```
!pip install -q --upgrade --force-reinstall \
numpy==1.26.4 \
scipy==1.11.4 \
scikit-learn==1.4.2 \
pandas==2.1.4 \
matplotlib==3.8.4 \
seaborn==0.13.2 \
shap==0.44.1 \
lime==0.2.0.1 \
xgboost==2.0.3 \
imbalanced-learn==0.12.2
```

```
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behav
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 2.1.4 which is incompatible.
giddy 2.3.8 requires scipy>=1.12, but you have scipy 1.11.4 which is incompatible.
rasterio 1.5.0 requires numpy>=2, but you have numpy 1.26.4 which is incompatible.
pytensor 2.36.3 requires numpy>=2.0, but you have numpy 1.26.4 which is incompatible.
jaxlib 0.7.2 requires numpy>=2.0, but you have numpy 1.26.4 which is incompatible.
jaxlib 0.7.2 requires scipy>=1.13, but you have scipy 1.11.4 which is incompatible.
opencv-contrib-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 1.26.4 which is
libpysal 4.14.1 requires scipy>=1.12.0, but you have scipy 1.11.4 which is incompatible.
mapclassify 2.10.0 requires scipy>=1.12, but you have scipy 1.11.4 which is incompatible.
xarray 2025.12.0 requires pandas>=2.2, but you have pandas 2.1.4 which is incompatible.
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 1.26.4 which is incompa
mizani 0.13.5 requires pandas>=2.2.0, but you have pandas 2.1.4 which is incompatible.
access 1.1.10.post3 requires scipy>=1.14.1, but you have scipy 1.11.4 which is incompatible.
tobler 0.13.0 requires numpy>=2.0, but you have numpy 1.26.4 which is incompatible.
tobler 0.13.0 requires pandas>=2.2, but you have pandas 2.1.4 which is incompatible.
tobler 0.13.0 requires scipy>=1.13, but you have scipy 1.11.4 which is incompatible.
opencv-python-headless 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 1.26.4 which i
spopt 0.7.0 requires scipy>=1.12.0, but you have scipy 1.11.4 which is incompatible.
umap-learn 0.5.11 requires scikit-learn>=1.6, but you have scikit-learn 1.4.2 which is incompatible.
esda 2.8.1 requires scipy>=1.12, but you have scipy 1.11.4 which is incompatible.
plotnine 0.14.5 requires pandas>=2.2.0, but you have pandas 2.1.4 which is incompatible.
inequality 1.1.2 requires scipy>=1.12, but you have scipy 1.11.4 which is incompatible.
gradio 5.50.0 requires pillow<12.0,>=8.0, but you have pillow 12.1.0 which is incompatible.
langchain-core 1.2.7 requires packaging<26.0.0,>=23.2.0, but you have packaging 26.0 which is incompatible.
hdbscan 0.8.41 requires scikit-learn>=1.6, but you have scikit-learn 1.4.2 which is incompatible.
tsfresh 0.21.1 requires scipy>=1.14.0; python_version >= "3.10", but you have scipy 1.11.4 which is incompatible.
jax 0.7.2 requires numpy>=2.0, but you have numpy 1.26.4 which is incompatible.
jax 0.7.2 requires scipy>=1.13, but you have scipy 1.11.4 which is incompatible.
```

```
import numpy as np
import sklearn
import shap

print("NumPy:", np.__version__)
print("Scikit-learn:", sklearn.__version__)
print("SHAP:", shap.__version__)
```

```
NumPy: 1.26.4
Scikit-learn: 1.4.2
SHAP: 0.44.1
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

import shap
from lime.lime_tabular import LimeTabularExplainer
```

```
df = pd.read_csv('student-por.csv', sep=';')
df.head()
```

| | school | sex | age | address | famsize | Pstatus | Medu | Fedu | Mjob | Fjob | ... | famrel | freetime | goout | Dalc | Walc | hea |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | GP | F | 18 | U | GT3 | A | 4 | 4 | at_home | teacher | ... | 4 | 3 | 4 | 1 | 1 | |
| **1** | GP | F | 17 | U | GT3 | T | 1 | 1 | at_home | other | ... | 5 | 3 | 3 | 1 | 1 | |
| **2** | GP | F | 15 | U | LE3 | T | 1 | 1 | at_home | other | ... | 4 | 3 | 2 | 2 | 3 | |
| **3** | GP | F | 15 | U | GT3 | T | 4 | 2 | health | services | ... | 3 | 2 | 2 | 1 | 1 | |
| **4** | GP | F | 16 | U | GT3 | T | 3 | 3 | other | other | ... | 4 | 3 | 2 | 1 | 2 | |

5 rows × 33 columns

```
# Preprocess the data
# Convert the target variable G3 to binary: pass (>=10) or fail (<10)
df['G3'] = df['G3'].apply(lambda x: 1 if x >= 10 else 0)
```

```
# Define features and target
features = ['G1', 'G2', 'absences', 'failures', 'Medu', 'Fedu', 'sex', 'school', 'Pstatus', 'famsize']
X = df[features]
y = df['G3']
```

```
# Encode categorical variables
X = pd.get_dummies(X, columns=['sex', 'school', 'Pstatus', 'famsize'], drop_first=True)
```

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from imblearn.over_sampling import SMOTE

# Apply SMOTE to handle class imbalance
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

```
from xgboost import XGBClassifier

# Model Training: Logistic Regression, Random Forest, XGBoost
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(random_state=42),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}
```

```
# Hyperparameter tuning for Random Forest and XGBoost
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None]
}
param_grid_xgb = {
    'n_estimators': [100, 200],
    'max_depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1]
}
```

```
from sklearn.model_selection import GridSearchCV

grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=5, scoring='f1')
grid_xgb = GridSearchCV(XGBClassifier(use_label_encoder=False, eval_metric='logloss'), param_grid_xgb, cv=5, scoring=
```

```
# Train models
models['Random Forest'] = grid_rf.fit(X_train_smote, y_train_smote).best_estimator_
models['XGBoost'] = grid_xgb.fit(X_train_smote, y_train_smote).best_estimator_
models['Logistic Regression'].fit(X_train_smote, y_train_smote)
```

```
LogisticRegression(max_iter=1000)
```

```
from sklearn.metrics import f1_score
```

```
# Evaluate models
results = {}
for name, model in models.items():
    y_pred = model.predict(X_test)
    results[name] = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'F1 Score': f1_score(y_test, y_pred)
    }

# Print model performance
for name, metrics in results.items():
    print(f"{name} - Accuracy: {metrics['Accuracy']:.3f}, F1 Score: {metrics['F1 Score']:.3f}")
```

```
Logistic Regression - Accuracy: 0.915, F1 Score: 0.951
Random Forest - Accuracy: 0.885, F1 Score: 0.933
XGBoost - Accuracy: 0.900, F1 Score: 0.943
```

```
# Save results to CSV
results_df = pd.DataFrame(results).T
results_df.to_csv('model_performance.csv')
```

```
!pip install 'aif360[LawSchoolGPA]'
!pip install 'aif360[Reductions]'
!pip install 'aif360[Scalers]'
!pip install 'aif360[Skewers]'
!pip install 'aif360[inFairness]'
```

Show hidden output

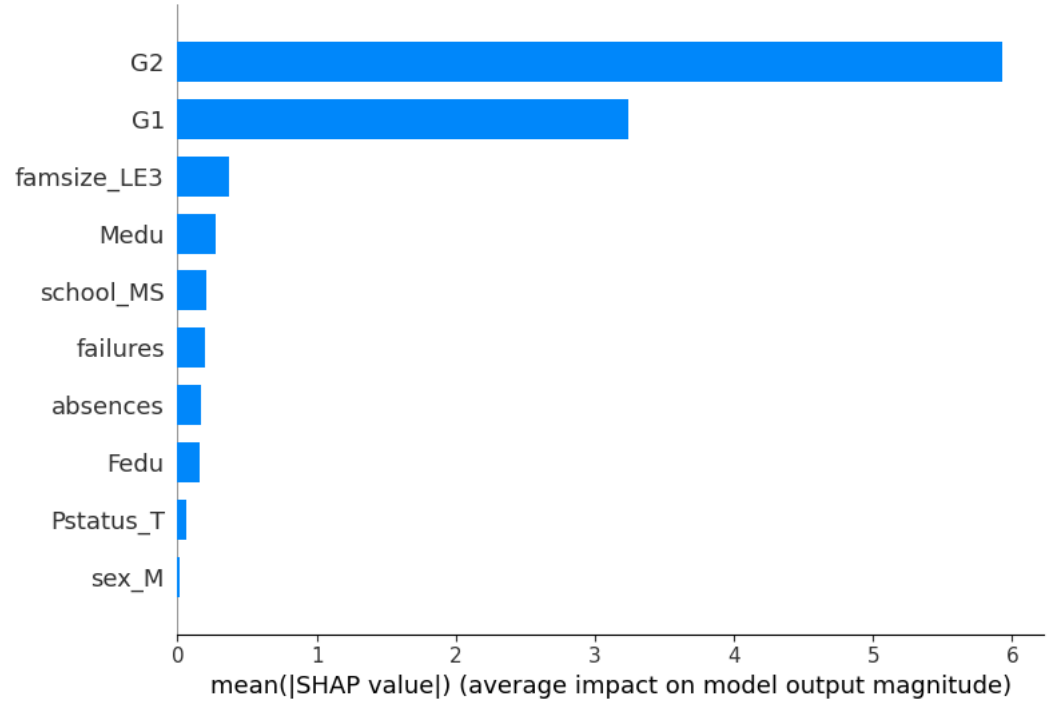
```
from aif360.datasets import BinaryLabelDataset

# Fairness Analysis with AIF360
# Define protected attributes (e.g., sex_M)
privileged_groups = [{'sex_M': 1}]
unprivileged_groups = [{'sex_M': 0}]
dataset_orig = BinaryLabelDataset(df=pd.concat([X_test, y_test], axis=1),
                                  label_names=['G3'],
                                  protected_attribute_names=['sex_M'])
```

```
from aif360.algorithms.preprocessing import Reweighing

# Apply Reweighing for fairness
rw = Reweighing(unprivileged_groups=unprivileged_groups, privileged_groups=privileged_groups)
dataset_transf = rw.fit_transform(dataset_orig)
```

```
from aif360.metrics import ClassificationMetric

# Evaluate fairness metrics

# Use the best performing model (e.g., Logistic Regression) to make predictions
best_model = models['Logistic Regression']
y_pred = best_model.predict(X_test)

# Create a BinaryLabelDataset for the predictions
# It's crucial that this dataset has the same structure as dataset_orig for comparison
dataset_pred = dataset_orig.copy()
dataset_pred.labels = y_pred.reshape(-1, 1) # Ensure shape matches expected (n_samples, 1)

metric = ClassificationMetric(dataset_orig, dataset_pred,
                              unprivileged_groups=unprivileged_groups,
                              privileged_groups=privileged_groups)
print(f"Demographic Parity Difference: {metric.statistical_parity_difference():.3f}")
print(f"Equalized Odds Difference: {metric.equal_opportunity_difference():.3f}")
```

```
Demographic Parity Difference: 0.038
Equalized Odds Difference: -0.028
```

```
# Explainability with SHAP
explainer = shap.LinearExplainer(models['Logistic Regression'], X_train_smote)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test, plot_type="bar")
```




```
import lime

# Explainability with LIME
lime_explainer = lime.lime_tabular.LimeTabularExplainer(
    X_train_smote.values,
    feature_names=X_train_smote.columns,
    class_names=['Fail', 'Pass'],
    mode='classification'
)
lime_exp = lime_explainer.explain_instance(X_test.iloc[0].values, models['Logistic Regression'].predict_proba)
lime_exp.show_in_notebook()
```

```
X does not have valid feature names, but LogisticRegression was fitted with feature names
```