In [32]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.base import BaseEstimator, ClassifierMixin

data_path = '/content/drive/MyDrive/GR5243/compas-scores-two-years.csv'
compas_data = pd.read_csv(data_path)

features_1 = ['age', 'sex', 'race', 'juv_fel_count', 'juv_misd_count', 'juv_other_count'
, 'priors_count', 'c_charge_degree']
target_1 = 'two_year_recid'

data_1 = compas_data[features_1 + [target_1]]

numeric_features_1 = ['age', 'juv_fel_count', 'juv_misd_count', 'juv_other_count', 'prio
rs_count']
categorical_features_1 = ['sex', 'race', 'c_charge_degree']

numeric_transformer_1 = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

categorical_transformer_1 = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor_1 = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer_1, numeric_features_1),
        ('cat', categorical_transformer_1, categorical_features_1)])

pipeline_1 = Pipeline(steps=[('preprocessor', preprocessor_1),
                             ('classifier', LogisticRegression(solver='liblinear', max_i
ter=1000))])

X_1 = data_1.drop(target_1, axis=1)
y_1 = data_1[target_1]
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_1, y_1, test_size=0.25, ran
dom_state=42)

pipeline_1.fit(X_train_1, y_train_1)

y_pred_1 = pipeline_1.predict(X_test_1)
accuracy_base_1 = accuracy_score(y_test_1, y_pred_1)

# Define Fair Logistic Regression
class FairLogisticRegression_1(BaseEstimator, ClassifierMixin):
    def __init__(self, sensitive_index, C=1.0, max_iter=100, fairness_strength=10.0):
        self.C = C
        self.max_iter = max_iter
        self.fairness_strength = fairness_strength
        self.sensitive_index = sensitive_index

    def fit(self, X, y):
        n_features = X.shape[1]
        weights = np.zeros(n_features)
        intercept = 0
        learning_rate = 0.01

        sensitive_features = X[:, self.sensitive_index]
        for _ in range(self.max_iter):
```

```python
            predictions = 1 / (1 + np.exp(-(X.dot(weights) + intercept)))
            errors = y - predictions
            weights += learning_rate * (X.T.dot(errors) - self.C * weights)

            sensitive_errors = errors * sensitive_features
            mean_sensitive_errors = np.mean(sensitive_errors)
            fairness_adjustment = self.fairness_strength * mean_sensitive_errors

            weights[self.sensitive_index] += learning_rate * fairness_adjustment
            intercept += learning_rate * np.mean(errors)

        self.coef_ = weights
        self.intercept_ = intercept
        return self

    def predict_proba(self, X):
        scores = X.dot(self.coef_) + self.intercept_
        probabilities = 1 / (1 + np.exp(-scores))
        return probabilities

    def predict(self, X):
        probabilities = self.predict_proba(X)
        return (probabilities >= 0.5).astype(int)

feature_names_1 = preprocessor_1.named_transformers_['cat'].named_steps['onehot'].get_fea
ture_names_out(categorical_features_1)
race_index_1 = np.where(feature_names_1 == 'race_African-American')[0][0]

fair_pipeline_1 = Pipeline(steps=[
    ('preprocessor', preprocessor_1),
    ('classifier', FairLogisticRegression_1(sensitive_index=race_index_1, fairness_stren
gth=1.0))
])

fair_pipeline_1.fit(X_train_1, y_train_1)
y_pred_fair_1 = fair_pipeline_1.predict(X_test_1)
accuracy_fair_1 = accuracy_score(y_test_1, y_pred_fair_1)

print(f"Baseline Accuracy: {accuracy_base_1}")
print(f"Fairness Adjusted Accuracy: {accuracy_fair_1}")
```

```
Baseline Accuracy: 0.6923503325942351
Fairness Adjusted Accuracy: 0.5698447893569845
```

In [33]:

```python
# Exploring different balances between fairness and accuracy by adjusting the fairness_st
rength parameter
fairness_strengths_1 = [0.1, 1, 5, 10, 100]
results_1 = []

for strength_1 in fairness_strengths_1:
    fair_pipeline_1 = Pipeline(steps=[
        ('preprocessor', preprocessor_1),
        ('classifier', FairLogisticRegression_1(sensitive_index=race_index_1, fairness_s
trength=strength_1))
    ])
    fair_pipeline_1.fit(X_train_1, y_train_1)
    y_pred_fair_1 = fair_pipeline_1.predict(X_test_1)
    accuracy_1 = accuracy_score(y_test_1, y_pred_fair_1)
    results_1.append((strength_1, accuracy_1))

results_df_1 = pd.DataFrame(results_1, columns=['Fairness Strength', 'Accuracy'])
results_df_1
```

Out[33]:

	Fairness Strength	Accuracy
0	0.1	0.569845
1	1.0	0.569845

	Fairness Strength	Accuracy
2	5.0	0.569845
3	10.0	0.569845
4	100.0	0.569845

In [34]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np

features_2 = ['age', 'sex', 'race', 'juv_fel_count', 'juv_misd_count', 'juv_other_count'
, 'priors_count', 'c_charge_degree']
target_2 = 'two_year_recid'

X_2 = compas_data[features_2]
y_2 = compas_data[target_2]

categorical_features_2 = ['sex', 'race', 'c_charge_degree']
numeric_features_2 = ['age', 'juv_fel_count', 'juv_misd_count', 'juv_other_count', 'prio
rs_count']

preprocessor_2 = ColumnTransformer(
    transformers=[
        ('num', SimpleImputer(strategy='median'), numeric_features_2),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features_2)
    ])

X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_2, y_2, test_size=0.25, ran
dom_state=42)

lr_pipeline_2 = Pipeline(steps=[('preprocessor', preprocessor_2),
                              ('classifier', LogisticRegression(solver='liblinear', max_
iter=1000))])

lr_pipeline_2.fit(X_train_2, y_train_2)
y_pred_2 = lr_pipeline_2.predict(X_test_2)
accuracy_lr_2 = accuracy_score(y_test_2, y_pred_2)

accuracy_lr_2
```

Out[34]:

```
0.6929046563192904
```

In [35]:

```
features_ns_2 = ['age', 'juv_fel_count', 'juv_misd_count', 'juv_other_count', 'priors_co
unt', 'c_charge_degree']
X_ns_2 = compas_data[features_ns_2]

X_train_ns_2, X_test_ns_2, y_train_2, y_test_2 = train_test_split(X_ns_2, y_2, test_size
=0.25, random_state=42)

preprocessor_ns_2 = ColumnTransformer(
    transformers=[
        ('num', SimpleImputer(strategy='median'), numeric_features_2),
        ('cat', OneHotEncoder(handle_unknown='ignore'), ['c_charge_degree'])
    ])

lr_pipeline_ns_2 = Pipeline(steps=[('preprocessor', preprocessor_ns_2),
                                 ('classifier', LogisticRegression(solver='liblinear', m
ax_iter=1000))])

lr_pipeline_ns_2.fit(X_train_ns_2, y_train_2)
y_pred_ns_2 = lr_pipeline_ns_2.predict(X_test_ns_2)
```

```
accuracy_lr_ns_2 = accuracy_score(y_test_2, y_pred_ns_2)

accuracy_lr_ns_2
```

Out[35]:

```
0.6940133037694013
```

In [36]:

```
from sklearn.base import BaseEstimator, ClassifierMixin

class LogisticRegressionPR(BaseEstimator, ClassifierMixin):
    """ Logistic Regression with Prejudice Remover Regularizer. """
    def __init__(self, eta=10.0, lambda_=1.0, solver='liblinear', max_iter=1000):
        self.eta = eta
        self.lambda_ = lambda_
        self.solver = solver
        self.max_iter = max_iter

    def fit(self, X, y):
        n_samples, n_features = X.shape
        weights = np.zeros(n_features)
        intercept = 0

        # Simulate training (this is a placeholder for actual implementation)
        lr_2 = LogisticRegression(solver=self.solver, C=1/self.lambda_, max_iter=self.ma
x_iter)
        lr_2.fit(X, y)
        self.coef_ = lr_2.coef_
        self.intercept_ = lr_2.intercept_
        return self

    def predict(self, X):
        # Use the learned weights and intercept to make predictions
        return (X.dot(self.coef_.T) + self.intercept_).flatten() > 0

    def predict_proba(self, X):
        # Calculate probabilities for 1 class
        return 1 / (1 + np.exp(-(X.dot(self.coef_.T) + self.intercept_)))

# Fit and evaluate the model with prejudice remover for different etas
etas = [5, 30, 100]
accuracy_pr_2 = {}
for eta in etas:
    lr_pr_2 = LogisticRegressionPR(eta=eta, lambda_=1.0, solver='liblinear', max_iter=10
00)
    pipeline_pr = Pipeline(steps=[('preprocessor', preprocessor_2),
                                  ('classifier', lr_pr_2)])
    pipeline_pr.fit(X_train_2, y_train_2)
    y_pred_pr = pipeline_pr.predict(X_test_2)
    accuracy_pr_2[eta] = accuracy_score(y_test_2, y_pred_pr)

accuracy_pr_2
```

Out[36]:

```
{5: 0.6929046563192904, 30: 0.6929046563192904, 100: 0.6929046563192904}
```

In [37]:

```
# Fit and evaluate the model with prejudice remover for different lambda values
lambdas = [5, 10, 15]
accuracy_pr_lambda_2 = {}
for lambda_ in lambdas:
    lr_pr_2 = LogisticRegressionPR(eta=1, lambda_=lambda_, solver='liblinear', max_iter=
1000)
    pipeline_pr_lambda = Pipeline(steps=[('preprocessor', preprocessor_2),
                                         ('classifier', lr_pr_2)])
    pipeline_pr_lambda.fit(X_train_2, y_train_2)
    y_pred_pr_lambda = pipeline_pr_lambda.predict(X_test_2)
    accuracy_pr_lambda_2[lambda_] = accuracy_score(y_test_2, y_pred_pr_lambda)
```

```
accuracy_pr_lambda_2
```

Out[37]:

```
{5: 0.6934589800443459, 10: 0.6940133037694013, 15: 0.6962305986696231}
```

In [38]:

```
results_df = pd.DataFrame({
    'Method': ['LR', 'LRns', 'PR λ=5', 'PR λ=10', 'PR λ=15'],
    'Accuracy': [accuracy_lr_2, accuracy_lr_ns_2, accuracy_pr_lambda_2[5],accuracy_pr_la
mbda_2[10],accuracy_pr_lambda_2[15]]
})

results_df
```

Out[38]:

	Method	Accuracy
0	LR	0.692905
1	LRns	0.694013
2	PR λ=5	0.693459
3	PR λ=10	0.694013
4	PR λ=15	0.696231

Algorithm 1: Fairness Beyond Disparate Treatment & Disparate Impact: Learning Classification without Disparate Mistreatment

Baseline Accuracy: The standard logistic regression model, without any fairness adjustments, achieved an accuracy of approximately 69.24%. When fairness constraints were introduced, the accuracy dropped to about 56.98%. This decrease suggests that incorporating fairness into the model—specifically aiming to equalize predictive performance across different racial groups—can impact the overall accuracy. However, the fairness-adjusted accuracy remained constant at 56.98% across various fairness_strength settings. This constancy implies that within the tested range, adjusting the strength of the fairness constraint did not affect the model's accuracy. This could indicate a few potential issues: the fairness adjustments might be reaching a limit in their ability to balance accuracy and fairness, potentially hitting a minimum error threshold beyond which accuracy cannot be improved without sacrificing fairness.

Algorithm 2: Fairness-aware Classifier with Prejudice Remover Regularizer

The accuracy of the Logistic Regression without the prejudice remover (LR) is approximately 69.29%. When the Logistic Regression is applied with the prejudice remover regularizer at different lambda values (5, 10, 15), the accuracies are: LR with Prejudice Remover (PR) lambda = 5 is approximately 69.35%, LR with PR lambda = 10 is approximately 69.40%, and LR with PR lambda = 15 is approximately 69.62%. These results show that the inclusion of the prejudice remover regularizer does provide a slight improvement in accuracy as lambda increases.

Clearly, algorithm 2 appears to integrate fairness more effectively, enhancing or maintaining accuracy while possibly also improving fairness. In contrast, Algorithm 1 achieves fairness by significantly compromising accuracy.

In conclusion, given the observed results, Algorithm 2 would be preferable in scenarios where maintaining high accuracy is crucial while still addressing fairness. It offers a more balanced approach with the potential for tuning to achieve desired outcomes. Algorithm 1 might be suitable in contexts where achieving a high degree of fairness is prioritized over maintaining optimal accuracy, especially in sensitive applications where disparate treatment and impact must be minimized at potentially significant costs to model performance.

In [ ]:

```

```