A Study on Workplace Accidents

Ethan Tan Wee En  
*p2012085*DAAA/FT/2A/03  
Diploma in Applied A.I. and Analytics

Singapore PolytechnicDover Road, Singapore  
ethantan.20@ichat.sp.edu.sg

*Abstract*

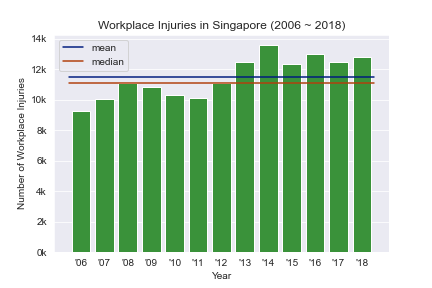
*Workplace accidents are frequent, and are, more often than not, benign. However, this does not mean that severe or even fatal accidents do not occur. From Manufacturing to Social Services, no industry is safe from harmful accidents. This paper seeks to investigate the most dangerous and hazardous types of workplace incidents and aims to outline the steps involved in designing a relevant machine learning model to predict the outcome of any such accident. The content comprises the general steps taken to build, train, score and evaluate the machine learning model. The data is obtained from the data.gov.sg website. The task is a classification problem, and the target of the machine learning model's predictions is the outcome of an accident at work, whether it be a minor or major injury or if it be fatal, given the details of the accident.*

Keywords

Technical Paper, Classification, Workplace, Accidents, Incidents

# Introduction

Workplace accidents are not new. In fact, more than 10,000 of such local cases are reported annually.



It is of this study's interest to investigate which are the more crucial factors of such workplace incidents (i.e., which lead to the most severe outcomes).

# Related Works

This paper is not the first of its kind to be published; other similar studies and research have been conducted [2][3] prior to the writing of this paper. This paper is neither a proof, an assertion, nor does it intend to compete with the above-mentioned works or other. Please refer to these articles [2][3] for more in-depth and comprehensive compilations.

# Experiment

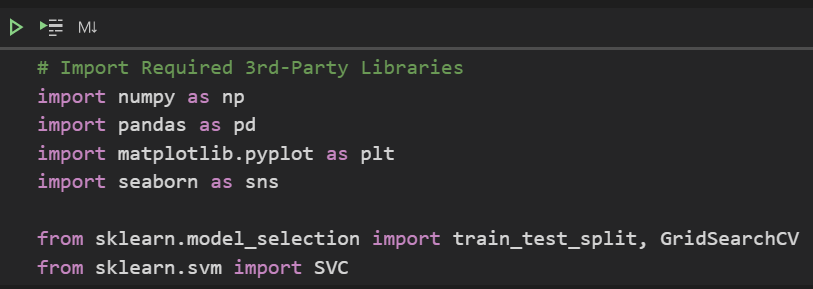
## Workspace

For this study, we will be using Python as our programming language of choice since there are numerous machine learning packages already available for use.

We will write and run all our code in a Jupyter notebook because it is convenient to observe all the output in an organized fashion.

## Dependencies

To lessen our workload and avoid reinventing the wheel, we will be utilizing third-party libraries, such as NumPy, Pandas, Matplotlib, Seaborn and Scikit-Learn.



This can be easily achieved as depicted in the figure above.

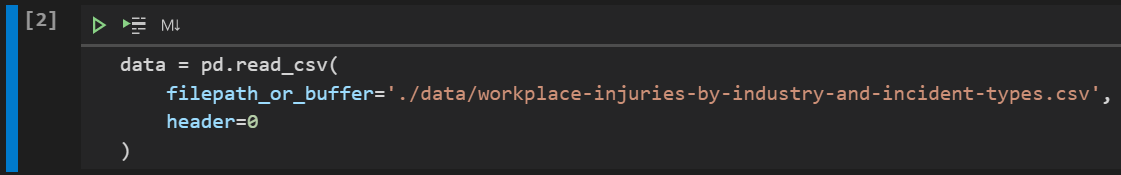
*Notes:*

1. *To keep this study simple, we will only consider the Support Vector Machine classification algorithm*
2. *These dependencies are not exhaustive (i.e., we might need to import other dependencies later)*

## Data

Before we begin to build any machine learning model, we need data, and, preferably, a lot of it. In our case, we have obtained our data from the data.gov.sg website [1]. The file of interest is titled "workplace-injuries-by-industry-and-incident-types", in Comma-Separated Values (csv) format.

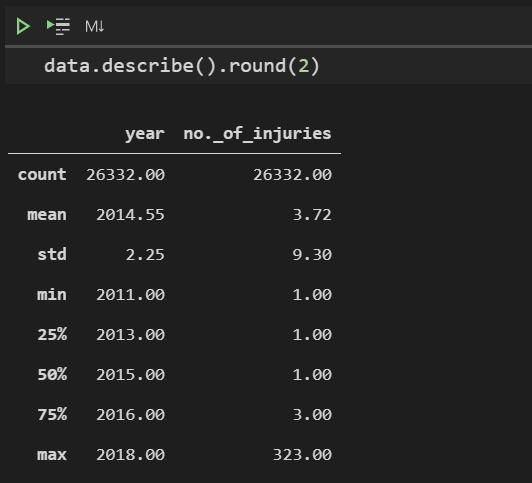
We will import our data using the *read\_csv* function defined in the *pandas* package. We will set the *header* parameter to 0 to specify that the first row of the file contains the header names.



The first thing we will do after importing the data is to inspect it. By the *head* method of the *pandas* *DataFrame* data structure, we can view the top 5 rows of our raw dataset.

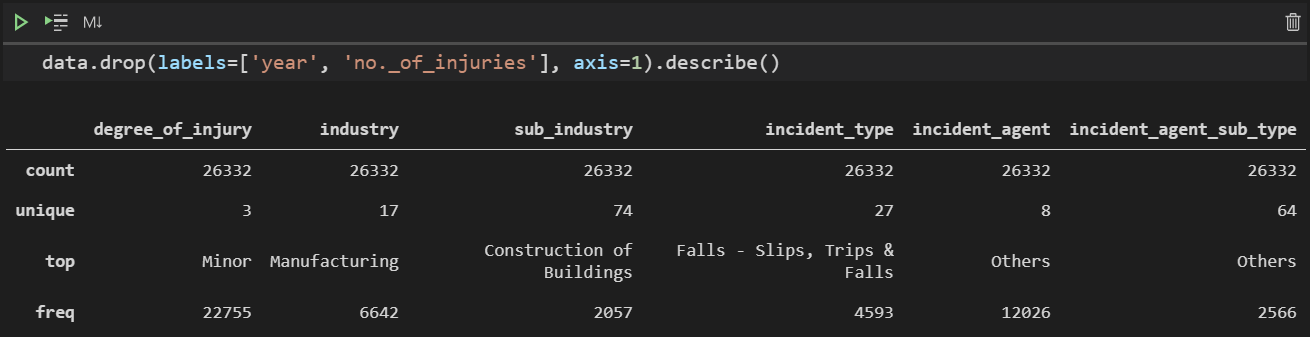


Next, we will take a look at the numerical summaries of the feature variables in the dataset. We will use the *describe* and *round* methods to generate the numerical summaries for the numerical variables in the dataset precise to 2 decimal places.



From this, we can tell that the dataset contains data from the year 2006 to 2018, and that there are only two numerical variables in the data.

We can use the same *describe* method to generate numerical summaries for the non-numerical variables. However, we need to drop the *year* and *no.\_of\_injuries­* columns temporarily as we want to focus on the non-numerical variables.

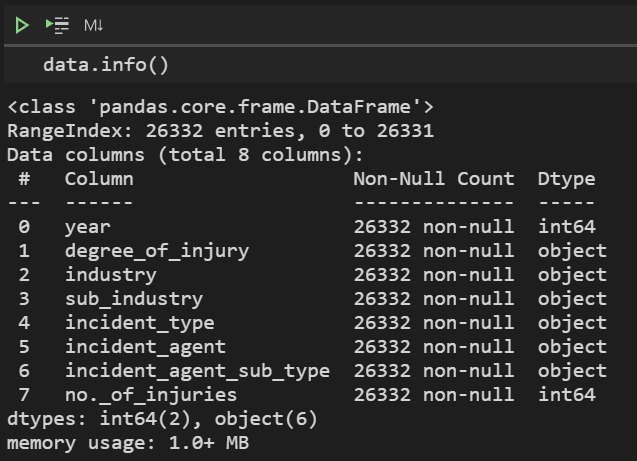


From this, we can tell that most workplace accidents are due to falls and the majority of incidents result in minor injuries. This is rather expected. There are 17 industries involved, which can be drilled down into 74 sub-industries.

Since these are currently categorical, we will have to conduct some form of encoding later on so that they are converted to numeric.

Preferably, we should combine some, if not most of the values to mitigate the *Curse of Dimensionality* when we encode the data subsequently.

We will perform one more inspection of the data using the *info* method to check if there are any missing values in the dataset.

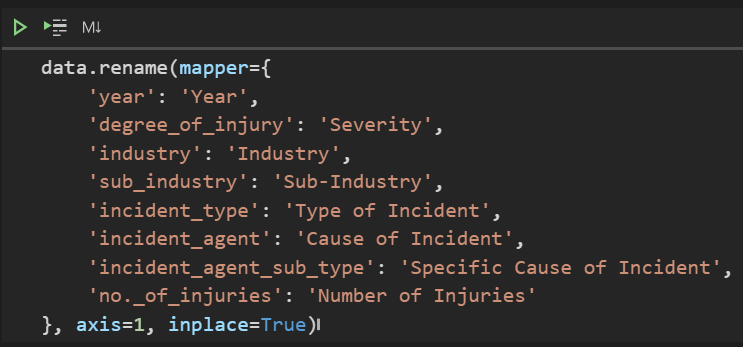


It seems there are no missing values in our data. We can thus skip any missing value imputation in the next step (data pre-processing).

## Pre-Processing

1. *Column Refactoring*

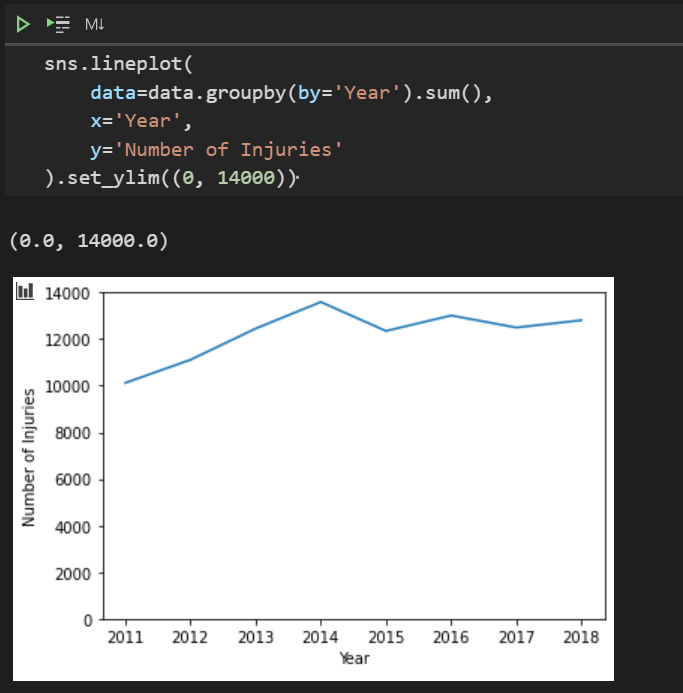
To improve readability and consistency, we will rename the columns in the *DataFrame*.



That's better. The columns have now been standardized, but the values have yet to be processed. Next, we will conduct exploratory data analysis.

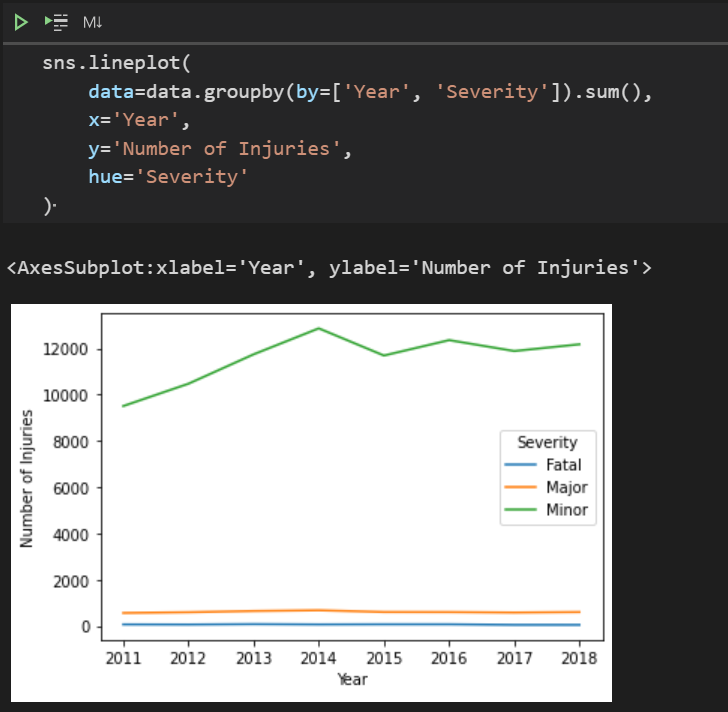
1. *Exploratory Data Analysis (EDA)*

In this step, we will explore the distributions and patterns in the data, mostly through graphs and plots.



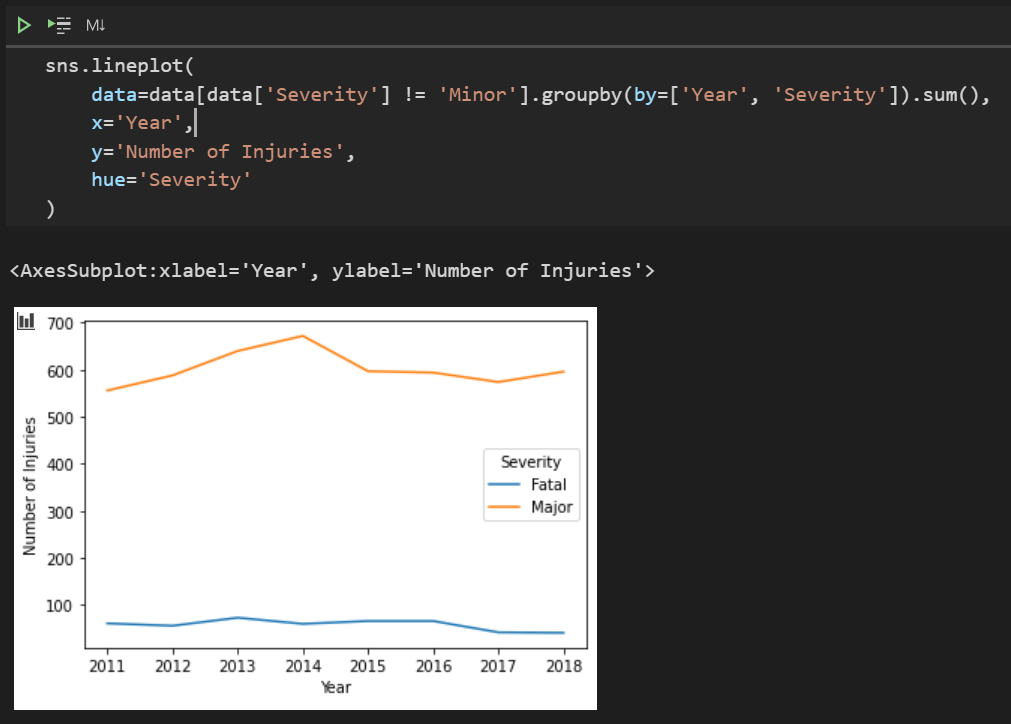
The number of workplace injuries peaked in 2014; nevertheless, number of workplace injuries seems to be on the rise.

Modifying the code above to account for the different severities of the accidents, we get the following plot:

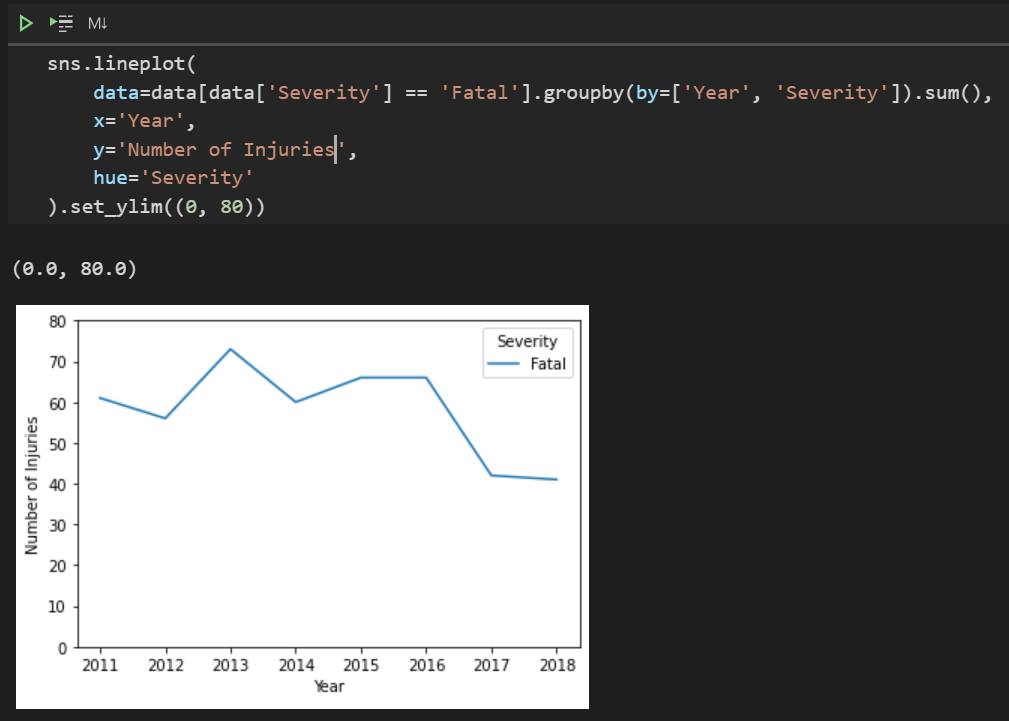


Minor injuries appear to make up most of the injuries. The overall trend seems to be represented by the trend of minor injuries.

Taking minor injuries out of consideration, we get the following plot instead:

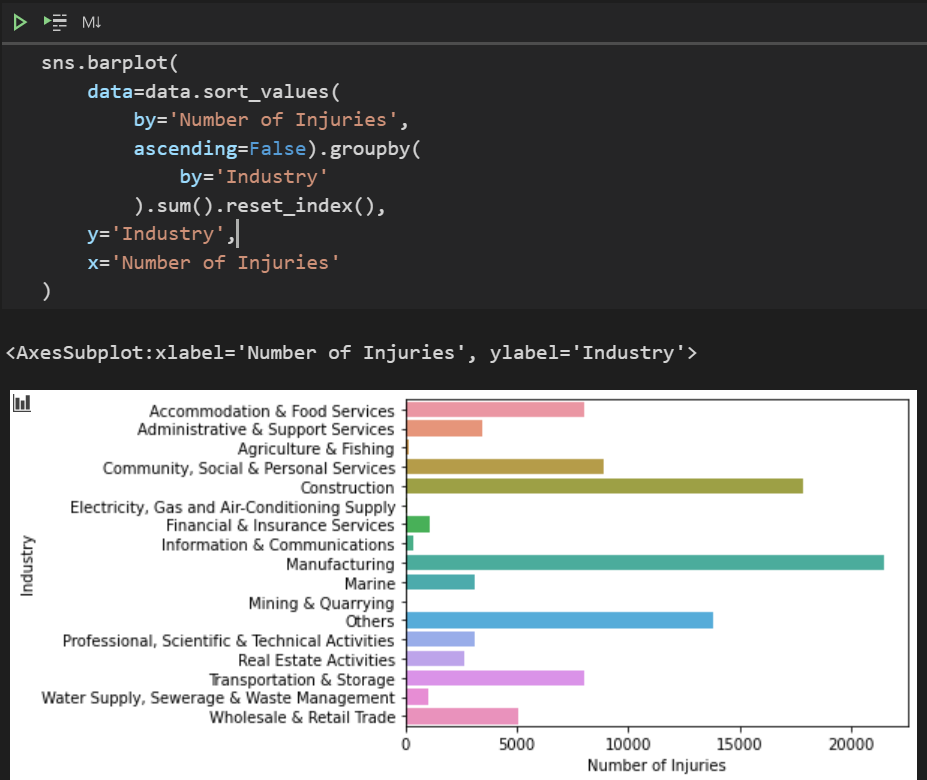


Number of major injuries are roughly the same throughout the years. Number of fatal injuries is still masked by the number of major industries, so let's focus only on fatal injuries.



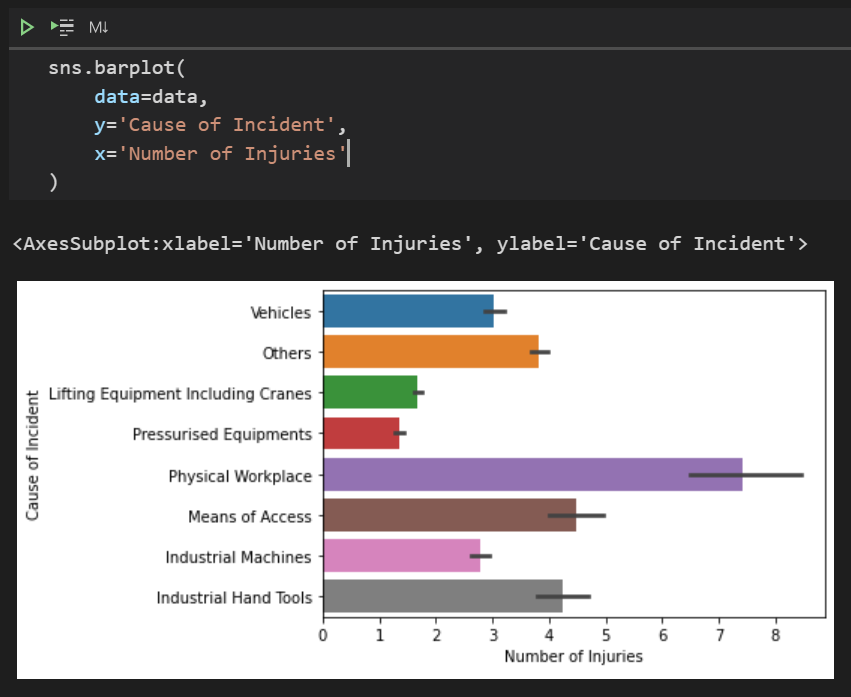
The number of fatal injuries seems to oscillate and fluctuate but follows a downward trend.

Now, let's investigate which industries do more accidents occur, and more severe ones. Merely plotting produces the following graph:



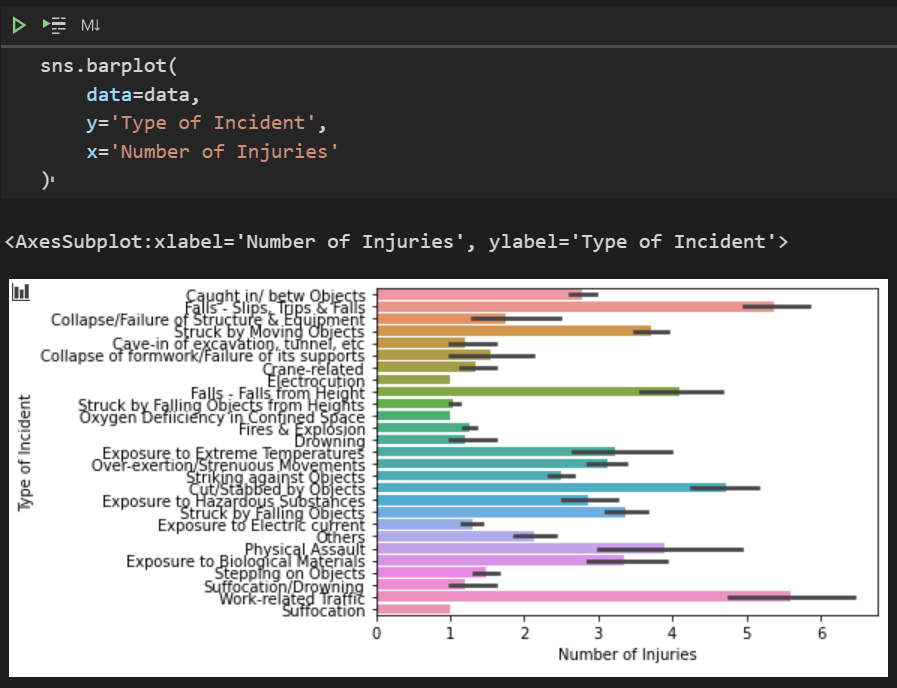
It is evident that there are too many categories for the column *Industry*. *Sub-Industry* is even more variable, with 74 possible categories. Encoding 74, or even 73, categories for one variable is definitely going to lead to a sparse matrix, which will prove challenging for our machine learning model to generalize

Plotting the *Number of Injuries* against the *Cause of Incident* reveals this:



Some of the categories for *Cause of Incident* can be grouped together too.

In addition, this is the plot for *Number of Injuries* against *Type of Incident*:

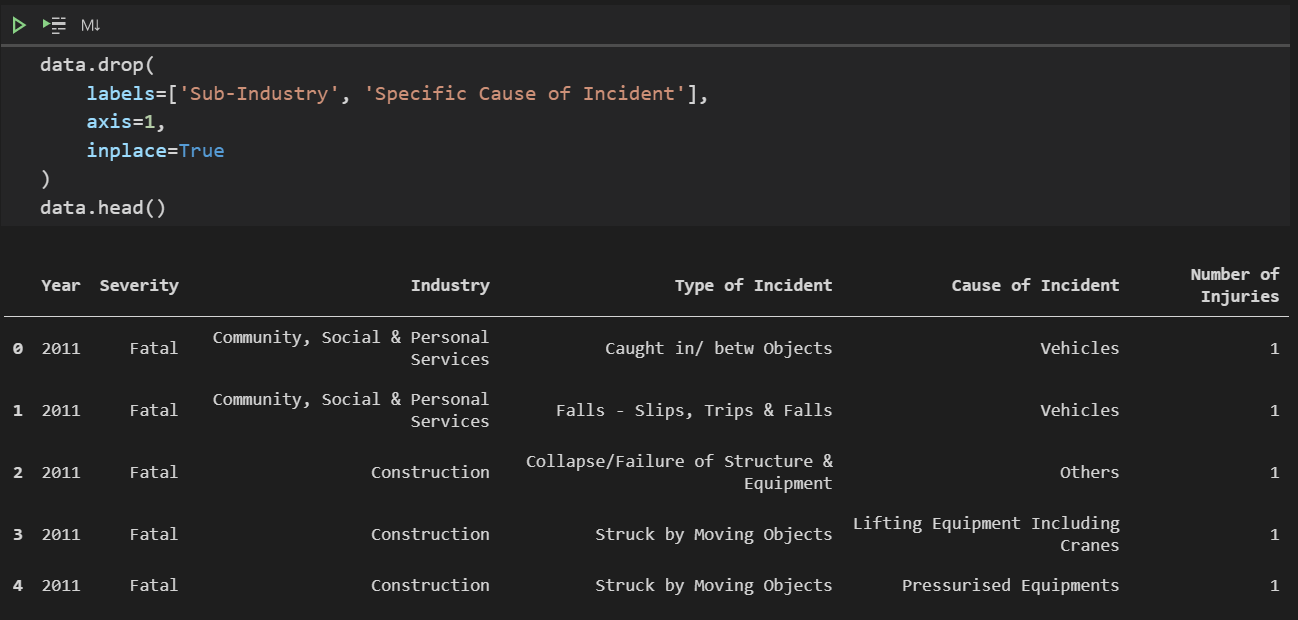


We have to merge categories for *Industry*, *Cause of Incident* and *Type of Incident*, in order to reduce the number of dimensions when we One-Hot Encode the data.

1. *Feature Selection*

Considering there are so many categories in *Industry*, *Cause of Incident* and *Type of Incident* already, we no longer need the sub-category columns, namely *Sub-Industry* and *Specific Cause of Incident*.

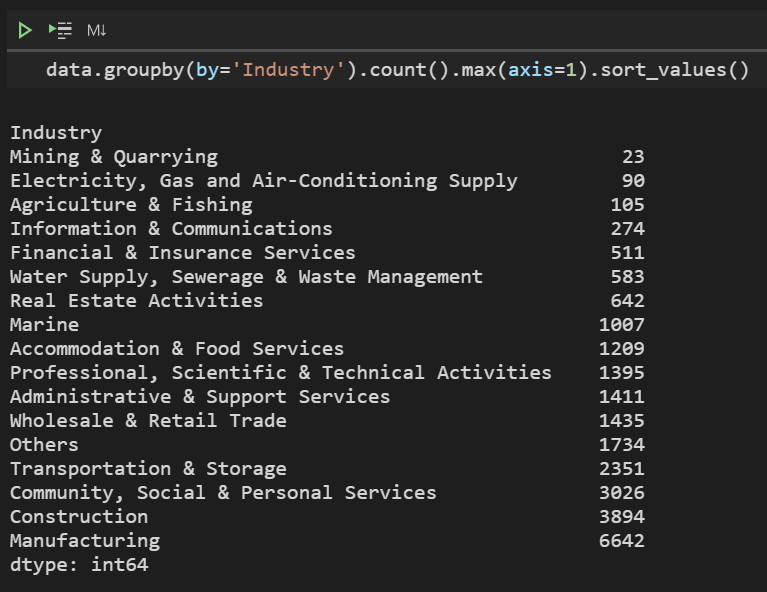
Hence, we can remove the columns using the *drop* method and specifying the removal to be performed *in-place*.



Now our dataset is more focused.

1. *Categorical Merging*

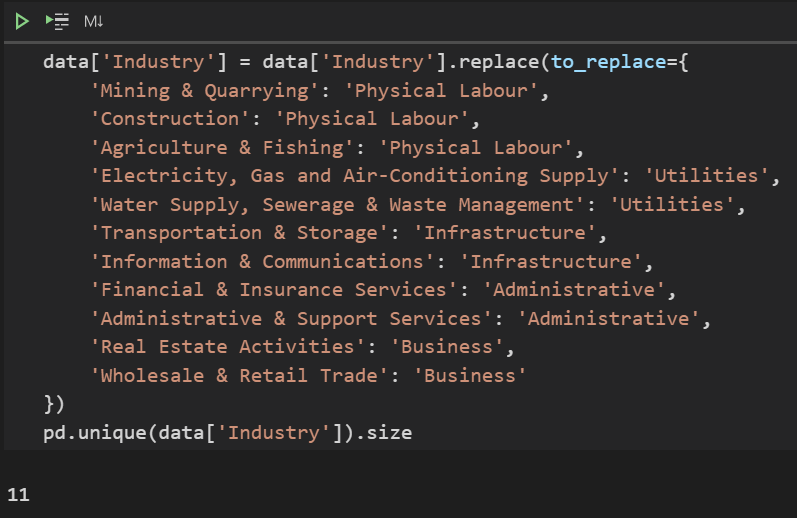
We will merge similar categories. For example, Electricity and Communications can be grouped as Utilities. To view the current possible values and their counts, we can use the *groupby* method, together with the *count* aggregate method to produce the following:



We aim to eliminate as many of the smaller categories as possible by merging them with larger ones. One possible merge here is

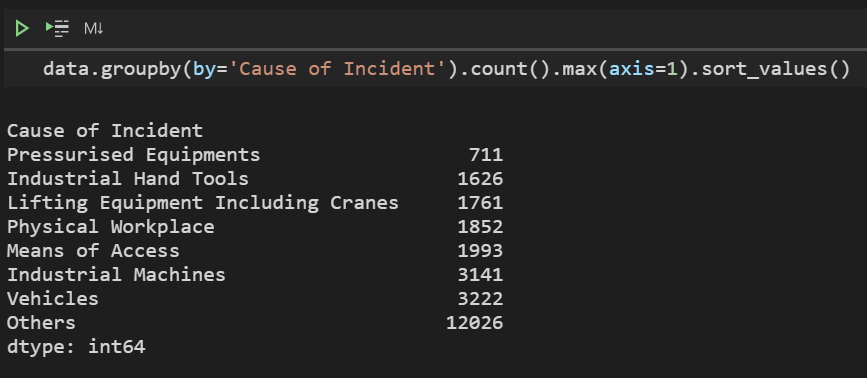
*Mining & Quarrying, Construction : Physical Labour*

Repeating the above process as many times as is logical and effective in reducing the number of possible values.



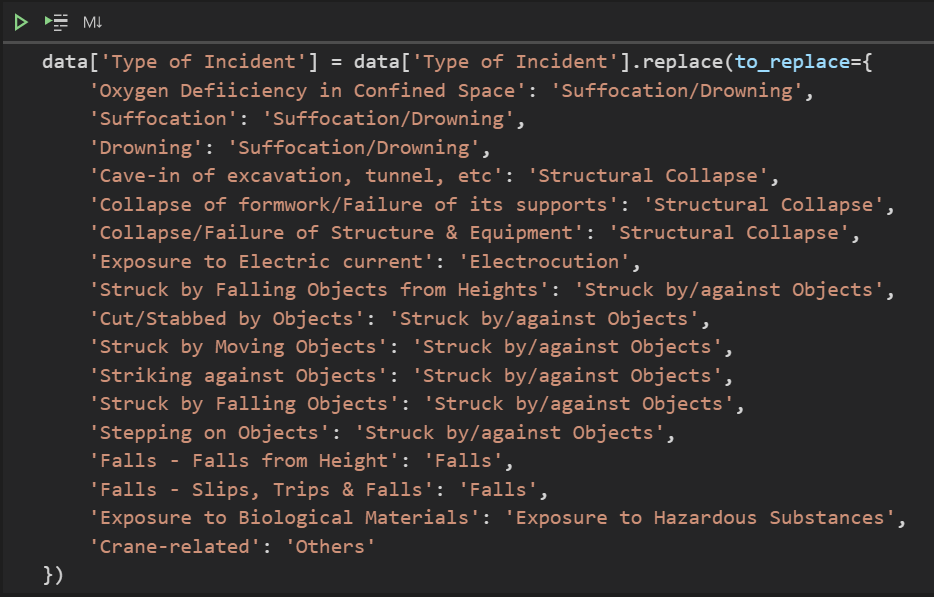
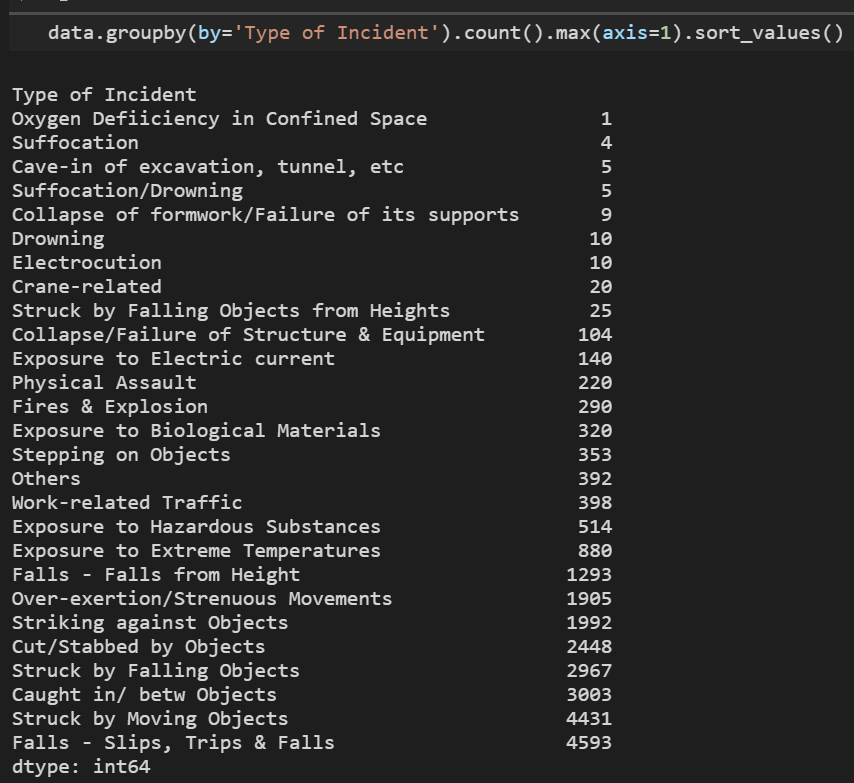
We successfully reduced the number of unique values for the *Industry* column from 17 to 11.

We'll apply the same logic to the *Cause of Incident* and *Type of Incident* columns.

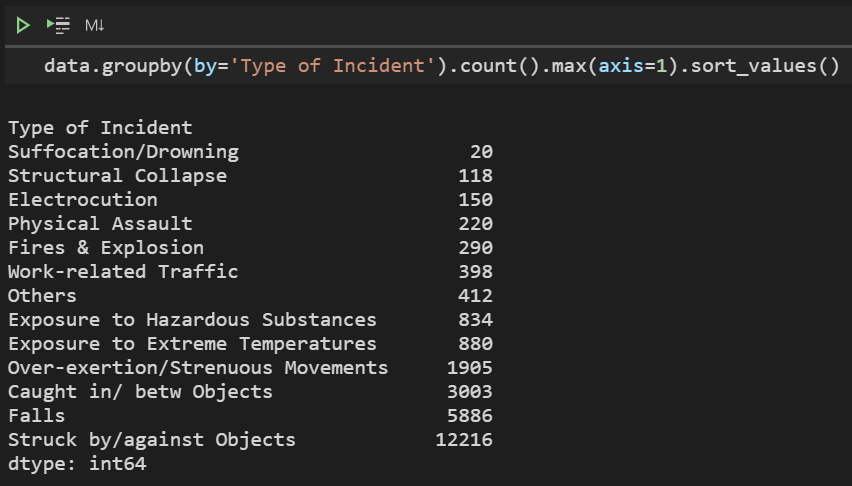


*Cause of Incident* only has 8 possible values, which is quite few already; so, we'll skip it for now.

Moving on to *Type of Incident*, we get a different result. We have 27 different values to handle.



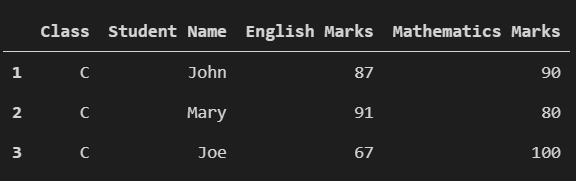
After processing, the number of unique values for the column *Type of Incident* from 27 to 13 has decreased to 13.



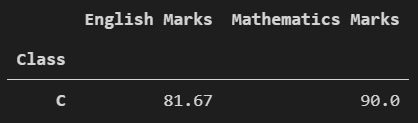
Now, *Industry, Cause of Incident* and *Type of Incident* have 11, 8 and 13 unique values, respectively. That's much more manageable than before.

1. *Disaggregation*

The data we're using is aggregated, as in they are not raw observations, but rather a collection of observations that have been somewhat summarised. Consider the following example:

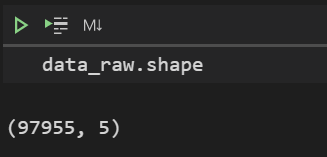
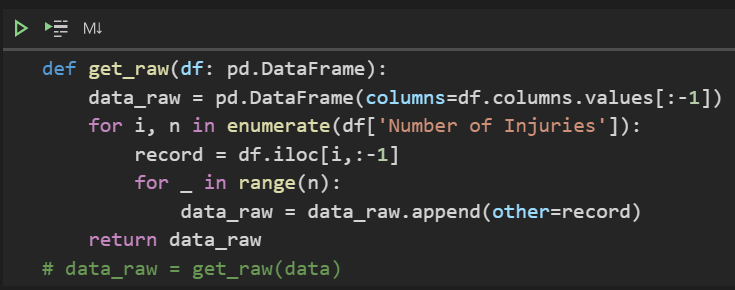


As John, Mary and Joe are from the same class, *C*, we can summarise their marks as follows:



We can aggregate the student data by *arithmetic mean*. This is a simplistic example, but it serves its purpose.

In our case, the data has been aggregated by count, as seen from *Number of Injuries*. Fortunately, it's quite simple to disaggregate our data. We will replicate as many records as there are *Number of Injuries* and remove the column (*Number of Injuries*). We will accomplish this using a simple for loop, iterating over the rows of our data.



There are many duplicate records now, which explains the sharp increase in the number of rows. That's our objective — to make the data raw.

1. *Encoding*

The machine learning algorithms in *sklearn do not recognize* textual attribute values; therefore, we have to encode the data (convert into numeric form). We have several options, such as Label Encoding and One-Hot Encoding. Label Encoding assigns each category a numeric (integral) value. For example,

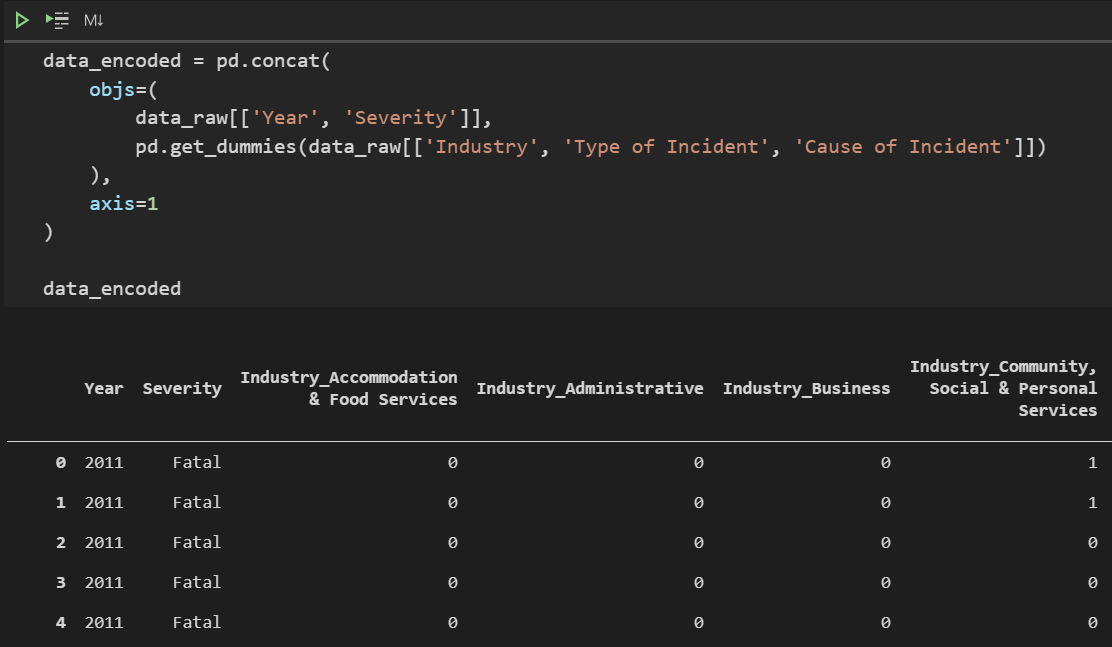
Gender

|  |  |
| --- | --- |
| Before Label Encoding | After Label Encoding |
| Male | 1 |
| Female | 2 |

However, as you might suspect, this does not make sense. Even though gender is qualitative (categorical), it is nominal and not ordinal (as in, the values cannot be logically sequenced). Male is not more valuable or more important than female, and vice versa.

One-Hot Encoding will be more suitable in this scenario. One-Hot Encoding makes use of dummy variables to represent the features binarily (1 for present / 0 for absent).

We can either make use of *pd.get\_dummies* or *sklearn.preprocessing.OneHotEncoder*. We'll go with *pd.get\_dummies*. Only the feature variables need to be encoded.



There are 34 columns in the encoded *DataFrame*. This number would be almost twice as large if we did not merge the values earlier.

## Data Partitioning

In this step, we'll split the data into a training and a test set. The training set will be used to select the best algorithm and hyper-parameters while the test set will be strictly used for final evaluation of our machine learning model at the end.



*train* will contain 75% of the data, while *test* will contain the remaining 25% of data. *random\_state* is specified merely for reproducibility of the experiment.

## Finding the Best Model

1. *Choosing an Algorithm*

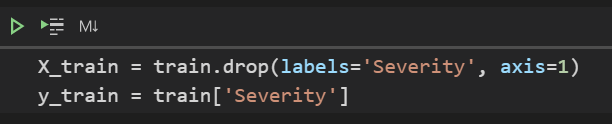
To keep this study simple, we will only consider the Support Vector Machine (or *SVM*) classification algorithm.

1. *Choosing the Best Hyper-Parameters*

We'll use the *GridSearchCV* function defined in *sklearn.model\_selection*. *GridSearchCV* basically automates Cross-Validation and Hyper-Parameter Comparisons for us.

Cross-Validation seems very abstract, but it's actually quite simple. It is essentially a for loop, training and testing a machine learning model iteratively, each time using a unique subset of the data for testing and leaving the rest for training the model.

First, we define the space of parameters in which we want *GridSearchCV* to explore. This space will contain a sample of key *SVC* possible hyper-parameters.



img…

This principle applies to both algorithm selection and selecting the best hyper-parameters. What are hyper-parameters?

Hyper-parameters are just "settings" of the machine learning algorithm which cannot be "learnt" from data. These require a more human touch, and, hence, have to be fine-tuned manually.

Well, by "manually", I mean that we can still do it programmatically. *GridSearchCV* will assist us in this regard.

…

It seems that … is the best set of hyper-parameters. Let's use them.

## Building the Model

We redefine a new model, called *model*. It is an instance of the … class, with … set to ….

## Training the Model

Then we fit the raw data to our new model.

## Scoring the Model

We call the method *predict* on *X\_test*.

## Evaluating the Model

Last, but not least, we evaluate our model.

# Discussions

Blah blah. blah.

# Conclusions

Blah blah blah.

##### Acknowledgment

Even though I am the sole author of this technical paper, I would like to acknowledge guidance from my Lecturer, Dr Peter Leong, as well as credit the following sources.

##### References

1. GovTech. (2019). *Workplace Injuries, Annual* [Online] Available at: <https://data.gov.sg/dataset/workplace-injuries-annual> [Accessed 29 May 2021]
2. Fatemeh Davoudi Kakhki, Steven A. Freeman, Gretchen A. Mosher. (2019). *Evaluating machine learning performance in predicting injury severity in agribusiness industries* [Online] Available at: <https://www.sciencedirect.com/science/article/pii/S092575351831107X> [Accessed 29 May 2021]
3. Zhang J.Y., Zi L.J., Hou Y.X., Deng D., Jiang W.T., Wang M.G. (2020). *A C-BiLSTM Approach to Classify Construction Accident Reports* [Online] Available at: <https://www.mdpi.com/2076-3417/10/17/5754/pdf> [Accessed 29 May 2021]