

lab4

December 9, 2023

Report the final test score along with the metric - *Classification Report*: The classification report shows a test accuracy of 0.79, with class 0 (non-default) having higher precision, recall, and F1-score than class 1 (default). This suggests a model that is better at predicting non-default cases. - *Precision-Recall Curve*: The average precision (AP) score is 0.54, indicating that the model has moderate precision across different recall levels for the positive class (default). - *ROC Curve*: The area under the ROC curve (AUC) is 0.78, suggesting that the model has a good ability to distinguish between the two classes.

```
[ ]: # Initialize Otter
import otter
grader = otter.Notebook("lab4.ipynb")
```

1 Lab 4: Putting it all together in a mini project

For this lab, **you can choose to work alone or in a group of up to four students**. You are in charge of how you want to work and who you want to work with. Maybe you really want to go through all the steps of the ML process yourself or maybe you want to practice your collaboration skills, it is up to you! Just remember to indicate who your group members are (if any) when you submit on Gradescope. If you choose to work in a group, you only need to use one GitHub repo (you can create one on [github.ubc.ca](https://github.com/ubc-ca) and set the visibility to “public”).

1.1 Submission instructions

rubric={mechanics}

You receive marks for submitting your lab correctly, please follow these instructions:

Follow the general lab instructions.

[Click here](#) to view a description of the rubrics used to grade the questions

Make at least three commits.

Push your .ipynb file to your GitHub repository for this lab and upload it to Gradescope.

Before submitting, make sure you restart the kernel and rerun all cells.

Also upload a .pdf export of the notebook to facilitate grading of manual questions (preferably WebPDF, you can select two files when uploading to gradescope)

Don't change any variable names that are given to you, don't move cells around, and don't include any code to install packages in the notebook.

The data you download for this lab SHOULD NOT BE PUSHED TO YOUR REPOSITORY (there is also a .gitignore in the repo to prevent this).

Include a clickable link to your GitHub repo for the lab just below this cell

It should look something like this https://github.ubc.ca/MDS-2020-21/DSCI_531_labX_yourcwl.

Points: 2

<https://github.com/joeywwwu/573-lab4>

1.2 Introduction

In this lab you will be working on an open-ended mini-project, where you will put all the different things you have learned so far in 571 and 573 together to solve an interesting problem.

A few notes and tips when you work on this mini-project:

Tips

1. Since this mini-project is open-ended there might be some situations where you'll have to use your own judgment and make your own decisions (as you would be doing when you work as a data scientist). Make sure you explain your decisions whenever necessary.
2. **Do not include everything you ever tried in your submission** – it's fine just to have your final code. That said, your code should be reproducible and well-documented. For example, if you chose your hyperparameters based on some hyperparameter optimization experiment, you should leave in the code for that experiment so that someone else could re-run it and obtain the same hyperparameters, rather than mysteriously just setting the hyperparameters to some (carefully chosen) values in your code.
3. If you realize that you are repeating a lot of code try to organize it in functions. Clear presentation of your code, experiments, and results is the key to be successful in this lab. You may use code from lecture notes or previous lab solutions with appropriate attributions.

Assessment We don't have some secret target score that you need to achieve to get a good grade. **You'll be assessed on demonstration of mastery of course topics, clear presentation, and the quality of your analysis and results.** For example, if you just have a bunch of code and no text or figures, that's not good. If you instead do a bunch of sane things and you have clearly motivated your choices, but still get lower model performance than your friend, don't sweat it.

A final note Finally, the style of this “project” question is different from other assignments. It'll be up to you to decide when you're “done” – in fact, this is one of the hardest parts of real projects. But please don't spend WAY too much time on this... perhaps “several hours” but not “many hours” is a good guideline for a high quality submission. Of course if you're having fun you're welcome to spend as much time as you want! But, if so, try not to do it out of perfectionism or getting the best possible grade. Do it because you're learning and enjoying it. Students from the past cohorts have found such kind of labs useful and fun and we hope you enjoy it as well.

1.3 1. Pick your problem and explain the prediction problem

rubric={reasoning}

In this mini project, you will pick one of the following problems:

1. A classification problem of predicting whether a credit card client will default or not. For this problem, you will use [Default of Credit Card Clients Dataset](#). In this data set, there are 30,000 examples and 24 features, and the goal is to estimate whether a person will default (fail to pay) their credit card bills; this column is labeled “default.payment.next.month” in the data. The rest of the columns can be used as features. You may take some ideas and compare your results with [the associated research paper](#), which is available through [the UBC library](#).

OR

2. A regression problem of predicting `reviews_per_month`, as a proxy for the popularity of the listing with [New York City Airbnb listings from 2019 dataset](#). Airbnb could use this sort of model to predict how popular future listings might be before they are posted, perhaps to help guide hosts create more appealing listings. In reality they might instead use something like vacancy rate or average rating as their target, but we do not have that available here.

Your tasks:

1. Spend some time understanding the problem and what each feature means. Write a few sentences on your initial thoughts on the problem and the dataset.
2. Download the dataset and read it as a pandas dataframe.
3. Carry out any preliminary preprocessing, if needed (e.g., changing feature names, handling of NaN values etc.)

Points: 3

The dataset from the UCI Machine Learning Repository focuses on the issue of customers’ default payments. The dataset contains 30,000 instances and 23 features, including demographic information like gender, education, marital status, and age, as well as financial details like the amount of given credit, history of past payments, bill statement amounts, and previous payment amounts

The dataset tackles the problem of predicting credit card default, a key issue for financial risk management. We find this dataset intriguing for its real-world application in risk assessment and its potential to refine predictive models. It challenges us to balance the nuances of individual financial behaviors with the broader patterns necessary for accurate default prediction, using advanced data science techniques.

```
[ ]: import sklearn
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
```

```
[ ]: df = pd.read_csv("./data/UCI_Credit_Card.csv")
df.rename(columns={"default.payment.next.month" : "target"}, inplace=True)
df.head()
```

```
[ ]:  ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0    1    20000.0    2         2         1    24      2      2     -1     -1
1    2   120000.0    2         2         2    26     -1      2      0      0
2    3    90000.0    2         2         2    34      0      0      0      0
3    4    50000.0    2         2         1    37      0      0      0      0
4    5    50000.0    1         2         1    57     -1      0     -1      0

    ...  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  \
0    ...      0.0      0.0      0.0      0.0     689.0      0.0
1    ...    3272.0    3455.0    3261.0      0.0    1000.0    1000.0
2    ...   14331.0   14948.0   15549.0   1518.0    1500.0    1000.0
3    ...   28314.0   28959.0   29547.0   2000.0    2019.0    1200.0
4    ...   20940.0   19146.0   19131.0   2000.0   36681.0   10000.0

    PAY_AMT4  PAY_AMT5  PAY_AMT6  target
0         0.0         0.0         0.0      1
1    1000.0         0.0    2000.0      1
2    1000.0    1000.0    5000.0      0
3    1100.0    1069.0    1000.0      0
4    9000.0     689.0     679.0      0
```

[5 rows x 25 columns]

```
[ ]: df.isna().sum()
```

```
[ ]: ID          0
LIMIT_BAL      0
SEX            0
EDUCATION      0
MARRIAGE       0
AGE            0
PAY_0          0
PAY_2          0
PAY_3          0
PAY_4          0
PAY_5          0
PAY_6          0
BILL_AMT1      0
BILL_AMT2      0
BILL_AMT3      0
BILL_AMT4      0
BILL_AMT5      0
BILL_AMT6      0
PAY_AMT1       0
PAY_AMT2       0
PAY_AMT3       0
PAY_AMT4       0
```

```
PAY_AMT5      0
PAY_AMT6      0
target        0
dtype: int64
```

1.4 2. Data splitting

```
rubric={reasoning}
```

Your tasks:

1. Split the data into train and test portions.

Make the decision on the `test_size` based on the capacity of your laptop.

Points: 1

```
[ ]: df_train, df_test = train_test_split(df, test_size=0.2, random_state=123)
```

1.5 3. EDA

```
rubric={viz,reasoning}
```

Perform exploratory data analysis on the train set.

Your tasks:

1. Include at least two summary statistics and two visualizations that you find useful, and accompany each one with a sentence explaining it.
2. Summarize your initial observations about the data.
3. Pick appropriate metric/metrics for assessment.

Points: 6

The summary statistics reveals significant diversity in credit behaviors among 30,000 clients, with a wide range in credit limits, bill amounts, and payment behaviors. Most clients appear to be consistent with their payments, as indicated by the repayment status fields, but there is a notable variance in payment amounts and behaviors. Approximately 22.12% of the clients are at risk of defaulting in the next month, as indicated by the target variable. This also indicate that we have a inbalance class in the target variable. The unique counts of variables shows a wide range of unique values across various financial attributes, reflecting the diversity of the client base. For example, there are 81 unique credit limits and 56 distinct age values, indicating varied client profiles.

As heatmap shown, there is a certain level of correlation between the PAY_0 to PAY_6 variables, indicating that past payment behavior is somewhat consistent over months. The bill amount variables (BILL_AMT1 to BILL_AMT6) also show moderate to high correlations with each other, suggesting that clients' bill amounts tend to be consistent over time. However, these bill amounts show little to moderate correlation with the PAY_X variables, indicating that higher bills do not necessarily correlate with delayed payments. However, we find that while there is some association between the amount billed and subsequent payment behavior, it's not a strong predictor. For example, higher bill amounts in June show a slight tendency towards delays in payment in July, but this relationship is not pronounced, suggesting other factors also play significant roles in determining payment behavior.

As countplot of the number of clients by sex and marriage status shown, there are more females(2) clients than males(1), but the proportion of defaults to non-defaults is similar between the two categories. The majority of clients are in categories married(1) and single(2), category unknown(0) and others(3) have significantly fewer clients. The proportion of defaults within each marriage category also appears to be relatively consistent, suggesting that marriage status may not be a strong standalone indicator of default risk.

```
[ ]: df_train.describe()
```

```
[ ]:
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE \
count	24000.000000	24000.000000	24000.000000	24000.000000	24000.000000
mean	14964.174292	167893.486667	1.603125	1.851958	1.553375
std	8660.479272	130109.666875	0.489260	0.790560	0.521452
min	1.000000	10000.000000	1.000000	0.000000	0.000000
25%	7467.750000	50000.000000	1.000000	1.000000	1.000000
50%	14975.000000	140000.000000	2.000000	2.000000	2.000000
75%	22460.250000	240000.000000	2.000000	2.000000	2.000000
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000

	AGE	PAY_0	PAY_2	PAY_3	PAY_4 \
count	24000.000000	24000.000000	24000.000000	24000.000000	24000.000000
mean	35.488458	-0.017542	-0.135292	-0.170042	-0.224292
std	9.217424	1.125331	1.199812	1.201709	1.170630
min	21.000000	-2.000000	-2.000000	-2.000000	-2.000000
25%	28.000000	-1.000000	-1.000000	-1.000000	-1.000000
50%	34.000000	0.000000	0.000000	0.000000	0.000000
75%	41.000000	0.000000	0.000000	0.000000	0.000000
max	79.000000	8.000000	8.000000	8.000000	8.000000

	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1 \
count	...	24000.000000	24000.000000	24000.000000	24000.000000
mean	...	43389.105625	40297.970375	38708.777542	5656.319917
std	...	64572.844994	60878.153831	59355.284889	16757.718059
min	...	-65167.000000	-61372.000000	-339603.000000	0.000000
25%	...	2310.000000	1744.250000	1200.000000	990.000000
50%	...	19032.000000	18019.000000	16812.500000	2100.000000
75%	...	54591.500000	50237.250000	49132.750000	5009.000000
max	...	891586.000000	927171.000000	961664.000000	873552.000000

	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5 \
count	2.400000e+04	24000.000000	24000.000000	24000.000000
mean	5.910454e+03	5280.658708	4763.854250	4805.837667
std	2.134743e+04	17973.951980	15162.056345	15251.828322
min	0.000000e+00	0.000000	0.000000	0.000000
25%	8.150000e+02	390.000000	281.750000	234.000000
50%	2.010000e+03	1801.500000	1500.000000	1500.000000
75%	5.000000e+03	4600.000000	4026.000000	4009.250000

```
max      1.227082e+06  896040.000000  621000.000000  426529.000000
```

	PAY_AMT6	target
count	24000.000000	24000.000000
mean	5277.577958	0.222167
std	18222.046645	0.415711
min	0.000000	0.000000
25%	110.750000	0.000000
50%	1500.000000	0.000000
75%	4000.000000	0.000000
max	528666.000000	1.000000

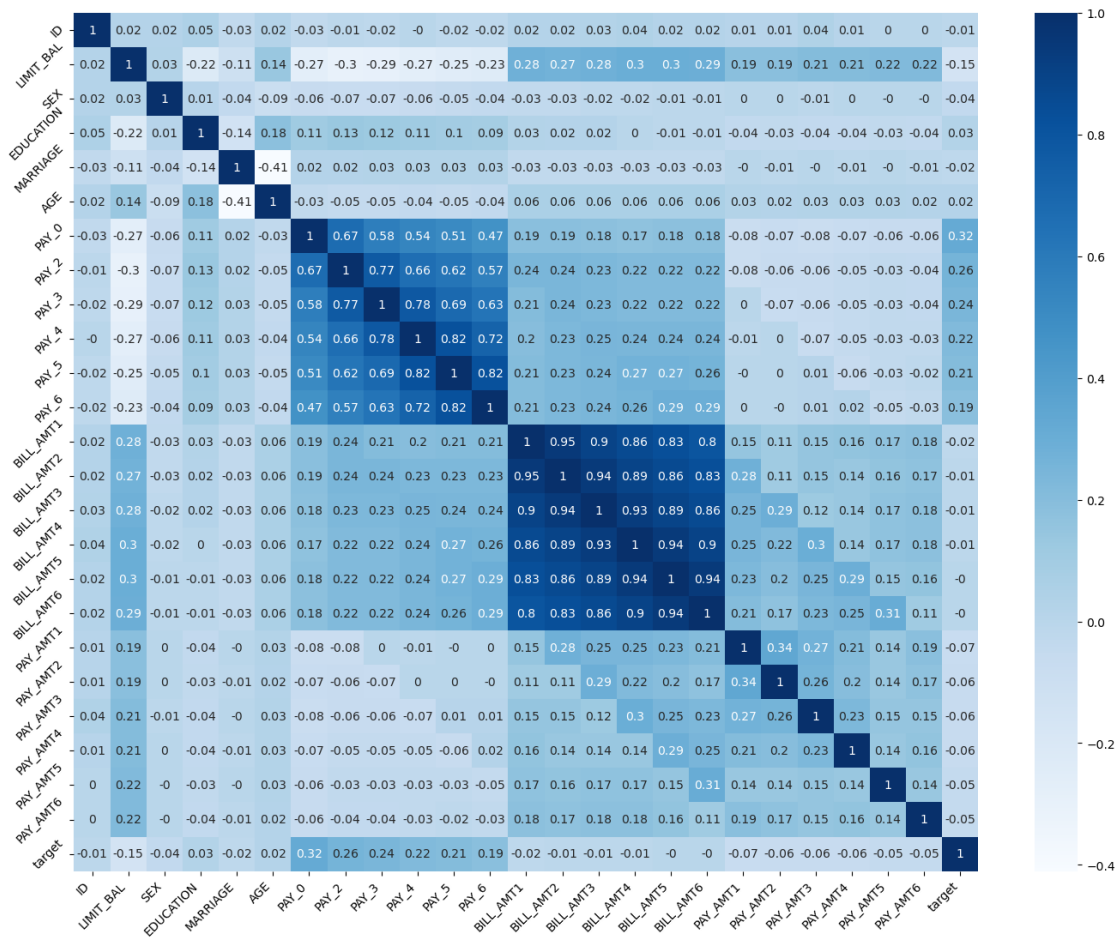
```
[8 rows x 25 columns]
```

```
[ ]: df_train.nunique()
```

```
[ ]: ID          24000
LIMIT_BAL      81
SEX             2
EDUCATION       7
MARRIAGE        4
AGE            56
PAY_0           11
PAY_2           11
PAY_3           11
PAY_4           11
PAY_5           10
PAY_6           10
BILL_AMT1      18691
BILL_AMT2      18339
BILL_AMT3      18144
BILL_AMT4      17751
BILL_AMT5      17313
BILL_AMT6      16991
PAY_AMT1        6903
PAY_AMT2        6928
PAY_AMT3        6559
PAY_AMT4        6037
PAY_AMT5        6008
PAY_AMT6        6032
target          2
dtype: int64
```

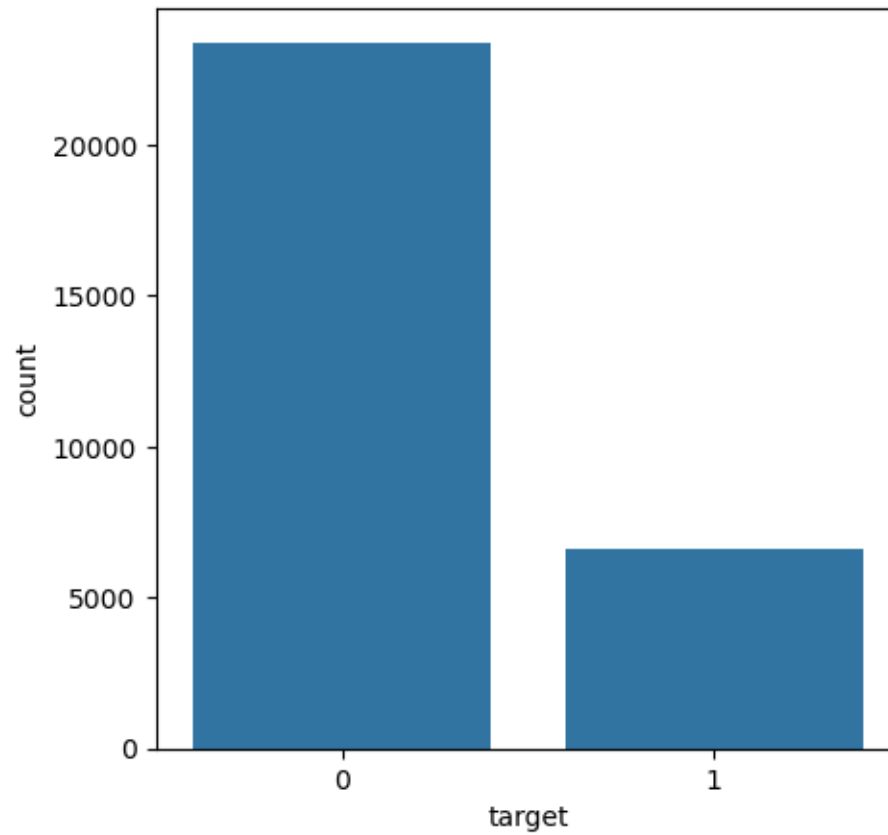
```
[ ]: corr = df_train.corr().round(2)
plt.figure(figsize=(17, 13))
sns.heatmap(corr, cmap="Blues", annot=True)
plt.xticks(rotation=45, ha='right')
```

```
plt.yticks(rotation=45)
plt.show()
```



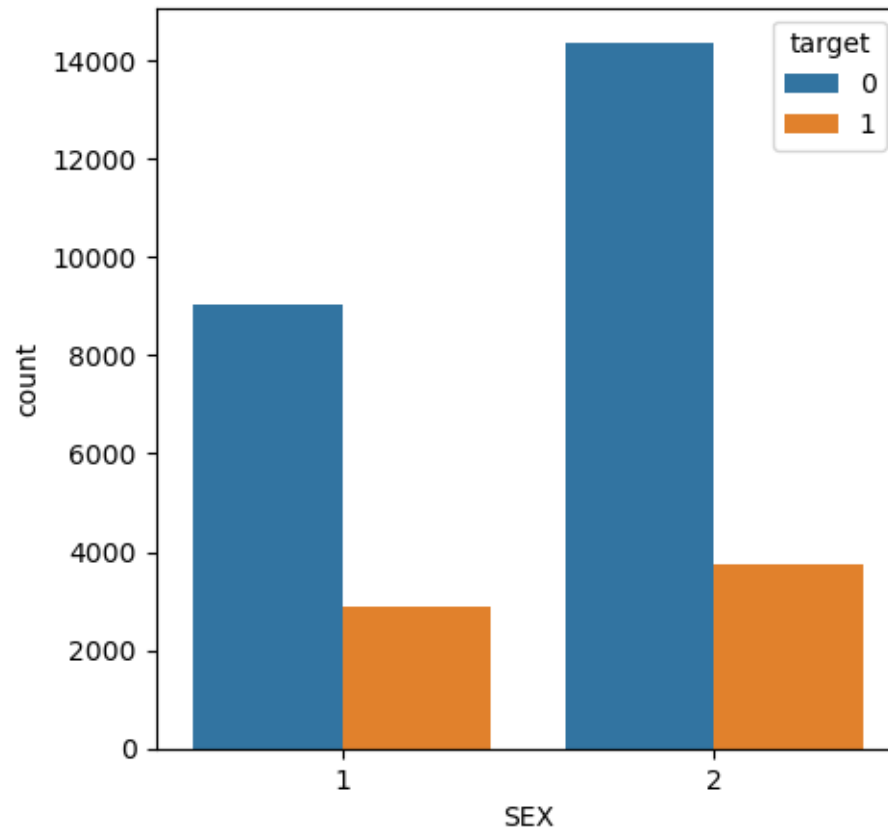
```
[ ]: plt.figure(figsize=(5,5))
sns.countplot(x = 'target', data = df)
```

```
[ ]: <Axes: xlabel='target', ylabel='count'>
```

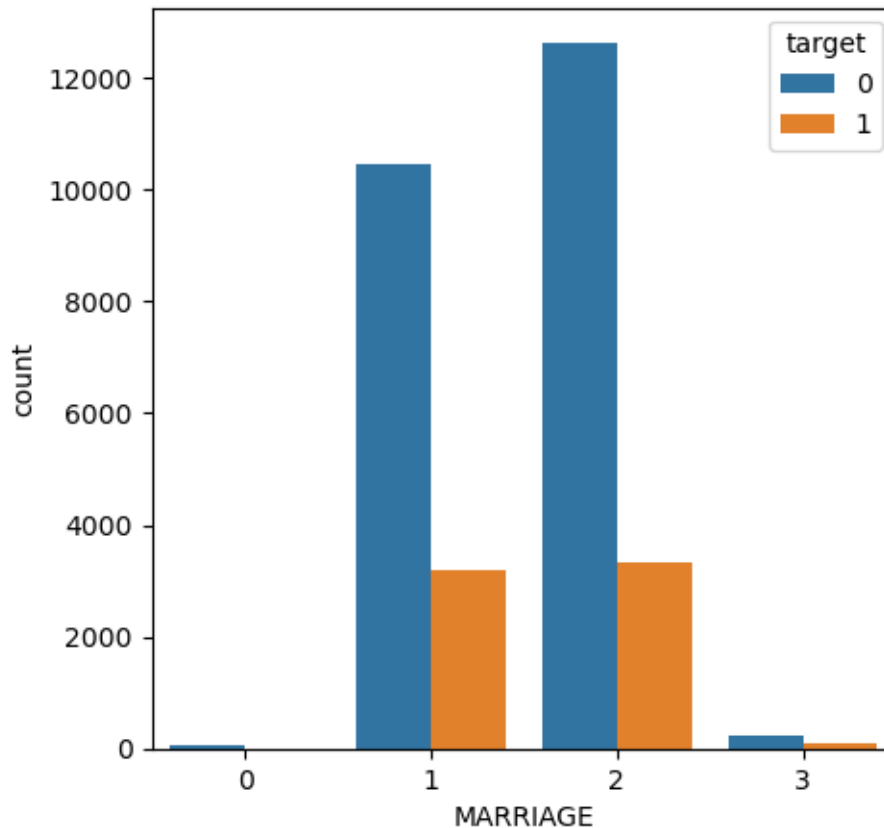
```
[ ]: plt.figure(figsize=(5,5))
     sns.countplot(x = 'SEX', hue = 'target', data = df)
```

```
[ ]: <Axes: xlabel='SEX', ylabel='count'>
```



```
[ ]: plt.figure(figsize=(5,5))  
     sns.countplot(x = 'MARRIAGE', hue = 'target', data = df)
```

```
[ ]: <Axes: xlabel='MARRIAGE', ylabel='count'>
```



```
[ ]: ...
```

```
[ ]: Ellipsis
```

1.6 4. Feature engineering (Challenging)

rubric={reasoning}

Your tasks:

1. Carry out feature engineering. In other words, extract new features relevant for the problem and work with your new feature set in the following exercises. You may have to go back and forth between feature engineering and preprocessing.

Points: 0.5

```
[ ]: df["total_pay"] = df['PAY_AMT1'] + df['PAY_AMT2'] + df['PAY_AMT3'] +  
      ↪df['PAY_AMT4'] + df['PAY_AMT5'] + df['PAY_AMT6']  
df["total_bill"] = df['BILL_AMT1'] + df['BILL_AMT2'] + df['BILL_AMT3'] +  
      ↪df['BILL_AMT4'] + df['BILL_AMT5'] + df['BILL_AMT6']  
df["account_balance"] = df["total_pay"] - df["total_bill"]
```

```
[ ]: #resplit since we did feature engineering
X, y = df.drop("target", axis=1), df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
↳random_state=123)
```

1.7 5. Preprocessing and transformations

rubric={accuracy,reasoning}

Your tasks:

1. Identify different feature types and the transformations you would apply on each feature type.
2. Define a column transformer, if necessary.

Points: 4

```
[ ]: from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import (
    StandardScaler,
    OneHotEncoder
)
from sklearn.compose import make_column_transformer
from sklearn.model_selection import cross_validate
from sklearn.model_selection import RandomizedSearchCV
```

```
[ ]: df[['BILL_AMT1', 'BILL_AMT2',
        'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
        'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']].nunique()
```

```
[ ]: BILL_AMT1    22723
BILL_AMT2    22346
BILL_AMT3    22026
BILL_AMT4    21548
BILL_AMT5    21010
BILL_AMT6    20604
PAY_AMT1      7943
PAY_AMT2      7899
PAY_AMT3      7518
PAY_AMT4      6937
PAY_AMT5      6897
PAY_AMT6      6939
dtype: int64
```

```
[ ]: df.columns
```

```
[ ]: Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0',
        'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
        'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
```

```

        'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'target',
        'total_pay', 'total_bill', 'account_balance'],
        dtype='object')

```

```

[ ]: categorical_features = ['SEX', 'EDUCATION', 'MARRIAGE', 'PAY_0', 'PAY_2',
    ↪ 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
drop_features = ['ID']
numeric_features = [
    'LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2',
    'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5',
    'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
    'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5',
    'PAY_AMT6', 'total_pay', 'total_bill',
    'account_balance'
]
passthrough_features = []
text_feature = "text"

categorical_transformer = OneHotEncoder(
    handle_unknown="ignore",
    sparse_output=False
)

numeric_transformer = StandardScaler()

preprocessor = make_column_transformer(
    (numeric_transformer, numeric_features),
    (categorical_transformer, categorical_features),
    ("drop", drop_features),
    #("passthrough", passthrough_features)
)

preprocessor

```

```

[ ]: ColumnTransformer(transformers=[('standardscaler', StandardScaler(),
    ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2',
    'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5',
    'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
    'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5',
    'PAY_AMT6', 'total_pay', 'total_bill',
    'account_balance']),
    ('onehotencoder',
    OneHotEncoder(handle_unknown='ignore',
        sparse_output=False),
    ['SEX', 'EDUCATION', 'MARRIAGE', 'PAY_0',
    'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5',
    'PAY_6'])),

```

```
('drop', 'drop', ['ID'])))
```

1.8 6. Baseline model

```
rubric={accuracy}
```

Your tasks: 1. Train a baseline model for your task and report its performance.

Points: 2

The results indicate that baseline model makes predictions using simple rules. It has the mean test and train accuracy of approximately 77.8%. However, the precision, recall, and F1 scores for both the training and testing sets are all 0, suggesting that while the classifier can correctly identify the majority class, it fails entirely at identifying the minority class (likely the default cases), which is critical for imbalanced datasets like ours.

```
[ ]: from sklearn.model_selection import cross_validate
      from sklearn.dummy import DummyClassifier
      from sklearn.ensemble import (
          RandomForestClassifier,
          AdaBoostClassifier
      )
      from sklearn.linear_model import LogisticRegression
      from sklearn.svm import SVC
      from sklearn.metrics import confusion_matrix
      from sklearn.model_selection import cross_val_predict

[ ]: classification_metrics = ["accuracy", "precision", "recall", "f1"]
      # The dummy model
      dc = DummyClassifier()

      cross_val_results = {}
      cross_val_results['dummy'] = pd.DataFrame(cross_validate(dc, X_train, y_train,
          ↪scoring=classification_metrics, return_train_score=True)).agg(['mean',
          ↪'std']).round(3).T
      cross_val_results['dummy']
```

```
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
```

```

`zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))

```

```

[ ]:
      mean    std
fit_time    0.002  0.001
score_time  0.005  0.002
test_accuracy  0.778  0.000
train_accuracy  0.778  0.000
test_precision  0.000  0.000
train_precision  0.000  0.000
test_recall    0.000  0.000
train_recall   0.000  0.000

```

```
test_f1          0.000  0.000
train_f1         0.000  0.000
```

1.9 7. Linear models

rubric={accuracy,reasoning}

Your tasks:

1. Try a linear model as a first real attempt.
2. Carry out hyperparameter tuning to explore different values for the regularization hyperparameter.
3. Report cross-validation scores along with standard deviation.
4. Summarize your results.

Points: 8

- The Logistic Regression model has achieved a mean validation accuracy of 82.0% and train accuracy of 82.2%, indicating that it performs consistently on both unseen and seen data. However, the validation precision is 68.2% which, while reasonably high, suggests that when the model predicts an instance to be a positive class (default), it is correct about 68.2% of the time. The validation recall is quite low at 35.3%, meaning it only correctly identifies 35.3% of all actual positive instances. The validation F1 score is at 46.5%, reflecting the model's struggle with correctly classifying the positive class (default cases) in the dataset. This suggests that while the model is fairly accurate overall, it is not as effective in identifying the more critical cases of default, which is typically a key objective in credit scoring models.
- As random search shown, The values in 'mean_test_score' column are around 0.533, which indicates that the mean validation accuracy is approximately 53.3% according to this output. This is significantly lower than the performance under default hyper-parameter, where the Logistic Regression model has a mean validation accuracy of 82.0% and a train accuracy of 82.2%. So, we decide to keep model with default hyper-parameter as optimal model. Although the random search performance is poor, it provides great insight on setting balanced model.

```
[ ]: # The logreg model pipeline
logreg_m = LogisticRegression(max_iter=1000, random_state=123)
logreg = make_pipeline(preprocessor, logreg_m)

cross_val_results['logreg'] = pd.DataFrame(cross_validate(logreg, X_train,
    ↳ y_train, scoring=classification_metrics, return_train_score=True)).
    ↳ agg(['mean', 'std']).round(3).T
cross_val_results['logreg']
```

```
[ ]:          mean    std
fit_time      0.300  0.032
score_time    0.008  0.000
test_accuracy  0.820  0.005
train_accuracy 0.822  0.001
test_precision 0.682  0.019
train_precision 0.690  0.006
```


test_recall	0.353	0.021
train_recall	0.357	0.005
test_f1	0.465	0.022
train_f1	0.471	0.005

```
[ ]: param_dist = {
    "logisticregression__class_weight": [None, 'balanced'],
    "logisticregression__C": np.logspace(-7, 5, 20),
    "logisticregression__max_iter" : [100, 500, 1000, 1500, 2000]
}
logreg_pipe = make_pipeline(preprocessor, LogisticRegression(random_state=123))
random_search = RandomizedSearchCV(
    logreg_pipe,
    param_dist,
    n_jobs=-1,
    n_iter=40,
    cv=5,
    scoring="f1",
    return_train_score=True,
    random_state=123
)
```

```
[ ]: random_search.fit(X_train, y_train)
best_params = random_search.best_params_
best_params
```

/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

```
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-  
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed  
to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(  
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-  
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed  
to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(  
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-  
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed  
to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(  
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-  
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed  
to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(  
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-  
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed  
to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-

```
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
regression
  n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
regression
  n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
regression
  n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
regression
  n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
regression
  n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
```

```
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
`n_iter_i = _check_optimize_result(`
`/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-`
`packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed`
`to converge (status=1):`
`STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.`

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
`n_iter_i = _check_optimize_result(`
`/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-`
`packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed`
`to converge (status=1):`
`STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.`

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
`n_iter_i = _check_optimize_result(`
`/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-`
`packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed`
`to converge (status=1):`
`STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.`

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
`n_iter_i = _check_optimize_result(`
`/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-`
`packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed`
`to converge (status=1):`
`STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.`

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
/Users/wenweiwu/miniconda3/envs/573/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

```
[ ]: {'logisticregression__max_iter': 100,
      'logisticregression__class_weight': 'balanced',
      'logisticregression__C': 23357.21469090121}
```

```
[ ]: results = pd.DataFrame(random_search.cv_results_)
sorted_results = results.sort_values(by="mean_test_score", ascending=False).
    ↪reset_index(drop=True)
sorted_results.loc[:4, ["param_logisticregression__C",
                        "param_logisticregression__max_iter",
                        "param_logisticregression__class_weight",
                        "mean_test_score",
                        "mean_fit_time",
                        "mean_train_score"]]
```

```
[ ]: param_logisticregression__C param_logisticregression__max_iter \
0          23357.214691          100
1           0.885867          2000
2           0.206914          1500
3           69.51928          2000
4           16.237767          2000

param_logisticregression__class_weight mean_test_score mean_fit_time \
0          balanced          0.533708          0.480225
1          balanced          0.533479          1.455362
2          balanced          0.533374          0.888140
3          balanced          0.533349          4.679381
4          balanced          0.533259          3.005124

mean_train_score
0          0.537503
1          0.537628
```

2	0.537598
3	0.537774
4	0.537677

1.10 8. Different models

rubric={accuracy,reasoning}

Your tasks: 1. Try out three other models aside from the linear model. 2. Summarize your results in terms of overfitting/underfitting and fit and score times. Can you beat the performance of the linear model?

Points: 10

Analyzing the results in terms of overfitting/underfitting:

- Support Vector Classifier (SVC): The SVC model shows a validation accuracy very close to the training accuracy (81.8% vs. 82.2%), indicating a good fit to the data without significant overfitting. The difference between training and validation precision is also relatively small (69.7% vs. 71.4%), suggesting that the model generalizes well to unseen data.
- Random Forest: There's a more noticeable difference between training and validation accuracy for the Random Forest model (83.4% vs. 81.6%). This indicates some overfitting, where the model performs better on the training data than on the unseen test data, although the difference is not extreme.
- AdaBoost: Similar to SVC, AdaBoost shows very little difference between its training and test accuracy (82.2% vs. 81.7%), which suggests that the model is neither overfitting nor underfitting significantly.

Considering fit and score times:

- SVC: It has the longest fit time (~90.620 seconds) and a moderate score time (~20.974 seconds). This could be an issue with larger datasets or in systems where rapid training and prediction are required.
- Random Forest: It has a much shorter fit time (~6.847 seconds) and the fastest score time (~0.146 seconds) among the three models. This makes Random Forest a more efficient choice for both training and scoring, although it does exhibit a slight tendency to overfit.
- AdaBoost: With a fit time (~21.894 seconds) longer than Random Forest and a score time (~0.441 seconds) that is also longer, AdaBoost is less efficient in terms of computational time compared to Random Forest but is more efficient than SVC.

No, above models cannot beat the performance of the linear model. Comparing these models to the Logistic Regression model, which had an validation accuracy of approximately 82% and an validation F1 score of 46.5%, we see that the Logistic Regression model has similar accuracy to Random Forest, SVC and AdaBoost but a slightly higher F1 score. This suggests that while the Logistic Regression model is comparable in identifying the majority class, it might be slightly more effective than other three models in correctly identifying the minority class (the defaults), as reflected in the F1 score.

```
[ ]: #
svc_m = SVC(random_state=123)
svc = make_pipeline(preprocessor, svc_m)

cross_val_results['svc'] = pd.DataFrame(cross_validate(svc,
                                                        X_train,
                                                        y_train,
                                                        ↪scoring=classification_metrics,
                                                        n_jobs=-1,
                                                        ↪return_train_score=True)).agg(['mean', 'std']).round(3).T

cross_val_results['svc']
```

```
[ ]:          mean    std
fit_time      31.083  0.512
score_time     5.333  0.138
test_accuracy  0.818  0.005
train_accuracy 0.822  0.001
test_precision 0.697  0.017
train_precision 0.714  0.007
test_recall    0.319  0.022
train_recall    0.333  0.003
test_f1        0.437  0.023
train_f1       0.454  0.002
```

```
[ ]: random_forest_m = RandomForestClassifier(max_depth=8, random_state=123)
random_forest = make_pipeline(preprocessor, random_forest_m)
cross_val_results['random_forest'] = pd.DataFrame(cross_validate(random_forest,
                                                                    X_train,
                                                                    y_train,
                                                                    ↪scoring=classification_metrics,
                                                                    n_jobs=-1,
                                                                    ↪return_train_score=True)).agg(['mean', 'std']).round(3).T

cross_val_results['random_forest']
```

```
[ ]:          mean    std
fit_time      1.765  0.053
score_time     0.036  0.003
test_accuracy  0.817  0.006
train_accuracy 0.836  0.003
test_precision 0.690  0.026
train_precision 0.779  0.009
```

test_recall	0.321	0.023
train_recall	0.365	0.010
test_f1	0.438	0.024
train_f1	0.497	0.011

```
[ ]: ada_boost_m = AdaBoostClassifier(n_estimators=200, random_state=123)
ada_boost = make_pipeline(preprocessor, ada_boost_m)
cross_val_results['ada_boost'] = pd.DataFrame(cross_validate(ada_boost,
                                                             X_train,
                                                             y_train,
                                                             n_jobs=-1,
                                                             scoring=classification_metrics,
                                                             return_train_score=True)).agg(['mean', 'std']).round(3).T
cross_val_results['ada_boost']
```

```
[ ]:
fit_time      mean    std
score_time    0.163  0.039
test_accuracy 0.817  0.005
train_accuracy 0.822  0.002
test_precision 0.669  0.024
train_precision 0.692  0.004
test_recall   0.345  0.014
train_recall  0.359  0.010
test_f1       0.456  0.016
train_f1      0.473  0.009
```

1.11 9. Feature selection (Challenging)

rubric={reasoning}

Your tasks:

Make some attempts to select relevant features. You may try RFECV, forward/backward selection or L1 regularization for this. Do the results improve with feature selection? Summarize your results. If you see improvements in the results, keep feature selection in your pipeline. If not, you may abandon it in the next exercises unless you think there are other benefits with using less features.

Points: 0.5

Type your answer here, replacing this text.

```
[ ]: ...
```

```
[ ]: Ellipsis
```

```
[ ]: ...
```

```
[ ]: Ellipsis
```

```
[ ]: ...
```

```
[ ]: Ellipsis
```

1.12 10. Hyperparameter optimization

```
rubric={accuracy,reasoning}
```

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. Briefly summarize your results. - [GridSearchCV](#)

- [RandomizedSearchCV](#) - [scikit-optimize](#)

Points: 6

- The Random Forest model, despite a validation F1 score of 42.8%, shines in its accuracy and computational efficiency. An accuracy rate of 81.6% demonstrates its reliability in making correct predictions. Beyond this, its swift score time of 0.146 seconds is particularly valuable in practice, where making quick predictions can be as important as making accurate ones. This speed ensures that the model can handle large volumes of data efficiently, keeping computational costs in check, which is a significant advantage in operational settings. These attributes make the Random Forest model a strong candidate for further development, especially when considering the balance between performance, speed, and practical utility.
- After running a randomized search to fine-tune the settings of a Random Forest model, the best result came from a model using 50 trees and allowing each tree to split up to 15 levels deep. This model chose features to consider at each split based on the 'log2' rule and gave equal importance to all classes. It scored about 54.1% in testing, which was one of the highest scores achieved, and it was also quick to build, taking just over 5 seconds on average. This makes it a practical option because it balances accuracy with the speed and care in treating all categories fairly in the predictions. Other model variations with more trees or different settings were also effective but took longer to run, which might not be ideal in a fast-paced or resource-constrained environment.

```
[ ]: param_dist_rf = {  
    "randomforestclassifier__n_estimators": [10, 50, 100, 200],  
    "randomforestclassifier__max_depth": [None, 5, 10, 15, 20],  
    "randomforestclassifier__min_samples_split": [2, 5, 10, 20, 40],  
    "randomforestclassifier__max_features": [None, 'sqrt', 'log2'],  
    "randomforestclassifier__class_weight": [None, "balanced"]  
}  
  
random_forest_pipe = make_pipeline(preprocessor,  
    ↪ RandomForestClassifier(random_state=0))
```



```

random_search_rf = RandomizedSearchCV(
    random_forest_pipe,
    param_dist_rf,
    n_jobs=-1,
    n_iter=40,
    cv=3,
    scoring="f1",
    return_train_score=True,
    random_state=123
)

```

```

[ ]: random_search_rf.fit(X_train, y_train)
random_search_rf.best_params_, random_search_rf.best_score_

```

```

[ ]: ({'randomforestclassifier__n_estimators': 50,
      'randomforestclassifier__min_samples_split': 20,
      'randomforestclassifier__max_features': 'log2',
      'randomforestclassifier__max_depth': 15,
      'randomforestclassifier__class_weight': 'balanced'},
      0.540998396338093)

```

```

[ ]: results_rf = pd.DataFrame(random_search_rf.cv_results_)
sorted_results_rf = results_rf.sort_values(by="mean_test_score",
      ↪ascending=False).reset_index(drop=True)

# Select and rename the columns to display the top 5 results
top_results_rf = sorted_results_rf.loc[
    :4,
    [
        "param_randomforestclassifier__n_estimators",
        "param_randomforestclassifier__max_depth",
        "param_randomforestclassifier__min_samples_split",
        "param_randomforestclassifier__max_features",
        "param_randomforestclassifier__class_weight",
        "mean_test_score",
        "mean_fit_time",
        "mean_train_score"
    ]
]
top_results_rf = top_results_rf.rename(
    columns={
        "param_randomforestclassifier__n_estimators": "N Estimators",
        "param_randomforestclassifier__max_depth": "Max Depth",
        "param_randomforestclassifier__min_samples_split": "Min Samples Split",
        "param_randomforestclassifier__max_features": "Max Features",
        "param_randomforestclassifier__class_weight": "Class Weight",
        "mean_test_score": "Mean Test Score",
    }
)

```

```

        "mean_fit_time": "Mean Fit Time",
        "mean_train_score": "Mean Train Score"
    }
)

top_results_rf

```

```

[ ]:  N Estimators Max Depth Min Samples Split Max Features Class Weight \
0          50         15          20        log2      balanced
1         200         10          20        log2      balanced
2         200         20          20        log2      balanced
3         100         10          10        log2      balanced
4         200         10           2        None      balanced

      Mean Test Score  Mean Fit Time  Mean Train Score
0          0.540998         1.015094         0.677732
1          0.540535         3.177600         0.602539
2          0.540471         4.378770         0.739242
3          0.539223         1.574902         0.613717
4          0.535357        36.186902         0.656666

```

1.13 11. Interpretation and feature importances

rubric={accuracy,reasoning}

Your tasks:

1. Use the methods we saw in class (e.g., `eli5`, `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.

Points: 8

1. The SHAP values indicate that recent repayment statuses, such as `PAY_0`, `PAY_2`, and `PAY_4`, are critical in the model's predictions. These features signify how timely a client is with their payments, with higher values suggesting delays. Delays in recent payments are seen as strong indicators of a potential default in the next month.
2. The credit limit is also a significant factor, with the SHAP analysis suggesting that clients with higher credit limits may have a higher risk of default. This might be because clients with higher limits have the ability to accumulate more debt, increasing the potential for default.
3. The amounts paid in previous months, such as `PAY_AMT1`, `PAY_AMT2`, etc., show that clients who have paid more towards their bill statements are less likely to default. These features seem to have a protective effect against the risk of default.

The SHAP findings correspond with the feature importance results from the Random Forest model. Both identify the repayment status variables (`PAY_0`, `PAY_2`, `PAY_4`) and the amount of credit extended (`LIMIT_BAL`) as highly influential in predicting default. Additionally, both analyses agree that higher previous payment amounts (`PAY_AMT1`, `PAY_AMT2`, etc.) decrease the likelihood of

default, indicating the model values recent payment behavior and credit utilization as key indicators of credit risk.

```
[ ]: import shap

[ ]: best_rf_model = random_search_rf.best_estimator_

[ ]: X_test_transformed = best_rf_model.named_steps['columntransformer'].
    ↪transform(X_test)
    explainer = shap.TreeExplainer(best_rf_model.
    ↪named_steps['randomforestclassifier'])

[ ]: shap_values = explainer.shap_values(X_test_transformed)

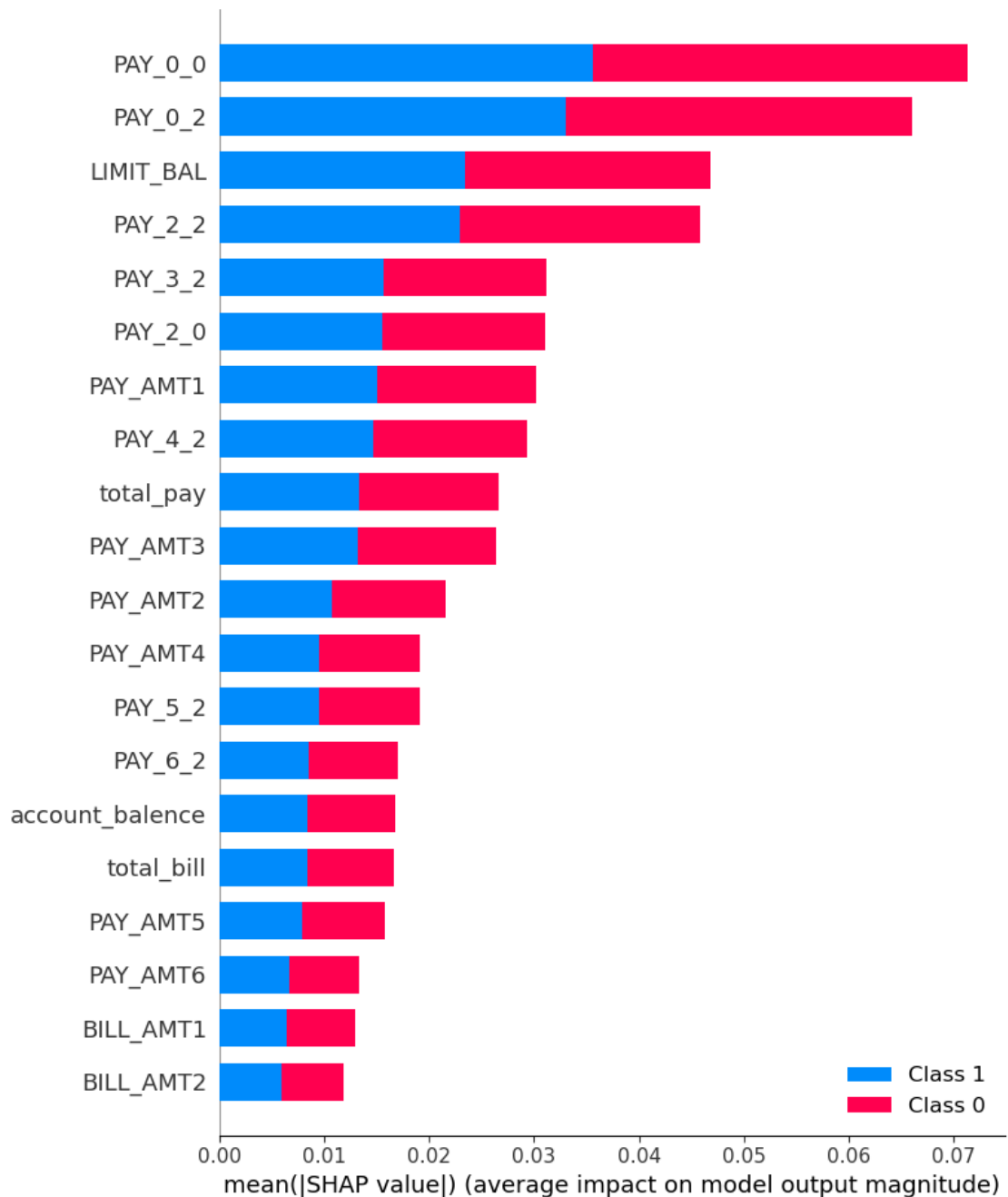
[ ]: best_rf_preprocessor = best_rf_model.named_steps['columntransformer']
    feature_names = list(best_rf_preprocessor.named_transformers_['standardscaler'].
    ↪get_feature_names_out()) + \
    list(best_rf_preprocessor.named_transformers_['onehotencoder'].
    ↪get_feature_names_out())

[ ]: shap.initjs()
    shap.force_plot(explainer.expected_value[1], shap_values[1][0, :],
    ↪X_test_transformed[0, :], feature_names=feature_names)

<IPython.core.display.HTML object>

[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x2866d6490>

[ ]: shap.summary_plot(shap_values, X_test_transformed, feature_names=feature_names)
```



```
[ ]: rf_feature_importances = best_rf_model.named_steps['randomforestclassifier'].
    ↪ feature_importances_

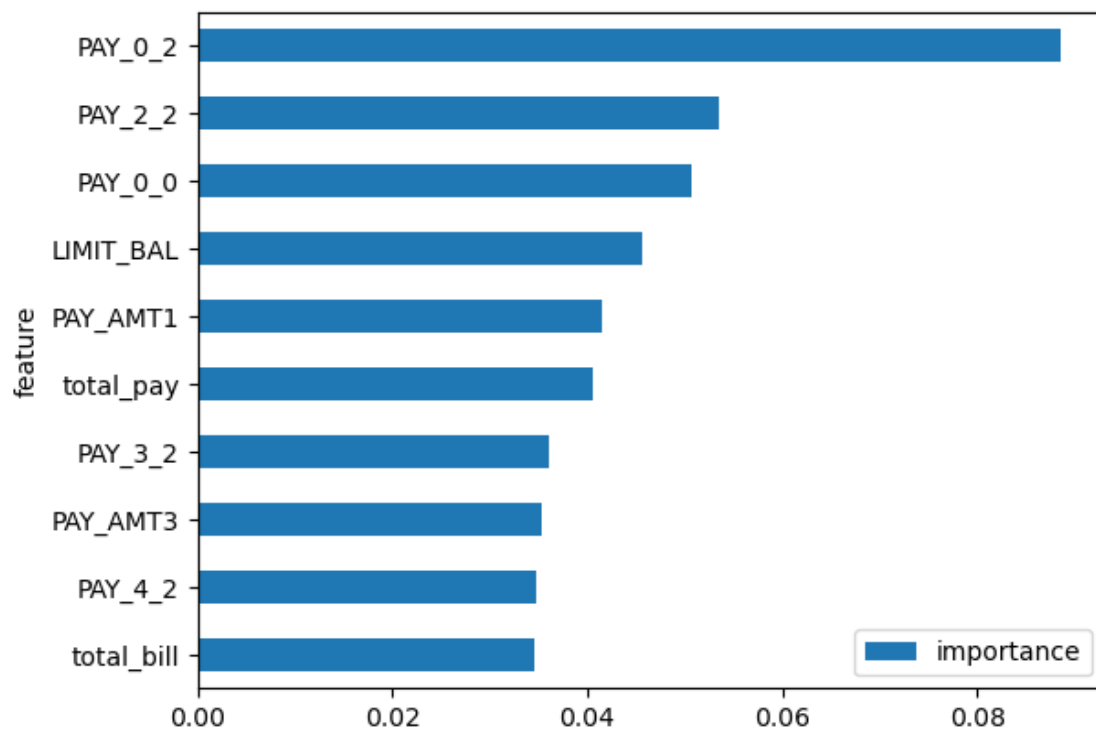
[ ]: feature_importances_df = pd.DataFrame({'feature' : feature_names, 'importance' :
    ↪ rf_feature_importances})
top_feature_importances_df = feature_importances_df.
    ↪ sort_values(by='importance', ascending=False) \
```

```
.head(10).sort_values(by='importance', ascending=True)
```

```
top_feature_importances_df
```

```
[ ]:      feature  importance
15  total_bill    0.034486
67   PAY_4_2     0.034684
10   PAY_AMT3    0.035231
56   PAY_3_2     0.036118
14  total_pay    0.040584
8    PAY_AMT1    0.041586
0    LIMIT_BAL   0.045600
32   PAY_0_0     0.050620
45   PAY_2_2     0.053488
34   PAY_0_2     0.088564
```

```
[ ]: top_feature_importances_df.plot.barh(x="feature", y='importance', rot=0)
plt.show()
```



```
[ ]: ...
```

```
[ ]: Ellipsis
```

[]: ...

[]: Ellipsis

[]: ...

[]: Ellipsis

1.14 12. Results on the test set

rubric={accuracy,reasoning}

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 them with SHAP force plots.

Points: 6

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?

- The validation scores from the cross-validation results show an validation accuracy of 81.6% and an validation F1 score 42.8%. When comparing these to the test scores you provided, the test accuracy reported in the classification report is slightly lower at 79%, and the test F1 score is 71%, which is higher than the validation F1 score of 42.8%. The slight discrepancies between validation and test scores can be normal, especially if the test dataset has different characteristics than the cross-validated training sets. However, since the difference is not substantial, it suggests that the model has generalized reasonably well to unseen data. The trust in these results would be stronger if the test set is large and representative of the problem space.
- Regarding optimization bias, there's less indication of it since the test F1 is relatively high compared to the validation F1 score. However, it's still essential to consider that the model might be better at predicting one class over the other, which could be masked when looking at the macro average. To ensure a lack of bias, it would be important that the model's hyperparameters were tuned using a representative validation set and that the test set was not used in any way during model selection.

3. Take one or two test predictions and explain them with SHAP force plots.

The SHAP force plot provided for client ID 22386, who has a credit limit (LIMIT_BAL) of 170,000 NT dollars, shows how different features are influencing the prediction of default (class 1) for this particular individual.

The red features indicate those pushing the prediction towards a higher likelihood of default: - PAY_0, PAY_2, PAY_3, PAY_4 are all set to 2, indicating a two-month payment delay for several consecutive months, which greatly increases the model's prediction towards default. - PAY_AMT4 is quite high at 13,000 NT dollars, but given the context of delayed payments in the other PAY features, this might suggest erratic payment behavior, which can be seen as risky by the model.

The blue features, which push the prediction towards non-default, are as follows: - LIMIT_BAL is having a strong negative impact on the prediction of default, which might be indicative of the model's learned pattern that higher credit limits are less likely to default, potentially due to the creditworthiness assessment carried out by the financial institution. - PAY_0_0 and PAY_AMT6 are lower amounts and might usually indicate lower risk, but in this case, their impact is not enough to offset the strong red signals from the delayed payment statuses.

This individual's prediction is heavily influenced by their recent payment history, with several delays indicating a pattern of behavior that the model associates with a higher risk of default, despite a relatively high credit limit and some recent payments.

```
[ ]: from sklearn.metrics import PrecisionRecallDisplay, RocCurveDisplay,
      ↪confusion_matrix, classification_report

[ ]: best_rf_pipe = make_pipeline(preprocessor,
      ↪RandomForestClassifier(random_state=0)).set_params(**random_search_rf.
      ↪best_params_)

[ ]: best_rf_pipe.fit(X_train, y_train)

[ ]: Pipeline(steps=[('columntransformer',
                      ColumnTransformer(transformers=[('standardscaler',
                                                         StandardScaler(),
                                                         ['LIMIT_BAL', 'AGE',
                                                         'BILL_AMT1', 'BILL_AMT2',
                                                         'BILL_AMT3', 'BILL_AMT4',
                                                         'BILL_AMT5', 'BILL_AMT6',
                                                         'PAY_AMT1', 'PAY_AMT2',
                                                         'PAY_AMT3', 'PAY_AMT4',
                                                         'PAY_AMT5', 'PAY_AMT6',
                                                         'total_pay', 'total_bill',
                                                         'account_balance']),
                                                         ('onehotencoder',
                                                         OneHotEncoder(handle_unknown='ignore',
                                                         sparse_output=False),
                                                         ['SEX', 'EDUCATION',
                                                         'MARRIAGE', 'PAY_0', 'PAY_2',
                                                         'PAY_3', 'PAY_4', 'PAY_5',
                                                         'PAY_6'])],
                                                         ('drop', 'drop', ['ID']))]),
                  ('randomforestclassifier',
                  RandomForestClassifier(class_weight='balanced', max_depth=15,
                                         max_features='log2',
                                         min_samples_split=20, n_estimators=50,
                                         random_state=0))])

[ ]: y_pred = best_rf_pipe.predict(X_test)
```

```
[ ]: conf_matrix = confusion_matrix(y_test, y_pred)
conf_matrix
```

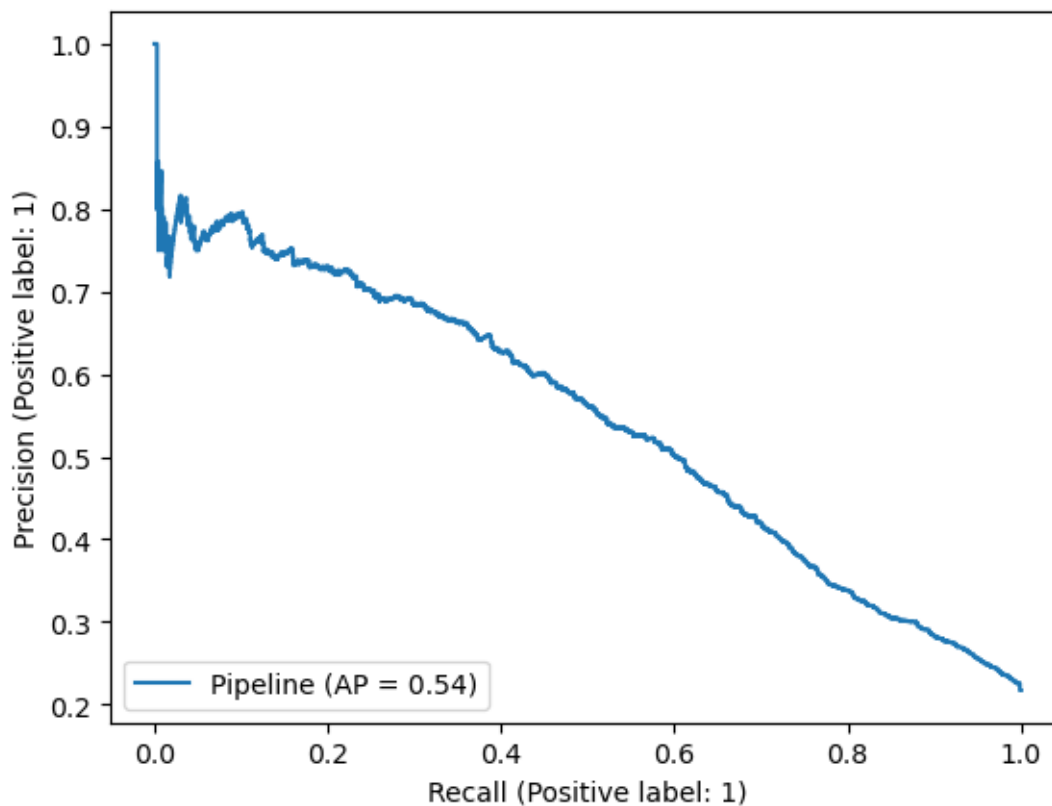
```
[ ]: array([[3993,  703],
          [ 549,  755]])
```

```
[ ]: class_report = classification_report(y_test, y_pred)
print(class_report)
```

	precision	recall	f1-score	support
0	0.88	0.85	0.86	4696
1	0.52	0.58	0.55	1304
accuracy			0.79	6000
macro avg	0.70	0.71	0.71	6000
weighted avg	0.80	0.79	0.80	6000

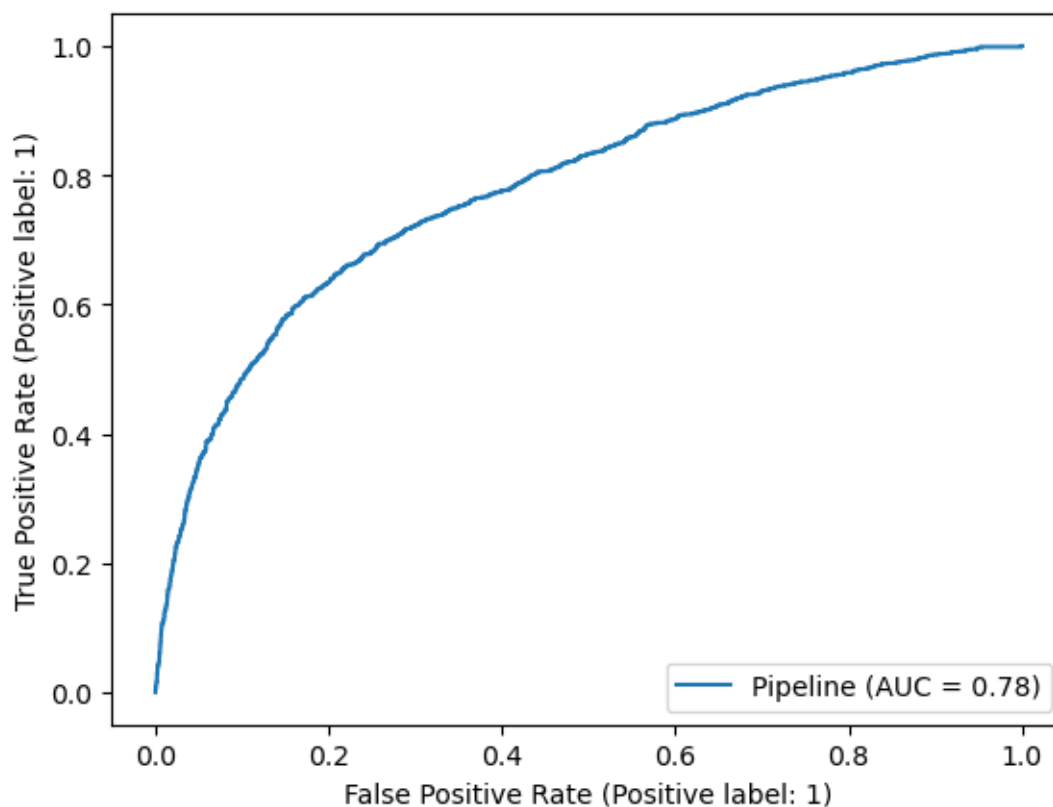
```
[ ]: PrecisionRecallDisplay.from_estimator(best_rf_pipe, X_test, y_test)
```

```
[ ]: <sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at
0x286994a90>
```




```
[ ]: RocCurveDisplay.from_estimator(best_rf_model, X_test, y_test)
```

```
[ ]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x286b08250>
```



```
[ ]: X_test[y_pred == 1][:3]
```

```
[ ]:
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	\
22386	22387	170000.0	2	1	2	30	2	2	2	
19209	19210	210000.0	2	1	2	30	-2	-2	-2	
120	121	50000.0	1	3	2	37	2	2	2	

	PAY_4	...	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	\
22386	2	...	170922.0	6800.0	6500.0	0.0	13000.0	
19209	-2	...	0.0	0.0	0.0	0.0	0.0	
120	3	...	51143.0	1000.0	4035.0	1000.0	1400.0	

	PAY_AMT5	PAY_AMT6	total_pay	total_bill	account_balance
22386	5500.0	1000.0	32800.0	972809.0	-940009.0
19209	0.0	0.0	0.0	0.0	0.0

```
120      2800.0      0.0    10235.0    290245.0    -280010.0
```

```
[3 rows x 27 columns]
```

```
[ ]: y_test[y_pred == 1][:3]
```

```
[ ]: 22386    1
      19209    0
      120     1
      Name: target, dtype: int64
```

1.15 13. Summary of results

```
rubric={reasoning}
```

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.

Points: 8

2. Write concluding remarks.

The Random Forest model was chosen as the optimal model due to its robust performance metrics and computational efficiency. It demonstrated a high accuracy rate of 81.6%, indicating reliable predictive capabilities. Moreover, its rapid score time of approximately 0.146 seconds suggests that it can process large volumes of data quickly, a crucial factor in operational environments where time efficiency is valued alongside accuracy. Additionally, Random Forest models offer interpretability through feature importance scores, allowing us to understand which factors most influence predictions.

3. Discuss other ideas that you did not try but could potentially improve the performance/interpretability. - Ensemble methods could be explored to improve performance, like Gradient Boosting or Stacking classifiers, which might offer a better balance between precision and recall. - Feature engineering, such as interaction terms or polynomial features, might uncover more complex relationships in the data. - Investigating model interpretability tools beyond SHAP, like LIME or partial dependence plots, could provide more insight into individual predictions and potentially reveal areas for model refinement. - Addressing class imbalance with techniques like SMOTE or targeted resampling could improve the minority class prediction. - Advanced neural networks, although more complex and less interpretable, might capture non-linear patterns better and improve predictive performance.

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

- *Classification Report*: The classification report shows a test accuracy of 0.79, with class 0 (non-default) having higher precision, recall, and F1-score than class 1 (default). This suggests a model that is better at predicting non-default cases.
- *Precision-Recall Curve*: The average precision (AP) score is 0.54, indicating that the model has moderate precision across different recall levels for the positive class (default).
- *ROC Curve*: The area under the ROC curve (AUC) is 0.78, suggesting that the model has a good ability to distinguish between the two classes.

```
[ ]: # Compare the average scores of all the models
pd.concat(
    cross_val_results,
    axis='columns'
).xs(
    'mean',
    axis='columns',
    level=1
).style.format(
    precision=2
).background_gradient(
    axis=None )
```

```
[ ]: <pandas.io.formats.style.Styler at 0x286b08f10>
```

```
[ ]: ...
```

```
[ ]: Ellipsis
```

1.16 14. Creating a data analysis pipeline (Challenging)

rubric={reasoning}

Your tasks:

- Convert this notebook into scripts to create a reproducible data analysis pipeline with appropriate documentation. Submit your project folder in addition to this notebook on GitHub and briefly comment on your organization in the text box below.

Points: 2

Type your answer here, replacing this text.

1.17 15. Your takeaway from the course (Challenging)

rubric={reasoning}

Your tasks:

What is your biggest takeaway from this course?

Points: 0.25

Type your answer here, replacing this text.

Restart, run all and export a PDF before submitting

Before submitting, don't forget to run all cells in your notebook to make sure there are no errors and so that the TAs can see your plots on Gradescope. You can do this by clicking the `Run` button or going to `Kernel -> Restart Kernel and Run All Cells...` in the menu. This is not only important for MDS, but a good habit you should get into before ever committing a notebook to GitHub, so that your collaborators can run it from top to bottom without issues.

After running all the cells, export a PDF of the notebook (preferably the WebPDF export) and upload this PDF together with the ipynb file to Gradescope (you can select two files when uploading to Gradescope)

1.18 Help us improve the labs

The MDS program is continually looking to improve our courses, including lab questions and content. The following optional questions will not affect your grade in any way nor will they be used for anything other than program improvement:

1. Approximately how many hours did you spend working or thinking about this assignment (including lab time)?

#Ans:

2. Do you have any feedback on the lab you be willing to share? For example, any part or question that you particularly liked or disliked?

#Ans: