# Practical Exercise — Lesson 2 (Week 5)

## Quick Fairness Probe: Detect and Explain Bias in Two Settings

## Learning outcomes

By the end, you will be able to:

1. Align a **fairness metric** to a concrete **harm to avoid** (access, missed protection, wrongful flags).
2. Compute **per-group metrics**, **gaps/ratios**, **worst-group** values, and (optionally) **95% CIs**.
3. Run a lightweight **LM bias probe** (CrowS-Pairs or WEAT) and interpret the score.
4. Recommend **one actionable mitigation** and explain **how you will re-measure**.

---

## Setup

1. Open a fresh Colab (CPU is fine).
2. Install/import the basics:
3. `!pip -q install datasets transformers torch scikit-learn --upgrade`
4. `# optional for WEAT:`
5. `# !pip -q install wefe`
6. 
7. `import numpy as np, pandas as pd, torch`
8. `from datasets import load_dataset`
9. `from transformers import pipeline, AutoTokenizer, AutoModelForCausalLM`
10. `from sklearn.metrics import confusion_matrix, precision_recall_curve`
11. Add a `SEED = 42` and set numpy/torch RNG for repeatability.

---

# Setting A — Group Fairness in a Classifier

### A1. Choose data & model

- **Preferred (toxicity):** `civil_comments` from HF (has identity columns like `male`, `female`, etc.).
- `ds = load_dataset("civil_comments")["train"].shuffle(seed=42).select(range(5000))`
- `text_col, label_col = "text", "toxicity"  # threshold label later (>=0.5)`

- **Fallback (if identity columns unavailable):** build **keyword slices** (e.g., strings containing identity terms). Note clearly this is an **approximation**.
- **Model:** any small `text-classification` pipeline (for speed):
- `clf = pipeline("text-classification",`
- `               model="distilbert-base-uncased-finetuned-sst-2-english",`
- `               return_all_scores=True, truncation=True)`

## A2. Prepare labels & run inference

1. If your dataset's label is continuous (e.g., toxicity 0–1), **binarize** at 0.5 into `y`.
2. Score a **2–5k** sample. Save: gold `y`, predicted label `ŷ`, and **score** `S` (probability of positive).
3. `def pos_score(out):  # expects list of dicts [{'label':'NEGATIVE','score':...}, ...]`
4. `    return next(d["score"] for d in out if d["label"].upper().startswith("POS"))`
5. 
6. `rows=[]`
7. `for r in ds:`
8. `    s = pos_score(clf(r[text_col]))`
9. `    rows.append(dict(text=r[text_col], y=int(r[label_col] >= 0.5), S=s))`
10. `df = pd.DataFrame(rows)`
11. `df["yhat"] = (df["S"] >= 0.5).astype(int)`

## A3. Declare harm → metric

Pick **one primary harm** and its **matching metric** (write this in a cell):

- **Over-moderation → FPR gap** (wrongful flags).
- **Missed protection → TPR/Recall gap** (Equal Opportunity).
- **Uneven trustworthiness → PPV/Calibration by group**.

## A4. Compute per-group metrics, gaps, worst-group

Select 2–4 groups (e.g., `male`, `female`, `LGBTQ` if available). For each group aaa: compute TP/FP/TN/FN, then **Precision, Recall, F1, FPR, FNR**. Report:

- a **per-group table**
- **Gap** = max–min for your primary metric
- **Ratio** = min/max (optional)
- **Worst-group** value

```
from sklearn.metrics import precision_recall_fscore_support

def group_mask(df, col):  # identity can be float probability or bool
    return (df[col] > 0.5) if df[col].dtype != bool else df[col]

def prf1_fpr_fnr(y, yhat):
```

```
    tn, fp, fn, tp = confusion_matrix(y, yhat, labels=[0,1]).ravel()
    p = tp/(tp+fp+1e-9); r = tp/(tp+fn+1e-9); f1 = 2*p*r/(p+r+1e-9)
    fpr = fp/(fp+tn+1e-9); fnr = fn/(fn+tp+1e-9)
    return p, r, f1, fpr, fnr

groups = [c for c in ds.column_names if c in
("male","female","lgbtq","black","white")]
rows=[]
for g in groups:
    m = group_mask(df, g)
    p,r,f1,fpr,fnr = prf1_fpr_fnr(df.loc[m,"y"], df.loc[m,"yhat"])
    rows.append(dict(group=g, n=int(m.sum()), precision=p, recall=r, f1=f1,
fpr=fpr, fnr=fnr))
rep = pd.DataFrame(rows)
# Example: primary metric = FPR
gap = rep.fpr.max() - rep.fpr.min()
ratio = rep.fpr.min() / (rep.fpr.max()+1e-9)
worst_group = rep.iloc[rep.fpr.idxmax()].group
rep, gap, ratio, worst_group
```

**Optional (recommended): 95% CIs** via bootstrap on the **gap** (1,000 resamples).

### A5. Threshold analysis (if you have scores)

Plot a **precision–recall curve** and choose a deployment threshold aligned to your harm (e.g., choose a point with **Precision ≥ 0.9** if wrongful flags are costly). Recompute your **primary gap** at the chosen threshold.

```
prec, rec, thr = precision_recall_curve(df["y"], df["S"])
# Pick threshold programmatically (example: F1-max) or justify a business
constraint
```

### A6. Mini-interpretation (3–4 sentences)

- Name the **primary gap** and **worst-group**.
- Explain the **user/business harm**.
- Propose **one mitigation** (e.g., threshold by slice; counterfactual data augmentation; domain adaptation) and say **how you will re-measure**.

**Deliverables for Setting A:**

- Table (per-group metrics + **gap/ratio** + **worst-group**), PR curve with chosen threshold (if scores), 3–4 sentence interpretation.

---

# Setting B — Stereotype Bias in a Language Model

Choose **one** option.

### Option B1 — CrowS-Pairs (LM preference test)

**Goal:** fraction of times the LM prefers a **stereotypical** over an **anti-stereotypical** sentence → **S-score**.

#### Steps

1. Load a small slice (≈300 pairs) and a compact causal LM (e.g., `gpt2`):
2. `ds = load_dataset("crows_pairs", "english")["test"].shuffle(seed=42).select(range(300))`
3. `tok = AutoTokenizer.from_pretrained("gpt2")`
4. `lm  = AutoModelForCausalLM.from_pretrained("gpt2"); lm.eval();`
5. Compute **sentence negative log-likelihood (NLL)**:
6. `def sent_nll(text):`
7. `    ids = tok(text, return_tensors="pt")`
8. `    with torch.no_grad():`
9. `        out = lm(**ids, labels=ids["input_ids"])`
10. `    return float(out.loss) * ids["input_ids"].size(1)  # approx NLL`
11. For each pair, compute NLL for stereotypical vs anti-stereotypical; count where **stereotypical is lower (preferred)**.
12. Report **overall S-score** and **by category** (e.g., gender, race, religion).
13. Write a 2–3 sentence **interpretation** (where bias concentrates; potential user-visible harm in completions).

### Option B2 — WEAT (embedding association test)

**Goal: effect size** ddd measuring association strength (e.g., *science↔male* vs *arts↔female*).

#### Steps

1. Install and import `wefe` (optional but easier), or implement manually.
2. Choose **target sets** (e.g., male/female names) and **attribute sets** (science/arts terms).
3. Compute **WEAT effect size** ddd and (optionally) a **permutation p-value**.
4. Report ddd overall and a brief interpretation (e.g., d=0.78d=0.78d=0.78 = large association).

#### Deliverables for Setting B:

- One numeric metric (S-score **or** WEAT effect size) **overall** and **by at least one category/slice**, plus a 2–3 sentence interpretation.

---

# Fairness Results Card (≤1 page, PDF)

Use the following template (copy/paste into a doc and export to PDF):

**Task & Data:** (dataset, sample size, groups)
**Primary harm → Metric:** (e.g., Over-moderation → FPR gap)
**Headline numbers:** Worst-group = __; Gap = __ (± CI if computed); Ratio = __
**Decision/threshold:** (if applicable; justify)
**LM probe:** (metric and brief finding; e.g., CrowS S=63% overall, 75% gender)
**Mitigation next step:** (one concrete action)
**Re-measurement plan:** (what you will recompute to confirm improvement)
**Limitations:** (sample size, proxy groups, etc.)