Python Code for Fire Fatality Profiling Scoring

Alfi Gözaçan*

March 7, 2022

*Humberside Fire and Rescue Service

Dataset Name	Dimensions	Description
dwellings	(433748, 57)	Addresses and Mosaic types of dwellings in Humberside
response	(433748, 4)	Flag denoting whether dwelling is in or out of 8-minute response radius
mosaic_means	(1281, 88)	Experian Mosaic UK 7 index means
exeter_data	(209342, 13)	Exeter GP over-65s data

Table 1: Table of datasets used.

1 Overview

The code used in the scoring methodology can be split into two parts: data cleaning and score assigning. The code can be found here). Now, the following snippet lists the modules to be imported.

```
import pandas as pd
import numpy as np
import datetime

from tqdm import tqdm
from thefuzz import process
from itertools import compress
```

2 Data Cleaning

Table 1 summarises the data to be loaded in to the computer. Each dataset is available in .csv format and will be referred to in any Python scripts by its name as listed in Table 1.

Firstly, the data is imported into a Python environment / Jupyter Notebook using pandas.

```
file_path = "C:\\Users\\...\\"
dwellings = pd.read_csv(file_path+"dwellings.csv", encoding="latin-1")
response = pd.read_csv(file_path+"response.csv")
mosaic_means = pd.read_csv(file_path+"mosaic_means.csv")
exeter = pd.read_csv(file_path+"exeter_data.csv")
```

Then, any entries from the Exeter data where the first address line contains "care home" or "residential" are removed, since the FFP is only concerned with private dwellings.

```
exeter.drop(exeter.index[exeter["Address_Line_1"].str.contains(
"care home", case=False, regex=True).replace(np.nan, False)], axis=0, inplace=True)

exeter.drop(exeter.index[exeter["Address_Line_1"].str.contains(
"residential", case=False, regex=True).replace(np.nan, False)], axis=0, inplace=True)

exeter.reset_index(drop=True, inplace=True)
```

Next, we construct a dataframe of all the multipliers against their factors, with the corresponding Mosaic indices for reference.

```
multipliers = pd.DataFrame({
    "Attribute" : ["Base score",
                   "Is living alone",
                   "Is aged 80 or over and female",
                   "Is aged 65 to 79 and male",
                   "Is aged 65 to 79 and female",
                   "Is aged 80 or over and male",
                   "Is a smoker and male",
                   "Is a smoker and female",
                   "Has restricted mobility",
                   "Regularly drinks alcohol once or more per day",
                   "Is living in social rented housing",
                   "Lives outside of 8 minute response zone"],
    "Multiplier" : [1.00, 1.78, 2.00, 2.08, 2.37, 2.63, 4.52, 6.68, 1.10, 1.10, 3.89, 1.10],
    "Mosaic_Index" : ["--", 31, "--", "--", "--", "--", [1033, 0], [1033, 0], 1278, 1049, 102, "--"]})
```

After that, the data is cleaned by changing some column names and odd data types. Also, unclassified households given a Mosaic type of ”U99” are removed and the response time radius in/out flag is joined onto the dataframe.

```
dwelling_cols = ["Addressbase UPRN",
                 "Sub Building Name",
                 "Building Name",
                 "Street Number",
                 "Dependent Street",
                 "Street",
                 "Double Dependent Locality",
                 "Dependent Locality",
                 "Town",
                 "Postcode",
                 "Addressbase Easting",
                 "Addressbase Northing",
                 "(H) Mosaic UK 7 Type Label",
                 "(PC) Output Area (OA)",
                 "(PC) Super Output Areas - Lower Layer",
                 "(PC) Super Output Areas - Middle Layer",
                 "(PC) Electoral Wards",
                 "(PC) Local Authority Districts and Unitary Authorities Code",
                 "(PC) Counties and Unitary Authorities Code"]

dwellings = dwellings[dwelling_cols]

dwellings.rename(columns = {
    "Addressbase UPRN" : "UPRN",
    "Addressbase Easting" : "Easting",
    "Addressbase Northing" : "Northing",
    "(H) Mosaic UK 7 Type Label" : "Type_Desc",
    "(PC) Output Area (OA)" : "OA",
    "(PC) Super Output Areas - Lower Layer" : "LSOA",
    "(PC) Super Output Areas - Middle Layer" : "MSOA",
    "(PC) Electoral Wards" : "Ward",
    "(PC) Local Authority Districts and Unitary Authorities Code" : "Local Authority Code",
    "(PC) Counties and Unitary Authorities Code" : "Counties Code"
}, inplace=True)

dwellings["Type_Desc"] = [x[:3] for x in dwellings["Type_Desc"]]

dwellings.drop(dwellings.index[dwellings["UPRN"].isnull()], axis=0, inplace=True)
dwellings.drop_duplicates(subset="UPRN", keep="first", inplace=True)
dwellings.drop(dwellings.index[dwellings["Type_Desc"] == "U99"], axis=0, inplace=True)
dwellings.reset_index(drop=True, inplace=True)

response.drop("OID_", axis=1, inplace=True)
response.rename(columns={
    "InOut" : "Response"
}, inplace=True)
dwellings = dwellings.merge(right=response, left_on="UPRN", right_on="UPRN", how="left")

dwellings.replace(np.nan, "", inplace=True)
exeter.replace(np.nan, "", inplace=True)
exeter["UPRN"].replace("", 0, inplace=True)

dwellings["UPRN"] = [int(x) for x in dwellings["UPRN"]]
exeter["UPRN"] = [int(x) for x in exeter["UPRN"]]

dwellings["Postcode"].replace(" ", "", regex=True, inplace=True)
exeter["Postcode"].replace(" ", "", regex=True, inplace=True)
```

Then, problematic Exeter entries that use historic UPRNs are address-matched to give them UPRNs corresponding to the address in `dwellings` with the closest match, using fuzzy matching.

```
bad_indices = exeter.index[~exeter["UPRN"].isin(dwellings["UPRN"])]

address_strings = []

for i in tqdm(range(len(dwellings))):
    string = " ".join(entry for entry in dwellings.iloc[i, 1:10])
    address_strings.append(string)
exeter_strings = []

for i in tqdm(bad_indices):
    string = " ".join(entry for entry in exeter.iloc[i, [2, 3, 4, 5, 7]])
    exeter_strings.append(string)

matching_indices = []
final_fuzz_ratios = []

for i in tqdm(range(len(exeter_strings))):
    viable_addresses = list(compress(address_strings, [x[-7:] == exeter_strings[i][-7:]
            for x in address_strings]))
    if len(viable_addresses) == 0:
        matching_indices.append(0)
        final_fuzz_ratios.append(0)
        continue
    pair = process.extractOne(exeter_strings[i], viable_addresses)
    address = pair[0]
    match_score = pair[1]
    matching_indices.append(address_strings.index(address))
    final_fuzz_ratios.append(match_score)

exeter.loc[bad_indices, "UPRN"] = list(dwellings.loc[matching_indices, "UPRN"])
```

Finally, some flags are assigned to the entries which were address-matched. Entries where the fuzz ratio is too low are removed due to the matching not being strong enough.

```
exeter.loc[bad_indices, "is_Matched"] = 1
exeter.loc[bad_indices, "Match_Score"] = final_fuzz_ratios

remove_indices = exeter.index[exeter["Match_Score"] < 95]
exeter.drop(remove_indices, axis=0, inplace=True)
exeter.reset_index(drop=True, inplace=True)

exeter.rename(columns={"Postcode" : "Postcode_2"}, inplace=True)

now = datetime.datetime.now()
year = now.year
exeter["Age"] = [year - x for x in exeter["Year_Of_Birth"]]

df = dwellings.merge(right=exeter, on="UPRN", how="left")
```

3 Assigning Scores

In this section of the code, each entry in the joined dataset is assigned a risk score based on the multipliers dataframe. Firstly, we must find those Mosaic types which over-represent the characteristics deemed indicative of fire risk and assign a list of binary values to each entry of the dataframe depending on which characteristics are over-represented. This is done via a simple join.

```
mosaic_scores = pd.DataFrame({
    "Mosaic_Type" : mosaic_means.columns[-66:]
})

for i in range(-66, 0):
    if mosaic_means.iloc[0, i] > mosaic_means.iloc[0, -82]:
        mosaic_scores.loc[i+66, "Male"] = 1
    if mosaic_means.iloc[31, i] > mosaic_means.iloc[31, -82]:
        mosaic_scores.loc[i+66, "Single"] = 1
    if mosaic_means.iloc[1033, i] > mosaic_means.iloc[1033, -82]:
        mosaic_scores.loc[i+66, "Smoker"] = 1
    if mosaic_means.iloc[1278, i] > mosaic_means.iloc[1278, -82]:
        mosaic_scores.loc[i+66, "Restricted_Mobility"] = 1
    if mosaic_means.iloc[1049, i] > mosaic_means.iloc[1049, -82]:
        mosaic_scores.loc[i+66, "Alcohol"] = 1
    if mosaic_means.iloc[102, i] > mosaic_means.iloc[102, -82]:
        mosaic_scores.loc[i+66, "Rented"] = 1

mosaic_scores.replace(np.nan, 0, inplace=True)

df = df.merge(right=mosaic_scores, left_on="Type_Desc", right_on="Mosaic_Type", how="left")
```

Then, other factors are brought in to calculate a final risk score for each dwelling.

```
for i in tqdm(range(len(df))):
    score = multipliers.iloc[0, 1]
    if df.loc[i, "Response"] == "Out":
        score = score * multipliers.iloc[11, 1]
    if df.loc[i, "Restricted_Mobility"] == 1:
        score = score * multipliers.iloc[8, 1]
    if df.loc[i, "Alcohol"] == 1:
        score = score * multipliers.iloc[9, 1]
    if df.loc[i, "Rented"] == 1:
        score = score * multipliers.iloc[10, 1]
    if df.loc[i, "Single"] == 1:
        score = score * multipliers.iloc[1, 1]
    if df.loc[i, "Gender"] == np.nan:
        if df.loc[i, "Smoker"] == 1:
            if df.loc[i, "Male"] == 1:
                score = score * multipliers.iloc[6, 1]
            else:
                score = score * multipliers.iloc[7, 1]
    else:
        if df.loc[i, "Gender"] == "M":
            if df.loc[i, "Smoker"] == 1:
                score = score * multipliers.iloc[6, 1]
            if df.loc[i, "Age"] >= 65 and df.loc[i, "Age"] <= 79:
                score = score * multipliers.iloc[3, 1]
            elif df.loc[i, "Age"] >= 80:
                score = score * multipliers.iloc[5, 1]
        else:
            if df.loc[i, "Smoker"] == 1:
                score = score * multipliers.iloc[7, 1]
            if df.loc[i, "Age"] >= 65 and df.loc[i, "Age"] <= 79:
                score = score * multipliers.iloc[4, 1]
            elif df.loc[i, "Age"] >= 80:
                score = score * multipliers.iloc[2, 1]

    df.loc[i, "Final_Score"] = score
```

After this is done, the data to be outputted is tidied a little bit more by removing some columns.

```
df.drop(["Type_Desc",,
         "Address_Line_5",
         "firearea",
         "firename",
         "Frailty_Score",
         "Frailty_Group"], axis=1, inplace=True)
```

Then, a flag is set depending on whether the entry is in the Exeter data. Also, duplicate UPRNs are removed from the data (where the highest risk score is kept).

```
df.insert(df.columns.get_loc("Gender"), "is_Exeter", [int(x) for x in ~df["Gender"].isnull()])

df = df.sort_values(by="Final_Score", ascending=False)
.drop_duplicates(subset="UPRN", keep="first").reset_index(drop=True)
```

Next, the risk scores are put into priority groups so that the current labels can be kept. The thresholds were chosen so that there is a reasonable number of households in each group, depending on its priority.

```
df["Final_Score"] = round(df["Final_Score"], 2)

df.loc[df.index[df["Final_Score"] > 0], "Priority"] = "NR"
df.loc[df.index[df["Final_Score"] > 5], "Priority"] = "F"
df.loc[df.index[df["Final_Score"] > 10], "Priority"] = "E"
df.loc[df.index[df["Final_Score"] > 16], "Priority"] = "D"
df.loc[df.index[df["Final_Score"] > 30], "Priority"] = "C"
df.loc[df.index[df["Final_Score"] > 55], "Priority"] = "B"
df.loc[df.index[df["Final_Score"] > 70], "Priority"] = "B+"
df.loc[df.index[df["Final_Score"] > 110], "Priority"] = "A"
df.loc[df.index[df["Final_Score"] > 130], "Priority"] = "A+"
```

Finally, the output is saved.

```
df.to_csv(file_path+"output.csv", index=False)
```