In [1]:

```python
import pandas as pd
import numpy as np
```

In [2]:

```python
df_train = pd.read_csv('preprocessing_data/train_data_final_feat_no_preprocessing.csv')
print(df_train.columns)
```

```
Index(['PUMA', 'AGEP', 'CIT', 'LANX', 'MAR', 'MIG', 'SCHL', 'SEX', 'WKHP',
       'WKL', 'WRK', 'DIS', 'ESR', 'HICOV', 'MSP', 'NATIVITY', 'OCCP', 'POBP',
       'POVPIP', 'PRIVCOV', 'PUBCOV', 'RAC1P', 'RC', 'year',
       'poverty_risk_score', 'CA_Region'],
      dtype='str')
```

In [3]:

```python
def check_nulls(data):
    null_counts = data.isnull().sum()
    return null_counts[null_counts > 0]

print("Nulls in Training Set:\n", check_nulls(df_train))
```

```
Nulls in Training Set:
 LANX     87133
MIG      16042
SCHL     50568
WKHP    885615
WKL     320549
WRK     501150
ESR     320549
MSP     296846
OCCP    742911
RC       24717
dtype: int64
```

In [4]:

```python
#Looking at unique values for some of the columns
```

unique_CIT= df_train['CIT'].value_counts() print(f"Unique values in 'CIT' column: {unique_CIT}")

unique_ESR = df_train['ESR'].value_counts() print(f"Unique values in 'ESR' column: {unique_CIT}")

unique_ESR = df_train['ESR'].value_counts() print(f"Unique values in 'ESR' column: {unique_ESR}")

unique_HICOV= df_train['HICOV'].value_counts() print(f"Unique values in 'HICOV' column: {unique_HICOV}")

unique_WKL= df_train['WKL'].value_counts() print(f"Unique values in 'WKL' column: {unique_WKL}")

In [5]:

```python
#Recoding PRIVCOV, PUBCOV, HICVOC
```

In [6]:

```python
#For baseline, we're going to want to binarize everything we can, ACS uses 1/2 rather thai
df_train['PRIVCOV'] = df_train['PRIVCOV'].map({1: 1, 2: 0})
df_train['PUBCOV'] = df_train['PUBCOV'].map({1: 1, 2: 0})
df_train['HICOV'] = df_train['HICOV'].map({1: 0, 2: 1})
df_train['DIS'] = df_train['DIS'].map({1: 1, 2: 0})
df_train['SEX'] = df_train['SEX'].map({1: 1, 2: 0})


#  Usually 1 we arent looking at data < 2018 {""0"": ""No and GQ records for 2016 and ear.
df_train['RC'] = df_train['RC'].fillna(0)
```

In [7]:

```python
unique_poverty_class= df_train['poverty_risk_score'].value_counts()
print(f"Unique values in 'poverty_class' column: {unique_poverty_class}")
```

```
Unique values in 'poverty_class' column: poverty_risk_score
0.0    1375161
1.0     268696
2.0     109241
3.0     104528
Name: count, dtype: int64
```

In [7]:

```python

```

In [8]:

```python
df_train['MAR'] = df_train['MAR'].map(lambda x: 1 if x == 1 else 0)
```

In [9]:

```python
#!!!!!! SPECIFICALLY FOR BASELINE ONLY !!!!
# There are going to be feature engineering that the baseline model will require for logis
# for ex Dropping features that are nearly identical due to the multicollinearity study
# keeping them can lead to mathematical noise in a logistic regression bc they are already
# For the specific model training piece we will reimplement these!

#For baseline we will keep Age, and update WKHP with na = 0, year leave as is, wont use PU
df_train = df_train.drop(columns=['MSP', 'WRK', 'WKL', 'NATIVITY'])

df_train['MAR'] = df_train['MAR'].map(lambda x: 1 if x == 1 else 0)
df_train['CIT'] = df_train['CIT'].apply(lambda x: 1 if x < 5 else 0)
df_train['ESR'] = df_train['ESR'].apply(lambda x: 1 if x in [1, 2] else 0)

df_train['WKHP'] = df_train['WKHP'].fillna(0)

# 1. LANX - Binary: Speaks other language
df_train['LANX'] = (df_train['LANX'] == 1.0).astype(int)

# 2. MIG - Binary: Moved in the last year
df_train['MIG'] = df_train['MIG'].isin([2, 3]).astype(int)

# 3. OCCP - Grouping (Top 10 + Other)
# We use the training data to find the top 10 to avoid leakage
df_train['OCCP'] = df_train['OCCP'].fillna('NILF')
top_10_occp = df_train['OCCP'].value_counts().nlargest(10).index
def group_occp(x):
    if x == 'NILF': return 'NILF'
    if x in top_10_occp: return x
    return 'Other'

df_train['OCCP_grouped'] = df_train['OCCP'].apply(group_occp)

# 4. RAC1P - Ensure it is treated as a string for One-Hot Encoding later
df_train['RAC1P'] = df_train['RAC1P'].astype(str)


def recode_education(val):
    if pd.isna(val): return 0
    if val <= 15: return 0   # No HS Diploma
    if val <= 17: return 1   # HS Diploma / GED
    if val <= 20: return 2   # Some College / Associate
    return 3                 # Bachelor's or Higher

df_train['SCHL_Tier'] = df_train['SCHL'].apply(recode_education)
df_train['Born_in_CA'] = (df_train['POBP'] == 6).astype(int)
```

In [10]:

```python
#We must drop columns SCHL , pobp, and cit (and use recoded versions) WE MUST use the sta
```

In [11]:

```
from sklearn.utils import resample
#converting to ints first
#df_train['poverty_class'] = df_train['poverty_class'].astype(int)


# Separate classes
df_stable = df_train[df_train['poverty_risk_score'] == 0]
df_at_risk = df_train[df_train['poverty_risk_score'] > 0] # Classes 1, 2, 3

# Downsample the 'Stable' group to match the total count of 'At Risk'
df_stable_downsampled = resample(df_stable,
                                 replace=False,
                                 n_samples=len(df_at_risk),
                                 random_state=42)

# Combine back
train_balanced = pd.concat([df_stable_downsampled, df_at_risk])

print("New Class Distribution:")
print(train_balanced['poverty_risk_score'].value_counts())
```

```
New Class Distribution:
poverty_risk_score
0.0    482465
1.0    268696
2.0    109241
3.0    104528
Name: count, dtype: int64
```

In [19]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# --- STEP 0: FINAL DATA CLEANUP (Crucial to prevent NaN Errors) ---
# Ensure targets are integers
y_train = train_balanced['poverty_risk_score'].astype(int)
y_test = df_train['poverty_risk_score'].astype(int)

# Identify the exact features we want to use (Excludes raw SCHL/OCCP/PUMA/POVPIP)
# We use the 'Tier' and 'Grouped' versions to avoid the NaN and String errors
final_features = [
    'AGEP', 'WKHP', 'SEX', 'DIS', 'CIT', 'Born_in_CA',
    'SCHL_Tier', 'OCCP_grouped', 'CA_Region', 'RAC1P', 'year'
]

# --- STEP 1: PREPARE RAW FEATURE MATRICES ---
X_train_raw = train_balanced[final_features].copy()
X_test_raw = df_train[final_features].copy()

# Fix any lingering NaNs in numeric columns before scaling
X_train_raw['WKHP'] = X_train_raw['WKHP'].fillna(0)
X_test_raw['WKHP'] = X_test_raw['WKHP'].fillna(0)

# --- STEP 2: ONE-HOT ENCODING (Solves 'NILF' String Error) ---
# This converts 'NILF', 'Other', and 'Region' strings into 0/1 columns
X_train = pd.get_dummies(X_train_raw,
                         columns=['SCHL_Tier', 'OCCP_grouped', 'CA_Region', 'RAC1P'],
                         drop_first=True)

X_test = pd.get_dummies(X_test_raw,
                        columns=['SCHL_Tier', 'OCCP_grouped', 'CA_Region', 'RAC1P'],
                        drop_first=True)
```

```python
# --- STEP 3: ALIGNMENT (Ensures Train and Test have identical columns) ---
X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

# --- STEP 4: SELECTIVE SCALING (Continuous only) ---
cont_features = ['AGEP', 'WKHP']
scaler = StandardScaler()

# Fit on train, transform both
X_train[cont_features] = scaler.fit_transform(X_train[cont_features].astype(float))
X_test[cont_features] = scaler.transform(X_test[cont_features].astype(float))

# --- STEP 5: MULTINOMIAL LOGISTIC REGRESSION ---
model = LogisticRegression(
    #multi_class='multinomial', # Set for 4-class target (0,1,2,3)
    solver='lbfgs',            # Standard solver for multinomial
    max_iter=5000,             # Sufficient for convergence
    random_state=42,
    n_jobs=-1                  # Speed up using all CPU cores
)

# Train the model
model.fit(X_train, y_train)

# --- STEP 6: EVALUATION ---
y_pred = model.predict(X_test)

print("=== Baseline Logistic Regression Performance (2024) ===")
print(classification_report(y_test, y_pred))
```

```
/Users/ingridaltamirano/IdeaProjects/Capstone-Data-Ingestion-Test/.venv/lib/python3.12/sit
e-packages/sklearn/linear_model/_logistic.py:1184: FutureWarning: 'n_jobs' has no effect s
ince 1.8 and will be removed in 1.10. You provided 'n_jobs=-1', please leave it unspecifie
d.
  warnings.warn(msg, category=FutureWarning)
/Users/ingridaltamirano/IdeaProjects/Capstone-Data-Ingestion-Test/.venv/lib/python3.12/sit
e-packages/sklearn/linear_model/_logistic.py:406: ConvergenceWarning: lbfgs failed to conv
erge after 5000 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=5000).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
=== Baseline Logistic Regression Performance (2024) ===
              precision    recall  f1-score   support

           0       0.80      0.86      0.83   1375161
           1       0.26      0.34      0.30    268696
           2       0.00      0.00      0.00    109241
           3       0.24      0.06      0.09    104528

    accuracy                           0.69   1857626
   macro avg       0.33      0.32      0.30   1857626
weighted avg       0.64      0.69      0.66   1857626
```

In [13]:

```python
null_counts = X_train.isnull().sum()
print(null_counts[null_counts > 0])
```

```
SCHL    28172
dtype: int64
```

In [ ]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# 1. One-Hot Encode categorical columns first
X = pd.get_dummies(train_balanced.drop(columns=['poverty_risk_score', 'POVPIP']),
                   columns=['OCCP_grouped', 'CA_Region', 'SCHL_Tier'],
                   drop_first=True)
y = train_balanced['poverty_risk_score']

# 2. Scale ONLY continuous features
cont_features = ['AGEP', 'WKHP']
scaler = StandardScaler()
X[cont_features] = scaler.fit_transform(X[cont_features])

# 3. Repeat the same encoding/scaling for your 2024 test set
# (Important: use the scaler.transform() from train on the test set)
```

In [ ]:
```python
from sklearn.metrics import classification_report

y_pred = model.predict(X_test_scaled)
print(classification_report(y_test, y_pred))
```