranger(init_ult_diff ~ .,
           data = df_train)

## ranger_recipe <-
##   recipe(formula = init_ult_diff ~ ., data = df_train) %>%
##   step_string2factor(one_of(ClaimNumber, Gender, MaritalStatus, PartTimeFu
llTime,
##     ClaimDescription, 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(75826)
## ranger_tune <-
##   tune_grid(ranger_workflow, resamples = stop("add your rsample object"),
grid = stop("add number of candidate points"))
```

Figure 90: Code chunk for usemodels

The templates will include some of the recommended data pre-processing steps, depending on the variable types in the data. Users could also adjust the details of the model template wherever it is deemed necessary. This has effectively helped the users, especially new users to perform necessary machine learning analysis.

8.3   Model Fitting, Tuning, and Evaluation

One of the key considerations while building machine learning models is to check whether the model is underfitting or overfitting. By improving the underfitting models, one could have a fitted model that could better capture the underlying relationship between the dependent variable (also known as the target variable) and independent variables (also known as predictors).

Meanwhile, model overfitting happens when the model instead of learning the underlying pattern of data, also learns the pattern of the noises. This could be an issue as the model

performance might deteriorate significantly when new data is presented to the model. In other words, an overfit model could have a good performance based on the training dataset, but not on a future dataset. Hence, there is no assurance to the users that the model will continue to be accurate for future datasets.

One common rule of thumb is to compare the performance of the fitted model based on the training and testing dataset. If the deviation in performances between train and test dataset is less than 5%, we can conclude that there is no sign of overfitting in the fitted model.

Another commonly used method to tackle overfitting the model is to apply the resampling technique in accessing the model performance. The key idea of resampling is to subset out a sample dataset from the training dataset to train the model and use the remaining training dataset to access the performance of the fitted model as shown under Figure 91 and Figure 92. This process of fitting a model based on a subset of data is repeated several times, which are usually specified by the users. Parameters of the model will be tuned during this process to find the set of parameters that give the best model performance with a given dataset. Nevertheless, this resampling technique would ensure a more stable model performance.
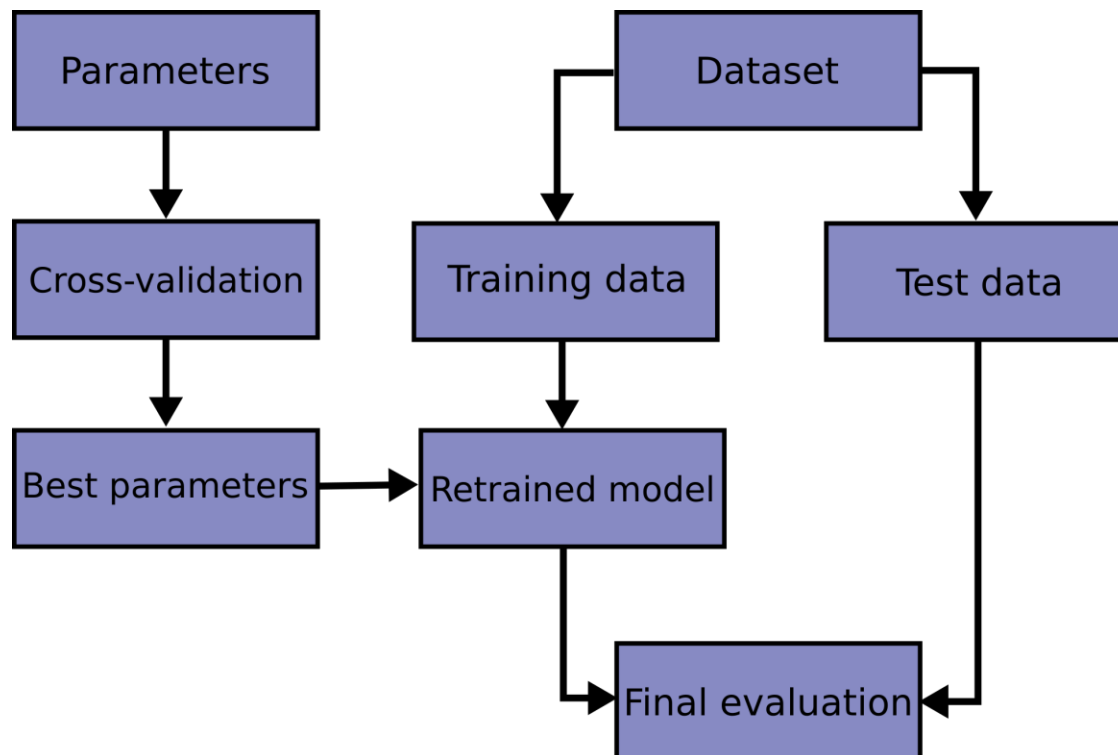
Figure 91: Grid Search Technique (Scikit-learn, n.d.)

In this analysis, K-fold cross-validation, ie. one of the more common resampling techniques will be used. Kuhn and Johnson explained in their book, Applied Predictive Modeling (Kuhn and Johnson 2013) that in k-fold cross-validation, the training data set is divided into k equal-size samples. Only one subset of the sample will be left out in model fitting for assessment later as shown below.
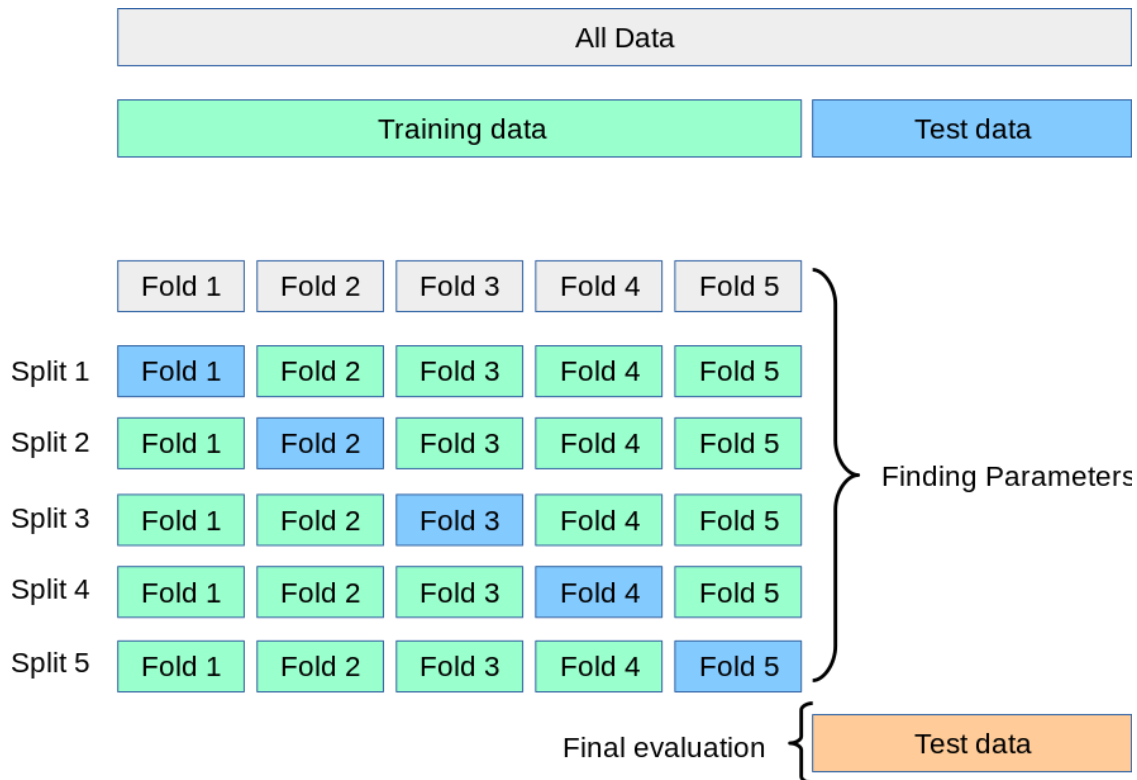
Figure 92: k-fold Cross Validation (Scikit-learn, n.d.)

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

## #  10-fold cross-validation using stratification
## # A tibble: 10 x 2
##     splits             id
##     <list>             <chr>
##  1 <split [25.1K/2.8K]> Fold01
##  2 <split [25.1K/2.8K]> Fold02
##  3 <split [25.1K/2.8K]> Fold03
##  4 <split [25.1K/2.8K]> Fold04
##  5 <split [25.1K/2.8K]> Fold05
##  6 <split [25.1K/2.8K]> Fold06
##  7 <split [25.1K/2.8K]> Fold07
##  8 <split [25.1K/2.8K]> Fold08
##  9 <split [25.1K/2.8K]> Fold09
## 10 <split [25.1K/2.8K]> Fold10
```

Figure 93: Dataset for k-fold Cross Validation

To indicate the parameters to be tuned by the algorithm, we will need to specify those parameters within the machine learning **parsnip** model function. **tune** function should also be used to mark the variables to be tuned during the cross-validation step.

While tuning the parameters would likely give a better model performance, the model fitting time is likely to increase as well. Hence if one does not want to tune all the parameters, he/she could specify the values of the parameters. For example, I have specified the number of trees should be 50, instead of tuning this tree parameter.

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

ranger_spec

## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = 50
##   min_n = tune()
##
## Computational engine: ranger
```

Figure 94: Model Specification

Once the recipe (which we have defined under Figure 63) and model specification are defined, we could chain them together as a workflow.

```
ranger_workflow <- workflow() %>%
  add_recipe(gen_recipe) %>%
  add_model(ranger_spec)
```

Figure 95: Workflow

Next, we could pass the workflow (which contains the information of recipe & model specification as shown under Figure 63 and Figure 94, the cross-validation datasets, and the number of grids required. Note that I am using the **grid_search** function in the code chunk below, where this function is performing model tuning via grid search. Other tuning methods can be found on the documentation page of the **tune** package.

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

Figure 96: Model Tuning

Once the model is tuned and the best set of parameters are found, we could select the best set of parameters from cross-validation by using the **select_best** function under the **tune** package. To generate the model performance based on the testing dataset, I will first finalize the workflow with the best set of parameters by using the **finalize_workflow** function from the **tune** package. Then I will use the **last_fit** function to fit the model with the best set of parameters and make the necessary predictions based on the testing dataset.

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

Figure 97: Finalising Model and Performing Predictions

Other considerations

While k-fold cross-validation is used in the analysis above, there are also other cross-validation methods. One of the other commonly used samplings is time-series cross-validation (also known as 'evaluation on a rolling forecasting origin'). This cross-validation method takes time sequence into considerations while performing the cross-validation. This would allow us to assess how well the model performs over time.

Following is how time-series cross-validation works:
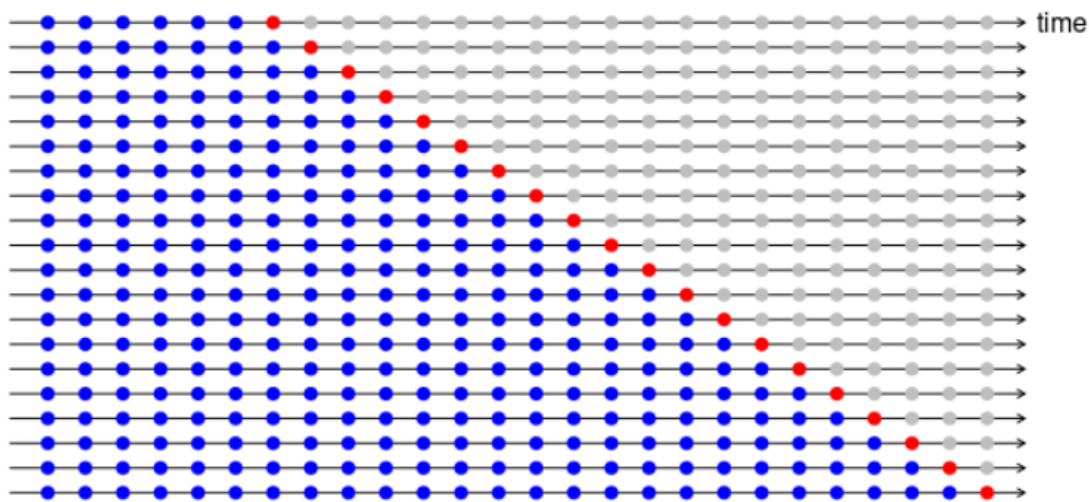
Figure 98: Time Series Cross-Validation (Hyndman and Athanasopoulos 2018)

The blue dot in the graph represents the training dataset to build the model and the red dot is used as the testing dataset.

As the focus of the research project is on how **tidymodels** can be used to help actuaries in their data analytics project, instead of finding the method that provides the highest accuracy, hence this method is out of the scope of this project.

8.4   Model Performance

Often, different types of models might be built so that one could compare which models tend to perform better for the given dataset. Different fitted models will be assessed based on a set of criteria (eg. model performance metrics, graphs). Commonly, one might use various model performances to compare the predictive power of the different fitted machine learning models. In general, below are the different types of model comparison (Kuhn and Silge 2021):

- *Within-model*: Compare same models with different features or pre-processing methods (eg. random forest fitted with all features, random forest fitted with top 10 most important variables)

- *Between-model*: Compare models fitted with different algorithms (eg. GLM vs random forest vs MARS)

In this sub-section, I will further explain how various methods could be used to compare the model performance.

8.4.1 Performance Metrics

Below are some of the metrics for different types of models:

R functions	Remarks
Regression	Root mean square error, R squared, Mean absolute scaled error
Classification	Confusion matrix, F1 score

Figure 99: Common Measurements for Model Performance

As we are working on a regression problem in this analysis, hence R squared, root mean square error, and mean absolute scaled error are being selected to measure the predictive performance of the various machine learning models.

Same as previous packages in tidymodels, this package also provides the users an unified interface on the model performance metrics.

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, ...)
```

Figure 100: Interface for Model Performance Metrics

Figure 100 shows the interfaces of the various performance metrics we would like to use in the analysis. Note that all the metrics have similar interface. It starts with defining which dataset to be used in the calculation, actual values, predicted values and any additional arguments for the relevant metrics. This has allowed us to loop through the various performance metrics to calculate the necessary values.

8.4.1.1 R squared

R squared is a measurement that calculates the coefficient of determination using correlation. (Scikit Learn, n.d.) explained that this measurement shows the proportion of variance that has been explained by the independent variables in the model. It provides the users an indication of how good the model is predicting the dependent variable. The best score for R squared is 1, which indicates that the model can fully explain the proportion of variance within the dependent variable.

Following is the formula of the R squared:

$$R^2 = 1 - \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 / \sum_{i=1}^{N}(y_i - \overline{y})^2$$

where y_i is the actual value of the target variable i, $\hat{y}_i$ is the fitted value of the target variable i, $\overline{y}$ is the mean of the dependent variable and N is the number of data points within the dataset. Higher R^2 implies the model has higher predictive power in explaining the variation of the target variable.

8.4.1.2 Root Mean Square Error (RMSE)

RMSE is one of the more common performance measurements widely used in the data science community. This measurement calculates the standard deviation of the residual terms.

Following is the formula for RMSE:

$$RMSE = \sqrt{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 / N}$$

where y_i is the actual value of target variable i, $\hat{y}_i$ is the predicted value of target variable i and N is the number of data points within the dataset. Lower RMSE indicates that on average, the predicted value is closer to the actual value.

One of the issues with this measurement is it assumes the target variable follows a normal distribution. It may not be so appropriate when the distribution of the target variable is skewed, which is quite common for the insurance claim data.

Besides, the measurement only looks at on average how well the model fits the data, which does not inform the users how well the model fits the different segments of the data. For example, we could have the scenario where two machine learning models have the same RMSE. However, one of the machine learning models only fits very well on the lower claim size and the fit on higher claim size can be further improved. This illustrates that this single measurement is unable to show that to the users.

8.4.1.3 Mean Absolute Scaled Error (MASE)

(Hallam, n.d.) explained that MASE is a metric that is scale-independent and symmetric. It is commonly used for comparing the forecasting error in the time series settings. Note that the data needs to arrange by time sequence if the underlying dataset is time-series data before using this metric to measure the model performance.

Below is the MASE formula for non-seasonal time series (Hyndman and Athanasopoulos 2018):

$$MASE = e_j / (\sum_{t=2}^{T} |y_t - y_{t-1}| / T - 1)$$

where e_j is the residual terms for target variable j, y_t is the target variable t and T is the number of the data points. MASE with less than 1 implies that the model does perform better than the average naive forecast on the training dataset.

8.4.1.4 Model Performance Comparison

To extract the various model performance from the machine learning models, one will need to specify the different model performances required. Figure 101 shows that I have specified the different required model metrics within the **metric_set** function from the **yardstick** package.

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

Figure 101: Setting the Model Metrics

Then, I have used the **collect_predictions** function to "collect" the predictions from the model before passing the predicted and actual value into the **model_metric** function created in Figure 101. Meanwhile, I have "tidied" the output into a table format so that I could join the model performance from various machine learning models for later comparison purposes.

```
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)
```

Figure 102: Extracting Model Performance

After fitting and tuning the model, the model performance results have been joined together as a tibble table as shown in the code chunk below. First, I have created an empty tibble by

using the **tibble** function from the **tibble** package. Next, I use for loop method to bind the different model performances together.

Overall Different Models Performances

```
model_metric_result <- tibble()

for (i in model_list){
  model_metric_result <- model_metric_result %>%
    bind_rows(get(paste0(i, "_metric")))

}

model_metric_result %>%
  bind_rows(lm_metric) %>%
  arrange(rmse)

## # 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
```

Figure 103: Model Performance Comparison

In general, the different model metrics tend to give consistent model performance results. In other words, all three metrics "agree" that ranger (ie. random forest) model has the highest accuracy among all the models.

Aside from that, ranger model has also outperformed the glmnet model. This is because, under GLM, the model assumes the predictors are independent of one another, which may not be an appropriate assumption. Besides, the partial regression plot looks different from the partial dependency plot for most variables. This seems to indicate that there are some interaction effects between the predictors since the partial dependency plot is measuring the average marginal effects of a variable. The two plots would look similar if there is no interaction effect.

Effect of Data Cleaning

To illustrate the importance of data cleaning, a model based on the original dataset is fitted for comparison purposes. By using the ranger model shown in Figure 103, below is the comparison of the model performance:

```
tibble() %>%
  bind_rows(ranger_org_metric) %>%
  bind_rows(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
```

Figure 104: Model Performance Comparison on With and Without Data Cleaning

As shown in Figure 104, the model performance improves after data cleaning. RMSE improves by about 96%, which indicates a significant improvement in model performance. This is because, in the earlier section, there are unreasonable values and outliers in the dataset. These data points would introduce more noise in the model fitting process, resulting in lower model performance.

Different Way of Looking at the Problem

Instead of modeling directly on the ultimate claim amount, the base model (ie. the model is shown under Figure 103) was built to predict the difference between the initial claim estimate and actual claim amount.

```
tibble() %>%
  bind_rows(ranger_ult_metric) %>%
  bind_rows(ranger_metric)

## # 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
```

Figure 105: Model Performance Comparison on Predicting Actual Claim Amount Directly vs Predicting on the Claim Difference between Initial Estimate and Actual Amount

The model results above show that the model accuracy has improved if the model is fitted on the claim difference between initial estimate and actual amount. This is because when we take the claim difference between initial estimate and actual claim amount, this has reduced the noise within the dataset. The standard deviation of the target variable has reduced significantly, translating into a higher model accuracy.

While this is not a silver bullet that this way of approaching the problem will work all the time, but it is worthwhile to think about how we could approach the data science problem differently, attempting to improve the model accuracy.

Model Performance under Different Text Mining Approaches

As discussed earlier in the report, I have also used various methods to perform text mining. Figure 106 is the comparison of the model performance of both methods. Note that the base random forest model is using method 1 to mine the features from the claim description fields.

```
tibble() %>%
  bind_rows(ranger_metric) %>%
  bind_rows(ranger_clmdesc_metric)

## # 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
```

Figure 106: Model Performance Comparison on Different Text Mining Method

Interestingly enough that the model that mines the text features by using method 2 performs better than method 1. One of the possible reasons is that the inverse document frequency included under td-idf has performed feature selection by dropping unimportant mined text variables. On the other hand, the extracted features under method 1 are not being filtered and all the extracted features were used to fit the model, which seems to create even more noise to the model. Nevertheless, while method 2 performs better in this scenario, method 1 does provide the users some flexibility on the text mining tasks.

8.4.2 Comparison by Using Visualization

Often the metric only a single figure, which does not inform the users how the model performed overall. To overcome this, I have plotted out the actual value versus the predicted value. A 45 degrees line is included so that we know which model has underpredicted or overpredicted the differences in the initial claim and ultimate claim value. The point will be above the diagonal line if the model has over-predicted the value and vice versa.
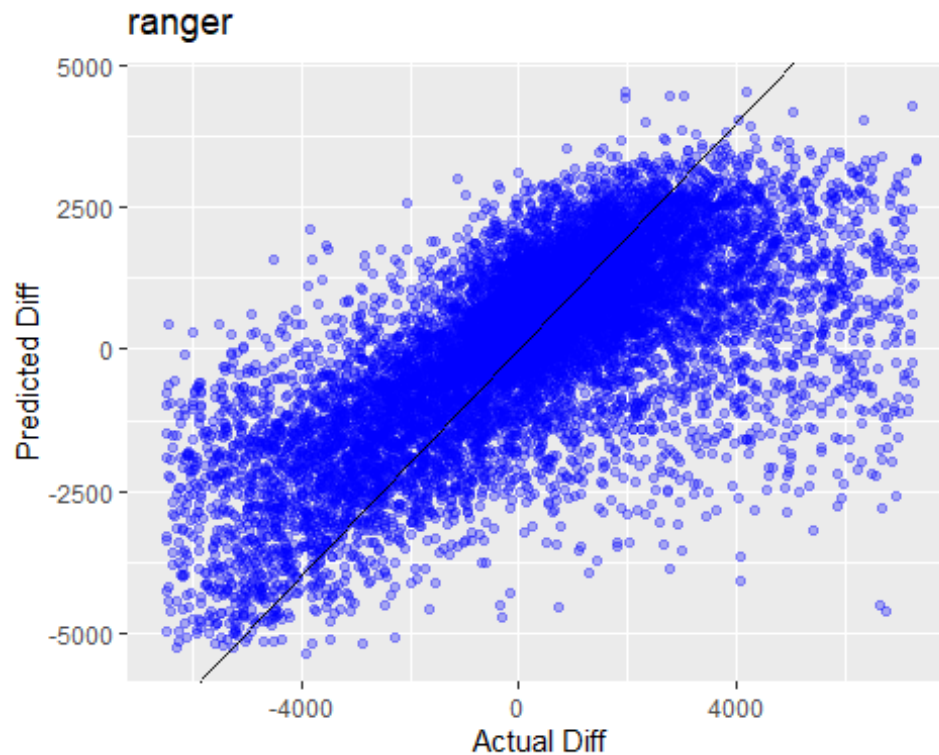
Figure 107: Graph on Actual Difference vs Predicted Difference

From the graph, the model tends to over-predict for the actual small difference and tends to under-predict for the actual large difference. The model could be further fined-tuned by understanding which variable could explain the difference between initial and actual claim amount.

8.4.3 Comparison by Using ANOVA

Often, as actuaries or statisticians, we are interested to find out whether the difference in model performance is indeed statistical different. In other words, the model difference is not due to randomness. To do so, we could perform a statistical test whether the model performance of the selected models is indeed different from one another.

The fundamental idea of such hypothesis testing is based on the ANOVA technique for comparing groups. Below is the null and alternative hypothesis for the ANOVA test:

H_0: The model performance of the selected models are the same

H_a: The model performance of the selected models are different from one another

In such a comparison, the model performance metrics selected will be the target variable and the model will be the predictors under the ANOVA model (Kuhn and Silge 2021). This can be done by using the various functions under **tidymodels** to fit a model on the difference in model performance metrics and perform the necessary test.

Apart from that, one could incorporate the prior distribution to the model parameters to allow for the variability or randomness of the model parameters. **tidyposterior** package can be used to fit Bayesian models, allowing the users to perform a comparison on these resampled models.

Nevertheless, the analysis of this model comparison method is beyond the scope of this capstone project. One could refer to **Chapter 11 in Tidy Modeling with R** for examples and more details.

8.5   Model Explainability

Model explainability has gained popularity over the years as there is increasing recognition of the ability to explain the results generated out from machine learning models. (Molnar 2021) explained that often the model performance is based on a single metric, which is an incomplete description of most real-world tasks. For example, a model with a very low RMSE does not inform the users on why the model recommends such figures.

In November 2019, The New York Times reported that Apple card was being investigated after gender discrimination complaints (Vigdor 2019). The credit card algorithm that determined the applicants' credit worthiness was reported to be "sexist" against women. The regulators were investigating the algorithm. Besides, the regulators around the world have been ramping up the relevant regulations on the governance of using artificial intelligence and machine learning models. Monetary Authority of Singapore (MAS) has released a set of generally accepted principles to promote fairness, ethics, accountability, and transparency (FEAT) in Singapore Financial Sector on 12 November 2018 (Monetary Authority of Singapore 2018).

Apart from that, in the PWC paper (Deppeler, Pattwell, and Jansen 2018), the authors identified 5 aspects of responsible AI, where interpretability is one of the important considerations for responsible AI.

Fairness: Are we minimizing bias in our data and AI models? Are we addressing bias when we use AI?

Interpretability: Can we explain how an AI model makes decisions? Can we ensure those decisions are accurate?

Robustness and security: Can we rely on an AI system's performance? Are our AI systems vulnerable to attack?

Governance: Who is accountable for AI systems? Do we have the proper controls in place?

System ethics: Do our AI systems comply with regulations? How will they impact our employees and customers?

Figure 108: 5 Aspects of Responsible AI from PWC Report

Nevertheless, these have highlighted the importance of model explainability.

8.5.1 Variable Importance

Variable importance, also known as feature importance, assigns the relative importance of each feature when making a prediction (Brownlee 2020). The author further explained that feature importance can be very useful in predictive modeling due to the following reasons:

- Variable importance scores can provide insights into the dataset

- Variable importance scores can provide insight into the model

- Variable importance can be used to improve a predictive model

To extract the variable importance from the model, **pull_workflow_fit** function from **workflow** package is used to returns the fitted model. As there are objects created during the model fitting as shown under Figure 109, I will further indicate in the formula to pass the fitted model into **vi** function.

```
names(ranger_fit)

## [1] "splits"       "id"            ".metrics"      ".notes"       ".predicti
ons"
## [6] ".workflow"
```

Figure 109: Objects under Fitted Model

```
ranger_vip <- pull_workflow_fit(ranger_fit$.workflow[[1]]) %>%
  vi()
```

Figure 110: Calculate Variable Importance

The function will return the variable importance of each variable as shown below.

```
ranger_vip

## # A tibble: 22 x 2
##    Variable                Importance
##    <chr>                        <dbl>
##  1 num_week_paid_init      34792664847.
##  2 WeeklyWages              9147427039.
##  3 injury_body              8158931749.
##  4 Age                      3930300520.
##  5 DateTimeOfAccident_year  3768714297.
##  6 DateReported_year        3415717440.
##  7 injury_cause             3139264182.
##  8 day_diff                 2587716682.
##  9 injury_type              2349171723.
## 10 HoursWorkedPerWeek       1945227667.
## # ... with 12 more rows
```

Figure 111: Variable Importance

It is easier to understand the variable importance of the different variables if they are illustrated in graph format. To do so, I will keep the top 10 variables based on the variable important values. Then, I will reorder the order of variables based on the importance value by using **fct_reorder** function. After that, I will pass the relevant values into **ggplot** function

```
ranger_vip %>%
  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")
```
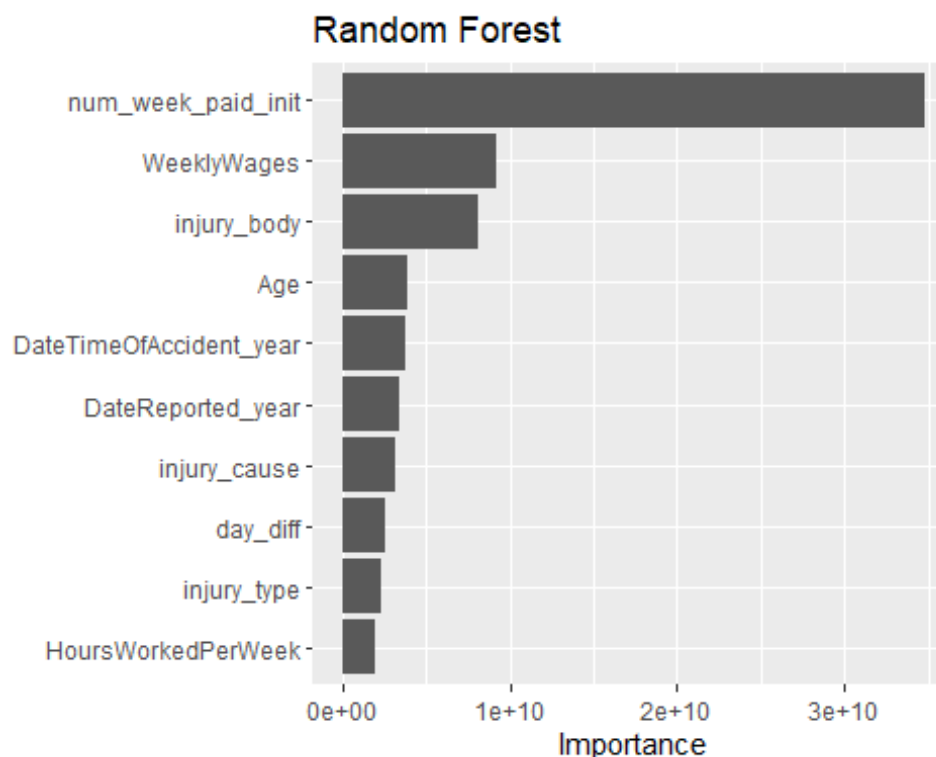

Figure 112: Graph on Variable Importance

From the graph, it is clear that some of the variables are more "important" in explaining the target variable. While variable importance is quite useful in showing us the relative importance between the different variables, it does not show how the value of the target variable changes when the independent variables change. For instance, the graph shows that num_week_paid_init is one of the most important variables. However, this graph does not show us how do the different values of num_week_paid_init affect the target variable. Hence,

to complement the understanding of how the model makes the decision, two of the common model explainability methods, ie. partial regression plot and partial dependency plot will be used to further explain the model predictions.

Also, Figure 112 also shows instead of trying to do all the analysis within the same packages, **tidymodels** has effectively leveraged on other packages to perform analysis. For instance, instead of including a plotting function in **tidymodels** package, we can pass the output from **tidymodels** to **ggplot2** to visualize the results. Also, as both functions are following tidy data concepts, it allows us to pass the output from one function to another function without needing to transform the data into required format.

8.5.2 Partial Regression Plot

A partial regression plot is known as added variable plot, adjusted variable plot, or individual coefficient plot. The key idea of a partial regression plot is to show the effect of adding variable to the model, given that the model already has one or more independent variables (National Institute of Standards & Technology 2002). It is often used to understand how the predicted value would change if we only vary one of the independent variables and fixing the values for the rest of the independent variables.

To generate such a graph, we can leverage on **step_profile** function in **recipe** package. To do so, first we need to fit the model by using **fit** function.

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

Figure 113: Fitted Model for Partial Regression Model

Next, we will create the revised recipe for partial regression plot purposes. Note that **step_profile** function from **recipe** package is used to create a dataset that fixes all variables but not the variable to be permutated over. Then, **prep** function is used to train the training data and **juice** function is used to finalize the dataset.

```
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()
```

Figure 114: Recipe for Partial Regression Plot

The code chunk above outputs a small dataset that will be used to generate the partial regression plot. The code will fix all the variables, except the variable we want to perform partial regression on.

Below is how the dataset looks like:

```
ranger_pdp

## # A tibble: 86 x 32
##    ClaimNumber DateTimeOfAccident  DateReported        Age Gender
```

```
##     <fct>         <dttm>                  <dttm>                  <dbl> <fct>
##  1 WC1592998    1996-06-21 07:00:00 1996-07-26 00:00:00    31 F
##  2 WC1592998    1996-06-21 07:00:00 1996-07-26 00:00:00    31 F
##  3 WC1592998    1996-06-21 07:00:00 1996-07-26 00:00:00    31 F
##  4 WC1592998    1996-06-21 07:00:00 1996-07-26 00:00:00    31 F
##  5 WC1592998    1996-06-21 07:00:00 1996-07-26 00:00:00    31 F
##  6 WC1592998    1996-06-21 07:00:00 1996-07-26 00:00:00    31 F
##  7 WC1592998    1996-06-21 07:00:00 1996-07-26 00:00:00    31 F
##  8 WC1592998    1996-06-21 07:00:00 1996-07-26 00:00:00    31 F
##  9 WC1592998    1996-06-21 07:00:00 1996-07-26 00:00:00    31 F
## 10 WC1592998    1996-06-21 07:00:00 1996-07-26 00:00:00    31 F
## # ... with 76 more rows, and 27 more variables: MaritalStatus <fct>,
## #   DependentChildren <dbl>, DependentsOther <dbl>, WeeklyWages <dbl>,
## #   PartTimeFullTime <fct>, HoursWorkedPerWeek <dbl>, DaysWorkedPerWeek <d
bl>,
## #   ClaimDescription <fct>, InitialIncurredClaimCost <dbl>,
## #   UltimateIncurredClaimCost <dbl>, day_diff <dbl>, acc_yr <dbl>,
## #   acc_qtr <dbl>, acc_mth <dbl>, acc_week <dbl>, report_yr <dbl>,
## #   report_qtr <dbl>, report_mth <dbl>, report_week <dbl>, acc_hr <dbl>,
## #   num_week_paid_init <dbl>, num_week_paid_ult <dbl>, injury_body <fct>,
## #   injury_side <fct>, injury_item <fct>, injury_cause <fct>, injury_type
<fct>
```

Figure 115: Dataset for Partial Regression Plot

Next, the fitted model and recipe are passed into **predict** function to generate the necessary predictions. The generated predictions will be then bind back with the dataset created under Figure 114. Also, to be more effective in visualizing different partial regression graph types for numeric variables and categorical variables, the dataset will be passed into the **ggplot** function. In the graph, num_week_paid_init is being set as the x-axis and the predicted value (ie. .pred) is set as the y axis. As num_week_paid_init is a numeric variable, hence **geom_path** function is used to plot out the line graph.

```r
predict(ranger_fit_pdp, ranger_pdp) %>%
  bind_cols(ranger_pdp) %>%
  ggplot(aes(x = num_week_paid_init, y = .pred)) +
    xlab("num_week_paid_init") +
    labs(title = "Partial Regression Plot on num_week_paid_init") +
  geom_path()
```
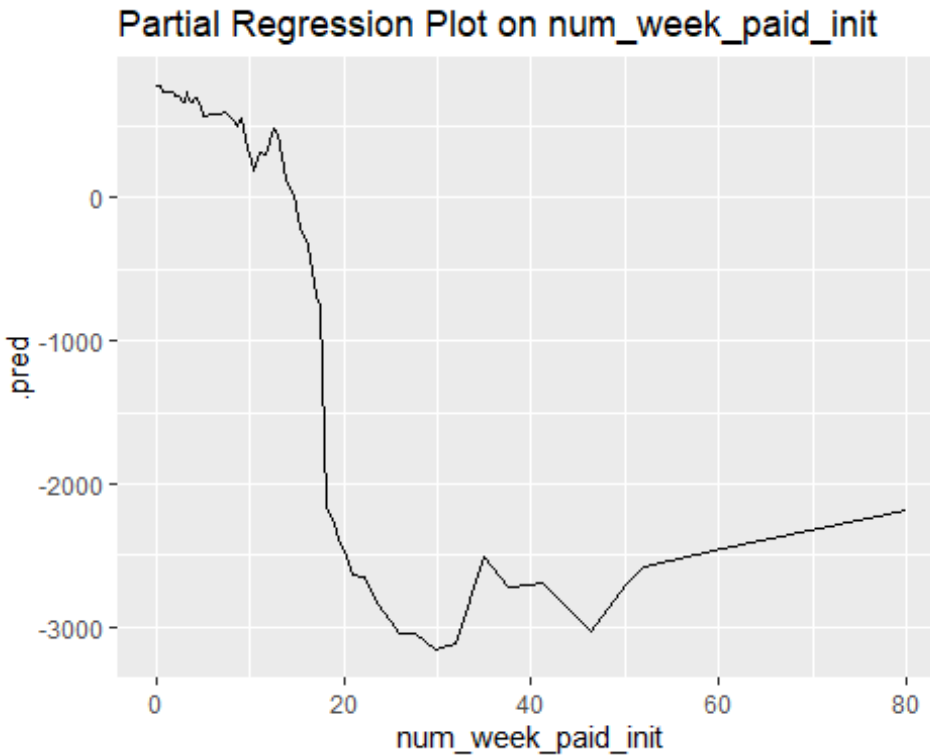
Figure 116: Partial Regression Plot

The graph above indicates that the claim tends to be understated if the num_week_paid_init is less than 15 weeks and overstated if the num_week_paid_init is more than 15 weeks. There is also a sharp drop at num_week_paid_init between 10 weeks and 20 weeks, which we may be able to improve the model accuracy by splitting dataset into shorter and longer paid weeks initially estimated.

One of the key issues of a partial dependency plot is the marginal effect of the selected independent variable is measured on a combination of the independent variables as shown in Figure 115. Hence, the marginal effect of the selected independent variable could be influenced by other independent variables used in generating the partial regression plot if the independent variables are not independent of one another. This also highlighted the second issue of partial regression plot, in which the plot assumes the independent variables are independent of one another.

Also, while the partial regression plot makes it easier to understand the marginal effect of the variable, there is a limit on how many variables can be illustrated at the same time while keeping it simple for the users to understand. It would be difficult to visualize the effect of three or more independent variables.

8.5.3 Partial Dependency Plot

A partial dependency plot is another method that allows the graph to depict the functional relationship between a small number of input variables and predictions (SAS 2008). Same as a partial regression plot, they show how the change in the independent variable affects

the value of the target variable. However, instead of illustrating the marginal effect of one independent variable based on a combination of other independent variables as shown under Figure 115, a partial dependency plot measures the average marginal effect of the independent variable by permuting the different combination of the independent variables and varying the selected independent variable under each combination. It has effectively reduced the bias in the effect measured. This method has also dampened the influence from the outliers which might exist within the dataset.

```
workflow_partial_num_week_paid_init <-
    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)
```
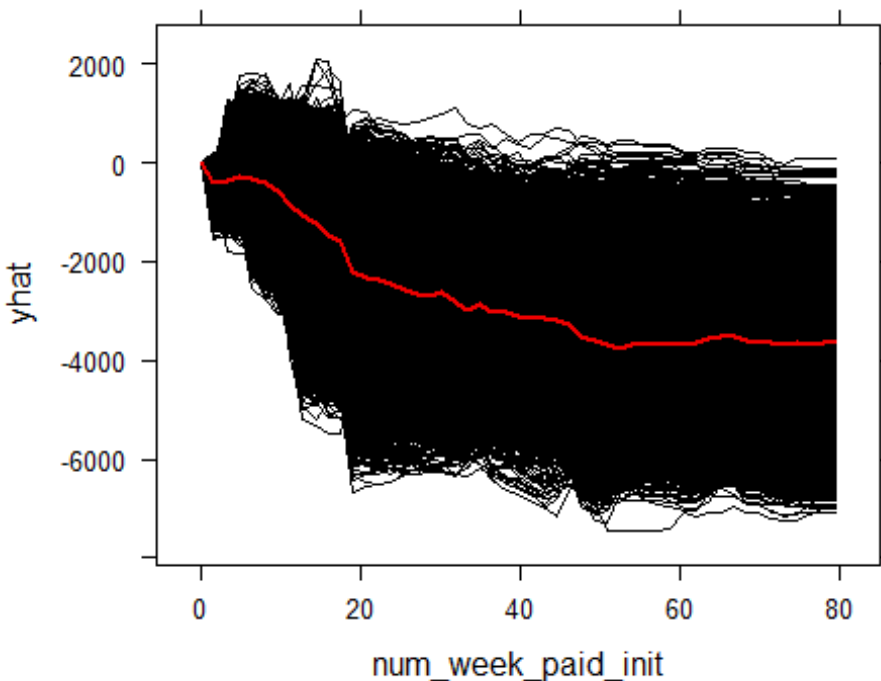

Figure 117: Partial Dependency Plot

Noted that the graph looks different from Figure 116. This is because the partial dependency plot measures the average marginal effect of the variable. The difference between partial regression plot and partial dependency plot also shows that there are some interaction effects not capturing under partial regression plot since partial regression plot only

measures the marginal effect based on one set of the combinations of variables as shown in Figure 115.

Although the partial dependency plot attempts to overcome the issue of the calculated marginal effect being influenced by the hidden interaction effects between the independent variables through permutation through the dataset, the underlying assumption is the independent variables are independent of one another. While the partial dependency plot could help one to visualize the marginal effect of how the values would change when the selected independent variable changes, the realistic number of independent variables to be used in a partial dependency plot is 2 (Molnar 2021). It would be challenging to visualize the marginal effect if the number of independent variables is more than 2.

8.6   Other Modeling Considerations

While the different algorithms can potentially help to improve the accuracy of the predictions, it is also crucial to consider other factors (eg. should we build different models based on different segments to improve the accuracy) that could improve the model performance.

Following are some of the other considerations while building machine learning models:

Any hidden structural pattern within the dataset

Is there any structural pattern within the dataset (eg. the claim amount for a certain age group is somewhat different from the rest)? If there is, we could consider splitting the dataset based on the observed structural pattern existing in the dataset and build separate machine learning models on it to improve the model performance.

In the partial regression plot of num_week_paid_init as shown under Figure 118, it seems to have some structures within the dataset. The claim amount seems to be overstated when the number of weeks initially estimated is short, understated when the number of weeks initially estimated is long.

```
predict(ranger_fit_pdp, ranger_pdp) %>%
  bind_cols(ranger_pdp) %>%
  ggplot(aes(x = num_week_paid_init, y = .pred)) +
    xlab("num_week_paid_init") +
    labs(title = "Partial Regression Plot on num_week_paid_init") +
  geom_path()
```
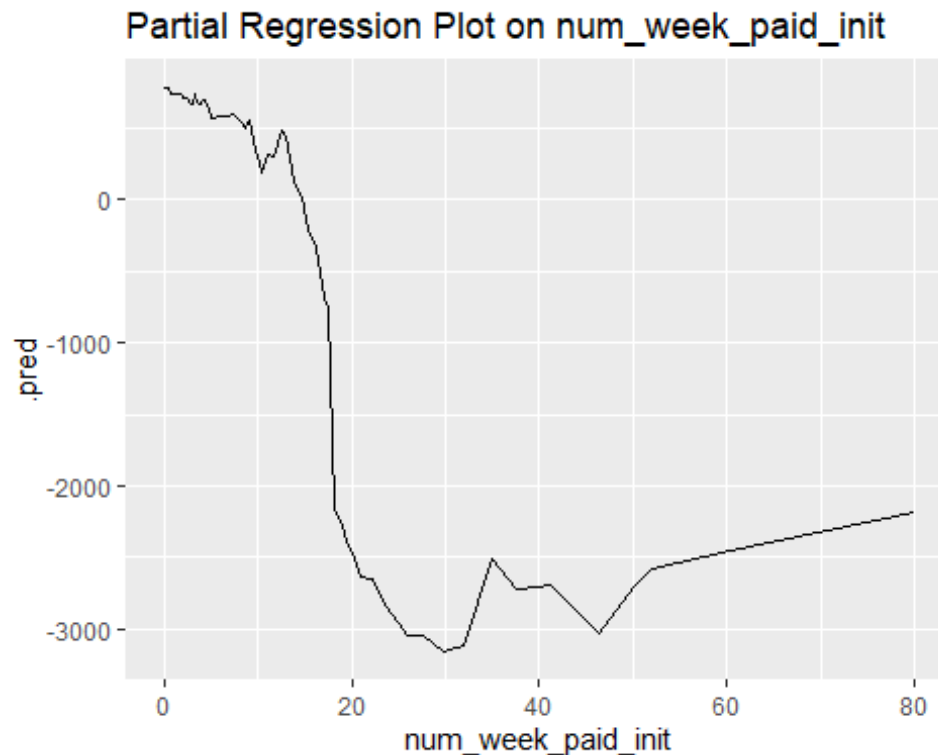
Figure 118: Partial Regression Plot

Therefore, the dataset is split into two sub-datasets and build one machine learning model on each sub-dataset.

Below is the performance of the model:

```
tibble() %>%
  bind_rows(ranger_metric) %>%
  bind_rows(ranger_20wk_metric)

## # 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_20wk  1554. 0.439 0.501
```

Figure 119: Model Performance Comparison between Original Model & Model with Subdata Set

The results show that the model accuracy has decreased slightly when I have built sub-models based on the num_week_paid_init.

A more complex model does not guarantee a better model performance

Does the model performance change significantly if we only fit the top n important variables? More variables do not guarantee a more accurate prediction from the model. Also, a simpler model could well have similar model performance as the complicated model.

To check whether a simpler model would provide a more accurate model, the top 10 most important variables shown under variable importance will be used to build a simpler model. First, I will re-use some of the model components created in the earlier steps. As I would like to fit the model with the top 10 most important variables, hence I would use **update_formula** function to update the formula in the workflow and assign this to the revised workflow a new name.

```
ranger_workflow_vip <-
  ranger_workflow %>%
  update_formula(init_ult_diff ~  num_week_paid_init +
                                    WeeklyWages +
                                    injury_body +
                                    Age +
                                    DateReported_year +
                                    DateTimeOfAccident_year +
                                    day_diff +
                                    injury_cause +
                                    HoursWorkedPerWeek +
                                    injury_type)
```

Figure 120: Update Workflow for Simpler Model

After fitting and tuning the model, below is the model performance:

```
tibble() %>%
  bind_rows(ranger_metric) %>%
  bind_rows(ranger_vip_metric)

## # 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
```

Figure 121: Model Performance Comparison between Models with All Variables and Model with Top 10 Variables

From results shown in Figure 121, it shows that the model performance of the simpler model is similar to the full model despite that the simpler model is only using lesser variables to build the model. This can be helpful when we want to deploy the model to production as a simpler model would imply fewer inputs to collect to predict the necessary outcome.

Any external factors could affect the target variables

It is also recommended to consider any external factors that could potentially affect the target variable while performing modeling. Some of the external factors could be a change in regulations on insurance payout, a change in terms and conditions stated in the policy contract, or a significant shift in insured mix. Bringing this additional information into the modeling process could potentially enhance the model performance.

As the chosen dataset is synthetic data, hence such considerations are not performed in this analysis.

9.0   Communication

(Grolemund and Wickham 2016) explained that it does not matter how great the analysis is unless we explain the results to others. Also, in the earlier section in this capstone report, one of the issues of the conventional communication method is to perform analysis and communication separately. In other words, once the analysis is performed, the users will cut and copy the relevant results into another separate document to report the necessary results. However, this may not be ideal as its prone to human error.

Therefore, to demonstrate how one could minimize human error while ensuring reproducibility, this entire report and presentation slides are written in RMarkdown and subsequently being rendered into the required formats (eg. PowerPoint, Word). Further formatting is done on the documents rendered from RMarkdown to ensure the output is more presentable. Additional files attached in the Appendix are also being attached to the report once the report is being rendered from RMarkdown.

Figure 122: Sample Presentation Slides

10.0 Conclusion

This capstone project has demonstrated how the various modern data science packages in R could provide actuaries another set of toolkits to complement the conventional actuarial

analysis. The report also demonstrates how to implement the entire analysis, from data importing, data cleaning, building model, and eventually communicating the results, entirely in RMarkdown. This would provide the readers a glimpse on how this analysis can be automated, allowing actuaries to have more time to perform the necessary analysis.

More importantly, it is also crucial to carefully examine and select the packages, instead of trying to use different individual packages and attempt to join them together to perform the necessary analysis. Therefore, **tidymodels** is used to demonstrate how the tidy modeling approach could shorten the required time to perform machine learning analysis, allowing actuaries to have more time to analyze the model results.

Besides, the modularized structure in **tidymodels** has also effectively allowed us to modularize the different model components and reused them through the analysis. This has enabled a more tidy way to create and maintain the machine learning analysis. Also, as the functions used in this analysis are following tidy data concepts, we are able to pass the output from one function to another function without needing much transformations. This would shorten the required time to prepare the analysis.

While there is a lot of active development in data science, there are many more other data science packages are being released from time to time. One could review the different new packages from time to time to see whether they could better assist actuaries in their actuarial analysis, allowing the insurers to gain a competitive advantage in this increasingly competitive market.

11.0 Acknowledgement

12.0 Appendix - R codes & HTML result files

Documents		Attachments
Dataset		[Link](#)
EDA Analysis	RMarkdown	**MITB_Capstone_Lok Jun Haur_EDA.Rmd**
	HTML	MITB_Capstone_Lok Jun Haur_EDA.html
Machine Learning Analysis	RMarkdown	**MITB_Capstone_Lok Jun Haur_ML.Rmd**
	HTML	MITB_Capstone_Lok Jun Haur_ML.html
Report	RMarkdown	**MITB_Capstone_Lok Jun Haur_Final Repo**
Presentation Slides	RMarkdown	**MITB_Capstone_Lok Jun Haur_Slides.Rmd**
	PDF	MITB_Capstone_Lok Jun Haur_Slides.pdf
	HTML	MITB_Capstone_Lok Jun Haur_Slides.htm
Poster	PowerPoint	MITB Capstone_Lok Jun Haur_Poster (po
	Image	MITB Capstone_Lok Jun Haur_Poster.JPG

13.0 Appendix - Other Filter-based Feature Selections

As discussed earlier in the report, the following methods were explored during the experimental stages to understand how these methods can be used to explain the underlying relationships between categorical target variable and predictors:

- Chi-square

- Mosaic Plot

- Cross Plot

- Plotar

These methods were not included in the main text of the report as the target variable of the selected variable is continuous. Hence, making these methods irrelevant for the analysis. Nevertheless, the discussion of the methods above can be found below.

Filter-based Feature Selection for Categorical Target Variable: Chi-square

One of the methods that can be used to understand whether the categorical input variables are independent of the categorical target variable (ie. there is no relationship between target and predictors since they are independent) is to perform a chi-square test. If the predictor is independent of the target variable, then it will be unlikely that the predictor has a predictive power to explain the target variable.

To perform the chi-square test, one could use **infer** package, ie. the chi-square package recommended on **tidymodels** page.
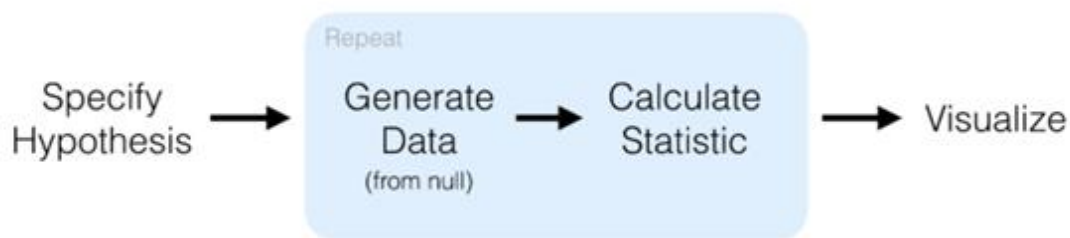

Figure 123: Workflow on how to infer package visualize statistical tests

Under the chi-square test, the null hypothesis is there is no relationship between the selected categorical variables. In other words, they are independent of one another. The alternative hypothesis is there is statistical evidence that there are some relationships between the categorical variables.

Infer package allows the users to visualize the chi-square test in graph format, making it easier for the audience to understand the results. This is done by generating the null hypothesis. Following is an example of how the output from **infer** package would look like:
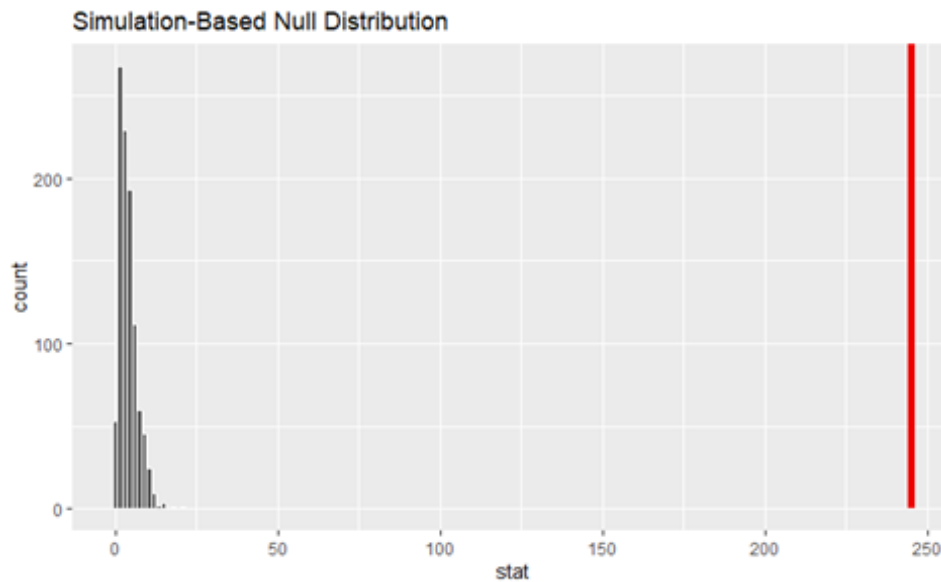
Figure 124: Example of Chi-square

Taking the example shown, we would reject the null hypothesis and conclude that there is statistical evidence that the selected variables are not independent.

Filter-based Feature Selection for Categorical Target Variable: Mosaic Plot

While chi-square tests whether the predictors are independent of the target variable, it does not inform the users which categories are different from one another if the predictors are not independent of the target variable. Hence, a mosaic plot is one of the easier ways to illustrate how the relationship with the target variables differs by different categories.

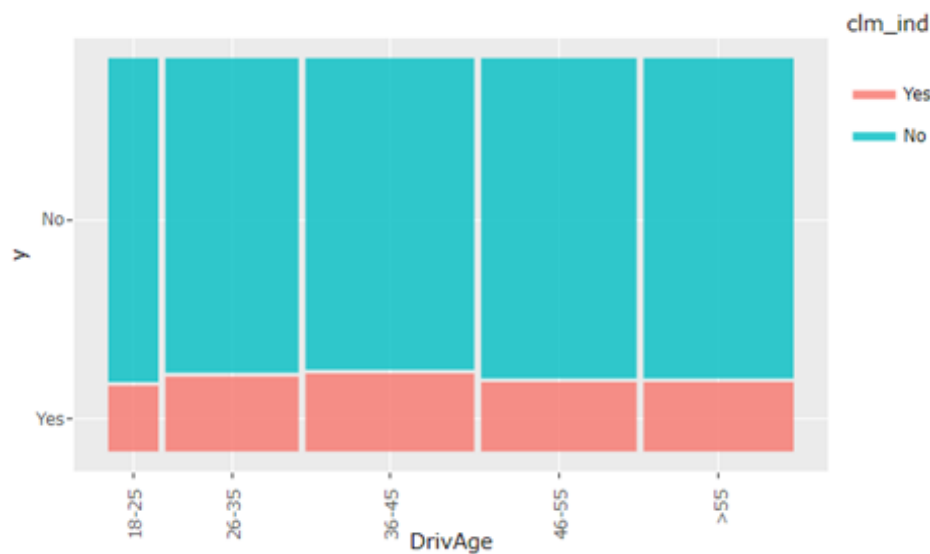Following is an example of a mosaic plot:

Figure 125: Example of Mosaic Plot

The benefit of using this package is an extension of **ggplot2**, allowing the users to customize the graph formatting based on their requirements.

Filter-based Feature Selection for Categorical Target Variable: Cross Plot

Alternatively, one could use the **cross_plot** function within **funModeling** package. Instead of plotting the relationship between all categorical input variables and the target variable one by one, this function will automatically generate all the graphs that show the relationship between categorical input variables and the target variable.

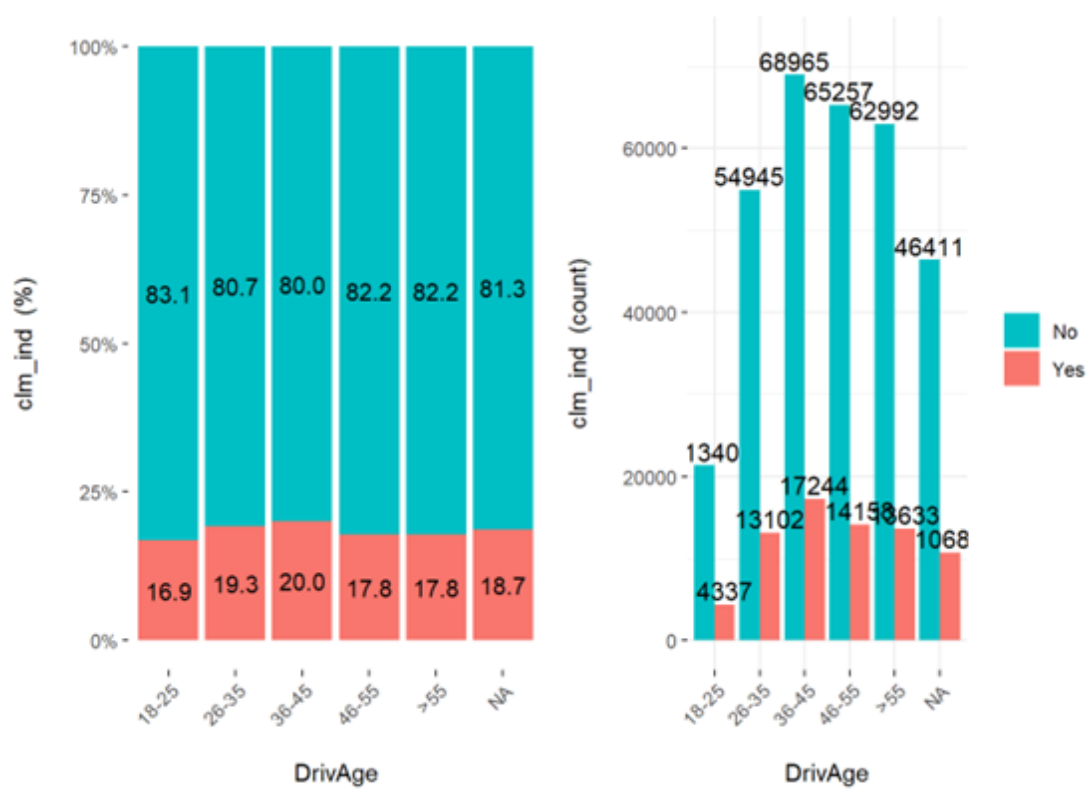Following is an output example from the **cross_plot** function:

Figure 126: Example of cross_plot

For a good predictor, we should be able to observe a significant difference in the proportion bar chart (ie. the graph on the left side).

Filter-based Feature Selection for Categorical Target Variable: Plotar

If the predictors are numeric variables, then **plotar** function from **funModeling** package can be used to plot out the density graph. Same as **cross_plot** function, this function will automatically generate all the graphs between continuous input variables and the categorical target variable.

```
plotar(data = data_log, target = "clm_ind", plot_type = "histdens")
```
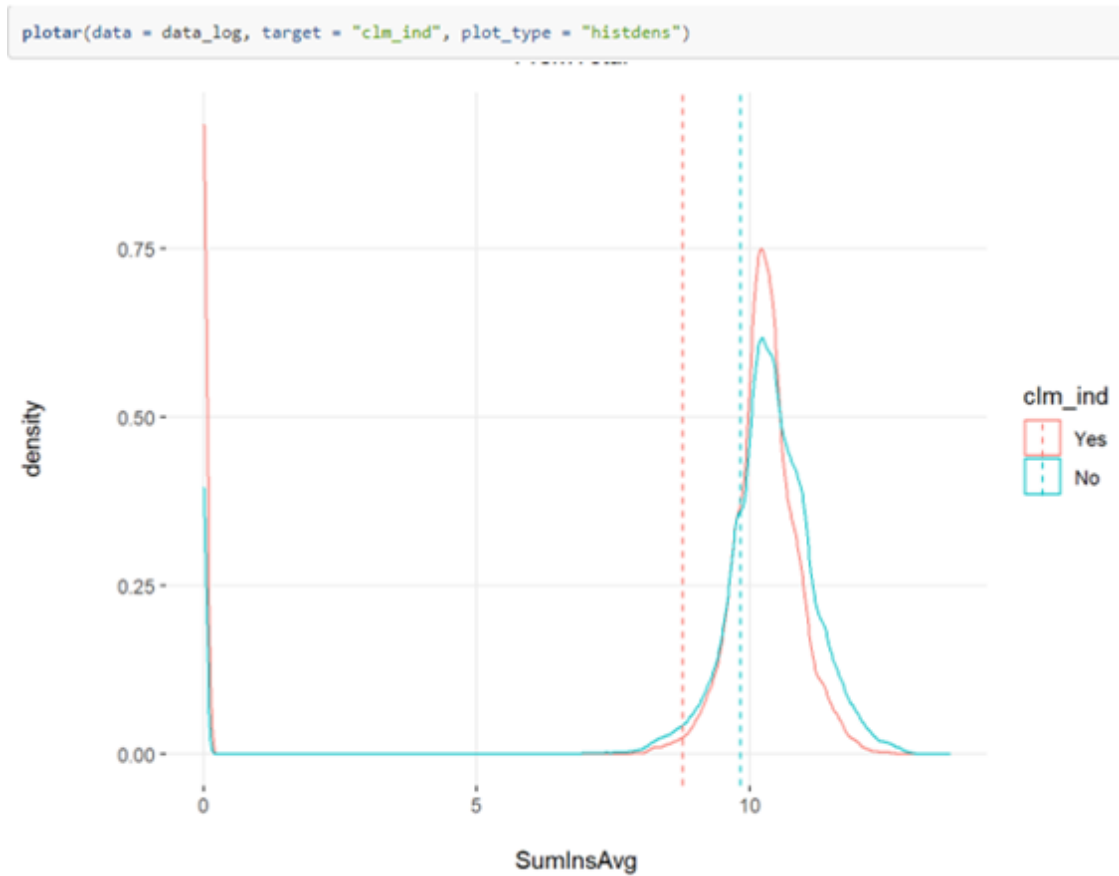

Figure 127: Example of plotar

The density graph will show whether the continuous input variables can explain the categorical target variable. Same as **cross_plot** function, if the different density graphs do not overlap much, this implies that the continuous input variable is a good predictor.

14.0 Reference

Alfredo, Spedicato Giorgio, Christophe Dutang, and Leonardo Petrini. 2018. "Machine Learning Methods to Perform Pricing Optimization: A Comparison with Standard Generalized Linear Models" 12:1: 69–89.

Baker, Monya. 2016. "1,500 Scientists Lift the Lid on Reproducibility." https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970.

Bhatt, Shweta, ed. n.d. "5 Things You Need to Know About Reinforcement Learning." https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html.

Bock, Tim. n.d. "What Is Reproducible Research?" https://www.displayr.com/what-is-reproducible-research/.

Brownlee, Jason, ed. 2020. "How to Calculate Feature Importance with Python." https://machinelearningmastery.com/calculate-feature-importance-with-python/.

Burns, Eileen, Gene Dan, Anders Larson, Bob Meyer, Zohair Motiwalla, Guy Yollin, and Milliman. 2019. "Considerations for Predictive Modeling in Insurance Applications," May.

Casas, Pablo, ed. 2019. *Data Science Live Book*.

Cleveland R User Group. 2021. *TidyModels by Max Kuhn (2/24/2021)*. https://www.youtube.com/watch?v=kAZe9UpMx_s.

Deppeler, Andreas, Jennifer Pattwell, and Mark Jansen. 2018. "Building Trust in Ai and Data Analytics: MAS 'Principles for the Use of Artificial Intelligence and Data Analytics' and Their Impact on Financial Services Organisations in Singapore," December. https://www.pwc.com/sg/en/publications/assets/building-trust-ai-data-analytics-122018.pdf.

Diana, Alex, Jim E. Griffin, Jaideep Oberoi, and Ji Yao. 2019. "Machine-Learning Methods for Insurance Applications - a Survey," January.

Franklin, Warren. 2016. "Machine Learning Algorithm Vs. Actuarial Science.who Will Win?" August. https://towardsdatascience.com/machine-learning-algorithm-vs-actuarial-science-who-will-win-b203f31145ce.

Grolemund, Garrett, and Hadley Wickham, eds. 2016. *R for Data Science*. https://r4ds.had.co.nz/explore-intro.html.

Hallam, Alex, ed. n.d. "Mean Absolute Scaled Error." https://yardstick.tidymodels.org/reference/mase.html.

Horst, Allison, ed. n.d. https://github.com/allisonhorst/stats-illustrations.

Hyndman, Rob J., and George Athanasopoulos, eds. 2018. *3.4 Evaluating Forecast Accuracy*. OTexts.com/fpp2.

Institute and Faculty of Actuaries. n.d. "What Is Data Science: An Actuarial Viewpoint." https://www.actuaries.org.uk/learn-and-develop/lifelong-learning/what-data-science-actuarial-viewpoint.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani, eds. 2013. *An Introduction to Statistical Learning with Applications in R*.

Kam, Tin Seong. 2019. "Hands-on Exercise 8: Visualising Correlation Matrix," November. https://rpubs.com/tskam/Corrgram.

———. 2020a. "Lesson 1: Introduction to Visual Analytics and Applications," January.

———. 2020b. "The Terrible Twins of Tidyverse - Introducing Tidyr and Dplyr," May.

Kuhn, Max, and Kjell Johnson, eds. 2013. *Applied Predictive Modeling*.

Kuhn, Max, and Julia Silge. 2021. *Tidy Modeling with R*. https://www.tmwr.org/.

Kuhn, Max, and Davis Vaughan, eds. n.d.a. "General Interface for Boosted Trees." https://parsnip.tidymodels.org/reference/boost_tree.html.

———, eds. n.d.b. "General Interface for Decision Tree Models." https://parsnip.tidymodels.org/reference/decision_tree.html.

———, eds. n.d.c. "General Interface for K-Nearest Neighbor Models." https://parsnip.tidymodels.org/reference/nearest_neighbor.html.

———, eds. n.d.d. "General Interface for Mars." https://parsnip.tidymodels.org/reference/mars.html.

———, eds. n.d.e. "General Interface for Random Forest Models." https://parsnip.tidymodels.org/reference/rand_forest.html.

Lang, Michel, Patrick Schratz, Raphael Sonabend, Marc Becker, and Damir Pulatov. 2021. "Mlr3viz." https://cran.r-project.org/web/packages/mlr3viz/mlr3viz.pdf#page=9&zoom=100,132,89.

McKinsey Company. 2020. "Reimagining Actuaries: A Q&A with Society of Actuaries' Greg Heidrich," August. https://www.mckinsey.com/~/media/McKinsey/Industries/Financial%20Services/Our%20Insights/Reimagining%20actuaries%20A%20Q%20and%20A%20with%20Society%20of%20Actuaries%20Greg%20Heidrich/Reimagining-actuaries-A-Q-and-A-with-Society-of-Actuaries-Greg-Heidrich.

Michel, Denuit, Hainaut Donatien, and Trufin Julien, eds. 2019. *Effective Statistical Learning Methods for Actuaries I - Glm and Extensions*.

Molnar, Christoph, ed. 2021. *Interpretable Machine Learning - a Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book/.

Monetary Authority of Singapore. 2018. "Principles to Promote Fairness, Ethics, Accountability and Transparency (Feat) in the Use of Artificial Intelligence and Data Analytics in Singapore's Financial Sector," November.

National Institute of Standards & Technology, ed. 2002. "Partial Regression Plot." https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/partregr.htm.

Patil, Indrajeet. n.d. "Ggstatsplot: Ggplot2 Based Plots with Statistical Details." https://indrajeetpatil.github.io/ggstatsplot/index.html.

Perkins, Steven, and Valerie du Preez. 2018. "Machine Learning for Actuaries," July.

Quantee. n.d. "Actuarial Data Science." https://quantee.ai/actuarial-data-science/.

RStudio. 2018. "Caret to Tidymodels." https://community.rstudio.com/t/caret-to-tidymodels/13606/3.

———. n.d. "Tidyverse." https://www.tidyverse.org/.

SAS, ed. 2008. "Interpret Model Predictions with Partial Dependence and Individual Conditional Expectation Plots." https://blogs.sas.com/content/subconsciousmusings/2018/06/12/interpret-model-predictions-with-partial-dependence-and-individual-conditional-expectation-plots/#:~:text=A%20partial%20dependence%20(PD)%20plot,flu%20increases%20linearly%20with%20fever.

Schelldorfer, Jurg, and Mario V Wuthrich. 2019. "Nesting Classical Actuarial Models into Neural Networks," January.

Scikit-learn, ed. n.d. "Cross-Validation: Evaluating Estimator Performance." https://scikit-learn.org/stable/modules/cross_validation.html.

Scikit Learn. n.d. "Metrics and Scoring: Quantifying the Quality of Predictions." https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score.

Sethi, Alakh, ed. 2020. "Want to Ace Data Science Hackathons? This Feature Engineering Guide Is for You." https://www.analyticsvidhya.com/blog/2020/06/feature-engineering-guide-data-science-hackathons/.

Sheng, Cliff, Matthew Leonard, Prashanth Gangu, Kang Liu, John Bi, Wayne Xu, and Karen Ji. 2016. "China Insuretech - Industry Report."

Shneiderman, Ben. 2005. "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualization." *IEEE Conference on Visual*.

Sievert, Carson, ed. 2019. *Interactive Web-Based Data Visualization with R, Plotly, and Shiny*. CRC Press.

Singh, Gurchetan. 2018. "A Simple Introduction to Anova (with Applications in Excel)." https://www.analyticsvidhya.com/blog/2018/01/anova-analysis-of-variance/.

Smith Hanley Associate LLC. 2018. "Will Data Scientists Replace Actuaries?" January. https://www.smithhanley.com/2018/01/11/will-data-scientists-replace-actuaries/.

Spedicato, Giorgio Alfredo, Christophe Dutang, and Leonardo Petrini. 2018. "Machine Learning Methods to Perform Pricing Optimization. A Comparison with Standard Glms." *ResearchGate*.

Sundaram, Ramya Bhaskar, ed. 2018. "An End-to-End Guide to Understand the Math Behind Xgboost." https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/.

Tierney, Nicholas, ed. 2020. *RMarkdown for Scientists*. https://rmd4sci.njtierney.com/index.html.

Tukey, John W. 1961. "The Future of Data Analysis." *The Annals of Mathematical Statistics*.

Vigdor, Neil, ed. 2019. "Apple Card Investigated After Gender Discrimination Complaints." https://www.nytimes.com/2019/11/10/business/Apple-credit-card-investigation.html.

Wickham, Hadley. 2020. "The Tidy Tools Manifesto." https://tidyverse.tidyverse.org/articles/manifesto.html.

Wickham, Hadley, and Jim Hester, eds. n.d. "Readr." https://readr.tidyverse.org/.

Zhou, John, and Debbie Deng. 2019. "GLM Vs. Machine Leaning - with Case Studies in Pricing."