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

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

GitHub URL : https://github.com/UBC-MDS/dsci_573_credit_default_nse_ark_ss

Contributors:

- Arjun Radhakrishnan
- Sneha Sunil
- Nikita Susan Easow

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.

In [2]:
```
import os

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier


from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler

import altair as alt
from sklearn.model_selection import (
    cross_val_score,
```

```
    cross_validate,
    train_test_split,
)

from sklearn.metrics import make_scorer

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.preprocessing import KBinsDiscretizer, PolynomialFeatures
```

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

As a part of this classification problem, we're trying to understand whether a credit card customer would default on the subsequent bill payment, given their payment history and other factors such as like age, gender, and the level of education.

There are a total of 24 featuers and a brief description of the type of features can be found below:

Categorical Features:

- MARRIAGE: Marital status

Ordinal Features:

- EDUCATION: Level of eduction

Binary Featus:

- SEX: Sex of the person. Although this field is not binary in nature, the data was recorded such that 1=male, 2=female.

Numeric Features:

- LIMIT_BAL: Amount of credit provided.
- ID: ID of the clint
- AGE: Age of the clint
- PAY_i, I RANGES FROM 0 TO 6: Whether credit is repayed or not for months from September to April, 2005.
- BILL_AMTi, i RANGES FROM 0 TO 6: Credit card bill for months from September to April, 2005.
- PAY_AMTi, i RANGES FROM 0 TO 6: Amount paid previously from September to April, 2005.

Some of the concerns analysing the data are:

1. The target name `default.payment.next.month` is not consistent as it used `.` over `_`. Also, PAY_i starts from 0 while BILL_AMTi and PAY_AMTi start from 1.
2. As per the metadata of the dataset, EDUCATION is defined with numbers 1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown. There is no clear reason why there are 2 different levels that signify unknown. Also, there is no clear reason why this is not combined with the category "Others". As a part of this analysis, we'll be merging values in 5 and 6 into 4. Out of the 4 possible values indication graduate school, university, high school, and others, others is a considerable minority class. To understand whether they should be higher or lower on the scale of ordinality, we'll rely on the EDA.
3. Although the current encoding for `EDUCATION` might be alright since the feature is ordinal, for better interpretability, it would be best if we can encode them such that higher qualifications take up higher values in the encoding scale. To perform this encoding, later on, it is best to replace the numbers with actual categories.
4. As per the metadata of the dataset, `MARRIAGE` is defined with numbers 1=married, 2=single, and 3=others, but the data contains 0 as well. We'll combine 0 along with the "Others" category as we did with `EDUCATION`.
5. Since `MARRIAGE` is not technically an ordinal feature, it'll be good to convert it to actual categories rather than the current representation of numbers because there is no inherent ordering between the categories. One Hot encoding would be preferred here.
6. Similar to `MARRIAGE` and `EDUCATION`, although `PAY_i` should be in one of -1, 1, 2, ..., 9, the data contains 0 and -2. Also, if the value here represents the repayment delays, ideally 0 should represent no payment delay. As we cannot drop these columns as there is a huge chunk of data with these undocumented values, we're transforming this feature to make -1 and -2 to 0.
7. To improve the final interpretation, the feature `SEX` can be transformed into its corresponding categories so that it can be binary encoded instead of encoding as 1s and 2s.

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

In [4]:
```
# No NaNs in any of the cols.
pd.isnull(credit_df).sum()
```

Out[4]:
```
ID                            0
LIMIT_BAL                     0
SEX                           0
EDUCATION                     0
MARRIAGE                      0
AGE                           0
PAY_0                         0
PAY_2                         0
PAY_3                         0
PAY_4                         0
PAY_5                         0
PAY_6                         0
BILL_AMT1                     0
BILL_AMT2                     0
BILL_AMT3                     0
BILL_AMT4                     0
BILL_AMT5                     0
BILL_AMT6                     0
PAY_AMT1                      0
PAY_AMT2                      0
PAY_AMT3                      0
PAY_AMT4                      0
PAY_AMT5                      0
PAY_AMT6                      0
default.payment.next.month    0
dtype: int64
```

In [5]:
```
# Feature Info
credit_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   ID                          30000 non-null  int64  
 1   LIMIT_BAL                   30000 non-null  float64
 2   SEX                         30000 non-null  int64  
 3   EDUCATION                   30000 non-null  int64  
 4   MARRIAGE                    30000 non-null  int64  
 5   AGE                         30000 non-null  int64  
 6   PAY_0                       30000 non-null  int64  
 7   PAY_2                       30000 non-null  int64  
 8   PAY_3                       30000 non-null  int64  
 9   PAY_4                       30000 non-null  int64  
 10  PAY_5                       30000 non-null  int64  
 11  PAY_6                       30000 non-null  int64  
 12  BILL_AMT1                   30000 non-null  float64
 13  BILL_AMT2                   30000 non-null  float64
 14  BILL_AMT3                   30000 non-null  float64
 15  BILL_AMT4                   30000 non-null  float64
 16  BILL_AMT5                   30000 non-null  float64
 17  BILL_AMT6                   30000 non-null  float64
 18  PAY_AMT1                    30000 non-null  float64
 19  PAY_AMT2                    30000 non-null  float64
 20  PAY_AMT3                    30000 non-null  float64
 21  PAY_AMT4                    30000 non-null  float64
 22  PAY_AMT5                    30000 non-null  float64
 23  PAY_AMT6                    30000 non-null  float64
 24  default.payment.next.month  30000 non-null  int64  
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

In [6]:
```python
# Feature describe
credit_df.describe()
```

Out[6]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000

8 rows × 25 columns

In [7]:
```python
# First 5 rows
credit_df.head()
```

Out[7]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5
0	1	20000.0	2	2	1	24	2	2	-1	-1	...	0.0	0.0
1	2	120000.0	2	2	2	26	-1	2	0	0	...	3272.0	3455.0
2	3	90000.0	2	2	2	34	0	0	0	0	...	14331.0	14948.0
3	4	50000.0	2	2	1	37	0	0	0	0	...	28314.0	28959.0
4	5	50000.0	1	2	1	57	-1	0	-1	0	...	20940.0	19146.0

5 rows × 25 columns

In [8]:
```
# Last 5 rows
credit_df.tail()
```

Out[8]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BIL
29995	29996	220000.0	1	3	1	39	0	0	0	0	...	88004.0	
29996	29997	150000.0	1	3	2	43	-1	-1	-1	-1	...	8979.0	
29997	29998	30000.0	1	2	2	37	4	3	2	-1	...	20878.0	
29998	29999	80000.0	1	3	1	41	1	-1	0	0	...	52774.0	
29999	30000	50000.0	1	2	1	46	0	0	0	0	...	36535.0	

5 rows × 25 columns

In [9]:
```
# Unique values for Sex
credit_df["SEX"].unique()
```

Out[9]:
```
array([2, 1], dtype=int64)
```

In [10]:
```
# Unique values for EDUCATION
credit_df["EDUCATION"].unique()
```

Out[10]:
```
array([2, 1, 3, 5, 4, 6, 0], dtype=int64)
```

In [11]:
```
# Unique values for MARRIAGE
credit_df["MARRIAGE"].unique()
```

Out[11]:
```
array([1, 2, 3, 0], dtype=int64)
```

In [12]:
```
# Unique values for PAY_0
credit_df["PAY_0"].unique()
```

Out[12]:
```
array([ 2, -1,  0, -2,  1,  3,  4,  8,  7,  5,  6], dtype=int64)
```

In [13]:
```
# Address Concern 1: Inconsistent syntax in column naming
credit_df = credit_df.rename(
    columns={
        "default.payment.next.month": "default_payment_next_month",
        "PAY_0": "PAY_1",
    }
)
```

In [14]:
```
# Address Concern 2 and 3: Remapping values in the EDUCATION feature
credit_df["EDUCATION"] = credit_df["EDUCATION"].apply(
```

```
    lambda cell_value:
    "Graduate" if cell_value == 1
    else "Undergrad" if cell_value == 2
    else "HighSchool" if cell_value == 3
    else "Others")
```

In [15]:
```
# Address Concern 4 and 5: Remapping values in the MARRIAGE feature
credit_df["MARRIAGE"] = credit_df["MARRIAGE"].apply(
    lambda cell_value:
    "Married" if cell_value == 1
    else "Single" if cell_value == 2
    else "Others")
```

In [16]:
```
# Address Concern 6: Remapping values in the PAY_i feature
for i in range(1, 7):
    query = f'PAY_{i}==-2 or PAY_{i}==-1 or PAY_{i}==0'
    credit_df.loc[credit_df.query(query).index, f'PAY_{i}'] = 0
```

In [17]:
```
# Address Concern 7: Remapping values in the SEX feature
credit_df["SEX"] = credit_df["SEX"].apply(
    lambda cell_value:
    "Male" if cell_value == 1
    else "Female")
```

In [18]:
```
# Drop ID column as it is useless. Doing it now itself to reduce the load on the column transforr
credit_df = credit_df.drop('ID', axis=1)
# change target data type
credit_df["default_payment_next_month"] = credit_df["default_payment_next_month"].astype("categoi
```

In [19]:
```
# Final State of the data frame after transformations
credit_df.head(7)
```

Out[19]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_
0	20000.0	Female	Undergrad	Married	24	2	2	0	0	0	...	0.0	
1	120000.0	Female	Undergrad	Single	26	0	2	0	0	0	...	3272.0	
2	90000.0	Female	Undergrad	Single	34	0	0	0	0	0	...	14331.0	1
3	50000.0	Female	Undergrad	Married	37	0	0	0	0	0	...	28314.0	2
4	50000.0	Male	Undergrad	Married	57	0	0	0	0	0	...	20940.0	1
5	50000.0	Male	Graduate	Single	37	0	0	0	0	0	...	19394.0	1
6	500000.0	Male	Graduate	Single	29	0	0	0	0	0	...	542653.0	48

7 rows × 24 columns

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 [20]:
```
train_df, test_df = train_test_split(credit_df, test_size=0.6, random_state=573)
X_train, y_train = train_df.drop(columns=['default_payment_next_month']), train_df['default_paym
X_test, y_test = test_df.drop(columns=['default_payment_next_month']), test_df['default_payment_
```

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

ANSWER 3.1

In [21]:
```
# Proportion of credit defaulters
target_classes = y_train.value_counts()
round(target_classes[1] / (target_classes[0] + target_classes[1]) * 100, 3)
```

Out[21]:
```
22.258
```

In [22]:
```
# Proportion of credit non-defaulters
round(target_classes[0] / (target_classes[0] + target_classes[1]) * 100, 3)
```

Out[22]:
```
77.742
```

Summary Statistic 1:
In the dataset, there is a class imbalance. While around 77.84 % of the people do not default on the bill payments, only 22.16% of the population defaults on their bill payments.

In [23]:
```
education_matrix = pd.crosstab(train_df.EDUCATION, train_df.default_payment_next_month)
education_matrix['percent_default'] = (education_matrix[1]/(education_matrix[0] + education_matr
education_matrix.sort_values(by='percent_default', ascending=False)
```

Out[23]:

default_payment_next_month	0	1	percent_default
EDUCATION			
HighSchool	1460	495	25.319693
Undergrad	4218	1335	24.041059
Graduate	3480	828	19.220056
Others	171	13	7.065217

Summary Statistic 2:

People with higher educational qualifications are are associated with lesser defaults in credit re-payments.

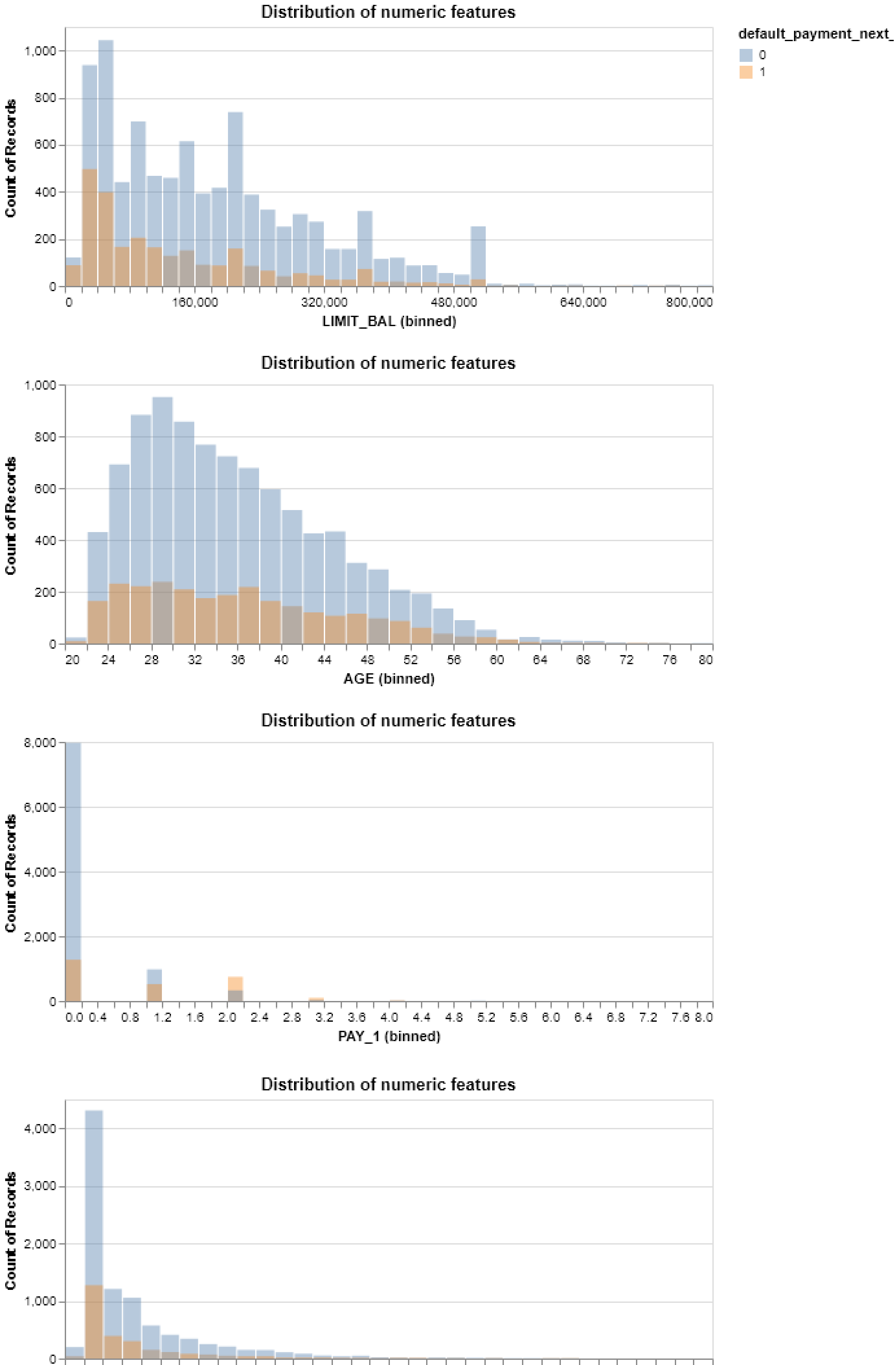
In [24]:

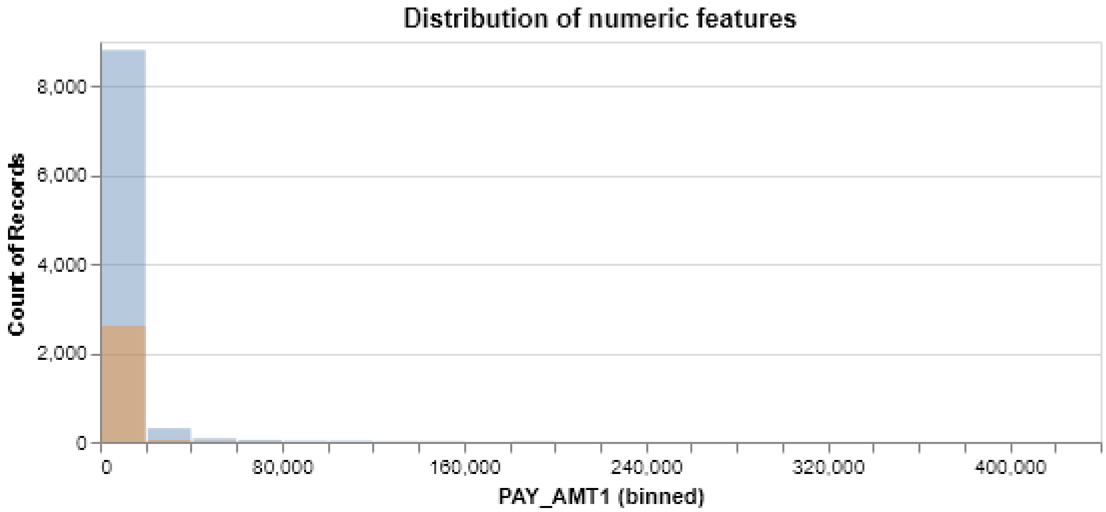
```
# Viz 1:
alt.data_transformers.disable_max_rows()
import altair as alt

numeric_cols = ["LIMIT_BAL", "AGE", "PAY_1", "BILL_AMT1", "PAY_AMT1"]

alt.Chart(train_df, title="Distribution of numeric features").mark_bar(opacity=0.4).encode(
    x=alt.X(alt.repeat(), type="quantitative", bin=alt.Bin(maxbins=40)),
    y=alt.Y("count()", stack=False),
    color="default_payment_next_month",
).properties(width=500, height=200).repeat(numeric_cols, columns=1)
```

Out[24]:


Distribution of numeric features
default_payment_next_
0
1
Count of Records
LIMIT_BAL (binned)
Distribution of numeric features
Count of Records
AGE (binned)
Distribution of numeric features
Count of Records
PAY_1 (binned)
Distribution of numeric features
Count of Records



`LIMIT_BAL` and `AGE` seems to have right-skewed distributions, with more people defaulting on the lower scales of the features and gradually declining as we move towards the right of the scale.

In [25]:
```
train_df_copy = train_df.copy()
train_df_copy["default_payment_next_month"] = train_df_copy["default_payment_next_month"].astype
corr_matrix = train_df_copy.corr().style.background_gradient()
corr_matrix
```

Out[25]:

	LIMIT_BAL	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_
LIMIT_BAL	1.000000	0.154492	-0.169637	-0.208634	-0.196010	-0.188107	-0.172252	-0.16315
AGE	0.154492	1.000000	0.006405	-0.006161	-0.011477	-0.001359	-0.010777	-0.01992
PAY_1	-0.169637	0.006405	1.000000	0.695877	0.512101	0.466081	0.427304	0.37708
PAY_2	-0.208634	-0.006161	0.695877	1.000000	0.665067	0.518167	0.466993	0.41414
PAY_3	-0.196010	-0.011477	0.512101	0.665067	1.000000	0.680737	0.554256	0.49924
PAY_4	-0.188107	-0.001359	0.466081	0.518167	0.680737	1.000000	0.756111	0.62331
PAY_5	-0.172252	-0.010777	0.427304	0.466993	0.554256	0.756111	1.000000	0.74895
PAY_6	-0.163153	-0.019921	0.377085	0.414146	0.499245	0.623312	0.748953	1.00000
BILL_AMT1	0.272810	0.055058	-0.005229	-0.001494	-0.029520	-0.032368	-0.021478	-0.02250
BILL_AMT2	0.270645	0.055220	0.004257	0.003525	-0.010300	-0.020278	-0.011871	-0.01352
BILL_AMT3	0.274812	0.055120	0.008647	0.009998	-0.005795	-0.005353	0.000845	-0.00145
BILL_AMT4	0.287978	0.052706	0.020708	0.023481	0.011184	0.007908	0.020678	0.01818
BILL_AMT5	0.289247	0.054457	0.029624	0.032171	0.021003	0.020610	0.033609	0.03881
BILL_AMT6	0.282965	0.050436	0.027382	0.033796	0.025420	0.025879	0.040261	0.04366
PAY_AMT1	0.216115	0.024068	-0.079856	-0.102459	-0.044140	-0.060613	-0.058041	-0.05268
PAY_AMT2	0.175175	0.020137	-0.055697	-0.059091	-0.074975	-0.037875	-0.037904	-0.03807
PAY_AMT3	0.209861	0.026567	-0.059619	-0.066763	-0.059851	-0.078013	-0.042366	-0.04008
PAY_AMT4	0.201190	0.022983	-0.057923	-0.050617	-0.048041	-0.055220	-0.065364	-0.03269
PAY_AMT5	0.215522	0.015554	-0.064696	-0.059842	-0.056076	-0.058783	-0.056764	-0.06671
PAY_AMT6	0.224077	0.021862	-0.046242	-0.042270	-0.054360	-0.054445	-0.047516	-0.04239
default_payment_next_month	-0.144233	0.020328	0.396090	0.332869	0.294801	0.289731	0.279805	0.25742

Features `PAY_1`, `PAY_2`, `PAY_3`, `PAY_4`, `PAY_5`, and `PAY_6` are positively correlated with each other. Similarly, features `BILL_AMT1`, `BILL_AMT2`, `BILL_AMT3`, `BILL_AMT4`, `BILL_AMT5`, and `BILL_AMT6` are positively heavily correlated with each other

ANSWER 3.2

From the initial analysis of the data, we saw that although there are no missing values in the dataset, certain features (`MARRIAGE`, `EDUCATION`, `PAY_i`) had undocumented values that needed manual corrections. This was done as a part of the pre-processing step. From summary statistic 1, we can see that there is a clear class imbalance with 22.16% of the people defaulting the credit repayment. From the EDA and summary statistic 2, we see that lower re-payment defaults are associated with people with higher qualifications. As the percentage of people in "Others" is fewer compared to the other categories and since they have a lower percent of in terms of credit defaults, it makes sense to place them on a higher level on the ordinal scale. Although this might not hold in all cases, we're hoping that this assumption holds in the production data as well. Since there is an inherent order in the `PAY_i` feature, it is best to consider this as an ordinal feature that is already encoded. From the correlation matrix, we see that time series features such as `PAY_i` and `BILL_AMTi` are heavily correlated. Surprisingly, the time series data for the features `PAY_AMTi` have minimal correlation when compared to the correlations visible in the clusters `PAY_i` and `BILL_AMTi`. Against the target `default_payment_next_month`, features `PAY_1` and `PAY_2` seem to be the highest correlated. From the visualizations of the key numeric features, we can see that `LIMIT_BAL` and `AGE` seems to have right-skewed distributions, with more people defaulting on the lower scales of the features and gradually declining as we move towards the right of the scale. From the distribution for credit repayment for September, we can see that the distribution is similar to an exponential distribution with a peak at 0 and declining as we look at the number of people who have consistently missed the repayment.

ANSWER 3.3

In this scenario, as we're interested in predicting whether the client would default or not in their credit repayment, the positive class is that the customer defaults.

- False Positives: Wrongly predicting that the customer would default when they actually don't.
- False Negatives: Wrongly predicting that the customer would repay the credit when they actually don't.

As there is a class imbalance, relying solely on accuracy alone is not ideal.

- Reducing false positives is important as the company's reputation and customer satisfaction are at stake. Too many wrong predictions and escalations could drive down the business and the number of clients using the service.
- Reducing false negatives is also crucial as it would help the company take the necessary steps to prepare for when the client misses the repayment and plan for risk management.

Hence, we'll be optimizing the F1 score while at the same time looking at Recall, Precision, and Accuracy.

In [26]:
```
# Primary metric is F1. We'll be scoring the models on the below metrics as well.
scoring_metrics = ["f1", "recall", "accuracy", "precision"]
```

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

As part of feature engineering, we're binning the `AGE` to the narrow possible values these features take to improve model performance.

In [27]:
```
# Referenced from Labs and Lecture Notes of DSCI 571 and DSCI 573
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 [28]:

```
# Without feature Engineering
scalable_features = [
        "LIMIT_BAL",
        "AGE",
        "PAY_1",
        "PAY_2",
        "PAY_3",
        "PAY_4",
        "PAY_5",
        "PAY_6",
        "BILL_AMT1",
        "BILL_AMT2",
        "BILL_AMT3",
        "BILL_AMT4",
        "BILL_AMT5",
        "BILL_AMT6",
        "PAY_AMT1",
        "PAY_AMT2",
        "PAY_AMT3",
        "PAY_AMT4",
        "PAY_AMT5",
        "PAY_AMT6",
    ]
categorical_feats = ["MARRIAGE"]
binary_feats = ["SEX"]
ordinal_features = ["EDUCATION"]

education_levels = [
    "HighSchool",
    "Undergrad",
    "Graduate",
    "Others",
]

ordinal_transformer = OrdinalEncoder(categories=[education_levels], dtype=int)

column_transformer = make_column_transformer(
        (StandardScaler(), scalable_features),
        (OrdinalEncoder(categories=[education_levels], dtype=int), ordinal_features),
        (OneHotEncoder(sparse=False, handle_unknown="ignore"), categorical_feats),
```

```
        (
            OneHotEncoder(sparse=False, handle_unknown="ignore", drop="if_binary"),
            binary_feats,
        )
    )

lr_pipe = make_pipeline(
    column_transformer, LogisticRegression(random_state=573, n_jobs=-1, max_iter=1000)
)

mean_std_cross_val_scores(
        lr_pipe,
        X_train,
        y_train,
        scoring=scoring_metrics,
        return_train_score=True,
    )
```

Out[28]:

```
fit_time           1.252 (+/- 0.834)
score_time         0.015 (+/- 0.013)
test_f1            0.440 (+/- 0.028)
train_f1           0.447 (+/- 0.012)
test_recall        0.325 (+/- 0.024)
train_recall       0.332 (+/- 0.013)
test_accuracy      0.816 (+/- 0.007)
train_accuracy     0.818 (+/- 0.002)
test_precision     0.682 (+/- 0.029)
train_precision    0.688 (+/- 0.008)
dtype: object
```

In [29]:

```
# With Feature Engineering
scalable_features = [
        "LIMIT_BAL",
        "PAY_1",
        "PAY_2",
        "PAY_3",
        "PAY_4",
        "PAY_5",
        "PAY_6",
        "BILL_AMT1",
        "BILL_AMT2",
        "BILL_AMT3",
        "BILL_AMT4",
        "BILL_AMT5",
        "BILL_AMT6",
        "PAY_AMT1",
        "PAY_AMT2",
        "PAY_AMT3",
        "PAY_AMT4",
        "PAY_AMT5",
        "PAY_AMT6",
    ]

categorical_feats = ["MARRIAGE"]
binary_feats = ["SEX"]
ordinal_features = ["EDUCATION"]
discretization_feats = ["AGE"]
education_levels = [
    "HighSchool",
    "Undergrad",
    "Graduate",
    "Others",
```

```
]

ordinal_transformer = OrdinalEncoder(categories=[education_levels], dtype=int)

column_transformer = make_column_transformer(
        (StandardScaler(), scalable_features),
        (KBinsDiscretizer(encode="onehot"), discretization_feats),
        (OrdinalEncoder(categories=[education_levels], dtype=int), ordinal_features),
        (OneHotEncoder(sparse=False, handle_unknown="ignore"), categorical_feats),
        (
            OneHotEncoder(sparse=False, handle_unknown="ignore", drop="if_binary"),
            binary_feats,
        )
    )

lr_pipe = make_pipeline(
    column_transformer, LogisticRegression(random_state=573, n_jobs=-1, max_iter=1000)
)

mean_std_cross_val_scores(
        lr_pipe,
        X_train,
        y_train,
        scoring=scoring_metrics,
        return_train_score=True,
    )
```

Out[29]:

```
fit_time            1.045 (+/- 0.009)
score_time          0.011 (+/- 0.003)
test_f1             0.445 (+/- 0.022)
train_f1            0.448 (+/- 0.014)
test_recall         0.330 (+/- 0.019)
train_recall        0.332 (+/- 0.014)
test_accuracy       0.817 (+/- 0.006)
train_accuracy      0.818 (+/- 0.002)
test_precision      0.683 (+/- 0.027)
train_precision     0.689 (+/- 0.006)
dtype: object
```

With feature engineering,

- The test F1 score improved from 0.440 to 0.445
- The test precision improved from 0.682 to 0.683
- The test recall improved from 0.325 to 0.330

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

In [30]:

```
# Numeric Features that needs scaling.
scalable_features = [
        "LIMIT_BAL",
        "PAY_1",
        "PAY_2",
        "PAY_3",
        "PAY_4",
        "PAY_5",
        "PAY_6",
        "BILL_AMT1",
        "BILL_AMT2",
        "BILL_AMT3",
        "BILL_AMT4",
        "BILL_AMT5",
        "BILL_AMT6",
        "PAY_AMT1",
        "PAY_AMT2",
        "PAY_AMT3",
        "PAY_AMT4",
        "PAY_AMT5",
        "PAY_AMT6",
    ]

# Categorical Feature that needs to be one hot encoded.
categorical_feats = ["MARRIAGE"]

# Binary Feature that needs to be one hot encoded and dropping the extra column.
binary_feats = ["SEX"]

# Ordinal Feature with inherent ordering specified by education_levels.
ordinal_features = ["EDUCATION"]
education_levels = [
    "HighSchool",
    "Undergrad",
    "Graduate",
    "Others",
]

# Feature engineering
discretization_feats = ["AGE"]

ordinal_transformer = OrdinalEncoder(categories=[education_levels], dtype=int)

column_transformer = make_column_transformer(
        (StandardScaler(), scalable_features),
        (KBinsDiscretizer(n_bins=5, encode="onehot"), discretization_feats),
        (OrdinalEncoder(categories=[education_levels], dtype=int), ordinal_features),
        (OneHotEncoder(sparse=False, handle_unknown="ignore"), categorical_feats),
        (
            OneHotEncoder(sparse=False, handle_unknown="ignore", drop="if_binary"),
            binary_feats,
        )
    )

column_transformer.fit(X_train)
column_transformer.verbose_feature_names_out = False
X_train_enc = pd.DataFrame(column_transformer.transform(X_train), index=X_train.index, columns=co
X_train_enc
```

Out[30]:

	LIMIT_BAL	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_
11997	-0.595569	0.857150	2.099797	-0.393042	-0.344872	-0.312016	-0.315299	-0.003847	-0.490496	-0.4
2943	-1.137066	-0.477282	-0.407798	2.146125	-0.344872	-0.312016	-0.315299	-0.428251	-0.397239	-0.3
9784	-0.672926	-0.477282	-0.407798	-0.393042	-0.344872	-0.312016	-0.315299	-0.257637	-0.254567	-0.1
27216	-0.286141	2.191581	2.099797	2.146125	2.242477	2.420973	2.429415	0.249834	0.303415	0.3
29783	-0.595569	0.857150	2.099797	-0.393042	-0.344872	-0.312016	2.429415	-0.072703	-0.062084	-0.0
...	...	...	...	...	...	...	...	...	...	
8144	-0.904996	-0.477282	-0.407798	2.146125	2.242477	-0.312016	-0.315299	-0.484119	-0.424591	-0.3
14870	-0.131428	0.857150	2.099797	-0.393042	-0.344872	-0.312016	-0.315299	-0.547697	-0.618833	-0.6
29361	1.106281	-0.477282	2.099797	-0.393042	-0.344872	-0.312016	-0.315299	-0.692336	-0.686836	-0.6
9822	-1.137066	-0.477282	-0.407798	-0.393042	-0.344872	-0.312016	-0.315299	-0.428773	-0.411432	-0.6
16928	-0.904996	-0.477282	-0.407798	-0.393042	-0.344872	-0.312016	-0.315299	-0.046809	-0.350844	-0.5

12000 rows × 29 columns

6. Baseline model

rubric={accuracy}

Your tasks:

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

Points: 2

In [31]:

```
results = {}
```

In [32]:

```
dummy_pipe = make_pipeline(
    column_transformer,
    DummyClassifier()
)

results["Dummy"] = mean_std_cross_val_scores(
    dummy_pipe, X_train, y_train, scoring=scoring_metrics, return_train_score=True
)

pd.DataFrame(results)
```

```
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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[32]:

	Dummy
fit_time	0.018 (+/- 0.001)
score_time	0.010 (+/- 0.001)
test_f1	0.000 (+/- 0.000)
train_f1	0.000 (+/- 0.000)
test_recall	0.000 (+/- 0.000)
train_recall	0.000 (+/- 0.000)
test_accuracy	0.777 (+/- 0.000)
train_accuracy	0.777 (+/- 0.000)
test_precision	0.000 (+/- 0.000)
train_precision	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

Compared to our baseline dummy classifier, logistic regression is performing relatively well in terms of accuracy, recall, precision, and F1 scores. Without hyperparameter optimization, we're getting an accuracy of 0.816 and an F1 score of 0.440. Since we're more interested in increasing F1 scores, we conducted hyperparameter optimization such that the new hyperparameters are now optimized for F1 scores. This improved the F1 score from 0.440 to 0.530, but consequently, the accuracy of the model dropped from 0.816 to 0.777, which is the same accuracy as the dummy classifier. The hyperparameter optimization also improved recall from 0.325 to 0.566 at the cost of lowering the precision from 0.682 to 0.499. Since the test and train cross-validation scores are relatively close, the model is not overfitting. The standard deviation of the overall scores seems to be minimal, which is good. The model is time efficient as well, as it took just around a second in terms of the `fit_time`.

In [33]:
```
lr_pipe = make_pipeline(
    column_transformer, LogisticRegression(random_state=573, n_jobs=-1, max_iter=1000)
)
```

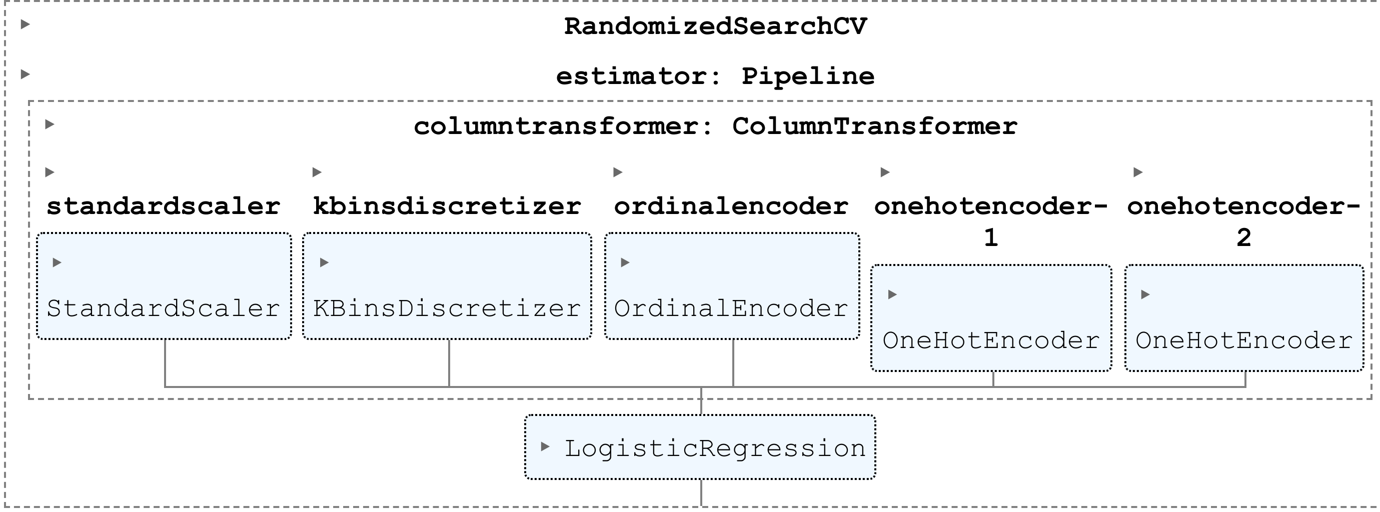
In [34]:
```
results["LR"] = mean_std_cross_val_scores(
        lr_pipe,
        X_train,
        y_train,
        scoring=scoring_metrics,
        return_train_score=True,
    )
```

In [35]:
```
from scipy.stats import uniform

grid_params = {
        "logisticregression__class_weight": [None, "balanced"],
        "logisticregression__C": uniform(1e-3, 1e3),
    }

lr_grid_search = RandomizedSearchCV(
        lr_pipe,
        param_distributions=grid_params,
        cv=10,
        n_jobs=-1,
        n_iter=100,
        random_state=573,
        scoring=scoring_metrics,
        refit="f1",
    )
lr_grid_search.fit(X_train, y_train)
```

Out[35]:



In [36]:

```
# Display best 3
results_df = pd.DataFrame(lr_grid_search.cv_results_)[
    [
        "mean_test_f1",
        "mean_test_recall",
        "mean_test_precision",
        "mean_test_accuracy",
        "rank_test_f1",
    ]
].set_index("rank_test_f1").sort_index().T
results_df.iloc[:, :3]
```

Out[36]:

rank_test_f1	1	1	1
mean_test_f1	0.531149	0.531149	0.531149
mean_test_recall	0.566077	0.566077	0.566077
mean_test_precision	0.500725	0.500725	0.500725
mean_test_accuracy	0.777583	0.777583	0.777583

In [37]:

```
print(f"Best F1 Score: {lr_grid_search.best_score_}")
print(f"Best C for Logestic Regression: {lr_grid_search.best_params_['logisticregression__C']}")
print(f"Best class_weight for Logestic Regression: {str.capitalize(lr_grid_search.best_params_['
```

```
Best F1 Score: 0.5311492749492353
Best C for Logestic Regression: 469.54077005672974
Best class_weight for Logestic Regression: Balanced
```

In [38]:

```
results["LR_Optimized"] = mean_std_cross_val_scores(
    lr_grid_search.best_estimator_, X_train, y_train, scoring=scoring_metrics, return_train_score
)

pd.DataFrame(results)
```

Out[38]:

	Dummy	LR	LR_Optimized
fit_time	0.018 (+/- 0.001)	0.997 (+/- 0.095)	0.679 (+/- 0.013)
score_time	0.010 (+/- 0.001)	0.011 (+/- 0.001)	0.010 (+/- 0.000)
test_f1	0.000 (+/- 0.000)	0.445 (+/- 0.022)	0.531 (+/- 0.018)
train_f1	0.000 (+/- 0.000)	0.448 (+/- 0.014)	0.534 (+/- 0.003)
test_recall	0.000 (+/- 0.000)	0.330 (+/- 0.019)	0.567 (+/- 0.019)
train_recall	0.000 (+/- 0.000)	0.332 (+/- 0.014)	0.569 (+/- 0.005)
test_accuracy	0.777 (+/- 0.000)	0.817 (+/- 0.006)	0.777 (+/- 0.009)
train_accuracy	0.777 (+/- 0.000)	0.818 (+/- 0.002)	0.779 (+/- 0.002)
test_precision	0.000 (+/- 0.000)	0.683 (+/- 0.027)	0.499 (+/- 0.018)
train_precision	0.000 (+/- 0.000)	0.689 (+/- 0.006)	0.504 (+/- 0.004)

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

On comparing the results of all the models so far, we can see that:

- With default hyperparameters, we can see that GradientBoostingClassifier has the highest F1 score of 0.482, SCV at 0.478, and RandomForestClassifier at 0.468. Overall, the F1 score of Optimized Logistic Regression is much better (0.530) when compared to the others. Without optimization, the F1 score of Logistic Regression (0.440) is the lowest compared to other baseline models. Seems like Logistic Regression with default hyperparameters was slightly underfitting as the new value of C increased, the model complexity and the score increased. From the newly added baseline models, as both the train and test scores of SVC are quite low, it could be underfitting.

- Among baseline models, GradientBoostingClassifier has the highest accuracy while RandomForestClassifier has the lowest. Once Logistic Regression was optimized for the F1 score, among Optimized Logistic Regression and other baseline models, Optimized Logistic Regression had the lowest accuracy at 0.777.

- Among baseline models, GradientBoostingClassifier has the highest recall while baseline Logistic Regression has the lowest. Once Logistic Regression was optimized for the F1 score, among Optimized Logistic Regression and other baseline models, Optimized Logistic Regression has the highest recall at 0.777.

- Among all models, RandomForestClassifier seems to be overfitting the most as the gap between the train and test scores is wider. Looking at other models, they seem to be much less overfitting as the test and train cross-validation scores are comparable. The fit time of GradientBoostingClassifier is the highest followed by SVC. SVC has the highest scoring time with RandomForestClassifier as the second.

As of now, the optimized linear model Logistic Regression is performing the best, both in terms of the fit times as well as the F1 score.

In [39]:
```
from sklearn.ensemble import GradientBoostingClassifier
models = {
    "GBC": make_pipeline(column_transformer, GradientBoostingClassifier(random_state=573)),
    "SVC": make_pipeline(column_transformer, SVC(random_state=573)),
    "RFC": make_pipeline(column_transformer, RandomForestClassifier(random_state=573))
}
```

In [40]:
```
for (name, model) in models.items():
    results[name] = mean_std_cross_val_scores(
        model, X_train, y_train, return_train_score=True, scoring=scoring_metrics
    )
```

In [41]:
```
pd.DataFrame(results).T
```

Out[41]:

	fit_time	score_time	test_f1	train_f1	test_recall	train_recall	test_accuracy	train_accuracy	test_pr
Dummy	0.018 (+/- 0.001)	0.010 (+/- 0.001)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.777 (+/- 0.000)	0.777 (+/- 0.000)	0.0
LR	0.997 (+/- 0.095)	0.011 (+/- 0.001)	0.445 (+/- 0.022)	0.448 (+/- 0.014)	0.330 (+/- 0.019)	0.332 (+/- 0.014)	0.817 (+/- 0.006)	0.818 (+/- 0.002)	0.6
LR_Optimized	0.679 (+/- 0.013)	0.010 (+/- 0.000)	0.531 (+/- 0.018)	0.534 (+/- 0.003)	0.567 (+/- 0.019)	0.569 (+/- 0.005)	0.777 (+/- 0.009)	0.779 (+/- 0.002)	0.4
GBC	2.368 (+/- 0.021)	0.013 (+/- 0.001)	0.482 (+/- 0.023)	0.520 (+/- 0.005)	0.378 (+/- 0.024)	0.408 (+/- 0.005)	0.819 (+/- 0.005)	0.832 (+/- 0.001)	0.6
SVC	2.189 (+/- 0.041)	0.614 (+/- 0.024)	0.479 (+/- 0.022)	0.505 (+/- 0.006)	0.373 (+/- 0.023)	0.395 (+/- 0.009)	0.820 (+/- 0.005)	0.828 (+/- 0.001)	0.6
RFC	1.437 (+/- 0.029)	0.046 (+/- 0.001)	0.469 (+/- 0.026)	0.997 (+/- 0.000)	0.370 (+/- 0.025)	0.996 (+/- 0.001)	0.814 (+/- 0.007)	0.999 (+/- 0.000)	0.6

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

Initially, there were 29 features of which 24 were selected and the rest were eliminated using `RFECV`. From looking at the scores, we can see that apart from SVC, the scores of all the other models have slightly reduced. Since there are only 29 features in total, we have decided to keep all of them for better scores.

In [42]:
```
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression, Ridge

fs_models = {
    "Dummy_RFECV": DummyClassifier(),
    "LR_RFECV" : LogisticRegression(random_state=573, n_jobs=-1, max_iter=1000),
    "LR_Optimized_RFECV": LogisticRegression(
        random_state=573,
        n_jobs=-1,
        max_iter=1000,
        C=lr_grid_search.best_params_["logisticregression__C"],
        class_weight=lr_grid_search.best_params_["logisticregression__class_weight"],
    ),
    "GBC_RFECV": GradientBoostingClassifier(random_state=573),
    "SVC_RFECV": SVC(random_state=573),
    "RFC_RFECV": RandomForestClassifier(random_state=573)
}

rfecv_results = {}

print(f'Total Number of features: {len(column_transformer.get_feature_names_out())}')

for (name, model) in fs_models.items():
    pipe_rfecv = make_pipeline(column_transformer, RFECV(Ridge()), model)
    rfecv_results[name] = mean_std_cross_val_scores(
        pipe_rfecv, X_train, y_train, return_train_score=True, scoring=scoring_metrics
    )

pipe_rfecv.fit(X_train, y_train)
print(f'After Feature Selection: {pipe_rfecv.named_steps["rfecv"].n_features_}')
```

```
Total Number of features: 29
```

```
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\sklearn\metrics\_classification.py:1334: Un
definedMetricWarning: 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))
```

```
After Feature Selection: 24
```

In [43]:

```python
pd.DataFrame({**results, **rfecv_results}).T
```

Out[43]:

	fit_time	score_time	test_f1	train_f1	test_recall	train_recall	test_accuracy	train_accuracy
Dummy	0.018 (+/- 0.001)	0.010 (+/- 0.001)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.777 (+/- 0.000)	0.777 (+/- 0.000)
LR	0.997 (+/- 0.095)	0.011 (+/- 0.001)	0.445 (+/- 0.022)	0.448 (+/- 0.014)	0.330 (+/- 0.019)	0.332 (+/- 0.014)	0.817 (+/- 0.006)	0.818 (+/- 0.002)
LR_Optimized	0.679 (+/- 0.013)	0.010 (+/- 0.000)	0.531 (+/- 0.018)	0.534 (+/- 0.003)	0.567 (+/- 0.019)	0.569 (+/- 0.005)	0.777 (+/- 0.009)	0.779 (+/- 0.002)
GBC	2.368 (+/- 0.021)	0.013 (+/- 0.001)	0.482 (+/- 0.023)	0.520 (+/- 0.005)	0.378 (+/- 0.024)	0.408 (+/- 0.005)	0.819 (+/- 0.005)	0.832 (+/- 0.001)
SVC	2.189 (+/- 0.041)	0.614 (+/- 0.024)	0.479 (+/- 0.022)	0.505 (+/- 0.006)	0.373 (+/- 0.023)	0.395 (+/- 0.009)	0.820 (+/- 0.005)	0.828 (+/- 0.001)
RFC	1.437 (+/- 0.029)	0.046 (+/- 0.001)	0.469 (+/- 0.026)	0.997 (+/- 0.000)	0.370 (+/- 0.025)	0.996 (+/- 0.001)	0.814 (+/- 0.007)	0.999 (+/- 0.000)
Dummy_RFECV	0.330 (+/- 0.019)	0.009 (+/- 0.000)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.777 (+/- 0.000)	0.777 (+/- 0.000)
LR_RFECV	0.837 (+/- 0.271)	0.010 (+/- 0.001)	0.442 (+/- 0.026)	0.447 (+/- 0.014)	0.327 (+/- 0.022)	0.331 (+/- 0.015)	0.816 (+/- 0.007)	0.818 (+/- 0.002)
LR_Optimized_RFECV	0.823 (+/- 0.273)	0.010 (+/- 0.001)	0.528 (+/- 0.017)	0.534 (+/- 0.004)	0.564 (+/- 0.018)	0.570 (+/- 0.005)	0.776 (+/- 0.009)	0.779 (+/- 0.003)
GBC_RFECV	1.599 (+/- 0.283)	0.011 (+/- 0.001)	0.483 (+/- 0.020)	0.515 (+/- 0.011)	0.379 (+/- 0.021)	0.405 (+/- 0.012)	0.820 (+/- 0.006)	0.830 (+/- 0.002)
SVC_RFECV	2.172 (+/- 0.151)	0.559 (+/- 0.012)	0.491 (+/- 0.028)	0.511 (+/- 0.014)	0.390 (+/- 0.031)	0.405 (+/- 0.017)	0.821 (+/- 0.006)	0.828 (+/- 0.003)
RFC_RFECV	1.313 (+/- 0.153)	0.046 (+/- 0.002)	0.456 (+/- 0.032)	0.988 (+/- 0.008)	0.368 (+/- 0.027)	0.984 (+/- 0.014)	0.805 (+/- 0.012)	0.995 (+/- 0.003)

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

After performing hyperparameter optimization, we can see that the F1 scores of SVC and Random Forest Classifier improved while the score of GradientBoostingClassifier slightly reduced. This could be either due to the minimal number of iterations or because the default hyperparameters were working well. It can also be noticed that as the training scores of the Random Forest Classifier slightly reduced, it is slightly less overfitting now. Similarly, since SVC was underfitting earlier, as the training and test scores of SVC have improved, it seems that SVC is fitting better now with the data.

In [44]:

```
# SVC
distributions = {
    "svc__class_weight": [None, "balanced"],
    "svc__gamma": 10.0 ** np.arange(-3, 5),
    "svc__C": 10.0 ** np.arange(-3, 5),
}

# Hyperparameter Optimization
svc_random_search = RandomizedSearchCV(
    models["SVC"],
    param_distributions=distributions,
    cv=10,
    n_jobs=-1,
    n_iter=10,
    random_state=573,
    scoring=scoring_metrics,
    refit="f1",
)

svc_random_search.fit(X_train, y_train)
results["SCV_Optimized"] = mean_std_cross_val_scores(
    svc_random_search.best_estimator_,
    X_train,
    y_train,
    scoring=scoring_metrics,
    return_train_score=True,
)
```

In [45]:

```
print(f"Best F1 Score: {svc_random_search.best_score_}")
print(f"Best Params: {svc_random_search.best_params_}")
```

```
Best F1 Score: 0.5347241732968526
Best Params: {'svc__gamma': 0.001, 'svc__class_weight': 'balanced', 'svc__C': 100.0}
```

In [46]:

```
# Random Forest
distributions = {
        "randomforestclassifier__class_weight": [None, "balanced"],
        "randomforestclassifier__max_depth": np.arange(5, 25, 2),
        "randomforestclassifier__max_features": [
            None,
            "sqrt",
            "log2",
            0.2,
            0.4,
            0.6,
            0.8,
```

```
            0.9,
        ],
    }

    # Hyperparameter Optimization
forest_random_search = RandomizedSearchCV(
    models["RFC"],
    param_distributions=distributions,
    cv=20,
    n_jobs=-1,
    random_state=573,
    verbose=0,
    n_iter=10,
    scoring="f1",
)

forest_random_search.fit(X_train, y_train)

results["RFC_Optimized"] = mean_std_cross_val_scores(
    forest_random_search.best_estimator_,
    X_train,
    y_train,
    scoring=scoring_metrics,
    return_train_score=True,
)
```

In [47]:
```
print(f"Best F1 Score: {forest_random_search.best_score_}")
print(f"Best Params: {forest_random_search.best_params_}")
```

```
Best F1 Score: 0.5225167079662476
Best Params: {'randomforestclassifier__max_features': 0.8, 'randomforestclassifier__max_depth': 11, 'randomforestclassifier__class_weight': 'balanced'}
```

In [48]:
```
# GradientBoostingClassifier
distributions = {
        "gradientboostingclassifier__max_depth": np.arange(1, 25, 2),
    }

# Hyperparameter Optimization
gradientboosting_search = RandomizedSearchCV(
    models["GBC"],
    param_distributions=distributions,
    cv=10,
    n_jobs=-1,
    random_state=573,
    verbose=0,
    n_iter=10,
    scoring="f1",
)

gradientboosting_search.fit(X_train, y_train)

results["GBC_Optimized"] = mean_std_cross_val_scores(
    gradientboosting_search.best_estimator_,
    X_train,
    y_train,
    scoring=scoring_metrics,
    return_train_score=True,
)
```

In [49]:
```
print(f"Best F1 Score: {gradientboosting_search.best_score_}")
```

```python
print(f"Best Params: {gradientboosting_search.best_params_}")
```

```
Best F1 Score: 0.47152487805822474
Best Params: {'gradientboostingclassifier__max_depth': 5}
```

In [50]:
```python
pd.DataFrame(results).T
```

Out[50]:

	fit_time	score_time	test_f1	train_f1	test_recall	train_recall	test_accuracy	train_accuracy	test_p
Dummy	0.018 (+/- 0.001)	0.010 (+/- 0.001)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.000 (+/- 0.000)	0.777 (+/- 0.000)	0.777 (+/- 0.000)	0
LR	0.997 (+/- 0.095)	0.011 (+/- 0.001)	0.445 (+/- 0.022)	0.448 (+/- 0.014)	0.330 (+/- 0.019)	0.332 (+/- 0.014)	0.817 (+/- 0.006)	0.818 (+/- 0.002)	0
LR_Optimized	0.679 (+/- 0.013)	0.010 (+/- 0.000)	0.531 (+/- 0.018)	0.534 (+/- 0.003)	0.567 (+/- 0.019)	0.569 (+/- 0.005)	0.777 (+/- 0.009)	0.779 (+/- 0.002)	0
GBC	2.368 (+/- 0.021)	0.013 (+/- 0.001)	0.482 (+/- 0.023)	0.520 (+/- 0.005)	0.378 (+/- 0.024)	0.408 (+/- 0.005)	0.819 (+/- 0.005)	0.832 (+/- 0.001)	0
SVC	2.189 (+/- 0.041)	0.614 (+/- 0.024)	0.479 (+/- 0.022)	0.505 (+/- 0.006)	0.373 (+/- 0.023)	0.395 (+/- 0.009)	0.820 (+/- 0.005)	0.828 (+/- 0.001)	0
RFC	1.437 (+/- 0.029)	0.046 (+/- 0.001)	0.469 (+/- 0.026)	0.997 (+/- 0.000)	0.370 (+/- 0.025)	0.996 (+/- 0.001)	0.814 (+/- 0.007)	0.999 (+/- 0.000)	0
SCV_Optimized	3.960 (+/- 0.316)	0.887 (+/- 0.044)	0.528 (+/- 0.013)	0.544 (+/- 0.004)	0.547 (+/- 0.029)	0.564 (+/- 0.021)	0.783 (+/- 0.007)	0.790 (+/- 0.008)	0
RFC_Optimized	3.793 (+/- 0.048)	0.035 (+/- 0.001)	0.516 (+/- 0.010)	0.776 (+/- 0.006)	0.473 (+/- 0.011)	0.781 (+/- 0.013)	0.802 (+/- 0.004)	0.900 (+/- 0.002)	0
GBC_Optimized	3.866 (+/- 0.085)	0.013 (+/- 0.001)	0.466 (+/- 0.025)	0.609 (+/- 0.006)	0.366 (+/- 0.026)	0.481 (+/- 0.008)	0.814 (+/- 0.006)	0.863 (+/- 0.001)	0

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.

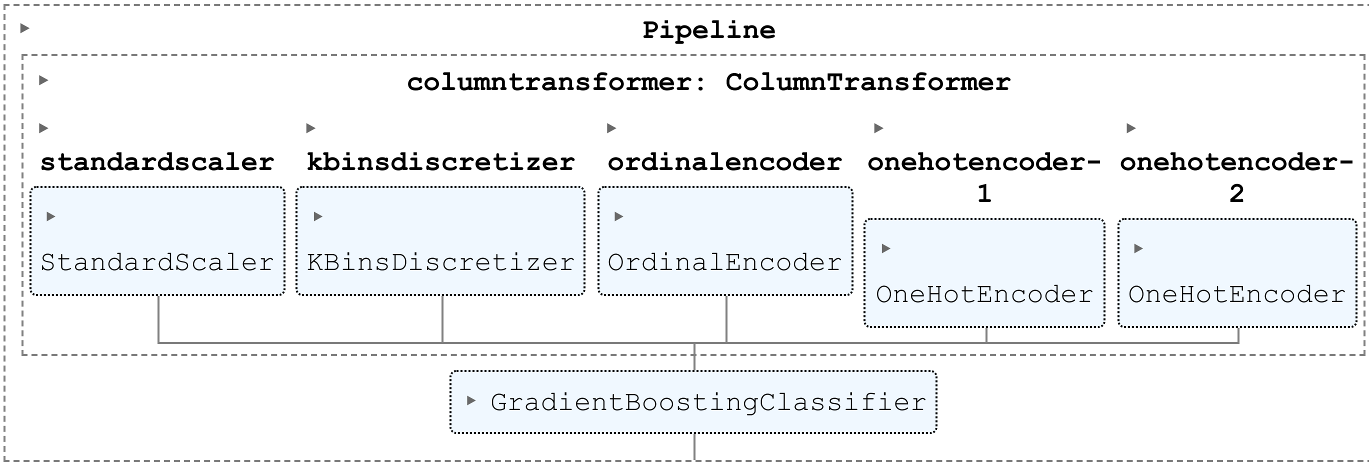
Points: 8

For this, we examine the most important features of the SVC model as per `eli5`. From the weight distribution of the features, we can see that feature `PAY_1` is the most important. As `PAY_1` denotes the repayment status in September 2005, and we're trying to predict if the client would default or not in

October 2005, it is logical that a person with higher re-payment delays in September is more likely to default in October as well, than a person with lower re-payment delay.

In [51]:
```
pipe_gbc = make_pipeline(column_transformer, GradientBoostingClassifier(random_state=573))
pipe_gbc.fit(X_train, y_train)
```

Out[51]:



In [52]:
```
import eli5

eli5.explain_weights(
    pipe_gbc.named_steps["gradientboostingclassifier"], feature_names=column_transformer.get_fea
)
```

Out[52]:

Weight	Feature
0.5795 ± 0.5122	PAY_1
0.0855 ± 0.1527	PAY_2
0.0283 ± 0.1222	PAY_3
0.0272 ± 0.1473	PAY_5
0.0264 ± 0.1662	PAY_AMT4
0.0249 ± 0.2106	PAY_AMT6
0.0241 ± 0.1340	PAY_4
0.0226 ± 0.2544	BILL_AMT1
0.0223 ± 0.1236	PAY_6
0.0221 ± 0.1954	PAY_AMT3
0.0207 ± 0.1624	LIMIT_BAL
0.0177 ± 0.2309	BILL_AMT4
0.0175 ± 0.2042	PAY_AMT1
0.0156 ± 0.2330	BILL_AMT2
0.0123 ± 0.2306	PAY_AMT2
0.0100 ± 0.1648	BILL_AMT3
0.0087 ± 0.1502	PAY_AMT5
0.0076 ± 0.0951	EDUCATION
0.0070 ± 0.1158	BILL_AMT6
0.0057 ± 0.1252	BILL_AMT5
… 9 more …	

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

Answer 1. On the test data, we're getting a F1 score of 0.525, recall of 0.558, accuracy of 0.777, and precision of 0.495.

Answer 2.

The cross-validation score for the optimized logistic regression model is 0.531 while the actual test score is 0.525. The test score seems to agree with the validation scores. Hence, the model seems to generalize well with unseen data as well. We can trust the test scores because the size of the train and the test set is quite large. Although there could be a bias introduced due to the imbalance in the target class, as we've set class weights as one of the hyperparameters, the bias introduced due to the imbalance should be minimal as almost all models choose `class_weight` as `balanced` for the best F1 scores. In addition, we're performing RandomizedSearchCV just 10 times (iterations), so it is less likely that we got lucky on the validation dataset. Hence, it is unlikely that we are experiencing optimization bias.

In [53]:
```
# Best model we got is Optimized LR
from sklearn.metrics import f1_score, get_scorer

test_scores = {}

for scorer in scoring_metrics:
    test_scores[scorer] = get_scorer(scorer)(
        lr_grid_search.best_estimator_, X_test, y_test
    )

test_scores = pd.DataFrame(
    test_scores,
    index=[0]
)
test_scores
```

Out[53]:

	f1	recall	accuracy	precision
0	0.525142	0.558386	0.777556	0.495635

In [54]:
```
# VALIDATION SCORES
pd.DataFrame(results["LR_Optimized"]).T
```

Out[54]:

	fit_time	score_time	test_f1	train_f1	test_recall	train_recall	test_accuracy	train_accuracy	test_precision	trai
0	0.679 (+/- 0.013)	0.010 (+/- 0.000)	0.531 (+/- 0.018)	0.534 (+/- 0.003)	0.567 (+/- 0.019)	0.569 (+/- 0.005)	0.777 (+/- 0.009)	0.779 (+/- 0.002)	0.499 (+/- 0.018)	

In [55]:
```
# Transformed Train Data
X_train_enc = pd.DataFrame(
    data=column_transformer.transform(X_train),
    columns=column_transformer.get_feature_names_out(),
    index=X_train.index,
)
X_train_enc.head()
```

Out[55]:

	LIMIT_BAL	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_
11997	-0.595569	0.857150	2.099797	-0.393042	-0.344872	-0.312016	-0.315299	-0.003847	-0.490496	-0.4
2943	-1.137066	-0.477282	-0.407798	2.146125	-0.344872	-0.312016	-0.315299	-0.428251	-0.397239	-0.3
9784	-0.672926	-0.477282	-0.407798	-0.393042	-0.344872	-0.312016	-0.315299	-0.257637	-0.254567	-0.1
27216	-0.286141	2.191581	2.099797	2.146125	2.242477	2.420973	2.429415	0.249834	0.303415	0.3
29783	-0.595569	0.857150	2.099797	-0.393042	-0.344872	-0.312016	2.429415	-0.072703	-0.062084	-0.0

5 rows × 29 columns

In [56]:
```
# Transformed Test Data
X_test_enc = pd.DataFrame(
    data=column_transformer.transform(X_test),
    columns=column_transformer.get_feature_names_out(),
    index=X_test.index,
)
X_test_enc.head()
```

Out[56]:

	LIMIT_BAL	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_
15298	1.647779	-0.477282	-0.407798	2.146125	-0.344872	-0.312016	-0.315299	0.225645	0.307173	0.3
13203	0.410070	-0.477282	-0.407798	-0.393042	-0.344872	-0.312016	-0.315299	0.018514	0.063261	0.1
1736	-0.672926	2.191581	-0.407798	-0.393042	-0.344872	-0.312016	-0.315299	0.369088	0.381061	0.3
9617	-0.208785	-0.477282	-0.407798	-0.393042	-0.344872	-0.312016	-0.315299	-0.690893	-0.685347	-0.6
5574	-1.137066	-0.477282	-0.407798	-0.393042	-0.344872	-0.312016	-0.315299	-0.430698	-0.431326	-0.3

5 rows × 29 columns

In [57]:
```
import shap

# Create a shap explainer object
logreg_explainer = shap.LinearExplainer(lr_grid_search.best_estimator_.named_steps["logisticregre

test_logreg_shap_values = logreg_explainer.shap_values(X_test_enc)
```

```
c:\Users\rkris\miniconda3\envs\573\lib\site-packages\tqdm\auto.py:22: TqdmWarning: IProgress not
found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/use
r_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [58]:
```
shap.initjs()
```

In [59]:
```
zero_ind = y_test[y_test == 0].index.tolist()
one_ind = y_test[y_test == 1].index.tolist()

ex_zero_index = zero_ind[10]
ex_one_index = one_ind[10]
```

In [60]:
```
# Example Obervation where the client does not default
X_test_enc.iloc[ex_zero_index]
```

Out[60]:

```
LIMIT_BAL          -0.904996
PAY_1              -0.477282
PAY_2              -0.407798
PAY_3              -0.393042
PAY_4              -0.344872
PAY_5              -0.312016
PAY_6              -0.315299
BILL_AMT1          -0.023775
BILL_AMT2          -0.026409
BILL_AMT3          -0.031780
BILL_AMT4           0.002958
BILL_AMT5          -0.327640
BILL_AMT6          -0.310765
PAY_AMT1           -0.229489
PAY_AMT2           -0.069944
PAY_AMT3           -0.211613
PAY_AMT4           -0.250339
PAY_AMT5           -0.270522
PAY_AMT6           -0.253047
AGE_0.0             0.000000
AGE_1.0             0.000000
AGE_2.0             0.000000
AGE_3.0             1.000000
AGE_4.0             0.000000
EDUCATION           0.000000
MARRIAGE_Married    0.000000
MARRIAGE_Others     0.000000
MARRIAGE_Single     1.000000
SEX_Male            0.000000
Name: 22417, dtype: float64
```

In [61]:

```
# Prediction on the test data example, 0 => The user will not default.
lr_grid_search.best_estimator_.named_steps["logisticregression"].predict(X_test_enc)[ex_zero_inde
```

```
X has feature names, but LogisticRegression was fitted without feature names
```

Out[61]:

```
0
```

In [62]:

```
# Probabilities of not defaulting and defaulting for this observation.
lr_grid_search.best_estimator_.named_steps["logisticregression"].predict_proba(X_test_enc)[ex_zer
```

```
X has feature names, but LogisticRegression was fitted without feature names
```

Out[62]:

```
array([0.64499631, 0.35500369])
```

In [63]:

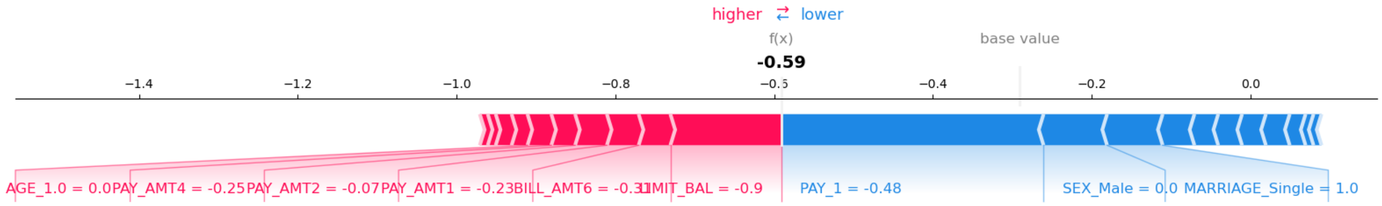
```
# SHAP Values for this observation
pd.DataFrame(
    test_logreg_shap_values[ex_zero_index, :],
    index=column_transformer.get_feature_names_out(),
    columns=["SHAP values"],
)
```

Out[63]:

	SHAP values
LIMIT_BAL	0.136474
PAY_1	-0.333240
PAY_2	-0.025807
PAY_3	-0.038115
PAY_4	-0.023462
PAY_5	-0.014257
PAY_6	-0.026888
BILL_AMT1	0.006495
BILL_AMT2	0.002493
BILL_AMT3	-0.004815
BILL_AMT4	-0.001228
BILL_AMT5	-0.025373
BILL_AMT6	0.041233
PAY_AMT1	0.041832
PAY_AMT2	0.037798
PAY_AMT3	0.002590
PAY_AMT4	0.030793
PAY_AMT5	0.005312
PAY_AMT6	-0.000741
AGE_0.0	-0.003726
AGE_1.0	0.031021
AGE_2.0	0.015118
AGE_3.0	0.000018
AGE_4.0	-0.008254
EDUCATION	0.022798
MARRIAGE_Married	-0.026463
MARRIAGE_Others	-0.000000
MARRIAGE_Single	-0.072525
SEX_Male	-0.075460

In [64]:

```
# Force Plot
shap.force_plot(
    logreg_explainer.expected_value,
    np.around(test_logreg_shap_values[ex_zero_index, :], decimals=2), # SHAP values associated w
    np.around(X_test_enc.iloc[ex_zero_index, :], decimals=2), # Feature vector of the example
    matplotlib=True,
)
```



Answer 3.

For this, we take an observation where the client is not defaulting and compute the `SHAP` values and plot the `SHAP` force_plot. Here, the base value represents the expected value of target class 1 in our data. If the raw score of an observation is above the base value, the associated target class would be 1. On the other hand, if the raw score of a model is less than the base value, the associated target class would be 0.

For this observation, the raw score is lower than the base value and hence, the target class is 0, i.e. not defaulting. The important features used in making the prediction are highlighted in red and blue - red represents features that pushed the model score higher, whereas blue represents features that pushed the model score lower. In this case, `LIMIT_BAL` , `BILL_AMT6` , and `PAY_AMT1` seemed to have been the reason to push the score towards the higher side, whereas `PAY_1` , `SEX_Male` , `MARRIAGE_Single` played an important role in pushing the score towards the lower side. Out of these features, `LIMIT_BAL` and `PAY_1` have more of an impact on the score since they are closer to the decision boundary. Out of these two, the magnitude of the impact seems to be greater for `PAY_1` because of the greater size of the bar.

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

In [65]:
```
# Summary table containing the training and test scores of the best model: LR_Optimized
train_scores = {}

for scorer in scoring_metrics:
    train_scores[scorer] = get_scorer(scorer)(
        lr_grid_search.best_estimator_, X_train, y_train
    )

train_scores = pd.DataFrame(
    train_scores,
```

```
    index=[0],
)

summary_df = pd.concat([train_scores.T, test_scores.T], axis = 1)
summary_df.columns = ['Train Scores', 'Test Scores']
summary_df
```

Out[65]:

	Train Scores	Test Scores
f1	0.534225	0.525142
recall	0.568326	0.558386
accuracy	0.779417	0.777556
precision	0.503984	0.495635

In [66]:

```
# Summary Table that highlights the most important features as per Logisticregression
coeffs = lr_grid_search.best_estimator_.named_steps["logisticregression"].coef_
data = {}
data['Coefficients'] = coeffs[0]
data['Features'] = column_transformer.get_feature_names_out()

df = pd.DataFrame(data, columns=["Features", "Coefficients"])
df.sort_values(by="Coefficients", ascending=False).style.background_gradient(cmap='Blues')
```

Out[66]:

	Features	Coefficients
1	PAY_1	0.657169
9	BILL_AMT3	0.302552
28	SEX_Male	0.184049
6	PAY_6	0.130618
3	PAY_3	0.111192
5	PAY_5	0.104336
4	PAY_4	0.100756
11	BILL_AMT5	0.071991
2	PAY_2	0.064323
25	MARRIAGE_Married	0.060144
23	AGE_4.0	0.041269
19	AGE_0.0	0.023290
18	PAY_AMT6	0.003096
22	AGE_3.0	0.000023
15	PAY_AMT3	-0.012955
17	PAY_AMT5	-0.017490
24	EDUCATION	-0.018093
8	BILL_AMT2	-0.031496
26	MARRIAGE_Others	-0.045690
7	BILL_AMT1	-0.078363
16	PAY_AMT4	-0.088960
10	BILL_AMT4	-0.094147
20	AGE_1.0	-0.106970
21	AGE_2.0	-0.107988
12	BILL_AMT6	-0.120460
0	LIMIT_BAL	-0.133754
27	MARRIAGE_Single	-0.164829
13	PAY_AMT1	-0.172300
14	PAY_AMT2	-0.199640

Answer 1: Summary

- We have an imbalanced class which would likely need a model with balanced weights. From hyperparameter tuning, most models did choose `class_weight` as `balanced` for the best F1 scores.
- Test scores of all models are greater than the baseline (dummy) model.
- Using hyperparameter tuning with the linear model (logistic regression) we were able to get a cross validation score of 0.53 with the best hyperparameter C as 469.540.
- Training non-linear models did not improve the score beyond the linear model.

- As our final model is linear, we have higher model interpretability and better efficiency in terms of model fit times when compared to non-linear models while also having a better F1 score.
- The test score we got (0.525) was similar to the validation scores (0.53), and we have good reason to trust the test score because of the large sample size of test and train data.
- On interpreting the important features of the model, we find that the column `PAY_1` has the greatest effect on whether or not the client will default in his/her next payment.

Answer 2

- A model to predict whether or not a client will default their next payment has been trained which has a validation F1 score of 0.53 and a test score of 0.525. Among the other models that we tried, the model with the best validation and test scores turned out to be the balanced optimized logistic regression model. We have a validation precision of 0.499 and a validation recall score of 0.567. Even though this was the best model that we have trained so far, this is nowhere near ready to be deployed in production. The data used for training was collected in 2005 and is not a good representative of the current trends. Additionally, with the current scores, this model would only be able to identify 56.7% of the clients who will default - any bank that uses this model is bound to miss identifying around 43% of the defaulters. In addition to that, because the precision is as low as 0.499, the model is correct only ~50% of the time in predicting whether the client would default or not. We are falsely assuming 50% of our clients will not be able to make the credit card payment and hence losing business from potential clients. Hence, there is a huge scope for improvement in this model. A few suggestions are addressed in the answer to the next question.

Answer 3

- The data we're looking at is really old as it is taken from 2005. Using this model on current production data is not ideal as spending patterns have changed. Also, the data had a lot of undocumented values which throws doubt on its integrity. Having data that is representative of the production data is crucial for this model to work well in production.
- Better feature engineering. From the data, we could see that how high or low the bill is from the limit could help us in the prediction process. Adding a feature for this could improve scores.
- Look closely into the features with high correlation and see how the model is performing by taking a subset of them. This could reduce the dimensionality and make the model more interpretable.
- Increase iterations for cross-validations in hyperparameter tuning to find a more optimized model
- Try different optimizers (RMSProp, Adam) rather than the default in sklearn.
- Try complex classifiers like Neural Networks

Answer 4

- Test score : 0.525
- Primary Metric - F1 score

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 3 key takeaways from this course are:

- Garbage in, Garbage out: At the end of the data, the quality of the data plays a crucial role in making sure the model performs well.
- Model interpretability is crucial to understand the reasons behind a certain prediction, not just to improve the model, but also to ensure that the reasons are ethical.
- In classification problems, it is crucial to look at metrics other than accuracy especially when there is a class imbalance.

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: