AI Bias Detection & Fairness Framework

Methodology for detecting and mitigating algorithmic discrimination

Version: 1.0

Author: AI Ethics & Governance Team

Category: AI Ethics | Bias Detection | Regulatory Compliance

Regulatory Framework: EU AI Act, UK Equality Act 2010, FCA Handbook

Last Updated: December 12, 2025

This framework establishes Monzo's methodology for detecting, measuring, and mitigating algorithmic bias in machine learning models used for financial decisioning. As AI systems increasingly impact credit access, fraud detection, and risk assessment, ensuring fairness across protected characteristics is both an ethical imperative and regulatory requirement under the EU AI Act and UK Equality Act 2010.

Regulatory Context & Legal Obligations

UK and EU legislation impose strict requirements on AI systems that affect financial decisions:

EU AI Act Requirements

Classification: Credit scoring and fraud detection are "high-risk AI systems" under Article 6

Obligations: Risk management system, data governance, technical documentation, human oversight, accuracy/robustness/cybersecurity measures

Article 10: Training data must be "relevant, representative, free of errors and complete" with examination for "possible biases"

Penalties: Up to €30M or 6% of global turnover for non-compliance

UK Equality Act 2010

Protected Characteristics: Age, disability, gender reassignment, marriage/civil partnership, pregnancy/maternity, race, religion/belief, sex, sexual orientation

Direct Discrimination: Treating someone less favorably because of protected characteristic

Indirect Discrimination: Applying provision/criterion/practice that disadvantages people with protected characteristic (even unintentionally)

Algorithm Context: ML models creating disparate impact on protected groups may constitute indirect discrimination unless objectively justified

✓ **COMPLIANCE: This framework ensures compliance with both EU AI Act Article 10 (bias examination in training data) and UK Equality Act requirements for non-discriminatory financial services.**

Protected Characteristics & Proxy Variables

While Monzo never directly uses protected characteristics as model features, correlated proxy variables can inadvertently encode bias. We must identify and monitor these proxies:

Protected Characteristic	Direct Use	Potential Proxy Variables
Race/Ethnicity	■ Never used	Postcode, surname patterns, geographic location, certain merchant preferences
Gender	■ Never used	Spending patterns (fashion vs. hardware), certain merchant categories, title (Mr/Ms)
Age	■■ Bucketed only	Account tenure, credit history length, technology adoption patterns
Disability	■ Never used	Accessibility feature usage, certain medical merchant patterns, mobility patterns
Religion	■ Never used	Certain merchant transactions (places of worship), dietary restrictions, geographic patterns
Sexual Orientation	■ Never used	Extremely difficult to proxy; potentially certain lifestyle spending patterns

■■ **WARNING: Age is the only protected characteristic used in models, and only in bucketed form (e.g., 18-25, 26-35) to prevent granular age discrimination. Even bucketed age use requires fairness monitoring.**

Fairness Metrics Framework

We measure fairness using multiple complementary metrics, as no single metric captures all aspects of fairness:

1. Demographic Parity (Statistical Parity)

Definition: Positive outcome rate should be equal across groups

Formula: P(■=1 | Group A) ≈ P(■=1 | Group B)

Application: Approval rates for credit/overdraft should be similar across demographic groups

Threshold: Accept if ratio between groups is 0.8-1.2 (80% rule from US EEOC)

```
# Python implementation def demographic_parity(y_pred, sensitive_attr): """ Calculate demographic parity ratio between groups. Returns value between 0 and 1 (1 = perfect parity). """ groups = np.unique(sensitive_attr) approval_rates = [] for group in groups: mask = sensitive_attr == group rate = y_pred[mask].mean() approval_rates.append(rate) min_rate = min(approval_rates) max_rate = max(approval_rates) # Demographic parity ratio return min_rate / max_rate if max_rate > 0 else 0 # Example usage parity_ratio = demographic_parity(predictions, age_groups) if parity_ratio < 0.8: print(f"BIAS ALERT: Demographic parity violation ({parity_ratio:.3f})")
```

2. Equal Opportunity (True Positive Rate Parity)

Definition: Among qualified applicants, approval rates should be equal

Formula: P(■=1 | Y=1, Group A) ≈ P(■=1 | Y=1, Group B)

Application: Customers who will repay loans should be approved at equal rates regardless of demographics

Why Important: Demographic parity alone can hide bias against qualified minority applicants

3. Equalized Odds

Definition: Both true positive AND false positive rates equal across groups

Formula: P(■=1 | Y=y, Group A) = P(■=1 | Y=y, Group B) for both y=0 and y=1

Application: Both good and bad credit risks treated similarly across demographics

Strongest Test: Most stringent fairness criterion - hardest to achieve

4. Calibration

Definition: Predicted probabilities should match actual outcomes within each group

Formula: For each score bucket, P(Y=1 | ■=s, Group A) ≈ P(Y=1 | ■=s, Group B)

Application: If model predicts 30% default risk, ~30% should actually default in each demographic group

Bias Detection Methodology

Step 1: Define Protected Groups

Since we don't collect race/religion/etc., we use proxy segmentation for fairness testing:

Age groups: 18-25, 26-35, 36-45, 46-55, 56-65, 66+

Gender: Male, Female, Other/Prefer not to say (self-reported at account opening)

Geographic proxies: Postcode-level Index of Multiple Deprivation (IMD) deciles 1-10

Ethnicity proxy: Postcode-level ethnic composition (Office for National Statistics data)

Step 2: Automated Bias Scanning

Run fairness metrics on validation dataset before each model deployment:

```
from aequitas.group import Group from aequitas.bias import Bias # Load validation data with
sensitive attributes df = pd.read_csv('validation_set.csv') # Create Aequitas groups g =
Group() xtab, _ = g.get_crosstabs(df, score_thresholds={'score':[0.5]},
attr_cols=['age_group', 'gender', 'imd_decile']) # Calculate bias metrics b = Bias()
bias_df = b.get_disparity(xtab, original_df=df, ref_groups_dict={'age_group':'26-35',
'gender':'Male', 'imd_decile':'5'}) # Flag significant disparities violations = bias_df[
```

```
(bias_df['fpr_disparity'] > 1.2) | (bias_df['fpr_disparity'] < 0.8) |
(bias_df['fnr_disparity'] > 1.2) | (bias_df['fnr_disparity'] < 0.8) ] if len(violations) >
0: print("BIAS VIOLATIONS DETECTED:") print(violations[['attribute_name',
'attribute_value', 'fpr_disparity', 'fnr_disparity']]) # Trigger manual review before
deployment
```