VirnyFlow User Study Tutorial Notes

Thank you for joining this tutorial and user study. In the next 60 minutes, you will work with VirnyFlow, a tool that helps data scientists build machine learning (ML) pipelines that are not only accurate, but also fair, stable, and aligned with real-world constraints. You will complete two hands-on tasks, using a Google Colab notebook and a HuggingFace space. These notes give you just enough background to complete each task confidently.

Background for Task 1: Virny (Evaluation Module)

In Task 1 you will use Virny, an evaluation module within VirnyFlow, to profile several models trained on the Diabetes Dataset 2019. You will compute accuracy and fairness metrics, interpret them across groups, and compare ML models across various dimensions.

Dataset Description

In all tasks, we will use the ***Diabetes Dataset 2019***, which contains 952 instances and a total of 18 variables. There are 17 independent variables and 1 dependent (target) variable. The target variable, "Diabetic," indicates whether a participant has diabetes (binary: 0 = non-diabetic, 1 = diabetic). The dataset consists of a mix of binary variables (e.g., Gender, Family History, High BP, Smoking, Alcohol, Regular Medicine, Junk Food, Gestational Diabetes, Urination Frequency), categorical variables (Physical Activity, Stress, Blood Pressure Level), and numerical variables (Age, BMI, Hours of Sleep, Hours of Sound Sleep, Number of Pregnancies). The sample is slightly imbalanced in terms of sex distribution, with 580 males (≈61%) and 372 females (≈39%).

Model Fairness

In this tutorial we focus on group fairness, where we care about how a model treats members of different demographic groups. **Accuracy alone is not enough:** traditional evaluation metrics such as accuracy, F1, precision, and recall tell us how often the model is correct *on average*, but they do not reveal *why* errors happen or *which* groups are disproportionately affected by them. A model may appear to perform well overall while still producing unequal error rates across demographic groups, making fairness analysis essential for understanding whether predictions are equitable.

Terminology:

- A **sensitive attribute** (e.g., sex, age) is an attribute that splits the population into groups that may receive unequal benefits or harms due to a model's predictions.

- A **disadvantaged group** (e.g., sex=Female, age>65) consists of individuals who tend to get fewer benefits or worse outcomes than other groups. For example, in loan approvals, applicants from a historically marginalized racial group may receive fewer approved applications compared to applicants from a majority group. Which group is considered disadvantaged depends on the domain (e.g., it may be easier for younger people to get jobs, but it may be harder for them to get loans).

Group fairness aims for the absence of systematic prejudice or favoritism toward any such group. What is considered fair depends on domain, law, and social context, so there is no single "correct" definition of fairness. Instead, practitioners must choose metrics that are legally and contextually appropriate for their task.

In the *Diabetes Dataset 2019*, we use **sex** as the sensitive attribute for fairness analysis, treating **Female** as the disadvantaged group and **Male** as the privileged group, due to well-documented differences in diagnostic patterns and under-treatment for women. In contrast, we do **not** treat age as a sensitive attribute: age is a genuine clinical risk factor for diabetes, meaning older individuals naturally have higher prevalence and different symptom profiles. Enforcing equal prediction or error rates across age groups would therefore be inappropriate and potentially misleading. This highlights that fairness analysis is **context-sensitive**, and whether an attribute should be considered sensitive depends on the domain and what constitutes unfair treatment in that setting.

Group Fairness Metrics

Group fairness metrics help us evaluate whether a model behaves equitably across demographic groups defined by a sensitive attribute A, such as sex, race, or age. Throughout this tutorial, we consider binary classification tasks, where Y denotes the true class label and $\widehat{Y}$ the predicted label. Both take values in $\{0, 1\}$. We treat **label 1 as a positive (advantageous) outcome** and **label 0 as a negative (disadvantageous) outcome**.

In the *diabetes* dataset, a positive prediction ($\widehat{Y} = 1$) corresponds to identifying a patient as high-risk for diabetes. Although the underlying health condition is undesirable, receiving a positive prediction is advantageous within the decision pipeline because it typically triggers additional clinical attention, earlier intervention, and access to preventive care. In contrast, a negative prediction ($\widehat{Y} = 0$) may mean missing needed follow-up, which is disadvantageous for patients who are actually at risk.

Fairness metrics typically compare **prediction rates** or **error rates** between a disadvantaged group ($A = 0$) and a privileged group ($A = 1$). Below are three commonly used fairness metrics, each with a clear description and a concrete example showing when it is appropriate to use.

1. Disparate Impact (DI)

What it measures:
Disparate Impact compares how frequently each group receives positive predictions, expressed as a ratio:

$$DI = \frac{P(\hat{Y} = 1 \mid A = 0)}{P(\hat{Y} = 1 \mid A = 1)}.$$

A DI value close to 1 means both groups receive positive predictions at similar rates. Values significantly above or below 1 indicate disparity: DI>1 indicates that the disadvantaged group (expression in the numerator) is flagged more often than the privileged group (expression in the denominator), while DI<1 indicates that the disadvantaged group is flagged less often..

When to use it:
Use DI when you want to measure access to *beneficial decisions*, regardless of whether predictions are correct. In the diabetes-screening context, DI helps identify whether certain groups are less likely to receive risk flags, and therefore less likely to receive preventive care.

Example (Diabetes Dataset 2019):
Suppose the model predicts high diabetes risk ($\hat{Y} = 1$) for:

- 30% of female patients (A = 0, disadvantaged group), and
- 45% of male patients (A = 1, privileged group).

Then:

$$DI = \frac{0.30}{0.45} = 0.67.$$

This indicates that female patients receive fewer positive predictions relative to male patients, which may translate into reduced access to early medical intervention.

2. FPR Difference (FPRD)

What it measures:
The **False Positive Rate (FPR)** captures how often the model incorrectly predicts a positive outcome for individuals who are truly negative:

$$FPR = P(\hat{Y} = 1 \mid Y = 0).$$

The **FPR Difference (FPRD)** compares this rate across groups:

$$FPRD = FPR_{A=0} - FPR_{A=1}.$$

A value near 0 indicates similar false positive rates across groups.
A positive value means the disadvantaged group receives more false positive predictions;
a negative value means they receive fewer false positives.

When to use it:
Use FPRD when false alarms create burdens or unnecessary costs. In the diabetes context, a false positive may lead to anxiety, unnecessary testing, or misallocation of medical resources.

Example (Diabetes Dataset 2019):
Suppose that, among the patients who are not actually diabetic (Y = 0):

- 12% of female patients (A = 0) are incorrectly flagged as diabetic,
- 7% of male patients (A = 1) are incorrectly flagged as diabetic.

Then:

$$FPRD = 0.12 - 0.07 = 0.05.$$

This shows that female patients experience higher rates of unnecessary alarms, indicating unequal burden.

3. FNR Difference (FNRD)

What it measures:
The **False Negative Rate (FNR)** captures how often the model incorrectly predicts a negative outcome for individuals who are truly positive:

$$FNR = P(\hat{Y} = 0 \mid Y = 1).$$

The **FNR Difference (FNRD)** compares this rate across groups:

$$FNRD = FNR_{A=0} - FNR_{A=1}.$$

A value near 0 indicates similar false negative rates for both groups.
A positive value means the disadvantaged group is *more likely to be missed* by the model.

When to use it:
Use FNRD when missing true cases produces serious harm. In diabetes screening, false negatives delay diagnosis and treatment, potentially leading to severe health deterioration.

Example (Diabetes Dataset 2019):
Suppose that, among the patients who do have diabetes (Y = 1):

- 28% of female patients (A = 0, disadvantaged group) are misclassified as not diabetic,
- 15% of male patients (A = 1, privileged group) are misclassified as not diabetic.

Then:

$$FNRD = 0.28 - 0.15 = 0.13.$$

This indicates that female patients are substantially more likely to be missed by the screening model, resulting in unequal quality of care.

How Virny Measures Fairness

Virny measures fairness by breaking down model performance across user-defined demographic groups, and then comparing selection rates and error rates between these groups. From the user's perspective, this is controlled entirely through the **config YAML**, where you specify the sensitive attributes (e.g., *sex*, *age*), along with which values should be treated as *disadvantaged*. For example:

`sensitive_attributes_dct: {'sex': 'female'}` or
`sensitive_attributes_dct: {'age': [19, 20, 21, 22, 23, 24, 25]}.`

Once these are defined, Virny uses its subgroup analyzers to compute group-specific metrics (FPR, FNR, selection rates, etc.) and its Metric Composer to turn these into disparity metrics such as Disparate Impact, FPR Difference (referred to in Virny as *Equalized Odds FPR*), and FNR Difference (referred to in Virny as *Equalized Odds FNR*).

Virny supports this process in a **highly flexible way**: you can specify **multiple sensitive attributes**, include **non-binary or categorical columns**, and even evaluate **intersectional groups** (e.g., based on combinations of *sex* and *race*). After reading the YAML, Virny automatically computes all subgroup metrics and returns them as a structured dataframe. The Task 1 Colab notebook in this tutorial provides a concrete example of how to define these settings in a YAML file and how Virny performs the resulting fairness analysis.

Background for Task 2: VirnyFlow

In Task 2, you will analyze pre-computed results using **VirnyFlow's visualization interface** to understand optimization dynamics, compare ML pipelines, and reason about trade-offs in responsible ML decision-making. Building on the evaluation logic introduced in Task 1, VirnyFlow integrates the Virny library to compute context-sensitive metrics for each pipeline and incorporates these metrics directly into a multi-objective Bayesian optimization process over the pipeline search space. In this way, VirnyFlow extends Virny from a standalone evaluation module into a **full pipeline optimization** and **experiment management** framework, enabling scalable execution, flexible objective specification, and transparent exploration of trade-offs.

System Workflow

VirnyFlow supports an iterative workflow that guides users through responsible ML pipeline development, from defining evaluation criteria to refining results over time.

- **Step 1: Define evaluation protocol.** The user starts by analyzing the problem context and selecting evaluation metrics that reflect what matters for the task. These may include

predictive performance metrics (e.g., F1), fairness metrics, and stability metrics. This evaluation protocol defines how each pipeline will be assessed.

- **Step 2: Configure experiment.** Next, the user configures an experiment by selecting pipeline components from VirnyFlow's search space, such as ML models, preprocessing methods, and fairness interventions. Along with the *evaluation protocol*, the user specifies the execution budget and optimization settings in an *experiment config* (see Figure 1). Default settings and a standard multi-objective Bayesian optimizer can be used to get started.
- **Step 3: Run experiment.** The user runs the experiment, and VirnyFlow executes and evaluates many candidate pipelines in parallel across multiple workers. During execution, the visualization interface allows users to monitor progress, observe optimization dynamics, and explore trade-offs between objectives.
- **Step 4: Refine experiment.** If the results do not meet expectations, the user can refine the experiment by adjusting pipeline components, changing optimization objectives, or trying different optimizers. VirnyFlow stores all results and supports side-by-side comparison of experiments, making it easy to understand the impact of changes.
- **Step 5: Continue experimentation.** As new data, features, or methods become available, users can continue experimenting and improving pipelines over time. By building on previous experiments and stored results, VirnyFlow supports long-term, iterative development of responsible ML pipelines.

Experiment Config

An *experiment config* is a YAML file that tells VirnyFlow what to optimize, how to measure it, and how to run the experiment. It corresponds to the configuration shown in Figure 1 and defines the pipeline search space, evaluation protocol, optimization objectives, execution budget, and Virny-specific evaluation settings in a single, reproducible specification.

`pipeline_args`

- `dataset`: dataset used for training and evaluation
- `sensitive_attrs_for_intervention`: sensitive attributes targeted by fairness interventions
- `null_imputers`: candidate missing-value imputation methods
- `fairness_interventions`: bias mitigation techniques applied during training
- `models`: ML models included in the search space

`optimisation_args`

- `ref_point`: reference point for multi-objective optimization
- `objectives`: optimization objectives, including metric name (e.g., F1, SRD, Label Stability), evaluation group (e.g., overall, sex, race, etc.), and weight in optimization

- `optimizer`: Bayesian optimizer configuration, including surrogate model and acquisition function
- `max_total_pipelines_num`: maximum total number of evaluated pipelines
- `num_workers`: number of parallel workers
- `num_pp_candidates`: number of pipeline candidates per iteration for parallelism
- `training_set_fractions_for_halting`: training data fractions used for pruning
- `exploration_factor`: degree of exploration in the search process
- `risk_factor`: trade-off between optimistic and conservative optimization

`virny_args`

- `bootstrap_fraction`: fraction of data used for bootstrap sampling
- `n_estimators`: number of bootstrap estimators for stability and uncertainty metrics
- `sensitive_attrs`: sensitive attributes and subgroup definitions, including intersectional groups

```yaml
pipeline_args:
  dataset: "folk_pubcov"
  sensitive_attrs_for_intervention: ["SEX", "RAC1P"]
  null_imputers: ["median-mode", "miss_forest", "datawig"]
  fairness_interventions: ["DIR", "AD"]
  models: ["lr_clf", "rf_clf", "lgbm_clf", "gandalf_clf"]

optimisation_args:
  ref_point: [0.40, 0.10, 0.10]
  objectives:
   - {name: "obj_1", metric: "F1", group: "overall", weight: 0.25}
   - {name: "obj_2", metric: "SRD", group: "SEX&RAC1P", weight: 0.5}
   - {name: "obj_3", metric: "Label_Stability", group: "overall",
      weight: 0.25}
  optimizer: {surrogate_type: "prf", acq_type: "ehvi",
              acq_optimizer_type: "random_scipy"}
  max_total_pipelines_num: 100
  num_workers: 32
  num_pp_candidates: 4
  training_set_fractions_for_halting: [0.5, 1.0]
  exploration_factor: 0.5
  risk_factor: 0.5

virny_args:
  bootstrap_fraction: 0.8
  n_estimators: 50
  sensitive_attrs: {SEX:'2', RAC1P:['2','3','4','5','6','7','8','9'],
                    SEX&RAC1P: None}
```

Figure 1: Experiment config example: Multi-objective optimization with model selection, fairness interventions, and null imputation.

Visualization Interface

VirnyFlow's visualization interface provides an interactive way to explore the results of an experiment and understand how different pipelines perform across multiple objectives. It contains four main pages, each serving a distinct purpose. The **Execution Progress** page shows the current status of the experiment, including how many pipelines have been run and which ones are performing well. The **Pipeline Performance** page provides a detailed profile of a selected pipeline, including its accuracy, fairness, stability metrics, and group-level breakdowns. The **Pipeline Optimization** page visualizes how the optimization process evolves over time, highlighting improvements in objectives, trade-offs, and hyperparameter patterns. Finally, the **Pipeline Comparison** page allows users to compare multiple experiment configurations or pipeline candidates side by side to understand how changes in settings affect performance.