Configuration Manual

MSc Research Project
Data Analytics

Megan Farrelly
Student ID: x19144440

School of Computing
National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Megan Farrelly
Student ID:	x19144440
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Vladimir Milosavljevic
Submission Due Date:	19th September 2022
Project Title:	Configuration Manual
Word Count:	1,692
Page Count:	11

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:	
Date:	17th September 2022

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

Megan Farrelly
x19144440

1 Introduction

This configuration manual describes in detail the steps required to replicate the experiments undertaken to fulfill the research titled "A Classification Approach to Identifying Female Victims of Intimate Partner Violence in Europe and the US." This manual documents the software and tools required, and where appropriate, how to implement these tools. While popular packages were used, the code was highly customised through the creation of functions that were used throughout the research to produce outputs that facilitated analysis.

2 Hardware Overview

All research was carried out on a 2020 MacBook Air using Mac OS Monterey Version 12.5 operating system, with one 1.1 GHz Quad-Core Intel Core i5 processor, four cores with 6 MB of L3 Cache, 8 GB of RAM, 500 GB of storage and Intel Iris Plus Graphics 1536 MB GPU.

3 Environment

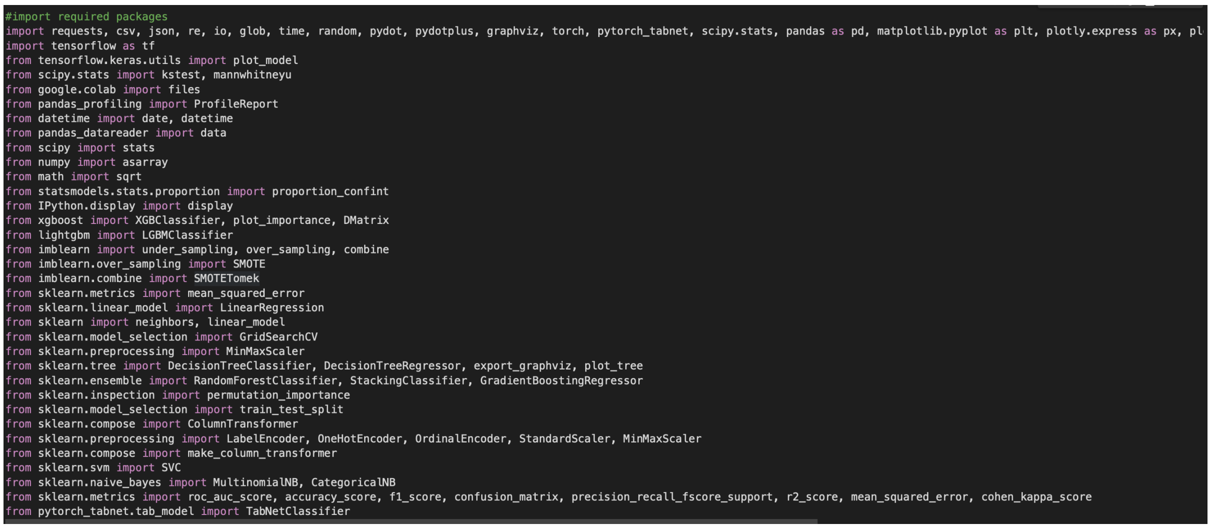
All experiments were undertaken in the Python programming language. These were completed by creating and running an ipynb file in Google Colab. Google Colab was chosen as the environment for running the experiments due to the cloud based nature of the platform, the option to use a GPU hardware accelerator and the fact that most Python packages did not need to be installed to a local drive. Given the large file sizes and large memory and disc space requirements, this environment was preferable. All code required to complete the experiments were written in an ipynb file, which was saved to a Google Drive account. An account is required to access both Google Drive and Colab.

4 Implementation

This section describes in detail the implementation of the code required to complete the experiments and produce the results in the accompanying Research Project. The most important pieces of code for replication is described through the use of screenshots of code, inputs and outputs

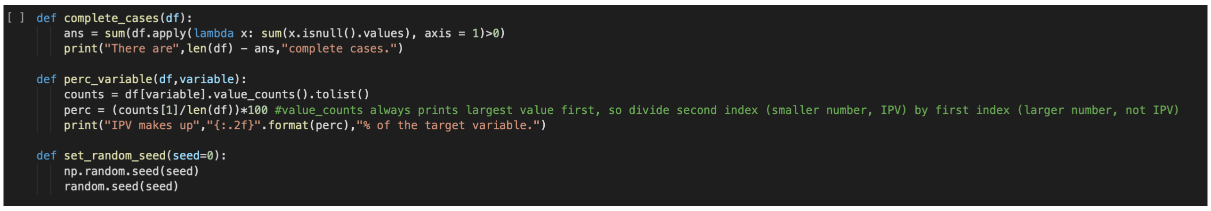
4.1 Establishment of Environment

A number of packages are required to complete research. These are imported at the beginning of each session (Figure 1). The pandas, numpy, scikit-learn, tensorflow and plotly packages are the most commonly used and heavily relied on packages for data manipulation, cleansing, transformation, modelling and graphing. Following this, the functions that are used throughout the research are loaded (Figure 2). This cell that is loaded at the beginning of each session contains 22 functions that are repeatedly called throughout the research. This allowed for quick manipulation of the code. For example, if a change was made to the code that determined the outputs of each model, the change was easily made for all models. Once the required packages and functions are loaded, the data can be imported for manipulation and testing.



```
#import required packages
import requests, csv, json, re, io, glob, time, random, pydot, pydotplus, graphviz, torch, pytorch_tabnet, scipy.stats, pandas as pd, matplotlib.pyplot as plt, plotly.express as px, pl
import tensorflow as tf
from tensorflow.keras.utils import plot_model
from scipy.stats import kstest, mannwhitneyu
from google.colab import files
from pandas_profiling import ProfileReport
from datetime import date, datetime
from pandas_datareader import data
from scipy import stats
from numpy import asarray
from math import sqrt
from statsmodels.stats.proportion import proportion_confint
from IPython.display import display
from xgboost import XGBClassifier, plot_importance, DMatrix
from lightgbm import LGBMClassifier
from imblearn import under_sampling, over_sampling, combine
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTETomek
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn import neighbors, linear_model
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz, plot_tree
from sklearn.ensemble import RandomForestClassifier, StackingClassifier, GradientBoostingRegressor
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, OrdinalEncoder, StandardScaler, MinMaxScaler
from sklearn.compose import make_column_transformer
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB, CategoricalNB
from sklearn.metrics import roc_auc_score, accuracy_score, f1_score, confusion_matrix, precision_recall_fscore_support, r2_score, mean_squared_error, cohen_kappa_score
from pytorch_tabnet.tab_model import TabNetClassifier
```

Figure 1: The packages required to complete the research.



```
[ ] def complete_cases(df):
        ans = sum(df.apply(lambda x: sum(x.isnull().values), axis = 1)>0)
        print("There are",len(df) - ans,"complete cases.")

    def perc_variable(df,variable):
        counts = df[variable].value_counts().tolist()
        perc = (counts[1]/len(df))*100 #value_counts always prints largest value first, so divide second index (smaller number, IPV) by first index (larger number, not IPV)
        print("IPV makes up","{:.2f}".format(perc),"% of the target variable.")

    def set_random_seed(seed=0):
        np.random.seed(seed)
        random.seed(seed)
```

Figure 2: An example of the functions that are called throughout the research.

4.2 Data Manipulation

All practical experiments are carried out in two parts. All data are uploaded and the European data, or European Union Agency for Fundamental Rights (FRA)[1] dataset, is explored, cleansed, transformed and modelled first. The same process is followed for the USA data, or National Crime Victimisation Survey (NCVS)[2] dataset, following this. To save memory, only the variables identified as of interest prior to implementation are

[1]https://bit.ly/3sVmbeg
[2]https://bit.ly/3t5B8u7

uploaded. To upload the data, the FRA and NCVS datasets in TAB and TSV formats respectively, are zipped and added to Google Drive (Figure 3). To replicate this research without changing the directory, it is recommended that the zipped file be added to the Google Drive homepage, "My Drive," highlighted in red in Figure 3, and not to any subsequent folders. The data are then imported into Google Colab by mounting the session to Google Drive and unzipping the files (Figure 4). Following this, cleansing can begin. The variables are renamed to a descriptor to enable intuitive analysis (Figure 5). Redundant values such as "Refused" and "No answer" are replaced with nulls. The number of complete cases are output (Figure 6). Data are further manipulated and variables with a high number of null values are removed until an acceptable number of complete cases remain for modelling. The rows where nulls exist are removed, and the target variable is created by concatenating existing columns such as physical and sexual violence and the percentage that IPV makes up of the target variable is output (Figure 7).

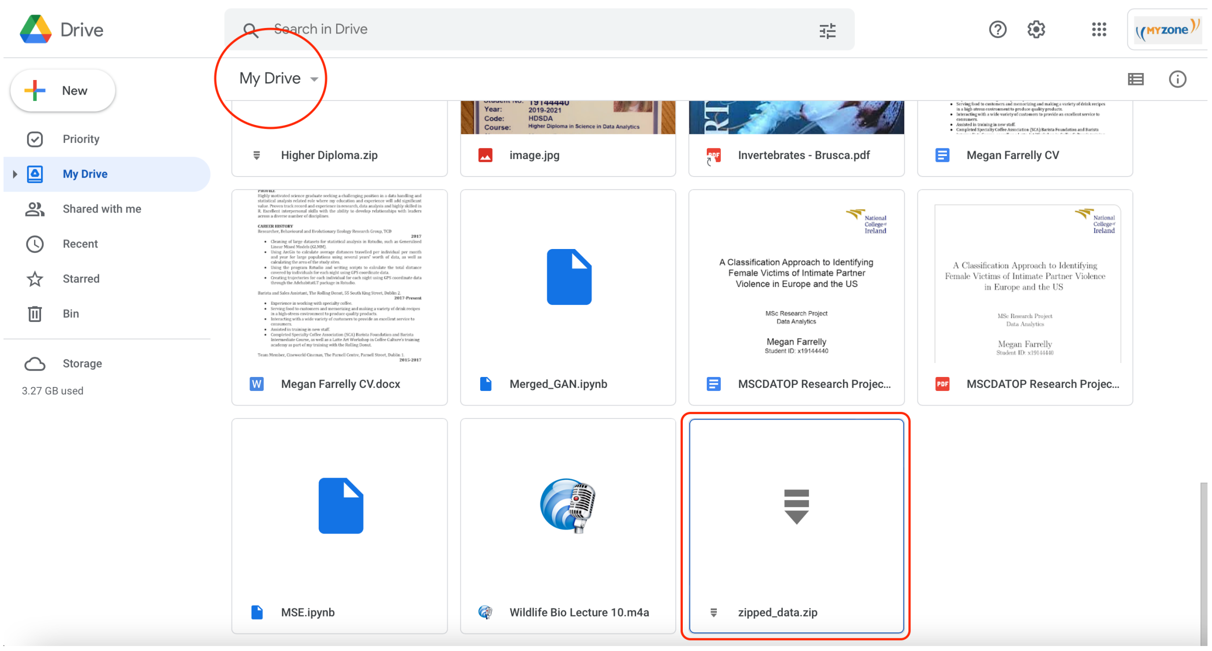
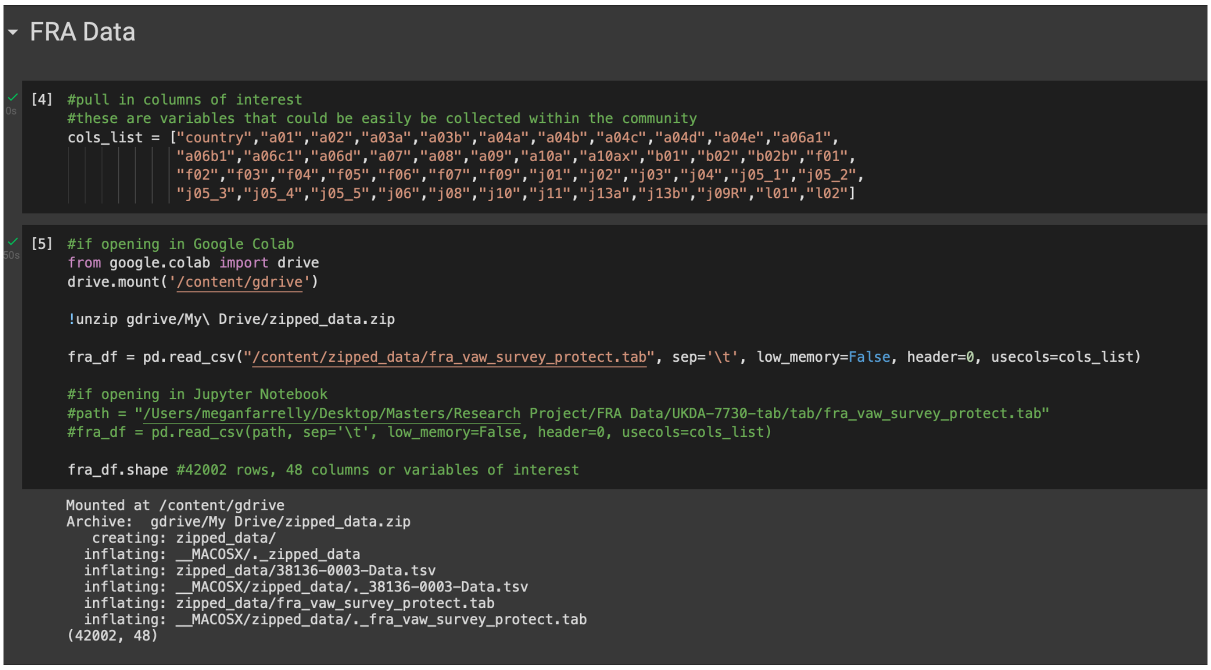


Figure 3: The zipped data files uploaded to Google Drive.

Following cleansing, the data are explored. Functions that output skewness, kurtosis, a histogram describing the distribution of numerical variables and a correlation plot are called. The final step before transformation is to rename the values of each variable to descriptors as these will later become the feature names following encoding (Figure 8). Some values are also consolidated in this step, for better analysis and to reduce the number of small categories. For transformation to begin, the data are split into training and test sets first to avoid data leakage[3]. These subsets are passed to a function that transforms the data by ordinal or one hot encoding, or normalisation dependent on the data type[4]. SMOTE Tomek is also applied to training sets only to solve class imbalance within the target variable (Figure 9). Finally, the test sets are label encoded and four transformed subsets are output for modelling. The function that completes this transformation is shown in Figure 10.

[3]https://machinelearningmastery.com/data-preparation-without-data-leakage/
[4]https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/



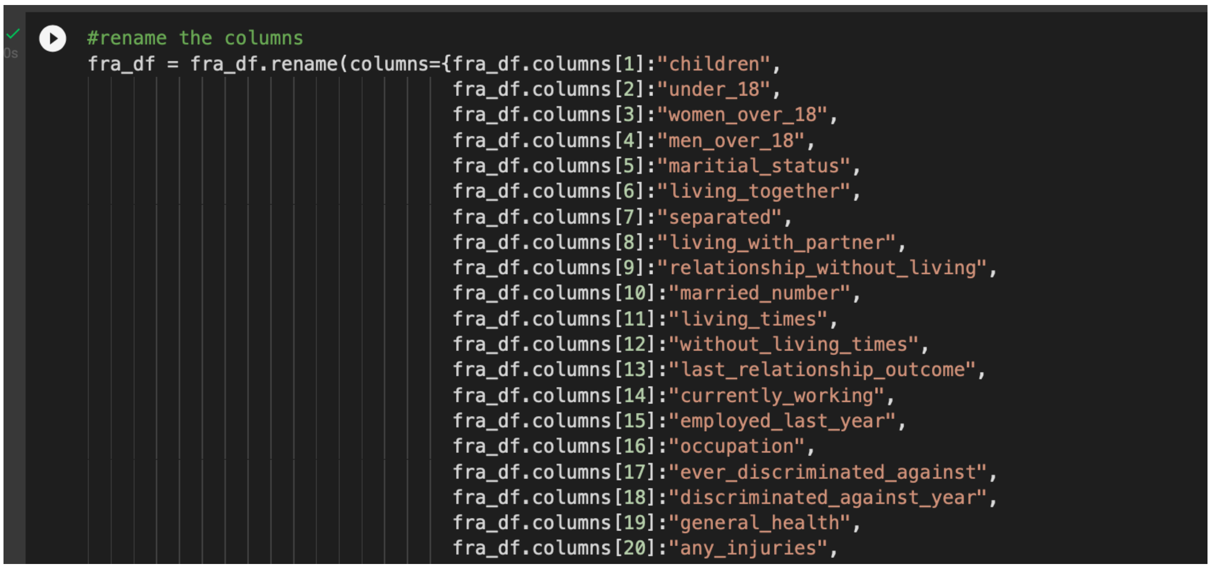
FRA Data

```
[4] #pull in columns of interest
    #these are variables that could be easily be collected within the community
    cols_list = ["country","a01","a02","a03a","a03b","a04a","a04b","a04c","a04d","a04e","a06a1",
                 "a06b1","a06c1","a06d","a07","a08","a09","a10a","a10ax","b01","b02","b02b","f01",
                 "f02","f03","f04","f05","f06","f07","f09","j01","j02","j03","j04","j05_1","j05_2",
                 "j05_3","j05_4","j05_5","j06","j08","j10","j11","j13a","j13b","j09R","l01","l02"]
```

```
[5] #if opening in Google Colab
    from google.colab import drive
    drive.mount('/content/gdrive')

    !unzip gdrive/My\ Drive/zipped_data.zip

    fra_df = pd.read_csv("/content/zipped_data/fra_vaw_survey_protect.tab", sep='\t', low_memory=False, header=0, usecols=cols_list)

    #if opening in Jupyter Notebook
    #path = "/Users/meganfarrelly/Desktop/Masters/Research Project/FRA Data/UKDA-7730-tab/tab/fra_vaw_survey_protect.tab"
    #fra_df = pd.read_csv(path, sep='\t', low_memory=False, header=0, usecols=cols_list)

    fra_df.shape #42002 rows, 48 columns or variables of interest
```

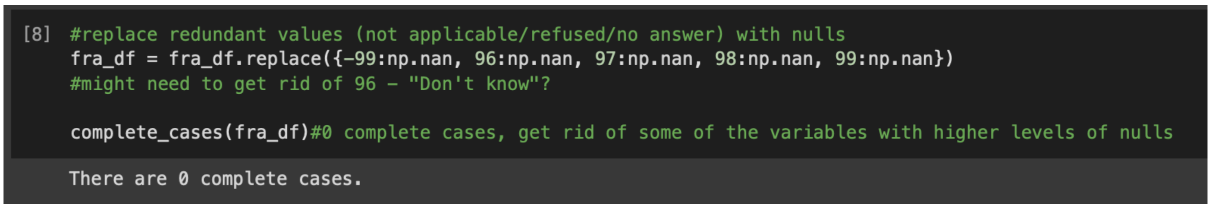
```
Mounted at /content/gdrive
Archive:  gdrive/My Drive/zipped_data.zip
   creating: zipped_data/
  inflating: __MACOSX/._zipped_data
  inflating: zipped_data/38136-0003-Data.tsv
  inflating: __MACOSX/zipped_data/._38136-0003-Data.tsv
  inflating: zipped_data/fra_vaw_survey_protect.tab
  inflating: __MACOSX/zipped_data/._fra_vaw_survey_protect.tab
(42002, 48)
```

Figure 4: The mounting process of Google Drive to Colab to unzip and upload the data.



```
#rename the columns
fra_df = fra_df.rename(columns={fra_df.columns[1]:"children",
                                fra_df.columns[2]:"under_18",
                                fra_df.columns[3]:"women_over_18",
                                fra_df.columns[4]:"men_over_18",
                                fra_df.columns[5]:"maritial_status",
                                fra_df.columns[6]:"living_together",
                                fra_df.columns[7]:"separated",
                                fra_df.columns[8]:"living_with_partner",
                                fra_df.columns[9]:"relationship_without_living",
                                fra_df.columns[10]:"married_number",
                                fra_df.columns[11]:"living_times",
                                fra_df.columns[12]:"without_living_times",
                                fra_df.columns[13]:"last_relationship_outcome",
                                fra_df.columns[14]:"currently_working",
                                fra_df.columns[15]:"employed_last_year",
                                fra_df.columns[16]:"occupation",
                                fra_df.columns[17]:"ever_discriminated_against",
                                fra_df.columns[18]:"discriminated_against_year",
                                fra_df.columns[19]:"general_health",
                                fra_df.columns[20]:"any_injuries",
```

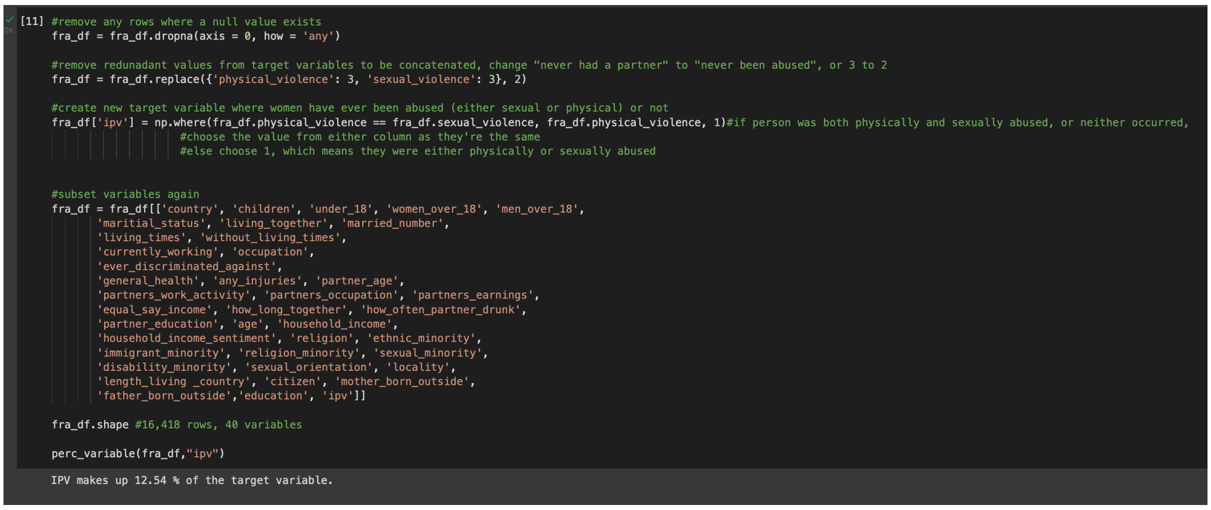
Figure 5: The variables for each dataset are renamed from their codes.



```
[8] #replace redundant values (not applicable/refused/no answer) with nulls
    fra_df = fra_df.replace({-99:np.nan, 96:np.nan, 97:np.nan, 98:np.nan, 99:np.nan})
    #might need to get rid of 96 - "Don't know"?

    complete_cases(fra_df)#0 complete cases, get rid of some of the variables with higher levels of nulls
```

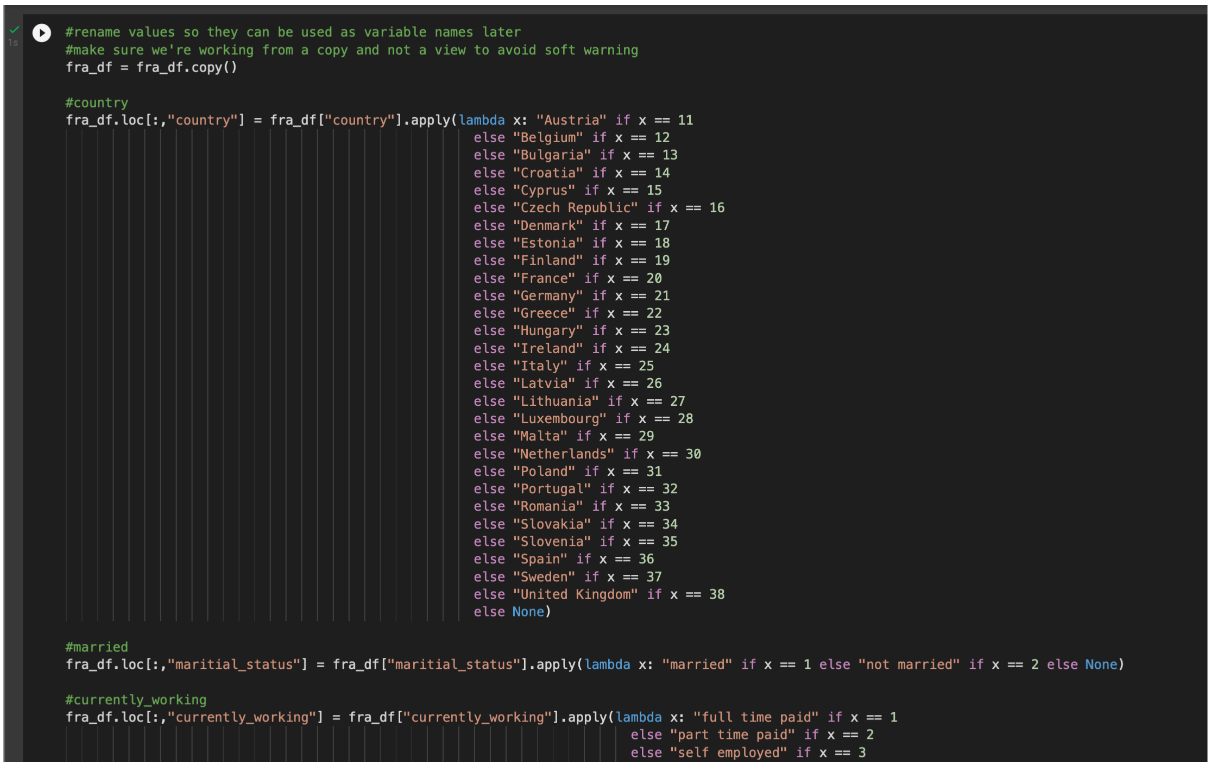
```
There are 0 complete cases.
```

Figure 6: Redundant values such as “No answer” are replaced with nulls and the number of complete cases is output.



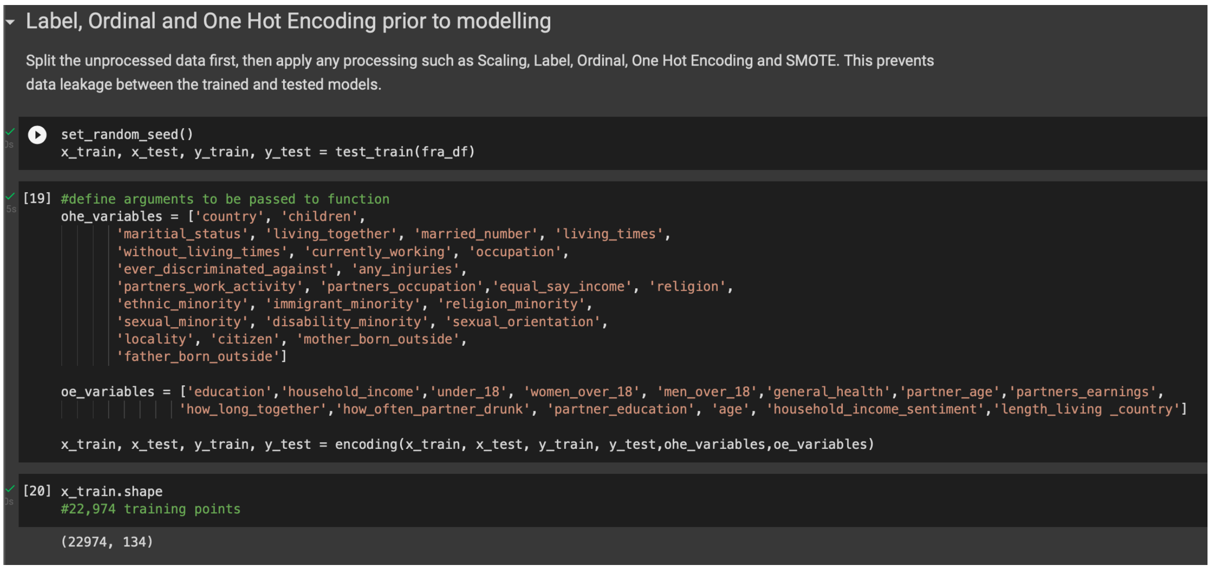
```
[11] #remove any rows where a null value exists
     fra_df = fra_df.dropna(axis = 0, how = 'any')

     #remove redunadant values from target variables to be concatenated, change "never had a partner" to "never been abused", or 3 to 2
     fra_df = fra_df.replace({'physical_violence': 3, 'sexual_violence': 3}, 2)

     #create new target variable where women have ever been abused (either sexual or physical) or not
     fra_df['ipv'] = np.where(fra_df.physical_violence == fra_df.sexual_violence, fra_df.physical_violence, 1)#if person was both physically and sexually abused, or neither occurred,
                                #choose the value from either column as they're the same
                                #else choose 1, which means they were either physically or sexually abused


     #subset variables again
     fra_df = fra_df[['country', 'children', 'under_18', 'women_over_18', 'men_over_18',
            'maritial_status', 'living_together', 'married_number',
            'living_times', 'without_living_times',
            'currently_working', 'occupation',
            'ever_discriminated_against',
            'general_health', 'any_injuries', 'partner_age',
            'partners_work_activity', 'partners_occupation', 'partners_earnings',
            'equal_say_income', 'how_long_together', 'how_often_partner_drunk',
            'partner_education', 'age', 'household_income',
            'household_income_sentiment', 'religion', 'ethnic_minority',
            'immigrant_minority', 'religion_minority', 'sexual_minority',
            'disability_minority', 'sexual_orientation', 'locality',
            'length_living _country', 'citizen', 'mother_born_outside',
            'father_born_outside','education', 'ipv']]

     fra_df.shape #16,418 rows, 40 variables

     perc_variable(fra_df,"ipv")

     IPV makes up 12.54 % of the target variable.
```

Figure 7: The final step in cleansing, when rows where nulls exist are removed, the target variable is created and the percentage that IPV makes of the target variable is printed.



```
#rename values so they can be used as variable names later
#make sure we're working from a copy and not a view to avoid soft warning
fra_df = fra_df.copy()

#country
fra_df.loc[:,"country"] = fra_df["country"].apply(lambda x: "Austria" if x == 11
                                                    else "Belgium" if x == 12
                                                    else "Bulgaria" if x == 13
                                                    else "Croatia" if x == 14
                                                    else "Cyprus" if x == 15
                                                    else "Czech Republic" if x == 16
                                                    else "Denmark" if x == 17
                                                    else "Estonia" if x == 18
                                                    else "Finland" if x == 19
                                                    else "France" if x == 20
                                                    else "Germany" if x == 21
                                                    else "Greece" if x == 22
                                                    else "Hungary" if x == 23
                                                    else "Ireland" if x == 24
                                                    else "Italy" if x == 25
                                                    else "Latvia" if x == 26
                                                    else "Lithuania" if x == 27
                                                    else "Luxembourg" if x == 28
                                                    else "Malta" if x == 29
                                                    else "Netherlands" if x == 30
                                                    else "Poland" if x == 31
                                                    else "Portugal" if x == 32
                                                    else "Romania" if x == 33
                                                    else "Slovakia" if x == 34
                                                    else "Slovenia" if x == 35
                                                    else "Spain" if x == 36
                                                    else "Sweden" if x == 37
                                                    else "United Kingdom" if x == 38
                                                    else None)

#married
fra_df.loc[:,"maritial_status"] = fra_df["maritial_status"].apply(lambda x: "married" if x == 1 else "not married" if x == 2 else None)

#currently_working
fra_df.loc[:,"currently_working"] = fra_df["currently_working"].apply(lambda x: "full time paid" if x == 1
                                                                        else "part time paid" if x == 2
                                                                        else "self employed" if x == 3
```

Figure 8: An example of the values for each variable being renamed to be used as feature names following later encoding.



Label, Ordinal and One Hot Encoding prior to modelling

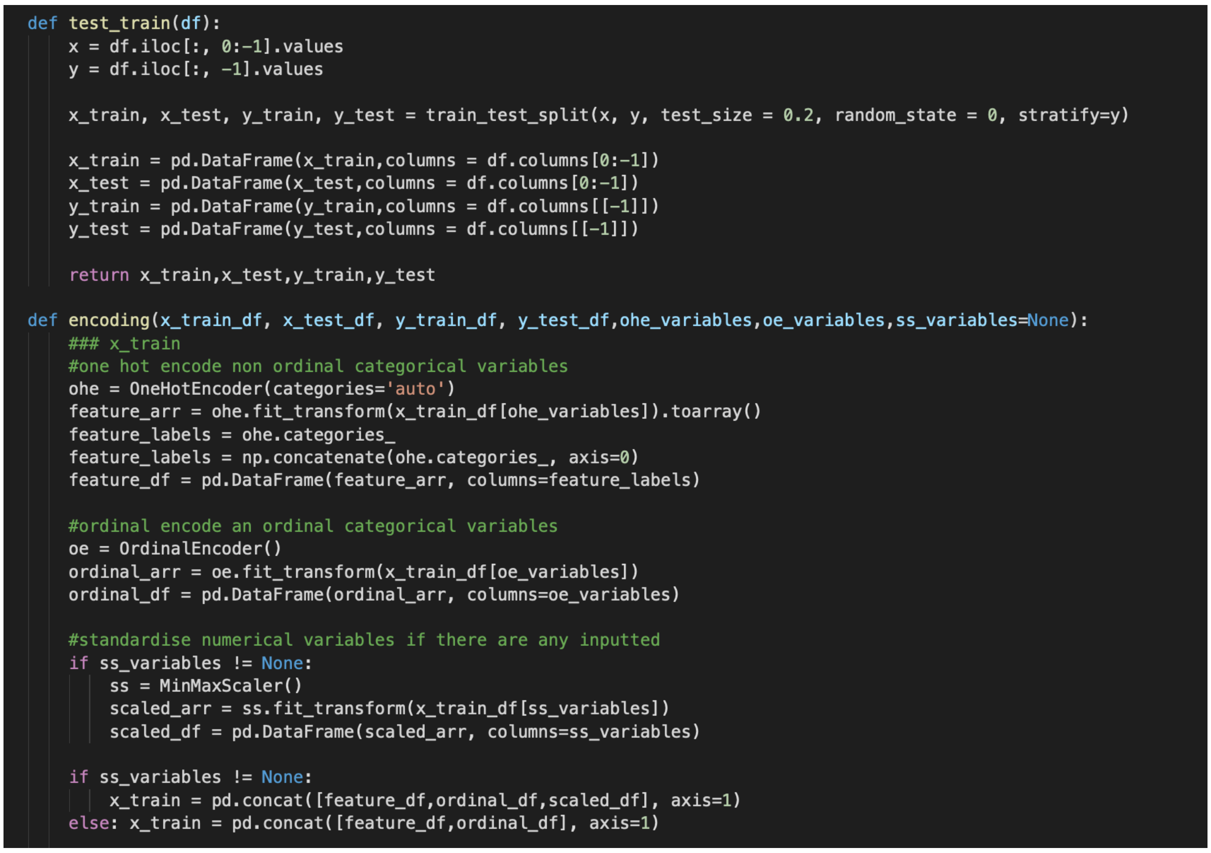
Split the unprocessed data first, then apply any processing such as Scaling, Label, Ordinal, One Hot Encoding and SMOTE. This prevents data leakage between the trained and tested models.

```
set_random_seed()
x_train, x_test, y_train, y_test = test_train(fra_df)
```

```
[19] #define arguments to be passed to function
     ohe_variables = ['country', 'children',
                 'maritial_status', 'living_together', 'married_number', 'living_times',
                 'without_living_times', 'currently_working', 'occupation',
                 'ever_discriminated_against', 'any_injuries',
                 'partners_work_activity', 'partners_occupation','equal_say_income', 'religion',
                 'ethnic_minority', 'immigrant_minority', 'religion_minority',
                 'sexual_minority', 'disability_minority', 'sexual_orientation',
                 'locality', 'citizen', 'mother_born_outside',
                 'father_born_outside']

     oe_variables = ['education','household_income','under_18', 'women_over_18', 'men_over_18','general_health','partner_age','partners_earnings',
                     'how_long_together','how_often_partner_drunk', 'partner_education', 'age', 'household_income_sentiment','length_living _country']

     x_train, x_test, y_train, y_test = encoding(x_train, x_test, y_train, y_test,ohe_variables,oe_variables)
```

```
[20] x_train.shape
     #22,974 training points

     (22974, 134)
```

Figure 9: The data are split into train and test sets, and the variables are passed to a function that transforms the data by ordinal or one hot encoding or normalisation, and applies SMOTE Tomek to training sets only.



```
def test_train(df):
    x = df.iloc[:, 0:-1].values
    y = df.iloc[:, -1].values

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0, stratify=y)

    x_train = pd.DataFrame(x_train,columns = df.columns[0:-1])
    x_test = pd.DataFrame(x_test,columns = df.columns[0:-1])
    y_train = pd.DataFrame(y_train,columns = df.columns[[-1]])
    y_test = pd.DataFrame(y_test,columns = df.columns[[-1]])

    return x_train,x_test,y_train,y_test

def encoding(x_train_df, x_test_df, y_train_df, y_test_df,ohe_variables,oe_variables,ss_variables=None):
    ### x_train
    #one hot encode non ordinal categorical variables
    ohe = OneHotEncoder(categories='auto')
    feature_arr = ohe.fit_transform(x_train_df[ohe_variables]).toarray()
    feature_labels = ohe.categories_
    feature_labels = np.concatenate(ohe.categories_, axis=0)
    feature_df = pd.DataFrame(feature_arr, columns=feature_labels)

    #ordinal encode an ordinal categorical variables
    oe = OrdinalEncoder()
    ordinal_arr = oe.fit_transform(x_train_df[oe_variables])
    ordinal_df = pd.DataFrame(ordinal_arr, columns=oe_variables)

    #standardise numerical variables if there are any inputted
    if ss_variables != None:
        ss = MinMaxScaler()
        scaled_arr = ss.fit_transform(x_train_df[ss_variables])
        scaled_df = pd.DataFrame(scaled_arr, columns=ss_variables)

    if ss_variables != None:
        x_train = pd.concat([feature_df,ordinal_df,scaled_df], axis=1)
    else: x_train = pd.concat([feature_df,ordinal_df], axis=1)
```

Figure 10: An example of the functions that transform the data ready for modelling.

4.3 Modelling

Following cleansing and transformation, modelling begins. Functions were created for each of the nine models so that modelling for each dataset could be easily completed with the ability to manipulate the inputs of each for optimisation (Figure 11). The function could then be called for either dataset with different specified parameters, and the same output would be produced each time (Figure 12). A function was also created that could be called within each model function to output the same metrics for each so performance could be easily compared (Figure 13). This function produces Accuracy, AUC and F1 score, along with a confusion matrix, by a number of methods, such as through scikit-learn functions, normal approximation, binomial proportion intervals and bootstrapped confidence intervals[5]. This was to decide which method was most appropriate for reporting error. Ultimately, bootstrapped confidence intervals are reported but these outputs are maintained to allow for comparisons (Pace, 2012). Each model function allowed for different parameters to be input for optimisation, fit the model to the training sets and made predictions from an unseen test set (Figure 11). The metrics function was then called to produce the output metrics by comparing the prediction set and unseen test target variable (Figure 11). Other models such as Random Forest, XGBoost, multilayer perceptron and TabNet also produced other outputs such as permutation and impurity based feature importance graphs, structure graphs, loss and accuracy graphs to monitor for overfitting (Figure 14, Figure 15). The function built to model a multilayer perceptron was created with the help of online tutorials[6]. The TabNet model was built with the aid of Arik and Pfister (2021).

The exact same process is followed to cleanse, transform and apply the same nine models to the NCVS data and produce the same outputs as is described above. The input parameters for each model are modified to optimise output. This is the main reasoning behind creating functions and loading them at the beginning of the file, as they are called multiple times throughout for either dataset.

Additional analysis was carried out on the FRA data to determine the rate of IPV by country. This is mainly completed through the use of plotly (Figure 16).

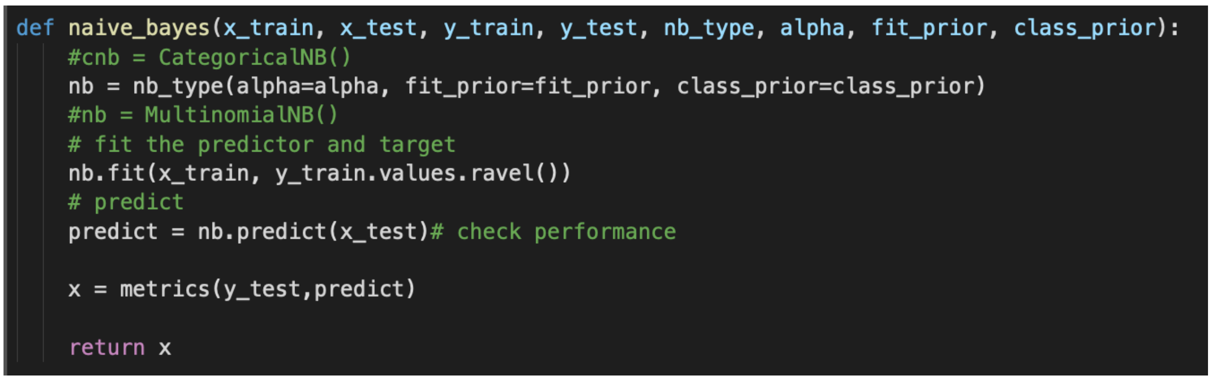
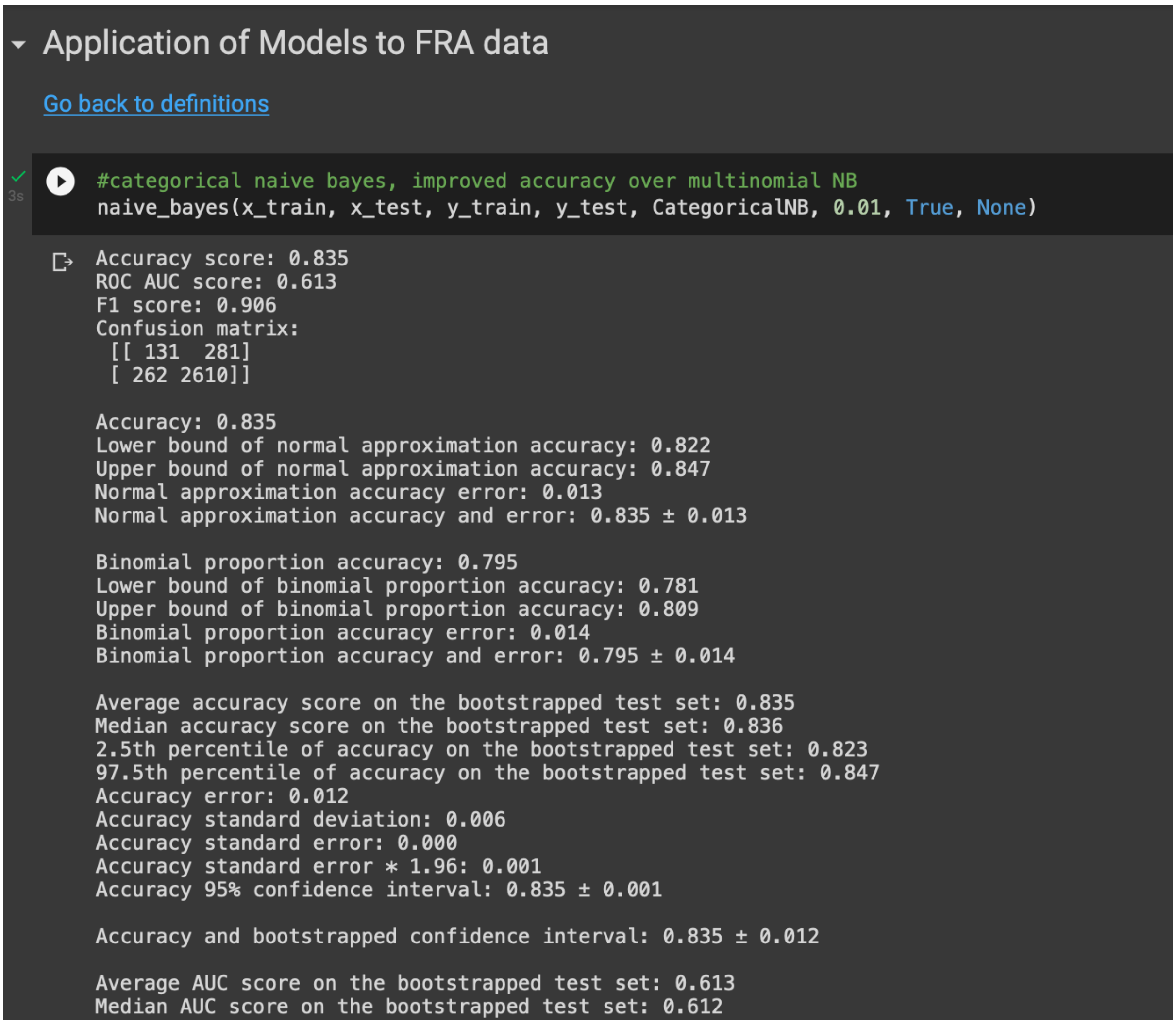

```
def naive_bayes(x_train, x_test, y_train, y_test, nb_type, alpha, fit_prior, class_prior):
    #cnb = CategoricalNB()
    nb = nb_type(alpha=alpha, fit_prior=fit_prior, class_prior=class_prior)
    #nb = MultinomialNB()
    # fit the predictor and target
    nb.fit(x_train, y_train.values.ravel())
    # predict
    predict = nb.predict(x_test)# check performance

    x = metrics(y_test,predict)

    return x
```

Figure 11: The Naïve Bayes modelling function, an example of the function created for each model.

[5]https://sebastianraschka.com/blog/2022/confidence-intervals-for-ml.html#method-4–confidence-intervals-from-retraining-models-with-different-random-seeds

[6]https://www.section.io/engineering-education/build-ann-with-keras/; https://machinelearningmastery.com/neural-networks-crash-course/



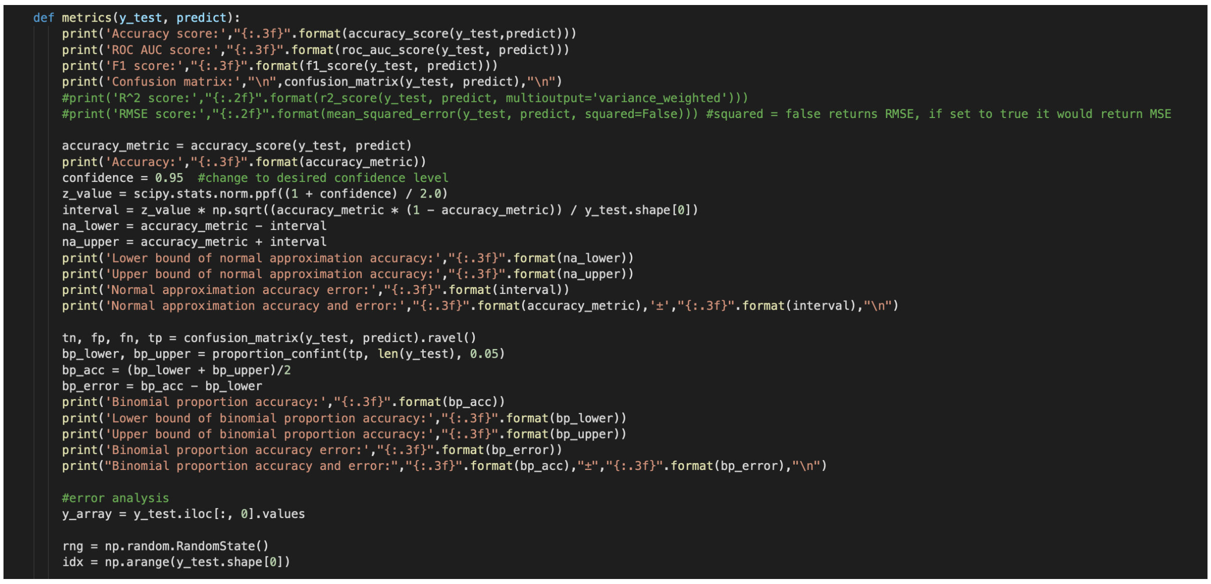
Application of Models to FRA data

Go back to definitions

```
#categorical naive bayes, improved accuracy over multinomial NB
naive_bayes(x_train, x_test, y_train, y_test, CategoricalNB, 0.01, True, None)
```

```
Accuracy score: 0.835
ROC AUC score: 0.613
F1 score: 0.906
Confusion matrix:
 [[ 131  281]
 [ 262 2610]]

Accuracy: 0.835
Lower bound of normal approximation accuracy: 0.822
Upper bound of normal approximation accuracy: 0.847
Normal approximation accuracy error: 0.013
Normal approximation accuracy and error: 0.835 ± 0.013

Binomial proportion accuracy: 0.795
Lower bound of binomial proportion accuracy: 0.781
Upper bound of binomial proportion accuracy: 0.809
Binomial proportion accuracy error: 0.014
Binomial proportion accuracy and error: 0.795 ± 0.014

Average accuracy score on the bootstrapped test set: 0.835
Median accuracy score on the bootstrapped test set: 0.836
2.5th percentile of accuracy on the bootstrapped test set: 0.823
97.5th percentile of accuracy on the bootstrapped test set: 0.847
Accuracy error: 0.012
Accuracy standard deviation: 0.006
Accuracy standard error: 0.000
Accuracy standard error * 1.96: 0.001
Accuracy 95% confidence interval: 0.835 ± 0.001

Accuracy and bootstrapped confidence interval: 0.835 ± 0.012

Average AUC score on the bootstrapped test set: 0.613
Median AUC score on the bootstrapped test set: 0.612
```

Figure 12: The function called to model Naïve Bayes and the output produced.



```
def metrics(y_test, predict):
    print('Accuracy score:',"{:.3f}".format(accuracy_score(y_test,predict)))
    print('ROC AUC score:',"{:.3f}".format(roc_auc_score(y_test, predict)))
    print('F1 score:',"{:.3f}".format(f1_score(y_test, predict)))
    print('Confusion matrix:',"\n",confusion_matrix(y_test, predict),"\n")
    #print('R^2 score:',"{:.2f}".format(r2_score(y_test, predict, multioutput='variance_weighted')))
    #print('RMSE score:',"{:.2f}".format(mean_squared_error(y_test, predict, squared=False))) #squared = false returns RMSE, if set to true it would return MSE

    accuracy_metric = accuracy_score(y_test, predict)
    print('Accuracy:',"{:.3f}".format(accuracy_metric))
    confidence = 0.95  #change to desired confidence level
    z_value = scipy.stats.norm.ppf((1 + confidence) / 2.0)
    interval = z_value * np.sqrt((accuracy_metric * (1 - accuracy_metric)) / y_test.shape[0])
    na_lower = accuracy_metric - interval
    na_upper = accuracy_metric + interval
    print('Lower bound of normal approximation accuracy:',"{:.3f}".format(na_lower))
    print('Upper bound of normal approximation accuracy:',"{:.3f}".format(na_upper))
    print('Normal approximation accuracy error:',"{:.3f}".format(interval))
    print('Normal approximation accuracy and error:',"{:.3f}".format(accuracy_metric),'±',"{:.3f}".format(interval),"\n")

    tn, fp, fn, tp = confusion_matrix(y_test, predict).ravel()
    bp_lower, bp_upper = proportion_confint(tp, len(y_test), 0.05)
    bp_acc = (bp_lower + bp_upper)/2
    bp_error = bp_acc - bp_lower
    print('Binomial proportion accuracy:',"{:.3f}".format(bp_acc))
    print('Lower bound of binomial proportion accuracy:',"{:.3f}".format(bp_lower))
    print('Upper bound of binomial proportion accuracy:',"{:.3f}".format(bp_upper))
    print('Binomial proportion accuracy error:',"{:.3f}".format(bp_error))
    print("Binomial proportion accuracy and error:","{:.3f}".format(bp_acc),"±","{:.3f}".format(bp_error),"\n")

    #error analysis
    y_array = y_test.iloc[:, 0].values

    rng = np.random.RandomState()
    idx = np.arange(y_test.shape[0])
```

Figure 13: An example of the function that defines the metrics output for each model.

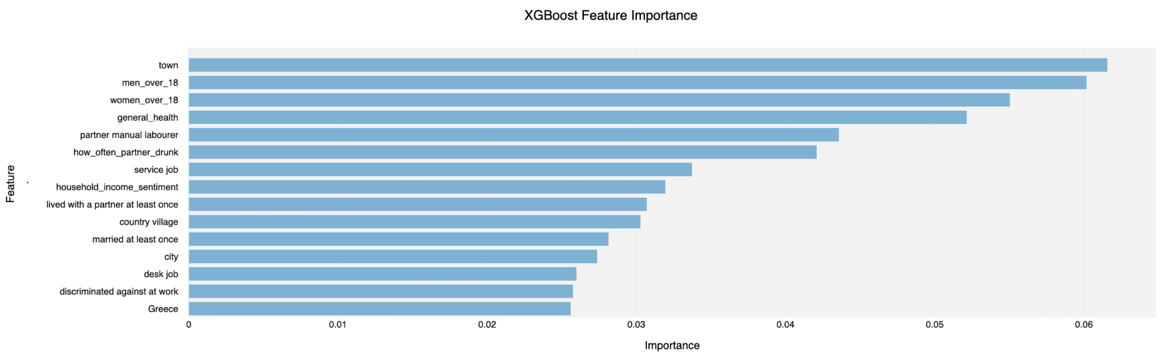
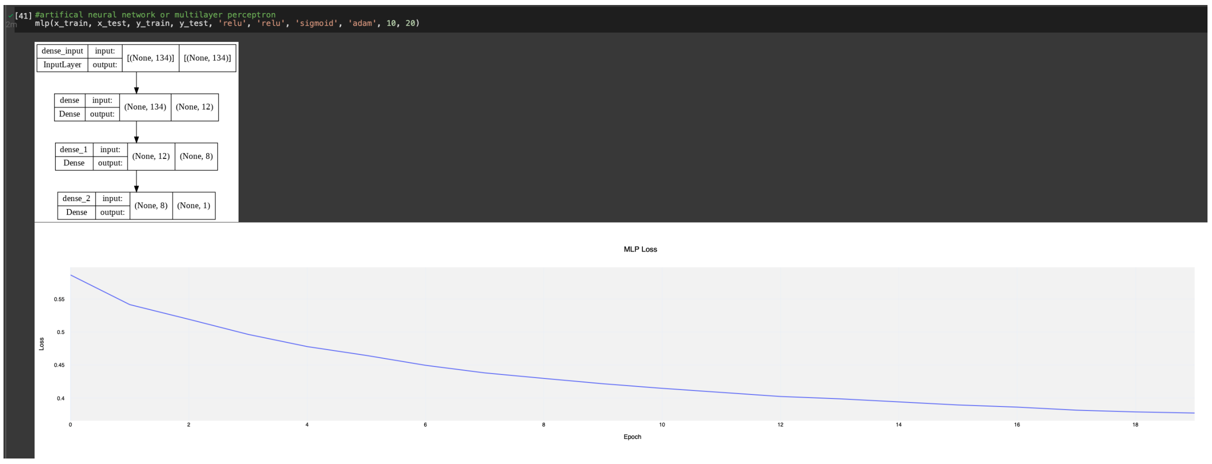


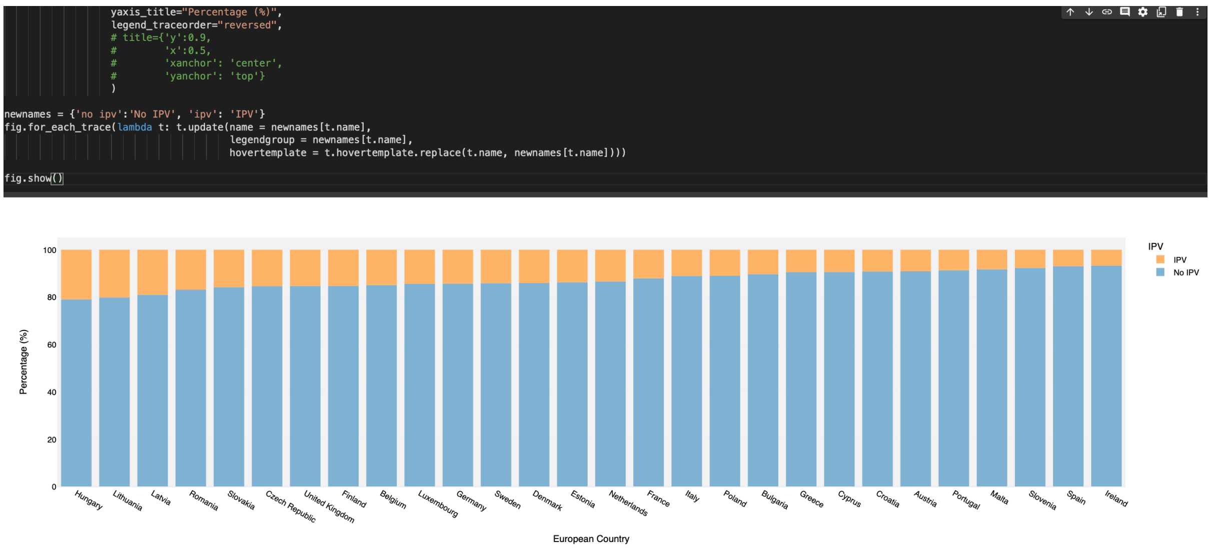
Figure 14: Additional impurity based feature importance barchart output when the XG-Boost function is called.



```
[41] #artifical neural network or multilayer perceptron
     mlp(x_train, x_test, y_train, y_test, 'relu', 'relu', 'sigmoid', 'adam', 10, 20)
```

dense_input | InputLayer | input: | output: | [(None, 134)] | [(None, 134)]
dense | Dense | input: | output: | (None, 134) | (None, 12)
dense_1 | Dense | input: | output: | (None, 12) | (None, 8)
dense_2 | Dense | input: | output: | (None, 8) | (None, 1)
MLP Loss
Loss
0.55
0.5
0.45
0.4
0
2
4
6
8
10
12
14
16
18
Epoch

Figure 15: Additional graphs describing the structure of the neural network and the loss on the test set output when the multilayer perceptron function is called.

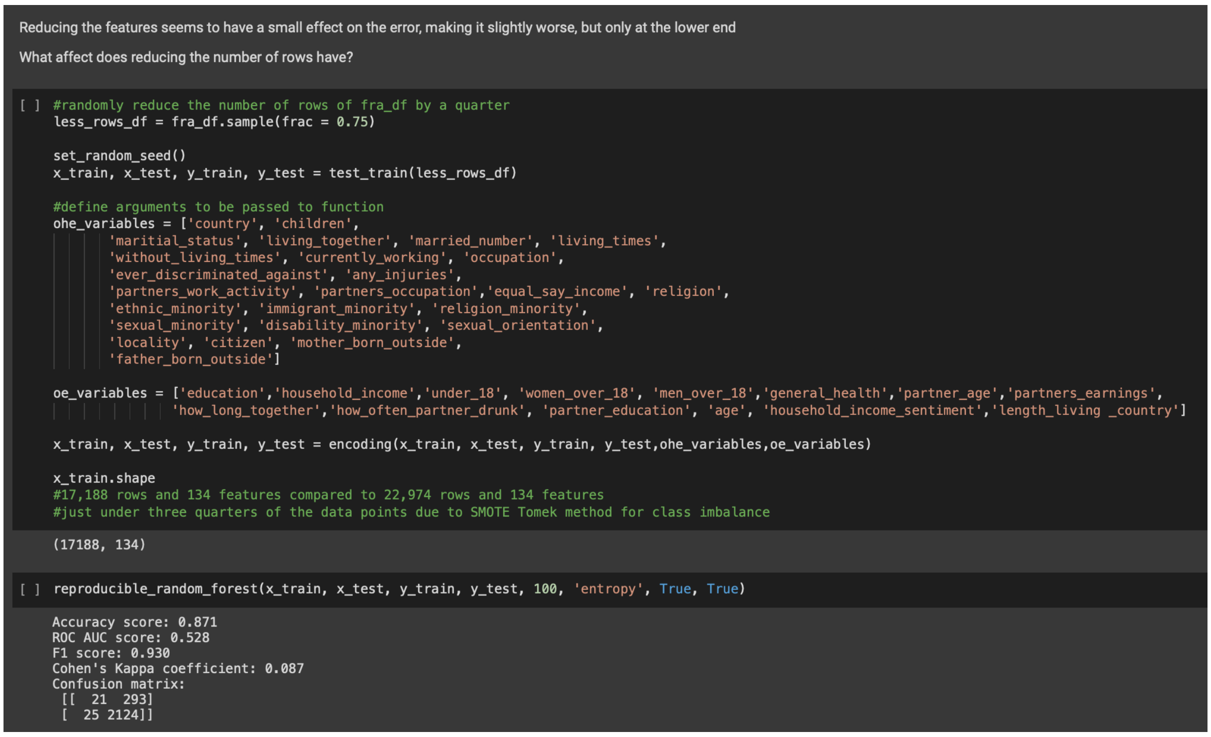


```
         yaxis_title="Percentage (%)",
         legend_traceorder="reversed",
         # title={'y':0.9,
         #       'x':0.5,
         #       'xanchor': 'center',
         #       'yanchor': 'top'}
         )

newnames = {'no ipv':'No IPV', 'ipv': 'IPV'}
fig.for_each_trace(lambda t: t.update(name = newnames[t.name],
                                      legendgroup = newnames[t.name],
                                      hovertemplate = t.hovertemplate.replace(t.name, newnames[t.name])))

fig.show()
```

Figure 16: Additional analysis of the FRA data using the plotly graphing package.

4.4 Error Analysis

The final analysis undertaken is error analysis. One model, Random Forest, is called repeatedly and passed either a randomly reduced number of features or training points or an artificially inflated amount of FRA data. This is to determine how accuracy and variance around the accuracy returned by a bootstrapped confidence interval changes. A second Random Forest function was created, with a seed set so that variance could be attributed to the changing amounts of data. Both features and data points in the training sets are randomly reduced by a quarter, a half and three quarters using pandas functions such as sample (Figure 17). Data points are inflated through random resampling with replacement by double, triple and quadruple also using the pandas sample function.



Reducing the features seems to have a small effect on the error, making it slightly worse, but only at the lower end

What affect does reducing the number of rows have?

```
[ ] #randomly reduce the number of rows of fra_df by a quarter
    less_rows_df = fra_df.sample(frac = 0.75)

    set_random_seed()
    x_train, x_test, y_train, y_test = test_train(less_rows_df)

    #define arguments to be passed to function
    ohe_variables = ['country', 'children',
                  'maritial_status', 'living_together', 'married_number', 'living_times',
                  'without_living_times', 'currently_working', 'occupation',
                  'ever_discriminated_against', 'any_injuries',
                  'partners_work_activity', 'partners_occupation','equal_say_income', 'religion',
                  'ethnic_minority', 'immigrant_minority', 'religion_minority',
                  'sexual_minority', 'disability_minority', 'sexual_orientation',
                  'locality', 'citizen', 'mother_born_outside',
                  'father_born_outside']

    oe_variables = ['education','household_income','under_18', 'women_over_18', 'men_over_18','general_health','partner_age','partners_earnings',
                    'how_long_together','how_often_partner_drunk', 'partner_education', 'age', 'household_income_sentiment','length_living _country']

    x_train, x_test, y_train, y_test = encoding(x_train, x_test, y_train, y_test,ohe_variables,oe_variables)

    x_train.shape
    #17,188 rows and 134 features compared to 22,974 rows and 134 features
    #just under three quarters of the data points due to SMOTE Tomek method for class imbalance

    (17188, 134)
```

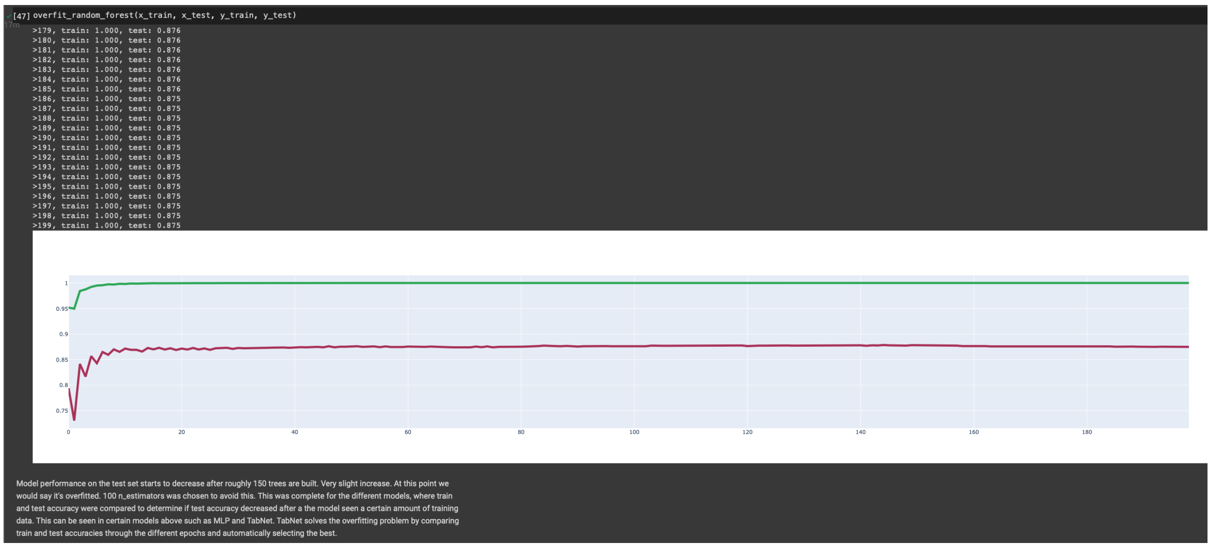
```
[ ] reproducible_random_forest(x_train, x_test, y_train, y_test, 100, 'entropy', True, True)

    Accuracy score: 0.871
    ROC AUC score: 0.528
    F1 score: 0.930
    Cohen's Kappa coefficient: 0.087
    Confusion matrix:
     [[  21  293]
     [  25 2124]]
```

Figure 17: An example of error analysis, where the FRA data is reduced by a quarter to determine the effect on accuracy.

4.5 Overfitting

When model optimisation occurred, all models were tested to ensure overfitting did not occur. For some models, this testing was left in the outputs produced by the final models, such as multilayer perceptron and TabNet, seen in the graphs describing loss and accuracy of the training and test sets over the number of epochs. For other models, additional outputs were generated to determine whether the models were being overfit dependent on the number of estimators being passed. To reduce overall runtime and output, these were not maintained following optimisation. An example of this work can be found at the bottom of the ipynb file, where a function applies random forest with an increasing number of estimators and outputs the accuracies returned on the training and test sets (Figure 18).

The final section in the ipynb file contains rough work, which includes pieces of code



```
[47] overfit_random_forest(x_train, x_test, y_train, y_test)
     >179, train: 1.000, test: 0.876
     >180, train: 1.000, test: 0.876
     >181, train: 1.000, test: 0.876
     >182, train: 1.000, test: 0.876
     >183, train: 1.000, test: 0.876
     >184, train: 1.000, test: 0.876
     >185, train: 1.000, test: 0.876
     >186, train: 1.000, test: 0.875
     >187, train: 1.000, test: 0.875
     >188, train: 1.000, test: 0.875
     >189, train: 1.000, test: 0.875
     >190, train: 1.000, test: 0.875
     >191, train: 1.000, test: 0.875
     >192, train: 1.000, test: 0.875
     >193, train: 1.000, test: 0.875
     >194, train: 1.000, test: 0.875
     >195, train: 1.000, test: 0.875
     >196, train: 1.000, test: 0.875
     >197, train: 1.000, test: 0.875
     >198, train: 1.000, test: 0.875
     >199, train: 1.000, test: 0.875
```

Model performance on the test set starts to decrease after roughly 150 trees are built. Very slight increase. At this point we would say it's overfitted. 100 n_estimators was chosen to avoid this. This was complete for the different models, where train and test accuracy were compared to determine if test accuracy decreased after a the model seen a certain amount of training data. This can be seen in certain models above such as MLP and TabNet. TabNet solves the overfitting problem by comparing train and test accuracies through the different epochs and automatically selecting the best.

Figure 18: An example of how models are tested for overfitting during the optimisation process.

that may have initially been used until a decision to change a part was made or until an alternative method was found. The progress of the research and how the code evolved can be visualised here.

5 Acknowledgements

I would like to thank Sebastian Raschka for his useful and succinct tutorial on the different ways to implement confidence intervals in machine learning (see Footnote 5). Special thanks to Ini-Abasi Affiah for his very helpful guide on how to build a multilayer perceptron that works for tabular data (see Footnote 6). To all those in scikit-learn and plotly for their comprehensive documentation on how to implement the models, produce the metrics and make nice graphs. Finally, thanks to the users on Stack Overflow who had the same technical issues as I did, and to those that solved them.

References

Arik, S. Ö. and Pfister, T. (2021), Tabnet: Attentive interpretable tabular learning, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 35, pp. 6679–6687.

Pace, L. (2012), *Beginning R: An introduction to statistical programming*, Apress.