Configuration Manual

MSc Research Project
AI for Business

Khin Yeik Mon
Student ID: x22180133

School of Computing
National College of Ireland

Supervisor: Victor Del Rosal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Khin Yeik Mon
Student ID:	x22180133
Programme:	AI for Business
Year:	2024
Module:	MSc Research Project
Supervisor:	Victor Del Rosal
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	538
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Khin Yeik Mon
Date:	6th August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Khin Yeik Mon
x22180133

1 Introduction

This document outlines the specific hardware and software requirements used to conduct the research project on Medicare fraud detection through data analysis. Additionally, it details the step-by-step process followed to successfully complete the project.

2 System Configuration

2.1 Hardware Requirements

- System OS: Windows 10
- Processor: i5
- RAM: 8 GB

2.2 Software Requirements

The project is implemented in Google Colab with the Python 3.

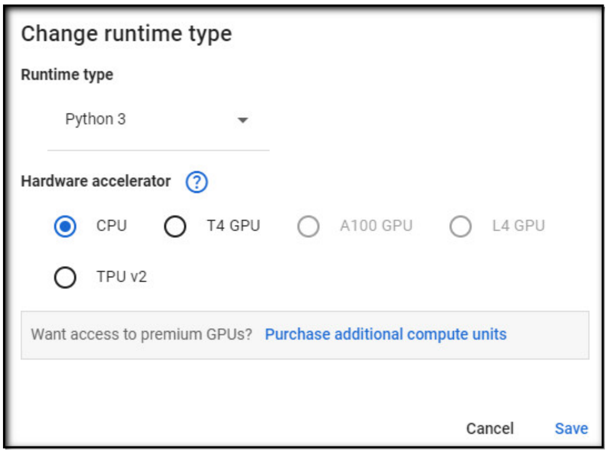

Figure 1: Google Colab Run Type Configuration

3 Implementation

3.1 Data Collection

The dataset was obtained from the Kaggle website and must be extracted before use. The dataset link is provided below:
https://www.kaggle.com/datasets/rohitrox/healthcare-provider-fraud-detection

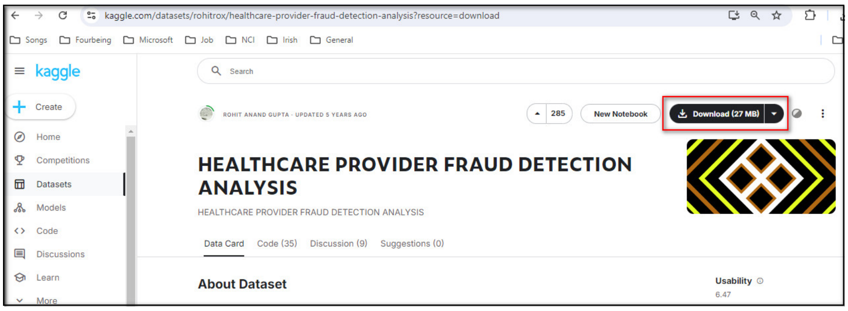

Figure 2: Dataset Link

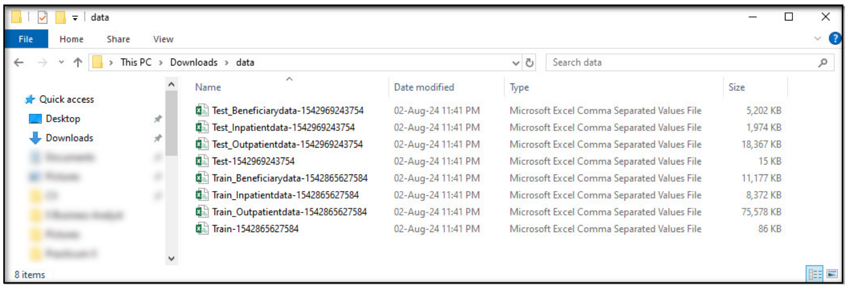

Figure 3: Dataset Extraction

The dataset consists of multiple CSV files and need to load all these files to Colab.

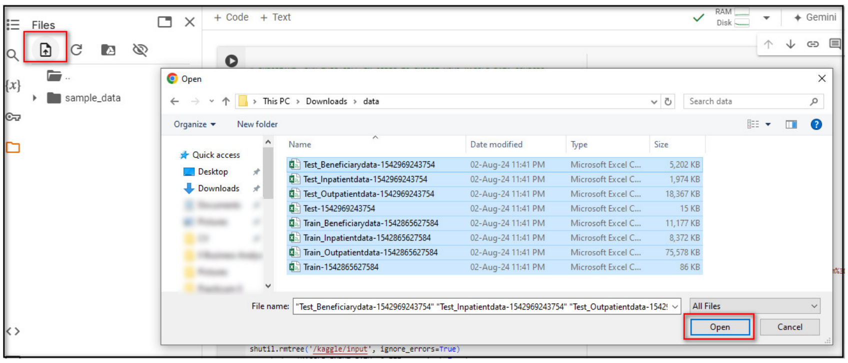

Figure 4: Uploading data files

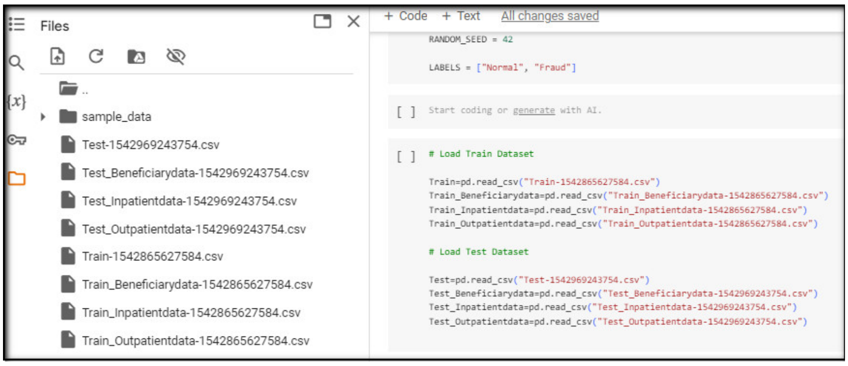

Figure 5: After Uploading data files

3.2 Feature Selection

Given that we have already extracted maximum information from these columns by grouping, we can now eliminate redundant columns from the dataset.

Figure 6: Feature Selection

3.3 Data Pre-processing

Replace missing values in numeric columns with zeros.

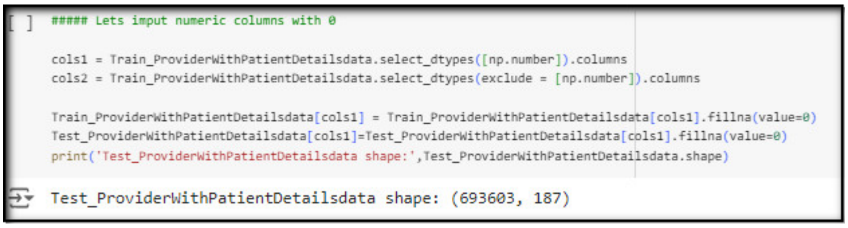

Figure 7: Data Pre-processing

3.4 Feature Engineering

Combining the training and test data for feature engineering can be tempting, especially when the test data seems to lack the full range of values present in the training data. However, this approach introduces data leakage, as the model gains access to information about the test set it shouldn't have during evaluation. Instead, we recommend exploring alternative feature engineering techniques that leverage only the training data. This ensures a more robust and unbiased evaluation of the model's performance on unseen data.

```
## Lets add both test and train datasets

Test_ProviderWithPatientDetailsdata=pd.concat([Test_ProviderWithPatientDetailsdata,
                                               Train_ProviderWithPatientDetailsdata[col_merge]])
```

Figure 8: Feature Engineering

By examining claim counts and specific combinations of provider, beneficiary, physician, diagnosis, and procedure codes, patterns indicative of organized fraud can be uncovered.

x=diagnosiscode_2chars.sort_values(ascending=True)

```
[ ] x=diagnosiscode_2chars.sort_values(ascending=True)

[ ] x.unique()
    #x.value_counts()[:10]

array(['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10',
       '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21',
       '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32',
       '33', '34', '35', '36', '37', '38', '39', '40', '41', '42', '43',
       '44', '45', '46', '47', '48', '49', '50', '51', '52', '53', '54',
       '55', '56', '57', '58', '59', '60', '61', '62', '63', '64', '65',
       '66', '67', '68', '69', '70', '71', '72', '73', '74', '75', '76',
       '77', '78', '79', '80', '81', '82', '83', '84', '85', '86', '87',
       '88', '89', '90', '91', '92', '93', '94', '95', '96', '97', '98',
       '99', 'E8', 'E9', 'V0', 'V1', 'V2', 'V4', 'V5', 'V6', 'V7', 'V8',
       'na'], dtype=object)
```

Above Data Shows that if we take only first 2 characters of diagnosis code for the purpose of grouping them, we might end up creating large sparse matrix, as each 'code' column will generate 120+ dummy columns.This will increase computational time and loose explicability.

Figure 9: Feature Engineering

3.5 Exploratory Data Analysis

Analyze the distribution of fraudulent and non-fraudulent cases in both the training and combined datasets.

```
#PLotting the frequencies of fraud and non-fraud Merged transactions in the data

sns.set_style('white',rc={'figure.figsize':(12,8)})
count_classes = pd.value_counts(Train_ProviderWithPatientDetailsdata['PotentialFraud'], sort = True)
print("Percent Distribution of Potential Fraud class:- \n",count_classes*100/len(Train_ProviderWithPatientDetailsdata))
LABELS = ["Non Fraud", "Fraud"]
#Drawing a barplot
count_classes.plot(kind = 'bar', rot=0,figsize=(10,6))

#Giving titles and labels to the plot
plt.title("Potential Fraud distribution in Aggregated claim transactional data")
plt.xticks(range(2), LABELS)
plt.xlabel("Potential Fraud Class ")
plt.ylabel("Number of PotentialFraud per Class ")

plt.savefig('PotentialFraudDistributionInMergedData')
```

```
Percent Distribution of Potential Fraud class:-
 PotentialFraud
No      61.878931
Yes     38.121069
Name: count, dtype: float64
```

Figure 10: Exploratory Data Analysis

The initial analysis reveals a higher prevalence of fraudulent claims compared to legitimate ones. To gain deeper insights, we will examine claim volumes and associated amounts across various categories such as beneficiary, physician, and diagnosis.

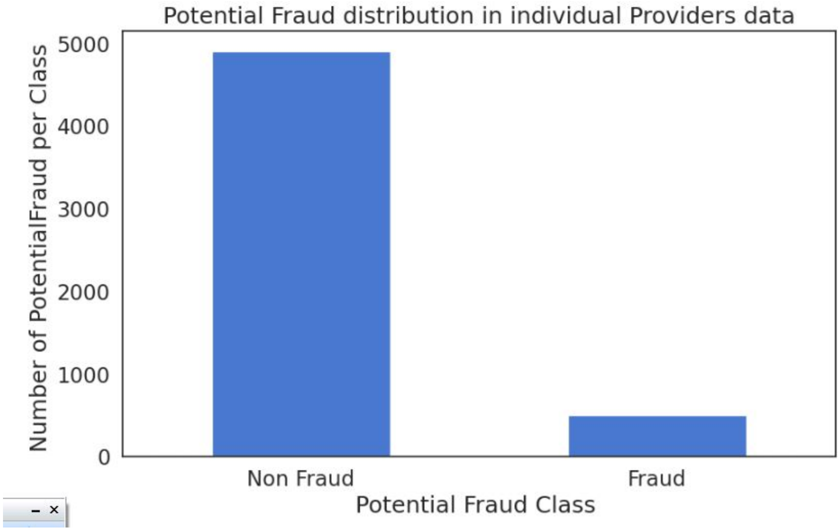

Figure 11: Exploratory Data Analysis

3.6 Model Building

1. Logistic Regression
   The "balanced" mode automatically adjusts class weights based on class frequency imbalance. Classes with fewer instances receive higher weights to counteract their underrepresentation in the dataset.

```
from sklearn.linear_model import LogisticRegressionCV

log = LogisticRegressionCV(cv=10,class_weight='balanced',random_state=123)

# The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies
#in the input data as ``n_samples / (n_classes * np.bincount(y))``.

log.fit(X_train,y_train)
```

```
LogisticRegressionCV
LogisticRegressionCV(class_weight='balanced', cv=10, random_state=123)
```

Figure 12: Model Building for Logistic regression

2. Random Forest

```
## Lets Apply Random Forest
from sklearn.ensemble import RandomForestClassifier
```

```
rfc = RandomForestClassifier(n_estimators=500,class_weight='balanced',random_state=123,max_depth=4)   # We will set max_depth =4
```

```
rfc.fit(X_train,y_train)  #fit the model
```

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=4, n_estimators=500,
                       random_state=123)
```

Figure 13: Model Building for Random Forest

3. Autoencoder
   Autoencoders will be used to learn patterns within normal transactions. By identifying significant reconstruction errors, we aim to detect potential fraudulent activity.

```
train_pca = np.array(train_pca)
test_pca =np.array(test_pca)
```

Figure 14: Converting data to array

```
X_train_F = X_train_pca[X_train_pca[:,-1] == 1]
print(X_train_F.shape)
```

Figure 15: Separating out the fraud records from the train data

```
from sklearn.metrics import precision_score,recall_score,accuracy_score,confusion_matrix
fraud = (recon_error[:,1]>thr)
print("Recall=",recall_score(y_test,fraud))
print("Precision=",precision_score(y_test,fraud))
print("Accuracy=",accuracy_score(y_test,fraud))
print("F1-Score",f1_score(y_test,fraud))
```

Figure 16: SKlearn Import Function

4 Evaluation

All models are evaluated using industry-standard metrics and the results are summarized as shown in table.

4.1 Logistic Regression

```
#Confusion matrix, Accuracy, sensitivity and specificity

from sklearn.metrics import confusion_matrix,accuracy_score,cohen_kappa_score,roc_auc_score,f1_score,auc

cm0 = confusion_matrix(y_train, log_train_pred_60,labels=[1,0])
print('Confusion Matrix Train : \n', cm0)

cm1 = confusion_matrix(y_val, log_val_pred_60,labels=[1,0])
print('Confusion Matrix Val: \n', cm1)

total0=sum(sum(cm0))
total1=sum(sum(cm1))
#####from confusion matrix calculate accuracy
accuracy0=(cm0[0,0]+cm0[1,1])/total0
print ('Accuracy Train: ', accuracy0)

accuracy1=(cm1[0,0]+cm1[1,1])/total1
print ('Accuracy Val: ', accuracy1)

sensitivity0 = cm0[0,0]/(cm0[0,0]+cm0[0,1])
print('Sensitivity Train : ', sensitivity0 )

sensitivity1 = cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity Val: ', sensitivity1 )


specificity0 = cm0[1,1]/(cm0[1,0]+cm0[1,1])
print('Specificity Train: ', specificity0)

specificity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity Val: ', specificity1)

KappaValue=cohen_kappa_score(y_val, log_val_pred_60)
print("Kappa Value :",KappaValue)
AUC=roc_auc_score(y_val, log_val_pred_60)

print("AUC         :",AUC)

print("F1-Score Train  : ",f1_score(y_train, log_train_pred_60))
```

Figure 17: Logistic Regression Evaluation

```
Confusion Matrix Train :
 [[ 269   85]
 [ 210 3223]]
Confusion Matrix Val:
 [[ 102   50]
 [  92 1379]]
Accuracy Train:  0.9221019276472141
Accuracy Val:  0.9125077017868145
Sensitivity Train :  0.7598870056497176
Sensitivity Val:  0.6710526315789473
Specificity Train:  0.9388290125254879
Specificity Val:  0.9374575118966689
Kappa Value : 0.5414360243701526
AUC          : 0.8042550717378081
F1-Score Train  :  0.6458583433373348
F1-Score Val  :  0.5895953757225434
```

Figure 18: Logistic Regression Evaluation Result

```
log_test_pred_60 = (log.predict_proba(X_teststd)[:,1]>0.60).astype(bool)
log_test_pred=pd.DataFrame(log_test_pred_60)
log_test_pred.head(2)
```

Figure 19: Prediction on Test data

4.2 Random Forest

```
#Confusion matrix, Accuracy, sensitivity and specificity
from sklearn.metrics import confusion_matrix,accuracy_score,cohen_kappa_score,roc_auc_score,f1_score,roc_curve

cm0 = confusion_matrix(y_train, rfc_train_pred,labels=[1,0])
print('Confusion Matrix Train : \n', cm0)

cm1 = confusion_matrix(y_val, rfc_val_pred,labels=[1,0])
print('Confusion Matrix Test: \n', cm1)

total0=sum(sum(cm0))
total1=sum(sum(cm1))
#####from confusion matrix calculate accuracy
accuracy0=(cm0[0,0]+cm0[1,1])/total0
print ('Accuracy Train : ', accuracy0)

accuracy1=(cm1[0,0]+cm1[1,1])/total1
print ('Accuracy Test : ', accuracy1)

sensitivity1 = cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity : ', sensitivity1 )

specificity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity : ', specificity1)

KappaValue=cohen_kappa_score(y_val, rfc_val_pred)
print("Kappa Value :",KappaValue)
AUC=roc_auc_score(y_val, rfc_val_pred)
print("AUC          :",AUC)


print("F1-Score Train",f1_score(y_train,rfc_train_pred))
print("F1-Score Validation : ",f1_score(y_val, rfc_val_pred))
```

Figure 20: Random Forest Evaluation

```
Confusion Matrix Train :
 [[ 319   35]
 [ 395 3038]]
Confusion Matrix Test:
 [[ 124   28]
 [ 188 1283]]
Accuracy Train :  0.8864536572484817
Accuracy Test :  0.866913123844732
Sensitivity :  0.8157894736842105
Specificity :  0.8721957851801495
Kappa Value : 0.4674031940494422
AUC         : 0.8439926294321801
F1-Score Train 0.5973782771535582
F1-Score Validation :  0.5344827586206896
```

Figure 21: Random Forest Evaluation Result

4.3 Autoencoder

```
predictions_unseen=autoencoder.predict(test_pca[:,:29])
predictions_unseen.shape
```

Figure 22: Prediction on Unseen data

```
submission_AutoEncoder=pd.DataFrame({"Provider":Test_category_removed_groupedbyProv_PF.Provider})
submission_AutoEncoder['PotentialFraud']=AE_Labels
submission_AutoEncoder.head(16)
```

Figure 23: Potential Fraud with Autoencoder