Machine_Learning_Project

April 14, 2026

Ben de Sousa, Edward Fry, Nolan Willoughby

[57]:
```
# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import ConfusionMatrixDisplay

# sklearn modules
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler # scaling features
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV, KFold, StratifiedKFold
from sklearn import metrics

# the warnings are too much for me
import warnings
warnings.filterwarnings('ignore')

# stats modules
from scipy.stats import chi2_contingency

# neural network modules
import keras

# Constants
DATA_FOLDER: str = "./data/"
UNICEF_DATA_CSV: str = DATA_FOLDER + "unicef_malawi.csv"
OUTPUT_FOLDER: str = "./output/"
```

1 Introduction

Mental health conditions affect 8% of children 5-9 globally, and 15% of those aged 10-19 (Unicef, 2023). In lower and middle-income countries (LMICs) access to mental health resources are limited. The early-detection of mental health conditions is important in treatment, especially in children, where their brains are rapidly developing and early intervention can reduce progression of depression from childhood into adult life. UNICEF aims to address this.They have provided us with their Multiple Indicator Cluster Survey (MICS) data, consisting of information on the child, parent, and household, collected from asking questions via a series of telephone calls. In this report we present, key insights into the features in MICS that best predict childhood depression in that data; highlighting important signals of depression. Though causality analysis these signals naturally lead into our recommendations.

The dataset in its raw form contains 87 unique features. In our EDA we highlight the most important 27 raw features which are expressed in terms of 78 processed features. We prioritise model interpretability because our model will inform resource allocation decisions aimed at reducing childhood depression in LMICs. Predictions must be explainable and defensible, supporting decisions that can be justified with clear logical argument. For this reason, we avoid deep neural networks in favour of inherently interpretable models. Specifically we utilised a logistic regression model in order to maximise the recall and ensure the lowest quantity of false negatives therefore minimising failure to predict depression.

The primary limitation of this work is the data rather than the model. The data is subject to recall bias, social desirability bias, and straight-lining questions to finish them quickly. This lack of dataset reliability manifested in the models fit to that data. Specifically, the Logistic Regression fit to the dataset had relatively low predictive power on depression. Switching the approach to a neural network, which in theory can be made increasingly complex to match any dataset with near-perfect predictive performance, led to predictions with similar predictive metrics. This would imply that the lack of predictive performance of the models is moreseo a product of the dataset's relative lack of predictive power. That said, the Logistic Regression was able to produce predictions with some predictive performance and found the most significant predictors to likely be symptoms of a household's wealth. Based on this, the most significant changes to be made to improve rates of childhood depression likely have to do with wealth generation in the country.

2 Exploratory Data Analysis and Feature Engineering

With 13,162 observations and 87 features, the Events Per Variable ratio (EPV 151) is well above the recommended threshold of 10–20, suggesting the dataset has sufficient rows for the dimensionality of the problem. The initals steps of the EDA will be focused around removing features which have no significant effect on the target. Furthermore, after removing redundant and near-zero variance features, the effective dimensionality is expected to be considerably lower, improving EPV further. Our target variable is "FCF26". The initial parts of this section will describe the helper functions used within the main EDA pipeline before covering the pipeline and its methodolgy.

2.1 Helper Functions

We use Cramér's V to measure the association between each feature and the target, removing features with Cramér's $V < 0.05$, corresponding to less than 0.25% of variance explained individually. While individual features explain modest variance; reflecting the multifactorial and self-reported

nature of the data, the combined predictive power of retained features is expected to be substantially higher, as the model learns from many weak signals jointly. This allows us to quickly filter the dataset before more careful analysis is carried out.

[58]:
```
from scipy.stats import chi2_contingency
import numpy as np

def perform_cramers_v(
    *,
    df: pd.DataFrame,
    min_score: float = 0.05,
    target_column_name: str,
    ignore_columns: list,
) -> pd.DataFrame:
    """Performs cramers v test on the columns in the provided 'df'
    against the 'target_column_name' column and returns a pd.DataFrame
    containing the signficiant features that meet the 'min_score'
    threshold. Ignores the 'ignore_columns'.
    """
    results = []
    df_without_target = df.drop(columns=ignore_columns)
    for col in df_without_target.columns:
        contingency_table = pd.crosstab(df[col], df[target_column_name])
        chi2, p, _, _ = chi2_contingency(contingency_table)
        n = contingency_table.sum().sum()
        min_dim = min(contingency_table.shape) - 1
        cramers_v = np.sqrt(chi2 / (n * min_dim))
        results.append({"feature_name": col, "p_value": p, "cramers_v":␣
 ↪cramers_v})
    results_df = pd.DataFrame(results).sort_values("cramers_v", ascending=False)
    return results_df[results_df["cramers_v"] > min_score]
```

For visualising the results of this Cramer's V test that will be carried out we introduce a further function to plot the association of the figures with the target variable.

[59]:
```
def plot_cramers_v_bar(significant_features_df: pd.DataFrame) -> None:
    """Bar chart of each significant feature's Cramér's V association with the␣
 ↪target."""
    plt.figure(figsize=(6, 4))
    sns.barplot(data=significant_features_df, x="cramers_v", y="feature_name")
    plt.title("Cramér's V Association with Target")
    plt.xlabel("Cramér's V")
    plt.ylabel("Feature")
    plt.tight_layout()
    plt.show()
```

This straightforward function allows to manually remove the selected features from the dataframe. This is primarily used for features which are determined to have no clear use case in this report.

[60]:
```python
def remove_features_from_df(
    *,
    df: pd.DataFrame,
    features_to_remove: list[str],
) -> pd.DataFrame:
    """Removes the features in 'features_to_remove' from the provided 'df' and␣
 ↪returns the df.
    """
    return df[~df["feature_name"].isin(features_to_remove)] # removes features␣
 ↪not in features_to_remove.
```

The next function addresses feature transformations. We need to transform categories of features into numerical representations, either one-hot encodings, binary, or ordinal. To allow for more direct comparison of features.

[61]:
```python
class FeatureNames:
    """Stores feature names used throughout.
    Does not containt all 87 features in the dataset.
    """
    CL3 = "CL3"
    CL13 = "CL13"
    FCD2H = "FCD2H"
    FCD2H_ANSWERED = "FCD2H_ANSWERED"
    FCD2J = "FCD2J"
    FCD2J_ANSWERED = "FCD2J_ANSWERED"
    LS1 = "LS1"
    LS2 = "LS2"
    LS3 = "LS3"
    LS4 = "LS4"
    FCF26 = "FCF26"
    FCF26_BINARY = "FCF26_binary"
    ETHNICITY = "ethnicity"
    WSCORE = "wscore"
    HH7 = "HH7"
    HC4 = "HC4"
    HC4_AGGREGATED = "HC4_AGGREGATED"
    HC5 = "HC5"
    HC5_AGGREGATED = "HC5_AGGREGATED"
    HC8 = "HC8"
    HC12 = "HC12"
    MA2 = "MA2"
    MA2_KNOWN = "MA2_KNOWN"
    MSTATUS = "MSTATUS"
    VT20 = "VT20"
    VT22A = "VT22A"
    VT22B = "VT22B"
    VT22C = "VT22C"
```

```
    VT22D = "VT22D"
    WS11 = "WS11"
    WS1 = "WS1"
    WS1_AGGREGATED = "WS1_AGGREGATED"
    WS4 = "WS4"
    WS7 = "WS7"

"""
====================
 feature_config map
====================
feature_config contains a map of feature name mapped to a dictionary of rules
as to how the feature will be transformed. "map" is used to map non-numeric
values to numeric ones. "numeric": True means the feature's values will be
coerced into numbers; i.e "30" will become 30.
"""
feature_config = {
    # LS1: To Mother: mother's happiness level, 4 categories
    FeatureNames.LS1: {
        "map": {
            "VERY UNHAPPY": 0,
            "SOMEWHAT UNHAPPY": 1,
            "NEITHER HAPPY NOR UNHAPPY": 2,
            "SOMEWHAT HAPPY": 3,
            "VERY HAPPY": 4,
            "NO RESPONSE": -1,
        },
    },
    # LS2 To Mother: mother's happiness level 0-10
    FeatureNames.LS2: {
        "map": {
            "NO RESPONSE": None,
        },
        "numeric": True,
    },
    # LS3: To Mother: Compared to this time last year, would you say that your␣
 ↪life has improved, stayed more or less the same, or worsened, overall?
    FeatureNames.LS3: {
        "map": {
            "WORSENED": 0,
            "MORE OR LESS THE SAME": 1,
            "IMPROVED": 2,
            "NO RESPONSE": -1,
        },
    },
    # LS4: To Mother: And in one year from now, do you expect that your life␣
 ↪will be better, will be more or less the same, or will be worse, overall?
```

```
    FeatureNames.LS4: {
        "map": {
            "WORSE": 0,
            "MORE OR LESS THE SAME": 1,
            "BETTER": 2,
            "NO RESPONSE": -1,
        },
    },
    # WS4: How long does it take for members of your household to go there, get␣
↪water, and come back?
    # "NO RESPONSE" implies they do not collect water and it is readily␣
↪available, as WS4 question is only asked for certain water types involving␣
↪travel.
    FeatureNames.WS4: {
        "map": {"DK": None, "MEMBERS DO NOT COLLECT": 0, "NO RESPONSE": 0, np.
↪nan: 0}, "numeric": True,
    },
    # WS7: In the last month, has there been any time when your household did␣
↪not have sufficient quantities of drinking water?
    FeatureNames.WS7: {
        "map": {
            "NO, ALWAYS SUFFICIENT": 0,
            "YES, AT LEAST ONCE": 1,
            "DK": -1,
            "NO RESPONSE": -1,
        },
    },
    # VT20: To Mother: How safe do you feel walking alone in your neighbourhood␣
↪after dark?
    FeatureNames.VT20: {
        "map": {
            "VERY UNSAFE": 0,
            "UNSAFE": 1,
            "NEVER WALK ALONE AFTER DARK": 2,
            "SAFE": 3,
            "VERY SAFE": 4,
            "NO RESPONSE": -1,
        },
    },
    # VT22: In the past 12 months, have you personally felt discriminated␣
↪against or harassed on the basis of the following grounds?
    # VT22A: A) Ethnic or immigration origin?
    FeatureNames.VT22A: {"map": {"NO": 0, "YES": 1, "DK": -1, "NO RESPONSE":␣
↪-1}},
    # VT22B: B) Sex?
```

```
    FeatureNames.VT22B: {"map": {"NO": 0, "YES": 1, "DK": -1, "NO RESPONSE":␣
↪-1}},
    # VT22C: C) Sexual orientation?
    FeatureNames.VT22C: {"map": {"NO": 0, "YES": 1, "DK": -1, "NO RESPONSE":␣
↪-1}},
    # VT22D: D) Age?
    FeatureNames.VT22D: {"map": {"NO": 0, "YES": 1, "DK": -1, "NO RESPONSE":␣
↪-1}},
    # MA2: How old is your (husband/partner)?
    # -1 represents information not known, we'll later create a binary feature␣
↪indicating whether husband/partner's age is known in the dataset or not.
    FeatureNames.MA2: {"map": {"DK": -1, "NO RESPONSE": -1, np.nan: -1},␣
↪"numeric": True},
    # CL3: Since last (day of the week) about how many hours did (name) engage␣
↪in (this activity/these activities), in total?
    # Map np.nan values to 0 because if all answers to question CL2 are no,b␣
↪then question CL3 is skipped, implying 0 child labour hours worked.
    FeatureNames.CL3: {"map": {np.nan: 0}, "numeric": True},
    # CL13: Since last (day of the week), about how many hours did (name)␣
↪engage in (this activity/these activities), in total?
    # Again map np.nan values to 0. If all answers to question CL2 areno then␣
↪question CL13 is skipped. There are 2 "NO RESPONSE" values. We map them to 0.
↪
    # Alternatively we could drop the two rows corresponding to "NO RESPONSE".
    FeatureNames.CL13: {
        "map": {
            "NO RESPONSE": 0,
            np.nan: 0
        },
        "numeric": True
    },
    # HC12: Does any member of your household have a mobile telephone?
    FeatureNames.HC12: {"map": {"NO": 0, "YES": 1, "NO RESPONSE": -1}},
    # MSTATUS: marital status of mother
    FeatureNames.MSTATUS: {
        "map": {
            "Never married/in union": 0,
            "Formerly married/in union": 1,
            "Currently married/in union": 2,
        },
    },
    #FCD: Question to child: Adults use certain ways to teach children the␣
↪right behaviour or to address a behaviour problem. I will read various
    # methods that are used. Please tell me if you or any other adult in your␣
↪household has used this method with (name) in the past month.
    # FCD2H: Called (him/her) dumb, lazy or another name like that.
```

```
    # We set "NO RESPONSE": -1, later we will introduce a new binary feature␣
↪that indentifies whether the question was answered or not.
    FeatureNames.FCD2H: {"map": {"NO": 0, "YES": 1, "NO RESPONSE": -1, np.nan:␣
↪-1}},
    # FCD2J: Hit or slapped (him/her) on the hand, arm, or leg.
    # Ditto: We set "NO RESPONSE": -1, and later introduce binary feature␣
↪identifying whether question was answered.
    FeatureNames.FCD2J: {"map": {"NO": 0, "YES": 1, "NO RESPONSE": -1, np.nan:␣
↪-1}},
    # HC8: Does your household have electricity?
    FeatureNames.HC8: {
        "map": {
            "NO": 0,
            "YES, OFF-GRID (GENERATOR/ISOLATED SYSTEM)": 1,
            "YES, INTERCONNECTED GRID": 2,
            "NO RESPONSE": -1,
        },
    },

}
```

This final helper function is to configue our features according to a dictionary. This config dictionary is keyed by feature name. Values in the dictionary is a map of rules to apply transformations to fields. "map" specifies how to map values. "numeric": True specifies to coerce the values into numeric values.

[62]:
```
def apply_feature_config(*, df: pd.DataFrame, config: dict[str, dict]) -> pd.
↪DataFrame:
    df = df.copy()
    for feature_name, rules in config.items():
        if feature_name not in df.columns:
            continue
        if value_map := rules.get("map"):
            # if there's a value map for the feature, then apply it.
            df[feature_name] = df[feature_name].map(value_map).
↪fillna(df[feature_name])
        if rules.get("numeric"):
            # if numeric field is non-empty then coerce values to numeric.
            df[feature_name] = pd.to_numeric(df[feature_name], errors="coerce")
    return df
```

2.2 Full Pipeline

Finally we can put all these helper functions together to form the full EDA pipeline. The first step is to load the data and then map the target variable to binary values. The target variable tracks whether a child is depressed or not as an ordinal variable with six imbalanced cateogries. Our mapping follow the structure of 0 being not depressed and 1 being depressed. We mapped the different catogeries as follows:

"NEVER": 0; if a child is never depressed this is natural assignment

"A FEW TIMES A YEAR": 0; we have reasoned that a few times a year could simply a one-off events, such as a funeral or missing out on an event they were looking forward to, rather than a pattern of depression

"MONTHLY: 0; this is irregular enough that it could be attributed to an event such as failing a class test. Once again too infrequent to determine a pattern.

"WEEKLY: 1; a recurring weekly is consistent enough to be a strong indicator of depression

"DAILY": 1; a daily frequency indicates a clear pattern of depression

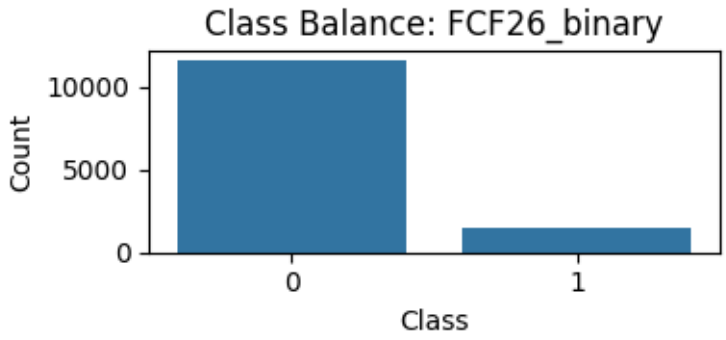
"NO RESPONSE"; these responses are dropped

This mapping is then applied to the main dataframe, creating the new target variable "FCF26_binary".

[63]:
```
  # Read unicef data into a Pandas DataFrame.
unicef_df = pd.read_csv(UNICEF_DATA_CSV)

# Drop rows corresponding to nan values in "FCF26". We do this because we
#    will use 'FCF26' to derrive the target variable.
print(f"Dropping {unicef_df[FeatureNames.FCF26].isnull().mean() * 100:2f} % of␣
 ↪rows corresponding to nan values in '{FeatureNames.FCF26}'.")
unicef_df = unicef_df.dropna(subset=[FeatureNames.FCF26])

# We introduce a depression_map that maps categories in the "FCF26" feature to
# binary values; where 0 = not depressed, and 1 = depressed.
depression_map = {
    "NEVER": 0,
    "A FEW TIMES A YEAR": 0,
    "MONTHLY": 0,
    "WEEKLY": 1,
    "DAILY": 1,
}
# Remove rows corresponding to "NO RESPONSE" on "FCF26", the feature we will␣
 ↪use to derive the target. This Removes 0.78% of total rows.
unicef_df = unicef_df[unicef_df[FeatureNames.FCF26] != "NO RESPONSE"]

# Map "FCF26" to binary feature of 0 = Not depressed, and 1 = Depressed
#    based on the depression_map.
unicef_df[FeatureNames.FCF26_BINARY] = unicef_df[FeatureNames.FCF26].
 ↪map(depression_map)
print(f"Target: {FeatureNames.FCF26_BINARY} has {unicef_df[FeatureNames.
 ↪FCF26_BINARY].isnull().mean() * 100:2f} % nan values")
```

```
Dropping 0.782556 % of rows corresponding to nan values in 'FCF26'.
Target: FCF26_binary has 0.000000 % nan values
```

[64]:
```
class_counts = unicef_df[FeatureNames.FCF26_BINARY].value_counts()
plt.figure(figsize=(4, 2))
sns.barplot(x=class_counts.index, y=class_counts.values)
plt.title(f"Class Balance: {FeatureNames.FCF26_BINARY}")
plt.xlabel("Class")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```

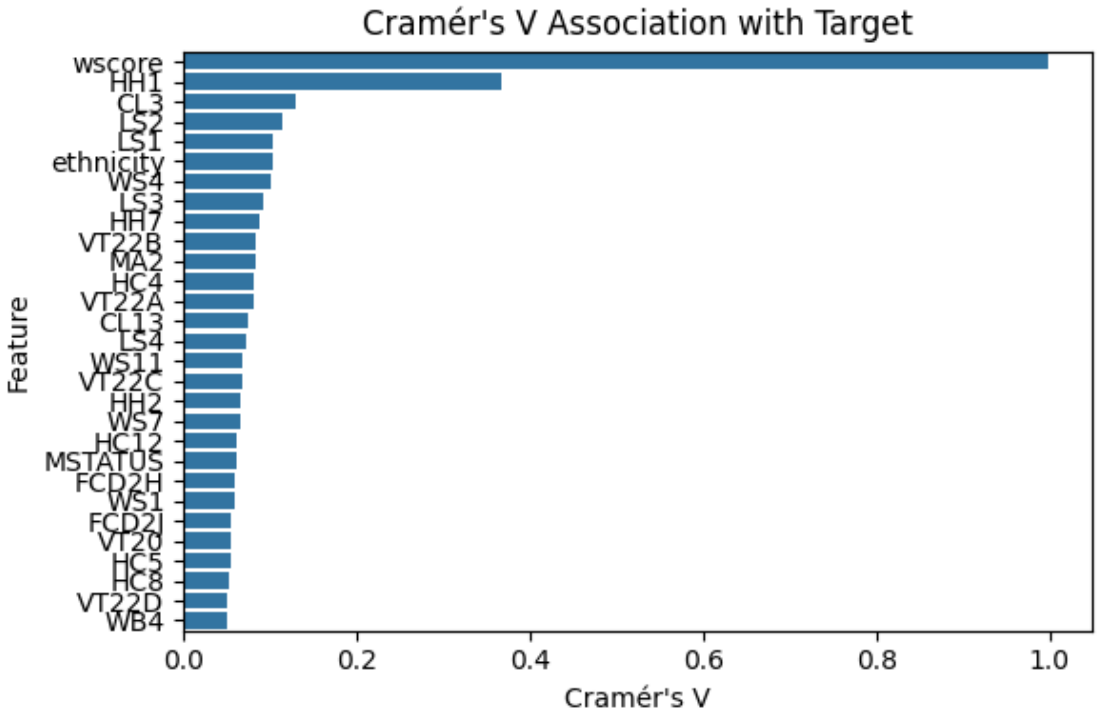


After transformation to a binary variable, we can see that the class distribution is severely imbalanced: 11,565 (88.7%) not depressed and 1,471 (11.3%) depressed. We adjust class weights in our model to address the imbalance, giving False values a weight of 1 and True values a weight of 7.86. We do this rather than duplicating data or applying Synthetic Minority Oversampling Technique (SMOTE). We consider SMOTE inappropriate here; our data represents real individuals, and introducing artificial respondents into the dataset risks distorting the learned relationships between features and depression outcomes.

As described previously we use Cramer's V test to filter for features which have a sufficient level of correlation with the target variable "FCF26_binary". This filtering is not enough to infer that the remaining variables have a causal link with the target however, with supporting evidence it serves as a good starting point.

[65]:
```
significant_features_df = perform_cramers_v(
    df=unicef_df,
    # consider p > 0.05 a significant feature. This is tunable to be more/less
    # aggressive (higher being stricter).
    min_score=0.05,
    target_column_name=FeatureNames.FCF26_BINARY,
    ignore_columns=[
        FeatureNames.FCF26_BINARY, # FCF26_binary is target so remove␣
↪FCF26_binary.
        FeatureNames.FCF26,  # target is derived from FCF26 so remove FCF26.
    ]
)
```

```
plot_cramers_v_bar(significant_features_df)
```



This visualisation clearly shows that the remainig features display similar levels of association with the target variable, with a notable outlier in "wscore" which address in the next section. "HH1", as the other outlier, is the household cluster within which a family resides which serves as a fairly good indicator of wealth. This is a sign that wealth may be directly linked to depression levels in children.

We remove w-score from our feature set as it represents a combined measure of wealth; arguably an alternative target variable rather than a predictor, and its inclusion would risk circular reasoning. Furthermore, although it could be an alternative target variable, it is unclear how it is actually obtained. It is preferable to avoid introudcing inexplicable elements into this report. Other wealth indicators are factors which are stil likely to be considered in the final model. We additionally remove ethnicity to prevent the introduction of racial bias into a model that will inform resource allocation decisions. While ethnicity shows a weak but non-negligible association with depression (Cramér's V = 0.1, accounting for ~1% of variance), this association likely reflects proxy effects of socioeconomic status, cultural differences in self-reporting, or experiences of discrimination; factors better captured by other features in the dataset. We note that any future reintroduction of ethnicity would require significant ethical justification. This leaves us with 27 features at the end of the feature selection stage.

[66]:
```
significant_features_df = remove_features_from_df(
    df=significant_features_df,
```

```
    features_to_remove = [
        FeatureNames.ETHNICITY,
        FeatureNames.WSCORE,
    ]
)
significant_features: list[str] = significant_features_df["feature_name"].
 ↪to_list()
```

We now apply these previous steps of filtering to a copy of the full dataframe to preserve the ingtegrity of the data nad ensure better reproduciblity. Following this we begin to merge and similar categorical features. In the following block we introduce aggregate variables for floor categories, roof categories and drinking water sources. A numercal orering is introdued within these aggreate features to map the lower level ones. They are ordered in terms of how much modern development they require, which should allow these aggregate features to serve as good wealth indicators. This reduces the number of features further whilst still being able to extract meaningful analysis.

The final step within this EDA pipeline is to transform a final set of features into a numerical format that our model can understand. For the nominal features we apply one-hot encoding and drop the first column in the encoding to remove redundant information. For the categorical features where a natural ordering makes sense, we transform them using a map, mapping the category to a numeric value our model will understand.

[67]:
```
# Filter the unicef dataframe, including only significant features and the␣
 ↪target "FCF26_binary".
unicef_df_filtered = unicef_df[
    significant_features + [FeatureNames.FCF26_BINARY]
].copy()

# floor_categories maps floor categories to higher level categories.
# Ordered ascending from least developed to most developed.
floor_categories = {
    # NATURAL FLOOR: 0
    "EARTH / SAND": 0,
    "DUNG": 0,
    # RUDIMENTARY FLOOR: 1
    "WOOD PLANKS": 1,
    "PALM / BAMBOO": 1,
    # FINISHED FLOOR: 2
    "PARQUET OR POLISHED WOOD": 2,
    "VINYL OR ASPHALT STRIPS": 2,
    "CERAMIC TILES": 2,
    "CEMENT": 2,
    "CARPET": 2,

}
unicef_df_filtered[FeatureNames.HC4_AGGREGATED] =␣
 ↪unicef_df_filtered[FeatureNames.HC4].map(floor_categories)
```

```
roof_categories = {
    # NO ROOF
    "NO ROOF": 0,
    # NATURAL ROOFING
    "THATCH / PALM LEAF": 1,
    "SOD": 1,
    # RUDIMENTARY ROOFING
    "RUSTIC MAT": 2,
    "PALM / BAMBOO": 2,
    "WOOD PLANKS": 2,
    "CARDBOARD": 2,
    # FINISHED ROOFING
    "IRON SHEETS / METAL / TIN": 3,
    "WOOD": 3,
    "CALAMINE / CEMENT FIBRE": 3,
    "CERAMIC TILES": 3,
    "CEMENT": 3,
    "ROOFING SHINGLES": 3,
}
unicef_df_filtered[FeatureNames.HC5_AGGREGATED] =␣
 ↪unicef_df_filtered[FeatureNames.HC5].map(roof_categories)

drinking_water_types = {
    # UN-PIPED WATER
    "TUBE WELL / BOREHOLE": 0,
    "DUG WELL: PROTECTED WELL": 0,
    "DUG WELL: UNPROTECTED WELL": 0,
    "SPRING: PROTECTED SPRING": 0,
    "SPRING: UNPROTECTED SPRING": 0,
    "RAINWATER": 0,
    "TANKER-TRUCK": 0,
    "CART WITH SMALL TANK ": 0,
    "WATER KIOSK": 0,
    "SURFACE WATER (RIVER, DAM, LAKE, POND, STREAM, CANAL, IRRIGATION CHANNEL)":
 ↪ 0,


    # PACKAGED WATER
    "PACKAGED WATER: BOTTLED WATER": 1,
    "PACKAGED WATER: SACHET WATER": 1,

    # PIPED WATER
    "PIPED WATER: PIPED INTO DWELLING": 2,
    "PIPED WATER: PIPED TO YARD / PLOT": 2,
    "PIPED WATER: PIPED TO NEIGHBOUR": 2,
    "PIPED WATER: PUBLIC TAP / STANDPIPE": 2,
}
```

```
unicef_df_filtered[FeatureNames.WS1_AGGREGATED] =␣
 ↪unicef_df_filtered[FeatureNames.WS1].map(drinking_water_types)#

nominal_features = [
    # HC4: Main material of the dwelling floor.
    FeatureNames.HC4,
    # HC5: Main material of the roof.
    FeatureNames.HC5,
    # HH7: Region
    FeatureNames.HH7,
    # WS1: What is the main source of drinking water used by members of your
    # household?
    FeatureNames.WS1,
    # WS11: What kind of toilet facility do members of your household usually
    # use?
    FeatureNames.WS11,
]
unicef_df_filtered = pd.get_dummies(
    unicef_df_filtered,
    columns=nominal_features,
    # We set drop_first=True to reduce redundant information, a matrix of all
    # zeroes implies the first category.
    drop_first=True,
)

# Categorical features
unicef_df_filtered = apply_feature_config(df=unicef_df_filtered,␣
 ↪config=feature_config)
```

Null handling and odd data quirks are the final transformations that need to be addressed. Regarding childhood discpline methods we introduce an additional binary feature to track whether the questions were actually answered. 1 if yes, 0 if no. This can provide some meaningful insight as the lack of an answer can indicate withholding information due to factors such as shame. "MSTATUS" is a categorical variable which contains a single numerical response as it does not make sense in the context of the question. Finally we identify null values that are remaing, which at this point have been reduced to less than 2.63%. Any remaing null values are filled with the mode as this is sufficient with such a small proporiton.

[68]:
```
# 10) Introduce additional features: Whether Violence Questions were answered
#     or not. 1 = Yes, 0 = No.
unicef_df_filtered[FeatureNames.FCD2H_ANSWERED] =␣
 ↪(unicef_df_filtered[FeatureNames.FCD2H]  != -1).astype(int)
unicef_df_filtered[FeatureNames.FCD2J_ANSWERED] =␣
 ↪(unicef_df_filtered[FeatureNames.FCD2J]  != -1).astype(int)

# And whether husband/partner age is known or not in the dataset.
```

```
unicef_df_filtered[FeatureNames.MA2_KNOWN] = (unicef_df_filtered[FeatureNames.
 ↪MA2]  != -1).astype(int)

# Drop the 1 row where MSTATUS is "9.0"; because the value "9.0" does not make␣
 ↪sense in the context of non-numerical answer categories.
unicef_df_filtered = unicef_df_filtered[unicef_df_filtered["MSTATUS"] != "9.0"]

null_summary = pd.DataFrame({
    "count": unicef_df_filtered.isnull().sum(),
    "% null": unicef_df_filtered.isnull().mean() * 100
})

# Identify null values: After the feature engineering steps, we have some null␣
 ↪values, all less than 2.63%.
print(null_summary[null_summary["count"] > 0].round(2))

# Fill null values with mode: Proportion of null values is small enough to fill␣
 ↪in with mode.
for col in unicef_df_filtered.columns:
    unicef_df_filtered[col] = unicef_df_filtered[col].
 ↪fillna(unicef_df_filtered[col].mode()[0])
```

```
                count  % null
LS2               343    2.63
LS1               279    2.14
WS4               189    1.45
LS3               279    2.14
VT22B             279    2.14
VT22A             279    2.14
LS4               279    2.14
VT22C             279    2.14
VT20              279    2.14
VT22D             279    2.14
WB4               279    2.14
HC4_AGGREGATED      4    0.03
HC5_AGGREGATED      4    0.03
WS1_AGGREGATED      7    0.05
```

Following the feature selection, including using Cramér's V and manual removal, we reduced the dataset to 27 relevant high-level variables based on their statistical association with the target. Following the further preprocessing stage the categorical variables were expanded and additional engineered features were introduced. This resulted in a final model input space of 78 features. The apparent increase in dimensionality reflects feature representation expansion rather than the introduction of new information. We can now split the data into training and testing sets as below, enabling the next step of model selection and tuning.

[69]:
```
X = unicef_df_filtered.drop(columns=[FeatureNames.FCF26_BINARY])
y = unicef_df_filtered[FeatureNames.FCF26_BINARY]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

3 Model Fitting and Tuning

For the purposes of this classification, we considered classification trees, bagging, random forests, support vector machines, and logisitic regression. The support vector machine approach failed to fit in a reasonable amount of runtime, as after over 45 minutes the pipeline still failed to produce results. For this reason, the support vector machine approach was removed from consideration for being impractical for the application and audience at hand. The primary metrics by which to assess a predictive model are specificity, recall, accuracy, and F1 Score. Of these metrics, the most pertinent is recall, or the rate at which the model correctly identifies children exhibiting depression as being depressed. In machine learning terms, this is the number of true positives (children with depression predicted to be depressed) divided by the sum of true positives and false negatives (children with depression predicted not to be depressed). We choose this metric as most important because failure to identify depression in children with depression, i.e. false negatives, lead to the most significant negative public health outcomes. That said, a model can achieve perfect recall by predicting positives for all data points, or in this case predicting every child is depressed. Therefore, while recall is the primary metric by which to assess model suitability, other performance metrics are also considered.

Model	Recall	Specificity	F1 Score	Accuracy
Decision Tree	0.91	0.13	0.34	0.29
Bagging	0.05	0.97	0.08	0.79
Logistic Regression	0.58	0.60	0.37	0.59
Random Forest	0.55	0.65	0.37	0.63

In this case, the approach with the best recall is the Classification Tree. However, the classification tree performs quite poorly across other performance metrics, and indeed achieved this high recall by dramatically overpredicting depression. The approach that achieves the highest recall balanced by reasonable performance across other metrics is Logistic Regression. This follows from the fact that logistic regression can be tuned specifically to perform better according to specific performance metrics, and thus could be tuned specifically to maximise recall. Logistic regression was therefore chosen as the approach from which to further develop a predictive model.

For the sake of model interpretation, we first define a function that will return the feature coefficients of the logistic regression and plot them. This allows for interpretation of which features are significant and which have little predictive power by looking at the magnitude of the associated coefficients. Features with high impact on the prediction will have coefficients significantly less than or greater than zero, while features with low impact on the prediction will have coefficients near zero.

[70]:
```
def get_coefs(m, plot = False, feature_names = None, figsize = (5,5), figtitle␣
↪= None, intercept = True, top_k = None, min_abs_coeff = None, sort_by_abs =␣
↪True):
    """Returns model coefficients in a data frame for a fitted linear model.

    Args:
        m: sklearn linear model object or pipeline with linear model as final␣
↪step
        plot: boolean value, should coefficients be plotted with error bars
        feature_names: list of feature names to use in the plot
        figsize: tuple defining figure size
        figtitle: string defining figure title
        intercept: boolean value, should intercept be included in the plot
    """
    # Extract intercept and coefficients into a single array
    w0 = m[-1].intercept_ if isinstance(m, sklearn.pipeline.Pipeline) else m.
↪intercept_
    w1 = m[-1].coef_ if isinstance(m, sklearn.pipeline.Pipeline) else m.coef_
    w = np.concatenate((w0.reshape(-1), w1.reshape(-1)))
    # Extract name of features
    if feature_names is None:
        feature_names = m[:-1].get_feature_names_out() if isinstance(m, sklearn.
↪pipeline.Pipeline) else m.feature_names_in_
    feature_names = np.concatenate((['intercept'], feature_names))
    # Create a data frame
    w_df = pd.DataFrame({'feature': feature_names, 'coef': w}).sort_values␣
↪("coef", ascending=False)
        # get absolute value of coeffs
    w_df["abs_coef"] = w_df["coef"].abs()

    if sort_by_abs:
        w_df = w_df.sort_values("abs_coef", ascending=False)

    if min_abs_coeff is not None:
        w_df = w_df[w_df["abs_coef"] >= min_abs_coeff]

    if top_k is not None:
        w_df = w_df.head(top_k)

    if plot:
        if not intercept:
            w_df = w_df[w_df['feature'] != 'intercept']
        plt.figure(figsize=(4,3))
        plt.barh(w_df['feature'], w_df['coef'])
        plt.ylabel('Features')
        plt.xlabel('Coefficient Value')
        plt.axvline(x=0, color=".5")
```

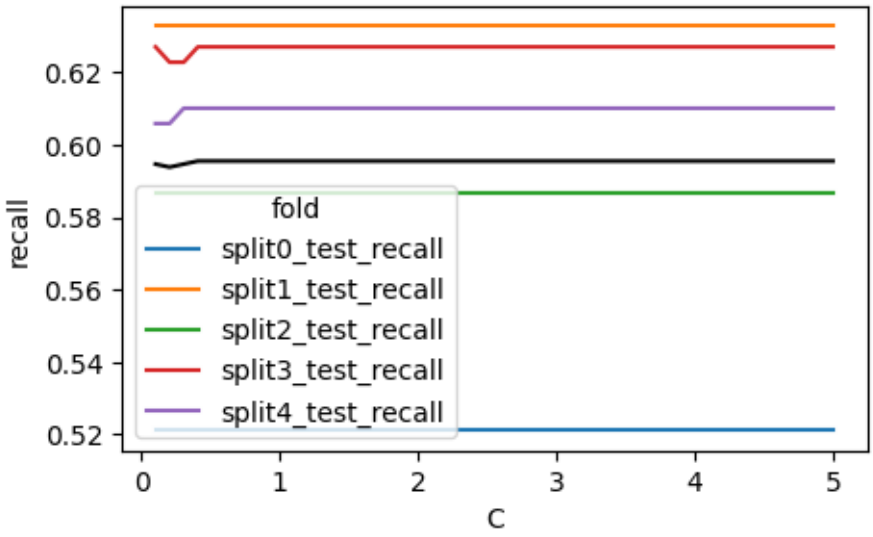
```
        if figtitle is not None:
            plt.title(figtitle)
        plt.grid()
        plt.show()

    return  w_df.drop(columns="abs_coef")
```

We now start by creating a logistic regression pipeline. Features are first standardized to have a mean of zero and variance of one, enabling the regression to assign coefficients/importance to features equivalently. Without this step, features with high mean or variance are punished by the regression and considered relatively unimportant because small changes in coefficient values associated with these features yield greater change in predictions compared to features with smaller mean or variance. Because of the imbalance in the data, we then assign entries with no reported depression a weight of 1 and entries with reported depression a weight of 7.86. This weighting is selected as there are approximately 7.86 times as many entries with no associated depression as there are with associated depression, therefore this weighting causes the regression to predict not based on the overall frequency of depression across the dataset, but instead on the case-by-case, observational basis desired. Because of the rigor of our exploratory data analysis, we use ridge regularization, or a penalty parameter of l2, to create a regression with coefficients balanced across all features. This tends coefficients toward zero without forcing them equal zero, giving predictions based on all features retained from the exploratory data analysis with the scale of coefficients still representing feature importance. The random seed is then set at 42 for reproducibility of results, and the solver and max iterations are set to balance quick convergence to a model with the quality obtained by that model.

Having chosen to obtain predictive coefficients by ridge regression, we must now tune the strength of the penalty parameter. We will do this by splitting the data into 5 stratified folds, or five sets with equal balance between positives (children exhibiting depression) and negatives (children exhibiting no depression). We then test different strengths of the penalty parameter by fitting the pipeline with across a set of strengths, fitting according to the best recall achieved at that strength, and plotting the recall achieved at all five sets and the average recall across all sets to determine which penalty parameter best balances predictive performance with model complexity.

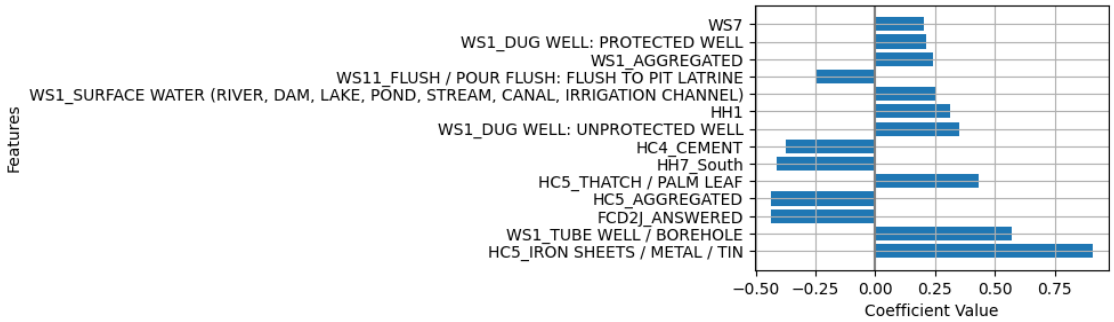
[71]:
```
# Pipeline
log_pipe_l2 = make_pipeline(
    StandardScaler(),
    LogisticRegression(class_weight={0:1, 1:7.86}, random_state=42,␣
 ↪penalty='l2', solver='liblinear',
        max_iter=1000)
)
# Possible C values:
C_list = np.linspace(0,5, num=50)
# Grid search CV:
log_rs = GridSearchCV(log_pipe_l2,
                      param_grid={'logisticregression__C': C_list},
                      scoring = ["accuracy", "f1","recall","precision"],␣
 ↪#Evaluation metrics to compute on validation sets
```

```
                      cv = StratifiedKFold(n_splits=5, shuffle=True,␣
 ↪random_state=42),
                      refit = "recall", # Refits the best model on the entire␣
 ↪dataset using the recall metric
                      return_train_score = True)

# Tune the model with grid search:
log_rs.fit(X_train, y_train)
# Extract only mean and split scores
cv_accuracy = pd.DataFrame(
    data = log_rs.cv_results_
).filter(
    # Extract the split#_test_recall and mean_test_recall columns
    regex = '(split[0-4]+|mean)_test_recall'
).assign(
    # Add the alphas as a column
    C = C_list
)
# Reshape the data frame for plotting
d = cv_accuracy.melt(
    id_vars=('C','mean_test_recall'),
    var_name='fold',
    value_name='recall'
)
# Plot the validation scores across folds
plt.figure(figsize=(5,3))
sns.lineplot(x='C', y='recall', color='black', errorbar=None, data = d)  # Plot␣
 ↪the mean score in black.
sns.lineplot(x='C', y='recall', hue='fold', data = d) # Plot the curves for␣
 ↪each fold in different colors
plt.show()
```



Based on the plot generated by this cross-validation search, we can determine that a penalty parameter of one is sufficient for the training set. We can then fit our pipeline with this parameter and evaluate its final performance on the test set and first get the features deemed most important by the regression.

[72]:
```
log_pipe_l2 = make_pipeline(
    StandardScaler(),
    LogisticRegression(class_weight={0:1, 1:7.86}, random_state=42,␣
 ↪penalty='l2', solver='liblinear',
        max_iter=1000, C=1)
)
log_pipe_l2.fit(X_train, y_train)
get_coefs(log_pipe_l2, plot=True, min_abs_coeff=0.2)
```



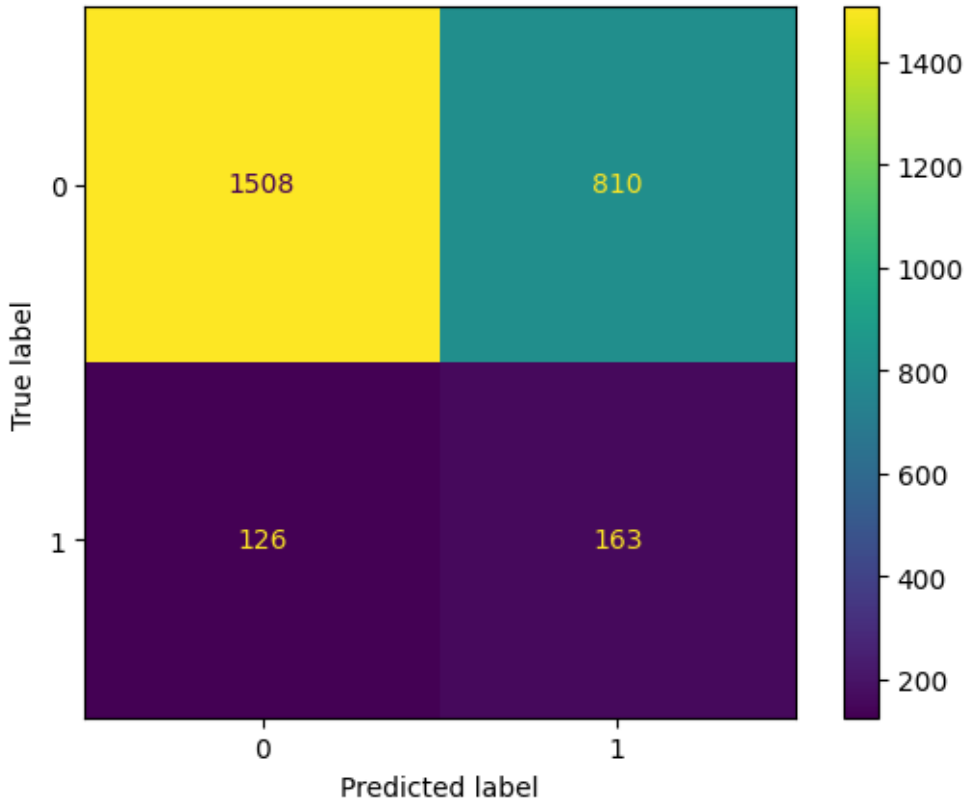
[72]:

	feature	coef
37	HC5_IRON SHEETS / METAL / TIN	0.906977

```
61                              WS1_TUBE WELL / BOREHOLE  0.571613
76                                       FCD2J_ANSWERED -0.436873
24                                       HC5_AGGREGATED -0.435547
43                                HC5_THATCH / PALM LEAF  0.432425
47                                             HH7_South -0.412077
26                                            HC4_CEMENT -0.375834
49                          WS1_DUG WELL: UNPROTECTED WELL  0.350101
1                                                     HH1  0.314517
59  WS1_SURFACE WATER (RIVER, DAM, LAKE, POND, STR…  0.253231
66      WS11_FLUSH / POUR FLUSH: FLUSH TO PIT LATRINE -0.246658
25                                       WS1_AGGREGATED  0.239772
48                           WS1_DUG WELL: PROTECTED WELL  0.212142
14                                                    WS7  0.204555
```

We can then assess the predictive performance of the model by producing a Confusion Matrix, showing the predictions the model makes versus the actual observed values, where a label of 1 indicates depression and a label of 0 indicates no depression. We also get all performance metrics, namely Accuracy, Precision, Recall, Specificity, and FScore.

[73]:
```
y_pred = log_pipe_l2.predict(X_test)
ConfusionMatrixDisplay.from_estimator(log_pipe_l2, X_test, y_test)
plt.show()
print(f"Accuracy: {metrics.accuracy_score(y_test, y_pred)}")
print(f"Precision: {metrics.precision_score(y_test, y_pred)}")
print(f"Sensitivity/Recall: {metrics.recall_score(y_test, y_pred)}")
print(f"Specificity: {metrics.recall_score(y_test, y_pred, pos_label=0.0)}")
print(f"F Score: {metrics.f1_score(y_test, y_pred)}")
```

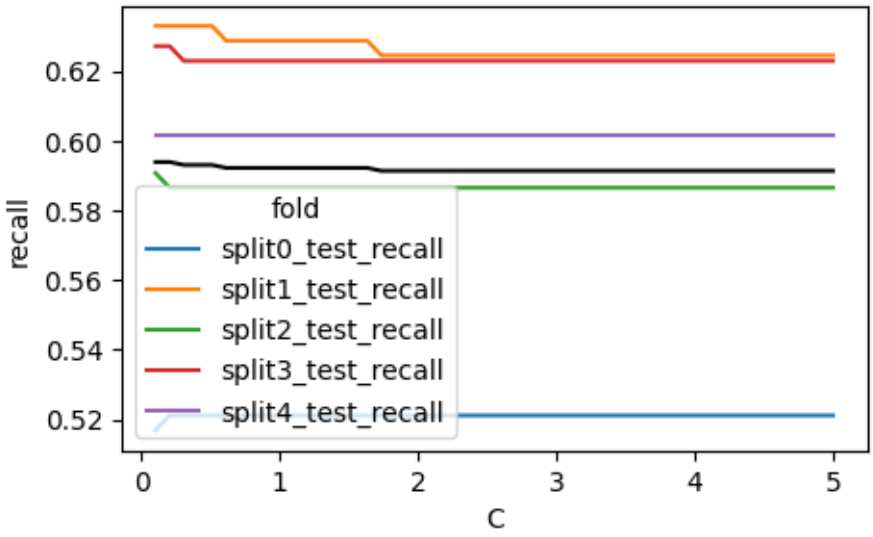



```
Accuracy: 0.6409666283084005
Precision: 0.16752312435765673
Sensitivity/Recall: 0.5640138408304498
Specificity: 0.6505608283002589
F Score: 0.2583201267828843
```

We can also consider balancing the data by assigning class weights automatically through the regression. This diminishes the interpretability of the class weights, but may increase predicitive performance. We do this by repeating the process as before, but instead assigning class weights by the built-in "balanced" method. We again start by cross-validating the penalty strength parameter.

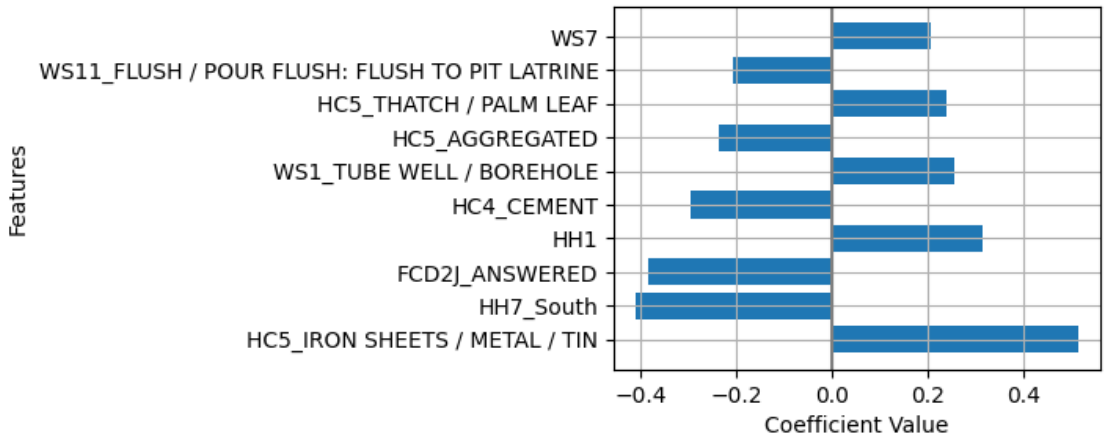
[74]:
```
log_pipe_bal = make_pipeline(
    StandardScaler(),
    LogisticRegression(class_weight="balanced", random_state=42, penalty='l2',␣
 ↪solver='liblinear',
        max_iter=1000)
)
# Grid search CV:
log_rs = GridSearchCV(log_pipe_bal,
                      param_grid={'logisticregression__C': C_list},
                      scoring = ["accuracy", "f1","recall","precision"],␣
 ↪#Evaluation metrics to compute on validation sets
```

```
                      cv = StratifiedKFold(n_splits=5, shuffle=True,␣
 ↪random_state=42),
                      refit = "recall", # Refits the best model on the entire␣
 ↪dataset using the accuracy metric
                      return_train_score = True)

# Tune the model with grid search:
log_rs.fit(X_train, y_train)
# Extract only mean and split scores
cv_accuracy = pd.DataFrame(
    data = log_rs.cv_results_
).filter(
    # Extract the split#_test_f1 and mean_test_f1 columns
    regex = '(split[0-4]+|mean)_test_recall'
).assign(
    # Add the alphas as a column
    C = C_list
)
# Reshape the data frame for plotting
d = cv_accuracy.melt(
    id_vars=('C','mean_test_recall'),
    var_name='fold',
    value_name='recall'
)
# Plot the validation scores across folds
plt.figure(figsize=(5,3))
sns.lineplot(x='C', y='recall', color='black', errorbar=None, data = d)  # Plot␣
 ↪the mean score in black.
sns.lineplot(x='C', y='recall', hue='fold', data = d) # Plot the curves for␣
 ↪each fold in different colors
plt.show()
```



This time, cross-validation indicates a suitable penalty strength of 0.5. We then use this parameter to fit the model, which produces the following significant features and associated coefficients.

[75]:
```
log_pipe_bal = make_pipeline(
    StandardScaler(),
    LogisticRegression(class_weight="balanced", random_state=42, penalty='l2',␣
 ↪solver='liblinear',
        max_iter=1000, C=.5)
)

log_pipe_bal.fit(X_train, y_train)
get_coefs(log_pipe_bal, plot=True, min_abs_coeff=0.2)
```

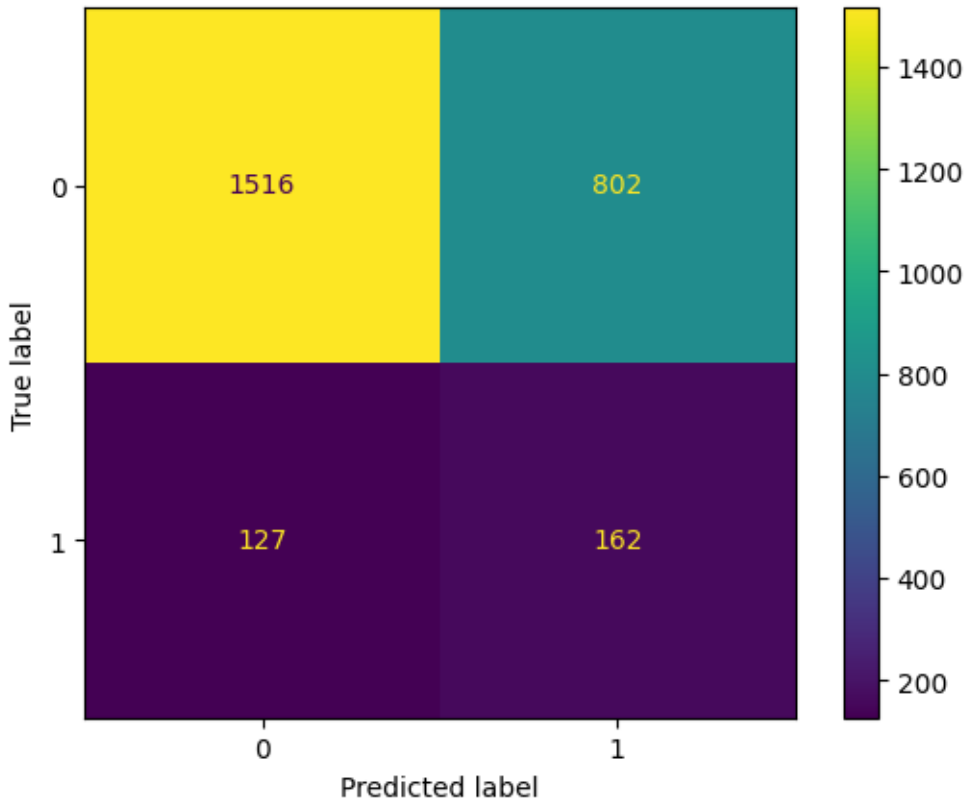


[75]:

	feature	coef
37	HC5_IRON SHEETS / METAL / TIN	0.513258
47	HH7_South	-0.409703
76	FCD2J_ANSWERED	-0.385476
1	HH1	0.311686
26	HC4_CEMENT	-0.295043
61	WS1_TUBE WELL / BOREHOLE	0.255502
24	HC5_AGGREGATED	-0.237985
43	HC5_THATCH / PALM LEAF	0.236871
66	WS11_FLUSH / POUR FLUSH: FLUSH TO PIT LATRINE	-0.207076
14	WS7	0.204365

We can then again assess the performance of the regression by the following Confusion Matrix and associated Accuracy, Precision, Recall, Specificity, and F Score.

[76]:
```
y_pred = log_pipe_bal.predict(X_test)
ConfusionMatrixDisplay.from_estimator(log_pipe_bal, X_test, y_test)
plt.show()

print(f"Accuracy: {metrics.accuracy_score(y_test, y_pred)}")
print(f"Precision: {metrics.precision_score(y_test, y_pred)}")
print(f"Sensitivity/Recall: {metrics.recall_score(y_test, y_pred)}")
print(f"Specificity: {metrics.recall_score(y_test, y_pred, pos_label=0.0)}")
print(f"F Score: {metrics.f1_score(y_test, y_pred)}")
```



```
Accuracy: 0.6436517069428462
Precision: 0.16804979253112035
Sensitivity/Recall: 0.5605536332179931
Specificity: 0.6540120793787748
F Score: 0.25857940941739826
```

4 Brief Comparison of Neural Network Performance

Following the review of more straightforward models we thought it pertinent to examine a basic neural network. Regardless of its performance it should give some further insight into the data.

The code below constructs the model for the neural network, we also track the history of the model training against the epochs.

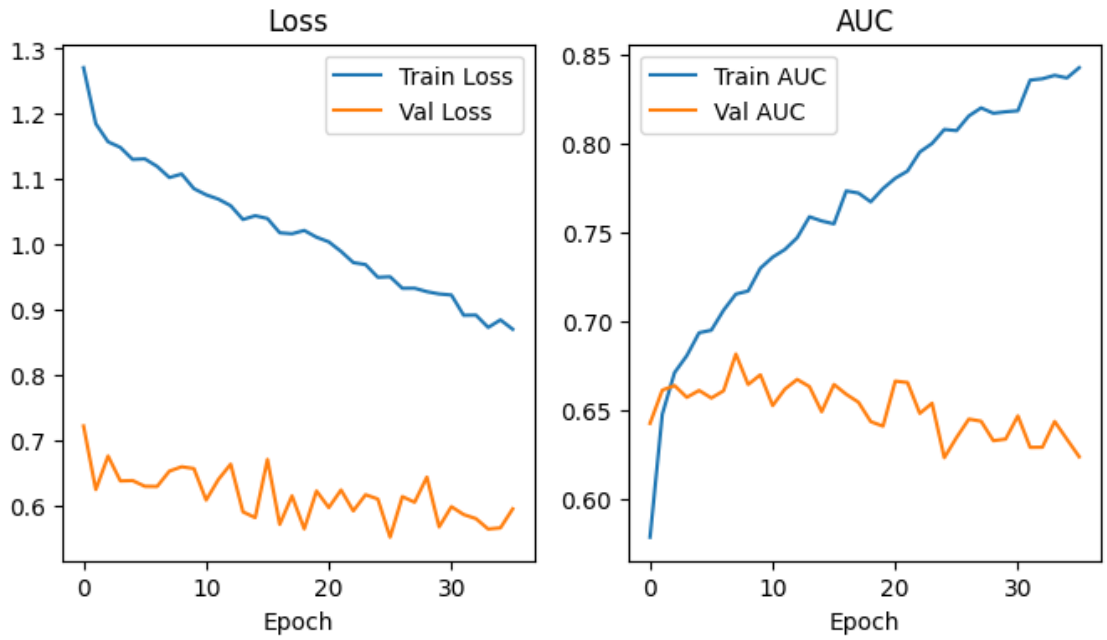
[ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.utils.class_weight import compute_class_weight

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Class weights
class_weights = compute_class_weight("balanced", classes=np.unique(y_train),␣
 ↪y=y_train)
class_weight_dict = dict(enumerate(class_weights))
class_weight_dict = {0: 1, 1: 7.86}

# Build model
D = X_train_scaled.shape[1]

input_layer = keras.Input(shape=(D,))
h1 = keras.layers.Dense(128, activation='relu')(input_layer)
h1 = keras.layers.Dropout(0.3)(h1)
h2 = keras.layers.Dense(64, activation='relu')(h1)
h2 = keras.layers.Dropout(0.2)(h2)
output_layer = keras.layers.Dense(1, activation='sigmoid')(h2)

model_nn = keras.Model(inputs=input_layer, outputs=output_layer)

model_nn.compile(
    loss='binary_crossentropy',
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    metrics=[
        keras.metrics.BinaryAccuracy(name='accuracy'),
```

```
        keras.metrics.AUC(name='auc'),
    ],
)

model_nn.summary()

# Early stopping
early_stop = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.001,
    mode='min',
    patience=10,
    restore_best_weights=True
)

history = model_nn.fit(
    X_train_scaled, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.1,
    class_weight=class_weight_dict,
    callbacks=[early_stop],
    shuffle=True,
    verbose=1
)
```

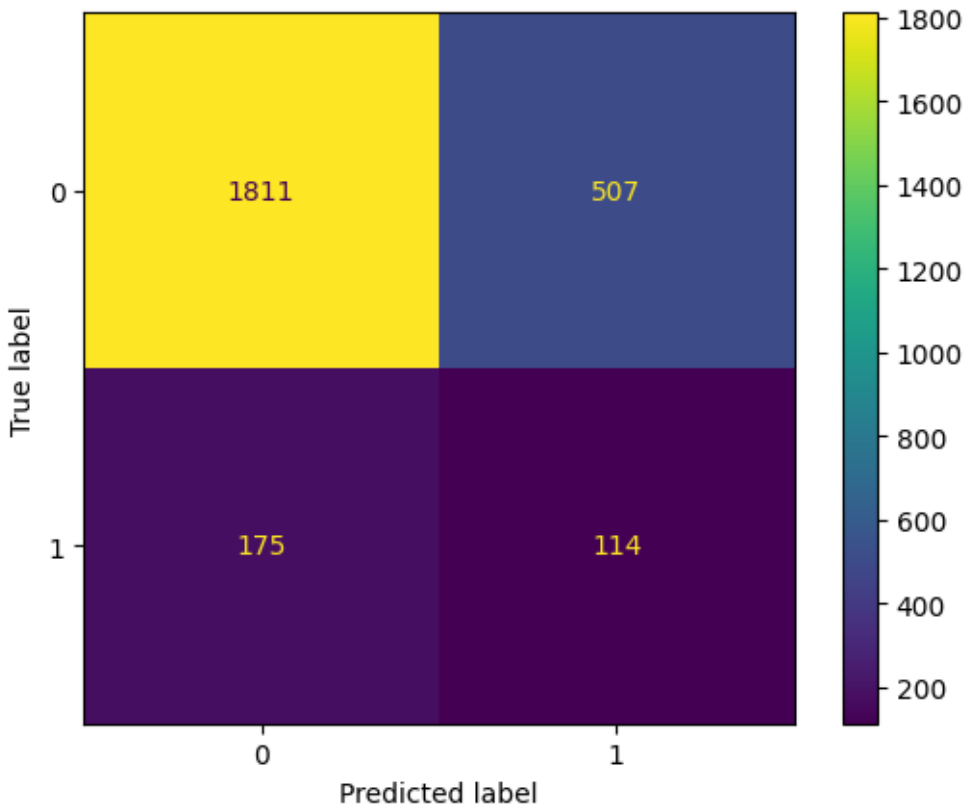
Below we plot the results from this neural network.

[78]:
```
# Plot training history
fig, ax = plt.subplots(1, 2, figsize=(8, 4))
ax[0].plot(history.history['loss'], label='Train Loss')
ax[0].plot(history.history['val_loss'], label='Val Loss')
ax[0].set_title('Loss')
ax[0].set_xlabel('Epoch')
ax[0].legend()
ax[1].plot(history.history['auc'], label='Train AUC')
ax[1].plot(history.history['val_auc'], label='Val AUC')
ax[1].set_title('AUC')
ax[1].set_xlabel('Epoch')
ax[1].legend()
plt.show()

# Predict
y_prob = model_nn.predict(X_test_scaled)
y_pred = (y_prob >= 0.5).astype(int)

print(f"Accuracy: {metrics.accuracy_score(y_test, y_pred)}")
```

```
print(f"Precision: {metrics.precision_score(y_test, y_pred)}")
print(f"Sensitivity/Recall: {metrics.recall_score(y_test, y_pred)}")
print(f"Specificity: {metrics.recall_score(y_test, y_pred, pos_label=0.0)}")
print(f"F Score: {metrics.f1_score(y_test, y_pred)}")

ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
```




```
82/82             0s 822us/step
Accuracy: 0.7383966244725738
Precision: 0.18357487922705315
Sensitivity/Recall: 0.3944636678200692
Specificity: 0.7812769628990509
F Score: 0.25054945054945055
```



Whilst the neural network did perform better on accuracy and specificity compared to the logistic regression model. This would indicate that improved performance in correctly identifying when a person was not depressed. However, the logistic regression does display better recall overall which means that it is better at detecting when a child is depressed. This would indicate that the logistic regression performs better for this task or at least that the neural network did not definitely yield clear impprovement. From a deployment standpoint this makes the explainable logistic regression model superior. It also gives credence to the idea that the limiting factor in this case is the dataset rather than the model's employed.

5 Model Interpretation, Performance, and Recommendations

Regardless of how the data is balanced to equally consider observations of depression and lack of depresssion, the main predictors of depression have to do with wealth. Specifically, water insufficiency and a lack of plumbing are significant indicators of likely depression, while flushing toilets and traditionally-finished roofing are significant indicators of likely non-depression. Beyond these more interpretable wealth indicators, the region in which the child lives and the household cluster to which they belong are also good predictors of depression in the model; features that likely also indirectly indicate household wealth. Additionally, the only other significant feature the logistic regressions consider significant is the presence of any answer to the question related to the child being punished by strikes to the hands, arms, or legs as negatively correlated with the presence of depression. That is, if that question is answered, it is less likely that the child in question

is reported as exhibiting signs of depression, regardless of the actual answer provided. All other features retained through exploratory data analysis and feature engineering still have some impact on the predictions by the nature of the ridge regularization used in fitting the model. However, their impact is not as significant on the prediction as the features detailed above and shown in the coefficient plots in the model tuning section.

Neither balancing method performs particularly well in predicting depression status in children by linear regression. Using balancing weights based on the presence of depression across the dataset achieves slightly better recall at the expense of slightly worse accuracy, precision, specificity, and F1 score when compared to the automatic balancing done based on the training data observed by the regression pipeline. These differences are minimal, so this difference likely comes down to random variation and likely does not indicate significant differences between the two balancing methods. It is therefore appropriate to talk of the results as pertaining to logistic regression as a method rather than two disparate applications of logistic regression. Applying logistic regression to this dataset yields a model that significantly overpredicts depression by a significant degree. Even still, it fails to predict ~45% of cases, while predicting depression in ~35% of cases where no depression was observed. ~80% of its predictions of depression are cases where no depression was observed, while ~10% of its predictions of no depression were cases where depression was observed. These predictions led to overall evaluations of predictive performance as measured by F1 Score of ~0.25. This would indicate poor predictive performance of Logistic Regression models to this dataset.

As a frame of reference, neural networks were fit to the same dataset in an effort to get maximum theoretical predictive performance. However, these applications of neural networks achieved similar predictive performance to the Logistic Regression models. This would imply that the dataset as a whole is not suited to predicting depression in children with much more assurance than observed in the Logistic Regression model, and certain changes to data collection ought to be made to better predict depression in children.

In general, the most significant predictive features are wealth-related. Importantly, though the actual features observed to have impact are features such as drinking water access and housing construction methods, these are likely sypmtoms of the broader issue of lack of wealth. Therefore, measures that increase household wealth would likely translate into the greatest reductions in observed childhood depression. Measures that improve features such as drinking water access or roof construction material may have impact, though this impact may not be as great as these features do not address the general lack of wealth in a household and instead address its symptoms, which may therefore yield less direct improvement in childhood depression rates. Ultimately, the performance of the Logistic Regression model is limited by the dataset's lack of predictive power on depression in children. Depending on feasibility, it may be useful to more aggressively cohort children by age in survey responses, as the current ages 5-17 bracket is quite large, and the factors that may cause depression in children at the lower range of that bracket are likely different than the factors that may cause depression in children at the upper range of that bracket. Additionally, where possible, it is likely useful to collect responses directly from the children as opposed to from their guardians. This would allow for better observation of the target feature, as a child's guardian will likely have less knowledge of a child's mental state than the child themselves, as well as better observation of other features that guardians may be more likely to answer dishonestly than children in an effort to appear more socially acceptable.

6 Generative AI statement

AI was used to aid in debugging and helping to determine improvement areas.

7 References

UNICEF (2024) Global Annual Results Report 2023: Mental health. UNICEF. Available at: https://www.unicef.org/reports/global-annual-results-report-2023-mental-health (Accessed: 14 April 2026)

[8]:
```
# Run the following to render to PDF
!jupyter nbconvert --to pdf Machine_Learning_Project.ipynb
```

```
[NbConvertApp] Converting notebook Machine_Learning_Project.ipynb to pdf
[NbConvertApp] Support files will be in Machine_Learning_Project_files/
[NbConvertApp] Making directory ./Machine_Learning_Project_files
[NbConvertApp] Writing 135824 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 349825 bytes to Machine_Learning_Project.pdf
```

[ ]:
```

```