Out[34]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)

10. Hyperparameter optimization

rubric={points:10}
Your tasks:

Make some attempts to optimize hyperparameters for the models you've tried and summarize your results. In at least one case you should be optimizing multiple hyperparameters for a single model. You may use `sklearn`'s methods for hyperparameter optimization or fancier Bayesian optimization methods.

- GridSearchCV
- RandomizedSearchCV
- scikit-optimize

Solution_10

Points: 10

Type your answer here, replacing this text.

In [35]:
```python
X_train_selected_df.head()
```

Out[35]:

	PAY_0	worst_payment_delay	avg_payment_amt	EDUCATION_0	EDUCATION_4
0	0.013770	-0.325570	-0.173458	0.0	0.0
1	-0.878738	-0.325570	4.185515	0.0	0.0
2	-1.771246	-1.816079	-0.517197	0.0	0.0
3	0.013770	-0.325570	-0.188086	0.0	0.0
4	0.906278	1.164940	-0.409324	0.0	0.0

In [36]:
```python
#Random Forest Hyperparameter Optimization
#With selected features
from sklearn.pipeline import make_pipeline

pipe_rf_1 = make_pipeline(
    RandomForestClassifier(class_weight="balanced", random_state=123)
)

param_distributions_rf = {
    'randomforestclassifier__n_estimators': randint(50, 500),  # Randomly se
    'randomforestclassifier__max_depth': randint(5, 50)  # Randomly select m
}

# Create and fit GridSearchCV
rd_rf = RandomizedSearchCV(pipe_rf_1,
                    param_distributions=param_distributions_rf,
                    cv=5,
                    n_jobs=-1,
                    return_train_score=True,
                    scoring = recall_scorer)

# Fit the pipeline on the original training data (not the transformed one)
rd_rf.fit(X_train_selected_df, y_train)
print("Best parameters for Random Forest:", rd_rf.best_params_)
print("Best cross-validation score for Random Forest:", rd_rf.best_score_)
```

```
Best parameters for Random Forest: {'randomforestclassifier__max_depth': 7,
'randomforestclassifier__n_estimators': 135}
Best cross-validation score for Random Forest: 0.642273235135498
```

In [37]:
```python
# Step 1: Extract the best parameters
best_params = rd_rf.best_params_

# Create a new RandomForestClassifier using the best parameters
best_rf_selected = RandomForestClassifier(
    n_estimators=best_params['randomforestclassifier__n_estimators'],
    max_depth=best_params['randomforestclassifier__max_depth'],
    class_weight="balanced",
    random_state=123
)


mean_std_scores = mean_std_cross_val_scores(best_rf_selected, X_train_select

random_forest_selected = pd.DataFrame(mean_std_scores).T
random_forest_selected.index = ['Random Forest Selected (Tuned)']
random_forest_selected
combined_scores = pd.concat([combined_scores, random_forest_selected])
combined_scores
```

Out[37]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)
Random Forest Selected (Tuned)	0.458 (+/- 0.004)	0.016 (+/- 0.000)	0.642 (+/- 0.012)	0.649 (+/- 0.003)

In [38]:

```
#Random Forest Hyperparameter Optimization
#Without selected features
from sklearn.pipeline import make_pipeline

pipe_rf = make_pipeline(
    preprocessor, RandomForestClassifier(class_weight="balanced", random_sta
)

param_distributions_rf = {
    'randomforestclassifier__n_estimators': randint(50, 500),  # Randomly se
    'randomforestclassifier__max_depth': randint(5, 50)  # Randomly select m
}

# Create and fit GridSearchCV
```

```
rd_rf = RandomizedSearchCV(pipe_rf,
                    param_distributions=param_distributions_rf,
                    cv=5,
                    n_jobs=-1,
                    return_train_score=True,
                    scoring = recall_scorer)

# Fit the pipeline on the original training data (not the transformed one)
rd_rf.fit(X_train, y_train)
print("Best parameters for Random Forest:", rd_rf.best_params_)
print("Best cross-validation score for Random Forest:", rd_rf.best_score_)
```

```
Best parameters for Random Forest: {'randomforestclassifier__max_depth': 5,
'randomforestclassifier__n_estimators': 474}
Best cross-validation score for Random Forest: 0.6318234259408856
```

In [39]:

```
# Create a new RandomForestClassifier using the best parameters
best_rf_unselected = rd_rf.best_estimator_

mean_std_scores = mean_std_cross_val_scores(best_rf_unselected, X_train, y_t

random_forest_unselected = pd.DataFrame(mean_std_scores).T
random_forest_unselected.index = ['Random Forest Unselected (Tuned)']
random_forest_unselected
combined_scores = pd.concat([combined_scores, random_forest_unselected])
combined_scores
```

Out[39]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)
Random Forest Selected (Tuned)	0.458 (+/- 0.004)	0.016 (+/- 0.000)	0.642 (+/- 0.012)	0.649 (+/- 0.003)
Random Forest Unselected (Tuned)	5.272 (+/- 0.029)	0.051 (+/- 0.000)	0.632 (+/- 0.013)	0.642 (+/- 0.002)

In [40]:
```python
#KNN Hyperparameter Optimization
#With selected features
pipe_knn_1 = make_pipeline(
    KNeighborsClassifier()
)

param_distributions_knn = {
    'kneighborsclassifier__n_neighbors': np.arange(1, 20, 2)
}

rs_knn = RandomizedSearchCV(pipe_knn_1,
                            param_distributions=param_distributions_knn,
                            n_iter=10,
                            cv=5,
                            n_jobs=-1,
                            random_state=123,
                            return_train_score=True,
                            scoring = recall_scorer)
rs_knn.fit(X_train_selected_df, y_train)

print("Best parameters for K-Nearest Neighbors:", rs_knn.best_params_)
print("Best cross-validation score for K-Nearest Neighbors:", rs_knn.best_sc
Best parameters for K-Nearest Neighbors: {'kneighborsclassifier__n_neighbor
s': 1}
Best cross-validation score for K-Nearest Neighbors: 0.37628438080977944
```

In [41]:
```
# Step 1: Extract the best parameters
best_knn_selected = rs_knn.best_estimator_

mean_std_scores = mean_std_cross_val_scores(best_knn_selected, X_train_selec

knn_selected = pd.DataFrame(mean_std_scores).T
knn_selected.index = ['KNN Selected (Tuned)']
knn_selected
combined_scores = pd.concat([combined_scores, knn_selected])
combined_scores
```

Out[41]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)
Random Forest Selected (Tuned)	0.458 (+/- 0.004)	0.016 (+/- 0.000)	0.642 (+/- 0.012)	0.649 (+/- 0.003)
Random Forest Unselected (Tuned)	5.272 (+/- 0.029)	0.051 (+/- 0.000)	0.632 (+/- 0.013)	0.642 (+/- 0.002)
KNN Selected (Tuned)	0.004 (+/- 0.001)	0.070 (+/- 0.002)	0.376 (+/- 0.027)	0.875 (+/- 0.018)

In [42]:
```
#KNN Hyperparameter Optimization
#Without selected features
pipe_knn = make_pipeline(
    preprocessor, KNeighborsClassifier()
)
```

```
param_distributions_knn = {
    'kneighborsclassifier__n_neighbors': np.arange(1, 20, 2)
}

rs_knn = RandomizedSearchCV(pipe_knn,
                            param_distributions=param_distributions_knn,
                            n_iter=10,
                            cv=5,
                            n_jobs=-1,
                            random_state=123,
                            return_train_score=True,
                            scoring = recall_scorer)
rs_knn.fit(X_train, y_train)

print("Best parameters for K-Nearest Neighbors:", rs_knn.best_params_)
print("Best cross-validation score for K-Nearest Neighbors:", rs_knn.best_sc
```

```
Best parameters for K-Nearest Neighbors: {'kneighborsclassifier__n_neighbor
s': 1}
Best cross-validation score for K-Nearest Neighbors: 0.3937690720054249
```

In [43]:

```
# Step 1: Extract the best parameters
best_knn_unselected = rs_knn.best_estimator_

mean_std_scores = mean_std_cross_val_scores(best_knn_unselected, X_train, y_

knn_unselected = pd.DataFrame(mean_std_scores).T
knn_unselected.index = ['KNN Unselected (Tuned)']
knn_unselected
combined_scores = pd.concat([combined_scores, knn_unselected])
combined_scores
```

Out[43]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)
Random Forest Selected (Tuned)	0.458 (+/- 0.004)	0.016 (+/- 0.000)	0.642 (+/- 0.012)	0.649 (+/- 0.003)
Random Forest Unselected (Tuned)	5.272 (+/- 0.029)	0.051 (+/- 0.000)	0.632 (+/- 0.013)	0.642 (+/- 0.002)
KNN Selected (Tuned)	0.004 (+/- 0.001)	0.070 (+/- 0.002)	0.376 (+/- 0.027)	0.875 (+/- 0.018)
KNN Unselected (Tuned)	0.011 (+/- 0.001)	0.050 (+/- 0.000)	0.394 (+/- 0.010)	0.998 (+/- 0.000)

In [44]:

```python
#LGBM Hyperparameter Optimization
#Unbalanced
#With selected features
pipe_lgbm_1 = Pipeline([
    ('classifier', LGBMClassifier(random_state=123))
])

param_distributions_lgbm = {
    'classifier__n_estimators': np.arange(50, 401, 10)
}

rs_lgbm = RandomizedSearchCV(pipe_lgbm_1,
                             param_distributions=param_distributions_lgbm,
                             n_iter=10,
                             cv=5,
                             n_jobs=-1,
                             random_state=123,
                             return_train_score=True,
                             scoring = recall_scorer)
rs_lgbm.fit(X_train_selected_df, y_train)

print("Best parameters for LightGBM:", rs_lgbm.best_params_)
print("Best cross-validation score for LightGBM:", rs_lgbm.best_score_)
```

```
Best parameters for LightGBM: {'classifier__n_estimators': 170}
Best cross-validation score for LightGBM: 0.3464142923133987
```

In [45]:
```
# Step 1: Extract the best parameters
best_lgbm_unbalanced_selected = rs_lgbm.best_estimator_

mean_std_scores = mean_std_cross_val_scores(best_lgbm_unbalanced_selected, X

lgbm_unbalanced_selected = pd.DataFrame(mean_std_scores).T
lgbm_unbalanced_selected.index = ['LGBM Unbalanced Selected (Tuned)']
lgbm_unbalanced_selected
combined_scores = pd.concat([combined_scores, lgbm_unbalanced_selected])
combined_scores
```

Out[45]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)
Random Forest Selected (Tuned)	0.458 (+/- 0.004)	0.016 (+/- 0.000)	0.642 (+/- 0.012)	0.649 (+/- 0.003)
Random Forest Unselected (Tuned)	5.272 (+/- 0.029)	0.051 (+/- 0.000)	0.632 (+/- 0.013)	0.642 (+/- 0.002)
KNN Selected (Tuned)	0.004 (+/- 0.001)	0.070 (+/- 0.002)	0.376 (+/- 0.027)	0.875 (+/- 0.018)
KNN Unselected (Tuned)	0.011 (+/- 0.001)	0.050 (+/- 0.000)	0.394 (+/- 0.010)	0.998 (+/- 0.000)
LGBM Unbalanced Selected (Tuned)	0.319 (+/- 0.002)	0.006 (+/- 0.001)	0.346 (+/- 0.007)	0.375 (+/- 0.004)

In [46]:
```
#LGBM Hyperparameter Optimization
#Unbalanced
#Without selected features
pipe_lgbm_unbalanced = make_pipeline(
    preprocessor, LGBMClassifier(random_state=123, verbose=-1)
)

param_distributions_lgbm = {
    'lgbmclassifier__n_estimators': np.arange(50, 401, 10)
}

rs_lgbm_unbalanced = RandomizedSearchCV(pipe_lgbm_unbalanced,
                             param_distributions=param_distributions_lgbm,
                             n_iter=10,
                             cv=5,
                                 n_jobs=-1,
                                 random_state=123,
                                 return_train_score=True,
                                 scoring = recall_scorer)
rs_lgbm_unbalanced.fit(X_train, y_train)

print("Best parameters for LightGBM:", rs_lgbm_unbalanced.best_params_)
print("Best cross-validation score for LightGBM:", rs_lgbm_unbalanced.best_s
```

```
Best parameters for LightGBM: {'lgbmclassifier__n_estimators': 100}
Best cross-validation score for LightGBM: 0.3752112285045272
```

In [47]:
```
# Step 1: Extract the best parameters
best_lgbm_unselected_unbalanced = rs_lgbm_unbalanced.best_estimator_

mean_std_scores = mean_std_cross_val_scores(best_lgbm_unselected_unbalanced,

lgbm_unselected_unbalanced = pd.DataFrame(mean_std_scores).T
lgbm_unselected_unbalanced.index = ['LGBM Unbalanced Unselected (Tuned)']
lgbm_unselected_unbalanced
combined_scores = pd.concat([combined_scores, lgbm_unselected_unbalanced])
combined_scores
```

Out[47]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)
Random Forest Selected (Tuned)	0.458 (+/- 0.004)	0.016 (+/- 0.000)	0.642 (+/- 0.012)	0.649 (+/- 0.003)
Random Forest Unselected (Tuned)	5.272 (+/- 0.029)	0.051 (+/- 0.000)	0.632 (+/- 0.013)	0.642 (+/- 0.002)
KNN Selected (Tuned)	0.004 (+/- 0.001)	0.070 (+/- 0.002)	0.376 (+/- 0.027)	0.875 (+/- 0.018)
KNN Unselected (Tuned)	0.011 (+/- 0.001)	0.050 (+/- 0.000)	0.394 (+/- 0.010)	0.998 (+/- 0.000)
LGBM Unbalanced Selected (Tuned)	0.319 (+/- 0.002)	0.006 (+/- 0.001)	0.346 (+/- 0.007)	0.375 (+/- 0.004)
LGBM Unbalanced Unselected (Tuned)	0.251 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)

In [48]:
```
#LGBM Hyperparameter Optimization
#Balanced
#With selected features
pipe_lgbm_balanced_selected = Pipeline([
    ('classifier', LGBMClassifier(random_state=123, verbose=-1, class_weight
])

param_distributions_lgbm = {
    'classifier__n_estimators': np.arange(50, 401, 10)
}

rs_lgbm_balanced_selected = RandomizedSearchCV(pipe_lgbm_balanced_selected,
                                     param_distributions=param_distributions_lgbm,
                             n_iter=10,
                             cv=5,
                             n_jobs=-1,
                             random_state=123,
                             return_train_score=True,
                             scoring = recall_scorer)
rs_lgbm_balanced_selected.fit(X_train_selected_df, y_train)

print("Best parameters for LightGBM:", rs_lgbm_balanced_selected.best_params
print("Best cross-validation score for LightGBM:", rs_lgbm_balanced_selectec
```

```
Best parameters for LightGBM: {'classifier__n_estimators': 100}
Best cross-validation score for LightGBM: 0.643553235499587
```

In [49]:

```
# Step 1: Extract the best parameters
best_lgbm_selected_balanced = rs_lgbm_balanced_selected.best_estimator_

mean_std_scores = mean_std_cross_val_scores(best_lgbm_selected_balanced, X_t

lgbm_selected_balanced = pd.DataFrame(mean_std_scores).T
lgbm_selected_balanced.index = ['LGBM Balanced Selected (Tuned)']
lgbm_selected_balanced
combined_scores = pd.concat([combined_scores, lgbm_selected_balanced])
combined_scores
```

Out[49]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)
Random Forest Selected (Tuned)	0.458 (+/- 0.004)	0.016 (+/- 0.000)	0.642 (+/- 0.012)	0.649 (+/- 0.003)
Random Forest Unselected (Tuned)	5.272 (+/- 0.029)	0.051 (+/- 0.000)	0.632 (+/- 0.013)	0.642 (+/- 0.002)
KNN Selected (Tuned)	0.004 (+/- 0.001)	0.070 (+/- 0.002)	0.376 (+/- 0.027)	0.875 (+/- 0.018)
KNN Unselected (Tuned)	0.011 (+/- 0.001)	0.050 (+/- 0.000)	0.394 (+/- 0.010)	0.998 (+/- 0.000)
LGBM Unbalanced Selected (Tuned)	0.319 (+/- 0.002)	0.006 (+/- 0.001)	0.346 (+/- 0.007)	0.375 (+/- 0.004)
LGBM Unbalanced Unselected (Tuned)	0.251 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LGBM Balanced Selected (Tuned)	0.195 (+/- 0.001)	0.004 (+/- 0.000)	0.644 (+/- 0.017)	0.677 (+/- 0.006)

In [50]:
```
#LGBM Hyperparameter Optimization
#Balanced
#Without Selected features
pipe_lgbm_balanced_unselected = make_pipeline(
    preprocessor, LGBMClassifier(random_state=123, verbose=-1, class_weight=
)

param_distributions_lgbm = {
    'lgbmclassifier__n_estimators': np.arange(50, 401, 10)
}
```

```
rs_lgbm_balanced_unselected = RandomizedSearchCV(pipe_lgbm_balanced_unselect
                             param_distributions=param_distributions_lgbm,
                             n_iter=10,
                             cv=5,
                             n_jobs=-1,
                             random_state=123,
                             return_train_score=True,
                             scoring = recall_scorer)
rs_lgbm_balanced_unselected.fit(X_train, y_train)

print("Best parameters for LightGBM:", rs_lgbm_balanced_unselected.best_para
print("Best cross-validation score for LightGBM:", rs_lgbm_balanced_unselect
```

```
Best parameters for LightGBM: {'lgbmclassifier__n_estimators': 110}
Best cross-validation score for LightGBM: 0.6207269036734304
```

In [51]:
```
# Step 1: Extract the best parameters
best_lgbm_balanced_unselected = rs_lgbm_balanced_unselected.best_estimator_

mean_std_scores = mean_std_cross_val_scores(best_lgbm_balanced_unselected, X

lgbm_balanced_unselected = pd.DataFrame(mean_std_scores).T
lgbm_balanced_unselected.index = ['LGBM Balanced Unselected (Tuned)']
lgbm_balanced_unselected
combined_scores = pd.concat([combined_scores, lgbm_balanced_unselected])
combined_scores
```

Out[51]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)
Random Forest Selected (Tuned)	0.458 (+/- 0.004)	0.016 (+/- 0.000)	0.642 (+/- 0.012)	0.649 (+/- 0.003)
Random Forest Unselected (Tuned)	5.272 (+/- 0.029)	0.051 (+/- 0.000)	0.632 (+/- 0.013)	0.642 (+/- 0.002)
KNN Selected (Tuned)	0.004 (+/- 0.001)	0.070 (+/- 0.002)	0.376 (+/- 0.027)	0.875 (+/- 0.018)
KNN Unselected (Tuned)	0.011 (+/- 0.001)	0.050 (+/- 0.000)	0.394 (+/- 0.010)	0.998 (+/- 0.000)
LGBM Unbalanced Selected (Tuned)	0.319 (+/- 0.002)	0.006 (+/- 0.001)	0.346 (+/- 0.007)	0.375 (+/- 0.004)
LGBM Unbalanced Unselected (Tuned)	0.251 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LGBM Balanced Selected (Tuned)	0.195 (+/- 0.001)	0.004 (+/- 0.000)	0.644 (+/- 0.017)	0.677 (+/- 0.006)
LGBM Balanced Unselected (Tuned)	0.276 (+/- 0.001)	0.007 (+/- 0.000)	0.621 (+/- 0.019)	0.800 (+/- 0.009)

11. Interpretation and feature importances

rubric={points:10}

Your tasks:

1. Use the methods we saw in class (e.g., `shap`) (or any other methods of your choice) to examine the most important features of one of the non-linear models.
2. Summarize your observations.

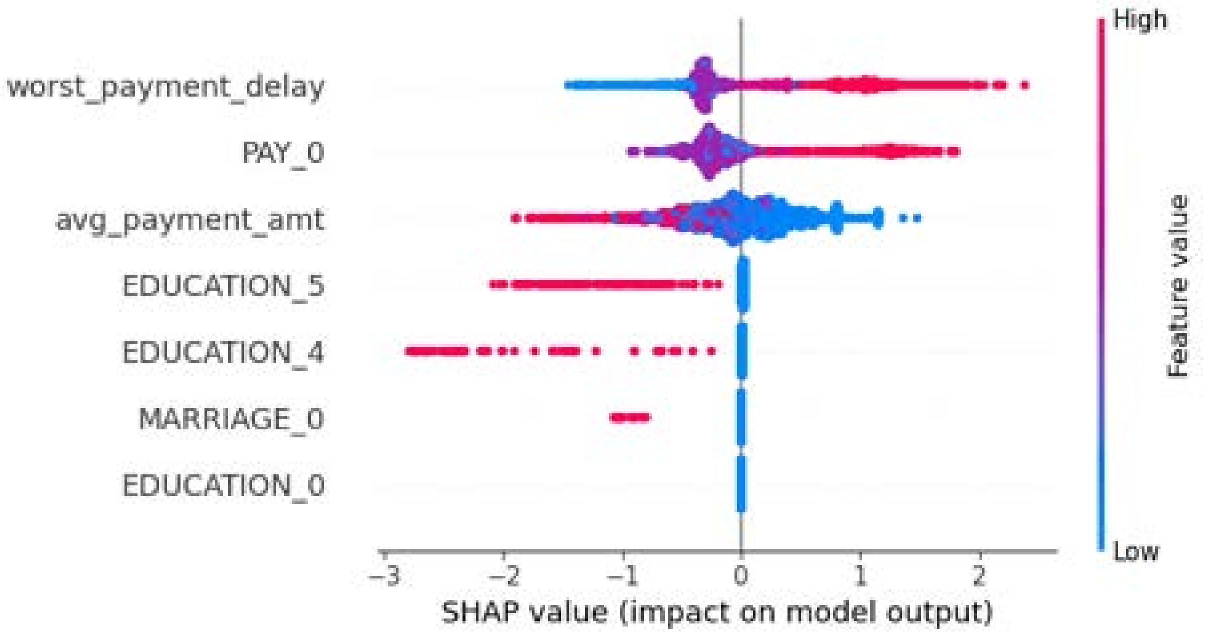
Solution_11

Points: 10

- While we are conducting the analysis, we found the dataset itself contains some values out of range. For example, we have 0 as value in 'Education' and 'Marriage', which is out of the defined range of these two features, which leads to the appearance of selected features like 'EDUCATION_0' and 'MARRIAGE_0'. Also, value of '-2' appears in PAY_0 as well, which leads to negative values in our 'worst_payment_delay'.
- The SHAP analysis across the four plots provides a detailed understanding of the most influential features and their effects on the model's predictions. Across all plots, "worst_payment_delay," "PAY_0," and "avg_payment_amt" consistently emerge as the most influential features for predicting default payments. High values of "worst_payment_delay" and "PAY_0" increase the risk of default, while high "avg_payment_amt" tends to reduce it. Demographic features like "EDUCATION" and "MARRIAGE" appear less impactful. This analysis indicates that timely payment history and sufficient average payments are key drivers in predicting default risk, and these findings are consistent across both individual and general feature importance diagrams.
- The minimal impact of EDUCATION_5, EDUCATION_4, and MARRIAGE_0, despite being selected by RFECV, may be due to their role in specific scenarios or interactions that are not prominent in the overall dataset. RFECV selects features that contribute positively to the model's performance across cross-validation folds, which might mean these features are beneficial when combined with others in certain subsets of the data. However, in the final model, their independent contribution is minimal, likely because their importance is context-dependent or interacts with other features in ways that are not captured in a straightforward manner. This highlights the difference between feature selection during model optimization and the final model's interpretation of feature importance.

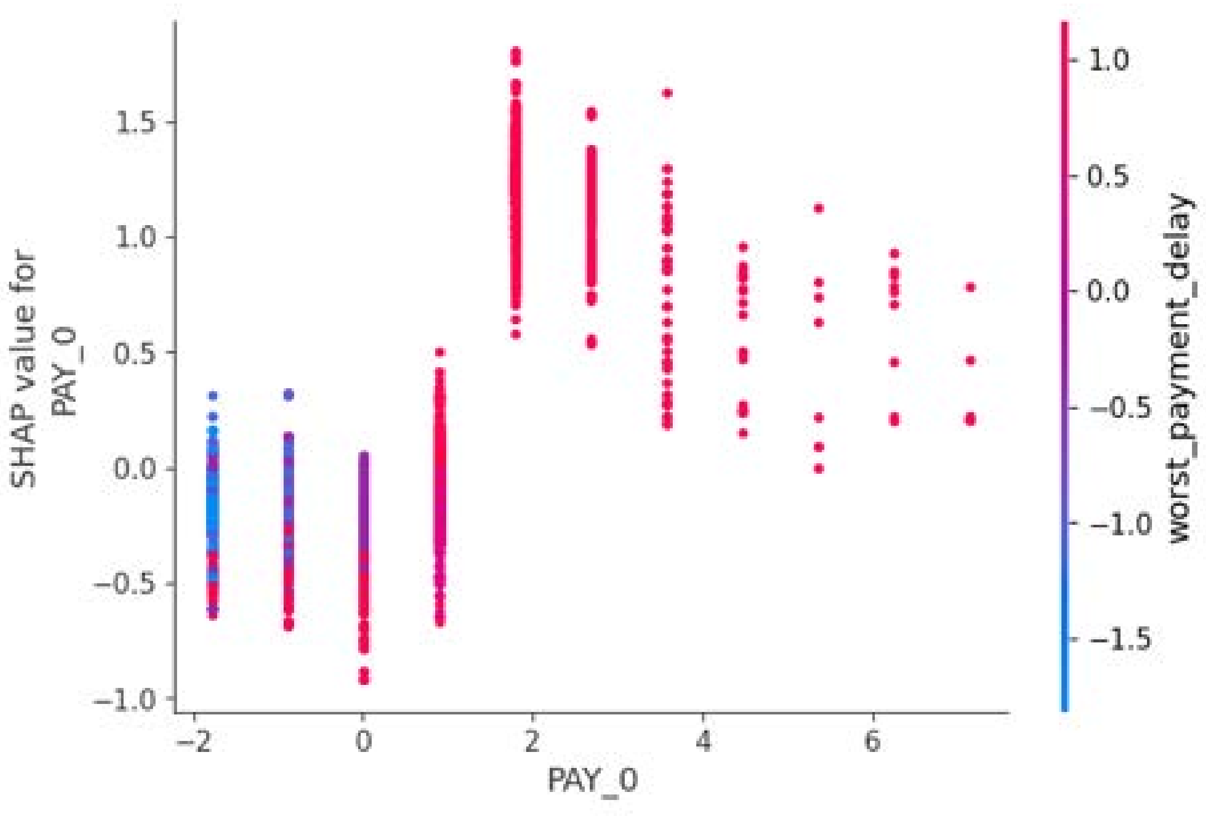
In [52]:
```python
import shap

# Initialize the SHAP TreeExplainer
```

```
explainer = shap.TreeExplainer(best_lgbm_selected_balanced.named_steps['clas

# Calculate SHAP values for the selected features
shap_values = explainer.shap_values(X_train_selected_df)

# Plot the SHAP summary plot for global feature importance
shap.summary_plot(shap_values, X_train_selected_df)
```



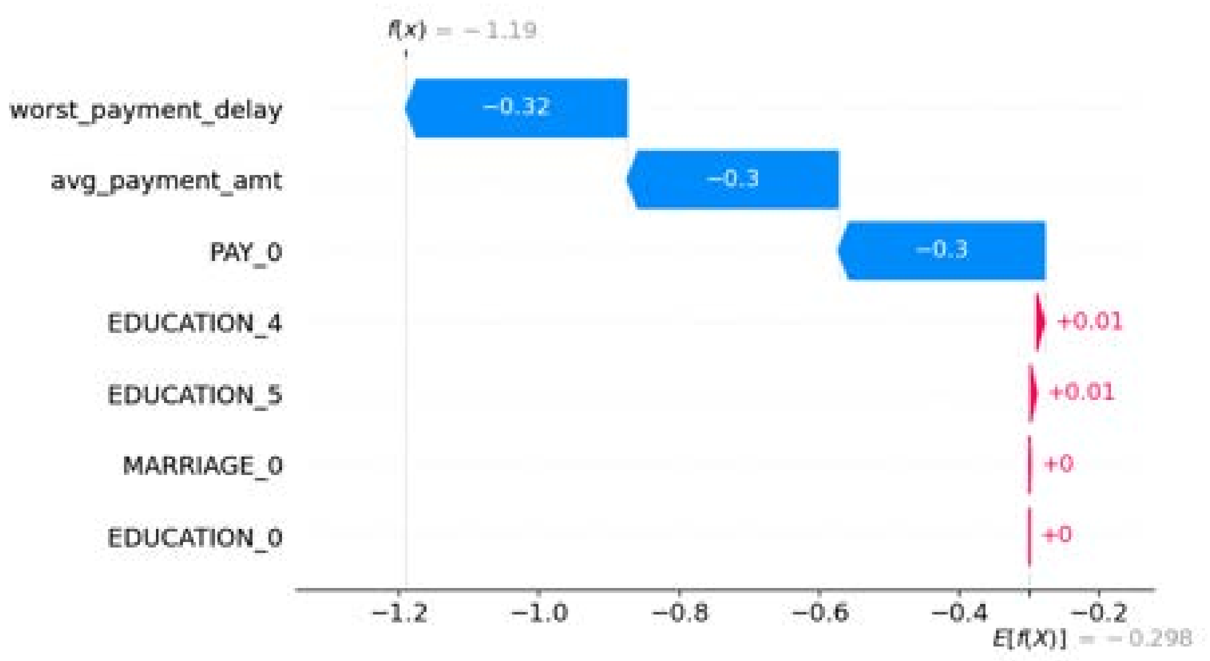
This plot gives a comprehensive overview of the impact of all selected features on the model's predictions. The features with the most significant influence are "worst_payment_delay" and "PAY_0," followed by "avg_payment_amt." The color indicates the feature value (red for high and blue for low). For example, higher "worst_payment_delay" values have a greater negative impact, pushing the SHAP value toward default risk. "EDUCATION" and "MARRIAGE" levels also exhibit variations but have a lower impact compared to the numeric features.

In [53]:
```
# Plot the SHAP dependence plot for the most important feature (for example,
shap.dependence_plot(0, shap_values, X_train_selected_df)
```



This dependency plot focuses on the relationship between "PAY_0" and its corresponding SHAP values. As the value of "PAY_0" increases, the SHAP value becomes positive, indicating an increase in the likelihood of default payment. High values of "worst_payment_delay" also appear to contribute to increased risk. The trend shows that higher "PAY_0" values, representing late payments, tend to increase the likelihood of default.

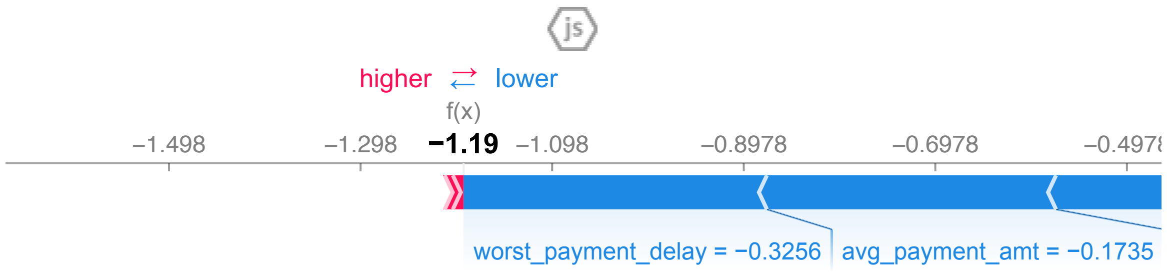
In [54]:
```
# Waterfall Plot
# Get SHAP values for the first instance in the training set
shap_values_instance = shap_values[0]
shap.plots._waterfall.waterfall_legacy(explainer.expected_value, shap_values
```



The waterfall plot illustrates how individual features contribute to a specific prediction. "worst_payment_delay", "avg_payment_amt" and "PAY_0" have strong negative contributions, driving the prediction towards a negative SHAP value, indicating a lower likelihood of default. Overall, features like education and marital status have minimal influence in this particular prediction.

In [55]:
```
# Force Plot
shap.initjs()
# Plot a force plot for the first instance
shap.force_plot(explainer.expected_value, shap_values_instance, X_train_sele
```

Out[55]:



The force plot provides a visual representation of the cumulative impact of features on a particular prediction. In this instance, features like "worst_payment_delay" and "avg_payment_amt" are pushing the model's prediction towards a non-default outcome, while "PAY_0" provides a minor push towards default. The plot effectively shows how these forces interact to form the final prediction.

12. Results on the test set

rubric={points:10}

Your tasks:

1. Try your best performing model on the test data and report test scores.
2. Do the test scores agree with the validation scores from before? To what extent do you trust your results? Do you think you've had issues with optimization bias?
3. Take one or two test predictions and explain these individual predictions (e.g., with SHAP force plots).

Solution_12

Points: 10

1.

In [56]:
```
combined_scores
```

Out[56]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)
Random Forest Selected (Tuned)	0.458 (+/- 0.004)	0.016 (+/- 0.000)	0.642 (+/- 0.012)	0.649 (+/- 0.003)
Random Forest Unselected (Tuned)	5.272 (+/- 0.029)	0.051 (+/- 0.000)	0.632 (+/- 0.013)	0.642 (+/- 0.002)
KNN Selected (Tuned)	0.004 (+/- 0.001)	0.070 (+/- 0.002)	0.376 (+/- 0.027)	0.875 (+/- 0.018)
KNN Unselected (Tuned)	0.011 (+/- 0.001)	0.050 (+/- 0.000)	0.394 (+/- 0.010)	0.998 (+/- 0.000)
LGBM Unbalanced Selected (Tuned)	0.319 (+/- 0.002)	0.006 (+/- 0.001)	0.346 (+/- 0.007)	0.375 (+/- 0.004)
LGBM Unbalanced Unselected (Tuned)	0.251 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LGBM Balanced Selected (Tuned)	0.195 (+/- 0.001)	0.004 (+/- 0.000)	0.644 (+/- 0.017)	0.677 (+/- 0.006)
LGBM Balanced Unselected (Tuned)	0.276 (+/- 0.001)	0.007 (+/- 0.000)	0.621 (+/- 0.019)	0.800 (+/- 0.009)

According to the test_score, the best performing model is Logistic regression with balanced after hyperparameter optimization.

In [57]:
```
y_pred = pipe_lr_balanced_tuned.predict(X_test)
y_pred_proba = pipe_lr_balanced_tuned.predict_proba(X_test)[:, 1]
```

```
In [58]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f

         # Calculate performance metrics
         accuracy = accuracy_score(y_test, y_pred)
         precision = precision_score(y_test, y_pred, average='binary')  # Use 'micro'
         recall = recall_score(y_test, y_pred, average='binary')
         f1 = f1_score(y_test, y_pred, average='binary')
         auc_roc = roc_auc_score(y_test, y_pred_proba)

         # Print the metrics
         print("Test Accuracy: ", accuracy)
         print("Test Precision: ", precision)
         print("Test Recall: ", recall)
         print("Test F1 Score: ", f1)
         # print("Test AUC-ROC: ", auc_roc)

         #print confusion matrix
         print("Confusion Matrix:")
         print(confusion_matrix(y_test, y_pred))
```

```
Test Accuracy:  0.7227777777777777
Test Precision:  0.4109410615434712
Test Recall:  0.6478439425051334
Test F1 Score:  0.5028890217174736
Confusion Matrix:
[[5243 1809]
 [ 686 1262]]
```

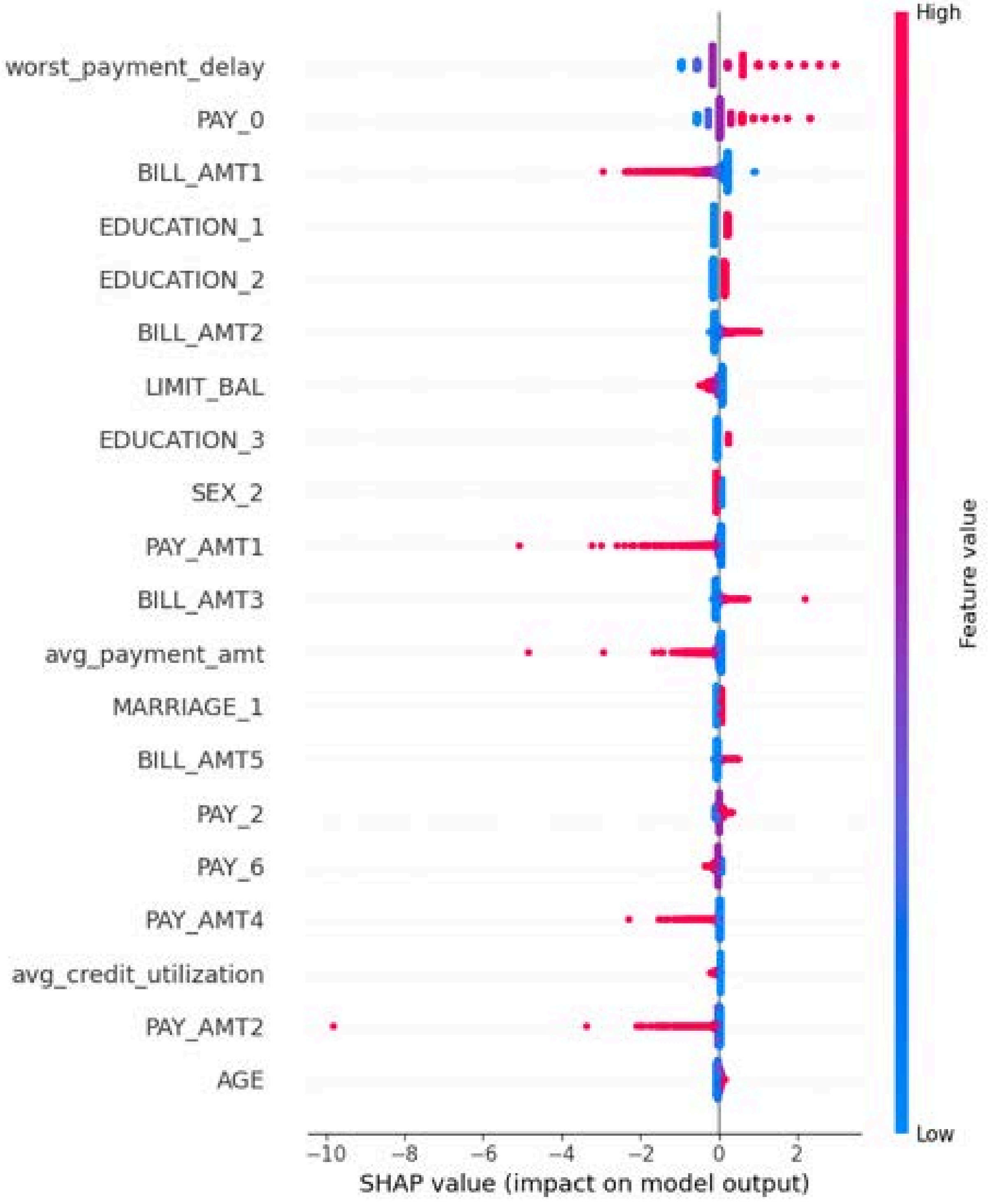
2. The test recall score of 0.648 is closely aligned with the validation recall of 0.654, suggesting that the model’s performance in terms of recall is consistent across both the validation and test datasets. This agreement indicates that the model generalizes well in identifying positive cases, and there are no significant discrepancies between the cross-validation and test results with regard to recall, which is our primary metric.

Given the close match between the validation and test recall scores, we belive we can trust the model's recall performance. The similar values suggest that the model's ability to identify true positives is stable, and there's no major overfitting or underfitting in terms of recall.

As for optimization bias, the slight increase in test accuracy (0.722) compared to validation accuracy (0.654) could hint at minor bias, but the difference is not significant enough to be a major issue since our metric we mainly used is recall. The consistent recall scores across validation and testing suggest that any optimization bias, if present, is minimal and doesn't heavily impact the model's ability to generalize on your key metric.

3.

In [59]:
```
import shap
import pandas as pd

# Fit your entire pipeline first
pipe_lr_balanced_tuned.fit(X_train, y_train)

# Use the preprocessor part of the pipeline to transform the test set
X_train_preprocessed = pipe_lr_balanced_tuned.named_steps['columntransformer
X_test_preprocessed = pipe_lr_balanced_tuned.named_steps['columntransformer'

# Extract feature names from the preprocessor
numeric_feature_names = numeric_features
binary_feature_names = pipe_lr_balanced_tuned.named_steps['columntransformer
categorical_feature_names = pipe_lr_balanced_tuned.named_steps['columntransf
all_feature_names = list(itertools.chain(numeric_feature_names, binary_featu

# Convert the transformed test data to a DataFrame with appropriate feature 
X_train_preprocessed_df = pd.DataFrame(X_train_preprocessed, columns=all_fea
X_test_preprocessed_df = pd.DataFrame(X_test_preprocessed, columns=all_featu

# Now initialize the SHAP explainer with the correct training data (DataFram
best_explainer = shap.LinearExplainer(pipe_lr_balanced_tuned.named_steps['lo

# Get SHAP values for the preprocessed test data (also as DataFrame with fea
best_shap_values = best_explainer.shap_values(X_test_preprocessed_df)

# Plot the SHAP summary plot for global feature importance with proper featu
shap.summary_plot(best_shap_values, X_test_preprocessed_df, feature_names=al
```



For the individual prediction analysis using SHAP force plots, the most influential feature is "worst_payment_delay," which significantly contributes to predicting a higher risk of default. "PAY_0" also plays a critical role, with higher values indicating greater risk. The "avg_payment_amt" influences predictions in both directions; lower values tend to increase default risk. Categorical features like "EDUCATION" and "MARRIAGE" have smaller effects. Overall, financial behavior indicators are the primary drivers of default predictions, while demographic features provide minor contributions.

13. Summary of results

rubric={points:12}
Imagine that you want to present the summary of these results to your boss and co-workers.

Your tasks:

1. Create a table summarizing important results.
2. Write concluding remarks.
3. Discuss other ideas that you did not try but could potentially improve the performance/interpretability .
4. Report your final test score along with the metric you used at the top of this notebook in the Submission instructions section.

Solution_13

Points: 12

Type your answer here, replacing this text.

1. table summarizing important results.

```
In [60]: combined_scores
```

Out[60]:

	fit_time	score_time	test_score	train_score
dummy	0.001 (+/- 0.000)	0.001 (+/- 0.000)	0.221 (+/- 0.022)	0.223 (+/- 0.009)
Logistic regression	0.040 (+/- 0.004)	0.004 (+/- 0.000)	0.238 (+/- 0.023)	0.236 (+/- 0.012)
Logistic regression_Balanced	0.045 (+/- 0.005)	0.004 (+/- 0.000)	0.653 (+/- 0.011)	0.653 (+/- 0.003)
LR_Balanced_Tuned	0.040 (+/- 0.000)	0.004 (+/- 0.000)	0.654 (+/- 0.011)	0.653 (+/- 0.002)
random forest	2.909 (+/- 0.040)	0.045 (+/- 0.001)	0.341 (+/- 0.009)	1.000 (+/- 0.000)
K-Nearest Neighbors	0.010 (+/- 0.001)	0.053 (+/- 0.009)	0.363 (+/- 0.005)	0.472 (+/- 0.005)
LightGBM	0.252 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LightGBM_balanced	0.256 (+/- 0.001)	0.007 (+/- 0.000)	0.620 (+/- 0.021)	0.788 (+/- 0.007)
LR_Balanced_Selected	0.012 (+/- 0.002)	0.002 (+/- 0.000)	0.641 (+/- 0.008)	0.641 (+/- 0.002)
Random Forest Selected (Tuned)	0.458 (+/- 0.004)	0.016 (+/- 0.000)	0.642 (+/- 0.012)	0.649 (+/- 0.003)
Random Forest Unselected (Tuned)	5.272 (+/- 0.029)	0.051 (+/- 0.000)	0.632 (+/- 0.013)	0.642 (+/- 0.002)
KNN Selected (Tuned)	0.004 (+/- 0.001)	0.070 (+/- 0.002)	0.376 (+/- 0.027)	0.875 (+/- 0.018)
KNN Unselected (Tuned)	0.011 (+/- 0.001)	0.050 (+/- 0.000)	0.394 (+/- 0.010)	0.998 (+/- 0.000)
LGBM Unbalanced Selected (Tuned)	0.319 (+/- 0.002)	0.006 (+/- 0.001)	0.346 (+/- 0.007)	0.375 (+/- 0.004)
LGBM Unbalanced Unselected (Tuned)	0.251 (+/- 0.001)	0.007 (+/- 0.000)	0.375 (+/- 0.013)	0.457 (+/- 0.004)
LGBM Balanced Selected (Tuned)	0.195 (+/- 0.001)	0.004 (+/- 0.000)	0.644 (+/- 0.017)	0.677 (+/- 0.006)
LGBM Balanced Unselected (Tuned)	0.276 (+/- 0.001)	0.007 (+/- 0.000)	0.621 (+/- 0.019)	0.800 (+/- 0.009)

2. Write concluding remarks.

- The summarized results from the table highlight the performance of various models evaluated for default prediction on the default of credit card clients dataset. Given the class imbalance in the dataset, models that incorporate balanced class weights,

such as logistic regression and LightGBM, performed better compared to their counterparts without class balancing.
- Logistic regression with balanced class weights after hyperparameter optimization achieved a notable test score of 0.654, indicating its effectiveness in addressing class imbalance and providing robust generalization.
- LightGBM models with balanced class weights or selected features also showed competitive results, with the highest test score of 0.644.
- Random Forest models demonstrated high train scores, indicative of overfitting, but had comparatively lower test scores, highlighting their limitations in handling the class imbalance effectively.
- Overall, logistic regression with balanced class weights and balanced LightGBM models demonstrated the best trade-off between performance and generalization capability.

3. Discuss other ideas that you did not try but could potentially improve the performance/interpretability .

To further improve the model's performance and interpretability, first, we can try using XGBoost, which is known for its strong performance in classification tasks, especially with imbalanced datasets, could enhance predictive accuracy. Second, we want to try adjusting the decision threshold of the classification models, which could help optimize for metrics like recall or precision, depending on the specific costs associated with false positives or false negatives in the context of default prediction.

4. Report your final test score along with the metric you used at the top of this notebook in the Submission instructions section.

Best Model: Logistic regression with balanced class after hyperparameter optimization

Best Test Score: 0.654

Metric: Recall

14. Your takeaway

rubric={points:2}
Your tasks:

What is your biggest takeaway from the supervised machine learning material we have learned so far? Please write thoughtful answers.

Solution_14

Points: 2

The biggest takeaway from the supervised machine learning material so far is the importance of thorough data preparation before diving into model building. Before starting on any tasks, it's crucial to take the time to fully investigate the dataset, understand the features, and define the target variable clearly. Spending time upfront thinking about how to preprocess the data, what pipeline to use, and which model to build can save significant effort later on. Throughout the project, we noticed that we often had to go back to earlier steps, whether to correct preprocessing, revisit feature understanding, or clarify details about the data itself. This experience taught us that a well-thought-out initial exploration and planning phase can prevent a lot of rework and lead to more effective and efficient model development.

PLEASE READ BEFORE YOU SUBMIT:

When you are ready to submit your assignment do the following:

1. Run all cells in your notebook to make sure there are no errors by doing `Kernel -> Restart Kernel and Clear All Outputs` and then `Run -> Run All Cells`.
2. Notebooks with cell execution numbers out of order or not starting from "1" will have marks deducted. Notebooks without the output displayed may not be graded at all (because we need to see the output in order to grade your work).
3. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission.
4. Make sure that the plots and output are rendered properly in your submitted file. If the .ipynb file is too big and doesn't render on Gradescope, also upload a pdf or html in addition to the .ipynb so that the TAs can view your submission on Gradescope.

This was a tricky one but you did it!