Fairness Project

In this project, we focused on addressing fairness in ML within the context of banking systems. More specifically, our aim was to measure how different fairness algorithms affect performance of various machine learning models. The task itself involves analyzing a marketing dataset from a financial institution to predict if a client will subscribe to a term deposit. This problem is a binary classification problem as the target variable illustrates the subscription status.

Discussion of related works

Addressing the issue of fairness in machine learning, particularly banking systems, has always made headlines and drawn attention through time. Research has been conducted exploring various bias mitigation techniques which include the same preprocessing methods that ensure equitable outcomes across demographic groups. Research conducted by Castelnovo et al. (2021) introduced the BeFair toolkit that allows us to identify and mitigate biases in banking sector models. The findings indicate that models that were trained without fairness constraints may show biases in predictions. Moreover, a survey conducted by Le Quy et al. (2022) featured the usage of datasets for fairness-aware machine learning, highlighting the importance of diverse and realistic datasets in evaluating fairness interventions. In addition, Borna B. (2023) evaluates how Disparate Impact Remover affects Light GBM model.

EDA and data preprocessing:

Data Loading and Exploration:

Our first step was to load and explore the data to understand its structure, identify any issues, and prepare it for analysis and model training.

We worked with the Bank Marketing dataset from the AIF360 library, which was formatted as a BinaryLabelDataset (we loaded the dataset directly from AIF360 library). The protected attribute for our study was age, with individuals aged 25 and above considered the privileged group (in the documentary, it was 25<=age<=60). During the initial data conversion process, any missing entries (10700 entries) were automatically removed, leaving us with 30,488 samples to analyze.

Dataset Overview:
The dataset contained 58 features that captured a mix of demographic, economic, and campaign-related attributes. We started by checking the data types and completeness, confirming that no missing values remained. All data types were float and we had no null values. However, we found 40 duplicate rows, which were removed to ensure the dataset's integrity, reducing the sample size to 30,448.

Feature Analysis and Correlations:
We explored the unique values of each feature to understand their variability. Most of the columns were consisted of 2 unique values since they were one-hot-encoded. Only one feature, "duration", had so many unique values: 1,441 unique values. Here is the columns with more than 2 unique values:

1	duration	1441
2	campaign	41
3	pdays	26
4	previous	8
5	emp.var.rate	10
6	cons.price.idx	26
7	cons.conf.idx	26
8	euribor3m	314
9	nr.employed	11

To uncover patterns in the data, we performed a correlation analysis between the features and the target variable (y). Since correlations were not too high, there was no risk of data leakage from features to target value, so we did not remove any columns based on this correlation. Here are the results of the most correlated columns with target column:

```
# Correlations
print(df.corr()['y'].sort_values(ascending=False))
```

```
y                              1.000000
duration                       0.393467
poutcome=success               0.319868
previous                       0.227922
month=mar                      0.146279
contact=cellular               0.143918
```

In addition, we checked correlation of features among themselves and flagged highly correlated (>0.9) pairs of features as candidates for removal to avoid redundancy and multicollinearity. Highly correlated features were:
Columns with very high correlation:
['euribor3m', 'nr.employed', 'default=yes', 'housing=yes', 'loan=yes', 'contact=telephone', 'poutcome=success']

After this, we checked correlations of these columns separately:

```
correlation_matrix['euribor3m'].sort_values(ascending=False).head(5)
```

```
euribor3m               1.000000
emp.var.rate            0.969395
nr.employed             0.944859
cons.price.idx          0.667212
poutcome=nonexistent    0.470611
Name: euribor3m, dtype: float64
```

```
correlation_matrix['nr.employed'].sort_values(ascending=False).head(5)
```

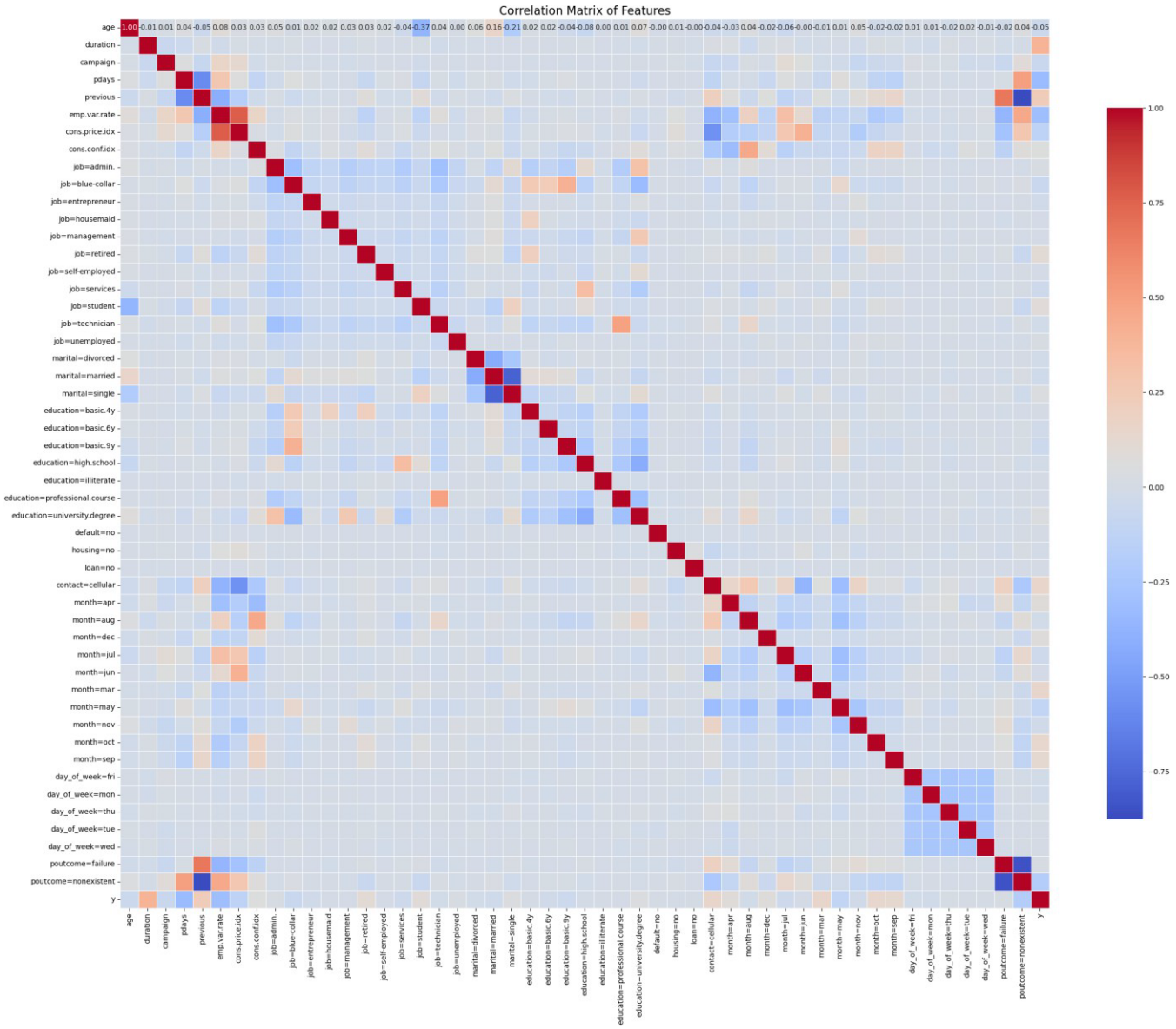
```
nr.employed       1.000000
euribor3m         0.944859
emp.var.rate      0.900335
cons.price.idx    0.488719
previous          0.488554
Name: nr.employed, dtype: float64
```

Here, correlations are very high. We decided to remove them.

```
correlation_matrix['default=yes'].sort_values(ascending=False).head(5)
```

```
default=no                        1.000000
default=yes                       1.000000
day_of_week=tue                   0.020149
job=unemployed                    0.019951
education=professional.course     0.014937
Name: default=yes, dtype: float64
```

In this case, ‘default’ was one-hot-encoded but both ‘yes’ and ‘no’ were there. However, to prevent multicollinearity we removed one of them. This case was the same for other one-hot-encoded features, too. After removing all these highly correlated features, we are left with 51 columns. Finally, to check correlation, we again checked correlation matrix:



Here, we realized that “poutcome=nonexistent” and “previous” columns are also highly correlated (darker color). Then we checked value_counts:

```
df['poutcome=nonexistent'].value_counts()
```

```
poutcome=nonexistent
1.0    25796
0.0     4652
Name: count, dtype: int64
```

```
df['previous'].value_counts()
```

```
previous
0.0    25796
1.0     3752
2.0      633
3.0      190
```

Here, poutcome=1 means previous = 0, so we remove previous column since poutcome already contains information related to previous.

Outcome
After preprocessing, we were left with a clean dataset containing 30,448 samples and 50 features. This refined dataset, saved as preprocessed_bank.csv, is ready for use in model training, evaluation, and fairness analysis.

Modeling

Describe your modeling steps. Which algorithm(s) did you apply and why? What design decisions did you make? How did you evaluate your model? Have you analyzed the errors? Insert any figures.

Introduction:
Our focus fell on 3 ML algorithms Logistic Regression, Random Forest, and Hist Gradient Boosting Classifier. These models were chosen because they offer a mix of simplicity, robustness, and strong predictive capabilities. To reduce biases in our data, we applied two fairness techniques: Disparate Impact Remover (DIR) and Reweighing. We specifically chose pre- processing techniques since these techniques transform the dataset beforehand and we wonder in this case how performance of the model is affected. We then proceeded to compare the models' performance and fairness across different scenarios.

Reasoning behind Algorithms:

1. **Logistic Regression (LR):**
    - Logistic Regression is simple, easy to interpret, and serves as a strong starting point for understanding fairness adjustments.
2. **Random Forest (RF):**
    - Random Forest was chosen because it is robust and can handle complex patterns in the data.
    - We used Grid Search to fine-tune its parameters to maximize its performance.
3. **HistGradientBoostingClassifier (HGB):**
    - This is a powerful model that performs well on structured data. We included it to test how fairness adjustments affect high-performing algorithms.

Steps:

1. **Data Preparation:**
    - We load the dataset as "dataframe" and Binary label dataset since we work with AIF360 library and this library requries dataset in this format. Protected attribute is 'age': if age>=25, this group is considered as privileged group whereas age<25 is unprivileged.
    - Before any prediction, we evaluate fairness of our original dataset to see if the dataset is fair or not. After evaluation, we split the dataset into 70% training data and 30% testing data to evaluate the models fairly.
    - To address biases in the training data, we applied two methods:
        - **DIR:** Adjusted feature values to make them fairer at different levels of intensity (repair levels from 0.1 to 1.0).

- **Reweighing:** Adjusted the importance (weights) of samples in the training data based on their group membership.

2. **Tuning Parameters:**

   - For Random Forest, we ran a Grid Search to find the best combination of parameters, like the number of trees and the maximum depth of each tree, ensuring the model performed its best. Best model parameters are:
     Best Random Forest Params: {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}
   - For Logistic Regression, we choose max iteration 2000 (we tried several iterations and if we chose less iterations, we got warning of non-convergence)
   - For Hist Boost Classifier, we use default parameters.

Note: in all cases, random state is set to 42 to ensure reusability.

3. **Evaluation Metrics:**
   - We used metrics like Accuracy – to measure overall corectness of predictions, as well as F1 Score to balance precision for imbalances datasets.
   - To evaluate fairness, we relied on:
     - **Disparate Impact (DI):** Compares the ratio of favorable outcomes between unprivileged and privileged groups. (ideal value = 1)
     - **Statistical Parity Difference (SPD):** Measures the difference in favorable outcomes between these groups. (ideal value = 0)

Experiments:

First experiment was to analyze fairness metrics on the original dataset. We evaluated fairness in both ways: with library (aif360, method: BinaryLabelDatasetMetric) and manually calculating with the help of numpy. In both cases, we got the same results:

Initial Dataset Fairness Metrics:
- Disparate Impact: 1.86
- Statistical Parity Difference: 0.11

Ideal value of Disparate Impact is 1; the result we got, 1,86 is too high and shows how unfair our data is. On the other hand, ideal value for SPD is 0; this results is not too much but still shows some unfairness. Next, we fit the original dataset to all three models and measure their performance metrics: Accuracy and F1 score. In addition, we evaluate fairness metrics based on the predictions of these models. Here are the results:

Logistic Regression:

- Accuracy: 89.75%, F1 Score: 50.42%
- Disparate Impact: 1.857, SPD: 0.064

Random Forest:

- Accuracy: 90.22%, F1 Score: 54.97%
- Disparate Impact: 1.488, SPD: 0.042

HistGradientBoostingClassifier

- Accuracy: 90.60%, F1 Score: 60.43%
- Disparate Impact: 1.451, SPD: 0.048

Next, we transformed the dataset with DisparateImpactRemover() where we set repair level to 0.1 (for starting). We transformed the whole dataset (differently from rew, where we only transform train dataset) as mentioned in https://www.kaggle.com/code/mohammadbolandraftar/bias-mitigation-disparate-impact-remover-aif360# . Next step was to evaluate all models on this transformed dataset:

Logistic Regression:

- Accuracy: 89.79%, F1 Score: 50.77%
- Disparate Impact: 1.837, SPD: 0.064

Random Forest:

- Accuracy: 90.37%, F1 Score: 56.31%
- Disparate Impact: 1.606, SPD: 0.054

HistGradientBoostingClassifier

- Accuracy: 90.51%, F1 Score: 59.84%
- Disparate Impact: 1.582, SPD: 0.061

Finally, we transformed the train dataset with Reweighing(). We converted transformed dataset to pandas dataframe and stored weights in “reweighed_dataframe_train” to use when we configure the models. Next, we again evaluated all the models. (This time, we gave additinal paramter “sample_weight” to each model, otherwise the results would not change). Here are the results:

Logistic Regression:

- Accuracy: 89.84%, F1 Score: 50.95
- Disparate Impact: 1.174, SPD: 0.0134

Random Forest:

- Accuracy: 90.27%, F1 Score: 55.44%
- Disparate Impact: 1.197, SPD: 0.017

HistGradientBoostingClassifier

- Accuracy: 90.54%, F1 Score: 59.74%
- Disparate Impact: 0.967, SPD: -0.004

In addition, we utilized MLflow for logging and tracking model experiments. The initial advantage was the ability to manage experiments in a systematic flow and record key parameters for later comparison. Nevertheless, its integration with fairness-specific libraries like AIF360 required additional customization. It helped us to keep track of metrics for each scenario separately, making comparison easier and more comfortable.

We were also able to include visualizations that included heatmaps for correlation analysis. Furthermore, bar plots of feature distribution and tables summarizing model performance and fairness metrics were able to create a clearer image and understand the impact of each of the fairness interventions on the dataset and models.

Discussion of results:

How do you interpret the results of the project? Discuss the key points. (100-200 words)
As a result of our experiments, first of all, we realized that the best model was Hist Boost Classifier with the highest accuracy and F1 score (0.905966, 0.604330). Additionally, the fairest model was also Hist Boost Classifier. We put the comparison table here to discuss the details (DIR values are with the best repair level):

Logistic Regression

Scenario	Accuracy	F1	Disparate Impact	Statistical Parity Difference
Original	0.8975	0.5042	1.8568	0.0644
DIR	0.8993	0.5031	2.4669	0.1069
Reweighing	0.8984	0.5095	1.1744	0.0135

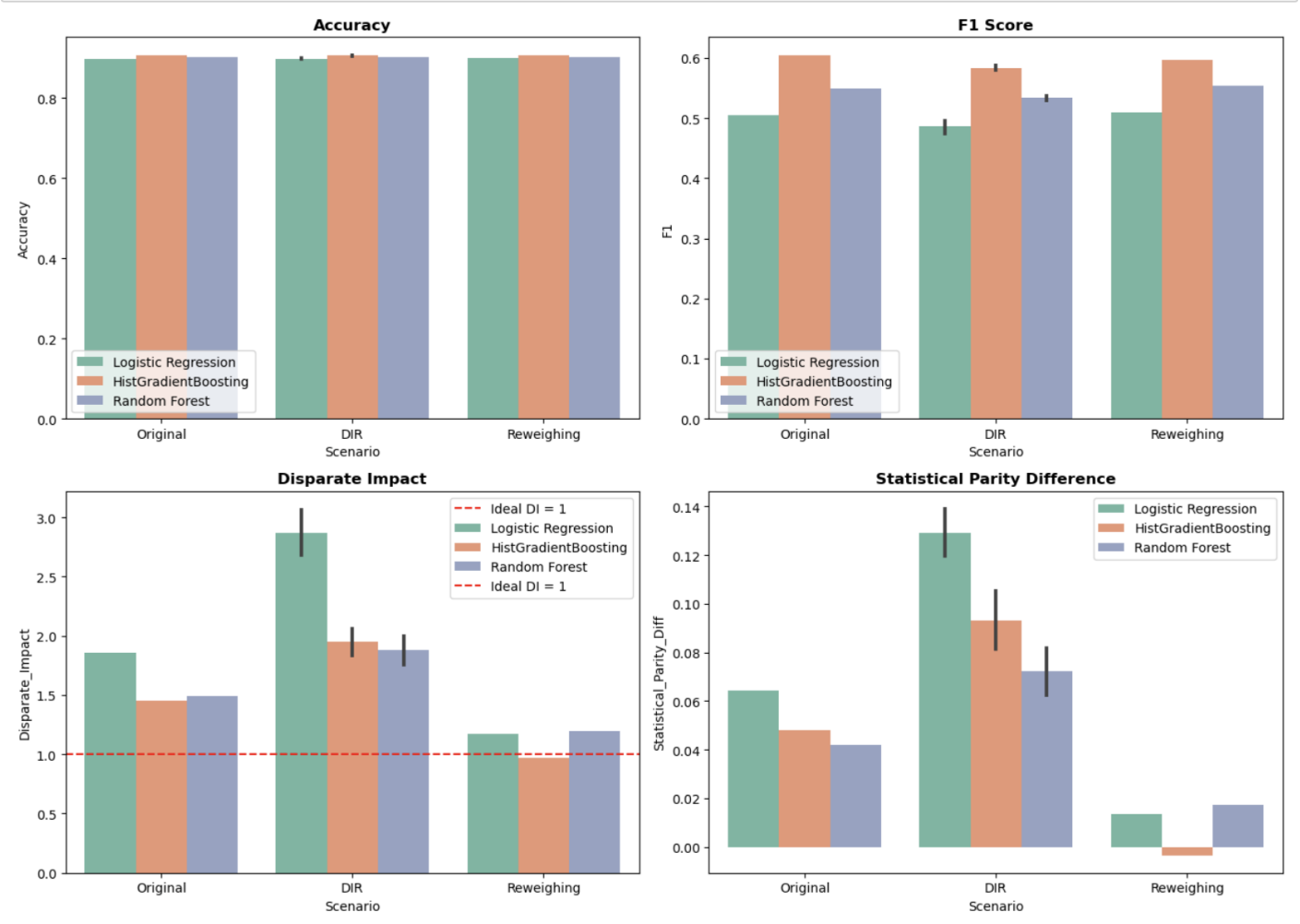
Random Forest

Scenario	Accuracy	F1	Disparate Impact	Statistical Parity Difference
Original	0.9022	0.5497	1.4876	0.0421
DIR	0.9002	0.5232	1.5936	0.0481
Reweighing	0.9027	0.5544	1.1969	0.0174

HistGradientBoosting

Scenario	Accuracy	F1	Disparate Impact	Statistical Parity Difference
Original	0.9060	0.6043	1.4507	0.0481
DIR	0.9062	0.5810	1.5260	0.0503
Reweighing	0.9054	0.5974	0.9666	-0.0035

We see that DIR affected our fairness metrics negatively in all scenarios: while it was expected to bring DI and SPD closer to 1 and 0 accordingly, it did the opposite. However, it did not affect the performance of the model a lot. On the other hand, Reweighing technique showed the best performance in all cases, specifically if we check Hist Gradient Boosting: originally, fairness metrics for DI was 1.45 and SPD 0.048, however, after we applied mitigator, it became almost perfectly fair (0.96 and 0, ideal values are 1 and 0). In addition, it does not affect model performance a lot (just minor changes).
All in all, we can conclude that pre-processing fairness techniques do not affect models' performance negatively (even it can improve in same cases as Reweighing does with Random Forest) while it makes them fairer. For our case, Reweighing shows perfect results by improving fairness a lot. Among the models, even though RF was tuned, default Hist Gradient Boosting classifier showed the best performance in terms of Fairness and Model Performance. We put some visualizations for comparison:



References

Castelnovo, A., Crupi, R., Del Gamba, G., Greco, G., Naseer, A., Regoli, D., & San Miguel Gonzalez, B. (2021). BeFair: Addressing Fairness in the Banking Sector. *IEEE International Conference on Big Data (Big Data)*. https://doi.org/10.1109/BigData50022.2020.9377894

Le Quy, T., Roy, A., Iosifidis, V., Zhang, W., & Ntoutsi, E. (2022). A survey on datasets for fairness -aware machine learning. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(3), e1452. https://doi.org/10.1002/widm.1452

Dataset - https://aif360.readthedocs.io/en/latest/modules/generated/aif360.datasets.BankDataset.html