

# PyCFRL: A Python library for counterfactually fair offline reinforcement learning via sequential data preprocessing

Jianhan Zhang<sup>1</sup>, Jitao Wang<sup>2</sup>, Chengchun Shi<sup>3</sup>, John D. Piette<sup>4</sup>, Donglin Zeng<sup>2</sup>, and Zhenke Wu<sup>2¶</sup>

<sup>1</sup> Department of Statistics, University of Michigan, USA <sup>2</sup> Department of Biostatistics, University of Michigan, USA <sup>3</sup> Department of Statistics, London School of Economics, UK <sup>4</sup> Department of Health Behavior and Health Equity, School of Public Health, University of Michigan, USA ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

Editor: [Open Journals](#) ↗

## Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## Summary

Reinforcement learning (RL) aims to learn and evaluate a sequential decision rule, often referred to as a “policy”, that maximizes expected discounted cumulative rewards to optimize the population-level benefit in an environment across possibly infinitely many time steps. RL has gained popularity in fields such as healthcare, banking, autonomous driving, and, more recently, large language model fine-tuning. However, the sequential decisions made by an RL algorithm, while optimized to maximize overall population benefits, may disadvantage certain individuals who are in minority or socioeconomically disadvantaged groups. A fairness-unaware RL algorithm learns an optimal policy that makes decisions based on the *observed* state variables. However, if certain values of the sensitive attribute influence the state variables and lead the policy to systematically withhold certain actions from an individual, unfairness will result. For example, Hispanics may under-report their pain levels due to cultural factors, misleading a fairness-unaware RL agent to assign less therapist time to these individuals (Piette et al., 2023). Deployment of RL algorithms without careful fairness considerations can raise concerns and erode public trust in high-stakes settings.

To formally define and address the fairness problem in the novel sequential decision-making settings, Wang et al. (2025) extended the concept of single-stage counterfactual fairness (CF) in a structural causal framework (Kusner et al., 2018) to the multi-stage setting and proposed a data preprocessing algorithm that ensures CF. A policy is counterfactually fair if, at every time step, the probability of assigning any action does not change had the individual’s sensitive attribute taken a different value, while holding constant other historical exogenous variables and actions. In this light, the data preprocessing algorithm ensures CF by constructing new state variables that are not impacted by the sensitive attribute(s). Reward preprocessing is also conducted, but with a different purpose to improve the value of the learned optimal policy rather than to ensure CF. We refer interested readers to Wang et al. (2025) for more technical details.

The PyCFRL library implements the data preprocessing algorithm proposed by Wang et al. (2025) and provides functionalities to evaluate the value (expected discounted cumulative reward) and counterfactual unfairness level achieved by any given policy. Here, “CFRL” stands for “Counterfactual Fairness in Reinforcement Learning”. The library produces preprocessed trajectories that can be used by an off-the-shelf offline RL algorithm, such as fitted Q-iteration (FQI) (Riedmiller, 2005), to learn an optimal CF policy. The library can also simply read in any policy following a required format and return its value and counterfactual unfairness level in the environment of interest, where the environment can be either pre-specified or learned

44 from the data.

## 45 Statement of Need

46 Many existing Python libraries implement algorithms designed to ensure fairness in machine  
 47 learning. For example, Fairlearn (Weerts et al., 2023) and aif360 (Bellamy et al., 2018)  
 48 provide tools for mitigating bias in single-stage machine learning predictions under statistical  
 49 association-based fairness criteria such as demographic parity and equal opportunity. However,  
 50 existing libraries do not focus on counterfactual fairness, which defines an individual-level  
 51 fairness concept from a causal perspective, and they cannot be easily extended to the general  
 52 RL setting. Scripts available from ml-fairness-gym (D'Amour et al., 2020) allow users to  
 53 simulate unfairness in sequential decision-making, but they neither implement algorithms that  
 54 reduce unfairness nor address CF. To our knowledge, Wang et al. (2025) is the first work to  
 55 study CF in RL. Correspondingly, PyCFRL is also the first code library to address CF in the RL  
 56 setting.

57 The contribution of PyCFRL is two-fold. First, PyCFRL implements a data preprocessing algorithm  
 58 that ensures CF in offline RL. For each individual in the data, the preprocessing algorithm  
 59 sequentially estimates and concatenates the counterfactual states under different sensitive  
 60 attribute values with the observed state at each time point into a new state vector. The  
 61 preprocessed data can then be directly used by existing RL algorithms for policy learning, and  
 62 the learned policy will be counterfactually fair up to finite-sample estimation accuracy. Second,  
 63 PyCFRL provides a platform for assessing RL policies based on CF. After passing in any policy  
 64 and a data trajectory from the environment of interest, users can estimate the value and  
 65 counterfactual unfairness level achieved by the policy in the environment of interest.

## 66 High-level Design

67 The PyCFRL library is composed of 5 major modules. The functionalities of the modules are  
 68 summarized in the table below.

Module	Functionalities
reader	Implements functions that read tabular trajectory data from either a .csv file or a pandas.DataFrame into an array format required by PyCFRL. Also implements functions that export trajectory data to either a .csv file or a pandas.DataFrame.
preprocessor	Implements the data preprocessing algorithm introduced in Wang et al. (2025).
agents	Implements an FQI algorithm (Riedmiller, 2005), which learns RL policies and makes decisions based on the learned policy. Users can also pass a preprocessor to the FQI; in this case, the FQI will be able to take in unprocessed trajectories, internally preprocess the input trajectories, and directly output counterfactually fair policies.
environment	Implements a synthetic environment that produces synthetic data as well as a simulated environment that estimates and simulates the transition dynamics of the unknown environment underlying some real-world RL trajectory data. Also implements functions for sampling trajectories from the synthetic and simulated environments.
evaluation	Implements functions that evaluate the value and counterfactual unfairness level of a policy. Depending on the user's needs, the evaluation can be done either in a synthetic environment or in a simulated environment.

69 A general PyCFRL workflow is as follows: First, simulate trajectories using environment or read

70 in trajectories using reader. Then, train a preprocessor using preprocessor and preprocess the  
71 training trajectory data. After that, pass the preprocessed trajectories into the FQI algorithm in  
72 agents to learn a counterfactually fair policy. Finally, use functions in evaluation to evaluate  
73 the value and counterfactual unfairness level of the trained policy.

74 In addition, PyCFRL also provides tools to check for potential non-convergence that may arise  
75 during the training of neural networks, FQI, or fitted-Q evaluation (FQE). More discussions  
76 about the sources, checks, and fixes of non-convergence in PyCFRL can be found in the  
77 “Common Issues” section of the documentation.

## 78 Data Example

79 We provide a data example showing how PyCFRL learns counterfactually fair policies from real-  
80 world trajectory data with unknown underlying transition dynamics. The example demonstrates  
81 policy learning and evaluation of both value and unfairness levels. This represents just  
82 one of many possible workflows. PyCFRL can also generate synthetic trajectory data and  
83 evaluate custom preprocessing methods. See the “Example Workflows” documentation for  
84 more examples.

85 We record the computing times of different workflows under different combinations of the  
86 number of individuals ( $N$ ) and the length of horizons ( $T$ ) in the “Computing Times” section  
87 of the PyCFRL documentation. For example, under  $N = 500$  and  $T = 10$ , the workflow  
88 presented in this data example (“real data workflow” in the documentation) ran for 378.6  
89 seconds on average in our computing environment.

### 90 Load Data

91 In this demonstration, we use an offline trajectory generated from a SyntheticEnvironment  
92 following some pre-specified transition rules. Although the data is actually synthesized, we  
93 treat it as if it is from some unknown environment for pedagogical convenience.

94 The trajectory contains 500 individuals (i.e.,  $N = 500$ ) and 10 transitions (i.e.,  $T = 10$ ). The  
95 sensitive attribute variable and the state variable are both univariate. The sensitive attribute is  
96 binary (0 or 1). The actions are also binary (0 or 1) and are sampled using a behavior policy  
97 that selects 0 or 1 randomly with equal probability. The trajectory is stored in a tabular format  
98 in a .csv file. We use read\_trajectory\_from\_csv() to load the trajectory from the .csv  
99 format into the array format required by PyCFRL.

```
zs, states, actions, rewards, ids = read_trajectory_from_csv(  
    path='../data/sample_data_large_uni.csv', z_labels=['z1'],  
    state_labels=['state1'], action_label='action', reward_label='reward',  
    id_label='ID', T=10)
```

100 We then split the trajectory data into a training set (80%) and a testing set (20%) using  
101 scikit-learn’s train\_test\_split(). The training set is used to train the counterfactually fair  
102 policy, while the testing set is used to evaluate the value and counterfactual unfairness level  
103 achieved by the policy.

```
(zs_train, zs_test, states_train, states_test,  
    actions_train, actions_test, rewards_train, rewards_test  
    ) = train_test_split(zs, states, actions, rewards, test_size=0.2)
```

### 104 Train Preprocessor & Preprocess Trajectories

105 We now train a SequentialPreprocessor and preprocess the trajectory. The SequentialPreprocessor  
106 ensures the learned policy is counterfactually fair by constructing new state variables that  
107 are not impacted by the sensitive attribute. Due to limited trajectory data, the data to be  
108 preprocessed will also be the data used to train the preprocessor, so we set cross\_folds=5

109 to reduce overfitting. In this case, `train_preprocessor()` will internally divide the training  
 110 data into 5 folds, and each fold is preprocessed using a model that is trained on the other 4  
 111 folds. We initialize the `SequentialPreprocessor`, and `train_preprocessor()` will take care  
 112 of both preprocessor training and trajectory preprocessing.

```
sp = SequentialPreprocessor(z_space=[[0], [1]], num_actions=2, cross_folds=5,
                           mode='single', reg_model='nn')
states_tilde, rewards_tilde = sp.train_preprocessor(
    zs=zs_train, xs=states_train, actions=actions_train, rewards=rewards_train)
```

### 113 Counterfactually Fair Policy Learning

114 Next, we train a counterfactually fair policy using the preprocessed data and FQI with `sp` as its  
 115 internal preprocessor. By default, the input data will first be preprocessed by `sp` before being  
 116 used for policy learning. However, since the training data `states_tilde` and `rewards_tilde`  
 117 are already preprocessed in our case, we set `preprocess=False` during training so that the  
 118 input trajectory will not be preprocessed again by the internal preprocessor (i.e., `sp`).

```
agent = FQI(num_actions=2, model_type='nn', preprocessor=sp)
agent.train(zs=zs_train, xs=states_tilde, actions=actions_train,
           rewards=rewards_tilde, max_iter=100, preprocess=False)
```

### 119 SimulatedEnvironment Training

120 Before moving on to the evaluation stage, there is one more step: We need to train a  
 121 `SimulatedEnvironment` that mimics the transition rules of the true environment that generated  
 122 the training trajectory, which will be used by the evaluation functions via Monte Carlo. To  
 123 do so, we initialize a `SimulatedEnvironment` and train it on the whole trajectory data (i.e.,  
 124 training set and testing set combined).

```
env = SimulatedEnvironment(num_actions=2, state_model_type='nn',
                          reward_model_type='nn')
env.fit(zs=zs, states=states, actions=actions, rewards=rewards)
```

### 125 Value and Counterfactual Unfairness Level Evaluation

126 We now use `evaluate_value_through_fqe()` and `evaluate_fairness_through_model()` to  
 127 estimate the value and counterfactual unfairness level achieved by the trained policy when  
 128 interacting with the environment of interest, respectively. The counterfactual unfairness level  
 129 is represented by a metric from 0 to 1, with 0 representing perfect fairness and 1 indicating  
 130 complete unfairness. We use the testing set for evaluation.

```
value = evaluate_reward_through_fqe(zs=zs_test, states=states_test,
                                   actions=actions_test, rewards=rewards_test, policy=agent, model_type='nn')
cf_metric = evaluate_fairness_through_model(env=env, zs=zs_test, states=states_test,
                                           actions=actions_test, policy=agent)
```

131 The estimated value is 7.358 and the CF metric is 0.042, which indicates our policy is close to  
 132 being perfectly counterfactually fair. Indeed, the CF metric should be exactly 0 if we know the  
 133 true dynamics of the environment of interest; the reason why it is not exactly 0 here is that  
 134 we need to estimate the dynamics of the environment of interest during preprocessing, which  
 135 can introduce finite-sample errors.

### 136 Comparisons against Baseline Methods

137 We can compare the sequential data preprocessing method in PyCFRL against a few baselines:  
 138 “Random”, which selects each action randomly with equal probability; “Full”, which uses all  
 139 variables, including the sensitive attribute, for policy learning; and “Unaware”, which uses all  
 140 variables except the sensitive attribute for policy learning. We implemented these baselines

and evaluated their values and counterfactual unfairness levels as part of the code example of the “Assessing Policies Using Real Data” workflow in the “[Example Workflows](#)” section of the PyCFRL documentation. We summarize below the values and CF metrics calculated in this code example, where “Ours” stands for outputs from the SequentialPreprocessor.

	Random	Full	Unaware	Ours
Value	−1.444	8.606	8.588	7.358
Counterfactual Unfairness Level	0	0.407	0.446	0.042

By definition, the “Random” baseline always achieves perfect CF. On the other hand, “Ours” resulted in much fairer policies than “Full” and “Unaware”, which suggests that the SequentialPreprocessor can effectively control counterfactual unfairness. Nevertheless, as a trade-off for better CF, “Ours” achieved a lower value than “Full” and “Unaware”.

## Conclusions

PyCFRL is a Python library that enables counterfactually fair reinforcement learning through data preprocessing. It also provides tools to calculate the value and unfairness level of a given policy. To our knowledge, it is the first library to address CF problems in the context of RL. The practical utility of PyCFRL can be further improved via extensions. First, the current PyCFRL implementation requires every individual in the offline dataset to have the same number of time steps. Extending the library to accommodate variable-length episodes can improve its flexibility and usefulness. Second, PyCFRL can further combine the preprocessor with popular offline RL algorithm libraries such as `d3rlpy` ([Seno & Imai, 2022](#)), or connect the evaluation functions with established RL environment libraries such as `gym` ([Towers et al., 2024](#)). Third, generalization to non-additive counterfactual states reconstruction can make PyCFRL more versatile. We leave these extensions to future updates.

## Acknowledgements

Jianhan Zhang and Jitao Wang contributed equally to this work. The authors declare no conflicts of interest.

## References

- Bellamy, R. K. E., Dey, K., Hind, M., Hoffman, S. C., Houde, S., Kannan, K., Lohia, P., Martino, J., Mehta, S., Mojsilovic, A., Nagar, S., Ramamurthy, K. N., Richards, J., Saha, D., Sattigeri, P., Singh, M., Varshney, K. R., & Zhang, Y. (2018). *AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias*.
- D’Amour, A., Srinivasan, H., Atwood, J., Baljekar, P., Sculley, D., & Halpern, Y. (2020). Fairness is not static: Deeper understanding of long term fairness via simulation studies. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 525–534. <https://doi.org/10.1145/3351095.3372878>
- Kusner, M. J., Loftus, J. R., Russell, C., & Silva, R. (2018). *Counterfactual Fairness*. <https://arxiv.org/abs/1703.06856>
- Piette, J. D., Thomas, L., Newman, S., Marinec, N., Krauss, J., Chen, J., Wu, Z., & Bohnert, A. S. B. (2023). An automatically adaptive digital health intervention to decrease opioid-related risk while conserving counselor time: Quantitative analysis of treatment decisions based on artificial intelligence and patient-reported risk measures. *Journal of Medical Internet Research*, 25, e44165. <https://doi.org/10.2196/44165>

- 180 Riedmiller, M. (2005). Neural fitted Q iteration – first experiences with a data efficient neural  
181 reinforcement learning method. In J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, & L.  
182 Torgo (Eds.), *Machine learning: ECML 2005* (pp. 317–328). Springer Berlin Heidelberg.  
183 ISBN: 978-3-540-31692-3
- 184 Seno, T., & Imai, M. (2022). d3rlpy: An offline deep reinforcement learning library. *Journal of*  
185 *Machine Learning Research*, 23(315), 1–20.
- 186 Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M.,  
187 Kallinteris, A., Krimmel, M., KG, A., & others. (2024). Gymnasium: A standard interface  
188 for reinforcement learning environments. *arXiv Preprint arXiv:2407.17032*.
- 189 Wang, J., Shi, C., Piette, J. D., Loftus, J. R., Zeng, D., & Wu, Z. (2025). *Counterfactually fair*  
190 *reinforcement learning via sequential data preprocessing*. <https://arxiv.org/abs/2501.06366>
- 191 Weerts, H., Dudík, M., Edgar, R., Jalali, A., Lutz, R., & Madaio, M. (2023). Fairlearn:  
192 Assessing and improving fairness of AI systems. In *Journal of Machine Learning Research*  
193 (No. 257; Vol. 24, pp. 1–8).

DRAFT