🤯 Welcome To the Visionary Data Generator!!!

This is a synthetic data generator that creates synthetic data based off real world known values, using a weighted probability algorithm.

It produces a .csv file, that allows model/ipf_model.py to train directly off it.

Approach:

- Sample covariates from sensible base rates and distributions
- Compute Homeless probability via a logistic model with domain-informed weights
- Calibrate intercept to match a target overall Homeless rate
- Save to model/data/synthetic_homelessness_data.csv

In [1]:
```
from __future__ import annotations

import os
from pathlib import Path
import numpy as np
import pandas as pd

from maths import *

print("Numpy: " + np.__version__)
print("Pandas: " + pd.__version__)
```

```
Numpy: 2.0.1
Pandas: 2.3.3
```

Part 1 - Base Variable Sampling

1.1 Macros & Age-Band Sampler

Due to Australian open datasets defining age in this format of age ranges, this function helps generate random intigers between age ranges with **np.random.Generator**

In [2]:
```
## -- Macros & Initialiser Lists -- ##
SEED = 42 # random seed for reproduction
TOTAL = 95359 # total population
TARGET_RATE = 0.533  # homeless population rate

genders = ['Male', 'Female']
ages = ['0-17', '18-24', '25-34', '35-44', '45-54', '55-64', '65+']
locations = ['ACT', 'NSW', 'NT', 'QLD', 'SA', 'TAS', 'VIC', 'WA']  # match c

def sample_age_from_band(band: str, random_generator: np.random.Generator) -
```

```
    ranges = {
        '0-17': (0, 17),
        '18-24': (18, 24),
        '25-34': (25, 34),
        '35-44': (35, 44),
        '45-54': (45, 54),
        '55-64': (55, 64),
        '65+': (65, 90),
    }
    lo, hi = ranges[band]
    return int(random_generator.integers(lo, hi + 1))
```

1.2 Sampling Covariate Variables off Known Australian Demographics Distrobutions

Note - Conditional Probabilities NOT considered in this model.

In [3]:
```
rng = np.random.default_rng(SEED) # creates random number generator object

# Attribute probabilities (Later used in RNG)
gender_p = np.array([0.389, 0.611])
age_counts = np.array([27295, 13804, 15977, 16519, 11576, 6479, 3709], dtype
age_p = age_counts / age_counts.sum()
loc_p = np.array([0.02, 0.35, 0.03, 0.24, 0.09, 0.02, 0.21, 0.04])
loc_p = loc_p / loc_p.sum()

# Binary feature probabilities (Later used in RNG)
p_drug = 0.14
p_mental = 0.28
p_indigenous = 0.30
p_dv = 0.42

# NOTE - random.choice(possible outcomes, generated length, probability)

# Sample covariates
gender = rng.choice(genders, size=TOTAL, p=gender_p)
location = rng.choice(locations, size=TOTAL, p=loc_p)
drug = (rng.random(TOTAL) < p_drug).astype(int)
mental = (rng.random(TOTAL) < p_mental).astype(int)
indigenous = (rng.random(TOTAL) < p_indigenous).astype(int)
dv = (rng.random(TOTAL) < p_dv).astype(int)

# Choose from age range
age_band = rng.choice(ages, size=TOTAL, p=age_p)
age_numeric = np.array([sample_age_from_band(b, rng) for b in age_band], dty

# Map categorical to numeric for model contribution
gender_num = (gender == 'Male').astype(int)  # align with ipf2 encoding (Mal
```

Part 2 - Logistic Model to Generate Homeless Probability

To consider the differing effects of mutliple predictors to the probability of becoming homeless, we chose to implement a **Logistic Regression Model**, defined by:

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k)}}$$

Our Implemention of the Logit Model

Our one works by first calculating the **linear combination (logit, Z)** of predictors for an individual entry, with weights heusistically defined and uniform for all entries:

$$Z_{Individual} = \sum w_{attribute} \times Attribute$$

Then apply the **logistic/sigmoid** function to the logit (Z):

$$P(Homeless) = \sigma(Z_{Individual})$$

Yielding our final probability for the entry:

$$P(Homeless) = \sigma(\sum w_{attribute} \times Attribute)$$

2.1 Defining Weights

In [4]:
```python
# Age weight map
w_age = {
    '0-17': -0.60,
    '18-24': 0.10,
    '25-34': 0.25,
    '35-44': 0.35,
    '45-54': 0.30,
    '55-64': 0.12,
    '65+': -0.25,
}

# Location weights map
w_location = {
    'ACT': -0.05,
    'NSW': 0.00,
    'NT': 0.10,
    'QLD': 0.02,
    'SA': 0.03,
    'TAS': -0.02,
    'VIC': 0.15,
    'WA': 0.00,
}

w_gender = float(os.getenv('W_GENDER', '0.08'))       # small effect
w_drug = float(os.getenv('W_DRUG', '1.1'))            # tuned up
w_mental = float(os.getenv('W_MENTAL', '0.7'))        # tuned up
```

```
w_indigenous = float(os.getenv('W_INDIG', '0.25'))
w_dv = float(os.getenv('W_DV', '1.3'))                # tuned up
```

2.2 Calculating the Logit (No Intercept Term)

$$Z_{Individual} = \sum w_{attribute} \times Attribute$$

In [5]:
```
age_eff = np.array([w_age[b] for b in age_band])
loc_eff = np.array([w_location[x] for x in location])

# NOTE - This is our LOGIT!!!
    # the Linear predictor without intercept
z_no_b = (
    w_gender * gender_num
    + w_drug * drug
    + w_mental * mental
    + w_indigenous * indigenous
    + w_dv * dv
    + age_eff
    + loc_eff
).astype(float)
```

2.3 Bias Calibration

TODO - Explain why this is needed

In [6]:
```
b0 = calibrate_intercept(z_no_b, TARGET_RATE)
```

2.4 Calculating Intercept Term & Applying the Logistic Funtion

The sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Has a domain of $x \in \{-\infty, \infty\}$, and has a range of $y \in \{0, 1\}$. This means it squeezes its input ($x \in \real$) into a probability of $\{0, 1\}$.

In [7]:
```
p_h = logistic(z_no_b + b0) # apply the logistic function

homeless = (rng.random(TOTAL) < p_h).astype(int) # apply the probability to
```

Part 3 - Exporting the Data

In [8]:
```
# Build DataFrame in the same schema as ipf2 output
df = pd.DataFrame({
    'Gender': (gender == 'Male').astype(int),  # ipf_model treats 1 as Male
    'Age': age_numeric,
```

```
    'Drug': drug,
    'Mental': mental,
    'Indigenous': indigenous,
    'DV': dv,
    'Homeless': homeless,
})

# One-hot encode locations as booleans, in the exact order used by ipf_model
for state in ['ACT', 'NSW', 'NT', 'QLD', 'SA', 'TAS', 'VIC', 'WA']:
    df[f'Location_{state}'] = (location == state)

# Reorder columns to match ipf2
cols = [
    'Gender', 'Age', 'Drug', 'Mental', 'Indigenous', 'DV',
    'Location_ACT', 'Location_NSW', 'Location_NT', 'Location_QLD',
    'Location_SA', 'Location_TAS', 'Location_VIC', 'Location_WA',
    'Homeless'
]
df = df[cols]

out_path = Path('model/data/synthetic_homelessness_data.csv')
out_path.parent.mkdir(parents=True, exist_ok=True)
df.to_csv(out_path, index=False)

print(f"Saved synthetic dataset to: {out_path} (n={len(df):,})")
print(f"Homeless rate: {df['Homeless'].mean()*100:.2f}% (target {TARGET_RATE
# Quick sanity: DV effect by gender (observed)
for g_val, g_name in [(0, 'Female'), (1, 'Male')]:
    sub = df[df['Gender'] == g_val]
    if len(sub) > 0:
        obs_dv1 = sub.loc[sub['DV'] == 1, 'Homeless'].mean()
        obs_dv0 = sub.loc[sub['DV'] == 0, 'Homeless'].mean()
        print(f"DV sanity ({g_name}): P(H=1|DV=1)={obs_dv1:.3f} vs DV=0={obs
```

```
Saved synthetic dataset to: model/data/synthetic_homelessness_data.csv (n=9
5,359)
Homeless rate: 53.16% (target 53.30%)
DV sanity (Female): P(H=1|DV=1)=0.690 vs DV=0=0.406
DV sanity (Male): P(H=1|DV=1)=0.704 vs DV=0=0.424
```