data_cleaning

December 7, 2025

1 Analysis Preparation

To begin cleaning the data and prepare for analysis, I will begin by loading the data and identifying the features I seek to predict.

[44]:
```
import polars as pl
df = pl.read_csv('data/CTDC_global_synthetic_data_v2025.csv')
y_variables = ['isForcedLabour', 'isSexualExploit']
```

My initial filter will restrict the data to entries where the country of exploitation is in North America. Also, I will remove 'type..' features from the dataset. These features contain more detailed information on the types of labor and sexual exploitation experienced and are therefore extremely confounding in terms of the predictors.

[45]:
```
df = (
    df
    .filter(pl.col('CountryOfExploitation').is_in(['USA', 'MEX', 'CAN']))
    .select(pl.exclude([c for c in df.columns if 'type' in c]))
    .drop('yearOfRegistration')
)
print(f"Number of rows in USA before data cleaning: {len(df)}")
```

```
Number of rows in USA before data cleaning: 117575
```

Next, I will remove any rows where there are no 1s for any of the three types of exploitation.

[46]:
```
has_exploit = (
    df['isForcedLabour'].is_not_null() |
    df['isSexualExploit'].is_not_null() |
    df['isOtherExploit'].is_not_null()
)

num_with = len(df.filter(has_exploit))
num_without = len(df.filter(~has_exploit))
original_rows = num_with + num_without
print(f"Number of rows with exploitation data: {num_with}")
print(f"Number of rows without exploitation data: {num_without}")
print(f"Original number of rows: {original_rows}")
```

```
# Filter to only those rows
df = df.filter(has_exploit)
```

```
Number of rows with exploitation data: 91043
Number of rows without exploitation data: 26532
Original number of rows: 117575
```

Now, according to codebook guidelines, I will encode binary features by converting all nulls to 0s. Please see initial pages in guidebook for more information

[47]:
```
feature_groups = {
    "means_of_control": [
        "meansDebtBondageEarnings",
        "meansThreats",
        "meansAbusePsyPhySex",
        "meansFalsePromises",
        "meansDrugsAlcohol",
        "meansDenyBasicNeeds",
        "meansExcessiveWorkHours",
        "meansWithholdDocs"
    ],
    "recruiter_relation": [
        "recruiterRelationIntimatePartner",
        "recruiterRelationFriend",
        "recruiterRelationFamily",
        "recruiterRelationOther"
    ]
}

# encode binary features including target variables
binary_cols = [col for cols in feature_groups.values() for col in cols]
binary_cols = binary_cols + y_variables
df = df.with_columns([
    pl.col(col).fill_null(0).cast(pl.Int64)
    for col in binary_cols
])

# verify targets are now binary and do not contain nulls
for var in binary_cols:
    unique_values = df[var].unique()
    print(f"{var} unique values: {unique_values.to_list()}")
```

```
meansDebtBondageEarnings unique values: [0, 1]
meansThreats unique values: [0, 1]
meansAbusePsyPhySex unique values: [0, 1]
meansFalsePromises unique values: [0, 1]
meansDrugsAlcohol unique values: [0, 1]
meansDenyBasicNeeds unique values: [0, 1]
meansExcessiveWorkHours unique values: [0, 1]
```

```
meansWithholdDocs unique values: [0, 1]
recruiterRelationIntimatePartner unique values: [0, 1]
recruiterRelationFriend unique values: [0, 1]
recruiterRelationFamily unique values: [0, 1]
recruiterRelationOther unique values: [0, 1]
isForcedLabour unique values: [0, 1]
isSexualExploit unique values: [0, 1]
```

Now, I will convert the gender column to dummy variables (one hot encoding). Male will be used as the reference category.

[48]:
```
df = df.with_columns(
    pl.col("gender").fill_null("Unknown"),
)
gender_dummies = df.select(pl.col("gender")).to_dummies()
df = pl.concat([df, gender_dummies], how="horizontal")
df = df.drop(
    ["gender", "gender_Man", "gender_Unknown"]
    )
for c in df.columns:
    print(c)
```

```
ageBroad
citizenship
CountryOfExploitation
traffickMonths
meansDebtBondageEarnings
meansThreats
meansAbusePsyPhySex
meansFalsePromises
meansDrugsAlcohol
meansDenyBasicNeeds
meansExcessiveWorkHours
meansWithholdDocs
isForcedLabour
isSexualExploit
isOtherExploit
recruiterRelationIntimatePartner
recruiterRelationFriend
recruiterRelationFamily
recruiterRelationOther
gender_Trans/Transgender/NonConforming
gender_Woman
```

Next, I will assign numerical encodings to the age bands, beginning at 1 for the youngest grouping and increasing to 9 for the eldest. Unknown ages will receive 0.

[49]:
```
# identify range of values
print(df['ageBroad'].unique().to_list())
```

```
age_mapping = {
    "0--8": 1,
    "09--17": 2,
    "18--20": 3,
    "21--23": 4,
    "24--26": 5,
    "27--29": 6,
    "30--38": 7,
    "39--47": 8,
    "48+": 9,
    "None": 0  # Unknown age
}
df = df.with_columns(
    pl.col('ageBroad')
    .fill_null("None")
    .replace(age_mapping).cast(pl.Int64)
    .alias('ageBroad')
)
# ensure range of values is now integer mapping
print(df['ageBroad'].unique().to_list())
```

```
['27--29', '0--8', '09--17', '39--47', None, '18--20', '30--38', '21--23', '24--
26', '48+']
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

I will follow a similar process for mapping the duration of the trafficking experience:

[50]:
```
print(df['traffickMonths'].unique().to_list())
duration_mapping = {
    "0--12 (0-1 yr)": 1,
    "13--24 (1-2 yrs)": 2,
    "25+ (2+ yrs)": 3,
    "None": 0  # Unknown duration
}
df = df.with_columns(
    pl.col("traffickMonths")
    .fill_null("None")
    .replace(duration_mapping).cast(pl.Int64)
    .alias("traffickMonths")
)

print(df['traffickMonths'].unique().to_list())
```

```
['0--12 (0-1 yr)', '25+ (2+ yrs)', '13--24 (1-2 yrs)', None]
[0, 1, 2, 3]
```

Now I will create a derived binary feature indicating whether the victim's citizenship is the same as the country of exploitation.

[51]:
```
citizenship_in_country = df['citizenship'] == df['CountryOfExploitation']
df = df.with_columns(
    pl.when(citizenship_in_country)
    .then(1)
    .otherwise(0)
    .alias('isCitizenOfCountry')
)
df = df.drop('citizenship', 'CountryOfExploitation')
```

Now, I will create a feature for my own use that will not be used as a predictor. This will be a string representation of the exploitation types indicated in that row. I will also use it to count the combinations of exploitation types seen to gain an understanding of distributions.

example: isForcedLabor=1, isSexualExploit=1, isOtherExploit=0 -> Labor + Sexual

[52]:
```
def identify_exploit_combinations(df):
    df = df.with_columns([
        (
            pl.col('isForcedLabour').cast(pl.String)
            + pl.col('isSexualExploit').cast(pl.String)
            + pl.col('isOtherExploit').cast(pl.String)
        ).alias('combo_code')
    ])
    code_to_label = {
        '100': 'Labour Only',
        '010': 'Sexual Only',
        '001': 'Other Only',
        '000': 'None',
        '110': 'Labour + Sexual',
        '101': 'Labour + Other',
        '011': 'Sexual + Other',
        '111': 'Labour + Sexual + Other',
    }
    # add labels to original rows
    df = df.with_columns(
        pl.col('combo_code').replace(code_to_label).alias('exploitation_types')
    )
    # counts for categories that actually occur
    combo_counts = (
        df
        .group_by('exploitation_types')
        .len()
        .rename({'len': 'count'})
    )

    # full set of labels
    all_labels = [
        'Labour Only',
```

```
        'Sexual Only',
        'Other Only',
        'Labour + Sexual',
        'Labour + Other',
        'Sexual + Other',
        'Labour + Sexual + Other',
        'None'

    ]
    # reindex so missing combos appear with 0
    combo_counts = (
        pl.DataFrame({'exploitation_types': all_labels})
        .join(combo_counts, on='exploitation_types', how='left')
        .with_columns(
            pl.col('count').fill_null(0).cast(pl.UInt32)
        )
        .sort('count', descending=True)
    )

    print(combo_counts)
    df = df.drop('combo_code', 'isOtherExploit')
    return df

df = identify_exploit_combinations(df)
```

```
shape: (8, 2)

 exploitation_types       count
 ---                      ---
 str                      u32

 Other Only               65
 Labour + Other           4
 Labour Only              0
 Sexual Only              0
 Labour + Sexual          0
 Sexual + Other           0
 Labour + Sexual + Other  0
 None                     0
```

Above, we see that there are no instances of "None", and a small number of instances after combinations of labor + sexual exploitation. For this reason, I will not consider "Other" as its own category of exploitation. Not due to a lack of value, but due to the nature of the dataset at hand.

Now, I will rename the columns for stylistic reasons.

[53]:
```
df = (
    df
    .rename({
        "ageBroad": "Age Band",
        "traffickMonths": "Trafficking Duration",
        "meansDebtBondageEarnings": "Means Debt Bondage Earnings",
        "meansThreats": "Means Threats",
        "meansAbusePsyPhySex": "Means Abuse",
        "meansFalsePromises": "Means False Promises",
        "meansDrugsAlcohol": "Means Drugs Alcohol",
        "meansDenyBasicNeeds": "Means Deny Basic Needs",
        "meansExcessiveWorkHours": "Means Excessive Work Hours",
        "meansWithholdDocs": "Means Withhold Docs",
        "recruiterRelationIntimatePartner": "Recruiter Relation Intimate␣
 ↪Partner",
        "recruiterRelationFriend": "Recruiter Relation Friend",
        "recruiterRelationFamily": "Recruiter Relation Family",
        "recruiterRelationOther": "Recruiter Relation Other",
        "gender_Trans/Transgender/NonConforming": "Gender:Transgender/
 ↪NonConforming",
        "gender_Woman": "Gender:Woman",
        "isCitizenOfCountry": "Is Citizen of Country"
        })
)
```

[58]:
```
print(df.describe())
```

```
shape: (9, 21)

 statistic  Age Band  Trafficki  Means      …  Gender:Tr  Gender:Wo  Is
 exploitat
 ---        ---       ng         Debt          ansgender  man
Citizen    ion_types
 str        f64       Duration   Bondage       /NonConfo  ---        of
 ---
                      ---        Earnings      rmi…       f64
Country    str
                      f64        ---           ---                   ---

                                 f64           f64                   f64


 count      91043.0   91043.0    91043.0    …  91043.0    91043.0
91043.0    69
 null_coun  0.0       0.0        0.0        …  0.0        0.0        0.0
 90974
```

```
  t

  mean        2.007238   0.00078     0.159485    …   0.002977    0.773843
0.113177    null
  std         2.643895   0.034274    0.36613     …   0.054477    0.418344
0.316811    null
  min         0.0        0.0         0.0         …   0.0         0.0         0.0
  Labour +

  Other
  25%         0.0        0.0         0.0         …   0.0         1.0         0.0
  null
  50%         0.0        0.0         0.0         …   0.0         1.0         0.0
  null
  75%         3.0        0.0         0.0         …   0.0         1.0         0.0
  null
  max         9.0        3.0         1.0         …   1.0         1.0         1.0
  Other

  Only
```

We see 0 in the "null_count" column for all features, so we are good to go. I will save the final dataset and begin analysis:

[55]:
```
df.write_csv('data/final_data.csv')
```