# Private Healthcare Administration Use Cases

## NUS-ISS Master of Technology (Intelligent Systems)


# Project Report



| Full Name | Student ID | Email |
|---|---|---|
| Mok Kay Yong | A0214617J | e0529481@u.nus.edu |
| Chong Keng Han | A0213547H | e0508648@u.nus.edu |
| Ly Duy Khang (Ken) | A0032571N | k.ly@u.nus.edu |
| Harry Chan | A0213530X | e0508631@u.nus.edu |

# 1. Executive Summary

Private healthcare organizations need to operate sustainably by making sure that it is operating on profit whilst providing their healthcare services *[1]*. They have to manage risks whilst from time to time providing treatments even to those patients without enough or proper insurance coverage (self-pay patients) *[2]* or for overseas patients, with or without insurance.

Some cases will turn into cases with write-off (either fully or partially unpaid) which can be due to various and many types of reasons. In use case #1 of this project, the team builds various machine learning models based on the technique that we have learned so far during this semester's course and then compares the models based on its ability to predict cases with significant write-off, accuracy, number of false positives and various other parameters. We also document our observation of those various models in this project documentation.

For use case #2, the team uses the same data set as use case #1 and transforms it into time-series weekly revenue data for the top 2 revenue contributing countries. The objective for use case #2 is to create a model to predict future weekly revenue numbers based on previous weeks revenue data. How many weeks of revenue data to be used for each model will depend on some statistical auto-correlation analysis. For this use case, results from simple multi-layer neural networks and ARIMA models are observed and compared.

Use case #3 is driven by the business need that In Singapore all inpatient cases are mandated by the Ministry of Health (MOH) to go through a financial counseling process to make sure patients or their next-of-kin are aware of the estimated cost before they proceed with the treatment and procedures *[3]*. In this use case, the team applies techniques including ensemble method learned as part of this semester's material. For the company where we take the data from, there is already an existing Hospital Bill Estimation model in production. Our team decided to try this use case to compare our model versus the current model in production as a comparison and potentially offer it as an alternative model to be considered.

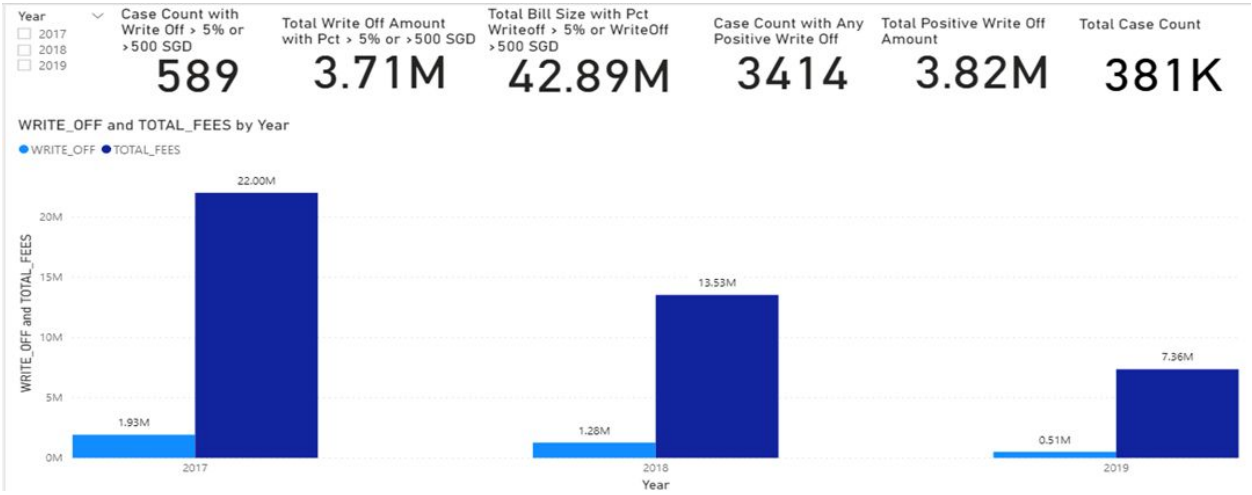
# 2. Business Problem Background and Objectives

In this section we will explain the business background for each use case.

## 2.1. Use Case #1: Prediction of Write-Off Cases

Some inpatient cases will involve write-off if full payment is not received. In this use case, we label significant write-off cases using the criteria if the write-off amount is more than 5% of the total hospital bill or if the amount of the write-off is more than 500 SGD.

Based on the data set, we can see the significant write-off cases better from the diagram below




As we can see from the above, the criteria used to label cases as having significant write-off is when the write-off amount is more than 5% of the hospital charges or when the write-off amount exceeds 500 SGD.

Based on the above criteria, there are 589 cases identified out of the total 3414 cases with positive amount write-off, within the total records of 381 thousand cases. The 589 cases represent about 0.15% of the total cases. The total write-off amount involved is about 3.71 million SGD spread over 3 years (from start of 2017 till end 2019). The total positive write-off amount is 3.82 million SGD, so it represents about 97% of total write-off amount for all cases with positive write-off. If we just take the annual average, it is about 1.23 million SGD per year.

Another observation is that the amount of write-off year by year is quite consistent which is in the range of 7-9% from the total hospital charges containing those significant write-off cases.

In the diagram we can see that the write-off cases are declining year on year from 2017 to 2019. It shows that operationally the Business Office department that handles patient billing has been doing a good job in controlling the collection process and minimizing risk cases with write-off using mechanisms like initial deposit, deposit top-up monitoring, requiring patients or patients next-of-kin to fill up credit card form as part of discharge process, etc for patients without sufficient insurance cover. Nevertheless if we see the amount, it is still quite substantial and it would be helpful for the company to be able to predict such occurrences and take the precautionary actions whenever possible, i.e. more thorough financial counselling, higher deposit rate, etc.

As mentioned previously, the number of cases with significant write-off is very minimal (about 0.15% from overall number of cases). This draws some similarity with fraud cases in financial transactions or eCommerce sites. So the objectives for this use case#1 are:

- Create a model with reasonable ability to detect positive cases and reasonable accuracy (we will elaborate more on what is “reasonable” in the observation part of this project report)
- Plan a simple implementation model with the expectation that it is feasible to make it into production to be used in real case

## 2.2. Use Case #2: Prediction of Inpatient Revenue based on Patient Country of Residence

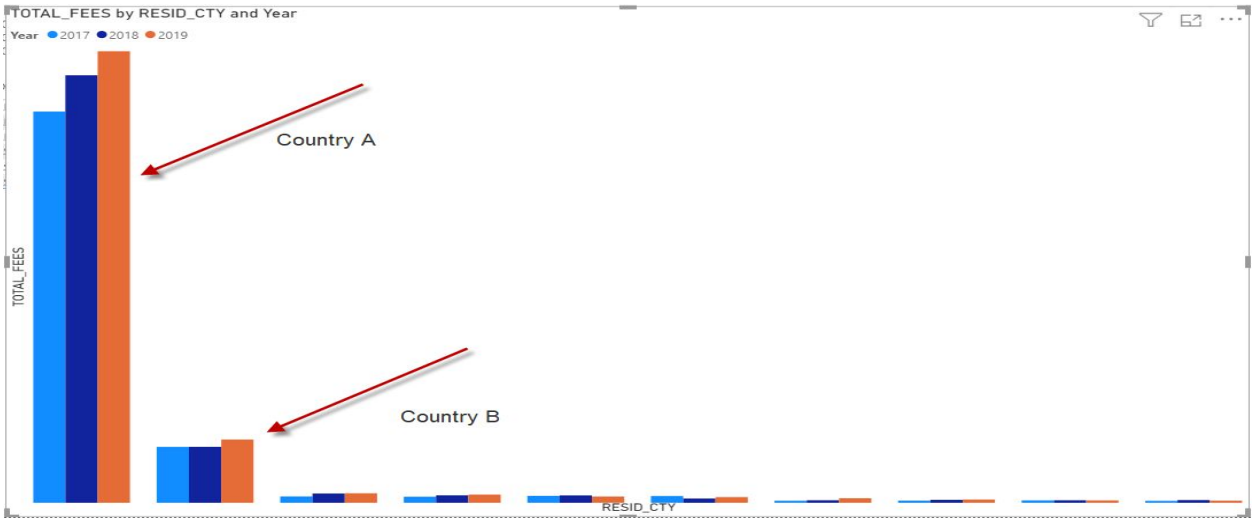
As a private hospital group with a number of experienced and well-known specialists and with international Joint Commission International (JCI) accreditation *[4] [5]*, the hospitals admit quite a significant number of foreign patients with relatively complex cases who prefer to get their treatment or procedure in Singapore instead of getting the treatment in their country of residence.

The Finance and Marketing department keeps track of the actual revenue and budget revenue of key countries to determine a set of strategic actions and resources assignment

in general. Due to the above reason, our team decided to do use case #2. The objectives of this use case are:

- Create a time-series prediction model for the top 2 countries as an example. For practicality, the time period that our team selected is on weekly period since the company management usually keep tracks of actual vs budget revenue on weekly basis
- Deploy the model with a simple front end as a prototype for company finance and marketing department so it can help on their weekly review and plan a set of proactive actions and resources allocation

For this use case#2, the team uses the same data set as use case #1 but massage the data into weekly time series revenue. As per the graph below, the team uses the 2 top country of residence (country A and country B) to build the production model with the expectation that the model is significant enough for the relevant departments in the company.

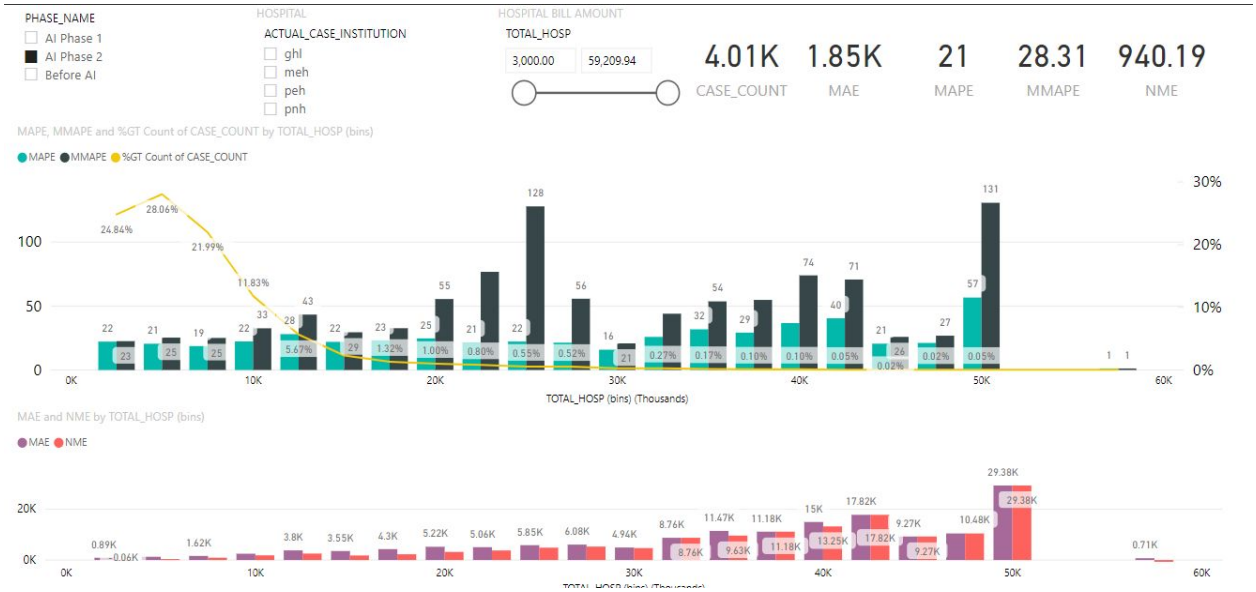



## 2.3. Use Case #3: Hospital Bill Estimation Upon Admission

As mentioned in the executive summary, the use case #3 is driven by statutory regulation and regulatory compliance to provide hospital bill estimation for patients upon their admission as part of the financial counseling process.

There is already an existing model in Production used by the company developed by an external vendor where the Intellectual Property of the AI model belongs to the vendor and it is a black-box for the company; the data rights remain with the company.

Below is the diagram showing accuracy measurement done for the current model which the project team will use as a benchmark later on.




The objectives for this use case #3 are:

- To explore alternative model and if the accuracy is comparable to the one built by vendor, it can be used as a ready deployable alternative
- To uncover more insights from the model and have more ability to explain the estimation result if it is possible as part of model characteristic/features
- To explore possibilities on cases for certain procedures or cases with some specific characteristics that have consistently low error levels. Business wise, potentially these cases can be set as price guarantee/fixed fee based on the model estimation, giving more assurance toward Patients and Payers or Insurers.

# 3. Data Set

The data set is taken from the hospital billing system that the company has. It comprises summarized billing data for 3 years from January 2017 till December 2019 (3 years of real data).

Data Set for Use Case #1 mainly consists of 3 main groups:

a. **Patient demographic data:** patient date of birth, citizenship, country of residence, race, religion, gender, marital status
b. **Admission data:** hospital code, case number (unique identifier of the patient episode/case), admission date, admitting doctor code, specialty, class/bed type, referral type that leads to the inpatient case
c. **Discharge data:** clinical codes (International Classification of Diseases/ICD Codes *[6]*, Table of Surgical Procedures/TOSP Codes *[7]*), payer codes, professional fee, hospital fee, write off amount, length of stay (LOS)

Data Set for Use Case #2 is derived from the same data as Use Case #1. From detail cases, the data gets summarised into weekly patient revenue based on patients' country of residence and formatted into a weekly time series data.

Data Set for Use Case #3 is quite similar with Data Set for Use Case #1 with some additional information:

a. **Estimated codes upon admission** (ICD and TOSP codes as advised by admitting specialists/doctor who is in charge/recommended for patients admission)
b. **Estimated Length of Stay** (LOS)
c. **Estimated HDU/ICU Stay**

# 4. Data Preparation / Pre-Processing

## 4.1 Exploratory Data Analysis

### 4.1.1 Check if data types are correct

The first task is to determine if there is a column which can be set as an index for the dataframe. This is done by comparing the number of unique values for each column versus the number of non-null values in the column. It is found that the ‘CASE_NUMBER’ has the highest number of unique values, but the number of unique values is 1 less than the number of non-null values. Further investigation on the ‘CASE_NUMBER’ that appeared twice, reveals that the row values are exactly the same, and therefore the duplicated row is dropped from the dataset

### 4.1.2 Check if data types are correct

In the dataset, some categorical columns are read as numbers. The columns are 'SPECIALTY_CODE', 'SPECIALTY_GRP', 'PACKAGE_CODE', 'PACKAGE_CODE1', 'PACKAGE_CODE2'. Therefore, these columns have to be explicitly defined as object type.

### 4.1.3 Check for null values

There are null values in the dataset, due to columns which are optional. For example, there are 5 ‘PAYER_CODE’ columns with their respective ‘PAYER_PAID_AMT’, and the ‘PAYER_CODE_1’ and ‘PAYER_PAID_AMT’ has to be filled, but the rest are optional. This is because each case number may have more than 1 payment source, for example, self pay and insurance payout.

Therefore, for categorical columns, the null values are set as empty strings, and for float numbers, the null values are set to 0.

### 4.1.4 Check for columns with only 1 value

There are columns which have only 1 value, and they are in the dataset for administrative purposes, such as printing invoices, or they are for future use.

The columns found are ‘CASE_TYPE’, ‘LANGUAGE’, ‘PACKAGE_CODE2’ and ‘PACKAGE_DESC2’.

### 4.1.5 Check for duplicate columns

In the dataset, there are categorical columns that are already represented by another column, and usually the other column is a description of the former. Using the following code, comparisons were made between categorical columns.

```
df.groupby(by=[column1])[column2'].nunique().max()
```

And the result is shown below.

```
PAYER_CODE_1 vs PAYER_NAME_1:  1
PAYER_CODE_2 vs PAYER_NAME_2:  1
PAYER_CODE_3 vs PAYER_NAME_3:  1
PAYER_CODE_4 vs PAYER_NAME_4:  1
PAYER_CODE_5 vs PAYER_NAME_5:  1
SPECIALTY_CODE vs SPECIALTY_DESC:  1
SPECIALTY_DESC vs SPECIALTY_GRP:  5
SPECIALTY_CODE vs SPECIALTY_GRP:  5
DISCHARGE_TYPE vs DISCHARGE_TYPE_DESC:  1
DOCTOR_CODE vs DOCTOR_NAME:  1
TOSP_DESC1, TOSP_DESC2, TOSP_DESC3, TOSP_DESC4 vs TOSP_STRING:  11
TOSP_CODE1, TOSP_CODE2, TOSP_CODE3, TOSP_CODE4 vs TOSP_STRING:  9
TOSP_CODE1 vs TOSP_DESC1:  1
TOSP_CODE2 vs TOSP_DESC2:  1
TOSP_CODE3 vs TOSP_DESC3:  1
TOSP_CODE4 vs TOSP_DESC4:  1
PATIENT_SID vs PATIENT_NUMBER:  1
DRG_CODE vs DRG_DESC:  2
PAYER_CODE1_V vs PAYER_NAME1_V:  1
PAYER_CODE2_V vs PAYER_NAME2_V:  1
PAYER_CODE3_V vs PAYER_NAME3_V:  1
PAYER_CODE4_V vs PAYER_NAME4_V:  1
PAYER_CODE_1 vs PAYER_CODE1_V:  3
PAYER_CODE_2 vs PAYER_CODE2_V:  4
PAYER_CODE_3 vs PAYER_CODE3_V:  2
PAYER_CODE_4 vs PAYER_CODE4_V:  1
PACKAGE_CODE vs PACKAGE_DESC:  2
PACKAGE_CODE1, PACKAGE_CODE2 vs PACKAGE_CODE:  2
PACKAGE_CODE1 vs PACKAGE_DESC1:  2
PACKAGE_CODE2 vs PACKAGE_DESC2:  1
ICD_CODE1, ICD_CODE2, ICD_CODE3 vs ICDCODE_STRING:  1
```

For the comparisons which yield 1, one of them can be dropped, as it is already represented by the other.

Further investigation is done on the suspected similar columns that have more than 1 unique value against each other, by looking at the actual unique values.

```
df.groupby(by=[column1])[column2].unique()
```

It was found that the differences are either missing space, spelling difference, punctuation difference or number that has decimal places, for example ('DRG_CODE', 'DRG_DESC'), ('PACKAGE_CODE', 'PACKAGE_DESC')

### 4.1.6 Investigate categorical columns with high number of unique values

Categorical columns with a high number of unique values will generate a large number of encoded columns.

It was found that the 'PATIENT_SID' and 'PATIENT_NUMBER' have 266574 unique values. After consulting with the domain expert, these 2 columns will be dropped, as they are IDs tagged to the patients.

The 'RESID_POSTALCODE' has 41533 unique values. We had an idea to convert 'RESID_POSTALCODE' with 'RESID_CTY' into geo position's latitude and longitude. However, due to time-constraint and after consultation with the domain expert, the 'RESID_POSTALCODE' is dropped, without generating the latitudes and longitudes.

### 4.1.7 Investigate categorical columns with potential misspelling

There are different representations of the same category found in 'NATIONALITY' and 'RESID_CTY' columns.

After consulting the domain expert, the 'NATIONALITY' is dropped, because the data is only accurate for local patients.

For 'RESID_CTY', a dictionary is used to map the values found in the dataset to the standard country country code.

## 4.2 Feature Engineering

### 4.2.1 Process dates into numbers

There are 3 dates in the dataset, 'ADMISSION_DTE', 'DISCHARGE_DTE' and ‘DOB’.

It was found that the difference between 'ADMISSION_DTE' and 'DISCHARGE_DTE' in days is the same as 'LOS_DAYS'.

As dates are difficult to be used in the models, 'ADMISSION_DTE' and ‘DOB’ is aggregated to ‘admission_age’.

Therefore, 'LOS_DAYS' and ‘admission_age’ are used in place of'ADMISSION_DTE', 'DISCHARGE_DTE' and ‘DOB’.

### 4.2.2 Normalisation of numeric columns

The standard scaler from scikit-learn is used on the numeric columns in the dataset. This is to ensure that the model training will not be biased towards columns or features that have high values.

The training portion of the dataset is used to fit the standard scaler, and the scaler is saved to pickle file for use during inference.

### 4.2.3 One hot encode categorical columns

For each categorical column in the dataset, the unique values are extracted for categories in the one hot encoder. The idea is to keep the categories fixed when more rows are added in future for training, where new categorical values, null values and empty string are treated as ‘not-one-of-these-categories’ or false in all columns generated from the encoder.

The training portion of the dataset is used to fit the one hot encoder, and the encoder is saved to pickle file for use during inference.

### 4.2.4 Binary categorical columns

There are categorical columns which have only 2 unique values. They are 'NONRESID_FLAG', 'DECEASED_FLAG' and 'VIP_FLAG'. These columns are converted to boolean dtypes, by mapping the 'Y' value to true and 'N' value to false.

### 4.2.5 Get feature importance

The number of columns after the dataset has been processed increases to 38885. Random forest regressor is used to further trim the number of columns. The random forest regressor is trained with the processed train dataset with the default parameters, and the feature importances from the trained random forest regressor is used to determine which columns to keep. The threshold is set at 0, that is the columns with feature importance above 0 are used. The feature importance numpy array is saved for inference.

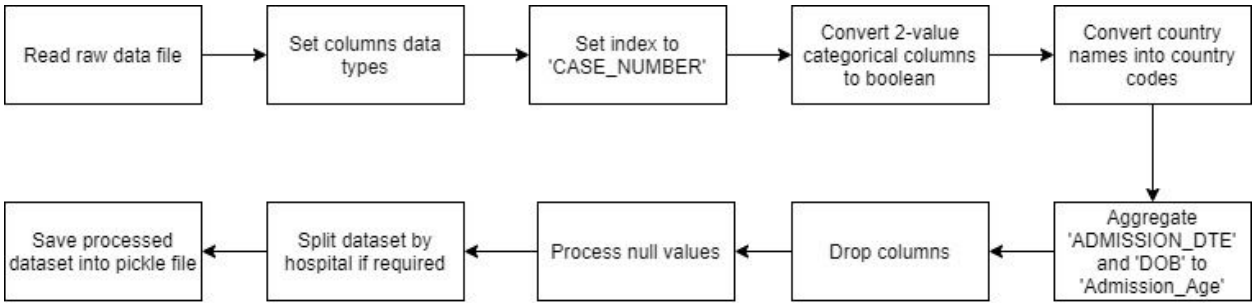
## 4.3 Data pipeline

A data pipeline is created in a python file for subsequent training and inferencing.

There are 3 functions in the data pipeline.

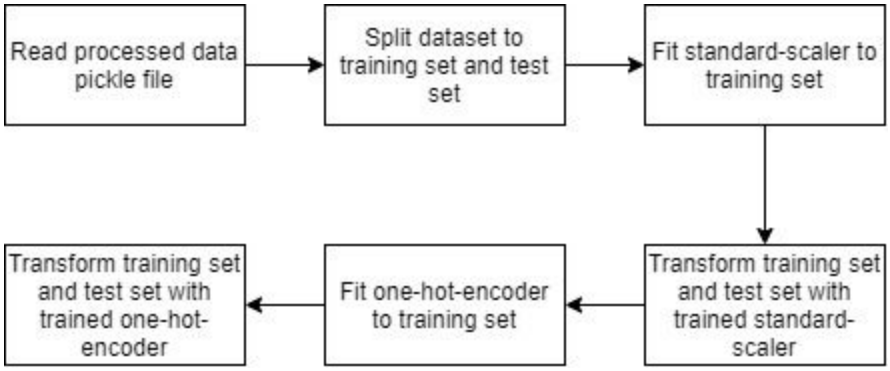
### 4.3.1 Process raw dataset

The first is to process the raw data based on the findings from the exploratory data analysis, and aggregate new features that are specified in the feature engineering.



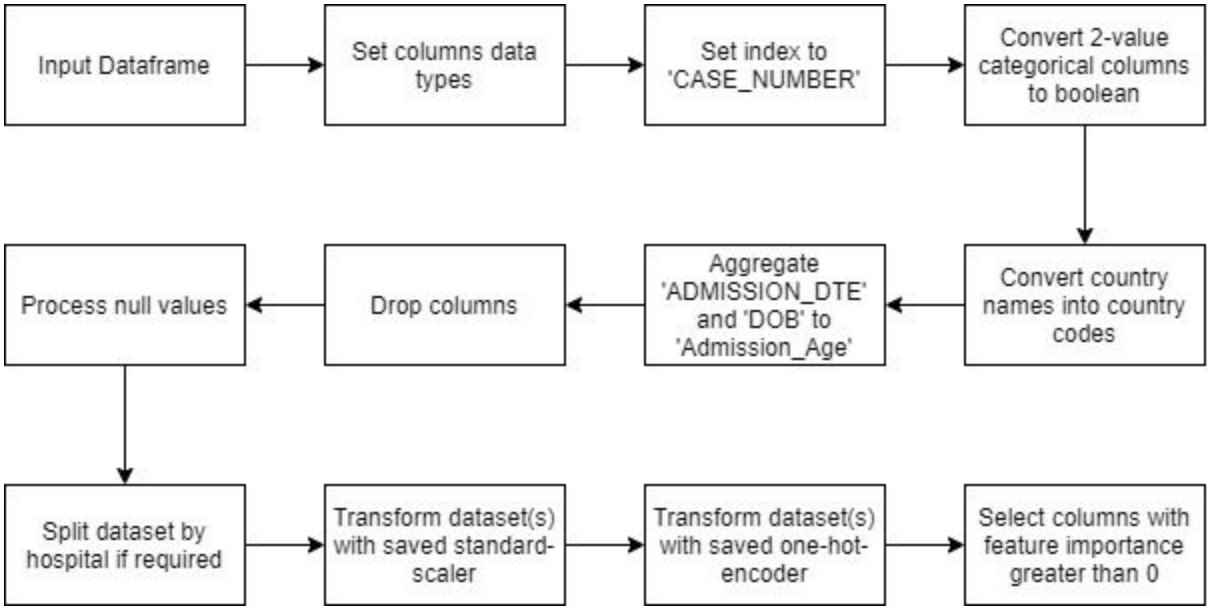
### 4.3.2 Transform train-test split from processed dataset

The second is to split the processed dataset into training set and test set, and fit standard-scaler and one hot encoder to the training set, then transform the training set and test set with the standard-scaler and encoder. The standard-scaler and the one-hot-encoder is saved to pickle file after they are fit with the training set, for inference.




### 4.3.3 Transform dataframe for inference

The third is for processing new data for inferencing, which has the whole pipeline from raw data processing and transform using the saved standard-scaler and one-hot-encoder.



## 4.4 Big dataset

### 4.4.1 Problem

The problem with processing a large dataset, which has 381210 rows and 57 columns, is the memory required for processing the dataset. Moreover, after one-hot-encode, the number of columns expanded to 38885. And during EDA and testing, the PC can run out of memory easily, and sometimes even causes BSOD (blue screen of death)

### 4.4.2 Dask

Initially Dask is used, as it has the capability of only compute on demand, which is usually at the end of a sequence of tasks, or in this case, the end of the data pipeline. However, the functionality of Dask is limited as it is not as straightforward as pandas to run Dask dataframe on scikit-learn models. Moreover, it is found that the end-computation requires large memory too.

### 4.4.3 Load-process-save using pickle

The final strategy that was used is to load-process-save. At every process, the dataset is loaded into memory, then processed or computed, then saved into pickle files. For example, the function to process raw data will load the dataset from the data file and then process, and finally saved into a pickle file. The same is also done for transforming the dataset into train set and test set, where the dataset is read from the pickle file, then split and processed, and finally saved into train pickle file and test pickle file.

### 4.4.4 Split into smaller dataset

Even with the final strategy to optimise PC resources used during data processing, the PC is not able to run some processes, especially when one-hot-encode transform and training the random forest regressor for feature importance. When the PC memory was upgraded to 64gb, it was able to one-hot-encode, but not able to train the random forest regressor for feature importance. Thus the dataset is split further by hospital for processing and training. However, the domain expert noted that it is not ideal as patients moving between hospitals are common, and most likely will not help in the model training.

#### 4.4.5 Cloud compute

Cloud compute is subsequently used for processing the data pipeline and model training. This is because the RAM memory and GPU memory allocated can be set before execution. The problem is that it is troublesome to debug. However, the speed of execution and the capability to handle huge memory requirements outweighs the debugging setback. Moreover, the large GPU memory is required for the training of the neural network models.

It was also found that breaking the task into smaller parallel jobs optimises the resources in the cloud, so that multiple CPU cores can be utilised. For example, breaking up the task of processing the dataset by hospitals, can be broken into jobs for each hospital.

# 5. Model Design, Observation and Comparison

## 5.1. Use Case #1: Prediction of Write-Off Cases

### 5.1.1. Model Design

As explained in the Section 2 on Business Problem, the number of significant write-off cases is very small as compared to the number of total transactions (only 589 records of interest out of 381 thousand records)

Based on the above fact and initial testing using basic model of some machine learning techniques, our observation shows the ability to detect the minority case is really low (standard model is able to detect only small number of cases in the test data, i.e. out of 196 cases in test data, only about 30 cases were detected).

Due to the above reason, the team decided to apply a technique called SMOTE (Synthetic Minority Oversampling Technique) taken from Machine Learning Master web article *[8]*. This technique is based on the computer science journal from Cornell University *[9]*.

Using SMOTE, the minority class in the training data set is artificially inflated/multiplied in quantity using SMOTE ratio. The maximum ratio is 1 where the minority class record number is made to be at the same number as the majority class. In our project, we experimented to use SMOTE ratio from 0.1, 0.2, 0.3 etc up to 1.0 (incremental of 0.1).

Below screen capture shows how the SMOTE inflates the training data from ratio 0.1 to 1.0. We can see the original minor classification 1 has only 393 records. With SMOTE ratio 0.1, the minority records for 1 are inflated to become 0.1 * majority class records = 0.1 * 253747 = 25374 records. The biggest ratio is 1.0 where the records for classification 1 are made to be the same number as the number of records for classification 0.

```
y_train unique values  {0: 253747, 1: 393}
y_train_unscaled_ss_01 unique values  {0: 253747, 1: 25374}
y_train_unscaled_ss_02 unique values  {0: 253747, 1: 50749}
y_train_unscaled_ss_03 unique values  {0: 253747, 1: 76124}
y_train_unscaled_ss_04 unique values  {0: 253747, 1: 101498}
y_train_unscaled_ss_05 unique values  {0: 253747, 1: 126873}
y_train_unscaled_ss_06 unique values  {0: 253747, 1: 152248}
y_train_unscaled_ss_07 unique values  {0: 253747, 1: 177622}
y_train_unscaled_ss_08 unique values  {0: 253747, 1: 202997}
y_train_unscaled_ss_09 unique values  {0: 253747, 1: 228372}
y_train_unscaled_ss_10 unique values  {0: 253747, 1: 253747}
```

Machine learning models tested for use case #1 are:

- Decision Tree
- Logistic Regression
- Naive Bayes
- Neural Network
- KNN (K Nearest Neighbour)

### 5.1.2. Observation and Comparison

#### 5.1.2.1. Decision Tree

##### 5.1.2.1.1. Decision Tree Without SMOTE

Without SMOTE, below is the result for the confusion matrix. The model can only recognize 36 out of 196 positive cases.

```
[[126639    235]
 [   160     36]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.13      0.18      0.15       196

    accuracy                           1.00    127070
   macro avg       0.57      0.59      0.58    127070
weighted avg       1.00      1.00      1.00    127070
```

5.1.2.1.2. Decision Tree With SMOTE

SMOTE ratio 0.1

```
Accuracy on training set: 1.000
Accuracy on test set: 0.995
[[126438    436]
 [   162     34]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.07      0.17      0.10       196

    accuracy                           1.00    127070
   macro avg       0.54      0.59      0.55    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.2

```
Accuracy on training set: 1.000
Accuracy on test set: 0.995
[[126425    449]
 [   157     39]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.08      0.20      0.11       196

    accuracy                           1.00    127070
   macro avg       0.54      0.60      0.56    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.3

```
Accuracy on training set: 1.000
Accuracy on test set: 0.995
[[126400    474]
 [   161     35]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.07      0.18      0.10       196

    accuracy                           1.00    127070
   macro avg       0.53      0.59      0.55    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.4

```
Accuracy on training set: 1.000
Accuracy on test set: 0.995
[[126445    429]
 [   163     33]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.07      0.17      0.10       196

    accuracy                           1.00    127070
   macro avg       0.54      0.58      0.55    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.5

```
Accuracy on training set: 1.000
Accuracy on test set: 0.995
[[126388    486]
 [   168     28]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.05      0.14      0.08       196

    accuracy                           0.99    127070
   macro avg       0.53      0.57      0.54    127070
weighted avg       1.00      0.99      1.00    127070
```

SMOTE ratio 0.6

```
Accuracy on training set: 1.000
Accuracy on test set: 0.995
[[126457    417]
 [   165     31]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.07      0.16      0.10       196

    accuracy                           1.00    127070
   macro avg       0.53      0.58      0.55    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.7

```
Accuracy on training set: 1.000
Accuracy on test set: 0.995
[[126446    428]
 [   161     35]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.08      0.18      0.11       196

    accuracy                           1.00    127070
   macro avg       0.54      0.59      0.55    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.8

```
Accuracy on training set: 1.000
Accuracy on test set: 0.995
[[126433    441]
 [   165     31]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.07      0.16      0.09       196

    accuracy                           1.00    127070
   macro avg       0.53      0.58      0.55    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.9

```
Accuracy on training set: 1.000
Accuracy on test set: 0.995
[[126418    456]
 [   168     28]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.06      0.14      0.08       196

    accuracy                           1.00    127070
   macro avg       0.53      0.57      0.54    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 1.0

```
Accuracy on training set: 1.000
Accuracy on test set: 0.995
[[126427    447]
 [   167     29]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.06      0.15      0.09       196

    accuracy                           1.00    127070
   macro avg       0.53      0.57      0.54    127070
weighted avg       1.00      1.00      1.00    127070
```

Based on the above observation, SMOTE doesn't really improve the capability of the model to identify the minority cases. The number of identified positive cases are hovering around 28-36 cases and the higher the SMOTE ratio doesn't improve the true positive detection as well.

We notice that with SMOTE, the number of false positive cases are increased but not much, less than 2 times as compared to the false positive cases for decision tree model without SMOTE.

#### 5.1.2.2. Logistic Regression

##### 5.1.2.2.1. Logistic Regression without SMOTE

Without SMOTE, the model can only detect 3 true positive cases out of 196 positive cases.

```
Training set score: 0.998
Test set score: 0.998
[[126872      2]
 [   193      3]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.60      0.02      0.03       196

    accuracy                           1.00    127070
   macro avg       0.80      0.51      0.51    127070
weighted avg       1.00      1.00      1.00    127070
```

5.1.2.2.2. Logistic Regression with SMOTE

SMOTE ratio 0.1

```
Training set score: 0.933
Test set score: 0.988
[[125541   1333]
 [   134     62]]
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    126874
           1       0.04      0.32      0.08       196

    accuracy                           0.99    127070
   macro avg       0.52      0.65      0.54    127070
weighted avg       1.00      0.99      0.99    127070
```

SMOTE ratio 0.2

```
Training set score: 0.888
Test set score: 0.977
[[124063   2811]
 [   117     79]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    126874
           1       0.03      0.40      0.05       196

    accuracy                           0.98    127070
   macro avg       0.51      0.69      0.52    127070
weighted avg       1.00      0.98      0.99    127070
```

SMOTE ratio 0.3

```
Training set score: 0.856
Test set score: 0.963
[[122267   4607]
 [   101     95]]
              precision    recall  f1-score   support

           0       1.00      0.96      0.98    126874
           1       0.02      0.48      0.04       196

    accuracy                           0.96    127070
   macro avg       0.51      0.72      0.51    127070
weighted avg       1.00      0.96      0.98    127070
```

SMOTE ratio 0.4

```
Training set score: 0.831
Test set score: 0.948
[[120416   6458]
 [    91    105]]
              precision    recall  f1-score   support

           0       1.00      0.95      0.97    126874
           1       0.02      0.54      0.03       196

    accuracy                           0.95    127070
   macro avg       0.51      0.74      0.50    127070
weighted avg       1.00      0.95      0.97    127070
```

SMOTE ratio 0.5

```
Training set score: 0.810
Test set score: 0.932
[[118326   8548]
 [    84    112]]
              precision    recall  f1-score   support

           0       1.00      0.93      0.96    126874
           1       0.01      0.57      0.03       196

    accuracy                           0.93    127070
   macro avg       0.51      0.75      0.50    127070
weighted avg       1.00      0.93      0.96    127070
```

SMOTE ratio 0.6

```
Training set score: 0.795
Test set score: 0.915
[[116167  10707]
 [    71    125]]
              precision    recall  f1-score   support

           0       1.00      0.92      0.96    126874
           1       0.01      0.64      0.02       196

    accuracy                           0.92    127070
   macro avg       0.51      0.78      0.49    127070
weighted avg       1.00      0.92      0.95    127070
```

SMOTE ratio 0.7

```
Training set score: 0.787
Test set score: 0.897
[[113840  13034]
 [    63    133]]
              precision    recall  f1-score   support

           0       1.00      0.90      0.95    126874
           1       0.01      0.68      0.02       196

    accuracy                           0.90    127070
   macro avg       0.50      0.79      0.48    127070
weighted avg       1.00      0.90      0.94    127070
```

SMOTE ratio 0.8

```
Training set score: 0.780
Test set score: 0.878
[[111423  15451]
 [    56    140]]
              precision    recall  f1-score   support

           0       1.00      0.88      0.93    126874
           1       0.01      0.71      0.02       196

    accuracy                           0.88    127070
   macro avg       0.50      0.80      0.48    127070
weighted avg       1.00      0.88      0.93    127070
```

SMOTE ratio 0.9

```
Training set score: 0.778
Test set score: 0.859
[[108978  17896]
 [    52    144]]
              precision    recall  f1-score   support

           0       1.00      0.86      0.92    126874
           1       0.01      0.73      0.02       196

    accuracy                           0.86    127070
   macro avg       0.50      0.80      0.47    127070
weighted avg       1.00      0.86      0.92    127070
```

SMOTE ratio 1.0

```
Training set score: 0.779
Test set score: 0.839
[[106529  20345]
 [    52    144]]
              precision    recall  f1-score   support

           0       1.00      0.84      0.91    126874
           1       0.01      0.73      0.01       196

    accuracy                           0.84    127070
   macro avg       0.50      0.79      0.46    127070
weighted avg       1.00      0.84      0.91    127070
```

We can see from the above observation that as we increase the SMOTE ratio, the ability of the logistic regression model to detect positive cases increases. The number of false positive cases also increases along with the SMOTE ratio. However for this use case, the importance is on the model's ability to detect positive cases. There are business rules that we can deploy to ignore the false positives, for example:

- If the patients have valid insurance
- If the patients can provide enough deposit
- If the patients have guarantee letter from company or any type of valid guarantor

5.1.2.3. Naive Bayes

5.1.2.3.1. Naive Bayes without SMOTE

```
Training set score: 0.948
Test set score: 0.948
[[120337   6537]
 [   106     90]]
              precision    recall  f1-score   support

           0       1.00      0.95      0.97    126874
           1       0.01      0.46      0.03       196

    accuracy                           0.95    127070
   macro avg       0.51      0.70      0.50    127070
weighted avg       1.00      0.95      0.97    127070
```

We can see even without SMOTE, Naive Bayes model can detect 90 cases out of the 196 positive cases

5.1.2.3.2. Naive Bayes with SMOTE

SMOTE ratio 0.1

```
Training set score: 0.897
Test set score: 0.935
[[118664   8210]
 [    91    105]]
              precision    recall  f1-score   support

           0       1.00      0.94      0.97    126874
           1       0.01      0.54      0.02       196

    accuracy                           0.93    127070
   macro avg       0.51      0.74      0.50    127070
weighted avg       1.00      0.93      0.96    127070
```

SMOTE ratio 0.2

```
Training set score: 0.865
Test set score: 0.931
[[118251   8623]
 [    89    107]]
              precision    recall  f1-score   support

           0       1.00      0.93      0.96    126874
           1       0.01      0.55      0.02       196

    accuracy                           0.93    127070
   macro avg       0.51      0.74      0.49    127070
weighted avg       1.00      0.93      0.96    127070
```

SMOTE ratio 0.3

```
Training set score: 0.838
Test set score: 0.930
[[118055   8819]
 [    88    108]]
              precision    recall  f1-score   support

           0       1.00      0.93      0.96    126874
           1       0.01      0.55      0.02       196

    accuracy                           0.93    127070
   macro avg       0.51      0.74      0.49    127070
weighted avg       1.00      0.93      0.96    127070
```

SMOTE ratio 0.4

```
Training set score: 0.816
Test set score: 0.928
[[117823   9051]
 [    87    109]]
              precision    recall  f1-score   support

           0       1.00      0.93      0.96    126874
           1       0.01      0.56      0.02       196

    accuracy                           0.93    127070
   macro avg       0.51      0.74      0.49    127070
weighted avg       1.00      0.93      0.96    127070
```

SMOTE ratio 0.5

```
Training set score: 0.797
Test set score: 0.926
[[117602   9272]
 [    86    110]]
              precision    recall  f1-score   support

           0       1.00      0.93      0.96    126874
           1       0.01      0.56      0.02       196

    accuracy                           0.93    127070
   macro avg       0.51      0.74      0.49    127070
weighted avg       1.00      0.93      0.96    127070
```

SMOTE ratio 0.6

```
Training set score: 0.780
Test set score: 0.925
[[117389   9485]
 [    86    110]]
              precision    recall  f1-score   support

           0       1.00      0.93      0.96    126874
           1       0.01      0.56      0.02       196

    accuracy                           0.92    127070
   macro avg       0.51      0.74      0.49    127070
weighted avg       1.00      0.92      0.96    127070
```

SMOTE ratio 0.7

```
Training set score: 0.766
Test set score: 0.924
[[117247   9627]
 [    86    110]]
              precision    recall  f1-score   support

           0       1.00      0.92      0.96    126874
           1       0.01      0.56      0.02       196

    accuracy                           0.92    127070
   macro avg       0.51      0.74      0.49    127070
weighted avg       1.00      0.92      0.96    127070
```

SMOTE ratio 0.8

```
Training set score: 0.753
Test set score: 0.922
[[117032   9842]
 [    86    110]]
              precision    recall  f1-score   support

           0       1.00      0.92      0.96    126874
           1       0.01      0.56      0.02       196

    accuracy                           0.92    127070
   macro avg       0.51      0.74      0.49    127070
weighted avg       1.00      0.92      0.96    127070
```

SMOTE ratio 0.9

```
Training set score: 0.743
Test set score: 0.921
[[116878   9996]
 [    86    110]]
              precision    recall  f1-score   support

           0       1.00      0.92      0.96    126874
           1       0.01      0.56      0.02       196

    accuracy                           0.92    127070
   macro avg       0.51      0.74      0.49    127070
weighted avg       1.00      0.92      0.96    127070
```

SMOTE ratio 1.0

```
Training set score: 0.733
Test set score: 0.920
[[116772  10102]
 [    86    110]]
              precision    recall  f1-score   support

           0       1.00      0.92      0.96    126874
           1       0.01      0.56      0.02       196

    accuracy                           0.92    127070
   macro avg       0.51      0.74      0.49    127070
weighted avg       1.00      0.92      0.96    127070
```

As we can see from above, the ability of the Naive Bayes model to detect positive cases increases as we increase the SMOTE ratio up to a certain level i.e. 0.5 as per the above observation. If we increase the ratio to a higher level, the ability to detect is not increased; we only observe that there are increases in false positive cases which is making the model inefficient.

5.1.2.4. Neural Network

5.1.2.4.1. Neural Network without SMOTE

*5.1.2.4.1.1. Two Layers MLPClassifier(16,16)*

```
Out[71]: MLPClassifier(hidden_layer_sizes=(16, 16), max_iter=1000, verbose=2)
```

```
In [72]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 0.999
Test set score: 0.998
Accuracy:  0.9983473675926655
[[126832     42]
 [   168     28]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.40      0.14      0.21       196

    accuracy                           1.00    127070
   macro avg       0.70      0.57      0.60    127070
weighted avg       1.00      1.00      1.00    127070
```

*5.1.2.4.1.2. Two Layers MLPClassifer(32,32)*

```
Out[73]: MLPClassifier(hidden_layer_sizes=(32, 32), max_iter=1000, verbose=2)
```

```
In [74]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 0.999
Test set score: 0.998
Accuracy:  0.9980483198237192
[[126798     76]
 [   172     24]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.24      0.12      0.16       196

    accuracy                           1.00    127070
   macro avg       0.62      0.56      0.58    127070
weighted avg       1.00      1.00      1.00    127070
```

*5.1.2.4.1.3. Two Layers MLPClassifier(64,64)*

```
Out[75]: MLPClassifier(hidden_layer_sizes=(64, 64), max_iter=1000, verbose=2)
```

```
In [76]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.9982214527425828
[[126824     50]
 [   176     20]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.29      0.10      0.15       196

    accuracy                           1.00    127070
   macro avg       0.64      0.55      0.57    127070
weighted avg       1.00      1.00      1.00    127070
```

*5.1.2.4.1.4. Two Layers MLPClassifier(128,128)*

```
Out[77]: MLPClassifier(hidden_layer_sizes=(128, 128), max_iter=1000, verbose=2)
```

```
In [78]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.9982214527425828
[[126815     59]
 [   167     29]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.33      0.15      0.20       196

    accuracy                           1.00    127070
   macro avg       0.66      0.57      0.60    127070
weighted avg       1.00      1.00      1.00    127070
```

*5.1.2.4.1.5. Two Layers MLPClassifier(256,256)*

```
Out[79]: MLPClassifier(hidden_layer_sizes=(256, 256), max_iter=1000, verbose=2)
```

```
In [80]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.9981270166050209
[[126797     77]
 [   161     35]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.31      0.18      0.23       196

    accuracy                           1.00    127070
   macro avg       0.66      0.59      0.61    127070
weighted avg       1.00      1.00      1.00    127070
```

Based on the above result, as we add in the number of nodes in each layer, the ability of the NN model to detect positive cases increases. We also observe that for NN, it can maintain a low number of false positive cases.

*5.1.2.4.1.6. Three Layers MLPClassifier(16,16,16)*

```
Out[81]: MLPClassifier(hidden_layer_sizes=(16, 16, 16), max_iter=1000, verbose=2)
```

```
In [82]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 0.999
Test set score: 0.998
Accuracy:  0.9984339340520972
[[126850     24]
 [   175     21]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.47      0.11      0.17       196

    accuracy                           1.00    127070
   macro avg       0.73      0.55      0.59    127070
weighted avg       1.00      1.00      1.00    127070
```

*5.1.2.4.1.7. Three Layers MLPClassifier(32,32,32)*

Out[83]:
```
MLPClassifier(hidden_layer_sizes=(32, 32, 32), max_iter=1000, verbose=2)
```

In [84]:
```
predictions = mlp.predict(X_test_scaled)

# Score
print("Training set score: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.9979302746517668
[[126789     85]
 [   178     18]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.17      0.09      0.12       196

    accuracy                           1.00    127070
   macro avg       0.59      0.55      0.56    127070
weighted avg       1.00      1.00      1.00    127070
```

*5.1.2.4.1.8. Three Layers MLPClassifier(64,64,64)*

```
Out[85]: MLPClassifier(hidden_layer_sizes=(64, 64, 64), max_iter=1000, verbose=2)
```

```
In [86]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.9980483198237192
[[126805     69]
 [   179     17]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.20      0.09      0.12       196

    accuracy                           1.00    127070
   macro avg       0.60      0.54      0.56    127070
weighted avg       1.00      1.00      1.00    127070
```

*5.1.2.4.1.8. Four Layers MLPClassifier(16,16,16,16)*

```
Out[87]: MLPClassifier(hidden_layer_sizes=(16, 16, 16, 16), max_iter=1000, verbose=2)
```

```
In [88]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 0.999
Test set score: 0.998
Accuracy:  0.9980640591799795
[[126803     71]
 [   175     21]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.23      0.11      0.15       196

    accuracy                           1.00    127070
   macro avg       0.61      0.55      0.57    127070
weighted avg       1.00      1.00      1.00    127070
```

*5.1.2.4.1.8. Four Layers MLPClassifier(32,32,32,32)*

```
Out[89]: MLPClassifier(hidden_layer_sizes=(32, 32, 32, 32), max_iter=1000, verbose=2)
```

```
In [90]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.997796490123554
[[126764    110]
 [   170     26]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.19      0.13      0.16       196

    accuracy                           1.00    127070
   macro avg       0.59      0.57      0.58    127070
weighted avg       1.00      1.00      1.00    127070
```

We tested the impact if we increase the number of layers but as shown in the above, the ability of the model to detect positive cases have not improved

5.1.2.4.2. Neural Network with SMOTE

*5.1.2.4.2.1. Two Layers MLPClassifier(128,128)*

Note: for MLPClassifer 2 layers, the (128,128) is chosen because in the observation without SMOTE, this configuration has the best ability to detect positive cases (29 out of 196 cases)

Without SMOTE

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.9982214527425828
[[126815     59]
 [   167     29]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.33      0.15      0.20       196

    accuracy                           1.00    127070
   macro avg       0.66      0.57      0.60    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.1

Out[91]:
```
MLPClassifier(hidden_layer_sizes=(128, 128), max_iter=1000, verbose=2)
```

In [92]:
```
predictions = mlp.predict(X_test_scaled)

# Score
print("Training set score: {:.3f}".format(mlp.score(X_train_scaled_ss_01, y_train_scaled_ss_01)))
print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.9976548359172109
[[126745    129]
 [   169     27]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.17      0.14      0.15       196

    accuracy                           1.00    127070
   macro avg       0.59      0.57      0.58    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.2

```
Out[93]: MLPClassifier(hidden_layer_sizes=(128, 128), max_iter=1000, verbose=2)
```

```
In [94]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled_ss_02, y_train_scaled_ss_02)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.99760761784843
[[126740    134]
 [   170     26]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.16      0.13      0.15       196

    accuracy                           1.00    127070
   macro avg       0.58      0.57      0.57    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.3

```
Out[95]: MLPClassifier(hidden_layer_sizes=(128, 128), max_iter=1000, verbose=2)
```

```
In [96]: predictions = mlp.predict(X_test_scaled)

         # Score
         print("Training set score: {:.3f}".format(mlp.score(X_train_scaled_ss_03, y_train_scaled_ss_03)))
         print("Test set score: {:.3f}".format(mlp.score(X_test_scaled, y_test)))

         print("Accuracy: ", metrics.accuracy_score(y_test, predictions))
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))
```

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.9976627055953412
[[126754    120]
 [   177     19]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.14      0.10      0.11       196

    accuracy                           1.00    127070
   macro avg       0.57      0.55      0.56    127070
```

From the above observation, we can see that the same configuration of MLPClassifier has not improved the ability to detect positive cases even with SMOTE; it only increases the number of false positive cases.

*5.1.2.4.2.2. Three Layers MLPClassifier(16,16,16)*

For the three layers MLPClassifier, this configuration is chosen because without SMOTE, it gave the best result (21 out of 196 positive cases in test data)

Without SMOTE

```
Training set score: 0.999
Test set score: 0.998
Accuracy:  0.9984339340520972
[[126850     24]
 [   175     21]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.47      0.11      0.17       196

    accuracy                           1.00    127070
   macro avg       0.73      0.55      0.59    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio 0.1

```
Training set score: 0.992
Test set score: 0.990
Accuracy:  0.990044857165342
[[125762   1112]
 [   153     43]]
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    126874
           1       0.04      0.22      0.06       196

    accuracy                           0.99    127070
   macro avg       0.52      0.61      0.53    127070
weighted avg       1.00      0.99      0.99    127070
```

SMOTE ratio 0.2

```
Training set score: 0.991
Test set score: 0.988
Accuracy:  0.9876288659793815
[[125454   1420]
 [   152     44]]
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    126874
           1       0.03      0.22      0.05       196

    accuracy                           0.99    127070
   macro avg       0.51      0.61      0.52    127070
weighted avg       1.00      0.99      0.99    127070
```

With the three layers or more layers of MLPClassifier, the increase of SMOTE ratio seems to increase the number of positive detection as well as false positives, but overall detection ability is still too low compared to others (non MLPClassifier).

*5.1.2.4.2.3. Four Layers MLPClassifier(32,32,32,32)*

We also notice that unlike other models where there is a trend of increase when the SMOTE ratio is increased before it gets stagnant. In MLPClassifier, the behaviour is less predictable.

For example with the above MLPClassifier contribution, we tested with increasing SMOTE ratio as below

Without SMOTE as below:

```
Training set score: 1.000
Test set score: 0.998
Accuracy:  0.997796490123554
[[126764    110]
 [   170     26]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.19      0.13      0.16       196

    accuracy                           1.00    127070
   macro avg       0.59      0.57      0.58    127070
weighted avg       1.00      1.00      1.00    127070
```

With SMOTE ratio = 0.1 (the detection of true positive increases, along with increase in the number of false positives) as below

```
Training set score: 0.999
Test set score: 0.995
Accuracy:  0.9951837569843394
[[126420    454]
 [   158     38]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.08      0.19      0.11       196

    accuracy                           1.00    127070
   macro avg       0.54      0.60      0.55    127070
weighted avg       1.00      1.00      1.00    127070
```

With SMOTE ratio = 0.2 (the detection of true positive is even lower than SMOTE ratio 0.1)

```
Training set score: 0.999
Test set score: 0.995
Accuracy:  0.9954828047532855
[[126464    410]
 [   164     32]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.07      0.16      0.10       196

    accuracy                           1.00    127070
   macro avg       0.54      0.58      0.55    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio = 0.5 (note: the increase in the true positive detection is not much or not meaningful as compared to SMOTE ratio = 0.1 for example, which is 38 true positives) as below:

```
Training set score: 1.000
Test set score: 0.996
Accuracy:  0.9955772408908475
[[126468    406]
 [   156     40]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.09      0.20      0.12       196

    accuracy                           1.00    127070
   macro avg       0.54      0.60      0.56    127070
weighted avg       1.00      1.00      1.00    127070
```

SMOTE ratio = 1.0 (the number of true positive cases detected is even lower as compared to SMOTE ratio = 0.1 which is 38 true positive cases. However we can see that the false positive cases are very stable from SMOTE ratio = 0.1 to SMOTE ratio = 1.0 which is less than 500 false positive cases)

```
Training set score: 0.999
Test set score: 0.995
Accuracy:  0.9949555363185646
[[126394    480]
 [   161     35]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.07      0.18      0.10       196

    accuracy                           0.99    127070
   macro avg       0.53      0.59      0.55    127070
weighted avg       1.00      0.99      1.00    127070
```

5.1.2.5. KNN

5.1.2.5.1. KNN without SMOTE

The model can detect only 11 true positive cases (out of total 196 true positive cases) in the test data

```
Training set score: 0.999
Test set score: 0.998
Accuracy:  0.9984968914771386
[[126868      6]
 [   185     11]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    126874
           1       0.65      0.06      0.10       196

    accuracy                           1.00    127070
   macro avg       0.82      0.53      0.55    127070
weighted avg       1.00      1.00      1.00    127070
```

5.1.2.5.2. KNN with SMOTE

SMOTE ratio 0.1 result is as below, we can see the ability to detect true positive cases jump straight away to 65 (out of 196 true positive cases).

```
Training set score: 0.991
Test set score: 0.985
Accuracy:  0.9850869599433383
[[125110   1764]
 [   131     65]]
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    126874
           1       0.04      0.33      0.06       196

    accuracy                           0.99    127070
   macro avg       0.52      0.66      0.53    127070
weighted avg       1.00      0.99      0.99    127070
```

SMOTE ratio 0.5 as below. We can see that the increase is not as significant

```
Training set score: 0.991
Test set score: 0.980
Accuracy:  0.9802156291807665
[[124485   2389]
 [   125     71]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    126874
           1       0.03      0.36      0.05       196

    accuracy                           0.98    127070
   macro avg       0.51      0.67      0.52    127070
weighted avg       1.00      0.98      0.99    127070
```

SMOTE ratio 1.0 as below. We can see that the ability to detect true positive cases have not improved; it is stagnant from a certain level of SMOTE ratio. However the number of false positive cases are also stable, about 2400 cases.

```
Training set score: 0.993
Test set score: 0.980
Accuracy:  0.9798536239867789
[[124439   2435]
 [   125     71]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    126874
           1       0.03      0.36      0.05       196

    accuracy                           0.98    127070
   macro avg       0.51      0.67      0.52    127070
weighted avg       1.00      0.98      0.99    127070
```

5.1.2.5. More Business Context on Use Case #1

For use case #1, we would like to reiterate that the emphasis is the model's ability to detect true positive cases because true positive cases are very rare cases (589 cases in total out of 381K cases).

The higher the ability to detect the true positive cases, it will be better for the business so write-off cases can be avoided by applying some mitigation actions i.e. applying more thorough financial counseling or enforcing measures such as taking in more deposits or some kind of payment guarantee (i.e. credit card amount blocking for example).

On the false positive impact, some simple rules can be applied along with the model deployment such that majority of false positive predictions can be ignored straight away i.e. simple rule such as if the case has valid insurance, valid guarantee letter from a company, deposit requested has been received, etc.

Based on the insight we can get from the data, almost all of the write-off cases involved self-pay (non-insurance) cases. However, the number of self-pay cases comprises of about 200K cases which is more than half of the total data population of 380K for the 3 years of data. So the model cannot just anyhow shoot that any self-pay cases will turn to write-off.

Based on our observation. The best model to predict the most number of true positive cases is from the Logistics Regression using SMOTE ratio of 0.9. It can detect 144 true positive cases out of 196 total true positive cases in the test data (73.47% Sensitivity level). Whilst the number of false positive cases is 17,896 cases, which is about 8.94% compared to the whole number of self pay cases (200K cases). Again we can apply simple business rules to ignore the false positive signals to be combined alongside the ML model.

## 5.2. Use Case #2: Prediction of Inpatient Revenue based on Patient Country of Residence

### 5.2.1. Country A

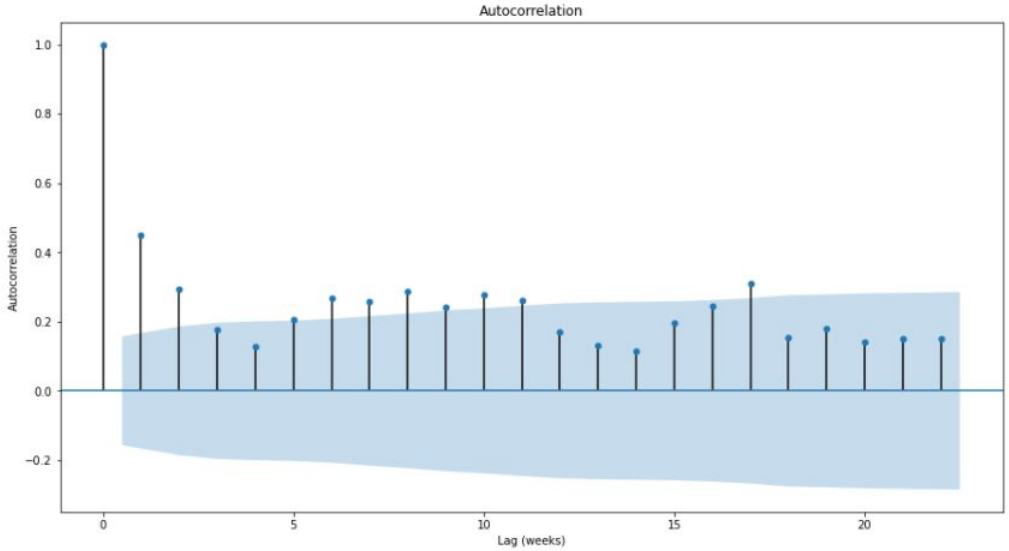
#### 5.2.1.1. Model Design

The initial model for the revenue prediction is taken from a github model created by Lisa Ong , a lecturer in NUS ISS [10]. This model demonstrates that a simple neural net model with 2 dense layers (1 hidden layer with 16 nodes and 1 output layer with 1 node) is able to be used for Singapore GDP (Gross Domestic Product) growth rate.

Using the similar algorithm, the team applies the model to Country A's weekly revenue data.

The first step is to plot the auto-correlation (plot-acf) using the python library taken from statsmodels.graphics.tsaplots. The auto-correlation will determine the number of lag in the time series data that we can use as the number of parameters/number of features in the time series prediction model.

```
In [4]: # Autocorrelation
        #pd.plotting.autocorrelation_plot(series)
        #plt.figure(figsize=(15, 8))
        #plt.show()
        from statsmodels.graphics.tsaplots import plot_acf
        fig, ax = plt.subplots(figsize=(15, 8))
        plot_acf(series['REVENUE'], ax=ax)
        ax.set_ylabel('Autocorrelation')
        ax.set_xlabel('Lag (weeks)')
        plt.show()
        # from the picture below we choose the first 6 lags
```




As we can see from the above plot, we can use lag = 2 which lies above the blue confidence band. Based on this then the time series is constructed such that t+0 and t+1 will be used for the prediction of t+2. In our context since weekly revenue data is used, the future week revenue will be predicted using the previous 2 weeks revenue data. The structure will be as below:

Out[21]:

| Start_Date | t+0 | t+1 | t+2 |
|---|---|---|---|
| 2017-01-01 | 21287436.14 | 20501369.84 | 20420697.12 |
| 2017-01-08 | 20501369.84 | 20420697.12 | 14804524.02 |
| 2017-01-15 | 20420697.12 | 14804524.02 | 16357977.05 |
| 2017-01-22 | 14804524.02 | 16357977.05 | 21415433.70 |
| 2017-01-29 | 16357977.05 | 21415433.70 | 22172450.06 |
| ... | ... | ... | ... |
| 2019-11-10 | 26765677.47 | 27290758.23 | 26749709.96 |
| 2019-11-17 | 27290758.23 | 26749709.96 | 24781045.48 |
| 2019-11-24 | 26749709.96 | 24781045.48 | 24907845.08 |
| 2019-12-01 | 24781045.48 | 24907845.08 | 23903176.17 |
| 2019-12-08 | 24907845.08 | 23903176.17 | 19289834.49 |

154 rows × 3 columns

From the above, then we can take the X from 2 columns t+0 and t+1 and the truth y is taken from t+2

In [23]:
```
# the target we want to predict (lowercase y is a convention for a vector)
y = df_windowed['t+2']

# the input data (uppercase X is a convention for a matrix)
X = df_windowed.drop(columns=['t+2'])

X.shape, y.shape
```

Out[23]:
```
((154, 2), (154,))
```

After the time series data is structured properly, then we create a simple neural network model structure as below

```
In [24]: # https://www.tensorflow.org/guide/keras/overview
         # Create a simple Neural Network with 2 Dense Layers
         from tensorflow.keras import layers

         model = tf.keras.Sequential()
         model.add(layers.Dense(16, input_shape=(2,), activation='relu'))
         model.add(layers.Dense(1))
         model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                48
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 65
Trainable params: 65
Non-trainable params: 0
_________________________________________________________________
```

The next step is to split the data into training and test. The shuffle parameter is set to false so that the time series will not be chosen randomly

```
In [25]: #X_train, X_test, y_train, y_test = train_test_split(X, y)
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, shuffle = False)

         X_train.shape, X_test.shape, y_train.shape, y_test.shape

Out[25]: ((103, 2), (51, 2), (103,), (51,))
```

The model is then trained with batch size = 8 and number of epochs = 250

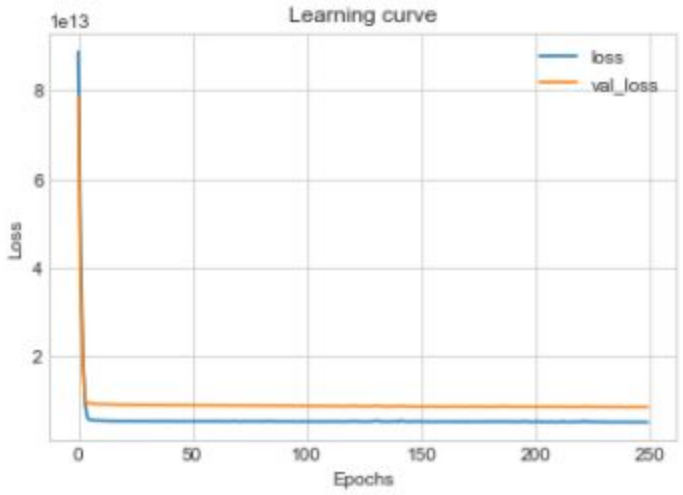
```
In [28]: # train!
         # Note: if you have more data, you can split the dataset 3 ways:
         #  train, validation, test
         # And then use the validation set (e.g. X_val, y_val) in validation_data

         model.compile(optimizer='Adam', loss='mse', metrics=['mape'])
         history = model.fit(X_train, y_train, batch_size=8, epochs=250,
                             validation_data=(X_test, y_test))
```

```
103/103 [==============================] - 0s 255us/sample - loss: 5272552883607.6113 - mape: 7.8681 - val_loss: 86424552349
21.4121 - val_mape: 8.7702
Epoch 245/250
103/103 [==============================] - 0s 252us/sample - loss: 5249191681809.3975 - mape: 7.8413 - val_loss: 86302293219
28.7842 - val_mape: 8.7204
Epoch 246/250
103/103 [==============================] - 0s 282us/sample - loss: 5293497523667.2627 - mape: 7.9972 - val_loss: 86152212284
03.4512 - val_mape: 8.7503
Epoch 247/250
103/103 [==============================] - 0s 237us/sample - loss: 5260376558224.1553 - mape: 7.9176 - val_loss: 86032581173
40.8623 - val_mape: 8.7685
Epoch 248/250
103/103 [==============================] - 0s 243us/sample - loss: 5280348140116.5049 - mape: 7.9072 - val_loss: 86085584222
97.0986 - val_mape: 8.7080
Epoch 249/250
103/103 [==============================] - 0s 223us/sample - loss: 5261901771825.7090 - mape: 7.8948 - val_loss: 86053570889
28.6279 - val_mape: 8.7290

Epoch 250/250
103/103 [==============================] - 0s 247us/sample - loss: 5263269328717.0488 - mape: 7.9484 - val_loss: 85982591854
63.2158 - val_mape: 8.7112
```

5.2.1.2. Observation

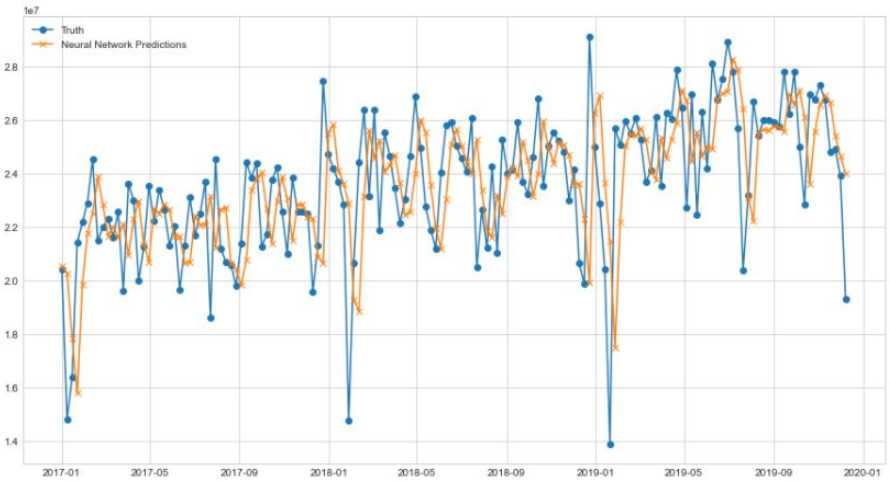
Below is the plot of the learning curve, it shows that the validation loss and training loss are going down in tandem/converged well. As per the training result in previous section, it achieved MAPE level of 8.7112% which suggest the model accuracy level is about 91% (100 - MAPE)

```
In [29]: # Check for overfitting, which is when val_loss starts to go up but
         # loss stays decreases or stays constant.

         plt.plot(history.history['loss'], label='loss')
         plt.plot(history.history['val_loss'], label='val_loss')
         plt.title('Learning curve')
         plt.ylabel('Loss')
         plt.xlabel('Epochs')
         plt.legend()
         plt.show()
```




If the result is plot and compared between the actual and prediction, the diagram is as below:

```
In [38]: fig, ax = plt.subplots(figsize=(15, 8))
         ax.plot(y, label='Truth', marker='o')
         ax.plot(df_pred, label='Neural Network Predictions', marker='x')
         ax.legend()
         plt.show()
```




We can see that the orange color (prediction) is able to follow nicely the ups and downs happening in blue color (the fact/the truth), implying that the prediction model is able to make fairly good predictions.

#### 5.2.1.3. Comparison with ARIMA Model

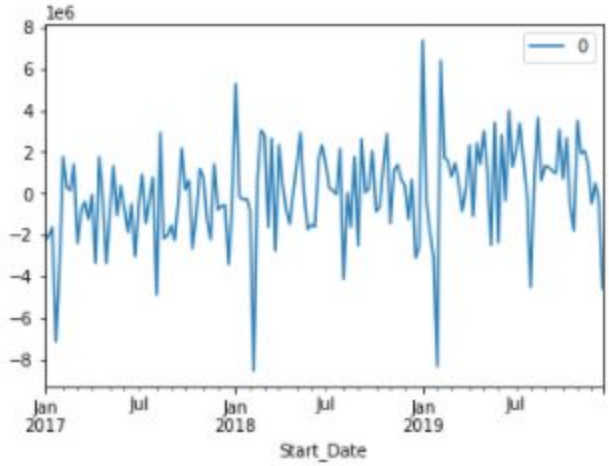
The team decided to compare the above with a simple ARIMA model that has comparable input parameters (using 2 weeks lag time series data to predict the following week)

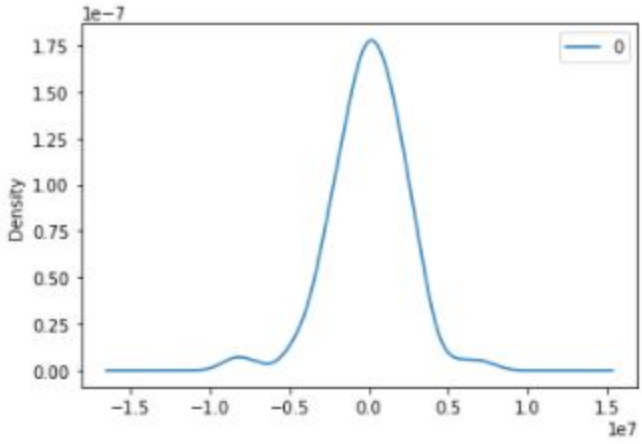
In [5]:
```
# ARIMA Model
from statsmodels.tsa.arima_model import ARIMA

model = ARIMA(series, order=(2,0,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                REVENUE   No. Observations:                  156
Model:                     ARMA(2, 0)   Log Likelihood               -2509.444
Method:                        css-mle   S.D. of innovations        2341755.263
Date:                Wed, 21 Oct 2020   AIC                           5026.888
Time:                        22:10:39   BIC                           5039.087
Sample:                    01-01-2017   HQIC                          5031.843
                         - 12-22-2019
==============================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.357e+07   3.86e+05     61.117      0.000    2.28e+07    2.43e+07
ar.L1.REVENUE     0.4032      0.080      5.032      0.000       0.246       0.560
ar.L2.REVENUE     0.1146      0.080      1.427      0.153      -0.043       0.272
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.6788           +0.0000j            1.6788            0.0000
AR.2           -5.1960           +0.0000j            5.1960            0.5000
-----------------------------------------------------------------------------
```

To make sure that the ARIMA model is sound, we check the residual. A good model will have normal curve pattern residual. Below result shows that the residual error which has normal curve pattern.

```
In [6]: # Plot residual errors
        residuals = pd.DataFrame(model_fit.resid)
        residuals.plot()
        plt.show()
        residuals.plot(kind='kde')
        plt.show()
        print(residuals.describe())
```



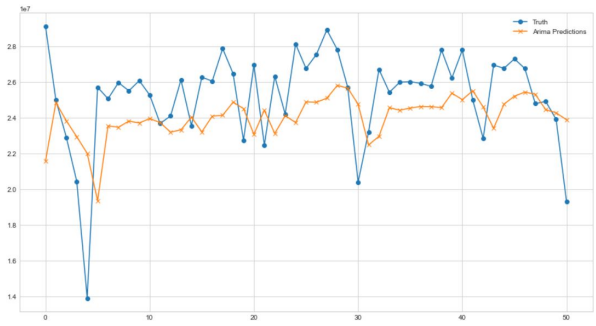

| | 0 |
|---|---|
| count | 1.560000e+02 |
| mean | -1.320242e+04 |
| std | 2.350893e+06 |
| min | -8.548220e+06 |
| 25% | -1.444079e+06 |
| 50% | 4.331409e+04 |
| 75% | 1.399843e+06 |
| max | 7.363637e+06 |

Below is the error measurement, side by side comparison between Neural Net model and ARIMA

| Neural Net Model | ARIMA |
|---|---|
| {'mape': 0.08711194339488464,<br>'me': -26311.134705882447,<br>'mae': 2038846.95,<br>'mpe': 0.010727621428671955,<br>'mse': 8598260400693.337,<br>'rmse': 2932279.0455025486} | {'mape': 0.09313337418658664,<br>'me': -1151095.0410977334,<br>'mae': 2247015.9371845583,<br>'mpe': -0.0334277048442642,<br>'mse': 8192242483941.755,<br>'rmse': 2862209.371087614} |

From the above result, MAPE and MAE wise, the neural net model is slightly better as compared to ARIMA model.

Another observation is if we plot the prediction on the test data. We can see from below graph comparison, the neural net model on the left has more ability to follow the blue curve as compared to ARIMA which tends to be more smooth (averaging effect)

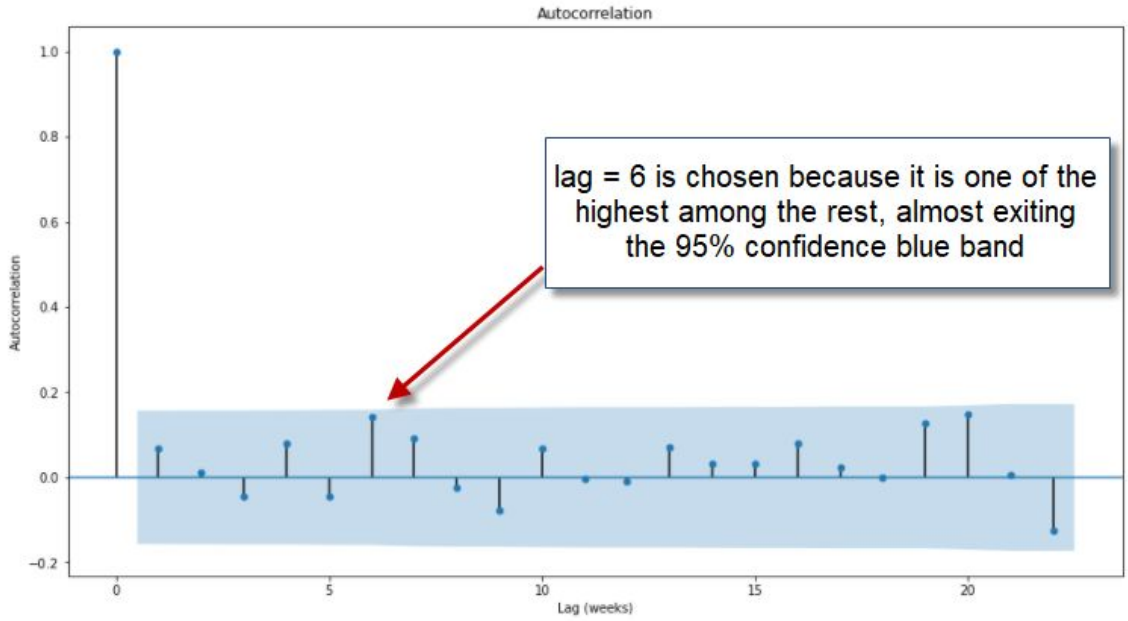
| Neural Net Model | ARIMA |
|---|---|
|  |  |

### 5.2.2. Country B

#### 5.2.2.1. Model Design

Country B's data is more challenging because based on the auto-correlation plot, there is no good number of lag data that we can use that is strong enough as per country A

In [13]:
```
# Autocorrelation
#pd.plotting.autocorrelation_plot(series)
#plt.figure(figsize=(15, 8))
#plt.show()
from statsmodels.graphics.tsaplots import plot_acf
fig, ax = plt.subplots(figsize=(15, 8))
plot_acf(series['REVENUE'], ax=ax)
ax.set_ylabel('Autocorrelation')
ax.set_xlabel('Lag (weeks)')
plt.show()
# from the picture below we choose the first 6 lags
```




Based on the above, the time-series data is structured as below where the [t+0], [t+1], [t+2], [t+3], [t+4], [t+5] are used to predict [t+6]

Out[31]:

| Start_Date | t+0 | t+1 | t+2 | t+3 | t+4 | t+5 | t+6 |
|---|---|---|---|---|---|---|---|
| 2017-01-01 | 2659489.14 | 3760130.60 | 2592270.63 | 2672616.29 | 3098990.90 | 3073205.85 | 3597591.52 |
| 2017-01-08 | 3760130.60 | 2592270.63 | 2672616.29 | 3098990.90 | 3073205.85 | 3597591.52 | 3514674.25 |
| 2017-01-15 | 2592270.63 | 2672616.29 | 3098990.90 | 3073205.85 | 3597591.52 | 3514674.25 | 3628235.25 |
| 2017-01-22 | 2672616.29 | 3098990.90 | 3073205.85 | 3597591.52 | 3514674.25 | 3628235.25 | 4006259.57 |
| 2017-01-29 | 3098990.90 | 3073205.85 | 3597591.52 | 3514674.25 | 3628235.25 | 4006259.57 | 3800597.55 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 2019-10-20 | 4345376.96 | 3520102.30 | 3711891.75 | 3846633.57 | 3362790.16 | 3657010.36 | 4463993.91 |
| 2019-10-27 | 3520102.30 | 3711891.75 | 3846633.57 | 3362790.16 | 3657010.36 | 4463993.91 | 3250925.17 |
| 2019-11-03 | 3711891.75 | 3846633.57 | 3362790.16 | 3657010.36 | 4463993.91 | 3250925.17 | 2366164.15 |
| 2019-11-10 | 3846633.57 | 3362790.16 | 3657010.36 | 4463993.91 | 3250925.17 | 2366164.15 | 3169237.36 |
| 2019-11-17 | 3362790.16 | 3657010.36 | 4463993.91 | 3250925.17 | 2366164.15 | 3169237.36 | 2294442.83 |

151 rows × 7 columns

In [32]:
```
# Formulate our problem

df_windowed.columns
```

Out[32]:
```
Index(['t+0', 't+1', 't+2', 't+3', 't+4', 't+5', 't+6'], dtype='object')
```

In [33]:
```
# the target we want to predict (lowercase y is a convention for a vector)
y = df_windowed['t+6']

# the input data (uppercase X is a convention for a matrix)
X = df_windowed.drop(columns=['t+6'])

X.shape, y.shape
```

Out[33]:
```
((151, 6), (151,))
```

Below is the neural net model structure that the team experimented and gave reasonable error level. Dropout layer of 0.25 was added to help generalize the model and based on our observation it helps to lower the error level

```
In [34]: # https://www.tensorflow.org/guide/keras/overview
         # Create a simple Neural Network with 2 Dense Layers
         from tensorflow.keras import layers

         model = tf.keras.Sequential()
         model.add(layers.Dense(32, input_shape=(6,), activation='relu'))
         model.add(layers.Dropout(0.25))
         model.add(layers.Dense(32, activation='relu'))
         model.add(layers.Dense(32, activation='relu'))
         model.add(layers.Dense(1))
         model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                224
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 2,369
Trainable params: 2,369
Non-trainable params: 0
_________________________________________________________________
```

Based on similar training epochs and batch size as done in country A, the result is as below.

```
In [38]: # train!
         # Note: if you have more data, you can split the dataset 3 ways:
         #  train, validation, test
         # And then use the validation set (e.g. X_val, y_val) in validation_data

         model.compile(optimizer='Adam', loss='mse', metrics=['mape'])
         history = model.fit(X_train, y_train, batch_size=8, epochs=250,
                             validation_data=(X_test, y_test))
```

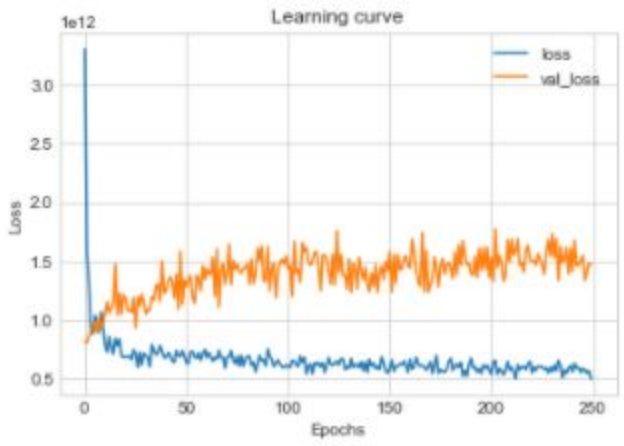
```
101/101 [==============================] - 0s 277us/sample - loss: 592721768326.3367 - mape: 28.6221 - val_loss: 14657553469
08.1599 - val_mape: 26.3482
Epoch 245/250
101/101 [==============================] - 0s 297us/sample - loss: 553437024215.4456 - mape: 28.9400 - val_loss: 15012308818
32.9600 - val_mape: 26.5383
Epoch 246/250
101/101 [==============================] - 0s 287us/sample - loss: 587268184408.7129 - mape: 28.8707 - val_loss: 15510890925
26.0801 - val_mape: 26.8873
Epoch 247/250
101/101 [==============================] - 0s 257us/sample - loss: 578750263985.4257 - mape: 28.3314 - val_loss: 13358264234
80.3201 - val_mape: 24.8380
Epoch 248/250
101/101 [==============================] - 0s 277us/sample - loss: 546920427722.7723 - mape: 28.8678 - val_loss: 14027403952
12.8000 - val_mape: 25.3934
Epoch 249/250
101/101 [==============================] - 0s 297us/sample - loss: 568088432761.6633 - mape: 28.8712 - val_loss: 14821238171
23.8401 - val_mape: 26.4101
Epoch 250/250
101/101 [==============================] - 0s 287us/sample - loss: 497891031465.8218 - mape: 27.5013 - val_loss: 14722305943
14.2400 - val_mape: 26.4788
```

The model achieved validation MAPE of 26,4788% which suggest accuracy of only about 73%

5.2.2.2. Observation

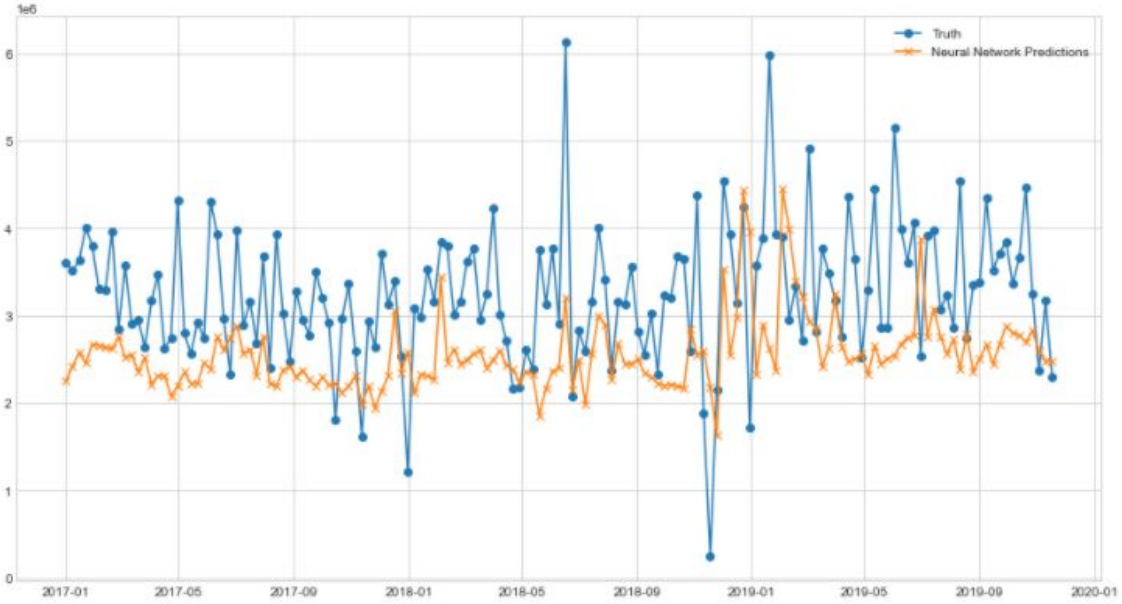
The learning curve diagram looks as below. It doesn't converge nicely as what we experience with country A with good auto-correlation plot

```
In [39]: # Check for overfitting, which is when val_loss starts to go up but
         # loss stays decreases or stays constant.

         plt.plot(history.history['loss'], label='loss')
         plt.plot(history.history['val_loss'], label='val_loss')
         plt.title('Learning curve')
         plt.ylabel('Loss')
         plt.xlabel('Epochs')
         plt.legend()
         plt.show()
```




The overall prediction looks as below. The model prediction (orange color) seems to not be able to follow the ups and downs happening with real data (blue color)

```
In [43]: fig, ax = plt.subplots(figsize=(15, 8))
         ax.plot(y, label='Truth', marker='o')
         ax.plot(df_pred, label='Neural Network Predictions', marker='x')
         ax.legend()
         plt.show()
```




### 5.2.2.3. Comparison with ARIMA Model

Below is the comparable ARIMA model for country B

In [14]:
```
# ARIMA Model
from statsmodels.tsa.arima_model import ARIMA

model = ARIMA(series, order=(6,0,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

```
C:\Users\HCAND\anaconda3\envs\psupr\lib\site-packages\statsmodels\tsa\base\tsa_model.py:162: ValueWarning: No frequency informa
tion was provided, so inferred frequency W-SUN will be used.
  % freq, ValueWarning)
```

ARMA Model Results

| | | | |
|---|---|---|---|
| Dep. Variable: | REVENUE | No. Observations: | 157 |
| Model: | ARMA(6, 0) | Log Likelihood | -2347.966 |
| Method: | css-mle | S.D. of innovations | 755886.626 |
| Date: | Thu, 22 Oct 2020 | AIC | 4711.932 |
| Time: | 17:05:22 | BIC | 4736.382 |
| Sample: | 01-01-2017 - 12-29-2019 | HQIC | 4721.862 |

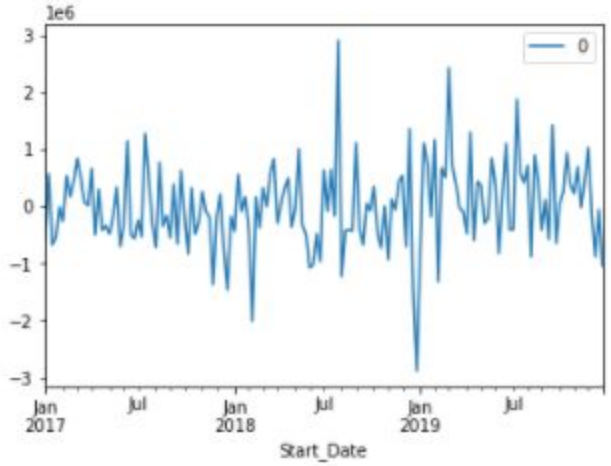
| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| const | 3.235e+06 | 7.55e+04 | 42.865 | 0.000 | 3.09e+06 | 3.38e+06 |
| ar.L1.REVENUE | 0.0877 | 0.079 | 1.106 | 0.269 | -0.068 | 0.243 |
| ar.L2.REVENUE | -0.0087 | 0.079 | -0.110 | 0.912 | -0.164 | 0.146 |
| ar.L3.REVENUE | -0.0427 | 0.079 | -0.541 | 0.588 | -0.197 | 0.112 |
| ar.L4.REVENUE | 0.0900 | 0.079 | 1.140 | 0.254 | -0.065 | 0.245 |
| ar.L5.REVENUE | -0.0694 | 0.080 | -0.870 | 0.384 | -0.226 | 0.087 |
| ar.L6.REVENUE | 0.1488 | 0.079 | 1.873 | 0.061 | -0.007 | 0.304 |

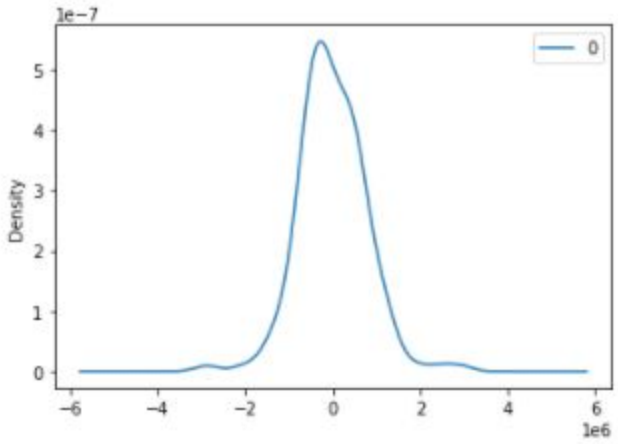
Roots

| | Real | Imaginary | Modulus | Frequency |
|---|---|---|---|---|
| AR.1 | -1.2592 | -0.0000j | 1.2592 | -0.5000 |
| AR.2 | -0.5583 | -1.2758j | 1.3926 | -0.3157 |
| AR.3 | -0.5583 | +1.2758j | 1.3926 | 0.3157 |
| AR.4 | 1.3803 | -0.0000j | 1.3803 | -0.0000 |
| AR.5 | 0.7310 | -1.2082j | 1.4121 | -0.1634 |
| AR.6 | 0.7310 | +1.2082j | 1.4121 | 0.1634 |

To make sure the model is proper, the observation below whether the residual has a normal curve is necessary. Based on the plot of residual curve below, indeed the residual has normal curve pattern

```
In [15]: # Plot residual errors
         residuals = pd.DataFrame(model_fit.resid)
         residuals.plot()
         plt.show()
         residuals.plot(kind='kde')
         plt.show()
         print(residuals.describe())
```



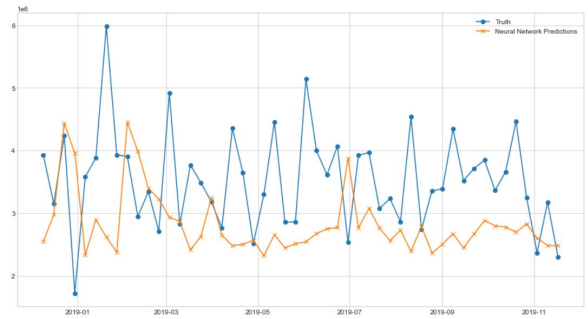
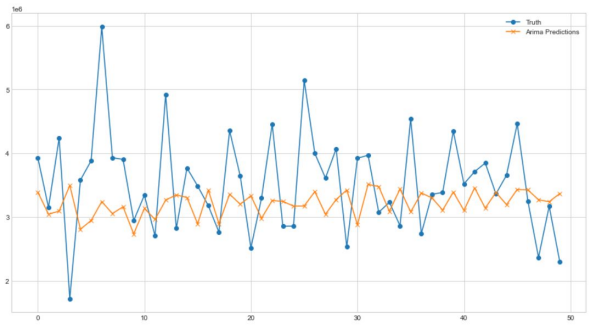

```
                  0
count  1.570000e+02
mean  -9.485347e+02
std    7.585214e+05
min   -2.879574e+06
25%   -4.781257e+05
50%   -5.795463e+04
75%    4.514433e+05
max    2.907925e+06
```

Below is the error measurement, side by side comparison between Neural Net model and ARIMA

| Neural Net Model | ARIMA |
|---|---|
| {'mape': 0.26478806958325907,<br>'me': -708250.2729999999,<br>'mae': 970446.3878,<br>'mpe': -0.15167938856403457,<br>'mse': 1472230696740.6387,<br>'rmse': 1213355.140402281} | {'mape': 0.19559368246618014,<br>'me': -318325.89605470805,<br>'mae': 686626.0844174443,<br>'mpe': -0.04131258542954698,<br>'mse': 757182062625.777,<br>'rmse': 870162.0898578477} |

From the above result, both MAPE and MAE wise the ARIMA model is better. RMSE wise, ARIMA also has the better result (less big errors).

Another observation is if we plot the prediction on the test data. We can see from below graph comparison, the neural net model on the left tends more to follow the blue curve (more volatile) as compared to ARIMA which tends to be more smooth (averaging effect)

| Neural Net Model | ARIMA |
|---|---|
|  |  |

From the above observations, we can see that ARIMA model is more stable and tends to stay in the middle of the ups and downs while the neural net model is less stable and tend to have bigger ups and downs.

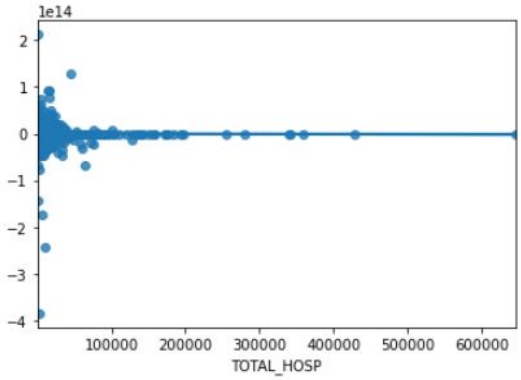
## 5.3. Use Case #3 Hospital Bill Estimation Upon Admission

### 5.3.1. Model Design

As this is a smaller dataset as compared to the one used in Use case 1& 2, there will not be a splitting of the dataset based on hospitals.  The regression problem will be trained with several models, including the ensemble methods taught in the class.

### 5.3.2. Observation and Comparison

5.3.2.1 Linear Regression



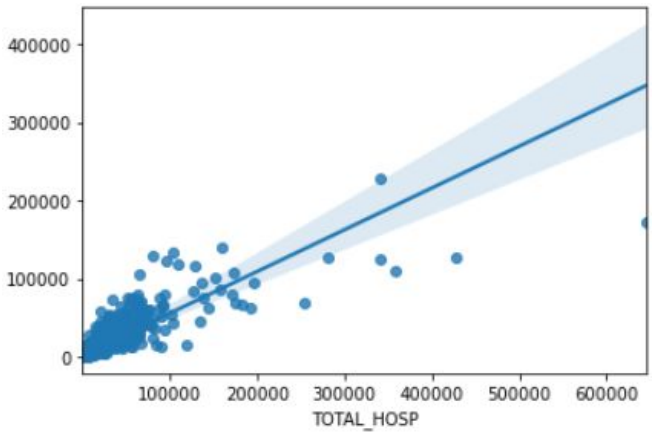

```
forecast_accuracy(y_pred, df_y_test)

{'mape': 128968586.85256241,
 'me': -26962606429.271862,
 'mae': 540683888662.51843,
 'mpe': -1148534.8089953386,
 'mse': 3.2087639831807435e+25,
 'rmse': 5664595292852.564}
```

The simple Linear Regression method yields the worst result among the 4 models, thus this hints that a non-linear regression model will better fit the desired solution.

5.3.2.2 Neural Networks

The Neural Net is created with the structure as below. Epochs = 100 , Batch_size = 128.

```
13]: NN_model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_absolute_error'])
     NN_model.summary()

     Model: "sequential"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
     dense (Dense)                (None, 4096)              58630144
     _________________________________________________________________
     dropout (Dropout)            (None, 4096)              0
     _________________________________________________________________
     dense_1 (Dense)              (None, 4096)              16781312
     _________________________________________________________________
     batch_normalization (BatchNo (None, 4096)              16384
     _________________________________________________________________
     activation (Activation)      (None, 4096)              0
     _________________________________________________________________
     dropout_1 (Dropout)          (None, 4096)              0
     _________________________________________________________________
     dense_2 (Dense)              (None, 4096)              16781312
     _________________________________________________________________
     batch_normalization_1 (Batch (None, 4096)              16384
     _________________________________________________________________
     activation_1 (Activation)    (None, 4096)              0
     _________________________________________________________________
     dropout_2 (Dropout)          (None, 4096)              0
     _________________________________________________________________
     dense_3 (Dense)              (None, 1)                 4097
     =================================================================
     Total params: 92,229,633
     Trainable params: 92,213,249
     Non-trainable params: 16,384
```



```
[8]: {'mape': 0.28837315690407184,
      'me': -637.7351468577169,
      'mae': 1808.4606558727216,
      'mpe': 0.0626254968571037,
      'mse': 50660855.39250669,
      'rmse': 7117.643949545853,
      'mmape': 0.2824793331206611}
```

The neural network yields the best MAE score among the 4 models.

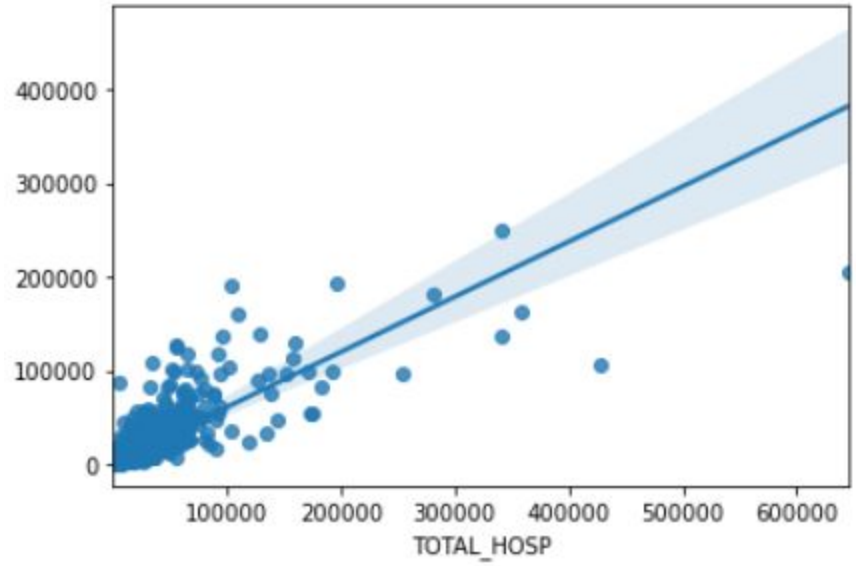
We explored the effects of a wider neural network (more nodes per layer) versus a deeper neural network (more layers). In preparation for a deeper network, batch normalisation is added for each layer, between the dense layer and its activation, so that the network will not be saturated when more layers are added. 0.2 dropout is added for each layer to prevent overfitting.

There are a few constraints while building the neural network to be trained on a PC. Tensorflow keras is used for the neural network, as it can perform parallel computation on the GPUs, which is very fast. However, the number of nodes per layer, the number of layers and the batch size determines the amount of GPU memory is required. Therefore, on a PC, we can only test up to 4096 nodes per layer, and 7 layers , using batch size of 32.

The findings are that when more than 4 layers, the performance deteriorates. And the more nodes per layer, the performance improves. Batch size of 128 performs better than batch size of 32.Therefore, for architectures that have less layers, 128 batch size is used when possible.

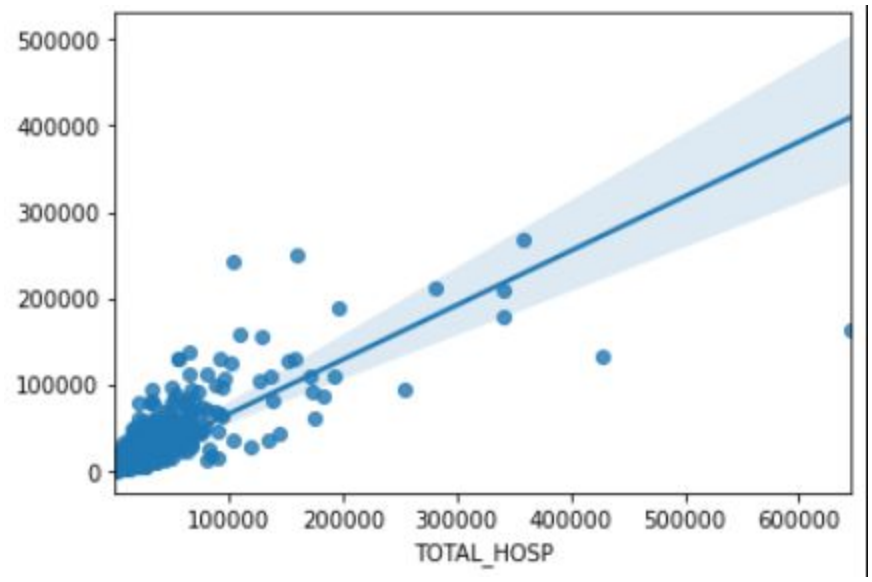
The best architecture is 3 layers with 4096 nodes per layer. However, this is based on training the model on a PC. We deduce that if the number of nodes per layer can have more than the number of features, which is 14313, the performance may be better, but it may require more than 50gb GPU memory.

5.3.2.3 Random Forest Regressor




```
[10]: forecast_accuracy(predicted_forest, df_y_test)

[10]: {'mape': 0.3382865575516256,
       'me': -522.6735844757893,
       'mae': 2058.176886314511,
       'mpe': 0.11845408199322385,
       'mse': 50180772.50922567,
       'rmse': 7083.83882575159}
```

5.3.2.4 XGBoost Regressor




```
[17]: forecast_accuracy(XGBpredictions, df_y_test)

[17]: {'mape': 0.46337955520919316,
       'me': -81.64403423137739,
       'mae': 2237.757650049478,
       'mpe': 0.26924646510863176,
       'mse': 49369558.733084954,
       'rmse': 7026.347467431778}
```

Both ensemble methods yield a similar result with MAE as the error metric of interest.

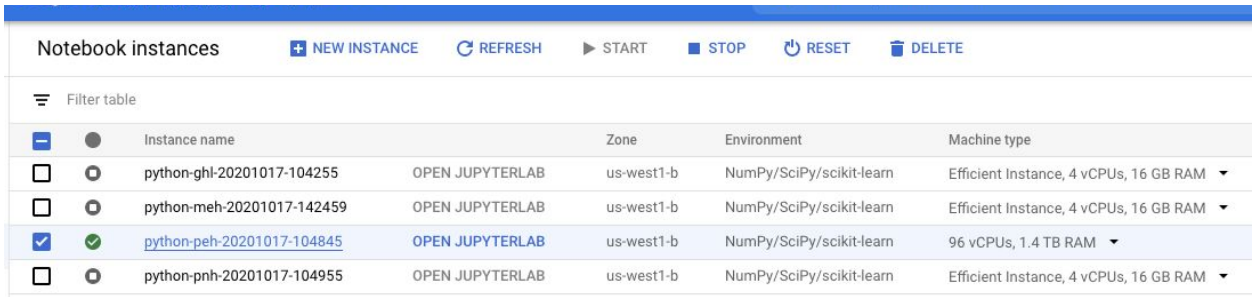
# 6. Design, System Architecture and Deployment

One of the goals that we set out for this project is to go beyond just classroom experiments, and try to implement a real-world end-to-end machine learning system and workflow. Some of the main challenges are: (1) the lack of standard toolings for collaboration, (2) Consistent data pipeline for training and serving the model - in different compute environments, and (3) the computational resources required to train large models. To address these challenges, we adopted Kubeflow [11] - an open-source platform that builds on top of Kubernetes [12] for easy deploying, scalability, and management. Kubeflow comes with a set of components like Cloud-managed **Jupyter Notebook** for collaboration and scalability, **Pipelines** for dependencies and orchestrations, and **Katib**[1] for model hyperparameters tuning.

## 6.1. Training System Components

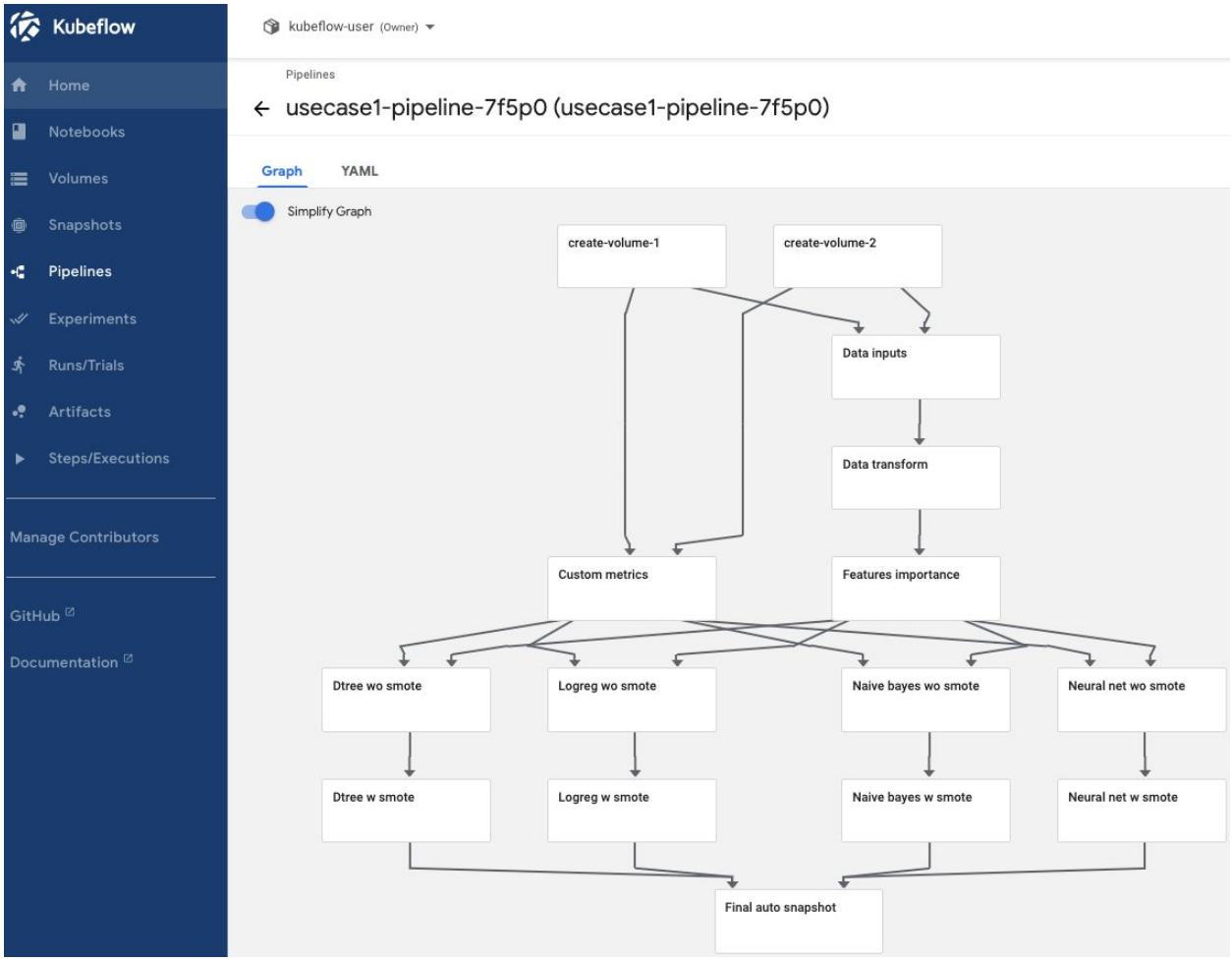
Kubeflow can be deployed manually on any computing infrastructure, or we can leverage the managed service option in public cloud for some of the Kubeflow components. For example, Google Cloud Platform (GCP) offers Managed JupyterLab notebook instances as part of the overall AI Platform.

During the training phase, we provisioned notebook instances with very large memory as depicted in the image below for an on-demand basis, and managed to process feature extractions and model trainings that require huge computational power like KNN and RandomForest. It also helps us parallelize several training pipelines for quicker iteration and improvement.




[1] We didn't use Katib extensively in this project, as the component was going through a major version upgrade, and there were some compatibility issues.

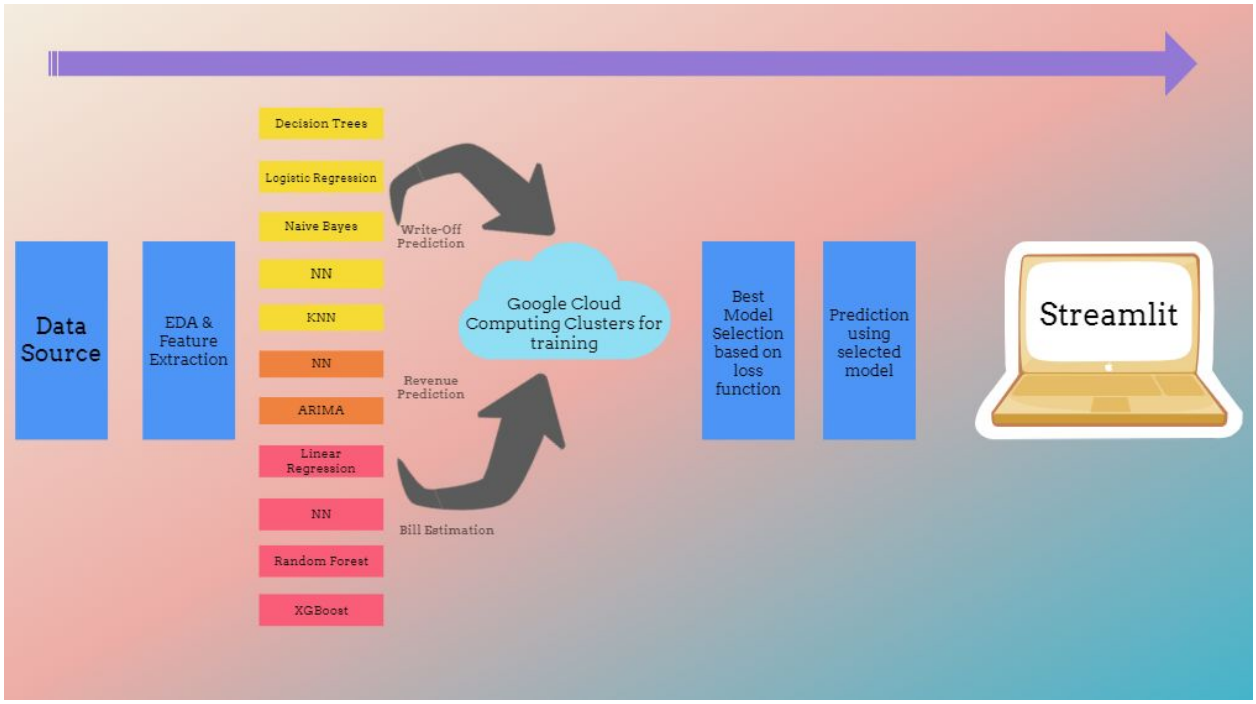
Kubeflow Pipeline allows the orchestration of the complete ML workflow via a series of steps, where different components are described and combined in the form of an acyclic graph. Each pipeline component is a self-contained set of code, with definitions for inputs and outputs, responsible for data preprocessing, data transformation, model training, and so on. Each component is then packaged as a Docker image, and will be executed accordingly to the defined graph topology. Again this approach enables us to build pipelines quickly, and scale up to train with more powerful infrastructure with consistency.




## 6.2. Deployment

Even though we have trained the ML models for all 3 use cases, due to timeline constraint, for demonstration purpose, we have only implemented the deployment for Model Serving for Use Case #3.

The overall architecture of the system is best captured with the following diagram:




Streamlit features a Single Page Application that resembles the user interface of the actual healthcare system to capture patient information upon hospital admission. The application will return the estimated hospital bill once the inputs are captured, and submitted to the model for inference.

# Hospital Bill Estimation upon Admission

Demo - Default values

CASE_NUMBER

1019003485

TREATMENT_CATEGORY

2BD

PATIENT_NUMBER

6114934

PRIMARY_DIAGNOSIS_SID

27276

CASE_INSTITUTION

GHL

SUBMIT

| | 0 |
|---|---|
| MARITAL_STATUS | MARRI |
| RELIGION | nan |
| NATIONALITY | SINGAPOREAN |
| RESID_CTY | SINGAPORE |
| RESID_POSTALCODE | nan |
| OCCUPATION | nan |
| RESID_GEOAREA | SINGAPORE |
| NONRESID_FLAG | N |
| IDENT_TYPE | PINK S'PORE IC |
| CONT_POSTAL | nan |
| CONT_RELATION | BROTHER |

Prediction: SGD $ 3918.47

The application is powered by Streamlit [13] - a popular Python open-source app framework for Machine Learning and Data Science teams, to quickly turn data scripts into shareable web apps. Both Frontend and Backend (the ML model and inference code) are packaged into a single Docker container image for easy deployment, as well as portable and consistent experience across any infrastructure environment.

To deploy the application that is accessible on local browser at port 80, execute the following command from terminal:

```
$ docker run -d -p 80:80 asia.gcr.io/my-spark-iss/st-usecase3:v1.2

e470d762d82d6813537ffa6aec5d1446b0d7c7bd6c718f80065c5be5a620ad18
```

To clean up, remove the running container with its unique id:

```
$ docker rm -vf e470d762d82d6813537ffa6aec5
```

The current container image carries an embedded model file, and while convenient this option inflates the size of the container image by quite a bit. An alternative deployment approach is to have the application pulling the ML model storing externally during the initialization phase, and thus significantly reduces the container image size, and at the same time facilitates better model update in the future without the need to rebuild the image.

# 7. Overall Conclusion

- For Use Case #1
    - Decision Tree model ability to detect positive cases are not improved with the usage of SMOTE. The usage of SMOTE increases the number of false positive cases in Decision Tree models as we increase the SMOTE ratio
    - Logistic Regression model ability to detect positive cases is very poor without SMOTE. With SMOTE, the logistic regression models ability to detect positive cases are significantly increased; however we need to observe the SMOTE ratio because we observe at some level the detection ability is not increased with the increasing SMOTE ratio and the number of false positive cases keep increasing if we increase the SMOTE ratio. For this project, the best model to detect the positive cases is Logistics Regression Model at SMOTE ratio of 0.9. The model can detect 73.47% of the all true positive cases in the test data.
    - Naive Bayes ability to detect positive cases is quite good even without SMOTE. With SMOTE, the detection ability is improved. In our observation, the ability to detect positive cases for Naive Bayes model reached the peak at SMOTE ratio of 0.5. After which, only false positives cases are increasing as we increase the SMOTE ratio
    - For Neural Network (MLPClassifier type)
        - If we have the same number of layer, the ability to detect positive cases only increases marginally when we raise the number of nodes
        - Adding the number of layers also doesn't seem to increase the ability to detect positive cases
        - Even with SMOTE, the ability of MLPClassifer model to detect minority positive cases are not improved
        - Even though the ability to detect minority positive cases is not good for MLPClassifer, there is a positive side that false positive cases are the lowest as compared to other models. It is shown in the F1 score for the minority classification (classification 1) because the F1 score formula takes into account the balance between precision and recall

scores. In the scenario where the business case tries to avoid false positives, then this model could be the better choice

- For KNN, the ability to detect true positive cases improves with SMOTE but it gets stagnant at much lower SMOTE ratio as compared to Logistics Regression model. Based our observation, SMOTE ratio of 0.5 achieved the same number of true positive detection as SMOTE ratio of 1.0 for KNN model
- Use case #1 emphasis is to find a model with good ability to detect rare write-off cases whilst countering the number of false positives by incorporating simple business rules to handle the increase of false positive cases from the model. Based on the project observation, logistic regression with SMOTE ratio = 0.9 achieved the best result with ability to detect true positive case at 73.47% (144 out of 196 total true positive cases in the test data)

- For Use Case #2, both the Neural Net and ARIMA works well for country A with a good auto-correlation shown in the statistical analysis. For Country B, there is no significant auto-correlation level, therefore the both aforementioned models have bigger error levels (less accuracy) and the ARIMA model performs better than the Neural Network model.

- For Use Case #3, using Mean Absolute Error(MAE) as the loss function, the Neural Network has comparable, if not slightly better results compared to the vendor implemented model. Further testing on new datasets have to be done for verification.

| | MAPE | ME | MAE | MPE | MSE | RMSE |
|---|---|---|---|---|---|---|
| *3rd Party Vendor Metric* | *21* | | *1850* | | | |
| PRS-PM ( Linear Regression) | 128968586 | -26962606429 | 5.40684E+11 | -1148534 | 3.21E+25 | 5.6646E+12 |
| **PRS-PM ( Neural Networks)** | **0.288** | **-637.735** | **1808.46** | **0.062625** | **50660855.39** | **7117** |
| PRS-PM ( Random Forest Regressor) | 0.33828 | -522 | 2058 | 0.11845 | 50180882 | 7083.83 |
| PRS-PM ( XGBoost Regressor) | 0.4633 | -81.644 | 2237 | 0.269246 | 49369558 | 7026.34 |

# 8. References

[1] Veatch, Robert M. 1983. “Ethical Dilemmas of For-Profit Enterprise in Health Care”. NCBI Resources, [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK216766/

[2] Raghavan, Prabha and Rajagopal, Davya. 30th Nov 2017. “Healthcare's dilemma: Balancing public needs with business sense”. The Economic Times, [Online]. Available: https://economictimes.indiatimes.com/industry/healthcare/biotech/pharmaceuticals/healthcares-dilemma-balancing-public-needs-with-business-sense/articleshow/61855883.cms?from=mdr

[3] Singapore Ministry of Health. 1st Oct 2015. “Licensing Terms and Conditions on Provision of Information on Charges and Financial Counselling Under Section 6(2)(a) and 6(5) of the Private Hospitals and Medical Clinics Act (CAP 248)”. [Online]. Available: https://www.moh.gov.sg/docs/librariesprovider5/default-document-library/addendum1ltcfc1-10-2015.pdf

[4] Wikipedia. “Joint Commission”. [Online]. Available: https://en.wikipedia.org/wiki/Joint_Commission

[5] Organization Website. “About JCI”. [Online]. Available: https://www.jointcommissioninternational.org/about-jci/

[6] Wikipedia, “International Classification of Diseases”. [Online]. Available: https://en.wikipedia.org/wiki/International_Classification_of_Diseases

[7] Singapore Ministry of Health, “Table of Surgical Procedures”. [Online]. Available: https://www.moh.gov.sg/docs/librariesprovider5/medisave/table-of-surgical-procedures.pdf

[8] Brownlee, Jason. 21st Aug 2020. “SMOTE for Imbalanced Classification with Python”.[Online]. Available: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

[9] Chawla, Bowyer, Hall, and Kegelmeyer. 9th June 2011. “SMOTE: Synthetic Minority Over-sampling Technique”. [Online]. Available: https://arxiv.org/abs/1106.1813

[10] Ong, Lisa. 11th June 2020. “Learn from History 101: Demo Notebook”. [Online]. Available: https://github.com/lisaong/stackup-workshops/blob/master/learn-history/learning_from_history.ipynb

[11] Kubeflow - The Kubeflow project is dedicated to making deployments of machine learning (ML) workflows on Kubernetes. Available: https://www.kubeflow.org/docs/about/kubeflow/

[12] Kubernetes - An open-source system for automating deployment, scaling, and management of containerized applications. Available: https://kubernetes.io/

[13] Streamlit - The fastest way to create data apps. Available: https://www.streamlit.io/