Injury Prediction in Mining Industry through Applied Machine Learning Approaches
Configuration Manual

MSc Research Project
Data Analytics

Akash Manjunatha
Student ID: x21141797

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akash Manjunatha
Student ID:	x21141797
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Christian Horn
Submission Due Date:	01/02/2023
Project Title:	Injury Prediction in Mining Industry through Applied Machine Learning Approaches Configuration Manual
Word Count:	1099
Page Count:	8

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:	Akash Manjunatha
Date:	28th January 2023

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):	

Injury Prediction in Mining Industry through Applied Machine Learning Approaches Configuration Manual

Akash Manjunatha
x21141797

1 Introduction

Detailed instructions on how to setup a system or device are contained in a configuration manual. The manual's objective is to completely outline how to conduct the research study. Additionally, it specifies the machine configuration required to build and run the models. The steps entail setting up both the minimal setup recommended for a project to succeed as well as the applications and packages that are required.

2 Project Files Detail

The Google Collab program is used in this project for data preparation and exploration, while the Jupiter notebook is used for modelling and evaluation.

Case study-1:

- Case-study-1-file-1(Google Collab): fetch the .TXT file of accident and mines data (raw data) from google drive After exploration and processing, the results will be exported as. XLSX file.
- Case-study-1-file-2(Jupyter Notebook): Modelling and evaluation are done in this file using the results of file-1 .XLSX using jupyter notebook.

The same process is followed in Case study 2 and 3.

3 System Specification

A system specification is a written description of a system's technical specs and requirements. It often contains information about the system's parts, functionality, design, and other technical characteristics. Figure 2 depicts the system configuration used to run this project, while Figure 1 depicts the Google Collab specification.
The simplest configuration would only require 8 GB of RAM, which would be adequate to provide the desired results but would require little longer processing times. However, the suggested configuration ensures that the code runs without any problems.

Google Collab	
Processor	2-core Xenon 2.2GHz
The GPU Instance	250GB
The RAM	13 GB
The Disk Space	33 GB
Max Lifetime of VM	12Hrs

Figure 1: Google Collab Specification

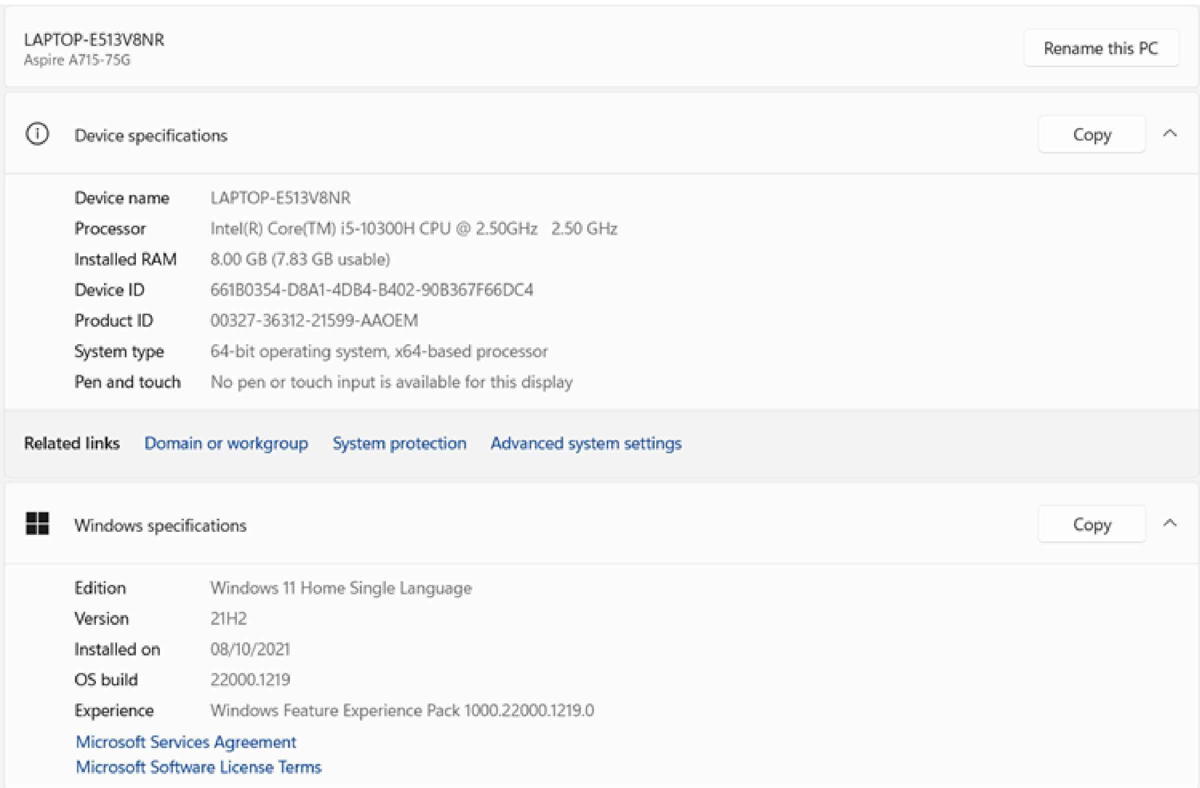


Figure 2: System Specification

4 Software Used

- Microsoft excel: Used for initial exploration.
- Tableau: Used for initial exploration and exploratory data analysis.
- Google Collab: Used for Exploration and processing.
- Jupyter Notebook: For the modelling and evaluation.

5 Download and Install

Based on the operating system, Python must first be installed; the installation of the most recent version is advised [1]. Python 3.10.2 for Windows 11 was downloaded and installed as the most recent version of the file.
A development environment is needed after Python has been installed to write, run, and view the output of code. Jupyter Notebook is the most common and user-friendly platform. It is bundled with the Python distribution, Anaconda [2], for which a suitable installation can be downloaded depending on the operating system. The Anaconda's dashboard is depicted in Figure 3, which also features pre-installed packages like the Jupyter notebook. The first step in developing Python code is to launch the Jupyter Notebook and create a new Python file.

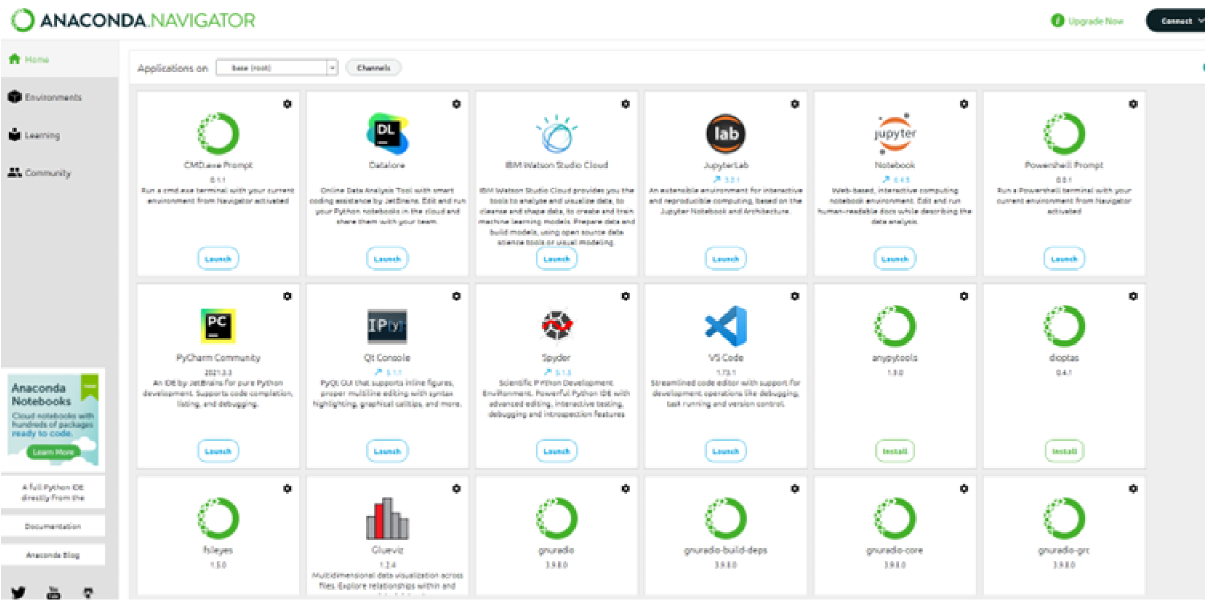


Figure 3: System Specification

It is simple to use Google Collab [3] Using a Google account, sign in and add a new file to the disk. Users have free access to computational tools like GPUs and TPUs through Google Colab, which can be utilized to execute computationally intensive tasks

[1]https://www.python.org/downloads/
[2]https://problemsolvingwithpython.com/01-Orientation/01.03-Installing-Anaconda-on-Windows/
[3]https://colab.research.google.com/

like machine learning model training (Bisong; 2019). allows users to leverage the cloud to run and utilize Jupyter notebooks

6 Project Development

After completing all these steps, you can launch the Jupyter notebook or Collab, click the new button on top of the file open, and load the scripted file using the file reference provided in the code section, you will have the option to run all of them simultaneously or each cell individually. The command "pip install package-name" should be used if a package needs to be installed.

6.1 Importing Library

The packages used in the project are displayed in Figure 4. The cloud platform comes with several necessary libraries already installed. If necessary, additional libraries should be imported. Since the PySpark is used in the processing to install PySpark on the system, Figure 5 steps should be followed

```
#installing all required packages for the project
from numpy import where
import numpy as np
import pandas as pd
from sklearn import preprocessing
from pandas import DataFrame

#installing the pyspark packages for the further analysis
from pyspark.sql import SparkSession, Window
from pyspark.sql import functions as F
from pyspark.sql.functions import regexp_extract
from pyspark.sql.types import IntegerType
from pyspark.sql.types import StringType
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches
from pyspark.sql.functions import col,isnan, when, count
from pyspark.sql.functions import *


from collections import Counter
from sklearn.datasets import make_classification
from matplotlib import pyplot
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

from time import time
from packaging import version

from matplotlib import pyplot
from matplotlib import pyplot as plt
import seaborn as sns
import missingno as msno

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from scipy.special import boxcox1p

from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.impute import KNNImputer
from sklearn.feature_selection import VarianceThreshold
from scipy import stats
from pylab import *

from sklearn.inspection import permutation_importance
from sklearn.feature_selection import RFE

import matplotlib.patches as mpatches
from sklearn.feature_selection import SelectKBest, chi2

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,plot_confusion_matrix,plot_roc_curve
from sklearn import linear_model
from keras import models, layers
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import NearestNeighbors
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
```

Figure 4: Packages used in the Project

1. Install Java on your computer by going to the Java website and clicking on the "Download" button to download the latest version of Java.
2. Set the JAVA_HOME environment variable on your system to point to the location where you installed Java.
3. Install the latest version of Apache Spark on your system by following the instructions on the Apache Spark website.
4. Set the SPARK_HOME environment variable on your system to point to the location where you installed Apache Spark.
5. Install PySpark on your system by using the following command. “pip install pyspark”

Figure 5: Procedure to Install PySpark

6.2 Importing Files

```
[ ] #Mounting drive data to Google colab
    from google.colab import drive
    drive.mount('/content/gdrive')

    Mounted at /content/gdrive

[ ] #fetching the accident data from the txt file.

    df_accident =  spark.read.format("csv").option("delimiter","|").option("header","true").load('/content/gdrive/My Drive/Accidents.txt')
```

Figure 6: Procedure to Mount and Fetch Document in Google Collab

```
In [2]: import pandas as pd

        DF_15 = pd.read_excel(r'C:\Users\akash\Desktop\ANALYSIS\all_category.xlsx')
```

Figure 7: Procedure to Fetch the File in Jupyter Notebook

In this research project, File-1 is processed in Google Collab. To run those, the following procedure must be followed. Figure 6 shows how to mount the Google drive to the collab. Only after one can fetch the data that has been put in the drive, the path should be set to the location where the file is. File-2 modeling and evaluation are done in jupyter notebook from the exported. .xlsx file of file-1. Figure 7 demonstrates how to get the excel file in the notebook. The path should be set to the local directory where the file is placed.

6.3 Processing

- Treatment of Missing Values: Missing values are imputed using the KNN imputer.
- Null Values handling: The utilities IsNull() and sum() are used to check for null values, and the duplicated () function is used to check for duplicates.
- Outliers Treatment: Only the most extreme outliers are revealed by using box graphs. are removed while keeping an eye out for data loss.
- Normalization: Box Cox and Max Scalar techniques are used to normalize the data after normalization graphs cheked.

- Feature selection: There are four steps in it: a determination of whether Constance features are present, a Pearson correlation analysis, a statistical analysis, and recursive feature deletion.
  The packages or libraries to perform the above tasks are shown in the Figure 4.

6.4 Modeling

Before modeling, the significant predictive variables are retrieved from the recursive feature elimination and loaded to the x-data and y-data functions with the target variable. If there is an imbalance in the data, smote will be used to balance it, and then the test train is split and fed into the models. The process is illustrated in Figure 8.

```
In [ ]: x_data = DF_15[X_train_rfe_model_1.columns]
        y_data = DF_15['DEGREE_INJURY_CD']

In [22]: # summarize class distribution
         counter = Counter(y_data)
         print("counter",counter)

         counter Counter({6: 94957, 4: 83460, 1: 2934})

In [23]: #oversampling using the SMOTE
         oversample = SMOTE(random_state = 42)
         x_data, y_data = oversample.fit_resample(x_data, y_data)

In [24]: # summarize class distribution
         counter = Counter(y_data)
         print("counter",counter)

         counter Counter({6: 94957, 4: 94957, 1: 94957})

In [25]: #test and train split
         x_train, x_test, y_train, y_test = train_test_split(x_data, y_data ,test_size = 0.3, random_state=0)
```

Figure 8: Code Snippet of Early Stage of Modeling.

In this study, five machine learning algorithms, and one deep learning algorithm are used. Each machine learning algorithm uses the following techniques, and the procedure should be repeated for other algorithms.

6.4.1 Model Without Tuning

Initially, each model is run through without any tuning. After this test and train results are evaluated, Figure 9 shows the random forest model running without tuning and the test-train code snippet.

6.4.2 Model With Tuning

The best parameter will be selected when these models are tuned with the various parameters; the best parameter selected by the grid-search CV is shown in Figure 10 The model is fed with n estimators=64, max depth=6, bootstrap=True, min samples leaf=1, min samples split=5, and the train test results of the same are checked. The packages or libraries to run the above tasks are displayed in Figure 4.

Model 1: Random Forest

1.1 without tuning

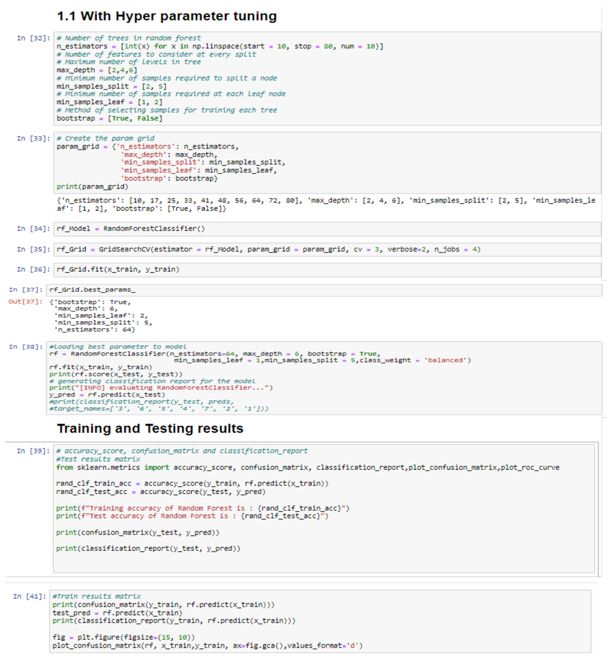
```
In [28]: rf = RandomForestClassifier()
         rf.fit(x_train, y_train)
         print(rf.score(x_test, y_test))
         # generating classification report for the model
         print("[INFO] evaluating RandomForestClassifier...")
         y_pred = rf.predict(x_test)
```

Training and Testing results

```
In [29]: #Test results matrix
         # accuracy_score, confusion_matrix and classification_report

         from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,plot_confusion_matrix,plot_roc_curve

         rand_clf_train_acc = accuracy_score(y_train, rf.predict(x_train))
         rand_clf_test_acc = accuracy_score(y_test, y_pred)

         print(f"Training accuracy of Random Forest is : {rand_clf_train_acc}")
         print(f"Test accuracy of Random Forest is : {rand_clf_test_acc}")

         print(confusion_matrix(y_test, y_pred))
         fig = plt.figure(figsize=(15, 10))

         print(classification_report(y_test, y_pred))
         plot_confusion_matrix(rf, x_test,y_test, ax=fig.gca(),values_format='d')
         #plot_roc_curve(rf, x_test,y_test)
```

```
In [30]: #Train results matrix

         print(confusion_matrix(y_train, rf.predict(x_train)))
         test_pred = rf.predict(x_train)
         print(classification_report(y_train, rf.predict(x_train)))
         fig = plt.figure(figsize=(15, 10))
         plot_confusion_matrix(rf, x_train,y_train, ax=fig.gca(),values_format= 'd')
```

Figure 9: Code Snippet of Model Without Tuning



1.1 With Hyper parameter tuning

```
In [32]: # Number of trees in random forest
         n_estimators = [int(x) for x in np.linspace(start = 10, stop = 80, num = 10)]
         # Number of features to consider at every split
         # Maximum number of levels in tree
         max_depth = [2,4,6]
         # Minimum number of samples required to split a node
         min_samples_split = [2, 5]
         # Minimum number of samples required at each leaf node
         min_samples_leaf = [1, 2]
         # Method of selecting samples for training each tree
         bootstrap = [True, False]

In [33]: # Create the param grid
         param_grid = {'n_estimators': n_estimators,
                       'max_depth': max_depth,
                       'min_samples_split': min_samples_split,
                       'min_samples_leaf': min_samples_leaf,
                       'bootstrap': bootstrap}
         print(param_grid)

         {'n_estimators': [10, 17, 25, 33, 41, 48, 56, 64, 72, 80], 'max_depth': [2, 4, 6], 'min_samples_split': [2, 5], 'min_samples_le
         af': [1, 2], 'bootstrap': [True, False]}

In [34]: rf_Model = RandomForestClassifier()

In [35]: rf_Grid = GridSearchCV(estimator = rf_Model, param_grid = param_grid, cv = 3, verbose=2, n_jobs = 4)

In [36]: rf_Grid.fit(x_train, y_train)

In [37]: rf_Grid.best_params_
Out[37]: {'bootstrap': True,
          'max_depth': 6,
          'min_samples_leaf': 2,
          'min_samples_split': 5,
          'n_estimators': 64}

In [38]: #Loading best parameter to model
         rf = RandomForestClassifier(n_estimators=64, max_depth = 6, bootstrap = True,
                                     min_samples_leaf = 1,min_samples_split = 5,class_weight = 'balanced')
         rf.fit(x_train, y_train)
         print(rf.score(x_test, y_test))
         # generating classification report for the model
         print("[INFO] evaluating RandomForestClassifier...")
         y_pred = rf.predict(x_test)
         #print(classification_report(y_test, preds,
         #target_names=['3', '6', '5', '4', '7', '2', '1']))
```

Training and Testing results

```
In [39]: # accuracy_score, confusion_matrix and classification_report
         #Test results matrix
         from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,plot_confusion_matrix,plot_roc_curve

         rand_clf_train_acc = accuracy_score(y_train, rf.predict(x_train))
         rand_clf_test_acc = accuracy_score(y_test, y_pred)

         print(f"Training accuracy of Random Forest is : {rand_clf_train_acc}")
         print(f"Test accuracy of Random Forest is : {rand_clf_test_acc}")

         print(confusion_matrix(y_test, y_pred))

         print(classification_report(y_test, y_pred))

In [41]: #Train results matrix
         print(confusion_matrix(y_train, rf.predict(x_train)))
         test_pred = rf.predict(x_train)
         print(classification_report(y_train, rf.predict(x_train)))

         fig = plt.figure(figsize=(15, 10))
         plot_confusion_matrix(rf, x_train,y_train, ax=fig.gca(),values_format='d')
```

Figure 10: Code Snippet of Model With Tuning

References

Bisong, E. (2019). *Building machine learning and deep learning models on Google cloud platform: A comprehensive guide for beginners*, Apress.