ARISTOTLE UNIVERSITY OF THESSALONIKI
COMPUTER SCIENCE DEPARTMENT
Msc DATA AND WEB SCIENCE

Final Report - Violence Against Women Dataset

MACHINE LEARNING

Valsamis Dimitrios - AI - AEM: 89
Kavargiris Dimitrios Christos- DWS - AEM: 87
Malama Andreana - DWS - AEM: 110

Thessaloniki, February 2022

Contents

Chapter 1

STUDY THE DATASET

1.1 IMPORT LIBRARIES

[1]:
```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.cluster import KMeans
import sklearn.datasets as ds
from sklearn.pipeline import Pipeline
import plotly
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score,␣
↪precision_score, f1_score, confusion_matrix, \
    classification_report

from sklearn.impute import SimpleImputer
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from collections import Counter
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import RandomOverSampler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from impyute.imputation.cs import fast_knn
# sys.setrecursionlimit(100000) #Increase the recursion limit of␣
↪the OS
pd.options.display.max_columns
pd.options.display.max_rows
```

[1]: 60

1.2 LOAD DATA

[2]:
```
df = pd.read_csv('/Users/andre/PycharmProjects/final_ML/VAW.csv')

df.head()
```

[2]:
```
          DATAFLOW FREQ: Frequency  TIME_PERIOD: Time  \
0  SPC:DF_VAW(1.0)       A: Annual               2013
1  SPC:DF_VAW(1.0)       A: Annual               2013
2  SPC:DF_VAW(1.0)       A: Annual               2013
3  SPC:DF_VAW(1.0)       A: Annual               2013
4  SPC:DF_VAW(1.0)       A: Annual               2013

  GEO_PICT: Pacific Island Countries and territories  \
0                                    CK: Cook Islands
1                                    CK: Cook Islands
2                                    CK: Cook Islands
3                                    CK: Cook Islands
4                                    CK: Cook Islands

                                       TOPIC: Topic  \
0  VAW_TOPIC_001: Types of violence against women...
1  VAW_TOPIC_001: Types of violence against women...
2  VAW_TOPIC_001: Types of violence against women...
3  VAW_TOPIC_001: Types of violence against women...
4  VAW_TOPIC_001: Types of violence against women...

                               INDICATOR: Indicator   SEX: Sex ␣
↪\
0  NUMPERRF: Number of persons in relative frequency  F: Female
1  NUMPERRF: Number of persons in relative frequency  F: Female
2  NUMPERRF: Number of persons in relative frequency  F: Female
3  NUMPERRF: Number of persons in relative frequency  F: Female
4  NUMPERRF: Number of persons in relative frequency  F: Female

        AGE: Age CONDITION: Women's condition  \
0  Y15T64: 15-64       EVPART: Ever-partnered
1  Y15T64: 15-64       EVPART: Ever-partnered
2  Y15T64: 15-64       EVPART: Ever-partnered
3  Y15T64: 15-64       EVPART: Ever-partnered
4  Y15T64: 15-64       EVPART: Ever-partnered

                          VIOLENCE_TYPE: Type of violence  ... OUTCOME:␣
↪Outcome  \
```

```
            0  CONT_ECON: At least one act of economic abusive  ...          ␣
↪_T: Any
            1                        EMO: Emotional violence  ...          ␣
↪_T: Any
            2                        EMO: Emotional violence  ...          ␣
↪_T: Any
            3                       PHYS: Physical violence  ...          ␣
↪_T: Any
            4                       PHYS: Physical violence  ...          ␣
↪_T: Any

              RESPONSE: Response HELP_REASON: Reason for searching help  \
            0           _T: Any                                 _T: Any
            1           _T: Any                                 _T: Any
            2           _T: Any                                 _T: Any
            3           _T: Any                                 _T: Any
            4           _T: Any                                 _T: Any

              HELP_PROVIDER: Help provider OBS_VALUE UNIT_MEASURE: Unit of␣
↪measure  \
            0                      _T: Any       6.2              PERCENT:␣
↪percent
            1                      _T: Any       9.6              PERCENT:␣
↪percent
            2                      _T: Any      26.7              PERCENT:␣
↪percent
            3                      _T: Any       6.7              PERCENT:␣
↪percent
            4                      _T: Any      30.2              PERCENT:␣
↪percent

              UNIT_MULT: Unit multiplier  OBS_STATUS: Observation Status  \
            0                        NaN                              NaN
            1                        NaN                              NaN
            2                        NaN                              NaN
            3                        NaN                              NaN
            4                        NaN                              NaN

              DATA_SOURCE: Data source  OBS_COMMENT: Comment
            0                     FHSS                   NaN
            1                     FHSS                   NaN
            2                     FHSS                   NaN
            3                     FHSS                   NaN
            4                     FHSS                   NaN

            [5 rows x 23 columns]
```

1.3 PLOTS

1.3.1 CATEGORICAL TO NUMERICAL

In order to study the original dataset we transform the categorical data to numerical with the use of a defined function. The '_T: Any' values are transformed to number 0.

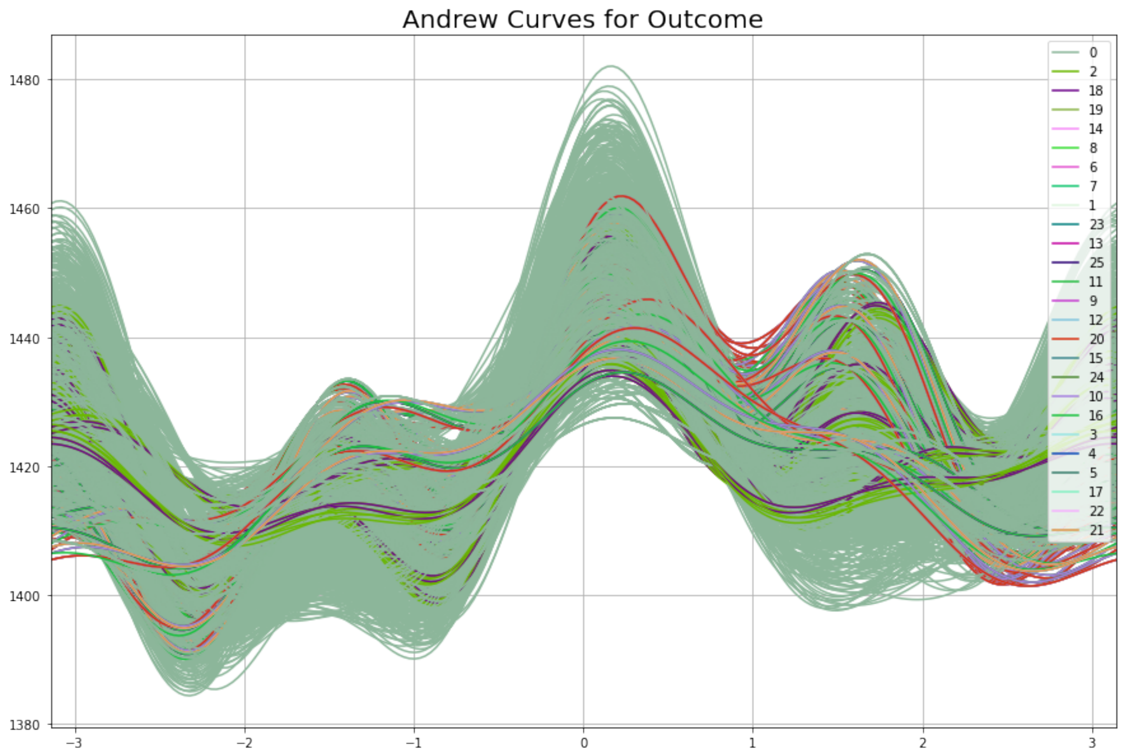
[3]:
```
df_numerical = df.copy(deep=True)
df_numerical.drop(['FREQ: Frequency','DATAFLOW', 'OBS_VALUE',␣
↪'OBS_COMMENT: Comment','OBS_STATUS: Observation Status',
'SEX: Sex', 'UNIT_MULT: Unit multiplier', 'AGE: Age', 'INDICATOR:␣
↪Indicator', 'UNIT_MEASURE: Unit of measure',
'HELP_PROVIDER: Help provider', 'HELP_REASON: Reason for searching␣
↪help', 'RESPONSE: Response', 'LIFEPER: Period of life'], axis=1,inplace=True)
def handle_non_numerical_data(df):
    columns = df.columns.values
    for column in columns:
        text_digit_vals = {}
        def convert_to_int(val):
            return text_digit_vals[val]

        if df[column].dtype != np.int64 and df[column].dtype != np.
↪float64:
            column_contents = df[column].values.tolist()
            unique_elements = set(column_contents)
            x = 1
            for unique in unique_elements:
                if unique == '_T: Any':
                    text_digit_vals[unique] = 0
                elif unique not in text_digit_vals:
                    text_digit_vals[unique] = x
                    x += 1
            df[column] = list(map(convert_to_int, df[column]))
    return df

df_numerical = handle_non_numerical_data(df_numerical)
```

1.3.2 ANDREW CURVES

In data visualization, an Andrews plot or Andrews curve is a way to visualize structure in high-dimensional data. It is basically a rolled-down, non-integer version of the Kent–Kiviat radar m chart, or a smoothed version of a parallel coordinate plot.

- It has been shown the Andrews curves are able to preserve means, distance (up to a constant)and variances. Which means that Andrews curves that are represented by functions close together suggest that the corresponding data points will also be close together.

```
[4]: from pandas import plotting
plt.rcParams['figure.figsize'] = (15, 10)

plotting.andrews_curves(df_numerical,'OUTCOME: Outcome')
plt.title('Andrew Curves for Outcome', fontsize = 20)
plt.show()
```



1.3.3 DISTRIBUTION PATTERNS

```
[11]: import warnings
warnings.filterwarnings('ignore')

print('CONDITION: Women's condition\n')
Z = df['CONDITION: Women's condition']
counter = Counter(Z)
for k,v in counter.items():
    per = v / len(Z) * 100
    print('Class=%s, Count=%d, Percentage=%.3f%%' % (k, v, per))

print('\nCONDITION: Women's condition - NUMERICAL\n')
Z_numerical = df_numerical['CONDITION: Women's condition']
counter = Counter(Z_numerical)
for k,v in counter.items():
```

```
    per = v / len(Z_numerical) * 100
    print('Class=%s, Count=%d, Percentage=%.3f%%' % (k, v, per))

print('\n\nVIOLENCE_TYPE: Type of violence\n')
Z = df['VIOLENCE_TYPE: Type of violence']
counter = Counter(Z)
for k,v in counter.items():
    per = v / len(Z) * 100
    print('Class=%s, Count=%d, Percentage=%.3f%%' % (k, v, per))

print('\nVIOLENCE_TYPE: Type of violence - NUMERICAL\n')
Z_numerical = df_numerical['VIOLENCE_TYPE: Type of violence']
counter = Counter(Z_numerical)
for k,v in counter.items():
    per = v / len(Z_numerical) * 100
    print('Class=%s, Count=%d, Percentage=%.3f%%' % (k, v, per))

plt.rcParams['figure.figsize'] = (18, 8)

plt.subplot(1, 2, 1)
sns.set(style = 'whitegrid')
sns.distplot(df_numerical['CONDITION: Women's condition'])
plt.title('Distribution of Women's condition', fontsize = 20)
plt.xlabel('Range of Women's condition')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
sns.set(style = 'whitegrid')
sns.distplot(df_numerical['VIOLENCE_TYPE: Type of violence'], color = 'red')
plt.title('Distribution of type of violence', fontsize = 20)
plt.xlabel('Range of type of violence')
plt.ylabel('Count')
plt.show()
```

```
CONDITION: Women's condition

Class=EVPART: Ever-partnered, Count=688, Percentage=34.127%
Class=EVPREG: Ever pregnant, Count=112, Percentage=5.556%
Class=_T: Any, Count=1056, Percentage=52.381%
Class=W4M: Working for money, Count=64, Percentage=3.175%
Class=CHI614: With children 6-14 years old, Count=96, Percentage=4.762%

CONDITION: Women's condition - NUMERICAL

Class=3, Count=688, Percentage=34.127%
Class=4, Count=112, Percentage=5.556%
Class=0, Count=1056, Percentage=52.381%
```

Class=2, Count=64, Percentage=3.175%
Class=1, Count=96, Percentage=4.762%

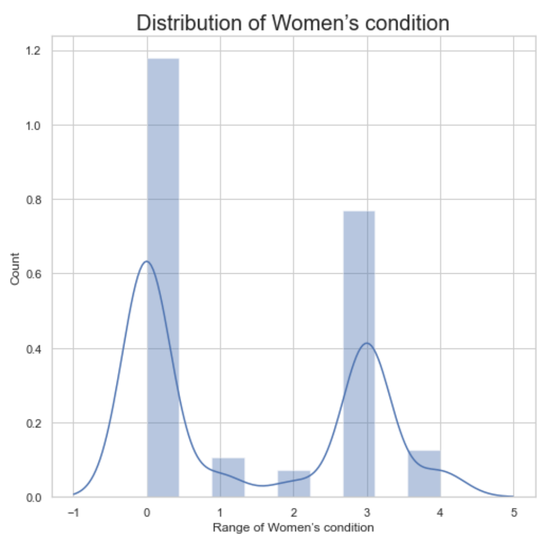
VIOLENCE_TYPE: Type of violence

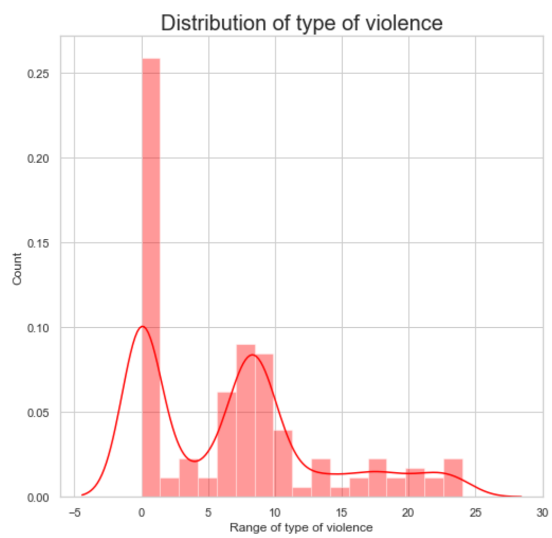
Class=CONT_ECON: At least one act of economic abusive, Count=16, Percentage=0.794%
Class=EMO: Emotional violence, Count=32, Percentage=1.587%
Class=PHYS: Physical violence, Count=256, Percentage=12.698%
Class=PHYSORSEX: Physical and/or sexual violence, Count=144, Percentage=7.143%
Class=SEX: Sexual violence, Count=240, Percentage=11.905%
Class=_T: Any, Count=704, Percentage=34.921%
Class=PHYS_MOD: Moderate physical violence, Count=16, Percentage=0.794%
Class=PHYS_SEV: Severe physical violence, Count=16, Percentage=0.794%
Class=PHYS_CHOK: Choked or burnt on purpose, Count=32, Percentage=1.587%
Class=PHYS_FIST: Hit with a fist or something else, Count=32, Percentage=1.587%
Class=PHYS_KICK: Kicked, dragged, beaten, Count=32, Percentage=1.587%
Class=PHYS_PUSH: Pushed or shoved, Count=32, Percentage=1.587%
Class=PHYS_SLAP: Slapped or having something thrown at them, Count=32, Percentage=1.587%
Class=PHYS_WEAP: Threatened with or had a gun, knife or weapon used on them, Count=32, Percentage=1.587%
Class=SEX_AFRAID: Having sexual intercourse because they were afraid of what partners could do, Count=32, Percentage=1.587%
Class=SEX_DEGRAD: Forced to perform degrading or humiliating sexual act(s), Count=32, Percentage=1.587%
Class=SEX_FORCE: Physically forced to have sexual intercourse when they did not want, Count=32, Percentage=1.587%
Class=EMO_HUM: Belittled or humiliated, Count=32, Percentage=1.587%
Class=EMO_INS: Insulted, Count=32, Percentage=1.587%
Class=EMO_SCA: Scared or intimidated, Count=32, Percentage=1.587%
Class=CONT_FRIENDS: Partner keeps her from seeing her friends, Count=32, Percentage=1.587%
Class=CONT_OTHMAN: Partner gets angry if she speaks with another man, Count=32, Percentage=1.587%
Class=CONT_UNFAITH: Partner often suspicious she is unfaithful, Count=32, Percentage=1.587%
Class=CONT_WHERE: Partner insists on knowing where she is at all times, Count=32, Percentage=1.587%
Class=SEX_CHILD: Child sexual abuse, Count=80, Percentage=3.968%

VIOLENCE_TYPE: Type of violence - NUMERICAL

Class=15, Count=16, Percentage=0.794%
Class=22, Count=32, Percentage=1.587%
Class=8, Count=256, Percentage=12.698%
Class=7, Count=144, Percentage=7.143%

```
Class=9, Count=240, Percentage=11.905%
Class=0, Count=704, Percentage=34.921%
Class=12, Count=16, Percentage=0.794%
Class=20, Count=16, Percentage=0.794%
Class=6, Count=32, Percentage=1.587%
Class=3, Count=32, Percentage=1.587%
Class=24, Count=32, Percentage=1.587%
Class=5, Count=32, Percentage=1.587%
Class=11, Count=32, Percentage=1.587%
Class=14, Count=32, Percentage=1.587%
Class=13, Count=32, Percentage=1.587%
Class=18, Count=32, Percentage=1.587%
Class=4, Count=32, Percentage=1.587%
Class=2, Count=32, Percentage=1.587%
Class=21, Count=32, Percentage=1.587%
Class=1, Count=32, Percentage=1.587%
Class=17, Count=32, Percentage=1.587%
Class=19, Count=32, Percentage=1.587%
Class=16, Count=32, Percentage=1.587%
Class=23, Count=32, Percentage=1.587%
Class=10, Count=80, Percentage=3.968%
```





In the Plots above we can see the Distribution pattern of Women's Condition and the Type of Violence.

First of all in the left plot the classes are refered as follows :

- class 0 : _T:Any
- class 1 : With children 6-14 years old
- class 2 : Working for money

- class 3 : Ever-partnered
- class 4 : Ever pregnant

We can not draw an accurate conclusion for the class 0 as we do not know the condition of the victim, but as we can see the next case with most samples is class 1 where the women are "Ever-partnered" and on the other hand the fewest victims are "Working women for money".

As far as the right plot is concerned most attacks have not been identified and are concentrated below class 0. Also the different types of violence are about the same levels with some categories standing out and these are the cases of "physical" and "sexual violence". The class of physical violence is the class 8 and the sexual violence is the class 9, as shown in the plot they are the areas with the highest concentration of observations.

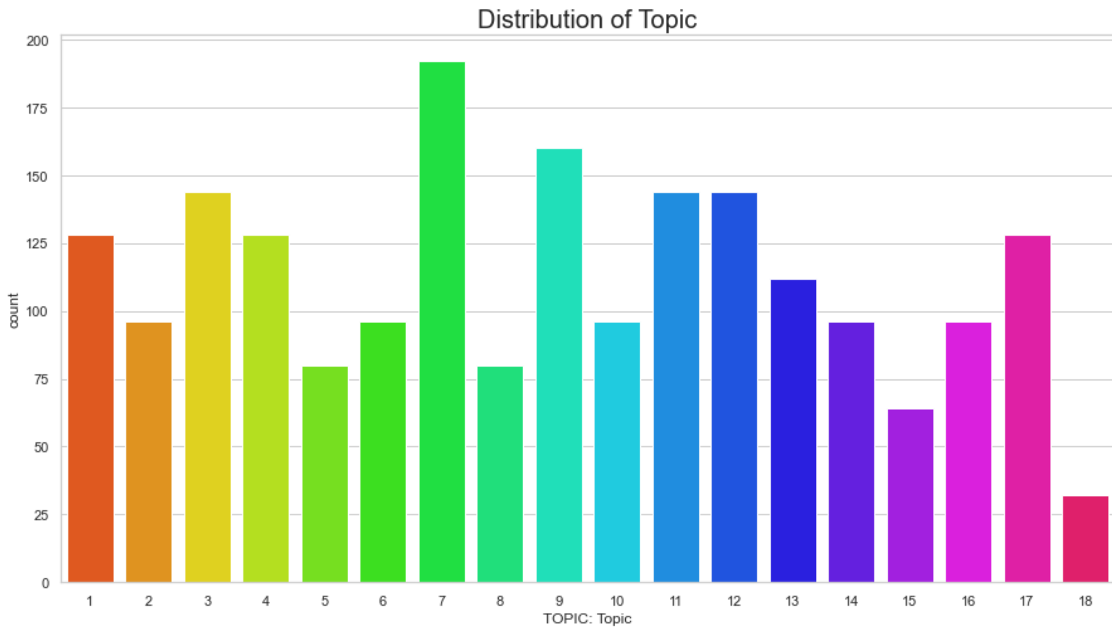
[13]:
```
print('TOPIC: Topic\n')
Z = df['TOPIC: Topic']
counter = Counter(Z)
for k,v in counter.items():
    per = v / len(Z) * 100
    print('Class=%s, Count=%d, Percentage=%.3f%%' % (k, v, per))

print('\nTOPIC: Topic - NUMERICAL\n')
Z_numerical = df_numerical['TOPIC: Topic']
counter = Counter(Z_numerical)
for k,v in counter.items():
    per = v / len(Z_numerical) * 100
    print('Class=%s, Count=%d, Percentage=%.3f%%' % (k, v, per))

plt.rcParams['figure.figsize'] = (15, 8)
sns.countplot(df_numerical['TOPIC: Topic'], palette = 'hsv')
plt.title('Distribution of Topic', fontsize = 20)
plt.show()
```

```
TOPIC: Topic

Class=VAW_TOPIC_001: Types of violence against women by partner, Count=160,
Percentage=7.937%
Class=VAW_TOPIC_002: Partner Physical violence by severity, Count=32,
Percentage=1.587%
Class=VAW_TOPIC_003: Act of physical violence by partners, Count=192,
Percentage=9.524%
Class=VAW_TOPIC_004: Acts of sexual violence by partners, Count=96,
Percentage=4.762%
Class=VAW_TOPIC_005: Acts of emotional  violence by partners, Count=96,
Percentage=4.762%
Class=VAW_TOPIC_006: Acts of controlling behaviours by partners, Count=128,
Percentage=6.349%
Class=VAW_TOPIC_007: Types of violence against women by others (non-partners),
Count=144, Percentage=7.143%
```

Class=VAW_TOPIC_008: Non-Partner Physical violence by type of perpetrator, Count=128, Percentage=6.349%
Class=VAW_TOPIC_009: Non-Partner Sexual violence by type of perpetrator, Count=128, Percentage=6.349%
Class=VAW_TOPIC_010: Child sexual abuse prevalence by type of perpetrator, Count=80, Percentage=3.968%
Class=VAW_TOPIC_011: Injuries from physical or sexual partner violence, Count=144, Percentage=7.143%
Class=VAW_TOPIC_012: Impact of partner violence on women's health and wellbeing, Count=144, Percentage=7.143%
Class=VAW_TOPIC_013: Impact of partner violence on women's reproductive health, Count=96, Percentage=4.762%
Class=VAW_TOPIC_014: Impact of partner violence on women's work who work for money, Count=64, Percentage=3.175%
Class=VAW_TOPIC_015: Impact of partner violence on the wellbeing of children, Count=96, Percentage=4.762%
Class=VAW_TOPIC_016: Responses to partner violence - women told others about violence or leave home, Count=112, Percentage=5.556%
Class=VAW_TOPIC_017: Seeking and receiving help, Count=80, Percentage=3.968%
Class=VAW_TOPIC_018: Reasons for seeking help, Count=96, Percentage=4.762%

TOPIC: Topic - NUMERICAL

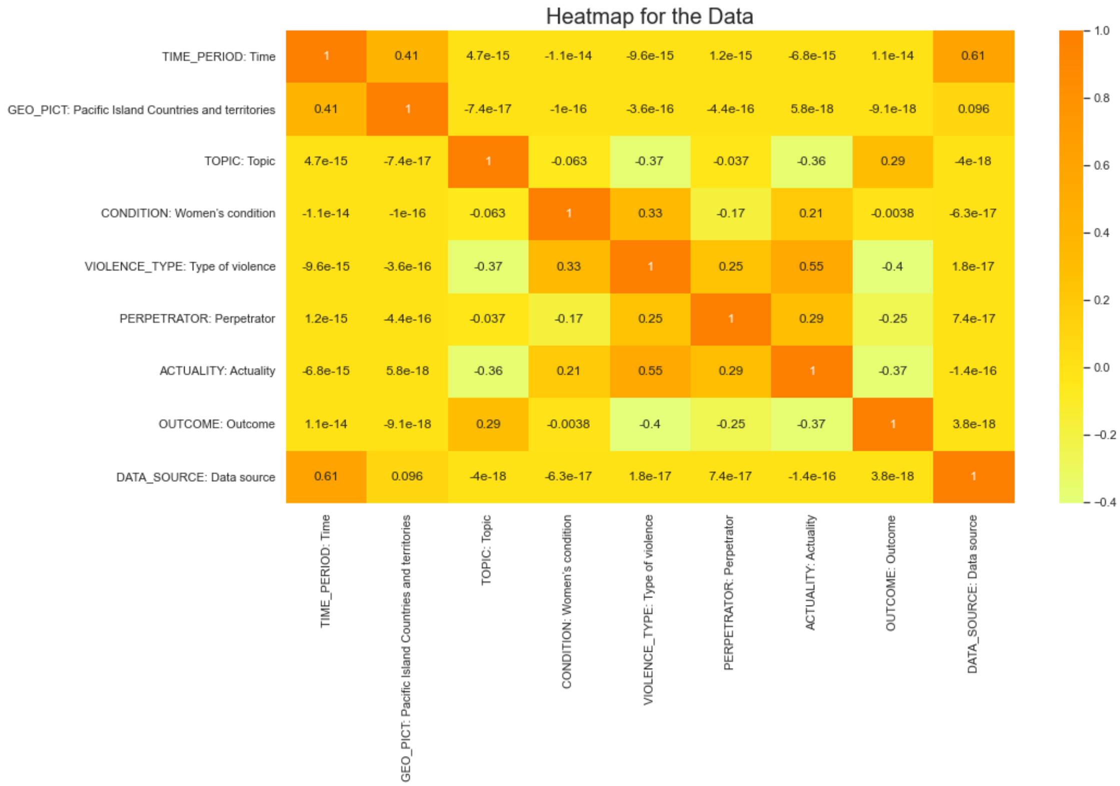
Class=9, Count=160, Percentage=7.937%
Class=18, Count=32, Percentage=1.587%
Class=7, Count=192, Percentage=9.524%
Class=2, Count=96, Percentage=4.762%
Class=10, Count=96, Percentage=4.762%
Class=1, Count=128, Percentage=6.349%
Class=12, Count=144, Percentage=7.143%
Class=4, Count=128, Percentage=6.349%
Class=17, Count=128, Percentage=6.349%
Class=8, Count=80, Percentage=3.968%
Class=3, Count=144, Percentage=7.143%
Class=11, Count=144, Percentage=7.143%
Class=16, Count=96, Percentage=4.762%
Class=15, Count=64, Percentage=3.175%
Class=14, Count=96, Percentage=4.762%
Class=13, Count=112, Percentage=5.556%
Class=5, Count=80, Percentage=3.968%
Class=6, Count=96, Percentage=4.762%



The Graph shows a more Interactive Chart about the distribution of each Topic, in other words the different incidences in our set for more clarity about the cases of violence where women fall victim.

The seventh class represents the acts of physical violence by partners which is the most frequent attack. With the exception of three cases the other classes are approximately at the same levels.

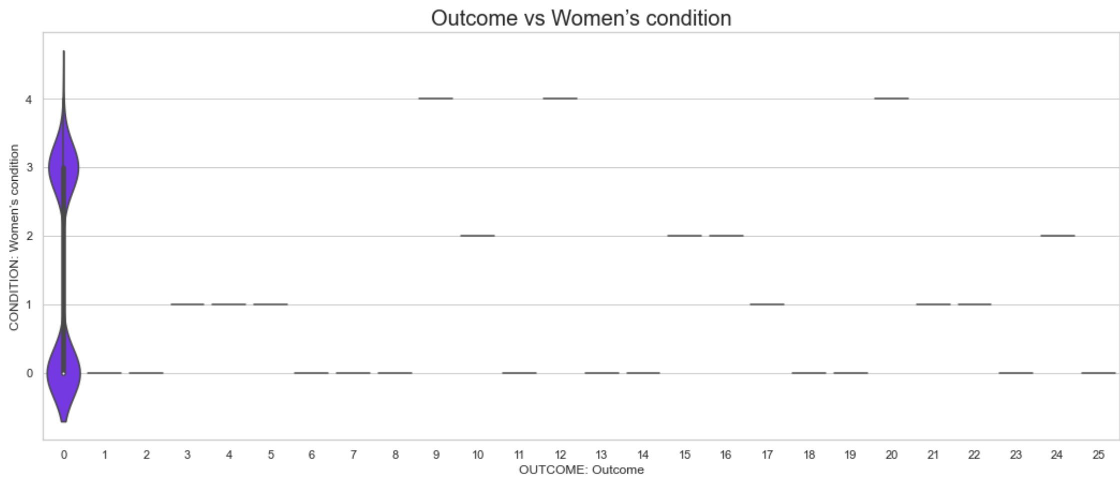
[14]:
```
plt.rcParams['figure.figsize'] = (15, 8)
sns.heatmap(df_numerical.corr(), cmap = 'Wistia', annot = True)
plt.title('Heatmap for the Data', fontsize = 20)
plt.show()
```



The Above Graph for Showing the correlation between the different attributes of dataset, this Heatmap reflects the most correlated features with Orange Color and least correlated features with yellow color.

We can clearly see that these attributes do not have good correlation among them, that's why we will proceed with all of the features.

[15]:
```
plt.rcParams['figure.figsize'] = (18, 7)
sns.violinplot(df_numerical['OUTCOME: Outcome'], df_numerical['CONDITION:␣
↪Women’s condition'],palette = 'rainbow')
plt.title('Outcome vs Women’s condition', fontsize = 20)
plt.show()
```



- A Bivariate Analysis between the Outcome and the Women's condition, to better visualize the Outcome of the different cases.
- In the most cases women are victims of an unspecified form of attack and they are gathered around the 0 (ANY) and 3 (EVPART) classes

Chapter 2

1st APPROACH - ALL LABELS

In the first implementation we face a MultiClass problem and we use all the different labels of the OUTCOME column in order to build a classifier model.

2.1 PREPROCESSING

2.1.1 COUNT THE MISSING VALUES

[16]:
```
# COUNT MISSING OR NULL VALUES
df = df.replace('_T: Any', np.NaN)
Y_null = df['OUTCOME: Outcome']
X_null = df.loc[:, df.columns != 'OUTCOME: Outcome']

x_null = X_null.isnull().sum()
y_null = Y_null.isnull().sum()
print('\nMissing values in X:\n', x_null)
print('\nNumber of rows with OUTCOME ANY:\n', y_null)
```

```
Missing values in X:
 DATAFLOW                                                0
FREQ: Frequency                                        0
TIME_PERIOD: Time                                      0
GEO_PICT: Pacific Island Countries and territories     0
TOPIC: Topic                                           0
INDICATOR: Indicator                                   0
SEX: Sex                                               0
AGE: Age                                               0
CONDITION: Women's condition                        1056
VIOLENCE_TYPE: Type of violence                      704
PERPETRATOR: Perpetrator                             256
ACTUALITY: Actuality                                 832
LIFEPER: Period of life                             1536
RESPONSE: Response                                  1904
```

```
HELP_REASON: Reason for searching help              1920
HELP_PROVIDER: Help provider                        1936
OBS_VALUE                                            782
UNIT_MEASURE: Unit of measure                          0
UNIT_MULT: Unit multiplier                          2016
OBS_STATUS: Observation Status                      1253
DATA_SOURCE: Data source                               0
OBS_COMMENT: Comment                                1003
dtype: int64

Number of rows with OUTCOME ANY:
 1472
```

2.1.2 SPLIT DATASET BASED ON OUTCOME COLUMN

We split the original dataset in 2 parts. The first dataset "df_not_any" contains all the rows that correspond to specific label in column "OUTCOME", while the second dataframe "df_with_any" contains all the rows that have ANY as value in the "OUTCOME" column.

[17]:
```
df_not_any = df[df['OUTCOME: Outcome'].notnull()]
X = df_not_any.loc[:, df.columns != 'OUTCOME: Outcome']
Y = df_not_any['OUTCOME: Outcome']
print("\nDATASET SHAPE WITH OUTCOME!=ANY: \n", df_not_any.shape)

df_with_any = df[df['OUTCOME: Outcome'].isnull()]
X_with_any = df_with_any.loc[:, df.columns != 'OUTCOME: Outcome'].reset_index()
print("DATASET SHAPE WITH OUTCOME=ANY: \n", df_with_any.shape)
```

```
DATASET SHAPE WITH OUTCOME!=ANY:
 (544, 23)
DATASET SHAPE WITH OUTCOME=ANY:
 (1472, 23)
```

2.1.3 SPLIT DATASET IN TRAIN AND TEST SET

We use the first dataset with rows that have specific label as "OUTCOME" in order to create our train and test set. Then we count the null values per column.

[18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=0)
print("\nX_train shape: \n", X_train.shape)
x_null = X_train.isnull().sum()
print("\nNumber of missing values in X_train before drop: \n", x_null)
```

```
X_train shape:
 (435, 22)
```

```
Number of missing values in X_train before drop:
 DATAFLOW                                                0
FREQ: Frequency                                         0
TIME_PERIOD: Time                                       0
GEO_PICT: Pacific Island Countries and territories      0
TOPIC: Topic                                            0
INDICATOR: Indicator                                    0
SEX: Sex                                                0
AGE: Age                                                0
CONDITION: Women's condition                          220
VIOLENCE_TYPE: Type of violence                       328
PERPETRATOR: Perpetrator                              120
ACTUALITY: Actuality                                  328
LIFEPER: Period of life                               394
RESPONSE: Response                                    435
HELP_REASON: Reason for searching help                435
HELP_PROVIDER: Help provider                          435
OBS_VALUE                                             205
UNIT_MEASURE: Unit of measure                           0
UNIT_MULT: Unit multiplier                            435
OBS_STATUS: Observation Status                        237
DATA_SOURCE: Data source                                0
OBS_COMMENT: Comment                                  218
dtype: int64
```

Having in mind that X_train has 435 rows, we understand that columns RESPONSE, HELP_REASON, HELP_PROVIDER and UNIT_MULT contain only null values in train set and we can drop them. We also believe that OBS_COMMENT and DATA_SOURCE columns don't provide any valuable information for the prediction of the OUTCOME and this is why we drop them too.

[19]:
```
X_train.drop(['RESPONSE: Response', 'HELP_REASON: Reason for searching help',␣
 ↪'HELP_PROVIDER: Help provider', 'UNIT_MULT: Unit multiplier',  'OBS_COMMENT:␣
 ↪Comment', 'DATA_SOURCE: Data source'], axis=1, inplace=True)
X_test.drop(['RESPONSE: Response', 'HELP_REASON: Reason for searching help',␣
 ↪'HELP_PROVIDER: Help provider', 'UNIT_MULT: Unit multiplier', 'OBS_COMMENT:␣
 ↪Comment', 'DATA_SOURCE: Data source'], axis=1, inplace=True)
x_null = X_train.isnull().sum()
print("\nNumber of missing values in X_train after drop and before impute: \n",␣
 ↪x_null)
```

```
Number of missing values in X_train after drop and before impute:
 DATAFLOW                                                0
FREQ: Frequency                                         0
TIME_PERIOD: Time                                       0
GEO_PICT: Pacific Island Countries and territories      0
TOPIC: Topic                                            0
```

```
INDICATOR: Indicator                          0
SEX: Sex                                      0
AGE: Age                                      0
CONDITION: Women’s condition                220
VIOLENCE_TYPE: Type of violence             328
PERPETRATOR: Perpetrator                    120
ACTUALITY: Actuality                        328
LIFEPER: Period of life                     394
OBS_VALUE                                   205
UNIT_MEASURE: Unit of measure                 0
OBS_STATUS: Observation Status              237
dtype: int64
```

2.1.4 COUNT NUMBER OF DIFFERENT CATEGORIES IN EACH COLUMN

[20]:
```
for col_name in X_train.columns:
    unique_cat = len(X_train[col_name].unique())
    print("Feature '{col_name}' has {unique_cat} unique cotegories".
 ↪format(col_name=col_name, unique_cat=unique_cat))
```

```
Feature 'DATAFLOW' has 1 unique cotegories
Feature 'FREQ: Frequency' has 1 unique cotegories
Feature 'TIME_PERIOD: Time' has 10 unique cotegories
Feature 'GEO_PICT: Pacific Island Countries and territories' has 13 unique
cotegories
Feature 'TOPIC: Topic' has 5 unique cotegories
Feature 'INDICATOR: Indicator' has 1 unique cotegories
Feature 'SEX: Sex' has 1 unique cotegories
Feature 'AGE: Age' has 1 unique cotegories
Feature 'CONDITION: Women’s condition' has 4 unique cotegories
Feature 'VIOLENCE_TYPE: Type of violence' has 4 unique cotegories
Feature 'PERPETRATOR: Perpetrator' has 2 unique cotegories
Feature 'ACTUALITY: Actuality' has 3 unique cotegories
Feature 'LIFEPER: Period of life' has 2 unique cotegories
Feature 'OBS_VALUE' has 174 unique cotegories
Feature 'UNIT_MEASURE: Unit of measure' has 1 unique cotegories
Feature 'OBS_STATUS: Observation Status' has 2 unique cotegories
```

2.1.5 IMPUTER, ONE-HOT-ENCODER AND SCALER

In order to fill the missing values in OBS_VALUE column which contains numerical values, we use Simple Imputer with the "mean" strategy. Then for both TIME_PERIOD and OBS_VALUE we use MinMax scaler.

For the missing values in Categorical columns we also use Simple Imputer but with the "most-frequent" strategy. Then we pass the values in the OneHotEncoder in order to transform them to numerical.

[21]:
```
numeric_features = ['TIME_PERIOD: Time', 'OBS_VALUE']
# Simple Imputer and Scaler for Numerical
numeric_transformer = Pipeline(
    steps=[("imputer", SimpleImputer(missing_values=np.nan, strategy="mean")),␣
 ↪("scaler", preprocessing.MinMaxScaler())])

categorical_features = ['DATAFLOW', 'FREQ: Frequency', 'GEO_PICT: Pacific Island␣
 ↪Countries and territories',
                        'TOPIC: Topic', 'INDICATOR: Indicator', 'SEX: Sex', 'AGE:
 ↪ Age', 'CONDITION: Women's condition',
                        'VIOLENCE_TYPE: Type of violence', 'PERPETRATOR:␣
 ↪Perpetrator', 'ACTUALITY: Actuality',
                         'LIFEPER: Period of life', 'UNIT_MEASURE: Unit of␣
 ↪measure', 'OBS_STATUS: Observation Status']

# Simple Imputer and One Hot Encoder for Categorical
categorical_transformer = Pipeline(
    steps=[("imputer", SimpleImputer(missing_values=np.nan,␣
 ↪strategy="most_frequent")), ("ohe", OneHotEncoder(handle_unknown='ignore',␣
 ↪sparse=False))])

preprocessor = ColumnTransformer(
    remainder='passthrough', #passthough features not listed
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)
# fit preprocessor
# transform train+test
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)
# to dataframe
column_names = preprocessor.transformers_[1][1]\
   .named_steps['ohe'].get_feature_names_out(categorical_features).tolist()

X_train = pd.DataFrame(X_train, columns=numeric_features+column_names)
X_test = pd.DataFrame(X_test, columns=numeric_features+column_names)
```

2.1.6 FEATURE IMPORTANCE

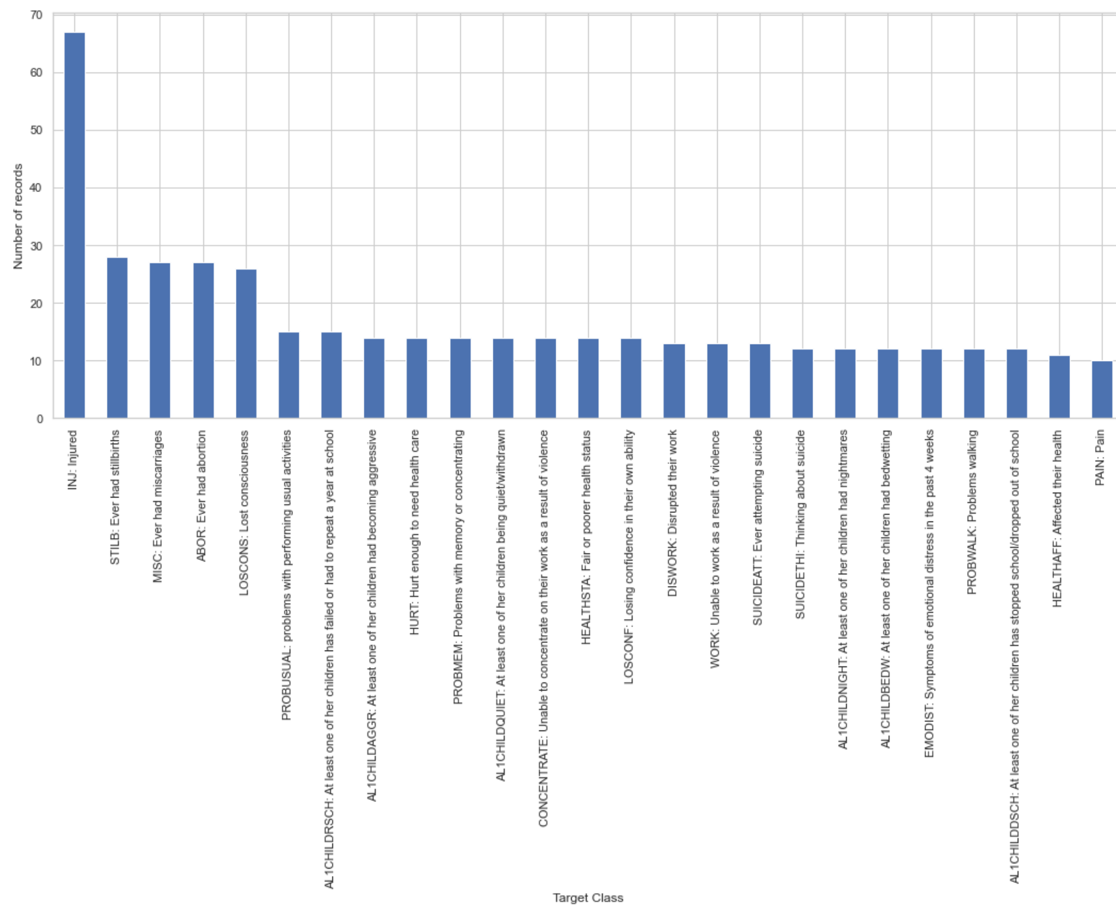
In order to test the feature importance we use the Random Forest model.

[22]:
```
rf = RandomForestClassifier()

rf.fit(X_train, y_train)
```

```
y_pred = rf.predict(X_test)

feature_importances = pd.DataFrame(rf.feature_importances_, index=X_train.
 ↪columns,
                                    columns=['importance']).
 ↪sort_values('importance', ascending=False)

print(feature_importances)
count = y_train.value_counts()
count.plot.bar()
plt.ylabel('Number of records')
plt.xlabel('Target Class')
plt.show()
# column_names of rows with 0.0 importance = columns with only 1 category
columns_with_0_importance = feature_importances[(feature_importances == 0).
 ↪all(axis=1)].index.tolist()
# ==================== DROP COLUMNS WITH 1 category IN TRAIN SET AND 0.0␣
 ↪FEATURE IMPORTANCE ====================
X_train.drop(columns_with_0_importance, axis=1, inplace=True)
X_test.drop(columns_with_0_importance, axis=1, inplace=True)
```

```
                                                    importance
OBS_VALUE                                             0.385201
TIME_PERIOD: Time                                     0.068517
TOPIC: Topic_VAW_TOPIC_011: Injuries from physi...    0.063643
TOPIC: Topic_VAW_TOPIC_013: Impact of partner v...    0.063294
VIOLENCE_TYPE: Type of violence_PHYSORSEX: Phys...    0.052571
TOPIC: Topic_VAW_TOPIC_012: Impact of partner v...    0.039820
CONDITION: Women's condition_EVPREG: Ever pregnant    0.030141
TOPIC: Topic_VAW_TOPIC_015: Impact of partner v...    0.024653
CONDITION: Women's condition_W4M: Working for m...    0.021137
ACTUALITY: Actuality_ALO12M: At least once in t...    0.017703
GEO_PICT: Pacific Island Countries and territor...    0.016662
TOPIC: Topic_VAW_TOPIC_014: Impact of partner v...    0.016483
GEO_PICT: Pacific Island Countries and territor...    0.016367
ACTUALITY: Actuality_ALOLIFE: At least once in ...    0.014883
VIOLENCE_TYPE: Type of violence_PHYS: Physical ...    0.014522
CONDITION: Women's condition_CHI614: With child...    0.014013
GEO_PICT: Pacific Island Countries and territor...    0.013920
GEO_PICT: Pacific Island Countries and territor...    0.013684
GEO_PICT: Pacific Island Countries and territor...    0.012988
GEO_PICT: Pacific Island Countries and territor...    0.012973
GEO_PICT: Pacific Island Countries and territor...    0.012956
GEO_PICT: Pacific Island Countries and territor...    0.012229
GEO_PICT: Pacific Island Countries and territor...    0.011799
GEO_PICT: Pacific Island Countries and territor...    0.011205
GEO_PICT: Pacific Island Countries and territor...    0.010469
```

```
GEO_PICT: Pacific Island Countries and territor...    0.010302
GEO_PICT: Pacific Island Countries and territor...    0.010188
VIOLENCE_TYPE: Type of violence_SEX: Sexual vio...    0.007676
INDICATOR: Indicator_NUMPERRF: Number of person...    0.000000
SEX: Sex_F: Female                                    0.000000
AGE: Age_Y15T64: 15-64                                0.000000
FREQ: Frequency_A: Annual                             0.000000
DATAFLOW_SPC:DF_VAW(1.0)                              0.000000
PERPETRATOR: Perpetrator_PARTNER: Partner             0.000000
LIFEPER: Period of life_PREGNANCY: During pregn...    0.000000
UNIT_MEASURE: Unit of measure_PERCENT: percent        0.000000
OBS_STATUS: Observation Status_O: Missing value       0.000000
```



As we see the columns that contain only one category have zero importance and we can drop them from our train and test set.

2.1.7 OVERSAMPLE

Because the distribution of our train-set samples in each OUTCOME label is imbalanced we implement an oversampling technique in order to create balanced clusters. We tested both the RandomOverSampler as well as the SMOTE oversampler and we noticed better results with the RandomOverSampler. Then we test the distribution of our data again.

[23]:
```
print('BEFORE OVERSAMPLING\n')
counter = Counter(y_train)
for k,v in counter.items():
    per = v / len(y_train) * 100
    print('Class=%s, Count=%d, Percentage=%.3f%%' % (k, v, per))

# smote = SMOTE(random_state=0)
# X_train, y_train = smote.fit_resample(X_train, y_train)

ros = RandomOverSampler(random_state=0)
X_train, y_train = ros.fit_resample(X_train, y_train)
print('\nAFTER OVERSAMPLING\n')
counter = Counter(y_train)
for k,v in counter.items():
    per = v / len(y_train) * 100
    print('Class=%s, Count=%d, Percentage=%.3f%%' % (k, v, per))
```

```
BEFORE OVERSAMPLING

Class=AL1CHILDNIGHT: At least one of her children had nightmares, Count=12,
Percentage=2.759%
Class=DISWORK: Disrupted their work, Count=13, Percentage=2.989%
Class=LOSCONS: Lost consciousness, Count=26, Percentage=5.977%
Class=AL1CHILDDSCH: At least one of her children has stopped school/dropped out
of school, Count=12, Percentage=2.759%
Class=PROBUSUAL: problems with performing usual activities, Count=15,
Percentage=3.448%
Class=LOSCONF: Losing confidence in their own ability, Count=14,
Percentage=3.218%
Class=INJ: Injured, Count=67, Percentage=15.402%
Class=PROBWALK: Problems walking, Count=12, Percentage=2.759%
Class=CONCENTRATE: Unable to concentrate on their work as a result of violence,
Count=14, Percentage=3.218%
Class=HEALTHAFF: Affected their health, Count=11, Percentage=2.529%
Class=HURT: Hurt enough to need health care, Count=14, Percentage=3.218%
Class=ABOR: Ever had abortion, Count=27, Percentage=6.207%
Class=PAIN: Pain, Count=10, Percentage=2.299%
Class=AL1CHILDRSCH: At least one of her children has failed or had to repeat a
year at school, Count=15, Percentage=3.448%
Class=PROBMEM: Problems with memory or concentrating, Count=14,
Percentage=3.218%
Class=EMODIST: Symptoms of emotional distress in the past 4 weeks, Count=12,
Percentage=2.759%
Class=AL1CHILDBEDW: At least one of her children had bedwetting, Count=12,
Percentage=2.759%
```

Class=AL1CHILDQUIET: At least one of her children being quiet/withdrawn, Count=14, Percentage=3.218%
Class=STILB: Ever had stillbirths, Count=28, Percentage=6.437%
Class=MISC: Ever had miscarriages, Count=27, Percentage=6.207%
Class=WORK: Unable to work as a result of violence, Count=13, Percentage=2.989%
Class=AL1CHILDAGGR: At least one of her children had becoming aggressive, Count=14, Percentage=3.218%
Class=HEALTHSTA: Fair or poorer health status, Count=14, Percentage=3.218%
Class=SUICIDETHI: Thinking about suicide, Count=12, Percentage=2.759%
Class=SUICIDEATT: Ever attempting suicide, Count=13, Percentage=2.989%

AFTER OVERSAMPLING

Class=AL1CHILDNIGHT: At least one of her children had nightmares, Count=67, Percentage=4.000%
Class=DISWORK: Disrupted their work, Count=67, Percentage=4.000%
Class=LOSCONS: Lost consciousness, Count=67, Percentage=4.000%
Class=AL1CHILDDSCH: At least one of her children has stopped school/dropped out of school, Count=67, Percentage=4.000%
Class=PROBUSUAL: problems with performing usual activities, Count=67, Percentage=4.000%
Class=LOSCONF: Losing confidence in their own ability, Count=67, Percentage=4.000%
Class=INJ: Injured, Count=67, Percentage=4.000%
Class=PROBWALK: Problems walking, Count=67, Percentage=4.000%
Class=CONCENTRATE: Unable to concentrate on their work as a result of violence, Count=67, Percentage=4.000%
Class=HEALTHAFF: Affected their health, Count=67, Percentage=4.000%
Class=HURT: Hurt enough to need health care, Count=67, Percentage=4.000%
Class=ABOR: Ever had abortion, Count=67, Percentage=4.000%
Class=PAIN: Pain, Count=67, Percentage=4.000%
Class=AL1CHILDRSCH: At least one of her children has failed or had to repeat a year at school, Count=67, Percentage=4.000%
Class=PROBMEM: Problems with memory or concentrating, Count=67, Percentage=4.000%
Class=EMODIST: Symptoms of emotional distress in the past 4 weeks, Count=67, Percentage=4.000%
Class=AL1CHILDBEDW: At least one of her children had bedwetting, Count=67, Percentage=4.000%
Class=AL1CHILDQUIET: At least one of her children being quiet/withdrawn, Count=67, Percentage=4.000%
Class=STILB: Ever had stillbirths, Count=67, Percentage=4.000%
Class=MISC: Ever had miscarriages, Count=67, Percentage=4.000%
Class=WORK: Unable to work as a result of violence, Count=67, Percentage=4.000%
Class=AL1CHILDAGGR: At least one of her children had becoming aggressive, Count=67, Percentage=4.000%
Class=HEALTHSTA: Fair or poorer health status, Count=67, Percentage=4.000%
Class=SUICIDETHI: Thinking about suicide, Count=67, Percentage=4.000%

```
Class=SUICIDEATT: Ever attempting suicide, Count=67, Percentage=4.000%
```

2.1.8 PCA

In order to reduce our number of features we use PCA technique in order to have minimum information loss.

[24]:
```
pca = PCA(n_components=7,  random_state=0)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

2.2 CLASSIFIERS

For the multiclassification problem we implement 4 different classifiers:

- RandomForestClassifier
- SVC
- KNeighborsClassifier
- MLPClassifier

For the SVC we implement Grid Search to find the best parameters. GridSearch CV helps to identify the parameters that will improve the performance for this particular model. Here, we should not confuse best_score_ attribute of grid_search with the score method on the test-set. The score method on the test-set gives the generalization performance of the model. Using the score method, we employ a model trained on the whole training set.

[29]:
```
# ================== MODEL RFC ========================
model_rf = RandomForestClassifier(random_state=0)
model_rf.fit(X_train, y_train)
y_pred = model_rf.predict(X_test)
# print(X_test.shape)
# y_test = y_test.to_numpy()
print("\n\nRANDOM FOREST\n")
print("Labels that do not appear in train set:\n", set(y_test) - set(y_pred))
print("\nRecall score:", recall_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Precision score:", precision_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred, average='macro', zero_division=0))

# print(classification_report(y_test, y_pred))

# ================== Grid-Search ========================


# import GridSearchCV
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
# instantiate classifier with default hyperparameters with kernel=rbf, C=1.0 and␣
↪gamma=auto
svc=SVC()

# declare parameters for hyperparameter tuning
parameters = [ {'C':[1, 10, 100, 1000], 'kernel':['linear']},
               {'C':[1, 10, 100, 1000], 'kernel':['rbf'], 'gamma':[0.1, 0.2, 0.
↪3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]},
               {'C':[1, 10, 100, 1000], 'kernel':['poly'], 'degree': [2,3,4]␣
↪,'gamma':[0.01,0.02,0.03,0.04,0.05]}
              ]

grid_search = GridSearchCV(estimator = svc,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)

grid_search.fit(X_train, y_train)

# examine the best model


# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))


# print parameters that give the best results
print('Parameters that give the best results :','\n\n', (grid_search.
↪best_params_))


# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n', (grid_search.
↪best_estimator_))

# calculate GridSearch CV score on test set

print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.
↪score(X_test, y_test)))

# ================= MODEL SVC =======================
```

```
model_svm = SVC(C=1000, gamma=0.8, kernel='rbf', random_state=0)
model_svm.fit(X_train, y_train)
y_pred = model_svm.predict(X_test)
# print(X_test.shape)

print("\n\nSVC\n")
print("Labels that do not appear in train set:\n", set(y_test) - set(y_pred))
print("\nRecall score:", recall_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Precision score:", precision_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred, average='macro', zero_division=0))

# print(classification_report(y_test, y_pred))

# ================== MODEL KNN ========================
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=10, weights='distance')
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)

print("\n\nKNN\n")
print("Labels that do not appear in train set:\n", set(y_test) - set(y_pred))
print("\nRecall score:", recall_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Precision score:", precision_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred, average='macro', zero_division=0))
# print(classification_report(y_test, y_pred))
# ================== MODEL MLP ========================
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(random_state=1, max_iter=1000)
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)

print("\n\nMLP\n")
print("Labels that do not appear in train set:\n", set(y_test) - set(y_pred))
print("\nRecall score:", recall_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Precision score:", precision_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred, average='macro', zero_division=0))
# print(classification_report(y_test, y_pred))
```

```
# print(y_test.to_numpy())
# print(y_pred)
```

```
RANDOM FOREST

Labels that do not appear in train set:
 {'AL1CHILDDSCH: At least one of her children has stopped school/dropped out of
school'}

Recall score: 0.08434482758620689
Precision score: 0.0775925925925926
Accuracy score: 0.25688073394495414
F1 score: 0.07809523809523808

-------------------------------------------

GridSearch CV best score : 0.2907
Parameters that give the best results :
 {'C': 1000, 'gamma': 0.8, 'kernel': 'rbf'}
Estimator that was chosen by the search :
 SVC(C=1000, gamma=0.8)
GridSearch CV score on test set: 0.2385

-------------------------------------------

SVC

Labels that do not appear in train set:
 {'AL1CHILDDSCH: At least one of her children has stopped school/dropped out of
school'}

Recall score: 0.05772413793103448
Precision score: 0.06952380952380953
Accuracy score: 0.23853211009174313
F1 score: 0.06059816207184628

-------------------------------------------

KNN

Labels that do not appear in train set:
 {'AL1CHILDDSCH: At least one of her children has stopped school/dropped out of
school'}

Recall score: 0.07296551724137931
```

```
Precision score: 0.08040293040293040
Accuracy score: 0.23853211009174313
F1 score: 0.07397979797979798

-------------------------------------------

MLP

Labels that do not appear in train set:
 {'AL1CHILDDSCH: At least one of her children has stopped school/dropped out of
school', 'SUICIDEATT: Ever attempting suicide'}

Recall score: 0.0823448275862069
Precision score: 0.06868783068783069
Accuracy score: 0.24770642201834864
F1 score: 0.07165079365079365
```

2.3 CONCLUSIONS

As we can see the results are very poor. The classifier that has slightly better results is the Random Forest Classifier and we used this model to make the predictions for our second dataset that contains the samples with the unknown OUTCOME. The results are saved in the "OUTCOME_WITH_LABELS.csv" file.

[30]:
```python
# ================ IMPLEMENT MODEL TO DATASET WITH OUTCOME ANY =================

# DROP COLUMNS
X_with_any_processed = X_with_any.copy(deep=True)
X_with_any_processed.drop(['RESPONSE: Response', 'HELP_REASON: Reason for␣
 ↪searching help', 'HELP_PROVIDER: Help provider',
                           'UNIT_MULT: Unit multiplier',  'OBS_COMMENT:␣
 ↪Comment', 'DATA_SOURCE: Data source'], axis=1, inplace=True)

# IMPLEMENT PREPROCESSOR
X_with_any_processed = preprocessor.transform(X_with_any_processed)

# to dataframe
X_with_any_processed = pd.DataFrame(X_with_any_processed,␣
 ↪columns=numeric_features+column_names)

# DROP non-significant columns
X_with_any_processed.drop(columns_with_0_importance, axis=1, inplace=True)
# PCA
X_with_any_processed = pca.transform(X_with_any_processed)

# PREDICT WITH SVM
y_prediction = model_svm.predict(X_with_any_processed)
```

```
y_prediction = pd.DataFrame(y_prediction, columns=['OUTCOME'])
# print(X_with_any.shape)
# print(y_prediction.shape)

predictions_with_any = pd.concat([X_with_any, y_prediction],  keys=X_with_any.
 ↪columns+['OUTCOME'], axis=1)
# print(predictions_with_any.shape)

# predictions_with_any.to_csv('Users/andre/PycharmProjects/final_ML/OUTCOME.
 ↪csv', encoding='utf-8')
```

Chapter 3

2nd APPROACH - MAKE CLUSTERS OF THE LABELS

For the second approach we decided to make clusters of the OUTCOME labels. After a lot of consideration we ended up with 5 clusters which we present above with their corresponding labels.

- CLUSTER-1 - CHILD AFFECTED:
  - AL1CHILDAGGR: At least one of her children had becoming aggressive
  - AL1CHILDBEDW: At least one of her children had bedwetting
  - AL1CHILDDSCH: At least one of her children has stopped school/dropped out of school
  - AL1CHILDNIGHT: At least one of her children had nightmares
  - AL1CHILDQUIET: At least one of her children being quiet/withdrawn
  - AL1CHILDRSCH: At least one of her children has failed or had to repeat a year at school
- CLUSTER-2 - LABOR ISSUES:
  - ABOR: Ever had abortion
  - MISC: Ever had miscarriages
  - STILB: Ever had stillbirths
- CLUSTER-3 - LIFE QUALITY ISSUES:
  - CONCENTRATE: Unable to concentrate on their work as a result of violence
  - LOSCONF: Losing confidence in their own ability
  - WORK: Unable to work as a result of violence
  - DISWORK: Disrupted their work
  - PROBMEM: Problems with memory or concentrating

- PROBUSUAL: problems with performing usual activities

- CLUSTER-4 - LIGHT HEALTH ISSUES:
  - EMODIST: Symptoms of emotional distress in the past 4 weeks
  - HEALTHAFF: Affected their health
  - HEALTHSTA: Fair or poorer health status
  - PAIN: Pain
- CLUSTER-5 - HEAVY HEALTH ISSUES:
  - SUICIDEATT: Ever attempting suicide
  - SUICIDETHI: Thinking about suicide
  - INJ: Injured
  - HURT: Hurt enough to need health care
  - LOSCONS: Lost consciousness

[31]:
```
# ================= MAKE CLUSTERS ===========================
# cluster 1 = CHILD AFFECTED
cluster_1 = ['AL1CHILDAGGR: At least one of her children had becoming␣
↪aggressive',
             'AL1CHILDBEDW: At least one of her children had bedwetting',
             'AL1CHILDDSCH: At least one of her children has stopped school/
↪dropped out of school',
             'AL1CHILDNIGHT: At least one of her children had nightmares',
             'AL1CHILDQUIET: At least one of her children being quiet/withdrawn',
             'AL1CHILDRSCH: At least one of her children has failed or had to␣
↪repeat a year at school']
# cluster 2 = LABOR ISSUES
cluster_2 = ['ABOR: Ever had abortion', 'MISC: Ever had miscarriages', 'STILB:␣
↪Ever had stillbirths']
# cluster 3 = LIFE QUALITY ISSUES
cluster_3 = ['CONCENTRATE: Unable to concentrate on their work as a result of␣
↪violence',
             'LOSCONF: Losing confidence in their own ability',
             'WORK: Unable to work as a result of violence',
             'DISWORK: Disrupted their work',
             'PROBMEM: Problems with memory or concentrating',
             'PROBUSUAL: problems with performing usual activities']
# cluster 4 = LIGHT HEALTH ISSUES
cluster_4 = ['EMODIST: Symptoms of emotional distress in the past 4 weeks',
             'HEALTHAFF: Affected their health',
             'HEALTHSTA: Fair or poorer health status',
             'PAIN: Pain']
# cluster 5 = HEAVY HEALTH ISSUES
cluster_5 = ['SUICIDEATT: Ever attempting suicide',
```

```
            'SUICIDETHI: Thinking about suicide',
            'PROBWALK: Problems walking',
            'INJ: Injured',
            'HURT: Hurt enough to need health care',
            'LOSCONS: Lost consciousness']

# ================= REPLACE OUTCOME WITH CLUSTERS ===============
for i in cluster_1:
    Y = Y.replace(i, 1)
for i in cluster_2:
    Y = Y.replace(i, 2)
for i in cluster_3:
    Y = Y.replace(i, 3)
for i in cluster_4:
    Y = Y.replace(i, 4)
for i in cluster_5:
    Y = Y.replace(i, 5)
```

3.1 PREPROCESSING AND CLASSIFIERS

The below implementation is the same as the above so we just present the code and the results.

[32]:
```
# ================= SPLIT TEST SET AND TRAIN SET ===========================

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=0)

# ================= COUNT NULL IN X_TRAIN ===================

x_null = X_train.isnull().sum()
# print("\nNumber of missing values in X_train before impute: \n", x_null)

# DROP COLUMN UNIT MULTIPLIER THAT ONLY HAS MISSING VALUES IN TRAIN: Number of␣
 ↪missing values in X_train = 435
# ALSO DROP COLUMNS WITHOUT MEANING LIKE OBS_COMMENT AND OBS_VALUE
X_train.drop(['RESPONSE: Response', 'HELP_REASON: Reason for searching help',␣
 ↪'HELP_PROVIDER: Help provider', 'UNIT_MULT: Unit multiplier',  'OBS_COMMENT:␣
 ↪Comment', 'DATA_SOURCE: Data source'], axis=1, inplace=True)
X_test.drop(['RESPONSE: Response', 'HELP_REASON: Reason for searching help',␣
 ↪'HELP_PROVIDER: Help provider', 'UNIT_MULT: Unit multiplier', 'OBS_COMMENT:␣
 ↪Comment', 'DATA_SOURCE: Data source'], axis=1, inplace=True)

# ================= COUNT NULL IN X_TRAIN AFTER IMPUTE===================
x_null = X_train.isnull().sum()
# print("\nNumber of missing values in X_train before impute: \n", x_null)
```

```
# ================= COUNT NUMBER OF CATEGORIES OF EACH COLUMN IN TRAIN␣
↪====================
print('NUMBER OF UNIQUE CATEGORIES OF EACH COLUMN:\n')
for col_name in X_train.columns:
    unique_cat = len(X_train[col_name].unique())
    print("Feature '{col_name}' has {unique_cat} unique cotegories".
↪format(col_name=col_name, unique_cat=unique_cat))

# ================= MOST-FREQUENT-IMPUTER AND ONE-HOT-ENCODING FOR CATEGORICAL␣
↪VALUES AND MEAN-IMPUTER AND MIN-MAX-SCALER FOR␣
↪NUMERICAL===========================
numeric_features = ['TIME_PERIOD: Time', 'OBS_VALUE']
# Simple Imputer and Scaler for Numerical
numeric_transformer = Pipeline(
    steps=[("imputer", SimpleImputer(missing_values=np.nan, strategy="mean")),␣
↪("scaler", preprocessing.MinMaxScaler())])

categorical_features = ['DATAFLOW', 'FREQ: Frequency', 'GEO_PICT: Pacific Island␣
↪Countries and territories',
                        'TOPIC: Topic', 'INDICATOR: Indicator', 'SEX: Sex', 'AGE:
↪ Age', 'CONDITION: Women's condition',
                        'VIOLENCE_TYPE: Type of violence', 'PERPETRATOR:␣
↪Perpetrator', 'ACTUALITY: Actuality',
                         'LIFEPER: Period of life', 'UNIT_MEASURE: Unit of␣
↪measure', 'OBS_STATUS: Observation Status']

# Simple Imputer and One Hot Encoder for Categorical
categorical_transformer = Pipeline(
    steps=[("imputer", SimpleImputer(missing_values=np.nan,␣
↪strategy="most_frequent")), ("ohe", OneHotEncoder(handle_unknown='ignore',␣
↪sparse=False))])

preprocessor = ColumnTransformer(
    remainder='passthrough', #passthough features not listed
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)
# fit preprocessor
# transform train+test
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)
# to dataframe
column_names = preprocessor.transformers_[1][1]\
   .named_steps['ohe'].get_feature_names_out(categorical_features).tolist()
```

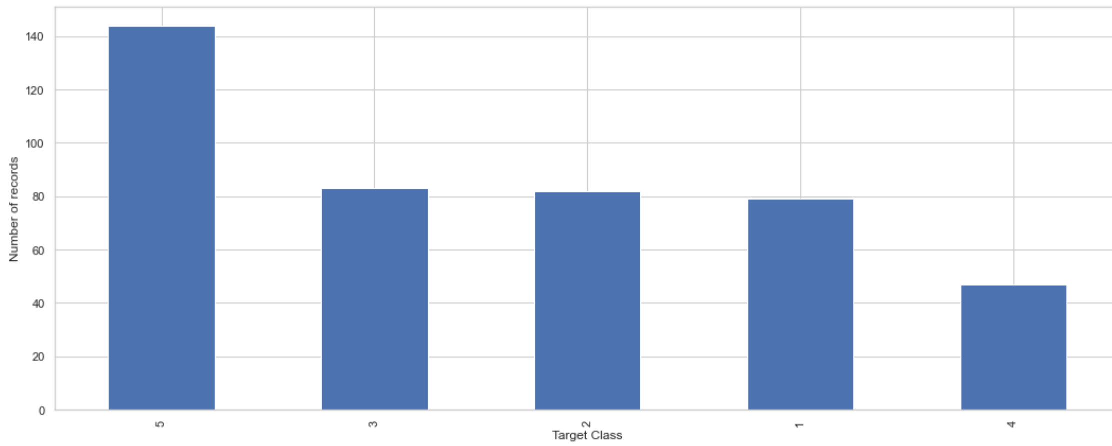
```
X_train = pd.DataFrame(X_train, columns=numeric_features+column_names)
X_test = pd.DataFrame(X_test, columns=numeric_features+column_names)

# ===================== FEATURE IMPORTANCE =====================
rf = RandomForestClassifier()

rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

feature_importances = pd.DataFrame(rf.feature_importances_, index=X_train.
 ↪columns,
                                   columns=['importance']).
 ↪sort_values('importance', ascending=False)

# print(feature_importances)
print('\n\nCLUSTER DISTRIBUTION IN X_TRAIN:\n')
count = y_train.value_counts()
count.plot.bar()
plt.ylabel('Number of records')
plt.xlabel('Target Class')
plt.show()
# column_names of rows with 0.0 importance = columns with only 1 category
columns_with_0_importance = feature_importances[(feature_importances == 0).
 ↪all(axis=1)].index.tolist()

# ==================== DROP COLUMNS WITH 1 category IN TRAIN SET AND 0.0␣
 ↪FEATURE IMPORTANCE =====================
X_train.drop(columns_with_0_importance, axis=1, inplace=True)
X_test.drop(columns_with_0_importance, axis=1, inplace=True)

#  ================== OVERSAMPLE ========================
print('BEFORE OVERSAMPLING\n')
counter = Counter(y_train)
for k,v in counter.items():
    per = v / len(y_train) * 100
    print('Class=%s, Count=%d, Percentage=%.3f%%' % (k, v, per))

# smote = SMOTE(random_state=0)
# X_train, y_train = smote.fit_resample(X_train, y_train)

ros = RandomOverSampler(random_state=0)
X_train, y_train = ros.fit_resample(X_train, y_train)
print('\nAFTER OVERSAMPLING\n')
counter = Counter(y_train)
for k,v in counter.items():
```

```
    per = v / len(y_train) * 100
    print('Class=%s, Count=%d, Percentage=%.3f%%' % (k, v, per))


# ================= PCA ===========================
pca = PCA(n_components=7,  random_state=0)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
# ===================================================== CLASSIFIERS␣
 ↪==========================================

# ================== MODEL RFC ========================
model_rf = RandomForestClassifier(random_state=0)
model_rf.fit(X_train, y_train)
y_pred = model_rf.predict(X_test)
# print(X_test.shape)
# y_test = y_test.to_numpy()
print("\n\nRANDOM FOREST\n")
print("Labels that do not appear in train set:\n", set(y_test) - set(y_pred))
print("\nRecall score:", recall_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Precision score:", precision_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred, average='macro', zero_division=0))

# print(classification_report(y_test, y_pred))

# ================== MODEL SVC ========================
from sklearn.svm import SVC
model_svm = SVC(gamma='scale', random_state=0)
model_svm.fit(X_train, y_train)
y_pred = model_svm.predict(X_test)
# print(X_test.shape)

print("\n\nSVC\n")
print("Labels that do not appear in train set:\n", set(y_test) - set(y_pred))
print("\nRecall score:", recall_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Precision score:", precision_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred, average='macro', zero_division=0))

# print(classification_report(y_test, y_pred))

# ================== MODEL KNN ========================
```

```python
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=10, weights='distance')
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)

print("\n\nKNN\n")
print("Labels that do not appear in train set:\n", set(y_test) - set(y_pred))
print("\nRecall score:", recall_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Precision score:", precision_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred, average='macro', zero_division=0))
# print(classification_report(y_test, y_pred))
# ================== MODEL MLP ========================
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(random_state=1, max_iter=1000)
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)

print("\n\nMLP\n")
print("Labels that do not appear in train set:\n", set(y_test) - set(y_pred))
print("\nRecall score:", recall_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Precision score:", precision_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0))
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred, average='macro', zero_division=0))
# print(classification_report(y_test, y_pred))
# print(y_test.to_numpy())
# print(y_pred)
```

```
NUMBER OF UNIQUE CATEGORIES OF EACH COLUMN:

Feature 'DATAFLOW' has 1 unique cotegories
Feature 'FREQ: Frequency' has 1 unique cotegories
Feature 'TIME_PERIOD: Time' has 10 unique cotegories
Feature 'GEO_PICT: Pacific Island Countries and territories' has 13 unique
cotegories
Feature 'TOPIC: Topic' has 5 unique cotegories
Feature 'INDICATOR: Indicator' has 1 unique cotegories
Feature 'SEX: Sex' has 1 unique cotegories
Feature 'AGE: Age' has 1 unique cotegories
Feature 'CONDITION: Women’s condition' has 4 unique cotegories
Feature 'VIOLENCE_TYPE: Type of violence' has 4 unique cotegories
Feature 'PERPETRATOR: Perpetrator' has 2 unique cotegories
Feature 'ACTUALITY: Actuality' has 3 unique cotegories
```

```
Feature 'LIFEPER: Period of life' has 2 unique cotegories
Feature 'OBS_VALUE' has 174 unique cotegories
Feature 'UNIT_MEASURE: Unit of measure' has 1 unique cotegories
Feature 'OBS_STATUS: Observation Status' has 2 unique cotegories


CLUSTER DISTRIBUTION IN X_TRAIN:
```




```
BEFORE OVERSAMPLING

Class=1, Count=79, Percentage=18.161%
Class=3, Count=83, Percentage=19.080%
Class=5, Count=144, Percentage=33.103%
Class=4, Count=47, Percentage=10.805%
Class=2, Count=82, Percentage=18.851%

AFTER OVERSAMPLING

Class=1, Count=144, Percentage=20.000%
Class=3, Count=144, Percentage=20.000%
Class=5, Count=144, Percentage=20.000%
Class=4, Count=144, Percentage=20.000%
Class=2, Count=144, Percentage=20.000%

-------------------------------------------

RANDOM FOREST

Labels that do not appear in train set:
 set()
```

```
Recall score: 0.8216817496229261
Precision score: 0.7983006535947712
Accuracy score: 0.8073394495412844
F1 score: 0.8070772238514173

-------------------------------------------

SVC

Labels that do not appear in train set:
 set()

Recall score: 0.9080128205128204
Precision score: 0.9096774193548388
Accuracy score: 0.8715596330275229
F1 score: 0.889697357203751

-------------------------------------------

KNN

Labels that do not appear in train set:
 set()

Recall score: 0.8645739064856712
Precision score: 0.8271052631578947
Accuracy score: 0.8348623853211009
F1 score: 0.8390151515151516

-------------------------------------------

MLP

Labels that do not appear in train set:
 set()

Recall score: 0.9080128205128204
Precision score: 0.9096774193548388
Accuracy score: 0.8715596330275229
F1 score: 0.889697357203751
```

3.2 CONCLUSIONS

Clustering our OUTCOME labels increased by far the scores of all the classifiers. SVC and MLP classifiers gave the best results with the F1_score reaching the value of 0.9.

We chose to implement the SVC model for the OUTCOME predictions on the dataset with the unknown values. The code is presented below.

The results were saved in the "OUTCOME_WITH_CLUSTERS.csv" file.

[33]:
```
# ================ IMPLEMENT MODEL TO DATASET WITH OUTCOME ANY =================

# DROP COLUMNS
X_with_any_processed = X_with_any.copy(deep=True)
X_with_any_processed.drop(['RESPONSE: Response', 'HELP_REASON: Reason for␣
↪searching help', 'HELP_PROVIDER: Help provider',
                           'UNIT_MULT: Unit multiplier',  'OBS_COMMENT:␣
↪Comment', 'DATA_SOURCE: Data source'], axis=1, inplace=True)

# IMPLEMENT PREPROCESSOR
X_with_any_processed = preprocessor.transform(X_with_any_processed)

# to dataframe
X_with_any_processed = pd.DataFrame(X_with_any_processed,␣
↪columns=numeric_features+column_names)

# DROP non-significant columns
X_with_any_processed.drop(columns_with_0_importance, axis=1, inplace=True)
# PCA
X_with_any_processed = pca.transform(X_with_any_processed)

# PREDICT WITH SVM
y_prediction = model_svm.predict(X_with_any_processed)
y_prediction = pd.DataFrame(y_prediction, columns=['OUTCOME'])
# print(X_with_any.shape)
# print(y_prediction.shape)

predictions_with_any = pd.concat([X_with_any, y_prediction],  keys=X_with_any.
↪columns+['OUTCOME'], axis=1)
# print(predictions_with_any.shape)
# predictions_with_any.to_csv('OUTCOME_WITH_CLUSTERS.csv', encoding='utf-8')
```