In [1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lab4.ipynb")
```

In [2]:
```
import sys
from hashlib import sha1

import matplotlib.pyplot as plt
import altair as alt
import altair_ally as aly
import numpy as np
import pandas as pd
from sklearn.dummy import DummyClassifier, DummyRegressor
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import (
    GridSearchCV,
    RandomizedSearchCV,
    cross_validate,
    train_test_split,
)
from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.linear_model import Ridge, LinearRegression
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.preprocessing import (
    MinMaxScaler,
    OneHotEncoder,
    OrdinalEncoder,
    StandardScaler,
)

from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.feature_selection import SelectFromModel
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegressionCV
import altair as alt
alt.data_transformers.enable('data_server')
aly.alt.data_transformers.enable('data_server')
alt.renderers.enable('mimetype')


from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from sklearn.metrics import roc_curve
from sklearn.metrics import RocCurveDisplay
from sklearn.datasets import fetch_california_housing
from scipy.stats import loguniform, randint, uniform

from sklearn.metrics import f1_score, mean_squared_error, mean_absolute_percentage_error
from sklearn.compose import TransformedTargetRegressor
from numpy import array
from numpy.linalg import norm
```

Lab 4: Putting it all together in a mini project

This lab is an optional group lab. You can choose to work alone of 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 of your GitHub repos.

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/UBC-MDS/dsci573_lab4_credit-card-class

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. 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

1.1 Problem definition

Overview

"Default of Credit Card Clients Dataset" contains data on demographics (i.e age, marriage status), previous repayment statuses, previous bill statements, and previous payment amounts for many clients. It also contains a column (`default.payment.next.month`) specifying whether the client will pay their credit card on time next month or not.

Problem

The problem we are trying to solve is predicting the target `default.payment.next.month` column (whether a client will pay on time or not) based on the rest of the data for a particular client. A value of '1' indicates that the user defaulted their payment (i.e did not pay on time).

Columns

1. Our initial thoughts are that the columns `PAY_0`, `PAY_2`, `PAY_3`, etc. will be very important in predicting our target. This is because this data shows the user's previous payment behavior, and it is likely that a client would show similar

behaviour in the future.
2. The column `EDUCATION` could also be important. This is because a higher education status (e.g. university, graduate school) could indicate that a client could be more capable of being able to make their payment (i.e higher paying job with more stable monthly/yearly income).
3. `SEX` should potentially not be an important feature. We should be sensitive to this feature, and may potentially need to remove it to avoid introducing any bias or unfairness in our prediction.
4. The `BILL_AMT*` and `PAY_AMT*` columns could potentially have some interaction effects and reveal trends leading up to the 'next month' we are interested in. For example, if there is an increase across the `BILL_AMT*` columns (a client needs to pay more each month) but a decrease across the `PAY_AMT*` columns, it could be a hint that it would be difficult for the client to make next month's payment.

1.2 Download dataset and read it as a pandas dataframe

In [3]:
```
df = pd.read_csv('data/UCI_Credit_Card.csv')
```

1.3 Preliminary preprocessing

We did not remove any NaN values at this stage (?why?), but we renamed two columns:

1. `PAY_0` was renamed to `PAY_1` to be more consistent with the rest of the `PAY_*` columns.
2. `default.payment.next.month` was renamed to `target` to be easier to refer to, and so that we are clear which column we are predicting.

In [4]:
```
df = df.rename(columns={'default.payment.next.month': 'target', 'PAY_0':'PAY_1'})
```

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

In [5]:
```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=123)
```

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

3.1 Explanation for each analysis

Please refer to the description above each EDA analysis.

3.2 Summary of EDA.

1. From the output of `train_df.info()`, it can be seen that most of the columns are numeric with datatypes being either int64 or float64, and that there are no null values in the dataframe.
2. From `value_counts` on the `target` column, it can be seen that there is a quite severe class imbalance, where about 78% of the target values are 0 while only 22% are 1. This indicates that there are a lot more negative cases than positive cases and we will need to address the class imbalance issue when training our models.
3. The histogram indicates that the distribution for target = 1 is centred at around mid-20s, while that of target = 0 is centred at around late-20s, indicating that age might have an association with the default payment. In addition, it can be seen that `BILL_AMT` is also associated with the target.
4. The correlation heat map suggests that there is a relatively strong correlation between `PAY_AMT` and `BILL_AMT`, and this is expected since that the payment amount of a credit card is related to the bill amount in reality. This suggests that there might potentially be multicollinearity in the dataset, and we might need to do some feature engineering and feature selection to address this problem.

3.3 Metric selection

1. In this case study, we will classify the clients who have a default payment as "positive" (i.e., when `target` = 1), and those who do not have a default payment as "negative" (i.e., when `target` = 0).
2. Since there is class imbalance in the dataset, accuracy is not a good metric because the class imbalance will introduce bias in the accuracy, and consequently make accuracy not representative of model performance.
3. In the case of predicting if a client will have a default payment or not, we should not look only at recall or precision. We normally use recall when we care more about false negative cases, and precision when we care more about false positive cases. However, in this case, we need to care about both recall and precision. We do not want recall to be too low since we do not want a client that will have default payment to be classified as negative, because this will result in a consequence that credit cards will be issued to those who are not able to make payment in time, causing a high deficit in credit card issuing companies/banks. On the contrary, if the precision is too low, those clients who have good credit history and are indeed eligible to get approval for credit cards will get rejected.
4. From the aforementioned discussion it can be seen that the best metric here to use will be f1, since f1 is the harmonic mean of both recall and precision, which is suitable for our case where we seek to find a balance between recall and precision.

EDA

The `train_df.head()` function outputs the first few rows of the data, and gives us a high level glance at how the data looks like.

In [6]:
```
train_df.head()
```

Out[6]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL
19682	19683	200000.0	2	2	1	46	0	0	0	0	...	103422.0	95206.0	
11062	11063	120000.0	2	1	1	32	-1	-1	-1	-1	...	476.0	802.0	
197	198	20000.0	2	1	2	22	0	0	0	0	...	8332.0	18868.0	
23620	23621	100000.0	2	5	2	34	0	0	0	0	...	23181.0	7721.0	
26031	26032	290000.0	2	2	2	29	0	0	0	0	...	8770.0	9145.0	

5 rows × 25 columns

`train_df.info()` looks at how many null values there are in each column, and also examines the types of data contained in each column.

In [7]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24000 entries, 19682 to 19966
Data columns (total 25 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   ID         24000 non-null  int64
 1   LIMIT_BAL  24000 non-null  float64
 2   SEX        24000 non-null  int64
 3   EDUCATION  24000 non-null  int64
 4   MARRIAGE   24000 non-null  int64
 5   AGE        24000 non-null  int64
 6   PAY_1      24000 non-null  int64
 7   PAY_2      24000 non-null  int64
 8   PAY_3      24000 non-null  int64
 9   PAY_4      24000 non-null  int64
 10  PAY_5      24000 non-null  int64
 11  PAY_6      24000 non-null  int64
 12  BILL_AMT1  24000 non-null  float64
 13  BILL_AMT2  24000 non-null  float64
 14  BILL_AMT3  24000 non-null  float64
 15  BILL_AMT4  24000 non-null  float64
 16  BILL_AMT5  24000 non-null  float64
 17  BILL_AMT6  24000 non-null  float64
 18  PAY_AMT1   24000 non-null  float64
 19  PAY_AMT2   24000 non-null  float64
 20  PAY_AMT3   24000 non-null  float64
 21  PAY_AMT4   24000 non-null  float64
 22  PAY_AMT5   24000 non-null  float64
 23  PAY_AMT6   24000 non-null  float64
 24  target     24000 non-null  int64
dtypes: float64(13), int64(12)
memory usage: 4.8 MB
```

`train_df.describe()` shows the the statistical details including mean, minimum, maximum, different percentiles, etc.

In [8]:
```
train_df.describe()
```

Out[8]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_
count	24000.000000	24000.000000	24000.000000	24000.000000	24000.000000	24000.000000	24000.000000	24000.00000
mean	14964.174292	167893.486667	1.603125	1.851958	1.553375	35.488458	-0.017542	-0.13529
std	8660.479272	130109.666875	0.489260	0.790560	0.521452	9.217424	1.125331	1.19981
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.00000
25%	7467.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.00000
50%	14975.000000	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.00000
75%	22460.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.00000
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.00000

8 rows × 25 columns

`train_df['target'].value_counts(normalize=True)` shows the normalized value counts for each category in the `target`. We use this function to see if there is any class imbalance in the data.

In [9]:
```
train_df['target'].value_counts(normalize=True)
```

Out[9]:
```
0    0.777833
1    0.222167
Name: target, dtype: float64
```

`train_df['EDUCATION'].value_counts()` looks at the counts of values of each category in the `EDUCATION` column.

In [10]:
```
train_df['EDUCATION'].value_counts()
```

Out[10]:
```
2    11198
1     8494
3     3938
5      229
4       91
6       39
0       11
Name: EDUCATION, dtype: int64
```

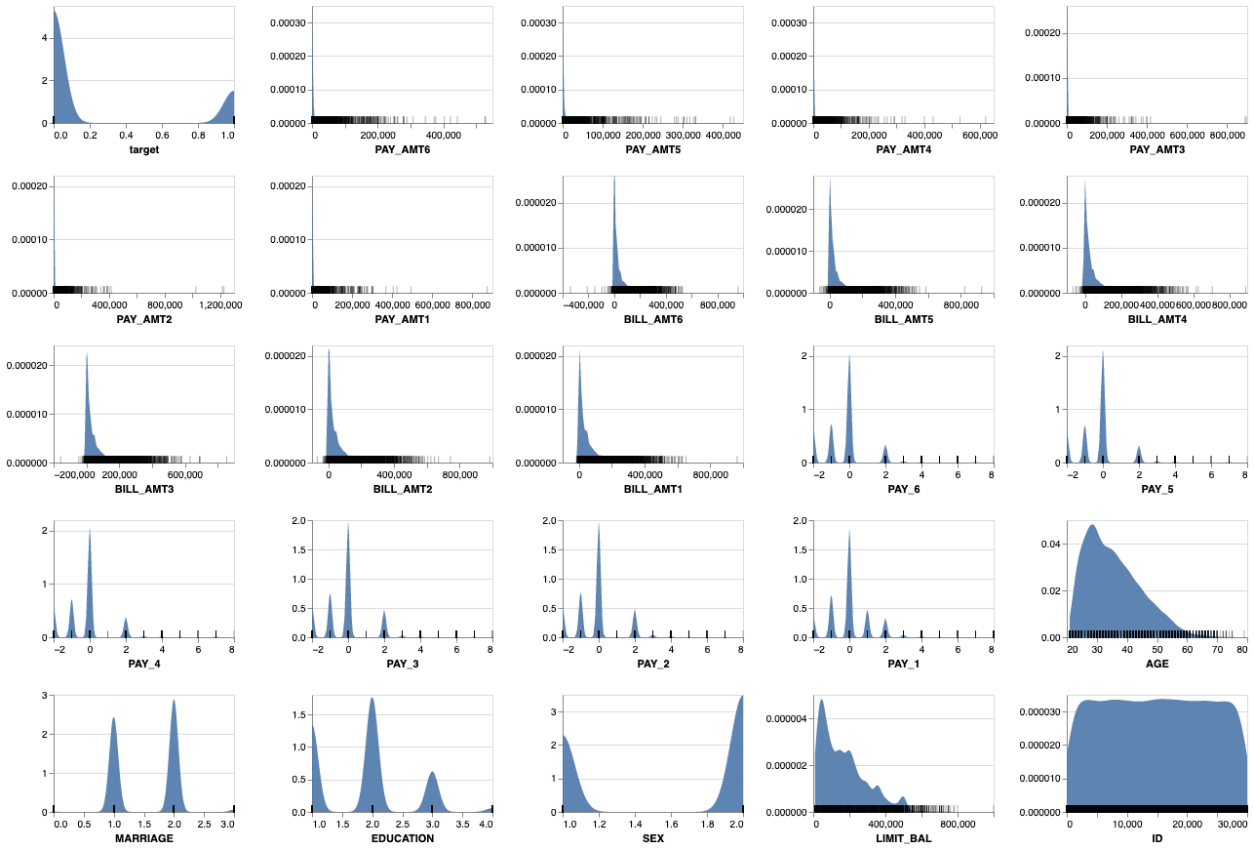
By looking at `train_df['EDUCATION'].value_counts()` as well as the description about each feature if this dataset on Kaggle, we made the following changes to the data: The levels 0, 5 and 6 in EDUCATION were changed to 4, since 4 refers to "others", and 0, 5 and 6 refer to "unknown".

In [11]:
```
train_df['EDUCATION'].replace({0: 4, 5: 4, 6: 4}, inplace=True)
test_df['EDUCATION'].replace({0: 4, 5: 4, 6: 4}, inplace=True)
```

`aly.dist(train_df)` gives us a general idea of how the data in each column is distributed.

In [12]:
```
aly.dist(train_df)
```

Out[12]:

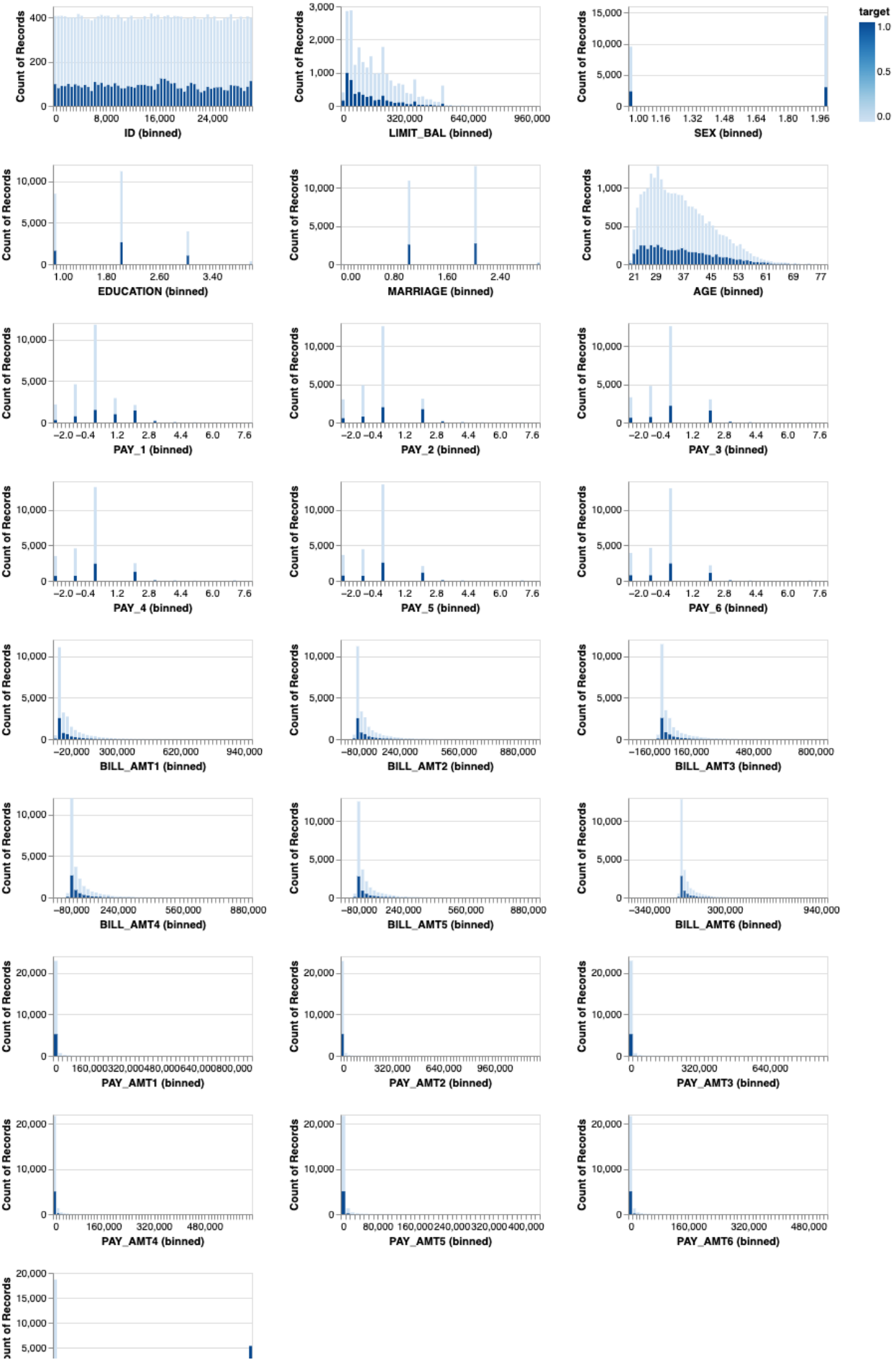


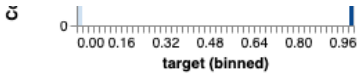
The complied histogram plot shows the shape of distribution of each column, and the histograms are grouped by colours with light blue indicating target = 0 and dark blue indicating target = 1.

In [13]:
```
alt.Chart(train_df, width=200, height=100).mark_bar().encode(
    x=alt.X(alt.repeat(), bin=alt.Bin(maxbins=70)),
    y='count()',
    color='target'
).repeat(
    train_df.select_dtypes('number').nunique().to_frame('count').index.tolist(),
    columns=3
).properties(
    title='Histogram showing the distribution of data in each feature'
)
```

Out[13]:

Histogram showing the distribution of data in each feature





The correlation heatmap shows the correlation coefficients between each pair of variables, and the colour is in accordance with the intensity.

In [14]:
```
train_df.corr('spearman').style.background_gradient()
```

Out[14]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4
ID	1.000000	0.028469	0.021694	0.039952	-0.027409	0.026562	-0.020636	-0.001546	-0.006584	-0.00050
LIMIT_BAL	0.028469	1.000000	0.060169	-0.268531	-0.115429	0.185045	-0.297013	-0.343210	-0.332020	-0.30801
SEX	0.021694	0.060169	1.000000	0.015623	-0.034660	-0.091806	-0.053721	-0.072805	-0.066877	-0.05985
EDUCATION	0.039952	-0.268531	0.015623	1.000000	-0.162214	0.158924	0.138693	0.171794	0.165045	0.15510
MARRIAGE	-0.027409	-0.115429	-0.034660	-0.162214	1.000000	-0.462261	0.019997	0.033960	0.040346	0.03973
AGE	0.026562	0.185045	-0.091806	0.158924	-0.462261	1.000000	-0.056751	-0.078259	-0.079111	-0.07555
PAY_1	-0.020636	-0.297013	-0.053721	0.138693	0.019997	-0.056751	1.000000	0.626059	0.548310	0.51610
PAY_2	-0.001546	-0.343210	-0.072805	0.171794	0.033960	-0.078259	0.626059	1.000000	0.802436	0.71532
PAY_3	-0.006584	-0.332020	-0.066877	0.165045	0.040346	-0.079111	0.548310	0.802436	1.000000	0.80310
PAY_4	-0.000507	-0.308016	-0.059852	0.155109	0.039739	-0.075554	0.516108	0.715323	0.803100	1.00000
PAY_5	-0.014848	-0.281681	-0.050116	0.139510	0.044412	-0.078913	0.484550	0.674924	0.718232	0.82150
PAY_6	-0.005927	-0.261785	-0.038584	0.126788	0.038160	-0.069302	0.461945	0.632789	0.670359	0.72928
BILL_AMT1	0.012719	0.052238	-0.043248	0.099178	-0.001109	0.007471	0.315844	0.572869	0.526818	0.51324
BILL_AMT2	0.012799	0.044509	-0.040394	0.096126	0.001563	0.007098	0.332046	0.554516	0.593251	0.56196
BILL_AMT3	0.019714	0.061552	-0.027774	0.082629	-0.003423	0.006361	0.314524	0.520239	0.560372	0.62009
BILL_AMT4	0.035414	0.075757	-0.021388	0.071644	0.000362	0.001605	0.304603	0.496316	0.531695	0.59147
BILL_AMT5	0.017515	0.081559	-0.012566	0.062884	-0.003043	0.005877	0.299338	0.476632	0.507870	0.55913
BILL_AMT6	0.021229	0.090824	-0.008118	0.058425	-0.001289	0.005530	0.287858	0.457259	0.484343	0.52944
PAY_AMT1	0.014044	0.270676	-0.003166	-0.040144	-0.008185	0.035641	-0.096829	0.022447	0.218532	0.18855
PAY_AMT2	0.054245	0.280822	0.008268	-0.050602	-0.018980	0.043351	-0.064678	0.084354	0.040167	0.24740
PAY_AMT3	0.092890	0.286323	0.022290	-0.041214	-0.014063	0.033097	-0.057278	0.085765	0.103918	0.07049
PAY_AMT4	0.019025	0.283519	0.013667	-0.041627	-0.021571	0.043230	-0.031067	0.095336	0.120145	0.14480
PAY_AMT5	0.011364	0.294502	0.013616	-0.049218	-0.017768	0.039578	-0.028266	0.099057	0.127454	0.16047
PAY_AMT6	0.037639	0.316610	0.036850	-0.053625	-0.021047	0.040332	-0.046358	0.080687	0.094928	0.14037
target	-0.013663	-0.164635	-0.043814	0.046648	-0.025359	0.006196	0.288651	0.215009	0.195484	0.17317

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

Feature engineering is carried out with the aim to add new features that might be useful for the prediction. We add several features based on the original features:

1. Ratio of amount paid of the month to billed amount of the month (BP_R1 to BP_R6 for the six months of data respectively). This feature could potentially be useful because being able to pay a larger proportion of the billed

amount is often associated with better financial status.

- Note that when there is zero division, the value of the ratio is infinity due to the zero denominator, or missing value when it is 0 divided by 0. We decide to code both of these cases to 1 for the reason that zero billed amount (not spending any money with the credit card) could be an indication that the person has enough money else where and has the ability to pay the full bill.

2. Ratio of billed amount of the month to the amount of given credit (CB_1 to CB_6). This feature could possibly help us understand the user's spending habits and whether the person is close to the limit of their credit limit.

Since these features are numeric variables, we will apply standard scaler to the variables in the preprocessing pipeline.

Note: We also tried out a model with two other features, with the aim to reduce redundancy in the features that are highly correlated with each other. Although tree-based models can handle correlated features pretty well, we want to explore if the model performs better when there are less potentially redundant features. The two features are:

- average of all six month's ratios of billed amount of the month to the amount of given credit(CB_1 to CB_6)
- average of all six months' billed amount (BILL_AMT1to BILL_AMT6)

We tested out the model with CB_1 to CB_6 and BILL_AMT1to BILL_AMT6 removed when including these two added features. However, the model did not perfomed better (in fact, with slight decrease in performance, thus we did not continue using the aggregated features.) (This exploration is done in Q11)

We continue analysis with keeping CB_1 to CB_6 and BILL_AMT1to BILL_AMT6 separated.

In [15]:
```
train_df['BP_R1'] = train_df['PAY_AMT1']/train_df['BILL_AMT1']
train_df['BP_R2'] = train_df['PAY_AMT2']/train_df['BILL_AMT2']
train_df['BP_R3'] = train_df['PAY_AMT3']/train_df['BILL_AMT3']
train_df['BP_R4'] = train_df['PAY_AMT4']/train_df['BILL_AMT4']
train_df['BP_R5'] = train_df['PAY_AMT5']/train_df['BILL_AMT5']
train_df['BP_R6'] = train_df['PAY_AMT6']/train_df['BILL_AMT6']

test_df['BP_R1'] = test_df['PAY_AMT1']/test_df['BILL_AMT1']
test_df['BP_R2'] = test_df['PAY_AMT2']/test_df['BILL_AMT2']
test_df['BP_R3'] = test_df['PAY_AMT3']/test_df['BILL_AMT3']
test_df['BP_R4'] = test_df['PAY_AMT4']/test_df['BILL_AMT4']
test_df['BP_R5'] = test_df['PAY_AMT5']/test_df['BILL_AMT5']
test_df['BP_R6'] = test_df['PAY_AMT6']/test_df['BILL_AMT6']
```

In [16]:
```
train_df['CB_1'] = train_df['BILL_AMT1']/train_df['LIMIT_BAL']
train_df['CB_2'] = train_df['BILL_AMT2']/train_df['LIMIT_BAL']
train_df['CB_3'] = train_df['BILL_AMT3']/train_df['LIMIT_BAL']
train_df['CB_4'] = train_df['BILL_AMT4']/train_df['LIMIT_BAL']
train_df['CB_5'] = train_df['BILL_AMT5']/train_df['LIMIT_BAL']
train_df['CB_6'] = train_df['BILL_AMT6']/train_df['LIMIT_BAL']

test_df['CB_1'] = test_df['BILL_AMT1']/test_df['LIMIT_BAL']
test_df['CB_2'] = test_df['BILL_AMT2']/test_df['LIMIT_BAL']
test_df['CB_3'] = test_df['BILL_AMT3']/test_df['LIMIT_BAL']
test_df['CB_4'] = test_df['BILL_AMT4']/test_df['LIMIT_BAL']
test_df['CB_5'] = test_df['BILL_AMT5']/test_df['LIMIT_BAL']
test_df['CB_6'] = test_df['BILL_AMT6']/test_df['LIMIT_BAL']
```

In [17]:
```
train_df.replace([np.inf, -np.inf, np.nan], 1, inplace=True)
test_df.replace([np.inf, -np.inf, np.nan], 1, inplace=True)

train_df.describe()
```

Out[17]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_
count	24000.000000	24000.000000	24000.000000	24000.000000	24000.000000	24000.000000	24000.000000	24000.00000
mean	14964.174292	167893.486667	1.603125	1.841000	1.553375	35.488458	-0.017542	-0.13529
std	8660.479272	130109.666875	0.489260	0.744586	0.521452	9.217424	1.125331	1.1998
min	1.000000	10000.000000	1.000000	1.000000	0.000000	21.000000	-2.000000	-2.00000
25%	7467.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.00000
50%	14975.000000	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.00000
75%	22460.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.00000
max	30000.000000	1000000.000000	2.000000	4.000000	3.000000	79.000000	8.000000	8.00000

8 rows × 37 columns

In [18]:
```
train_df.corr('spearman').style.background_gradient()
```

Out[18]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_
ID	1.000000	0.028469	0.021694	0.039952	-0.027409	0.026562	-0.020636	-0.001546	-0.006584	-0.00050
LIMIT_BAL	0.028469	1.000000	0.060169	-0.268531	-0.115429	0.185045	-0.297013	-0.343210	-0.332020	-0.30801
SEX	0.021694	0.060169	1.000000	0.015623	-0.034660	-0.091806	-0.053721	-0.072805	-0.066877	-0.05985
EDUCATION	0.039952	-0.268531	0.015623	1.000000	-0.162214	0.158924	0.138693	0.171794	0.165045	0.15510
MARRIAGE	-0.027409	-0.115429	-0.034660	-0.162214	1.000000	-0.462261	0.019997	0.033960	0.040346	0.03973
AGE	0.026562	0.185045	-0.091806	0.158924	-0.462261	1.000000	-0.056751	-0.078259	-0.079111	-0.07555
PAY_1	-0.020636	-0.297013	-0.053721	0.138693	0.019997	-0.056751	1.000000	0.626059	0.548310	0.51610
PAY_2	-0.001546	-0.343210	-0.072805	0.171794	0.033960	-0.078259	0.626059	1.000000	0.802436	0.71532
PAY_3	-0.006584	-0.332020	-0.066877	0.165045	0.040346	-0.079111	0.548310	0.802436	1.000000	0.80310
PAY_4	-0.000507	-0.308016	-0.059852	0.155109	0.039739	-0.075554	0.516108	0.715323	0.803100	1.00000
PAY_5	-0.014848	-0.281681	-0.050116	0.139510	0.044412	-0.078913	0.484550	0.674924	0.718232	0.82150
PAY_6	-0.005927	-0.261785	-0.038584	0.126788	0.038160	-0.069302	0.461945	0.632789	0.670359	0.72928
BILL_AMT1	0.012719	0.052238	-0.043248	0.099178	-0.001109	0.007471	0.315844	0.572869	0.526818	0.51324
BILL_AMT2	0.012799	0.044509	-0.040394	0.096126	0.001563	0.007098	0.332046	0.554516	0.593251	0.56196
BILL_AMT3	0.019714	0.061552	-0.027774	0.082629	-0.003423	0.006361	0.314524	0.520239	0.560372	0.62009
BILL_AMT4	0.035414	0.075757	-0.021388	0.071644	0.000362	0.001605	0.304603	0.496316	0.531695	0.59147
BILL_AMT5	0.017515	0.081559	-0.012566	0.062884	-0.003043	0.005877	0.299338	0.476632	0.507870	0.55913
BILL_AMT6	0.021229	0.090824	-0.008118	0.058425	-0.001289	0.005530	0.287858	0.457259	0.484343	0.52944
PAY_AMT1	0.014044	0.270676	-0.003166	-0.040144	-0.008185	0.035641	-0.096829	0.022447	0.218532	0.18855
PAY_AMT2	0.054245	0.280822	0.008268	-0.050602	-0.018980	0.043351	-0.064678	0.084354	0.040167	0.24740
PAY_AMT3	0.092890	0.286323	0.022290	-0.041214	-0.014063	0.033097	-0.057278	0.085765	0.103918	0.07049
PAY_AMT4	0.019025	0.283519	0.013667	-0.041627	-0.021571	0.043230	-0.031067	0.095336	0.120145	0.14480
PAY_AMT5	0.011364	0.294502	0.013616	-0.049218	-0.017768	0.039578	-0.028266	0.099057	0.127454	0.16047
PAY_AMT6	0.037639	0.316610	0.036850	-0.053625	-0.021047	0.040332	-0.046358	0.080687	0.094928	0.14037
target	-0.013663	-0.164635	-0.043814	0.046648	-0.025359	0.006196	0.288651	0.215009	0.195484	0.17317
BP_R1	-0.008219	0.117758	0.031160	-0.103520	-0.008176	0.011526	-0.341986	-0.521494	-0.296934	-0.31091
BP_R2	0.029790	0.133150	0.046799	-0.110664	-0.023944	0.019019	-0.331470	-0.427147	-0.549934	-0.33038
BP_R3	0.076862	0.149876	0.055256	-0.088193	-0.018959	0.016932	-0.299097	-0.380119	-0.414094	-0.54440
BP_R4	-0.028111	0.155960	0.033628	-0.093195	-0.030365	0.035941	-0.262861	-0.350992	-0.370738	-0.41159
BP_R5	-0.018979	0.148227	0.020956	-0.088726	-0.015221	0.025350	-0.272616	-0.352656	-0.367529	-0.39680
BP_R6	0.010034	0.154370	0.040790	-0.085599	-0.026771	0.032347	-0.279888	-0.353806	-0.378374	-0.39165
CB_1	0.000709	-0.410179	-0.080434	0.212252	0.049075	-0.061544	0.403740	0.655760	0.598993	0.57244
CB_2	0.002124	-0.409644	-0.076982	0.205954	0.048675	-0.063487	0.419934	0.634627	0.667882	0.62127
CB_3	0.013974	-0.394171	-0.064586	0.193096	0.043631	-0.064616	0.409693	0.608539	0.641846	0.69033
CB_4	0.031712	-0.383270	-0.058852	0.182106	0.048224	-0.074041	0.405685	0.592597	0.622240	0.66892
CB_5	0.005787	-0.360711	-0.047691	0.164679	0.043599	-0.067185	0.399491	0.571930	0.598837	0.64004
CB_6	0.010689	-0.320880	-0.036772	0.151747	0.041570	-0.062852	0.384160	0.549088	0.572100	0.60995

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

5.1 Identify feature types and transformations

Please see the table below for a description of which transformation will be applied on each feature.

Feature	Transformation	Explanation
ID	drop	A categoric feature with no missing values. I will drop this column because every ID is unique.
LIMIT_BAL	scaling	A numeric feature with no missing values. It represents the amount of given credit in NT dollars for an entry. It will be a good idea to apply scaling, as the range of values (10000 to 1000000).
SEX	one-hot encoding with "binary = True"	A binarial categoric feature with no missing values. It represents whether entry is male or female. We should apply binarial one-hot encoding.
EDUCATION	ordinal encoding	An ordinal feature with no missing values. It represents the education level of an entry about 1=graduate school, 2=university, 3=high school, 4=others. We should apply ordinal encoding.
MARRIAGE	one-hot encoding	A categoric feature with no missing values. It represents the marriage status of an entry about 1=married, 2=single, 3=others.
AGE	scaling	A numeric feature with no missing values. It represents the age of an entry. It will be a good idea to apply scaling, as the range of values (21 to 79).
PAY_1	ordinal encoding	An ordinal feature with no missing values. It represents the Repayment status of an entry in first period about -2=no credit to pay, -1=pay on time, 0=pay on time but not full amount, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months and above. We should apply ordinal encoding.
PAY_2	ordinal encoding	An ordinal feature with no missing values. It represents the Repayment status of an entry in second period about -2=no credit to pay, -1=pay on time, 0=pay on time but not full amount, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months and above. We should apply ordinal encoding.
PAY_3	ordinal encoding	An ordinal feature with no missing values. It represents the Repayment status of an entry in third period about -2=no credit to pay, -1=pay on time, 0=pay on time but not full amount, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months and above. We should apply ordinal encoding.
PAY_4	ordinal encoding	An ordinal feature with no missing values. It represents the Repayment status of an entry in fourth period about -2=no credit to pay, -1=pay on time, 0=pay on time but not full amount, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months and above. We should apply ordinal encoding.
PAY_5	ordinal encoding	An ordinal feature with no missing values. It represents the Repayment status of an entry in fifth period about -2=no credit to pay, -1=pay on time, 0=pay on time but not full amount, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months and above. We should apply ordinal encoding.
PAY_6	ordinal encoding	An ordinal feature with no missing values. It represents the Repayment status of an entry in sixth period about -2=no credit to pay, -1=pay on time, 0=pay on time but not full amount, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months and above. We should apply ordinal encoding.
BILL_AMT1	scaling	A numeric feature with no missing values. It represents the amount of bill statement in the first period of an entry. It will be a good idea to apply scaling, as the range of values (-15308 to 964511).
BILL_AMT2	scaling	A numeric feature with no missing values. It represents the amount of bill statement in the second period of an entry. It will be a good idea to apply scaling, as the range of values (-67526 to 983931).
BILL_AMT3	scaling	A numeric feature with no missing values. It represents the amount of bill statement in the third period of an entry. It will be a good idea to apply scaling, as the range of values (-157264 to 855086).
BILL_AMT4	scaling	A numeric feature with no missing values. It represents the amount of bill statement in the fourth period of an entry. It will be a good idea to apply scaling, as the range of values (-65167 to 891586).
BILL_AMT5	scaling	A numeric feature with no missing values. It represents the amount of bill statement in the fifth period of an entry. It will be a good idea to apply scaling, as the range of values (-61372 to 927171).
BILL_AMT6	scaling	A numeric feature with no missing values. It represents the amount of bill statement in the fifth period of an entry. It will be a good idea to apply scaling, as the range of values (-339603 to 961664).
PAY_AMT1	scaling	A numeric feature with no missing values. It represents the amount of previous payment in the first period of an entry. It will be a good idea to apply scaling, as the range of values (0 to 873552).
PAY_AMT2	scaling	A numeric feature with no missing values. It represents the amount of previous payment in the second period of an entry. It will be a good idea to apply scaling, as the range of values (0 to 1227082).
PAY_AMT3	scaling	A numeric feature with no missing values. It represents the amount of previous payment in the third period of an entry. It will be a good idea to apply scaling, as the range of values (0 to 896040).

Feature	Transformation	Explanation
PAY_AMT4	scaling	A numeric feature with no missing values. It represents the amount of previous payment in the fourth period of an entry. It will be a good idea to apply scaling, as the range of values (0 to 621000).
PAY_AMT5	scaling	A numeric feature with no missing values. It represents the amount of previous payment in the fifth period of an entry. It will be a good idea to apply scaling, as the range of values (0 to 426529).
PAY_AMT6	scaling	A numeric feature with no missing values. It represents the amount of previous payment in the sixth period of an entry. It will be a good idea to apply scaling, as the range of values (0 to 528666).
BP_R1	scaling	A numeric feature with no missing values. It represents the rate of the previous payment to the bill amount in the first period of an entry. It will be a good idea to apply scaling, as the range of values (-8706.60 to 11453.666667).
BP_R2	scaling	A numeric feature with no missing values. It represents the rate of the previous payment to the bill amount in the second period of an entry. It will be a good idea to apply scaling, as the range of values (-10259.50 to 4444.33).
BP_R3	scaling	A numeric feature with no missing values. It represents the rate of the previous payment to the bill amount in the third period of an entry. It will be a good idea to apply scaling, as the range of values (-82150 to 6333.33).
BP_R4	scaling	A numeric feature with no missing values. It represents the rate of the previous payment to the bill amount in the fourth period of an entry. It will be a good idea to apply scaling, as the range of values (-17266.667 to 8891.357).
BP_R5	scaling	A numeric feature with no missing values. It represents the rate of the previous payment to the bill amount in the fifth period of an entry. It will be a good idea to apply scaling, as the range of values (-4077.25 to 1738.42).
BP_R6	scaling	A numeric feature with no missing values. It represents the rate of the previous payment to the bill amount in the sixth period of an entry. It will be a good idea to apply scaling, as the range of values (-11349.57 to 2643.33).
CB_1	scaling	A numeric feature with no missing values. It represents the rate of the bill amount to the amount of given credit in the first period of an entry. It will be a good idea to apply scaling, as the range of values (-0.2309 to 6.4553).
CB_2	scaling	A numeric feature with no missing values. It represents the rate of the bill amount to the amount of given credit in the second period of an entry. It will be a good idea to apply scaling, as the range of values (-0.965 to 6.3805).
CB_3	scaling	A numeric feature with no missing values. It represents the rate of the bill amount to the amount of given credit in the third period of an entry. It will be a good idea to apply scaling, as the range of values (-0.925 to 10.689).
CB_4	scaling	A numeric feature with no missing values. It represents the rate of the bill amount to the amount of given credit in the fourth period of an entry. It will be a good idea to apply scaling, as the range of values (-1.375 to 5.147).
CB_5	scaling	A numeric feature with no missing values. It represents the rate of the bill amount to the amount of given credit in the fifth period of an entry. It will be a good idea to apply scaling, as the range of values (-0.877 to 4.936).
CB_6	scaling	A numeric feature with no missing values. It represents the rate of the bill amount to the amount of given credit in the sixth period of an entry. It will be a good idea to apply scaling, as the range of values (-1.51 to 3.886).

5.2 Define a column transformer

The following code drops our target column, creates lists of different column types, and creates a column transformer to preprocess the columns. It also transforms our training (X_train) and displays what the transformed columns look like.

In [19]:
```
X_train = train_df.drop(columns='target')
y_train = train_df['target']
X_test = test_df.drop(columns='target')
y_test = test_df['target']
```

In [20]:
```
edu_feat = ['EDUCATION']
pay_feat = ['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
binary_feat = ['SEX']
cat_feat = ['MARRIAGE']
drop_feat = ['ID']
num_feat =[
'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',
'BP_R1',
'BP_R2',
'BP_R3',
'BP_R4',
'BP_R5',
'BP_R6',
'CB_1',
'CB_2',
'CB_3',
'CB_4',
'CB_5',
'CB_6'
]
```

In [21]:

```
pay_order = [-2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8]

preprocessor = make_column_transformer(
    (StandardScaler(),num_feat),
    (OneHotEncoder(drop='if_binary', dtype = int), binary_feat),
    (OneHotEncoder(handle_unknown="ignore", sparse=False, dtype = int), cat_feat),
    (OrdinalEncoder(dtype=int), edu_feat),
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    ("drop", drop_feat)
)

preprocessor
```

Out[21]:

ColumnTransformer

standardscaler	**onehotencoder-1**	**onehotencoder-2**	**ordinalencoder-1**	**ordinalencoder-2**	**ordinalencoder-3**	**ord**
StandardScaler	OneHotEncoder	OneHotEncoder	OrdinalEncoder	OrdinalEncoder	OrdinalEncoder	Orc

In [22]:

```
transformed = preprocessor.fit_transform(X_train)
column_names = (
    num_feat
    + binary_feat
    + preprocessor.named_transformers_["onehotencoder-2"].get_feature_names_out().tolist()
    + edu_feat
    + pay_feat
)

X_train_enc = pd.DataFrame(transformed, columns=column_names)

X_train_enc
```

Out[22]:

	LIMIT_BAL	AGE	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT
0	0.246770	1.140423	0.757746	0.761414	0.830643	0.929712	0.901952	0.444775	-0.117221	-0.0426
1	-0.368109	-0.378471	-0.647736	-0.684308	-0.677352	-0.664583	-0.648784	-0.646675	-0.298634	-0.2616
2	-1.136707	-1.463396	-0.443283	-0.415538	-0.405142	-0.542919	-0.352021	-0.327893	-0.248029	-0.2285
3	-0.521828	-0.161486	0.553566	0.595190	0.585249	-0.312957	-0.535129	-0.597934	-0.038927	-0.0983
4	0.938509	-0.703949	-0.596367	-0.573988	-0.560226	-0.536136	-0.511737	-0.483417	-0.270109	-0.2065
...	...	...	...	...	...	...	...	...	...	
23995	1.630247	0.163991	-0.513329	-0.486736	-0.489221	-0.450634	-0.399971	-0.390079	-0.265693	-0.2159
23996	1.476528	2.008362	-0.696214	-0.688890	-0.682087	-0.631999	-0.619578	-0.535225	-0.337542	-0.2768
23997	-0.906127	-0.703949	-0.052824	-0.008798	-0.002588	-0.061468	-0.517010	-0.500383	-0.218072	-0.1831
23998	-1.059847	-1.571888	-0.308918	-0.287844	-0.254504	-0.194329	-0.178458	-0.373079	-0.218192	-0.1831
23999	1.553387	0.055498	-0.658206	-0.326360	-0.638407	-0.606198	-0.600572	-0.613585	1.201721	-0.1360

24000 rows × 38 columns

6. Baseline model

rubric={accuracy}

Your tasks:

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

Points: 2

6.1 Performance of the baseline model

The baseline model `DummyClassifier` scored 0 for recall, precision and f1, which is expected since there is a severe class imbalance in the data. Since there are significantly more negative cases than positive cases, the model will predict negative, i.e. `target` = 0 for all the examples. Therefore, there will be 0 true positives in prediction, consequently making recall and precision to be 0, since $recall = \frac{TP}{TP+FN}$ and $precision = \frac{TP}{TP+FP}$. The f1 score is also 0 because f1 is the harmonic mean of recall and precision. Although the baseline model gave a reasonable accuracy score of 0.778, we should not trust this score due to reasons stated above. Overall, the performance of the baseline model is not good, but the low scores also further imply that we should address the class imbalance in the dataset.

In [23]:
```
classification_metrics = ["accuracy", "precision", "recall", "f1"]

dc = DummyClassifier(strategy="most_frequent", random_state=123)

cross_val_results = {}

dummy_result = cross_validate(dc, X_train, y_train, cv=5,return_train_score=True, scoring=classification_

pd.DataFrame(dummy_result)
```

```
/Users/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1
334: 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/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1
334: 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/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1
334: 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/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1
334: 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/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1
334: 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/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1
334: 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/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1
334: 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/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1
334: 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/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1
334: 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/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1
334: 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))
```

Out[23]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision	test_recall	train_recall	test_f1	train_
0	0.002217	0.009050	0.777917	0.777813	0.0	0.0	0.0	0.0	0.0	0
1	0.002054	0.005214	0.777917	0.777813	0.0	0.0	0.0	0.0	0.0	0
2	0.001665	0.003082	0.777917	0.777813	0.0	0.0	0.0	0.0	0.0	0
3	0.001849	0.003092	0.777708	0.777865	0.0	0.0	0.0	0.0	0.0	0
4	0.002158	0.003319	0.777708	0.777865	0.0	0.0	0.0	0.0	0.0	0

In [24]:

```
cross_val_results['dummy'] = pd.DataFrame(dummy_result).agg(['mean', 'std']).round(3).T

cross_val_results['dummy']
```

Out[24]:

	mean	std
fit_time	0.002	0.000
score_time	0.005	0.003
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

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.

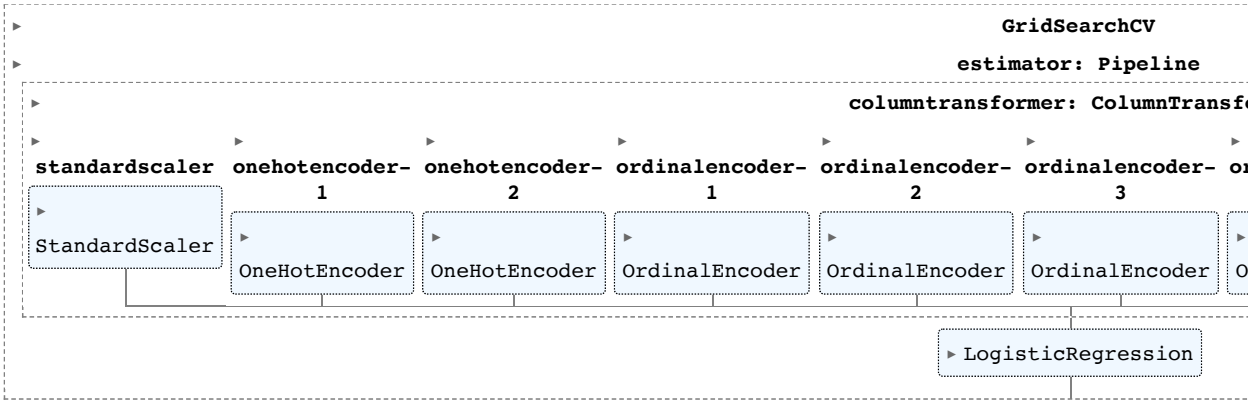
Points: 8

7.4 Summary of results on the linear model

1. From hyperparameter tuning on `LogisticRegression` model using `GridSearchCV` on the regularization parameter `C`, the best `C` is found to be 10. Since the default hyperparameter `C` of `LogisticRegression` is 1, the `C` value calculated by `GridSearchCV` is larger than default, suggesting that the model is applying less penalty than default to the coefficients.
2. Using the optimal hyperparameter suggested by `GridSearchCV`, the 10-fold cross-validation on the training data produced a training accuracy of 0.809 and a CV accuracy of 0.809. The training and validation scores for all the other metrics are also very consistent, suggesting that the model is not suffering from overfitting.
3. It can be seen that the model shows a high validation accuracy of 0.809 and a relatively high precision score of 0.712, while a low recall of 0.237, which again suggests that there is the class imbalance issue that we need to address. Since there are a lot more negatives than positives in the data, the number of false negative predictions is large and thus a low recall score is resulted.
4. The low training f1 score of 0.355 and validation f1 score of 0.356 is also a result of low recall.
5. Overall, the cross-validation results using the hyperparameter found by `GridSearchCV` shows that this model does not display ideal performance. For next steps, we should consider taking `class_weight` into account in hyperparameter tuning in hopes of alleviating class imbalance problem.

In [25]:
```
param_tun = {"logisticregression__C": [10**-2,10**-1,10**0,10**1,10**2]}

logreg = make_pipeline(preprocessor, LogisticRegression(max_iter=1000, random_state=123))

grid_search = GridSearchCV(logreg, param_tun, cv=5, scoring=classification_metrics, refit='f1')

grid_search.fit(X_train, y_train)
```

Out[25]:



In [26]:
```
grid_search.best_params_
```

Out[26]:
```
{'logisticregression__C': 10}
```

In [27]:
```
logreg_tun = make_pipeline(preprocessor, LogisticRegression(C=10, max_iter=1000, random_state=123))

logreg_result = cross_validate(logreg_tun, X_train, y_train, cv=5, return_train_score=True, scoring=class
```

```
pd.DataFrame(logreg_result)
```

Out[27]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision	test_recall	train_recall	test_f1	tra
0	0.260943	0.011697	0.806250	0.811094	0.674359	0.715152	0.246717	0.248945	0.361264	0.36
1	0.233588	0.013434	0.811875	0.809635	0.732194	0.708817	0.241088	0.243085	0.362738	0.3(
2	0.161202	0.013309	0.804792	0.810104	0.685879	0.718927	0.223265	0.238631	0.336872	0.35
3	0.162673	0.008459	0.815625	0.806458	0.743316	0.707797	0.260544	0.219226	0.385843	0.33
4	0.149442	0.011945	0.807083	0.808750	0.722397	0.712241	0.214620	0.233294	0.330925	0.35

In [28]:
```
cross_val_results['logreg'] = pd.DataFrame(logreg_result).agg(['mean', 'std']).round(3).T

cross_val_results['logreg']
```

Out[28]:

	mean	std
fit_time	0.194	0.050
score_time	0.012	0.002
test_accuracy	0.809	0.005
train_accuracy	0.809	0.002
test_precision	0.712	0.030
train_precision	0.713	0.005
test_recall	0.237	0.018
train_recall	0.237	0.011
test_f1	0.356	0.022
train_f1	0.355	0.013

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

8.1 Try out three other models

We tried the following models:

1. Decision Tree Classifier
2. Random Forest Classifier
3. K-Nearest Neighbors Classifier
4. XGBoost Classifier

8.2 Summarize results

Overfitting

- The decision tree, the random forest, and xgboost classifiers are all heavily overfitting. The decision tree classifier has very high mean cross-validation training scores for accuracy (0.999), precision (0.999), recall (0.998), and f1 (0.999), but low mean test scores for accuracy (0.727), precision (0.643), recall (0.407), and f1 (0.398). Similar trends are seen for random forest and xgboost (although xgboost has a farily high test accuracy of 0.812). This indicates that these

classifiers are memorizing the specific trends in the training data that may not exist in unseen data. Thus, they are not generalizing well to the test data.
- The k-nearest neighbors classifier on the other hand is not overfitting. It has similar mean training and test cross-validation scores for accuracy (0.843 and 0.793 respectively), precision (0.727 and 0.553), recall (0.467 and 0.358), and f1 (0.568 and 0.434). As the training scores are not very high, this indicates that the model is not overfitting to the training data.

Underfitting

- The decision tree, random forest, and xgboost classifiers do not appear to be underfitting to the training data. As previously discussed, the training scores for these models are quite high, indicating that they are describing the trends in the training data well.
- However, the k-nearest neighbors classifier appears to be underfitting. It has relatively lower mean training cross-validation scores for precision (0.727), recall (0.467), and f1 (0.568) when compared to the other two classifiers. This could indicate that the model is too simple, and is not learning the trends in the training data well.

Fit and score times

- The k-nearest neighbors classifier fits the fastest, followed by decision trees. The random forest classifier and xgboost classifiers are much slower to fit than the other two classifiers.
- The decision tree classifier has the fastest mean cross-validation score time, followed by the random forest and xgboost classifiers. K-nearest neighbors is the slowest.

Comparison to linear model

- Unlike decision tree, random forest, and xgboost, the logistic regression model does not appear to overfit. It has similar mean train and test cross-validation scores for accuracy (0.809 and 0.809 respectively), precision (0.713 and 0.712), recall (0.237 and 0.237), and f1 (0.355 and 0.356).
- Similar to k-nearest neighbors, the logistic regression appears to have lower scores for precision, recall, and f1, indicating possible underfitting.
- The logistic regression model is not as fast at fitting the data as k-nearest neighbours (0.537), but is faster than the other non-linear classifiers.
- The logistic regression is not as fast at scoring as the decision tree (0.050), but is faster than random forest, xgboost, and knn.
- Overall, the logistic regression model beats all the non-linear models in terms of precision. However, all non-linear models beat logistic regression in terms of accuracy, recall, and f1. The logistic regression also has a good balance of fast fit and scoring times.

In [29]:
```
# a) cross-validate on a decision tree classifier

dt = make_pipeline(preprocessor, DecisionTreeClassifier(random_state=123))

dt_result = cross_validate(dt, X_train, y_train, cv=5, return_train_score=True, scoring=classification_me

pd.DataFrame(dt_result)
```

Out[29]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision	test_recall	train_recall	test_f1	tr
0	0.679605	0.012614	0.726042	0.999271	0.386921	1.0	0.399625	0.996718	0.393170	0.9
1	0.648202	0.017710	0.725417	0.999583	0.385455	1.0	0.397749	0.998125	0.391505	0.9
2	0.616057	0.011747	0.728125	0.999531	0.393778	1.0	0.415572	0.997890	0.404382	0.9
3	0.566953	0.009956	0.725833	0.999323	0.391833	1.0	0.422680	0.996952	0.406673	0.9
4	0.579025	0.010020	0.730417	0.999479	0.394419	1.0	0.397376	0.997655	0.395892	0.9

In [30]:
```
# save results to the final dataframe

cross_val_results['dt'] = pd.DataFrame(dt_result).agg(['mean', 'std']).round(3).T

cross_val_results['dt']
```

Out[30]:

	mean	std
fit_time	0.618	0.047
score_time	0.012	0.003
test_accuracy	0.727	0.002
train_accuracy	0.999	0.000
test_precision	0.390	0.004
train_precision	1.000	0.000
test_recall	0.407	0.012
train_recall	0.997	0.001
test_f1	0.398	0.007
train_f1	0.999	0.000

In [31]:
```
# b) cross-validate on a random forest classifier

rf = make_pipeline(preprocessor, RandomForestClassifier(random_state=123))

rf_result = cross_validate(rf, X_train, y_train, cv=5, return_train_score=True, scoring=classification_me

pd.DataFrame(rf_result)
```

Out[31]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision	test_recall	train_recall	test_f1	tr
0	7.461219	0.098412	0.810417	0.999271	0.626214	0.999061	0.363039	0.997656	0.459620	0.9
1	8.122867	0.238649	0.813750	0.999583	0.636943	1.000000	0.375235	0.998125	0.472255	0.9
2	6.246757	0.091701	0.810000	0.999531	0.634615	0.999765	0.340525	0.998125	0.443223	0.9
3	8.645878	0.562277	0.822917	0.999271	0.665649	0.999061	0.408622	0.997655	0.506388	0.9
4	7.876358	0.651074	0.813750	0.999479	0.652557	0.999062	0.346767	0.998593	0.452876	0.9

In [32]:
```
# save results to the final dataframe

cross_val_results['rf'] = pd.DataFrame(rf_result).agg(['mean', 'std']).round(3).T

cross_val_results['rf']
```

Out[32]:

	mean	std
fit_time	7.671	0.904
score_time	0.328	0.263
test_accuracy	0.814	0.005
train_accuracy	0.999	0.000
test_precision	0.643	0.016
train_precision	0.999	0.000
test_recall	0.367	0.027
train_recall	0.998	0.000
test_f1	0.467	0.024
train_f1	0.999	0.000

In [33]:
```
# c) cross-validate on a k-nearest neighbors classifier

knn = make_pipeline(preprocessor, KNeighborsClassifier())

knn_result = cross_validate(knn, X_train, y_train, cv=5, return_train_score=True, scoring=classification_

pd.DataFrame(knn_result)
```

Out[33]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision	test_recall	train_recall	test_f1	tra
0	0.128867	0.433721	0.785000	0.842083	0.525298	0.724038	0.331144	0.467417	0.406214	0.56
1	0.033914	0.176467	0.792917	0.842865	0.550562	0.726022	0.367730	0.470230	0.440945	0.57
2	0.069740	0.175004	0.790000	0.842917	0.542151	0.725958	0.349906	0.470699	0.425314	0.5
3	0.348217	0.374388	0.798750	0.842031	0.571027	0.728656	0.380506	0.460258	0.456693	0.56
4	0.283799	0.193367	0.798542	0.842708	0.574850	0.727938	0.359888	0.466120	0.442651	0.56

In [34]:
```
# save results to the final dataframe

cross_val_results['KNN'] = pd.DataFrame(knn_result).agg(['mean', 'std']).round(3).T

cross_val_results['KNN']
```

Out[34]:

	mean	std
fit_time	0.173	0.137
score_time	0.271	0.124
test_accuracy	0.793	0.006
train_accuracy	0.843	0.000
test_precision	0.553	0.021
train_precision	0.727	0.002
test_recall	0.358	0.019
train_recall	0.467	0.004
test_f1	0.434	0.019
train_f1	0.568	0.003

In [35]:
```
# d) cross-validate on a xgboost classifier

from xgboost import XGBClassifier

xgb = make_pipeline(preprocessor, XGBClassifier())

xgb_result = cross_validate(xgb, X_train, y_train, cv=5, return_train_score=True, scoring=classification_

pd.DataFrame(xgb_result)
```

Out[35]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision	test_recall	train_recall	test_f1	tra
0	2.216017	0.439020	0.803750	0.904323	0.599678	0.923318	0.349906	0.620956	0.441943	0.74
1	1.707368	0.079490	0.804375	0.904010	0.604269	0.918191	0.345216	0.623535	0.439403	0.74
2	1.540947	0.025988	0.811458	0.901667	0.636210	0.920438	0.352720	0.610173	0.453832	0.73
3	1.823027	0.103180	0.821667	0.903281	0.664587	0.914886	0.399250	0.622509	0.498829	0.74
4	2.038635	0.638132	0.815208	0.904323	0.656794	0.911805	0.353327	0.630246	0.459476	0.7

In [36]:
```
# save results to the final dataframe

cross_val_results['xgb'] = pd.DataFrame(rf_result).agg(['mean', 'std']).round(3).T

cross_val_results['xgb']
```

Out[36]:

	mean	std
fit_time	7.671	0.904
score_time	0.328	0.263
test_accuracy	0.814	0.005
train_accuracy	0.999	0.000
test_precision	0.643	0.016
train_precision	0.999	0.000
test_recall	0.367	0.027
train_recall	0.998	0.000
test_f1	0.467	0.024
train_f1	0.999	0.000

In [37]:
```
# display summary of all models (dummy, linear model, other models)

pd.concat(cross_val_results, axis=1)
```

Out[37]:

	dummy		logreg		dt		rf		KNN		xgb	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
fit_time	0.002	0.000	0.194	0.050	0.618	0.047	7.671	0.904	0.173	0.137	7.671	0.904
score_time	0.005	0.003	0.012	0.002	0.012	0.003	0.328	0.263	0.271	0.124	0.328	0.263
test_accuracy	0.778	0.000	0.809	0.005	0.727	0.002	0.814	0.005	0.793	0.006	0.814	0.005
train_accuracy	0.778	0.000	0.809	0.002	0.999	0.000	0.999	0.000	0.843	0.000	0.999	0.000
test_precision	0.000	0.000	0.712	0.030	0.390	0.004	0.643	0.016	0.553	0.021	0.643	0.016
train_precision	0.000	0.000	0.713	0.005	1.000	0.000	0.999	0.000	0.727	0.002	0.999	0.000
test_recall	0.000	0.000	0.237	0.018	0.407	0.012	0.367	0.027	0.358	0.019	0.367	0.027
train_recall	0.000	0.000	0.237	0.011	0.997	0.001	0.998	0.000	0.467	0.004	0.998	0.000
test_f1	0.000	0.000	0.356	0.022	0.398	0.007	0.467	0.024	0.434	0.019	0.467	0.024
train_f1	0.000	0.000	0.355	0.013	0.999	0.000	0.999	0.000	0.568	0.003	0.999	0.000

9. Feature selection (Challenging)

rubric={reasoning}

Your tasks:

Make some attempts to select relevant features. You may try `RFECV`, forward 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.

In [38]:
```
from sklearn.linear_model import LogisticRegressionCV
lr_fs = make_pipeline(preprocessor, LogisticRegressionCV(penalty='l1',
                                                         solver='liblinear',
                                                         max_iter=20000,
                                                         tol=0.01,
                                                         random_state=123))

fs_cv = cross_validate(
    lr_fs, X_train, y_train,
    scoring=classification_metrics,
```

```
    return_train_score=True, cv=10)

pd.DataFrame(fs_cv)
```

Out[38]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision	test_recall	train_recall	test_f1	tr
0	4.564493	0.074425	0.802500	0.810000	0.657754	0.716241	0.230769	0.239842	0.341667	0.3
1	5.123812	0.221401	0.806667	0.808565	0.690608	0.709861	0.234522	0.234007	0.350140	0.3
2	4.156850	0.044958	0.810833	0.809954	0.739394	0.717146	0.228893	0.238800	0.349570	0.3
3	4.125186	0.233456	0.811667	0.808981	0.728814	0.710181	0.242026	0.236924	0.363380	0.3
4	3.973458	0.089755	0.804583	0.809537	0.672043	0.719692	0.234522	0.233799	0.347705	0.3
5	3.512931	0.051263	0.806667	0.809167	0.719745	0.713296	0.212008	0.235882	0.327536	0.3
6	4.013917	0.314688	0.827500	0.805833	0.818182	0.708190	0.287054	0.214420	0.425000	0.3
7	4.685376	0.263484	0.807500	0.809213	0.689840	0.714286	0.242026	0.235466	0.358333	0.3
8	3.688552	0.373488	0.804167	0.809259	0.716216	0.714829	0.198502	0.235098	0.310850	0.3
9	4.096683	0.113010	0.808750	0.808380	0.721893	0.711625	0.228464	0.230930	0.347084	0.3

In [39]:
```
cv_df = pd.DataFrame(fs_cv).agg(['mean', 'std']).round(3).T

cv_df
```

Out[39]:

	mean	std
fit_time	4.194	0.478
score_time	0.178	0.118
test_accuracy	0.809	0.007
train_accuracy	0.809	0.001
test_precision	0.715	0.045
train_precision	0.714	0.004
test_recall	0.234	0.023
train_recall	0.234	0.007
test_f1	0.352	0.030
train_f1	0.352	0.009

In [40]:
```
lr_fs.fit(X_train, y_train)

best = lr_fs.named_steps['logisticregressioncv']
```

In [41]:
```
n_coefs = best.coef_.shape[1]
n_coefs
```

Out[41]: 38

In [42]:
```
n_coefs_nonzero = norm(best.coef_.flatten(), 0)
n_coefs_nonzero
```

Out[42]: 38.0

In [43]:
```
best.C_
```

Out[43]: array([21.5443469])

In [44]:
```
feature_name = (
    lr_fs.named_steps["columntransformer"].get_feature_names_out().tolist()
)

coefs_pn = pd.DataFrame({
    'variable' : feature_name,
    'coef' : best.coef_.flatten()
}).sort_values(by='coef', ascending=False)
coefs_pn
```

Out[44]:

	variable	coef
32	ordinalencoder-2__PAY_1	0.551473
33	ordinalencoder-3__PAY_2	0.102524
3	standardscaler__BILL_AMT2	0.088127
6	standardscaler__BILL_AMT5	0.084286
34	ordinalencoder-4__PAY_3	0.080164
25	standardscaler__CB_6	0.079332
21	standardscaler__CB_2	0.079002
4	standardscaler__BILL_AMT3	0.077082
1	standardscaler__AGE	0.058103
37	ordinalencoder-7__PAY_6	0.041054
23	standardscaler__CB_4	0.040236
5	standardscaler__BILL_AMT4	0.030522
36	ordinalencoder-6__PAY_5	0.026985
19	standardscaler__BP_R6	0.020659
17	standardscaler__BP_R4	0.020581
15	standardscaler__BP_R2	0.013968
16	standardscaler__BP_R3	0.012712
22	standardscaler__CB_3	0.006533
35	ordinalencoder-5__PAY_4	-0.004337
18	standardscaler__BP_R5	-0.007017
14	standardscaler__BP_R1	-0.008552
12	standardscaler__PAY_AMT5	-0.023631
13	standardscaler__PAY_AMT6	-0.042789
24	standardscaler__CB_5	-0.046864
11	standardscaler__PAY_AMT4	-0.060344
31	ordinalencoder-1__EDUCATION	-0.078313
10	standardscaler__PAY_AMT3	-0.099803
7	standardscaler__BILL_AMT6	-0.103846
26	onehotencoder-1__SEX_2	-0.143251
2	standardscaler__BILL_AMT1	-0.156642
0	standardscaler__LIMIT_BAL	-0.164878
8	standardscaler__PAY_AMT1	-0.225564
9	standardscaler__PAY_AMT2	-0.229890
20	standardscaler__CB_1	-0.296482
28	onehotencoder-2__MARRIAGE_1	-1.771790
30	onehotencoder-2__MARRIAGE_3	-1.806132
29	onehotencoder-2__MARRIAGE_2	-1.943291
27	onehotencoder-2__MARRIAGE_0	-2.493875

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.

- GridSearchCV
- RandomizedSearchCV
- scikit-optimize

Points: 6

- `LogisticRegression`: We did hyperparameter optimization on `C` and `class_weight`. The best `C` is 0.01 and the best `class_weight` is "balanced". The best "f1 score" of the optimized parameters is 0.482. After finding the best hyperparameters, we applied cross-validation on the optimized model. The accuracy scores on the training set and validation set are both 0.699, with standard errors being 0.005 and 0.008. The scores on the training set and validation set are both 0.390, with standard errors being 0.006 and 0.012. The recall scores on the training set and validation set are 0.631 and 0.630, with standard errors being 0.002 and 0.025. The f1 score of the training data and testing data are both 0.482, with standard errors being 0.005 and 0.016.

- `DecisionTree`: We did hyperparameter optimization on `max_depth` and `class_weight`. The best `max_depth` is 5 and the best `class_weight` is "balanced". The best "f1 score" of the optimized parameters is 0.519. After finding the best hyperparameters, we applied cross-validation on the optimized model. The accuracy scores on the training set and validation set are 0.768 and 0.758, with standard errors being 0.022 and 0.031. The precision scores on the training set and validation set are 0.486 and 0.470, with standard errors being 0.034 and 0.047. The recall scores on the training set and validation set are 0.606 and 0.586, with standard errors being 0.047 and 0.032. The f1 scores on the training set and validation set are 0.537 and 0.519, with standard errors being 0.005 and 0.021.

- `RandomForestClassfier`: We did hyperparameter optimization on `max_depth` and `class_weight`. The best `max_depth` is 10 and the best `class_weight` is "balanced". The best "f1 score" of the optimized parameters is 0.539. After finding the best hyperparameters, we applied cross-validation on the optimized model. The accuracy scores on the training set and validation set are 0.846 and 0.792, with standard errors being 0.002 and 0.007. The precision scores on the training set and validation set are 0.644 and 0.531, with standard errors being 0.005 and 0.014. The recall scores on the training set and validation set are 0.689 and 0.545, with standard errors being 0.004 and 0.019. The f1 scores on the training set and validation set are 0.666 and 0.538, with standard errors being 0.001 and 0.016. The reason why we did not include `n_estimators` in hyperparameter optimization is that `n_estimators` does not have the problem of fundamental tradeoff, and that the score goes up with the number of trees without causing overfitting problem. Considering efficiency as well as memory capacity of computers, we do not choose to take into account very large values for `n_estimators` and therefore use the default value of 100 for our model.

- `KNeighborsClassifier`: We did hyperparameter optimization on `n_neighbors` and `leaf_size`. The best `n_neighbors` is 17 and the best `leaf_size` is 40. The best "f1 score" of the optimized parameters is 0.436. After finding the best hyperparameters, we applied cross-validation on the optimized model. The accuracy scores on the training set and validation set are 0.822 and 0.810, with standard errors being 0.001 and 0.004. The precision scores on the training set and validation set are 0.688 and 0.639, with standard errors being 0.004 and 0.022. The recall scores on the training set and validation set are 0.361 and 0.331, with standard errors being 0.008 and 0.009. The f1 scores on the training set and validation set are 0.474 and 0.436, with standard error as 0.007 and 0.009.

- `XGBoostClassifier`: We did hyperparameter optimization on `max_depth`, `learning_rate` and `class_weight`. The best `max_depth` is 5, the best `learning_rate` is 0.1 and the best `class_weight` is None. The best "f1 score" of the optimized parameters is 0.472. After finding the best hyperparameters, we applied cross-validation on the optimized model. The accuracy scores on the training set and validation set is 0.842 and 0.819, with standard errors being 0.002 and 0.007. The precision scores on the training set and validation set are 0.764 and 0.673, with standard errors being 0.007 and 0.030. The recall scores on the training set and validation set are 0.418 and 0.363, with standard errors being 0.007 and 0.017. The f1 scores on the training set and validation set are 0.541 and 0.472, with standard errors being 0.007 and 0.020.

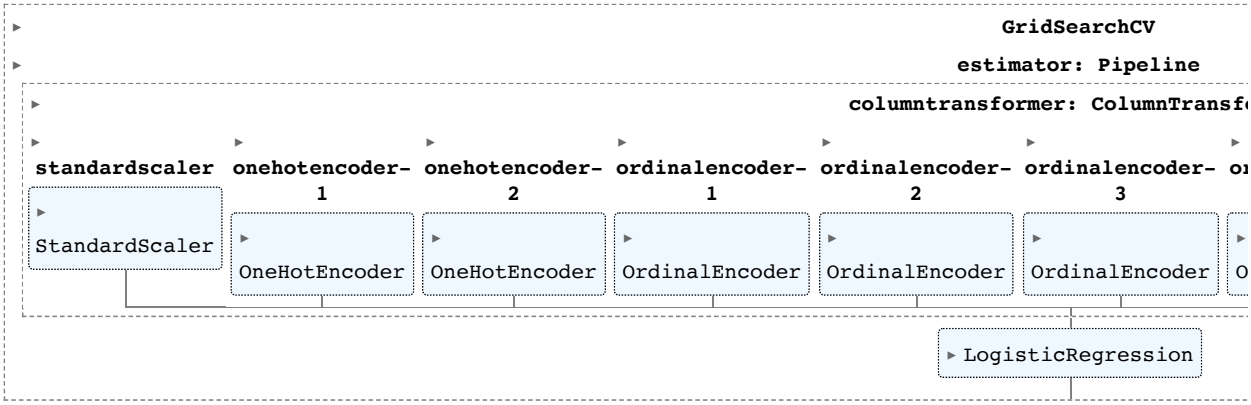
- In conclusion, the best performing model on f1 is `RandomForestClassfier`, the best performing model on accuracy and precision is `XGBoost`, and the best performing model on recall is `LogisticRegression`. Since we mainly focus

on analyzing the False Negative and False Positive cases, we choose f1 score as our main scoring metric. `RandomForestClassfier` will be the best model.

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision	test_recall	train_recall	test_f1	train_f1
Logistic Regression	0.076 (+/- 0.004)	0.009 (+/- 0.001)	0.699 (+/- 0.008)	0.699 (+/- 0.005)	0.390 (+/- 0.012)	0.390 (+/- 0.006)	0.630 (+/- 0.025)	0.631 (+/- 0.003)	0.482 (+/- 0.016)	0.482 (+/- 0.005)
KNN	0.014 (+/- 0.001)	0.126 (+/- 0.003)	0.810 (+/- 0.004)	0.822 (+/- 0.001)	0.639 (+/- 0.022)	0.688 (+/- 0.004)	0.331 (+/- 0.009)	0.361 (+/- 0.008)	0.436 (+/- 0.009)	0.474 (+/- 0.007)
Decision Tree	0.163 (+/- 0.001)	0.008 (+/- 0.001)	0.758 (+/- 0.031)	0.768 (+/- 0.022)	0.470 (+/- 0.047)	0.486 (+/- 0.034)	0.586 (+/- 0.032)	0.606 (+/- 0.047)	0.519 (+/- 0.021)	0.537 (+/- 0.005)
Random Forest	2.896 (+/- 0.043)	0.039 (+/- 0.001)	0.792 (+/- 0.007)	0.846 (+/- 0.002)	0.531 (+/- 0.014)	0.644 (+/- 0.005)	0.545 (+/- 0.019)	0.689 (+/- 0.004)	0.538 (+/- 0.016)	0.666 (+/- 0.001)
XGBoost	0.859 (+/- 0.013)	0.010 (+/- 0.000)	0.819 (+/- 0.007)	0.842 (+/- 0.002)	0.673 (+/- 0.030)	0.764 (+/- 0.007)	0.363 (+/- 0.017)	0.418 (+/- 0.007)	0.472 (+/- 0.020)	0.541 (+/- 0.007)

In [45]:
```
param_logreg = {"logisticregression__C": [10**-2,10**-1,10**0,10**1,10**2],
                "logisticregression__class_weight": ["balanced", None]}

search_logreg = GridSearchCV(logreg,
                             param_logreg,
                             n_jobs=-1,
                             cv=5,
                             scoring=classification_metrics,
                             refit='f1')

search_logreg.fit(X_train, y_train)
```

Out[45]:



In [46]:
```
search_logreg.best_params_
```

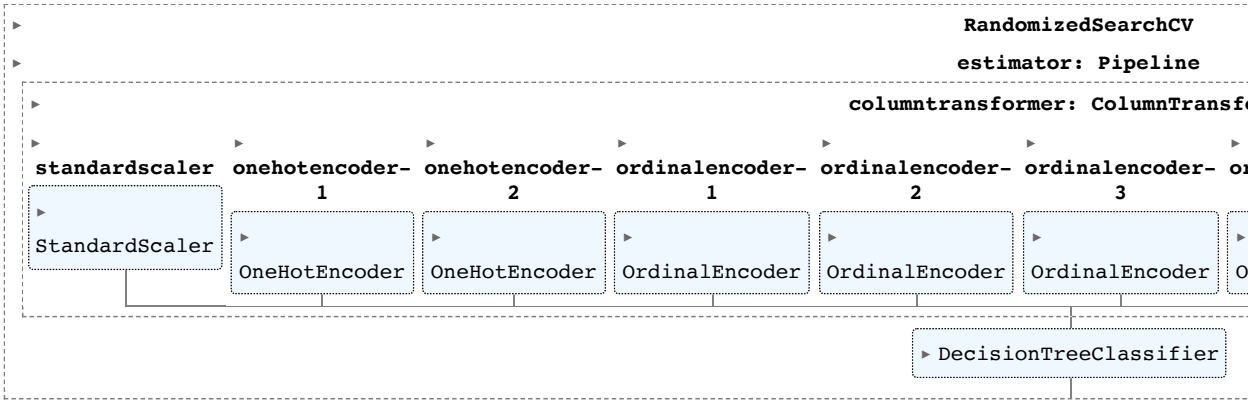
Out[46]:
```
{'logisticregression__C': 0.01, 'logisticregression__class_weight': 'balanced'}
```

In [47]:
```
search_logreg.best_score_
```

Out[47]:
```
0.4815184067080212
```

In [48]:
```
param_dt = {"decisiontreeclassifier__max_depth": np.arange(1, 50, 1),
            "decisiontreeclassifier__class_weight": ["balanced", None]}

search_dt = RandomizedSearchCV(dt,
                               param_dt,
                               n_jobs=-1,
                               n_iter=20,
                               cv=5,
                               scoring=classification_metrics,
                               refit='f1',
                               random_state = 123)

search_dt.fit(X_train, y_train)
```

Out[48]:



In [49]:
```
search_dt.best_params_
```

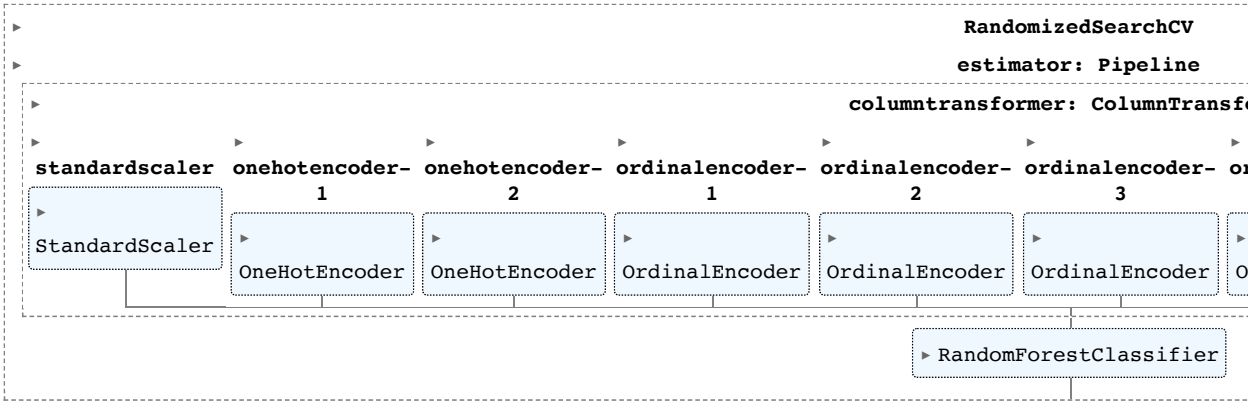
Out[49]:
```
{'decisiontreeclassifier__max_depth': 5,
 'decisiontreeclassifier__class_weight': 'balanced'}
```

In [50]:
```
search_dt.best_score_
```

Out[50]:
```
0.5194584848801401
```

In [51]:
```
param_rf = {"randomforestclassifier__max_depth": np.arange(10, 500, 10),
            "randomforestclassifier__class_weight": ["balanced", None]}

search_rf = RandomizedSearchCV(rf,
                               param_rf,
                               n_jobs=-1,
                               cv=5,
                               n_iter=20,
                               scoring=classification_metrics,
                               refit='f1',
                               random_state = 123)

search_rf.fit(X_train, y_train)
```

Out[51]:



In [52]:
```
search_rf.best_params_
```

Out[52]:
```
{'randomforestclassifier__max_depth': 10,
 'randomforestclassifier__class_weight': 'balanced'}
```

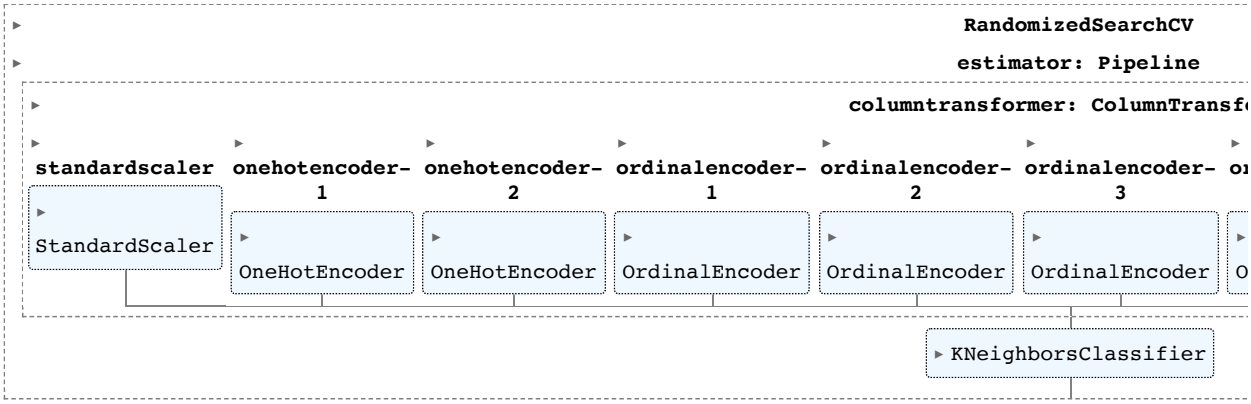
In [53]:
```
search_rf.best_score_
```

Out[53]:
```
0.5392590097352461
```

In [54]:
```
param_knn = {"kneighborsclassifier__n_neighbors": np.arange(1, 30, 1),
             "kneighborsclassifier__leaf_size": np.arange(1, 50, 1)}

search_knn = RandomizedSearchCV(knn,
                                param_knn,
                                n_jobs=-1,
                                cv=5,
```

```
                                scoring=classification_metrics,
                                refit='f1',
                                random_state = 123)

search_knn.fit(X_train, y_train)
```

Out[54]:



In [55]:
```
search_knn.best_params_
```

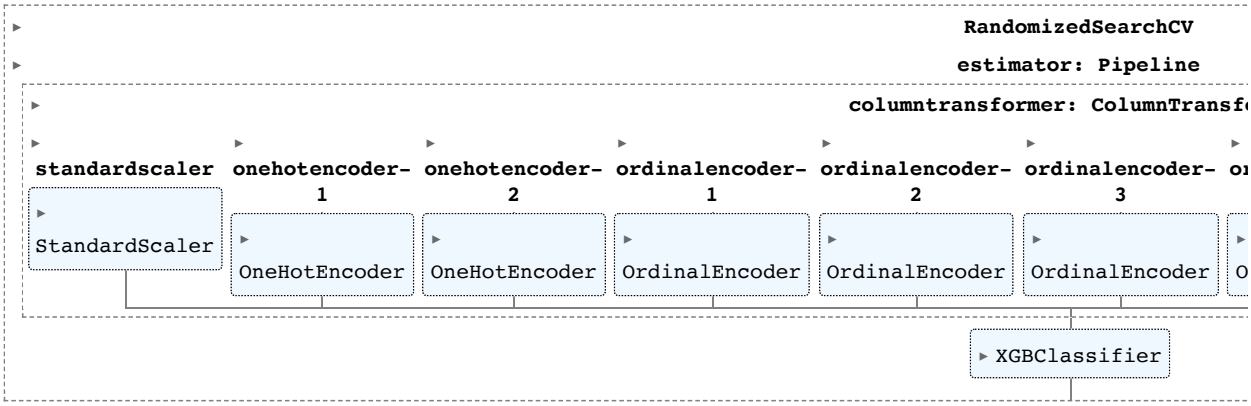
Out[55]:
```
{'kneighborsclassifier__n_neighbors': 17,
 'kneighborsclassifier__leaf_size': 40}
```

In [56]:
```
search_knn.best_score_
```

Out[56]:
```
0.43561477347353283
```

In [57]:
```
xgb = make_pipeline(preprocessor, XGBClassifier())

param_xgb = {"xgbclassifier__learning_rate": [0.1, 0.3, 0.5, 0.7, 0.9],
             "xgbclassifier__max_depth": range(3,20),
              "xgbclassifier__class_weight": ['balanced', None]}

search_xgb = RandomizedSearchCV(xgb,
                                param_xgb,
                                n_jobs=-1,
                                cv=5,
                                n_iter=20,
                                scoring=classification_metrics,
                                refit='f1',
                                random_state = 123)

search_xgb.fit(X_train, y_train)
```

Out[57]:



In [58]:
```
search_xgb.best_params_
```

Out[58]:
```
{'xgbclassifier__max_depth': 5,
 'xgbclassifier__learning_rate': 0.1,
 'xgbclassifier__class_weight': None}
```

In [59]:
```
search_xgb.best_score_
```

Out[59]: 0.4715741773453123

In [60]:
```
# This function was modified from lecture notes:

#https://github.ubc.ca/MDS-2022-23/DSCI_571_sup-learn-1_students/blob/master/lectures/04_preprocessing-pi

def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
    """
    Returns mean and std of cross validation

    Parameters
    ----------
    model :
        scikit-learn model
    X_train : numpy array or pandas DataFrame
        X in the training data
    y_train :
        y in the training data

    Returns
    ----------
        pandas Series with mean scores from cross_validation
    """

    scores = cross_validate(model, X_train, y_train, **kwargs)

    mean_scores = pd.DataFrame(scores).mean()
    std_scores = pd.DataFrame(scores).std()
    out_col = []

    for i in range(len(mean_scores)):
        out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))

    return pd.Series(data=out_col, index=mean_scores.index)
```

In [61]:
```
models = {
    "Logistic Regression": LogisticRegression(C=0.01, class_weight='balanced'),
    "KNN": KNeighborsClassifier(n_neighbors=17, leaf_size=40),
    "Decision Tree": DecisionTreeClassifier(max_depth=5, class_weight='balanced'),
    "Random Forest": RandomForestClassifier(max_depth=10, class_weight='balanced'),
    "XGBoost": XGBClassifier(max_depth=5, learning_rate=0.1, class_weight=None)
}
```

In [62]:
```
results_dict = {}
for m in models.items():
    model = m[1]
    pipe = make_pipeline(preprocessor, model)
    results_dict[f"{m[0]}"] = mean_std_cross_val_scores(
    pipe, X_train, y_train, cv=5, return_train_score=True, scoring=classification_metrics
)
results_df = pd.DataFrame(results_dict).T
results_df
```

Out[62]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision	test_recall	train_recall	test_f
Logistic Regression	0.114 (+/- 0.048)	0.013 (+/- 0.004)	0.699 (+/- 0.008)	0.699 (+/- 0.005)	0.390 (+/- 0.012)	0.390 (+/- 0.006)	0.630 (+/- 0.025)	0.631 (+/- 0.003)	0.482 (+/ 0.016
KNN	0.022 (+/- 0.003)	0.185 (+/- 0.019)	0.810 (+/- 0.004)	0.822 (+/- 0.001)	0.639 (+/- 0.022)	0.688 (+/- 0.004)	0.331 (+/- 0.009)	0.361 (+/- 0.008)	0.436 (+/ 0.009
Decision Tree	0.185 (+/- 0.012)	0.010 (+/- 0.000)	0.757 (+/- 0.031)	0.768 (+/- 0.022)	0.469 (+/- 0.047)	0.486 (+/- 0.034)	0.586 (+/- 0.032)	0.606 (+/- 0.047)	0.519 (+/ 0.021
Random Forest	3.331 (+/- 0.114)	0.047 (+/- 0.007)	0.793 (+/- 0.007)	0.845 (+/- 0.001)	0.532 (+/- 0.016)	0.642 (+/- 0.003)	0.549 (+/- 0.015)	0.689 (+/- 0.004)	0.54 (+/ 0.016
XGBoost	1.297 (+/- 0.330)	0.018 (+/- 0.006)	0.819 (+/- 0.007)	0.842 (+/- 0.002)	0.673 (+/- 0.030)	0.764 (+/- 0.007)	0.363 (+/- 0.017)	0.418 (+/- 0.007)	0.472 (+/ 0.020

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

We examined the feature_importances_ attributes and with eli5 (essentially the same, but for practice reason we did both)

- the model deemed the historical repayment status to be most important when predicting the payment behavior for next month (PAY_1 to PAY_5). This makes sense because if a person paid on time previously, then he/she tends to be more trustworthy on paying on time in the future and has the ability to pay, and vice versa.
- The amount of previous payment is the next most important feature in the model. This also makes sense because how much one could pay can indicate the person's financial status.
- LIMIT_BAL, amount of given credit is the third most important feature in the model. This makes sense because credit card companies often issue the credit limit based on a preliminaty assessment of the person's financial status and the limit given is associated with one's credibility.

In the permutation importance plot, the pattern is in general same as feature_importances_, with some slight differences:

- We can clearly see that PAY_1 is a very useful feature in the model, followed by PAY_2 and LIMIT_BAL.
- The next important feature is the newly created feature: CB_3 and CB_2, the ratio of billed amount to credit limit for August and July respectively.

We also used SHAP summary plot to investigate the global feature importance.

- From the SHAP summary plot, we can see that PAY_1, PAY_2 and PAY_3 are the top 3 features that has large effect on the prediction. The patterns is clear from the color of the plot. Higher feature value of PAY_1, PAY_2 and PAY_3 means that the person have delayed the parment by more months, is driving the prediction towards "default payment".

- Limit_BAL is the 4th important feature. Lower amount of given credit drives the prediction towards "default payment". This matches with how credit limit is issued: a person with better financial status is often issued with higher limit.

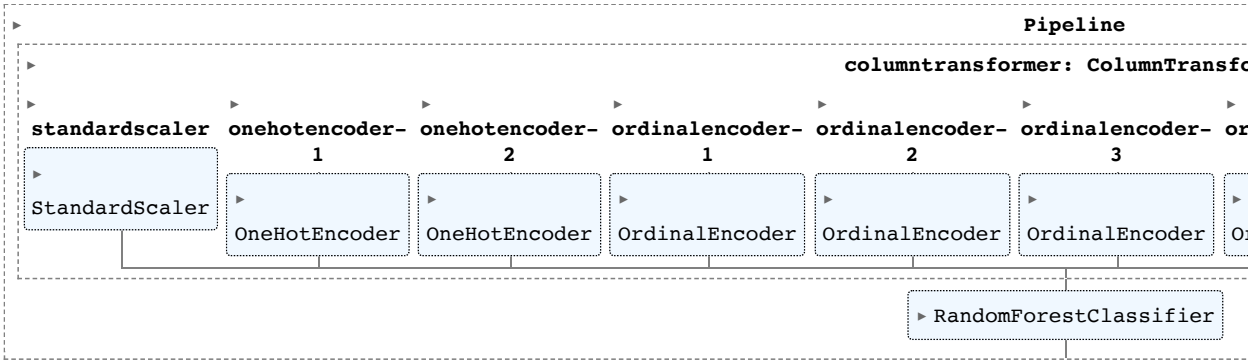
- PAY_AMT1 is the 5th important feature. Lower amoung of previous payment drives the prediction towards "deafult payment". This makes sense because the person may not be able to pay the full billed amount due to financial difficulties.

- Then, PAY_4, PAY_AMT2, PAY_AMT3 are the 6th, 7th and 8th important feature. They all shows consistent pattern as the associated variable similar to them, which we have explained above.

- The 9th important feature is CB_2. Higher CB_2 value is driving the prediction towards "default payment". This can be explained by the fact that when one cannot make payment on time, the billed amount is transferred to the next month, and thus the billed amount will reach the allowed credit limit.

In [63]:
```
rf_best = make_pipeline(preprocessor, RandomForestClassifier(max_depth=10, class_weight='balanced', randc

rf_best.fit(X_train, y_train)
```

Out[63]:



In [64]:

```
# Feature Importance
data = {
    "Importance": rf_best.named_steps["randomforestclassifier"].feature_importances_,
}
rf_importance_df = pd.DataFrame(
    data=data,
    index=X_train_enc.columns,
).sort_values(by="Importance", ascending=False)

rf_importance_df
```

Out[64]:

	Importance
PAY_1	0.192565
PAY_2	0.098037
PAY_3	0.057510
PAY_4	0.052256
PAY_5	0.033521
PAY_AMT1	0.032409
LIMIT_BAL	0.031064
CB_2	0.030379
PAY_AMT2	0.027886
CB_1	0.027068
BILL_AMT1	0.025819
PAY_6	0.024730
CB_3	0.024082
PAY_AMT3	0.023914
CB_5	0.021447
BP_R1	0.021216
PAY_AMT4	0.021070
CB_4	0.019710
BP_R2	0.019240
BILL_AMT2	0.018086
CB_6	0.017977
PAY_AMT5	0.017579
PAY_AMT6	0.017142
BP_R3	0.015921
BILL_AMT3	0.015882
BILL_AMT4	0.015431
BP_R4	0.015191
BILL_AMT6	0.014898
BP_R6	0.014435
AGE	0.014380
BP_R5	0.013985
BILL_AMT5	0.013762
EDUCATION	0.004282
SEX	0.002522
MARRIAGE_2	0.002010
MARRIAGE_1	0.001880
MARRIAGE_3	0.000621
MARRIAGE_0	0.000096

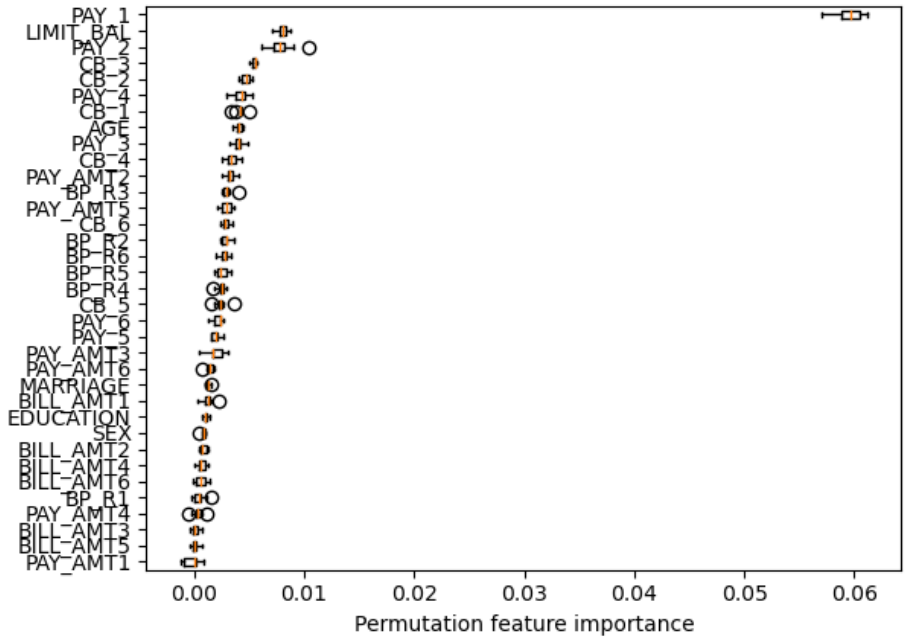
In [65]:
```
import eli5

eli5.explain_weights(rf_best.named_steps["randomforestclassifier"], feature_names=column_names)
```

Out[65]:

Weight	Feature
0.1926 ± 0.2329	PAY_1
0.0980 ± 0.2203	PAY_2
0.0575 ± 0.1601	PAY_3
0.0523 ± 0.1563	PAY_4
0.0335 ± 0.1101	PAY_5
0.0324 ± 0.0494	PAY_AMT1
0.0311 ± 0.0515	LIMIT_BAL
0.0304 ± 0.0324	CB_2
0.0279 ± 0.0381	PAY_AMT2
0.0271 ± 0.0259	CB_1
0.0258 ± 0.0243	BILL_AMT1
0.0247 ± 0.0830	PAY_6
0.0241 ± 0.0263	CB_3
0.0239 ± 0.0341	PAY_AMT3
0.0214 ± 0.0297	CB_5
0.0212 ± 0.0328	BP_R1
0.0211 ± 0.0285	PAY_AMT4
0.0197 ± 0.0185	CB_4
0.0192 ± 0.0197	BP_R2
0.0181 ± 0.0165	BILL_AMT2
… 18 more …	

In [66]:

```
from sklearn.inspection import permutation_importance

def get_permutation_importance(model):
    ''' This function is adapted from lecture 8 course notes'''
    X_train_perm = X_train.drop(columns=["ID"])
    result = permutation_importance(model, X_train_perm, y_train, n_repeats=10, random_state=123)
    perm_sorted_idx = result.importances_mean.argsort()
    plt.boxplot(
        result.importances[perm_sorted_idx].T,
        vert=False,
        labels=X_train_perm.columns[perm_sorted_idx],
    )
    plt.xlabel('Permutation feature importance')
    plt.show()

get_permutation_importance(rf_best)
```



In [67]:

```
# Global feature importance using SHAP – averages
# code adapted from lecture notes lecture 8

import shap
# Create a shap explainer object
rf_explainer = shap.TreeExplainer(rf_best.named_steps["randomforestclassifier"])
train_rf_shap_values = rf_explainer.shap_values(X_train_enc)
values = np.abs(train_rf_shap_values[1]).mean(0)
pd.DataFrame(data=values, index=X_train_enc.columns, columns=["SHAP"]).sort_values(
```

```
    by="SHAP", ascending=False
)
```

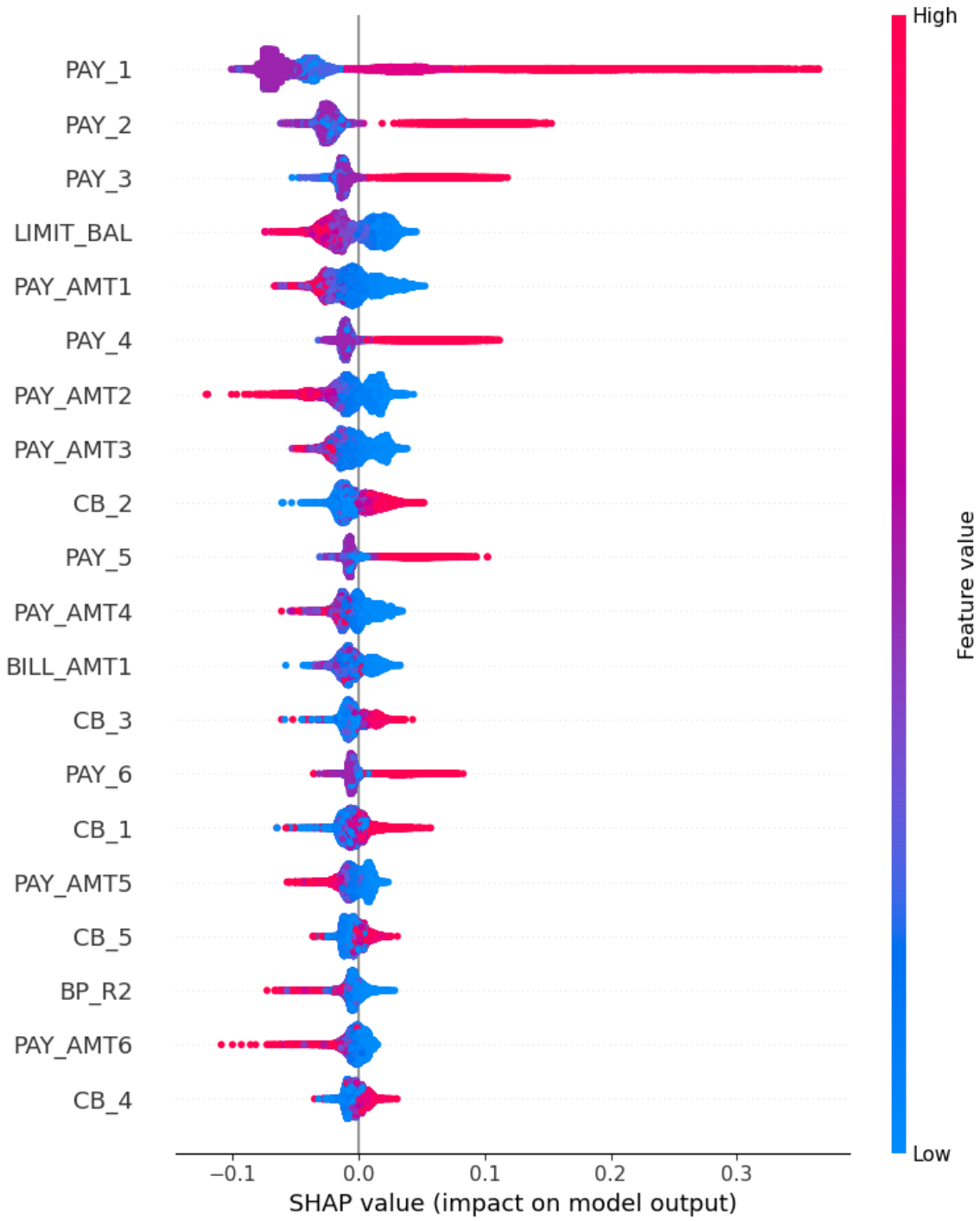
```
/Users/shirley/opt/miniconda3/envs/573/lib/python3.10/site-packages/tqdm/auto.py:22: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Out[67]:

	SHAP
PAY_1	0.067550
PAY_2	0.033845
PAY_3	0.018906
LIMIT_BAL	0.017725
PAY_AMT1	0.015604
PAY_4	0.015466
PAY_AMT2	0.014810
PAY_AMT3	0.012679
CB_2	0.011916
PAY_5	0.010844
PAY_AMT4	0.010266
BILL_AMT1	0.009865
CB_3	0.008429
PAY_6	0.008225
CB_1	0.007958
PAY_AMT5	0.007949
CB_5	0.006649
BP_R2	0.006077
PAY_AMT6	0.005925
CB_4	0.005786
BP_R1	0.005405
CB_6	0.005194
BP_R3	0.004874
BILL_AMT2	0.004336
BILL_AMT3	0.003870
BP_R6	0.003622
BILL_AMT4	0.003494
BP_R4	0.003379
BP_R5	0.003198
BILL_AMT6	0.003147
BILL_AMT5	0.002876
AGE	0.002347
SEX	0.001246
MARRIAGE_1	0.001009
EDUCATION	0.000966
MARRIAGE_2	0.000832
MARRIAGE_3	0.000075
MARRIAGE_0	0.000015

In [68]:
```
shap.summary_plot(train_rf_shap_values[1], X_train_enc)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored
```



(In case the plot is easier to be interpreted with explaination, the interpretation of this plot is copied to right beneath the plot here)

- From the SHAP summary plot, we can see that PAY_1, PAY_2 and PAY_3 are the top 3 features that has large effect on the prediction. The patterns is clear from the color of the plot. Higher feature value of PAY_1, PAY_2 and PAY_3 means that the person have delayed the parment by more months, is driving the prediction towards "default payment".
- Limit_BAL is the 4th important feature. Lower amount of given credit drives the prediction towards "default payment". This matches with how credit limit is issued: a person with better financial status is often issued with higher limit.
- PAY_AMT1 is the 5th important feature. Lower amoung of previous payment drives the prediction towards "deafult payment". This makes sense because the person may not be able to pay the full billed amount due to financial difficulties.

- Then, PAY_4, PAY_AMT2, PAY_AMT3 are the 6th, 7th and 8th important feature. They all shows consistent pattern as the associated variable similar to them, which we have explained above.

- The 9th important feature is CB_2. Higher CB_2 value is driving the prediction towards "default payment". This can be explained by the fact that when one cannot make payment on time, the billed amount is transferred to the next month, and thus the billed amount will reach the allowed credit limit.

Examine a model with averaged BILL_AMT and Bill:creadit ratio

(Note: we did not end up using the aggregated (averaged) features)

As mentioned in question 4, feature engineering, we examined if the model could be more interpretable while still achieving comparable perfomance with less features. We aggregated the highly correlated features CB_1 to CB_6 and BILL_AMT1to BILL_AMT6, by taking their 6-months average and drop the individual features. However:

- the model did not show improvement
- The feature importance shows that the two averages are relatively important features. However, we should take this result with a grain of salt because the imputiry-based feature important is often biased towards numeric features (high cardinality features). The permutation importance does not exhibit such bias.
- Looking at the permutation importance plot, the newly added features are not neccesarily more important than the ones we identified above.

We suspect that there is added information in the person's payment behavior in each individual month. Thus we keep the features separated in our analysis.

In [69]:
```
# Correlation > 0.8
# compute and take abs value
corr_df = X_train_enc.corr().abs()
# Set the diagonal to 0
corr_df[corr_df == 1] = 0
corr_df.style.background_gradient()

# BILL_AMT > 0.9 = take average to summarize all
# CB > 0.8 = take average to summarize all
train_df_11 = train_df.copy()
test_df_11 = test_df.copy()

train_df_11['BILL_AMT_AVG'] = train_df_11[['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5
test_df_11['BILL_AMT_AVG'] = test_df_11[['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5',

train_df_11['CB_AVG'] = train_df_11[['CB_1', 'CB_2', 'CB_3', 'CB_4', 'CB_5','CB_6']].mean(axis=1)
test_df_11['CB_AVG'] = test_df_11[['CB_1', 'CB_2', 'CB_3', 'CB_4', 'CB_5','CB_6']].mean(axis=1)

X_train_11 = train_df_11.drop(columns='target')
y_train_11 = train_df_11['target']
X_test_11 = test_df_11.drop(columns='target')
y_test_11 = test_df_11['target']

# create new preprocessor
edu_feat = ['EDUCATION']
pay_feat = ['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
binary_feat = ['SEX']
cat_feat = ['MARRIAGE']
drop_feat_11 = ['ID',
'BILL_AMT1',
'BILL_AMT2',
'BILL_AMT3',
'BILL_AMT4',
'BILL_AMT5',
'BILL_AMT6',
'CB_1',
'CB_2',
'CB_3',
'CB_4',
'CB_5',
'CB_6' ]
num_feat_11 =[
'LIMIT_BAL',
```

```
'AGE',
'BILL_AMT_AVG',
'PAY_AMT1',
'PAY_AMT2',
'PAY_AMT3',
'PAY_AMT4',
'PAY_AMT5',
'PAY_AMT6',
'BP_R1',
'BP_R2',
'BP_R3',
'BP_R4',
'BP_R5',
'BP_R6',
'CB_AVG'
]
pay_order = [-2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8]

preprocessor_11 = make_column_transformer(
    (StandardScaler(),num_feat_11),
    (OneHotEncoder(drop='if_binary', dtype = int), binary_feat),
    (OneHotEncoder(handle_unknown="ignore", sparse=False, dtype = int), cat_feat),
    (OrdinalEncoder(dtype=int), edu_feat),
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    (OrdinalEncoder(categories=[pay_order], dtype=int, handle_unknown = 'use_encoded_value', unknown_valu
    ("drop", drop_feat_11)
)
preprocessor_11
transformed_11 = preprocessor_11.fit_transform(X_train_11)
column_names_11 = (
    num_feat_11
    + binary_feat
    + preprocessor_11.named_transformers_["onehotencoder-2"].get_feature_names_out().tolist()
    + edu_feat
    + pay_feat
)

X_train_enc_11 = pd.DataFrame(transformed_11, columns=column_names_11)
X_train_enc_11.head(2)
```

Out[69]:

	LIMIT_BAL	AGE	BILL_AMT_AVG	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	BP_R1	...
0	0.246770	1.140423	0.812437	-0.117221	-0.042650	-0.110198	-0.149314	-0.122994	-0.207312	0.009269	...
1	-0.368109	-0.378471	-0.694749	-0.298634	-0.261604	-0.267318	-0.261306	-0.315106	-0.271741	0.010529	...

2 rows × 28 columns

In [70]:

```
# train model with average features
rf_best_11 = make_pipeline(preprocessor_11, RandomForestClassifier(max_depth=10, class_weight='balanced',
dt_result_11 = cross_validate(rf_best_11, X_train_11, y_train_11, cv=5, return_train_score=True, scoring=
pd.DataFrame(dt_result_11)
```

```
[14:44:02] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:45:18] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:46:45] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:43:24] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:43:36] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:45:53] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:46:21] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:47:54] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:44:05] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:45:18] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:47:45] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:43:24] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:43:49] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:45:18] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:46:02] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:46:33] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:48:22] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:43:40] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:45:16] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
```

```
Parameters: { "class_weight" } are not used.

[14:45:58] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:46:26] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:48:23] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.
```

Out[70]:

	fit_time	score_time	test_accuracy	train_accuracy	test_precision	train_precision	test_recall	train_recall	test_f1	tr
0	2.451494	0.048547	0.786250	0.845417	0.518832	0.650231	0.516886	0.658462	0.517857	0.6
1	2.363718	0.056160	0.792083	0.843542	0.531423	0.643084	0.539400	0.664791	0.535382	0.6
2	2.245630	0.046461	0.796875	0.844323	0.542484	0.644555	0.545028	0.667370	0.543753	0.6
3	2.294618	0.044947	0.805417	0.840729	0.561290	0.635466	0.570759	0.663775	0.565985	0.6
4	2.109399	0.041786	0.798125	0.840365	0.545794	0.633571	0.547329	0.667292	0.546561	0.6

In [71]:

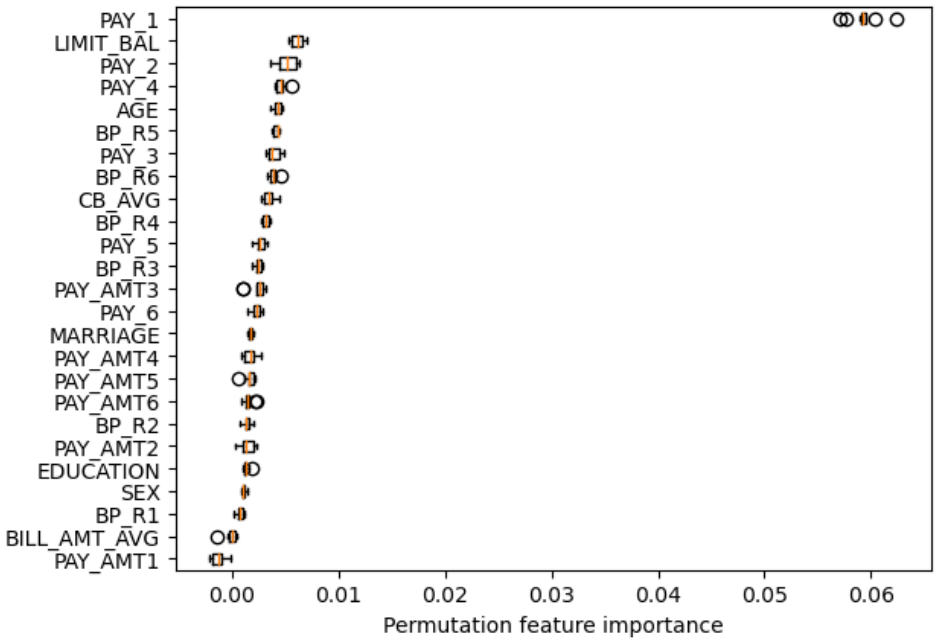
```
# Check importance
rf_best_11.fit(X_train_11, y_train_11)

eli5.explain_weights(rf_best_11.named_steps["randomforestclassifier"], feature_names=column_names_11)
```

```
[14:43:24] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:43:36] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:45:27] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:47:48] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:48:24] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:44:05] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:45:46] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:48:25] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:45:27] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:46:46] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:47:44] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:47:54] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.

[14:48:29] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1667849614592/work/src/learner.cc:7
67:
Parameters: { "class_weight" } are not used.
```

Out[71]:

Weight	Feature
0.2251 ± 0.2463	PAY_1
0.1165 ± 0.2596	PAY_2
0.0740 ± 0.1827	PAY_3
0.0530 ± 0.0374	CB_AVG
0.0451 ± 0.1356	PAY_4
0.0392 ± 0.0284	BILL_AMT_AVG
0.0381 ± 0.0465	PAY_AMT2
0.0372 ± 0.0521	PAY_AMT1
0.0370 ± 0.0408	LIMIT_BAL
0.0296 ± 0.0335	PAY_AMT4
0.0295 ± 0.0335	PAY_AMT3
0.0293 ± 0.0337	BP_R1
0.0290 ± 0.0847	PAY_5
0.0269 ± 0.0295	BP_R2
0.0256 ± 0.0838	PAY_6
0.0232 ± 0.0140	BP_R3
0.0214 ± 0.0209	PAY_AMT6
0.0210 ± 0.0195	PAY_AMT5
0.0208 ± 0.0119	BP_R4
0.0207 ± 0.0133	BP_R6
… *8 more* …	

In [72]:
```
def get_permutation_importance_11(model):
    ''' This function is adapted from lecture 8 course notes'''
    X_train_perm = X_train_11.drop(columns=drop_feat_11)
    result = permutation_importance(model, X_train_perm, y_train_11, n_repeats=10, random_state=123)
    perm_sorted_idx = result.importances_mean.argsort()
    plt.boxplot(
        result.importances[perm_sorted_idx].T,
        vert=False,
        labels=X_train_perm.columns[perm_sorted_idx],
    )
    plt.xlabel('Permutation feature importance')
    plt.show()

get_permutation_importance_11(rf_best_11)
```



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

1. The best performing model, i.e. `RandomForestClassifier` shows an f1 score of 0.538 on the test set. From the confusion matrix and classification report on the test data shown below, it can be seen that there are more true negative cases than false positive, false negative and true positive. The number of false negative cases and false positive cases are similar, which might be a result of using the `class_weight = 'balanced'` that deals with class imbalance problem. The precision, recall and f1 score are very close in values, which further proves the effect of balancing the classes in the target.
2. The test score agrees with the validation score of 0.541 from before. There might be some degree of optimization bias in the model, since the mean training score of 0.667 is higher than the mean validation score of 0.541, which suggests that there is overfitting in the model. The `max_depth` of 10 obtained from hyperparameter optimization on the

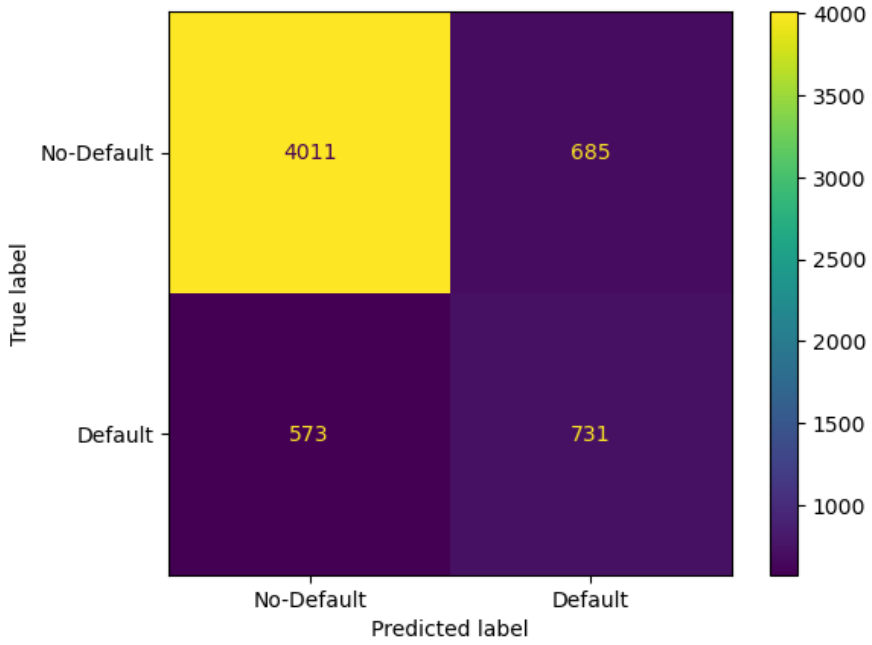
`RandomForestClassifier` is quite a large number, which can consequently lead to a complex model, resulting in overfitting. Therefore we should not fully trust the results.

In [73]:
```
y_predict = rf_best.predict(X_test)
best_score = f1_score(y_test, y_predict)
best_score
```

Out[73]:
```
0.5375000000000001
```

In [74]:
```
confmat_rf_best = ConfusionMatrixDisplay.from_estimator(
      rf_best, X_test, y_test, values_format="d", display_labels=["No-Default", "Default"]
  )
confmat_rf_best
```

Out[74]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x145fcce20>
```



In [75]:
```
print(
    classification_report(
        y_test, rf_best.predict(X_test), target_names=["no-default", "default"]
    )
)
```

```
              precision    recall  f1-score   support

  no-default       0.88      0.85      0.86      4696
     default       0.52      0.56      0.54      1304

    accuracy                           0.79      6000
   macro avg       0.70      0.71      0.70      6000
weighted avg       0.80      0.79      0.79      6000
```

In [76]:
```
X_test_enc = pd.DataFrame(
    data=preprocessor.transform(X_test),
    columns=column_names,
    index=X_test.index,
)
X_test_enc
```

Out[76]:

	LIMIT_BAL	AGE	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT
25665	-0.982987	-1.029426	-0.300983	-0.346026	-0.484529	-0.671954	0.061206	-0.337159	-0.259964	-0.2300
16464	-0.675548	2.550825	0.335277	0.296127	0.090264	0.115932	0.165482	0.181306	-0.145865	-0.1782
22386	0.016191	-0.595456	1.429288	1.543630	1.663729	1.768395	2.099254	2.227535	0.068249	0.0276
10149	0.246770	0.597961	-0.374887	-0.678558	-0.682087	-0.671954	-0.661958	-0.652167	-0.293263	-0.2768
8729	-0.906127	0.814945	-0.584234	-0.575956	-0.550711	-0.529090	-0.507269	-0.490140	-0.269512	-0.2230
...	...	...	...	...	...	...	...	...	...	...
9940	-0.060669	1.682885	-0.677401	-0.668072	-0.643985	-0.633904	-0.581271	-0.590352	-0.258115	-0.1610
11351	-0.752408	1.899870	-0.504618	-0.476939	-0.452760	-0.422618	-0.391955	-0.369727	-0.244807	-0.2177
29732	-1.213567	1.031930	-0.610041	-0.603494	-0.535923	-0.574899	-0.671979	-0.655874	-0.337542	-0.0748
9088	-0.906127	1.574393	-0.072224	-0.360757	-0.682087	-0.619300	-0.616473	-0.652167	-0.202318	-0.2768
23862	1.015369	-0.595456	-0.694565	-0.658316	-0.670844	-0.628994	-0.616391	-0.413094	-0.146343	-0.2768

6000 rows × 38 columns

In [77]:
```
import shap

rf_explainer = shap.TreeExplainer(rf_best.named_steps["randomforestclassifier"])
rf_explainer
```

Out[77]:
```
<shap.explainers._tree.Tree at 0x1696e91e0>
```

In [78]:
```
train_rf_shap_values = rf_explainer.shap_values(X_train_enc)
```

In [79]:
```
test_rf_shap_values = rf_explainer.shap_values(X_test_enc)
```

In [80]:
```
shap.initjs()
y_test_reset = y_test.reset_index(drop=True)
```

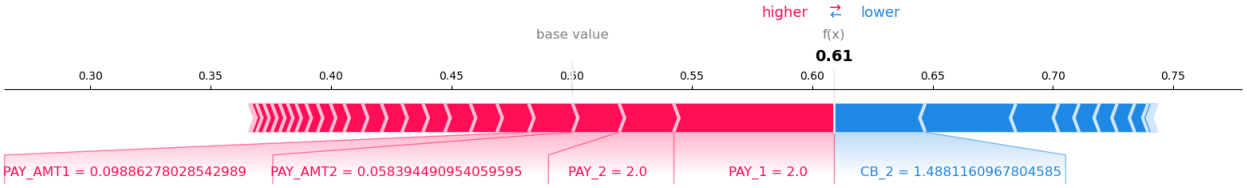
In [81]:
```
default_1 = y_test_reset[y_test_reset == 1].index.tolist()
default_0 = y_test_reset[y_test_reset == 0].index.tolist()

default_1_index = default_1[8]
default_0_index = default_0[8]
```

In [82]:
```
rf_best.named_steps["randomforestclassifier"].fit(X_train_enc, y_train)
```

Out[82]:
```
RandomForestClassifier

RandomForestClassifier(class_weight='balanced', max_depth=10, random_state=123)
```

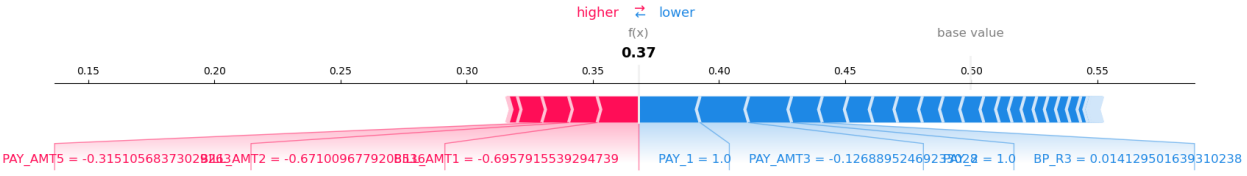
From the SHAP plot for this example, it can be seen that this particular prediction shows a raw score of 0.61, which is higher than the base value of 0.5. Among all the features, it can be seen that `PAY_1` = 2.0, `PAY_2` =2.0, `PAY_AMT2` = 0.058 and `PAY_AMT1` = 0.099 are pushing the target towards 1 (shown in red), while `CB_2` = 1.488 is pushing the target towards 0 (shown in blue). Since the effect of the red region is larger than that of the blue region, the resultant prediction is shifted towards a higher value compared to the base value, i.e., closer to 1.

In [83]:
```
shap.force_plot(
    rf_explainer.expected_value[0],
    test_rf_shap_values[0][default_0_index, :],
    X_test_enc.iloc[default_0_index, :],
    matplotlib=True,
)
```



From the SHAP plot for this example, it can be seen that this particular prediction shows a raw score of 0.37, which is lower than the base value of 0.5. Among all the features, it can be seen that `PAY_AMT5` = -0.315, `BILL_AMT2` = -0.671, and `BILL_AMT1` = -0.696 are pushing the target towards 1 (shown in red), while `PAY_1` = 1.0, `PAY_AMT3` = -0.127, `PAY_2` = 1.0 and `BP_R3` = 0.014 are pushing the target towards 0 (shown in blue). Since the effect of the blue region is larger than that of the red region, the resultant prediction is shifted towards a lower value compared to the base value, i.e., closer to 0.

In [84]:
```
shap.force_plot(
    rf_explainer.expected_value[1],
    test_rf_shap_values[1][default_1_index, :],
    X_test_enc.iloc[default_1_index, :],
    matplotlib=True,
)
```



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

13.1 Table summarizing important results

Step	Description of Results	f1 Scores
Exploratory Data Analysis	• There is high class imbalance in our data (78% of the target values are 0) • Distributions of `AGE` and `BILL_AMT*` when colored by target value suggest that these features may be associated with the target • A correlation heat map shows correlations between `PAY_AMT*` columns and `BILL_AMT*`	N/A
Data Preprocessing and Transformation	• We added the columns `BP_*` (amount paid / amount billed per month, higher ratio could indicate more ability to pay/better financial status) and `CB_*` (amount billed / amount of given credit per month) columns	N/A

Step	Description of Results	f1 Scores
	• We applied scaling on numeric columns, ordinal encoding on `EDUCATION` and `PAY_*` , and OHE on `SEX` (binary OHE) and `MARRIAGE`	
Baseline Model	We trained a Dummy Classifier and got a mean cross-validation f1-score of 0.000	0.000
Linear Model	• We trained a logistic regression model with gridsearch hyperparameter optimization • We found C = 10 (larger than default value of 1) • Overall, this model had a poor mean cross-validation f1 score	0.356
Non-Linear Models	• We tried the following non-linear models (without hyperparameter optimization), the mean cross-validation scores are in the right column: • Decision Tree Classifier, Random Forest, KNN, Xgboost • Random Forest and Xgboost had the best f1-scores	• Decision Tree = 0.398 • Random Forest = 0.467 • KNN = 0.434 • Xgboost = 0.467
Feature Selection	• We tried feature selection using L1-regularization • We found 38 features, with a best C of 21.5 • We obtained a mean cross-validation f1 test score of 0.352 • As this did not improve our model, we will abandon feature selection	0.352
Hyperparameter Optimization	• Logistic Regression best hyperparameters: C = 0.01, class_weight= balanced • Decision Tree best hyperparameters: max_depth = 10, class_weight = balanced • Random Forest best hyperparameters: max_depth = 10, class_weight= balanced • KNN best hyperparameters: n_neighbors = 17, leaf_size = 40 • Xgboost best hyperparameters: max_depth = 5, learning_rate = 0.1, class_weight = none	• Logistic Regression = 0.482 • Decision Tree = 0.519 • Random Forest = 0.541 • KNN = 0.436 • Xgboost = 0.472
Feature Importances	• `feature_importances_attributes` and eli5 suggested that the features `PAY_*` were most important • Permutation suggested that `PAY_1` , `PAY_2` , and `LIMIT_BAL` were most important • SHAP suggested that the features `PAY_1` , `PAY_2` , and `PAY_3` were most important	N/A
Scoring on Test Data	• We used our best Random Forest model (class_weight=balanced, max_depth=10) • Test f1-score was 0.538 • Our training score was slightly higher, suggesting possible overfitting • We should not trust our results too much or be overly optimistic about them	0.538

13.2 Concluding remarks

We were provided with a dataset containing information about a client's demographics and financial information in order to see if we could predict whether they would miss their bill payment in the next month. Through exploratory data analysis and preprocessing, we ended up with a training set of 38 columns/features and 24000 training observations. Due to class imbalance, we could not use accuracy as a metric; instead we chose f1 as it helps combine recall and precision.

A baseline dummy model gave an f1-score of 0. We then tried a linear model (logistic regression) and multiple non-linear models with hyperparameter optimization to see which one might give good performance. We found that the Random Forest Classifier, with max_depth=10, learning_rate=0.1, and class_weight=none gave us the best f1-score on training data (0.541).

Through analysis of feature importance, we found that the `PAY_*` features were driving portions of the prediction. This makes sense, as a client's historical repayment patterns would be a good indication of their future behaviour.

Finally, we scored our model on our test data and received an f1 of 0.538. Overall, we should not be too optimistic with our results. Our test f1-score indicates there is still a lot of misclassification. This model would likely need to be greatly improved upon if it were to be deployed in the real world.

13.3 Limitations and Future Directions

When comparing different models, we noticed that tree-based model seems to be appropriate for this question. We carried out the analysis with one of the gradient boosted trees, xgboost. Its perfomance did not beat the random forest model. If time and computation resources allows, future studies could be carried out using other gradient boosted trees such as light GBM or cat boost. These methods could learn from the mistake made by the previous trees and correct the mistakes, thus they might perform better.

It is worthwhile to mention that, in this study we focused on improving the f1 score, which is a combination of recall and precision. In practice, it would be better to have a conversation with the domain expert and see if f1 is indeed the metric to be optimized, if recall (i.e. capturing true positives) or if precision (ensuring that the false positive is low) is of higher priority. Depending on the client requirement, a different model could have been chosen. For example we noticed that analogy based KNN model performed well on precision.

One of the drawbacks of using random forest model is that the model is not as interpretable as linear models. Logistic regression did not perform well on the dataset. Several variables are correlated with each other to some extend, which could make the prediction of logistic regression not stable. Further feature engineering and feature selection (e.g. using PCA analysis to reduce the dimensionality of the dataset) could possibly help, preferably under the guidance of input from domain experts.

Finaly, during our analysis we observed that there is strong correlation between user's spending habits and payment habbits from one month to another. For example, if the person did not pay the entire bill this month, the bill is accumulated and the bill amount can be higher next month. Also, we noticed that the feature from certain months could be more important than other months(e.g. PAY_1 is the most important feature compared to other month's repayment status). These are strong indications that time series analysis could be employed to capture the pattern over the 6 months of period.

13.4 Final Test Score Report

According to our metric selection, we conclude that the best metric to use in this classification problem is the f1 score, which considering is the harmonic mean of both recall and precision. It considers the trade-off between recall and precision based on its equation $f1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} = \frac{\frac{TP}{TP+FP} \times \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}}$. In other words, f1 cares about both False Positive, people who can pay the credits but be predicted as cannot, and False Negative, people who cannot pay the credits but be predicted as can. Both these two cases will cause the credit card company loses profits for losing potential customers and being unable to get the credits back. To generalize and compare the model scoring metrics, we still show all the other classification scoring metrics' accuracy, precision, and recall results during the models' cross-validation and refit f1 with them during the hyperparameter tuning. We choose `RandomForestClassifier` as our best model based on the results and its f1 score of the test set is 0.538. It is a roughly good f1 score since the harmonic mean discourages hugely unequal values and extremely low values. It shows the precision and recall are balanced, and the False Positive and False Negative classes are eliminated.

14. Creating a data analysis pipeline (Challenging)

rubric={reasoning}

Your tasks:

- In 522 you learned how build a reproducible data analysis pipeline. Convert this notebook into scripts and 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.

15. Your takeaway from the course (Challenging)

rubric={reasoning}

Your tasks:

What is your biggest takeaway from this course?

Points: 0.25

Our takeaways

1. We cannot always rely on accuracy as a measure in classification, sometimes there is class imbalance so we need to look at confusion matrices and analyze our results carefully.
2. We should always be skeptical and critical when judging the models and we should not fully trust or rely on the results of our models.
3. Model transparency and interpretation is an important factor to help vlaidating and leveraging domain expert's knowledge to further imporve the model. A black-box model that performs very well does not mean it is the best model in practice.

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 ►► 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)

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: