Question 1: Load

- Programmatically download and load into your favorite analytical tool the transactions data. This data, which is in line-delimited JSON format, can be found here
- Please describe the structure of the data. Number of records and fields in each record?
- Please provide some additional basic summary statistics for each field. Be sure to include a count of null, minimum, maximum, and unique values where appropriate.

```
!unzip transactions.zip
```

```
Archive:  transactions.zip
  inflating: transactions.txt
```

```python
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm

data = []

with open("transactions.txt", "r") as f:
    for line in f:
        if line.strip():  # skip empty lines
            data.append(json.loads(line))

data[0]
```


```
{'accountNumber': '737265056',
 'customerId': '737265056',
 'creditLimit': 5000.0,
 'availableMoney': 5000.0,
 'transactionDateTime': '2016-08-13T14:27:32',
 'transactionAmount': 98.55,
 'merchantName': 'Uber',
 'acqCountry': 'US',
 'merchantCountryCode': 'US',
 'posEntryMode': '02',
 'posConditionCode': '01',
 'merchantCategoryCode': 'rideshare',
 'currentExpDate': '06/2023',
 'accountOpenDate': '2015-03-14',
 'dateOfLastAddressChange': '2015-03-14',
 'cardCVV': '414',
 'enteredCVV': '414',
 'cardLast4Digits': '1803',
 'transactionType': 'PURCHASE',
```

```
 'echoBuffer': '',
 'currentBalance': 0.0,
 'merchantCity': '',
 'merchantState': '',
 'merchantZip': '',
 'cardPresent': False,
 'posOnPremises': '',
 'recurringAuthInd': '',
 'expirationDateKeyInMatch': False,
 'isFraud': False}

df = pd.DataFrame(data)

df.head()
```

{"type":"dataframe","variable_name":"df"}

Some of these columns dont seem to have values in them

```
df["transactionDateTime"] = pd.to_datetime(df["transactionDateTime"])
df["accountOpenDate"] = pd.to_datetime(df["accountOpenDate"])
df["dateOfLastAddressChange"] =
pd.to_datetime(df["dateOfLastAddressChange"])
df["currentExpPeriod"] = pd.to_datetime(df["currentExpDate"],
format="%m/%Y")
numerical_columns = df.select_dtypes(include=[np.number,
"datetime64[ns]"]).columns
categorical_columns = df.select_dtypes(exclude=[np.number,
"datetime64[ns]"]).columns

df.describe()
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 8,\n \"fields\": [\n   {\n   \"column\": \"creditLimit\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 273388.01597613463,\n   \"min\": 250.0,\n   \"max\": 786363.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   10759.464458526152,\n   15000.0,\n   786363.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"availableMoney\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 274673.47898207273,\n   \"min\": -1005.63,\n   \"max\": 786363.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   6250.725369288736,\n   7500.0,\n   786363.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"transactionDateTime\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"1970-01-01 00:00:00.000786363\",\n   \"max\": \"2016-12-30 23:59:45\",\n   \"num_unique_values\": 7,\n   \"samples\": [\n

\"786363\",\n   \"2016-07-06 01:58:58.395681536\",\n
\"2016-10-05 13:52:03.500000\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"transactionAmount\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 277890.328464602,\n
\"min\": 0.0,\n   \"max\": 786363.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
136.98579095150708,\n   191.48,\n   786363.0\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"accountOpenDate\",\n   \"properties\": {\n   \"dtype\":
\"date\",\n   \"min\": \"1970-01-01 00:00:00.000786363\",\n
\"max\": \"2015-12-31 00:00:00\",\n   \"num_unique_values\": 7,\n
\"samples\": [\n   \"786363\",\n   \"2014-02-03
01:11:17.352825856\",\n   \"2015-05-04 00:00:00\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"dateOfLastAddressChange\",\n
\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":
\"1970-01-01 00:00:00.000786363\",\n   \"max\": \"2016-12-30
00:00:00\",\n   \"num_unique_values\": 7,\n   \"samples\":
[\n   \"786363\",\n   \"2015-04-14
06:46:41.127723520\",\n   \"2016-06-06 00:00:00\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"currentBalance\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
275087.6938219826,\n   \"min\": 0.0,\n   \"max\": 786363.0,\
n   \"num_unique_values\": 8,\n   \"samples\": [\n
4508.739089237413,\n   5291.095,\n   786363.0\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"currentExpPeriod\",\n   \"properties\": {\n   \"dtype\":
\"date\",\n   \"min\": \"1970-01-01 00:00:00.000786363\",\n
\"max\": \"2033-08-01 00:00:00\",\n   \"num_unique_values\": 7,\n
\"samples\": [\n   \"786363\",\n   \"2026-09-25
23:50:42.724542720\",\n   \"2030-03-01 00:00:00\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 786363 entries, 0 to 786362
Data columns (total 30 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   accountNumber             786363 non-null  object
 1   customerId                786363 non-null  object
 2   creditLimit               786363 non-null  float64
 3   availableMoney            786363 non-null  float64
 4   transactionDateTime       786363 non-null  datetime64[ns]
```

```
 5   transactionAmount         786363 non-null  float64
 6   merchantName              786363 non-null  object
 7   acqCountry                786363 non-null  object
 8   merchantCountryCode       786363 non-null  object
 9   posEntryMode              786363 non-null  object
 10  posConditionCode          786363 non-null  object
 11  merchantCategoryCode      786363 non-null  object
 12  currentExpDate            786363 non-null  object
 13  accountOpenDate           786363 non-null  datetime64[ns]
 14  dateOfLastAddressChange   786363 non-null  datetime64[ns]
 15  cardCVV                   786363 non-null  object
 16  enteredCVV                786363 non-null  object
 17  cardLast4Digits           786363 non-null  object
 18  transactionType           786363 non-null  object
 19  echoBuffer                786363 non-null  object
 20  currentBalance            786363 non-null  float64
 21  merchantCity              786363 non-null  object
 22  merchantState             786363 non-null  object
 23  merchantZip               786363 non-null  object
 24  cardPresent               786363 non-null  bool
 25  posOnPremises             786363 non-null  object
 26  recurringAuthInd          786363 non-null  object
 27  expirationDateKeyInMatch  786363 non-null  bool
 28  isFraud                   786363 non-null  bool
 29  currentExpPeriod          786363 non-null  datetime64[ns]
dtypes: bool(3), datetime64[ns](4), float64(4), object(19)
memory usage: 164.2+ MB
```

```
len(df), len(df.columns)
```

```
(786363, 30)
```

```
def summarize(df):
    numeric_stats = {}
    categorical_stats = {}

    for col in numerical_columns:
        s = df[col]
        nulls = s.isnull().sum()
        uniques = s.nunique()
        numeric_stats[col] = {
            "Null Count": nulls,
            "Unique Values": uniques,
            "Min": s.min(),
            "Max": s.max(),
            "Mean": s.mean(),
            "Median": s.median(),
        }
    for col in categorical_columns:
        s = df[col]
```

```
        nulls = s.isnull().sum()
        uniques = s.nunique()
        empty_strings = (s == "").sum()
        categorical_stats[col] = {
            "Null Count": nulls,
            "Unique Values": uniques,
            "Most Frequent": s.mode().iloc[0] if not s.mode().empty
else "N/A",
            "Frequency Count" : s.value_counts().iloc[0] if not
s.value_counts().empty else "N/A",
            "Empty String Count": empty_strings,
        }

    numeric_df = pd.DataFrame(numeric_stats).T
    categorical_df = pd.DataFrame(categorical_stats).T

    return numeric_df, categorical_df

num_cols_stats, cat_cols_stats = summarize(df)
```

Answer 1:

The data is provided as a .txt file containing line-delimited JSON. Each line represents a credit card transaction and is structured as a JSON object. There are 786,363 records and 29 fields in each record.

Some statistics for each of the fields are below, split by whether numerical or categorical. It is worth noting that while there are no null values in any of the columns, there are some columns in the categorical data which have empty strings, with some of them being entirely empty. It might be beneficial to drop them entirely during later analysis and feature engineering.

```
num_cols_stats
```

	Null Count	Unique Values	Min
creditLimit	0.0	10.0	250.0
availableMoney	0.0	521916.0	-1005.63
transactionDateTime	0	776637	2016-01-01 00:01:02
transactionAmount	0.0	66038.0	0.0
accountOpenDate	0	1820	1989-08-22 00:00:00
dateOfLastAddressChange	0	2184	1989-08-22 00:00:00
currentBalance	0.0	487318.0	0.0

```
currentExpPeriod                0           165  2019-12-01 00:00:00

                                          Max
Mean  \
creditLimit                           50000.0
10759.464459
availableMoney                        50000.0
6250.725369
transactionDateTime       2016-12-30 23:59:45  2016-07-06
01:58:58.395681536
transactionAmount                     2011.54
136.985791
accountOpenDate           2015-12-31 00:00:00  2014-02-03
01:11:17.352825856
dateOfLastAddressChange  2016-12-30 00:00:00  2015-04-14
06:46:41.127723520
currentBalance                       47498.81
4508.739089
currentExpPeriod          2033-08-01 00:00:00  2026-09-25
23:50:42.724542720

                                       Median
creditLimit                            7500.0
availableMoney                        3184.86
transactionDateTime       2016-07-08 05:03:57
transactionAmount                        87.9
accountOpenDate           2014-09-05 00:00:00
dateOfLastAddressChange  2016-01-13 00:00:00
currentBalance                        2451.76
currentExpPeriod          2026-10-01 00:00:00

cat_cols_stats

                         Null Count Unique Values  Most Frequent  \
accountNumber                     0          5000      380680241
customerId                        0          5000      380680241
merchantName                      0          2490           Uber
acqCountry                        0             5             US
merchantCountryCode               0             5             US
posEntryMode                      0             6             05
posConditionCode                  0             4             01
merchantCategoryCode              0            19  online_retail
currentExpDate                    0           165        03/2029
cardCVV                           0           899            869
enteredCVV                        0           976            869
cardLast4Digits                   0          5246            593
transactionType                   0             4       PURCHASE
echoBuffer                        0             1
```

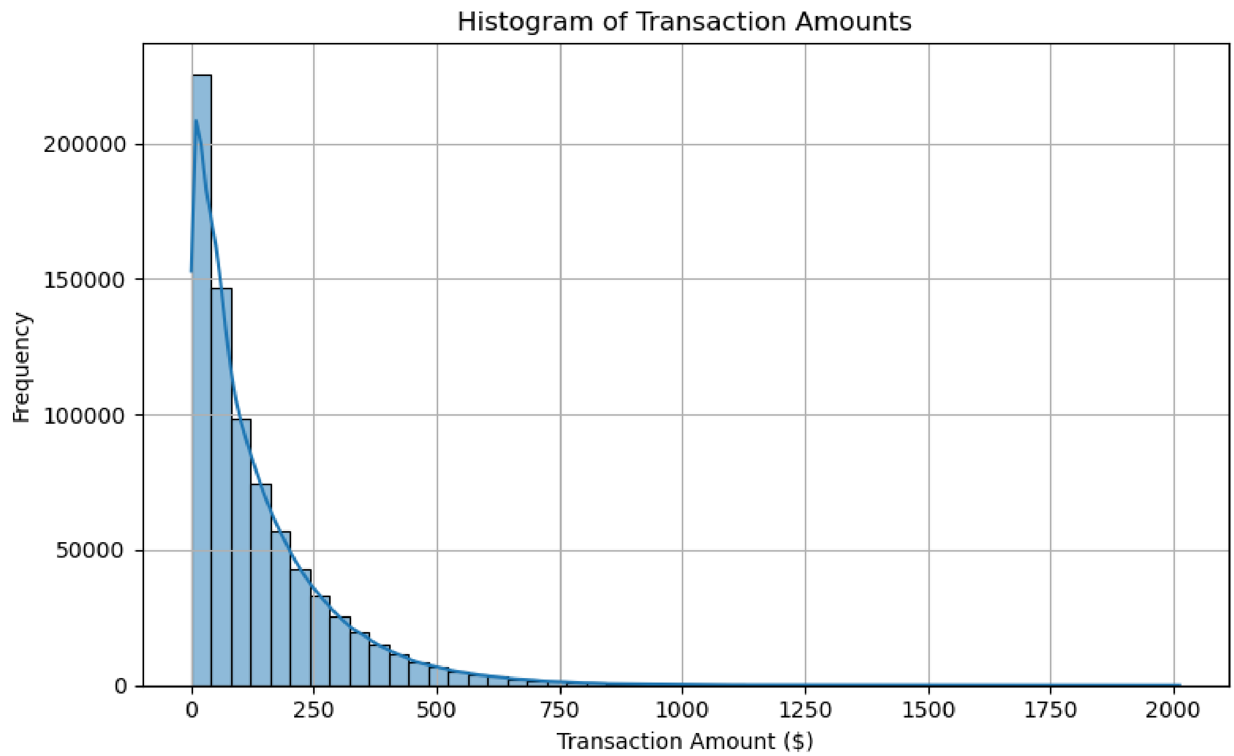
```
merchantCity                       0             1
merchantState                      0             1
merchantZip                        0             1
cardPresent                        0             2           False
posOnPremises                      0             1
recurringAuthInd                   0             1
expirationDateKeyInMatch           0             2           False
isFraud                            0             2           False

                         Frequency Count Empty String Count
accountNumber                      32850                  0
customerId                         32850                  0
merchantName                       25613                  0
acqCountry                        774709               4562
merchantCountryCode               778511                724
posEntryMode                      315035               4054
posConditionCode                  628787                409
merchantCategoryCode              202156                  0
currentExpDate                      5103                  0
cardCVV                            33749                  0
enteredCVV                         33424                  0
cardLast4Digits                    32946                  0
transactionType                   745193                698
echoBuffer                        786363             786363
merchantCity                      786363             786363
merchantState                     786363             786363
merchantZip                       786363             786363
cardPresent                       433495                  0
posOnPremises                     786363             786363
recurringAuthInd                  786363             786363
expirationDateKeyInMatch          785320                  0
isFraud                           773946                  0
```

Question 2: Plot

- Plot a histogram of the processed amounts of each transaction, the transactionAmount column.
- Report any structure you find and any hypotheses you have about that structure.

```
plt.figure(figsize=(8, 5))
sns.histplot(df["transactionAmount"], bins=50, kde=True)
plt.title("Histogram of Transaction Amounts")
plt.xlabel("Transaction Amount ($)")
plt.ylabel("Frequency")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/seaborn/
_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Answer 2

The histogram is right skewed, with most transactions clustering towards the left (median=$87.9).The longer tail extends towards higher amounts, with purchases of over $750 being very rare. The peak of the histogram is at very low amounts, indicating most transactions are probably less than $20. I suspect this is because consumers tend to spend smaller amounts more frequently rather than making large purchases regularly. These smaller transactions could be recurring daily/weekly purchases like groceries, transit, fast food, coffees as opposed to larger purchases which are often non recurring.

Question 3: Data Wrangling - Duplicate Transactions

You will notice a number of what look like duplicated transactions in the data set. One type of duplicated transaction is a reversed transaction, where a purchase is followed by a reversal.

Another example is a multi-swipe, where a vendor accidentally charges a customer's card multiple times within a short time span.

1. Can you programmatically identify reversed and multi-swipe transactions?
2. What total number of transactions and total dollar amount do you estimate for the reversed transactions? For the multi-swipe transactions? (please consider the first transaction to be "normal" and exclude it from the number of transaction and dollar amount counts)
3. Did you find anything interesting about either kind of transaction?

Identifying Reversals

```
df['transactionType'].unique()
```

```
array(['PURCHASE', 'ADDRESS_VERIFICATION', 'REVERSAL', ''],
dtype=object)
```

```
df['transactionType'].value_counts()
```

```
transactionType
PURCHASE                745193
REVERSAL                 20303
ADDRESS_VERIFICATION     20169
                           698
Name: count, dtype: int64
```

Assumption: Since most credit card transactions will be purchase and these empty transactionType values are a very small fraction of all transactions, I will replace these with the type "PURCHASE"

```
df['transactionType'] = df['transactionType'].replace('', 'PURCHASE')

df_copy = df.copy()
df_copy = df_copy.sort_values(by=["accountNumber",
"transactionDateTime"])

df_copy
```

{"type":"dataframe","variable_name":"df_copy"}

I am assuming that in a lot of the cases, the reversal will be immediately preceded by the corresponding purchase i.e same amount, same merchant, same customer but opposite transactionType

```
# Sort by these columns for convenience
df_sorted = df_copy.sort_values(by=["customerId", "merchantName",
"transactionDateTime"]).copy()
df_sorted = df_sorted.reset_index(drop=False)  # preserve original
```

```
index
df_sorted.rename(columns={"index": "original_index"}, inplace=True)

# Find back-to-back reversal-purchase pairs by using indices
reversal_indices = df_sorted[df_sorted["transactionType"]==
"REVERSAL"].index.values.astype(int)
purchase_indices = reversal_indices - 1

prev_index_df = pd.DataFrame(purchase_indices,
columns=["purchase_index"])
reversal_index_df = pd.DataFrame(reversal_indices,
columns=["reversal_index"])

purchase_amounts = pd.DataFrame(df_sorted.loc[purchase_indices,
"transactionAmount"].values, columns=["purchase_amount"])
reversal_amounts = pd.DataFrame(df_sorted.loc[reversal_indices,
"transactionAmount"].values, columns=["reversal_amount"])

# Combine and filter exact matches
quick_match_candidates = pd.concat([prev_index_df, purchase_amounts,
reversal_index_df, reversal_amounts], axis=1)
# quick_match_candidates =
quick_match_candidates[~(quick_match_candidates["reversal_amount"] ==
0)]
quick_match_candidates["amounts_match"] =
quick_match_candidates[["purchase_amount",
"reversal_amount"]].apply(lambda x: x[0] == x[1], axis=1)

# Capture matched indices from fast pass
fast_reversal_indices =
quick_match_candidates.loc[quick_match_candidates["amounts_match"],
"reversal_index"].values
fast_purchase_indices =
quick_match_candidates.loc[quick_match_candidates["amounts_match"],
"purchase_index"].values
fast_matched_indices = np.concatenate((fast_reversal_indices,
fast_purchase_indices))

# Extract rows from original DataFrame
fast_matches = df_sorted.loc[fast_matched_indices, ["original_index",
"customerId", "merchantName", "transactionAmount",
"transactionDateTime", "transactionType"]]
fast_matches = fast_matches.sort_values(by=["customerId",
"merchantName", "transactionDateTime"])

# Remove matched indices from original data
matched_indices = fast_matches["original_index"].unique()
df_unmatched = df_copy.drop(index=matched_indices)
```

```
<ipython-input-20-3d2f57c4706e>:19: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer
keys will always be treated as labels (consistent with DataFrame
behavior). To access a value by position, use `ser.iloc[pos]`
  quick_match_candidates["amounts_match"] =
quick_match_candidates[["purchase_amount",
"reversal_amount"]].apply(lambda x: x[0] == x[1], axis=1)
```

```
fast_matches
```


```
{"summary":"{\n  \"name\": \"fast_matches\",\n  \"rows\": 26032,\n
\"fields\": [\n    {\n      \"column\": \"original_index\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
227955,\n        \"min\": 38,\n        \"max\": 786301,\n
\"num_unique_values\": 26032,\n        \"samples\": [\n
474416,\n          687301,\n          660634\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"customerId\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2866,\n        \"samples\": [\n
\"795213218\",\n          \"954773577\",\n          \"441663479\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"merchantName\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1853,\n        \"samples\": [\n
\"Subway #228067\",\n          \"Shake Shack #174098\",\n
\"Domino's Pizza #643011\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"transactionAmount\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 151.85113470897363,\n
\"min\": 0.0,\n        \"max\": 1435.64,\n
\"num_unique_values\": 10364,\n        \"samples\": [\n
117.62,\n          130.01,\n          197.98\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"transactionDateTime\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2016-01-01 00:47:37\",\n        \"max\": \"2016-12-30 23:16:52\",\n
\"num_unique_values\": 26021,\n        \"samples\": [\n
\"2016-05-06 23:43:36\",\n          \"2016-08-04 01:21:11\",\n
\"2016-04-12 00:55:02\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"transactionType\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"PURCHASE\",\n          \"REVERSAL\",\n
\"ADDRESS_VERIFICATION\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"fast_matches"}
```

fast_matches dataframe has pairs of purchases and then reversals. Hence, if there are 25254 rows here, that means half of those are reversals. Hence, out of the 20303 reversal transactions, 12627 reversal transactions have been paired with a corresponding purchase right before it.

For the remaining reversals, it means that there have been purchases between the pair of purchase and reversal. Hence, I will look at groups of customerId, merchantName and transactionAmount, split by purchases and reversals. For each reversal, I will start look for a purchase group that has the same key values and a timestamp before the reversal. Once identified, that purchase group will be removed from consideration for future reversals.

```
reversals =
df_unmatched[df_unmatched["transactionType"]=="REVERSAL"].copy()
purchases = df_unmatched[df_unmatched["transactionType"]==
"PURCHASE"].copy()

group_keys = ["customerId", "merchantName", "transactionAmount"]
matched_rows = []

for keys, group_reversals in tqdm(reversals.groupby(group_keys),
total=reversals.groupby(group_keys).ngroups, desc="Matching
reversals"):
    group_purchases = purchases[
        (purchases["customerId"] == keys[0]) &
        (purchases["merchantName"] == keys[1]) &
        (purchases["transactionAmount"] == keys[2])
    ]

    used_purchases = set()
    group_reversals =
group_reversals.sort_values("transactionDateTime")

    for _, rev_row in group_reversals.iterrows():
        # Only consider purchases before this reversal
        prior_purchases = group_purchases[
            group_purchases["transactionDateTime"] <
rev_row["transactionDateTime"]
        ].sort_values("transactionDateTime", ascending=False)

        for _, pur_row in prior_purchases.iterrows():
            if pur_row.name not in used_purchases:
                matched_rows.append({
                    "customerId": keys[0],
                    "merchantName": keys[1],
                    "transactionAmount": keys[2],
                    "purchase_time": pur_row["transactionDateTime"],
                    "reversal_time": rev_row["transactionDateTime"],
                    "purchase_index": pur_row.name,
                    "reversal_index": rev_row.name,
                    "time_diff_sec": (rev_row["transactionDateTime"] -
pur_row["transactionDateTime"]).total_seconds()
```

```
                })
                used_purchases.add(pur_row.name)
                break
remaining_match_df = pd.DataFrame(matched_rows)
#remaining_match_df =
pd.read_parquet('dataframes/remaining_match_df.parquet')
```

```
Matching reversals: 100%|██████████| 7267/7267 [36:35<00:00,
3.31it/s]
```

```
#remaining_match_df.to_parquet('dataframes/
remaining_match_df.parquet', index=True)
```

I will get the indices for these identified purchases and reversals and create a df with the structure of the original df

```
fast_indices = fast_matches["original_index"].values
remaining_indices = pd.concat([
    remaining_match_df["purchase_index"],
    remaining_match_df["reversal_index"]
]).values

all_matched_indices = np.unique(np.concatenate([fast_indices,
remaining_indices]))

matched_transactions_df =
df_copy.loc[all_matched_indices].sort_values(by=["customerId",
"merchantName", "transactionDateTime"])

matched_transactions_df
```

{"type":"dataframe","variable_name":"matched_transactions_df"}

```
matched_transactions_df.to_parquet('dataframes/
all_matched_reversals.parquet', index=True)
```

Identifying Multiswipes

```
df_copy = df.copy()

#Filter purchases
purchase_df = df_copy[df_copy["transactionType"].str.upper() ==
"PURCHASE"].copy()
#Sort
purchase_df = purchase_df.sort_values(by=["customerId",
"merchantName", "transactionAmount", "transactionDateTime"])

purchase_df["is_duplicate"] = purchase_df.duplicated(
    subset=["customerId", "merchantName", "transactionAmount"],
```

```
keep=False
)

#Calculate the time difference between the transactions with the same customerId, merchantName, and transactionAmount
purchase_df["timeDiff"] = purchase_df.groupby(
    ["customerId", "merchantName", "transactionAmount"]
)["transactionDateTime"].diff().dt.total_seconds()

purchase_df['timeDiff']
```

```
541917          NaN
541962          NaN
541920          NaN
541904          NaN
541925          NaN
           ...
108113    2688551.0
108114    2705645.0
108115    2609864.0
108109          NaN
108107          NaN
Name: timeDiff, Length: 745891, dtype: float64
```

These NaN values indicate the first of the multiswipe transaction since no prior for time difference and hence have NaN.

```
for delta in [120,140,150,180,240,300]:
  print(len(purchase_df[
    (purchase_df["is_duplicate"]) &
    (purchase_df["timeDiff"] <= delta)
]))
```

```
4944
5765
6172
7457
7457
7457
```

Since 180 seconds i.e 3 minutes is a reasonable time to assume for classifying as multiswipe and there aren't more up until 5 minutes, after which it seems unreasonable to attempt to classify as multiswipe, I chose 3 minutes as the threshold. Since the first occurences of a multiswipe have timeDiff as NaN, the comparison with 180 will always return False, and I dont need to worry about not including the first occurences.

```
multi_swipe_candidates = purchase_df[
    (purchase_df["is_duplicate"]) &
    (purchase_df["timeDiff"] <= 180)
```

```
].copy()
multi_swipe_candidates = multi_swipe_candidates.sort_values(
    by=["customerId", "merchantName", "transactionAmount",
"transactionDateTime"]
)
multi_swipe_candidates
```

{"type":"dataframe","variable_name":"multi_swipe_candidates"}

I will also check which of the multiswipe purchases was later reversed.

```
multi_swipe_indices = set(multi_swipe_candidates.index)
matched_indices = set(matched_transactions_df.index)
overlap_indices = multi_swipe_indices.intersection(matched_indices)
multi_swipe_reversed_df =
matched_transactions_df.loc[list(overlap_indices)].copy()
multi_swipe_reversed_df =
multi_swipe_reversed_df.sort_values(by=["customerId", "merchantName",
"transactionDateTime"])
```

I will create df_flagged to keep flags for these purchase-reverse transaction and multiswipes

```
df_flagged = df.copy()
df_flagged["purchase_will_be_reversed"] = df_flagged.index.isin(

matched_transactions_df[matched_transactions_df["transactionType"]==
"PURCHASE"].index
)
df_flagged["reversal_matches_purchase"] = df_flagged.index.isin(

matched_transactions_df[matched_transactions_df["transactionType"]==
"REVERSAL"].index
)
df_flagged["is_multi_swipe"] =
df_flagged.index.isin(multi_swipe_candidates.index)
df_flagged["is_multi_swipe_purchase_reversed"] =
df_flagged.index.isin(multi_swipe_reversed_df.index)
```

Summary

```
total_matched_amount =
matched_transactions_df["transactionAmount"].sum() / 2
total_multi_swipe_amount =
multi_swipe_candidates["transactionAmount"].sum()
total_reversal_amount =
df_copy[df_copy.transactionType=='REVERSAL'].transactionAmount.sum()
total_multi_swipe_reversed_amount =
```

```
multi_swipe_reversed_df.transactionAmount.sum()
total_multi_swipe_not_reversed_amount = total_multi_swipe_amount -
total_multi_swipe_reversed_amount
total_unidentified_reversals_amount = total_reversal_amount -
total_matched_amount
total_fast_reversals_amount =
fast_matches[fast_matches.transactionType=="REVERSAL"].transactionAmou
nt.sum()

total_identified_reversals = (len(matched_transactions_df))//2
total_reversals = len(df_copy[df_copy['transactionType']=='REVERSAL'])
total_unidentified_reversals = total_reversals -
total_identified_reversals
total_fast_reversals =
len(fast_matches[fast_matches.transactionType=="REVERSAL"])
total_multi_swipes_reversed = len(multi_swipe_reversed_df)
print(f"Total Reversals: {total_reversals}")
print(f"Identified Reversals: {total_identified_reversals}")
print(f"Total Fast Reversals: {total_fast_reversals}")
print(f"Unidentified Reversals: {total_unidentified_reversals}")
print()
print(f"Total Reversals Amount: ${total_reversal_amount:,.2f}")
print(f"Total Identified Reversals Amount: $
{total_matched_amount:,.2f}")
print(f"Total Fast Reversals Amount: $
{total_fast_reversals_amount:,.2f}")
print(f"Total Unidentified Reversals Amount: $
{total_unidentified_reversals_amount:,.2f}")
print()
print(f"Total Multi-Swipes identified: {len(multi_swipe_candidates)}")
print(f"Total Multi-Swipes Reversed: {total_multi_swipes_reversed}")
print(f"Total Multi-Swipe Amount: ${total_multi_swipe_amount:,.2f}")
print(f"Total Multi-Swipe Reversed Amount: $
{total_multi_swipe_reversed_amount:,.2f}")
print(f"Total Multi-Swipe Not Reversed Amount: $
{total_multi_swipe_not_reversed_amount:,.2f}")
```

```
Total Reversals: 20303
Identified Reversals: 18165
Total Fast Reversals: 13016
Unidentified Reversals: 2138

Total Reversals Amount: $2,821,792.50
Total Identified Reversals Amount: $2,669,343.17
Total Fast Reversals Amount: $1,900,955.97
Total Unidentified Reversals Amount: $152,449.33

Total Multi-Swipes identified: 7457
Total Multi-Swipes Reversed: 152
Total Multi-Swipe Amount: $1,104,006.71
```

```
Total Multi-Swipe Reversed Amount: $25,849.02
Total Multi-Swipe Not Reversed Amount: $1,078,157.69
```

Exploring Trends in Reversals

```
matched_indices = matched_transactions_df.index.values
unidentified_reversals = df.drop(index=matched_indices)
[df.drop(index=matched_indices).transactionType=='REVERSAL']

unidentified_reversals['customerId'].value_counts().sort_values().rese
t_index().iloc[0]
```

```
customerId    153938865
count                 1
Name: 0, dtype: object
```

```
df[df.customerId=='153938865']
[['transactionDateTime','merchantName','transactionAmount','transactio
nType']]
```

{"summary":"{\n \"name\": \"df[df\",\n \"rows\": 47,\n \"fields\": [\n   {\n   \"column\": \"transactionDateTime\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"2016-01-05 18:46:49\",\n   \"max\": \"2016-12-28 14:37:40\",\n   \"num_unique_values\": 47,\n   \"samples\": [\n   \"2016-08-11 21:53:36\",\n   \"2016-11-03 03:30:28\",\n   \"2016-08-11 09:31:43\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"merchantName\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 16,\n   \"samples\": [\n   \"Eazy Repair\",\n   \"Shell Auto Body\",\n   \"Convenient Tire\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"transactionAmount\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 100.09234541111645,\n   \"min\": 0.0,\n   \"max\": 390.98,\n   \"num_unique_values\": 44,\n   \"samples\": [\n   31.49,\n   38.41,\n   114.71\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"transactionType\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"PURCHASE\",\n   \"REVERSAL\",\n   \"ADDRESS_VERIFICATION\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

At index 14037, there is a reversal for $163.14 at Shell Auto Body. Preceding it, there are 3 purchases at the same merchant for $52.78, $118.46 and $336.05. This is possibly a case where there were purchases at the same merchant totalling higher than the reversal. This is to say that a part of the total purchases was reversed. I suspect the 2138 unidentified reversals are of this kind too where a part of the purchase was reversed.

```
matched_transactions_df[matched_transactions_df.isFraud].transactionTy
pe.value_counts()
```

```
transactionType
REVERSAL                310
PURCHASE                304
ADDRESS_VERIFICATION      6
Name: count, dtype: int64
```

```
fraud_transactions_reversed =
matched_transactions_df[(matched_transactions_df.isFraud) &
(matched_transactions_df.transactionType=='REVERSAL')].transactionAmou
nt.sum()
fraud_transactions_reversed
```

```
np.float64(73556.73)
```

Hence, from the identified reversals, I can say that at least 304 of the fraud transactions, totalling $73,556 were reversed.

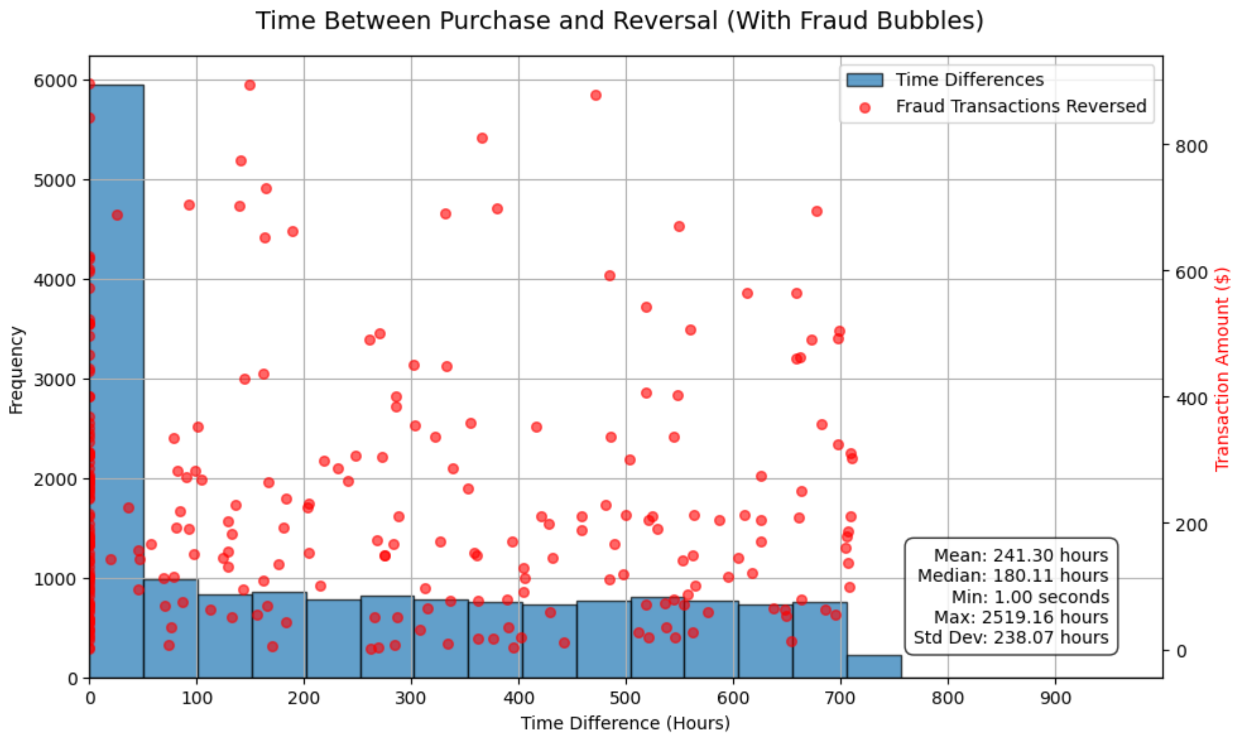
```
time_diffs = []
bubble_points = []  # (time_diff_in_minutes, transactionAmount)

for i in range(0, len(matched_transactions_df) - 1, 2):
    t1 = matched_transactions_df.iloc[i]
    t2 = matched_transactions_df.iloc[i + 1]

    if {"PURCHASE",
"REVERSAL"}.issubset({t1["transactionType"].upper(),
t2["transactionType"].upper()}):
        diff_minutes = abs((t2["transactionDateTime"] -
t1["transactionDateTime"]).total_seconds()) / 60
        time_diffs.append(diff_minutes)

        # Check if the REVERSAL transaction is marked as fraud
        if t1["transactionType"].upper() == "REVERSAL" and t1.isFraud:
            bubble_points.append((diff_minutes/60,
t1["transactionAmount"]))
        elif t2["transactionType"].upper() == "REVERSAL" and
t2.isFraud:
            bubble_points.append((diff_minutes/60,
t2["transactionAmount"]))

mean = np.mean(time_diffs)
median = np.median(time_diffs)
minimum = np.min(time_diffs)
```

```
maximum = np.max(time_diffs)
std_dev = np.std(time_diffs)
```

Lets see what the graph for times for reversal looks like, and also see the time taken for the fraud transactions to be reversed as well.

```
def format_time(mins):
    if mins < 1:
        return f"{mins * 60:.2f} seconds"
    elif mins < 60:
        return f"{mins:.2f} minutes"
    else:
        hrs = mins / 60
        return f"{hrs:.2f} hours"

stats_text = f"Mean: {format_time(mean)}\n" \
             f"Median: {format_time(median)}\n" \
             f"Min: {format_time(minimum)}\n" \
             f"Max: {format_time(maximum)}\n" \
             f"Std Dev: {format_time(std_dev)}"

# Add text box with statistics in the lower right
time_diffs = [time_diff/60 for time_diff in time_diffs]
fig, ax1 = plt.subplots(figsize=(10, 6))

# Histogram on primary y-axis
counts, bins, patches = ax1.hist(time_diffs, bins=50,
edgecolor="black", alpha=0.7, label="Time Differences")
ax1.set_xlabel("Time Difference (Hours)")
ax1.set_ylabel("Frequency")
ax1.set_xticks(np.arange(0, 1000, 100))
ax1.set_xlim(0, 1000)
ax1.grid(True)

# Create second y-axis
ax2 = ax1.twinx()

# Plot fraud reversal bubbles on secondary axis
if bubble_points:
    bubble_x, bubble_y = zip(*bubble_points)
    ax2.scatter(bubble_x, bubble_y, s=30, c='red', alpha=0.6,
label="Fraud Transactions Reversed")
    ax2.set_ylabel("Transaction Amount ($)", color='red')

# Titles and legend
fig.suptitle("Time Between Purchase and Reversal (With Fraud
Bubbles)", fontsize=14)
fig.tight_layout()
fig.legend(loc="upper right", bbox_to_anchor=(1, 1),
```

```
bbox_transform=ax1.transAxes)
plt.xticks(np.arange(0,1000,100))
ax1.text(0.95, 0.05, stats_text,
         transform=ax1.transAxes,
         horizontalalignment='right',
         verticalalignment='bottom',
         bbox=dict(facecolor='white', alpha=0.8,
boxstyle='round,pad=0.5'))
plt.show()
```



We can see that most reversals happen within the first 50 hours. There are no obvious patterns for the time taken to reverse fraudulent transactions and the amount, although it is visible that often, they are reversed in a couple of hours itself, indicated by the cluster of bubbles sticking to the left.

```
df_copy[df_copy["transactionType"]=="REVERSAL"]
["customerId"].value_counts().reset_index()
```


{"summary":"{\n \"name\":
\"df_copy[df_copy[\\\"transactionType\\\"]==\\\"REVERSAL\\\"]
[\\\"customerId\\\"]\",\n \"rows\": 3023,\n \"fields\": [\n   {\n
\"column\": \"customerId\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 3023,\n
\"samples\": [\n   \"977789958\",\n   \"334209018\",\n
\"146382124\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

```
\"std\": 23,\n        \"min\": 1,\n        \"max\": 907,\n
\"num_unique_values\": 89,\n        \"samples\": [\n          46,\n
36,\n          68\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


It was interesting that the first customer here had 907 reversals.

```
suspicious_customer = df_flagged[df_flagged.customerId=='380680241']
suspicious_customer.isFraud.value_counts()

isFraud
False    32067
True       783
Name: count, dtype: int64

df_flagged.columns[-4:]

Index(['purchase_will_be_reversed', 'reversal_matches_purchase',
       'is_multi_swipe', 'is_multi_swipe_purchase_reversed'],
      dtype='object')
```

This customer even had 783 fraud transactions

```
flags = ["purchase_will_be_reversed", "reversal_matches_purchase",
"is_multi_swipe", "is_multi_swipe_purchase_reversed"]
suspicious_customer[suspicious_customer.isFraud][flags+
['transactionType']].value_counts()

purchase_will_be_reversed  reversal_matches_purchase  is_multi_swipe
is_multi_swipe_purchase_reversed  transactionType
False                      False                      False
False                             PURCHASE               744
                           True                       False
False                             REVERSAL                14
True                       False                      False
False                             PURCHASE                13
False                      False                      True
False                             PURCHASE                 8
                                                      False
False                             REVERSAL                 2

ADDRESS_VERIFICATION      1
True                       False                      True
True                              PURCHASE                 1
Name: count, dtype: int64
```

This customer had 743 fraudulent charges on his card which were never reversed. It is surprising how the card was not closed after a good number of these.

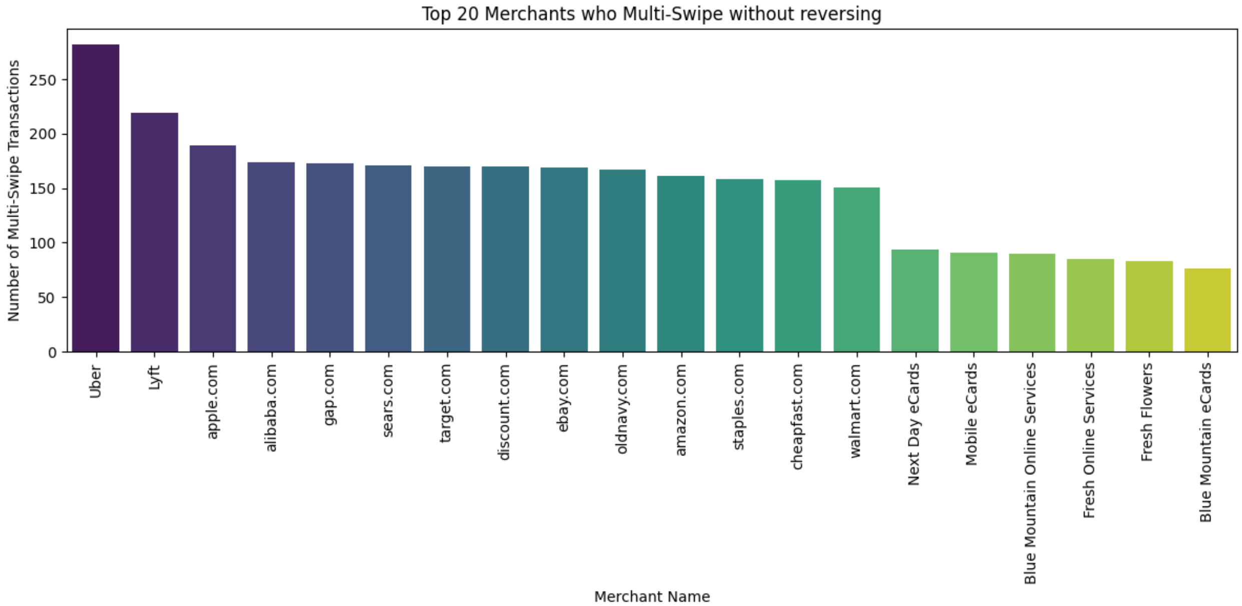
Exploring Multiswipes

```
merchant_swipe_counts =
multi_swipe_candidates["merchantName"].value_counts().reset_index()
merchant_swipe_counts.columns = ["merchantName", "multi_swipe_count"]
merchant_swipe_counts
```

{"summary":"{\n \"name\": \"merchant_swipe_counts\",\n \"rows\": 1283,\n \"fields\": [\n   {\n   \"column\": \"merchantName\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1283,\n   \"samples\": [\n   \"WSC #13185\",\n   \"Powerlifting #982788\",\n   \"KFC #383929\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"multi_swipe_count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 20,\n   \"min\": 1,\n   \"max\": 283,\n   \"num_unique_values\": 56,\n   \"samples\": [\n   283,\n   175,\n   30\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"merchant_swipe_counts"}

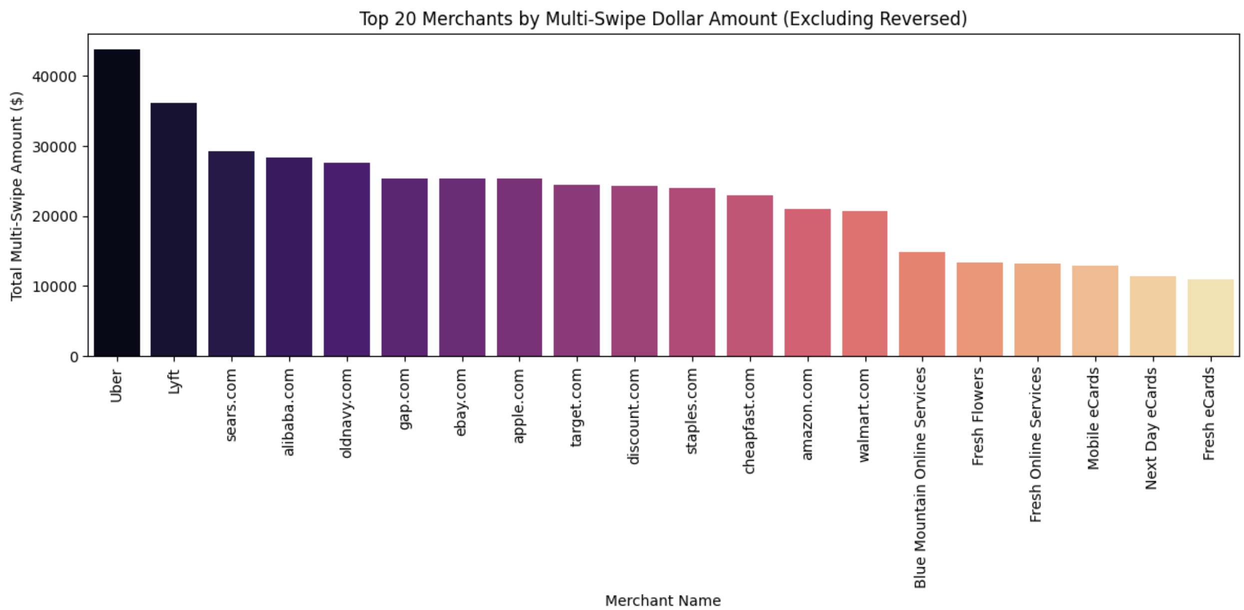
```
multi_swipe_not_reversed_df=multi_swipe_candidates.drop(index=multi_sw
ipe_reversed_df.index)
top_merchants =
multi_swipe_not_reversed_df['merchantName'].value_counts().reset_index
().head(20)

plt.figure(figsize=(12, 6))
sns.barplot(data=top_merchants, x='merchantName', y='count',
palette='viridis')
plt.xticks(rotation=90)
plt.xlabel('Merchant Name')
plt.ylabel('Number of Multi-Swipe Transactions')
plt.title('Top 20 Merchants who Multi-Swipe without reversing')
plt.tight_layout()
plt.show()
```

```
<ipython-input-93-936427627925>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=top_merchants, x='merchantName', y='count',
palette='viridis')
```




```
merchant_amounts = (
    multi_swipe_not_reversed_df.groupby('merchantName')
['transactionAmount']
        .sum()
        .reset_index()
        .rename(columns={'transactionAmount': 'totalAmount'})
        .sort_values('totalAmount', ascending=False)
)

top_amounts = merchant_amounts.head(20)

# 4. Plot
plt.figure(figsize=(12, 6))
sns.barplot(data=top_amounts, x='merchantName', y='totalAmount',
palette='magma')
plt.xticks(rotation=90)
plt.xlabel('Merchant Name')
plt.ylabel('Total Multi-Swipe Amount ($)')
plt.title('Top 20 Merchants by Multi-Swipe Dollar Amount (Excluding
Reversed)')
plt.tight_layout()
plt.show()
```

```
<ipython-input-94-db50f7df92e2>:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=top_amounts, x='merchantName', y='totalAmount',
palette='magma')
```



We can see that rideshare apps are the highest multiswipe merchants whose transactions were not reversed, both by frequency as well as by dollar amount

Answer 3

1. I was able to programmatically identify reversals of purchases as well as multiswipes (within a threshold of 3 minutes)

- Total instant reversals identified: 13016
- Dollar Amount for identified instant reversals: $1,900,955.97
- Total Multiswipes identified: 7457
- Dollar Amount for identified multiswipes: $1,104,006.71

1. Out of the 7457 multiswipe transactions totalling $1.1M, only 152 were reversed, totalling $25,849.02. This means that multi-swipes happen often and rack up over a million dollars, but almost never get reversed. This could be the customers tolerating small accidental charges or them simply not noticing them, but the vendors earn a lot of money through these multi-swipes.

Question 4: Model

Fraud is a problem for any bank. Fraud can take many forms, whether it is someone stealing a single credit card, to large batches of stolen credit card numbers being used on the web, or even a mass compromise of credit card numbers stolen from a merchant via tools like credit card skimming devices.

1. Each of the transactions in the dataset has a field called isFraud. Please build a predictive model to determine whether a given transaction will be fraudulent or not. Use as much of the data as you like (or all of it).

2. Provide an estimate of performance using an appropriate sample, and show your work.
3. Please explain your methodology (modeling algorithm/method used and why, what features/data you found useful, what questions you have, and what you would do next with more time)

Data Cleaning

```
cat_cols_stats
```

{"summary":"{\n \"name\": \"cat_cols_stats\",\n \"rows\": 22,\n \"fields\": [\n   {\n   \"column\": \"Null Count\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"0\",\n   \"max\": \"0\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"0\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Unique Values\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": 1,\n   \"max\": 5246,\n   \"num_unique_values\": 12,\n   \"samples\": [\n   1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Most Frequent\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 12,\n   \"samples\": [\n   \"\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Frequency Count\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"5103\",\n   \"max\": \"786363\",\n   \"num_unique_values\": 16,\n   \"samples\": [\n   \"32850\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Empty String Count\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"0\",\n   \"max\": \"786363\",\n   \"num_unique_values\": 7,\n   \"samples\": [\n   \"0\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"cat_cols_stats"}

I will drop the columns with all empty strings.

```
drop_cols = ["posOnPremises", "recurringAuthInd",
"merchantZip","merchantState","merchantCity","echoBuffer"]
df = df.drop(columns=drop_cols)

df[df.accountNumber != df.customerId]
```

{"type":"dataframe"}

Since accountNumber is redundant, I will drop it

```
df = df.drop(columns=["accountNumber"])
```

```
df.head()
```

{"type":"dataframe","variable_name":"df"}

I will drop non fraud duplicates with a reasonable subset of columns, which will most likely be subscriptions

```
fraud_df = df[df['isFraud']]
nonfraud_df = df[~df['isFraud']].drop_duplicates(keep='first',subset =
["customerId","transactionAmount","merchantName","accountOpenDate","me
rchantCategoryCode","cardLast4Digits"])

df = pd.concat([fraud_df, nonfraud_df]).sort_index()
```

Feature Engineering

Card CVV entered wrong

```
df[df.cardCVV != df.enteredCVV]
["isFraud"].value_counts(normalize=True)

isFraud
False    0.967946
True     0.032054
Name: proportion, dtype: float64

df["cvvMatch"] = df.cardCVV == df.enteredCVV
```

Age of account

```
df["accountAge"] = (df.transactionDateTime -
df.accountOpenDate).dt.days
```

Time since address was changed

```
df["sinceDateOfLastAddressChange"] = (
    df.transactionDateTime - df.dateOfLastAddressChange
).dt.days
```

Do country codes match

```
df["countryMatch"] = df.acqCountry == df.merchantCountryCode
```

Date properties

```
df["dayOfMonth"] = df.transactionDateTime.dt.day
df["month"] = df.transactionDateTime.dt.month
df["dayOfYear"] = df.transactionDateTime.dt.dayofyear
df["weekOfYear"] = df.transactionDateTime.dt.isocalendar().week
df["dayOfWeek"] = df.transactionDateTime.dt.dayofweek
df["quarter"] = df.transactionDateTime.dt.quarter
df["hour"] = df.transactionDateTime.dt.hour
```

Weekday or weekend

```
df["weekday"] = df.dayOfWeek < 5
```

What part of day (I made 4 parts)

```
df["partOfDay"] = pd.cut(df.hour, bins=4, labels=[0, 1, 2,3])
```

The average fraud rate for each customer based on their previous transactions — current transaction excluded.

```
df["avgFraud"] = (
    df.groupby("customerId")["isFraud"]
    .transform(lambda x: x.shift().expanding().mean())
    .fillna(0)
)

df.columns

Index(['customerId', 'creditLimit', 'availableMoney',
'transactionDateTime',
       'transactionAmount', 'merchantName', 'acqCountry',
       'merchantCountryCode', 'posEntryMode', 'posConditionCode',
       'merchantCategoryCode', 'currentExpDate', 'accountOpenDate',
       'dateOfLastAddressChange', 'cardCVV', 'enteredCVV',
'cardLast4Digits',
       'transactionType', 'currentBalance', 'cardPresent',
       'expirationDateKeyInMatch', 'isFraud', 'currentExpPeriod',
'cvvMatch',
       'accountAge', 'sinceDateOfLastAddressChange', 'countryMatch',
       'dayOfMonth', 'month', 'dayOfYear', 'weekOfYear', 'dayOfWeek',
       'quarter', 'hour', 'weekday', 'partOfDay', 'avgFraud'],
      dtype='object')
```

I will now add some features, mainly focusing on 1)Extracting temporal features for transaction activity 2)Customer-Merchant Activity and Familiarity

```
# Sort data by customer and transaction time
df = df.sort_values(["customerId", "transactionDateTime"])
```

```
# Set datetime index for time-based rolling features
df = df.set_index("transactionDateTime")

# Transaction count in recent time windows
df["transactionCountLast1hr"] = df.groupby("customerId")
["transactionAmount"].rolling("1h").count().reset_index(level=0,
drop=True)
df["transactionCountLast24hr"] = df.groupby("customerId")
["transactionAmount"].rolling("1d").count().reset_index(level=0,
drop=True)
df["transactionCountLast7d"] = df.groupby("customerId")
["transactionAmount"].rolling("7d").count().reset_index(level=0,
drop=True)

# Amount spent in recent time windows
df["amountSpentLast24hr"] = df.groupby("customerId")
["transactionAmount"].rolling("1d").sum().reset_index(level=0,
drop=True)
df["amountSpentLast7d"] = df.groupby("customerId")
["transactionAmount"].rolling("7d").sum().reset_index(level=0,
drop=True)

#Reset index back to normal for non-rolling features
df = df.reset_index()

# Unique merchants visited today
df["date"] = df["transactionDateTime"].dt.date
df["numMerchantsVisitedToday"] = df.groupby(["customerId", "date"])
["merchantName"].transform("nunique")

# Statistical amounts (past week)
df = df.sort_values(["customerId", "transactionDateTime"])
df["meanTransactionAmountPastWeek"] = df.groupby("customerId")
["transactionAmount"].transform(
    lambda x: x.rolling(window=7, min_periods=1).mean()
)


# First transaction of the day
df["isFirstTransactionToday"] = df.groupby(["customerId",
"date"]).cumcount() == 0

# Is new merchant for customer
df["isNewMerchantForCustomer"] = df.groupby("customerId")
["merchantName"].transform(lambda x: ~x.duplicated())

# Merchant frequency (normalized)
merchant_freq = df["merchantName"].value_counts(normalize=True)
df["merchantTransactionFrequency"] =
df["merchantName"].map(merchant_freq)
```

```python
# Is Domestic transaction
df["isDomesticTransaction"] = df["acqCountry"] ==
df["merchantCountryCode"]

# Frequent merchant country for each customer
customer_country_mode = df.groupby("customerId")
["merchantCountryCode"].agg(
    lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan
)
df["frequentMerchantCountryCode"] =
df["customerId"].map(customer_country_mode)
df["isFrequentMerchantCountry"] = df["merchantCountryCode"] ==
df["frequentMerchantCountryCode"]

# Time since last transaction
df["timeSinceLastTransaction"] = df.groupby("customerId")
["transactionDateTime"].diff().dt.total_seconds()

#Saving unbinned dollar amounts for later, in case
df["availableMoneyOg"] = df["availableMoney"]
df["transactionAmountOg"] = df["transactionAmount"]
# Binning available money and transaction amount
df["availableMoney"] = pd.cut(
    df.availableMoney,
    bins=[-5000, -1000, -500, -100, 0, 100, 500, 1000, 5000, 50000],
    labels=[0, 1, 2, 3, 4, 5, 6, 7, 8],
)
df["transactionAmount"] = pd.qcut(df.transactionAmount, 4, labels=[0,
1, 2, 3])

df.to_parquet('dataframes/df_cleaned.parquet', index=True)
```

Sampling

```python
import pandas as pd
df = pd.read_parquet('dataframes/df_cleaned.parquet')

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler,
OneHotEncoder,OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_selection import VarianceThreshold
from sklearn.metrics import precision_score, recall_score,
roc_auc_score, accuracy_score
```

```
from sklearn.ensemble import
RandomForestClassifier,HistGradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
import joblib

df.columns
```

```
Index(['transactionDateTime', 'customerId', 'creditLimit',
'availableMoney',
       'transactionAmount', 'merchantName', 'acqCountry',
       'merchantCountryCode', 'posEntryMode', 'posConditionCode',
       'merchantCategoryCode', 'currentExpDate', 'accountOpenDate',
       'dateOfLastAddressChange', 'cardCVV', 'enteredCVV',
'cardLast4Digits',
       'transactionType', 'currentBalance', 'cardPresent',
       'expirationDateKeyInMatch', 'isFraud', 'currentExpPeriod',
'cvvMatch',
       'accountAge', 'sinceDateOfLastAddressChange', 'countryMatch',
       'dayOfMonth', 'month', 'dayOfYear', 'weekOfYear', 'dayOfWeek',
       'quarter', 'hour', 'weekday', 'partOfDay', 'avgFraud',
       'transactionCountLast1hr', 'transactionCountLast24hr',
       'transactionCountLast7d', 'amountSpentLast24hr',
'amountSpentLast7d',
       'date', 'numMerchantsVisitedToday',
'meanTransactionAmountPastWeek',
       'isFirstTransactionToday', 'isNewMerchantForCustomer',
       'merchantTransactionFrequency', 'isDomesticTransaction',
       'frequentMerchantCountryCode', 'isFrequentMerchantCountry',
       'timeSinceLastTransaction', 'availableMoneyOg',
'transactionAmountOg'],
      dtype='object')
```

The sampling I am doing is primarily from customers who have been victims of fraud transactions as I believe. I take all the fraud transactions from them. For the non fraud transactions, I sample from the same customers because I believe this will encode more information. I sample 7.5% from these non fraud cases, which I settled on after some trial and error on the models I trained below. For some noise, I am also 1% from the customers who never experienced fraud transactions.

```
def custom_sample(input_df):
    #Get all customers who experienced fraud
    fraud_customers = input_df[input_df["isFraud"] == True]
["customerId"].unique()

    # All fraud transactions from those customers
    fraud_df = input_df[(input_df["customerId"].isin(fraud_customers))
& (input_df["isFraud"] == True)]

    # Sample 7.5% of non-fraud transactions from those same customers,
```

```
based on trial and error
    nonfraud_df_from_fraud_customers = input_df[
        (input_df["customerId"].isin(fraud_customers)) &
(input_df["isFraud"] == False)
    ]
    nonfraud_sampled =
nonfraud_df_from_fraud_customers.sample(frac=0.075, random_state=42)

    # Sample from customers who never had fraud in 2016
    nonfraud_2016_df = input_df[
        (input_df["transactionDateTime"].dt.year == 2016)
        & (~input_df["customerId"].isin(fraud_customers))
    ]

    # Identify customers who had no fraud at all in 2016
    nonfraud_customers_2016 = nonfraud_2016_df.groupby("customerId")
["isFraud"].sum()
    nonfraud_customers_2016 =
nonfraud_customers_2016[nonfraud_customers_2016 == 0].index

    # Sample transactions from these clean customers to add noise
    noise_sample =
input_df[input_df["customerId"].isin(nonfraud_customers_2016)].sample(
frac=0.01, random_state=42)

    # Combine all three subsets
    final_sampled_df = pd.concat([fraud_df, nonfraud_sampled,
noise_sample], axis=0).sample(frac=1,
random_state=42).reset_index(drop=True)
    return final_sampled_df

final_sampled_df = custom_sample(df)

final_sampled_df.isFraud.value_counts()
```

```
isFraud
False    45657
True     12417
Name: count, dtype: int64
```

```
final_sampled_df = final_sampled_df.drop(
    [
        "currentExpDate",
        "accountOpenDate",
        "dateOfLastAddressChange",
        "cardCVV",
        "enteredCVV",

"cardLast4Digits","currentExpPeriod","date","availableMoneyOg",
"transactionAmountOg"
    ],
```

```
    axis=1,
)
```

Train Test Split, preprocessers and evaluation function

```
target = "isFraud"

numeric_features = [
    "creditLimit", "availableMoney", "transactionAmount",
"currentBalance",
    "accountAge", "sinceDateOfLastAddressChange", "dayOfMonth",
"month",
    "dayOfYear", "weekOfYear", "dayOfWeek", "quarter", "hour",
"weekday",
    "transactionCountLast1hr", "transactionCountLast24hr",
"transactionCountLast7d",
    "amountSpentLast24hr", "amountSpentLast7d",
"numMerchantsVisitedToday",
    "meanTransactionAmountPastWeek",
    "merchantTransactionFrequency", "timeSinceLastTransaction"
]
categorical_features = [
    "merchantName", "acqCountry", "merchantCountryCode",
"posEntryMode",
    "posConditionCode", "merchantCategoryCode", "transactionType",
    "frequentMerchantCountryCode"
]
ordinal_features = [
    "cardPresent", "expirationDateKeyInMatch", "cvvMatch",
"countryMatch",
    "isFirstTransactionToday", "isNewMerchantForCustomer",
    "isDomesticTransaction", "isFrequentMerchantCountry"
]
features = numeric_features+categorical_features+ordinal_features
final_sampled_df = final_sampled_df[features+[target]].dropna()
X = final_sampled_df[features]
y = final_sampled_df[target]
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
test_size=0.2, random_state=42,stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.2, random_state=42,stratify=y_train_full)

# ColumnTransformers for preprocessing
preprocessor = ColumnTransformer(transformers=[
    ("num", StandardScaler(), numeric_features),
    ("cat", OneHotEncoder(handle_unknown="ignore",
```

```
sparse_output=False), categorical_features),
    ("ord", OrdinalEncoder(), ordinal_features)
])

def evaluate_pipeline(pipeline):
    y_proba = pipeline.predict_proba(X_test)[:, 1]  # probability for
class 1
    y_pred = pipeline.predict(X_test)
    # Metrics
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_proba)
    train_score = pipeline.score(X_train, y_train)
    test_score = pipeline.score(X_test, y_test)

    # Output
    print(f"Precision:      {precision:.4f}")
    print(f"Recall:         {recall:.4f}")
    print(f"AUC-ROC:        {auc:.4f}")
    print(f"Train Accuracy: {train_score:.4f}")
    print(f"Test Accuracy:  {test_score:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
```

I'll test the performance on Logistic Regression Classifier, Random Forest Classifier and HistGradientBoostingClassifier. I am using a VarianceThreshold selector to filter features that dont vary much.

Logistic Regression Classifier

```
log_reg_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("selector", VarianceThreshold()),
    ("classifier", LogisticRegression(max_iter =
400,class_weight="balanced", solver='newton-cg',n_jobs=-1))
])

%%time
# Step 6: Fit
log_reg_pipeline.fit(X_train, y_train)
```

```
CPU times: user 1.66 s, sys: 986 ms, total: 2.65 s
Wall time: 1min 43s

Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
StandardScaler(),
```

```
                                                    ['creditLimit',
                                                     'availableMoney',

'transactionAmount',
                                                     'currentBalance',
                                                     'accountAge',

'sinceDateOfLastAddressChange',
                                                     'dayOfMonth',
'month',
                                                     'dayOfYear',
'weekOfYear',
                                                     'dayOfWeek',
'quarter',
                                                     'hour', 'weekday',

'transactionCountLast1hr',

'transactionCountLast24hr',
                                                     'transactio...

'frequentMerchantCountryCode']),
                                                   ('ord',
OrdinalEncoder(),
                                                    ['cardPresent',

'expirationDateKeyInMatch',
                                                     'cvvMatch',
'countryMatch',

'isFirstTransactionToday',

'isNewMerchantForCustomer',

'isDomesticTransaction',

'isFrequentMerchantCountry'])])),
                ('selector', VarianceThreshold()),
                ('classifier',
                 LogisticRegression(class_weight='balanced',
max_iter=400,
                                    n_jobs=-1, solver='newton-cg'))])

evaluate_pipeline(log_reg_pipeline)

Precision:      0.3816
Recall:         0.7076
AUC-ROC:        0.7569
Train Accuracy: 0.7039
Test Accuracy:  0.6928
Classification Report:
```

```
              precision    recall  f1-score   support

       False       0.90      0.69      0.78      9097
        True       0.38      0.71      0.50      2469

    accuracy                           0.69     11566
   macro avg       0.64      0.70      0.64     11566
weighted avg       0.79      0.69      0.72     11566

Confusion Matrix:
[[6266 2831]
 [ 722 1747]]
```

Random Forest

```
rf_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("selector", VarianceThreshold()),
    ("classifier", RandomForestClassifier(n_estimators=100,
class_weight="balanced", n_jobs=-1, random_state=42))
])

%%time
rf_pipeline.fit(X_train, y_train)

CPU times: user 34.5 s, sys: 736 ms, total: 35.3 s
Wall time: 7.62 s

Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
StandardScaler(),
                                                  ['creditLimit',
                                                   'availableMoney',

'transactionAmount',
                                                   'currentBalance',
                                                   'accountAge',

'sinceDateOfLastAddressChange',
                                                   'dayOfMonth',
'month',
                                                   'dayOfYear',
'weekOfYear',
                                                   'dayOfWeek',
'quarter',
                                                   'hour', 'weekday',

'transactionCountLast1hr',
```

```
'transactionCountLast24hr',
                                                            'transactio...

'frequentMerchantCountryCode']),
                                                          ('ord',
OrdinalEncoder(),
                                                           ['cardPresent',

'expirationDateKeyInMatch',
                                                            'cvvMatch',
'countryMatch',

'isFirstTransactionToday',

'isNewMerchantForCustomer',

'isDomesticTransaction',

'isFrequentMerchantCountry'])])),
                ('selector', VarianceThreshold()),
                ('classifier',
                 RandomForestClassifier(class_weight='balanced',
n_jobs=-1,
                                        random_state=42))])
```

```
evaluate_pipeline(rf_pipeline)
```

```
Precision:      0.7617
Recall:         0.1385
AUC-ROC:        0.7889
Train Accuracy: 1.0000
Test Accuracy:  0.8068
Classification Report:
              precision    recall  f1-score   support

       False       0.81      0.99      0.89      9097
        True       0.76      0.14      0.23      2469

    accuracy                           0.81     11566
   macro avg       0.79      0.56      0.56     11566
weighted avg       0.80      0.81      0.75     11566

Confusion Matrix:
[[8990  107]
 [2127  342]]
```

Gradient Boosting

```
gb_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("selector", VarianceThreshold()),
    ("classifier",
HistGradientBoostingClassifier(max_iter=200,learning_rate=0.1,max_dept
h=3,random_state=42))
])
```

```
%%time
gb_pipeline.fit(X_train, y_train)
```

```
CPU times: user 1min 1s, sys: 769 ms, total: 1min 2s
Wall time: 14.3 s
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
StandardScaler(),
                                                  ['creditLimit',
                                                   'availableMoney',

'transactionAmount',
                                                   'currentBalance',
                                                   'accountAge',

'sinceDateOfLastAddressChange',
                                                   'dayOfMonth',
'month',
                                                   'dayOfYear',
'weekOfYear',
                                                   'dayOfWeek',
'quarter',
                                                   'hour', 'weekday',

'transactionCountLast1hr',

'transactionCountLast24hr',
                                                   'transactio...

'frequentMerchantCountryCode']),
                                                 ('ord',
OrdinalEncoder(),
                                                  ['cardPresent',

'expirationDateKeyInMatch',
                                                   'cvvMatch',
'countryMatch',

'isFirstTransactionToday',
```

```
'isNewMerchantForCustomer',

'isDomesticTransaction',

'isFrequentMerchantCountry'])])),
                ('selector', VarianceThreshold()),
                ('classifier',
                 HistGradientBoostingClassifier(max_depth=3,
max_iter=200,
                                                  random_state=42))])

evaluate_pipeline(gb_pipeline)

Precision:      0.6703
Recall:         0.2503
AUC-ROC:        0.7815
Train Accuracy: 0.8180
Test Accuracy:  0.8137
Classification Report:
              precision    recall  f1-score   support

       False       0.83      0.97      0.89      9097
        True       0.67      0.25      0.36      2469

    accuracy                           0.81     11566
   macro avg       0.75      0.61      0.63     11566
weighted avg       0.79      0.81      0.78     11566

Confusion Matrix:
[[8793  304]
 [1851  618]]
```

Once I got a general idea of how the models perform, I did a grid search. In case of fraud detection, it is important to be able to correctly identify fraud cases. But the fraud transactions are very underrepresented (~2%) in the data. The training and test data has been prepared such that the fraud cases are still underrepresented but the extent has been reduced to allow the model to meaningfully learn to be able to identify them. Aiming for a very high recall will mean a trade-off in terms of precision. While false positives are not as costly as false negatives in the case of fraud detection, a really low precision will quickly rack up enough costs for the bank due to the number of transactions that the bank processes in a year. Hence, I am doing the grid search with objective of maximizing the ROC-AUC score to find that model that best balances the TPR and FPR, so that I can identify which of the three models and at what parameters should I use. In fraud detection, the probabilities given by the model for fraud cases will be lower since it is so rare and the conventional threshold of 0.5 will hamper the model performance. Hence, I will be setting a custom threshold on the model with the best ROC-AUC since that model will be able to distinguish between the classes the best at different thresholds, allowing me to test which threshold balances Precision and Recall the best. Hence, the plan is: Find model with best ROC-AUC using train data --> Check F1 score with validation data at different thresholds to choose best threshold -> Use that threshold to make predictions on test data and get final metrics

GridSearch ROC AUC

```
from sklearn.model_selection import GridSearchCV


# Logistic Regression
log_reg_params = {
    'classifier__C': [0.001,0.01, 0.1, 1, 10,100],
    'classifier__solver': ['newton-cg', 'lbfgs'],
    'classifier__max_iter': [100]
}

# Random Forest
rf_params = {
    'classifier__n_estimators': [80,160, 240],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [4,5,6]
}


# GradientBoostingClassifier
gb_params = {
    'classifier__learning_rate': [0.001,0.01, 0.1],
    'classifier__max_depth': [None, 10,20],
}

log_reg_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("selector", VarianceThreshold()),
    ("classifier", LogisticRegression(random_state=42))
])
rf_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("selector", VarianceThreshold()),
    ("classifier", RandomForestClassifier(random_state=42))
])
gb_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("selector", VarianceThreshold()),
    ("classifier", HistGradientBoostingClassifier(random_state=42))
])

# Logistic Regression Grid Search
log_reg_grid = GridSearchCV(log_reg_pipeline, log_reg_params,
cv=5,scoring='roc_auc', n_jobs=-1,verbose=3)

# Random Forest Grid Search
rf_grid = GridSearchCV(rf_pipeline, rf_params, cv=5,
scoring='roc_auc', n_jobs=-1,verbose=3)
```

```python
# GradientBoosting Grid Search
gb_grid = GridSearchCV(gb_pipeline, gb_params, cv=5,
scoring='roc_auc', n_jobs=-1,verbose=3)

%%time
log_reg_grid.fit(X_train, y_train)
joblib.dump(log_reg_grid.best_estimator_,
"models/log_reg_grid_auc_roc.pkl")
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
```

```
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

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

```
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

```
https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
```

```
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

[CV 3/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.721 total time=  19.7s
[CV 3/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.738 total time=  25.5s
[CV 3/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.738 total time=  10.9s
[CV 4/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.757 total time=  46.2s
[CV 3/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.754 total time= 1.5min
[CV 3/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.747 total time= 3.9min
[CV 3/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.741 total time=11.5min
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.821 total time=  18.9s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.827 total time=  35.8s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.841 total time=  53.5s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.826 total time=  35.0s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.839 total time=  52.3s
```

```
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.828 total time=  34.7s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.838 total time=  51.4s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.793 total time=  13.7s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.783 total time=   9.8s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.793 total time=  13.7s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.780 total time=   9.8s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.792 total time=  13.7s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.826 total time=  20.4s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.834 total time=  29.9s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.814 total time=  11.0s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.823 total time=  20.3s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.821 total time=  11.0s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.809 total time=  11.2s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.833 total time=  20.3s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.813 total time=  29.1s
[CV 4/5] END classifier__learning_rate=0.001,
classifier__max_depth=None;, score=0.791 total time= 1.2min
[CV 1/5] END classifier__learning_rate=0.001,
classifier__max_depth=20;, score=0.801 total time= 1.1min
```

```
[CV 1/5] END classifier__learning_rate=0.01,
classifier__max_depth=None;, score=0.835 total time= 1.1min
[CV 2/5] END classifier__learning_rate=0.01,
classifier__max_depth=10;, score=0.841 total time= 1.1min
[CV 3/5] END classifier__learning_rate=0.01,
classifier__max_depth=20;, score=0.819 total time= 1.1min
[CV 4/5] END classifier__learning_rate=0.1,
classifier__max_depth=None;, score=0.868 total time=  59.1s
[CV 5/5] END classifier__learning_rate=0.1, classifier__max_depth=10;,
score=0.868 total time=  54.7s
[CV 5/5] END classifier__learning_rate=0.1, classifier__max_depth=20;,
score=0.868 total time=  56.4s
[CV 1/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.724 total time=   4.8s
[CV 2/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.734 total time=   4.8s
[CV 3/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.711 total time=   4.8s
[CV 5/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.715 total time=   4.8s
[CV 1/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.744 total time=   9.1s
[CV 2/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.755 total time=   9.0s
[CV 3/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.729 total time=   8.9s
[CV 4/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.753 total time=  38.6s
[CV 3/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.745 total time= 1.1min
[CV 4/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.743 total time= 3.2min
[CV 1/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.749 total time=  10.7s
[CV 2/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.759 total time=  10.7s
[CV 3/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.742 total time=  10.8s
[CV 4/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.748 total time=  11.0s
[CV 5/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.743 total time=  10.8s
CPU times: user 1min 52s, sys: 4.2 s, total: 1min 57s
Wall time: 16min 53s

['log_reg_grid_auc_roc.pkl']
```

```
%%time
rf_grid.fit(X_train, y_train)
joblib.dump(rf_grid.best_estimator_, "models/rf_grid_auc_roc.pkl")
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
[CV 5/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.725 total time=  18.3s
[CV 5/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.725 total time=   8.4s
[CV 1/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.753 total time=  10.8s
[CV 2/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.760 total time=  10.6s
[CV 5/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.745 total time=  10.6s
[CV 1/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.760 total time=  11.0s
[CV 2/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.769 total time=  10.8s
[CV 3/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.746 total time=  11.0s
[CV 4/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.756 total time=  10.8s
[CV 4/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.757 total time= 1.7min
[CV 1/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.760 total time=  10.9s
[CV 2/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.769 total time=  10.7s
[CV 3/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.747 total time=  10.7s
[CV 4/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.755 total time=  11.1s
[CV 5/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.754 total time=  11.0s
[CV 1/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.753 total time=14.0min
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.841 total time=  19.0s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.838 total time=  36.4s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.829 total time=  18.5s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.819 total time=  18.5s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.835 total time=  35.3s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=240;,
```

```
score=0.849 total time=  52.2s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.831 total time=  18.5s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.839 total time=  51.3s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.779 total time=   5.8s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.792 total time=   9.8s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.782 total time=   9.8s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.794 total time=  13.6s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.781 total time=   5.8s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.785 total time=   9.7s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.782 total time=  13.7s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.803 total time=   9.8s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.780 total time=  13.7s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.816 total time=  11.1s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.818 total time=  20.6s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.819 total time=  11.1s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.808 total time=  11.0s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.835 total time=  20.5s
[CV 3/5] END classifier__max_depth=20,
```

```
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.812 total time=  29.6s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.817 total time=  11.1s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.826 total time=  29.2s
[CV 2/5] END classifier__learning_rate=0.001,
classifier__max_depth=None;, score=0.810 total time= 1.1min
[CV 5/5] END classifier__learning_rate=0.001,
classifier__max_depth=10;, score=0.796 total time= 1.1min
[CV 5/5] END classifier__learning_rate=0.001,
classifier__max_depth=20;, score=0.796 total time= 1.1min
[CV 5/5] END classifier__learning_rate=0.01,
classifier__max_depth=None;, score=0.831 total time= 1.1min
[CV 1/5] END classifier__learning_rate=0.01,
classifier__max_depth=20;, score=0.835 total time= 1.1min
[CV 2/5] END classifier__learning_rate=0.1,
classifier__max_depth=None;, score=0.874 total time=  58.5s
[CV 3/5] END classifier__learning_rate=0.1, classifier__max_depth=10;,
score=0.858 total time=  56.0s
[CV 4/5] END classifier__learning_rate=0.1, classifier__max_depth=20;,
score=0.868 total time=  57.7s
[CV 5/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.715 total time=  14.6s
[CV 4/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.723 total time=   4.8s
[CV 5/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.735 total time=  23.2s
[CV 3/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.739 total time=  38.2s
[CV 2/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.766 total time= 1.1min
[CV 3/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.740 total time= 3.2min
[CV 5/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.735 total time= 9.3min
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.781 total time=  21.2s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.788 total time=  41.0s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.782 total time= 1.0min
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.782 total time=  40.6s
```

```
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.783 total time=  20.8s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.765 total time=  20.5s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.789 total time=  39.7s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.791 total time=  58.2s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.738 total time=   9.3s
[CV 4/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.726 total time=  18.2s
[CV 4/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.726 total time=   7.8s
[CV 5/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.745 total time=  26.8s
[CV 2/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.770 total time=  44.1s
[CV 5/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.753 total time=  11.0s
[CV 1/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.761 total time=  10.7s
[CV 2/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.769 total time=  11.0s
[CV 3/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.747 total time=  10.9s
[CV 4/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.757 total time=  10.9s
[CV 5/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.754 total time=  11.0s
[CV 1/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.758 total time= 4.9min
[CV 5/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.742 total time=12.4min
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.835 total time=  19.1s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.842 total time=  53.2s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.841 total time=  18.6s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=80;,
```

```
score=0.831 total time=  18.6s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.838 total time=  52.1s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.841 total time=  18.4s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.837 total time=  18.2s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.842 total time=  34.6s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.797 total time=   5.8s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.800 total time=   5.8s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.777 total time=   5.9s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.787 total time=   5.8s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.804 total time=   9.8s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.780 total time=   9.9s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.801 total time=  13.7s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.784 total time=   5.8s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.793 total time=   9.8s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.785 total time=   5.8s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.799 total time=   5.7s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.780 total time=   5.8s
[CV 5/5] END classifier__max_depth=10,
```

```
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.793 total time=   9.8s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.822 total time=  11.1s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.836 total time=  20.7s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.813 total time=  30.0s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.820 total time=  11.1s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.825 total time=  29.3s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.831 total time=  11.0s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.824 total time=  20.2s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.834 total time=  29.1s
[CV 3/5] END classifier__learning_rate=0.001,
classifier__max_depth=None;, score=0.775 total time= 1.1min
[CV 3/5] END classifier__learning_rate=0.001,
classifier__max_depth=10;, score=0.775 total time= 1.1min
[CV 4/5] END classifier__learning_rate=0.001,
classifier__max_depth=20;, score=0.791 total time= 1.2min
[CV 1/5] END classifier__learning_rate=0.01,
classifier__max_depth=10;, score=0.835 total time= 1.1min
[CV 2/5] END classifier__learning_rate=0.01,
classifier__max_depth=20;, score=0.842 total time= 1.1min
[CV 3/5] END classifier__learning_rate=0.1,
classifier__max_depth=None;, score=0.859 total time=  59.2s
[CV 4/5] END classifier__learning_rate=0.1, classifier__max_depth=10;,
score=0.870 total time=  56.7s
[CV 1/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.724 total time=  15.5s
[CV 2/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.755 total time=  22.9s
[CV 4/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.744 total time=   9.3s
[CV 5/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.746 total time=  38.8s
[CV 4/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.753 total time= 1.2min
```

```
[CV 5/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.742 total time= 2.7min
[CV 4/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.737 total time=11.1min
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.777 total time=  21.4s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.781 total time=  40.6s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.784 total time= 1.0min
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.773 total time=  39.6s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.783 total time=  59.4s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.771 total time=  38.8s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.786 total time=  57.9s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.747 total time=   5.6s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.744 total time=   5.5s
[CV 1/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.731 total time=   8.7s
[CV 2/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.739 total time=   9.0s
[CV 3/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.721 total time=   8.6s
[CV 4/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.748 total time=  27.6s
[CV 3/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.748 total time=  46.4s
[CV 2/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.772 total time= 1.5min
[CV 2/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.761 total time= 4.0min
[CV 4/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.741 total time=13.7min
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.836 total time=  18.7s
```

```
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.846 total time=  36.0s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.830 total time=  52.7s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.847 total time=  35.2s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.827 total time=  51.6s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.846 total time=  35.0s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.828 total time=  51.1s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.781 total time=  13.8s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.805 total time=   9.9s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.782 total time=  13.7s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.787 total time=   9.8s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.800 total time=  13.7s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.812 total time=  11.3s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.828 total time=  20.4s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.828 total time=  29.7s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.816 total time=  20.3s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.825 total time=  29.6s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=160;,
```

```
score=0.818 total time=  20.2s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.826 total time=  28.6s
[CV 1/5] END classifier__learning_rate=0.001,
classifier__max_depth=10;, score=0.801 total time= 1.2min
[CV 2/5] END classifier__learning_rate=0.001,
classifier__max_depth=20;, score=0.810 total time= 1.2min
[CV 3/5] END classifier__learning_rate=0.01,
classifier__max_depth=None;, score=0.819 total time= 1.2min
[CV 4/5] END classifier__learning_rate=0.01,
classifier__max_depth=10;, score=0.832 total time= 1.2min
[CV 5/5] END classifier__learning_rate=0.01,
classifier__max_depth=20;, score=0.831 total time= 1.2min
[CV 1/5] END classifier__learning_rate=0.1, classifier__max_depth=10;,
score=0.869 total time=  57.1s
[CV 2/5] END classifier__learning_rate=0.1, classifier__max_depth=20;,
score=0.874 total time= 1.0min
[CV 3/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.711 total time=  16.0s
[CV 3/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.729 total time=  23.8s
[CV 1/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.754 total time=  40.7s
[CV 1/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.757 total time= 1.1min
[CV 2/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.755 total time= 2.6min
[CV 2/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.748 total time=10.8min
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.765 total time=  21.4s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.789 total time= 1.0min
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.781 total time=  20.8s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.777 total time=  20.9s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.790 total time=  58.9s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.786 total time=  20.5s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=80;,
```

```
score=0.777 total time=  20.4s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.779 total time=  39.2s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.747 total time=   5.6s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.755 total time=   5.6s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.728 total time=   5.6s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.749 total time=   5.6s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.751 total time=   9.4s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.732 total time=   9.3s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.749 total time=  13.0s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.740 total time=  12.9s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.739 total time=   9.2s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.747 total time=   5.6s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.724 total time=   5.5s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.754 total time=   9.3s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.732 total time=  12.9s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.763 total time=  10.6s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.764 total time=  19.3s
[CV 1/5] END classifier__max_depth=20,
```

```
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.769 total time=  10.7s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.840 total time=  36.1s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.849 total time=  53.1s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.835 total time=  18.7s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.839 total time=  35.7s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.836 total time=  18.6s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.821 total time=  18.5s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.837 total time=  35.2s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.848 total time=  51.8s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.791 total time=   9.9s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.780 total time=  14.0s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.790 total time=   5.9s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.793 total time=  13.8s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.782 total time=   5.9s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.783 total time=   9.8s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.780 total time=  13.9s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.823 total time=  11.2s
```

```
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.826 total time=  29.9s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.829 total time=  11.1s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.826 total time=  20.5s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.836 total time=  29.9s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.817 total time=  11.2s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.824 total time=  20.6s
[CV 1/5] END classifier__learning_rate=0.001,
classifier__max_depth=None;, score=0.801 total time= 1.1min
[CV 4/5] END classifier__learning_rate=0.001,
classifier__max_depth=10;, score=0.791 total time= 1.2min
[CV 2/5] END classifier__learning_rate=0.01,
classifier__max_depth=None;, score=0.842 total time= 1.2min
[CV 3/5] END classifier__learning_rate=0.01,
classifier__max_depth=10;, score=0.819 total time= 1.2min
[CV 4/5] END classifier__learning_rate=0.01,
classifier__max_depth=20;, score=0.832 total time= 1.2min
[CV 5/5] END classifier__learning_rate=0.1,
classifier__max_depth=None;, score=0.868 total time=  58.8s
[CV 1/5] END classifier__learning_rate=0.1, classifier__max_depth=20;,
score=0.871 total time=  58.8s
[CV 2/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.734 total time=  15.0s
[CV 1/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.744 total time=  24.4s
[CV 5/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.735 total time=   9.2s
[CV 1/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.754 total time=  10.3s
[CV 2/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.764 total time=  10.4s
[CV 3/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.739 total time=  10.4s
[CV 5/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.746 total time=  10.2s
[CV 1/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.754 total time=  10.2s
[CV 2/5] END classifier__C=1, classifier__max_iter=100,
```

```
classifier__solver=lbfgs;, score=0.763 total time=  10.3s
[CV 3/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.744 total time=  10.4s
[CV 4/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.752 total time=  10.5s
[CV 5/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.745 total time=  10.4s
[CV 1/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.752 total time= 2.7min
[CV 3/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.734 total time=11.4min
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.777 total time=  21.3s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.782 total time=  41.0s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.782 total time=  20.9s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.767 total time=  20.8s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.790 total time=  39.9s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.791 total time=  58.9s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.781 total time=  20.6s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.791 total time=  58.2s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.734 total time=   5.6s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.755 total time=   9.4s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.749 total time=   9.2s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.756 total time=  13.1s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=80;,
```

```
score=0.752 total time=   5.5s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.736 total time=   5.5s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.751 total time=  13.0s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.751 total time=   5.6s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.750 total time=   9.4s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.757 total time=  12.9s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.753 total time=  10.7s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.766 total time=  19.2s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.766 total time=  28.1s
[CV 2/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.739 total time=  18.8s
[CV 2/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.760 total time=  29.9s
[CV 1/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.762 total time=  50.1s
[CV 1/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.764 total time= 1.6min
[CV 4/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.747 total time= 3.7min
[CV 2/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.753 total time=11.0min
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.837 total time=  19.1s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.841 total time=  36.0s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.843 total time=  53.2s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.838 total time=  35.6s
```

```
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.841 total time=  52.1s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.835 total time=  35.1s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.842 total time=  51.0s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.790 total time=   5.8s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.801 total time=   5.8s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.793 total time=   9.7s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.801 total time=  13.8s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.790 total time=   5.8s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.786 total time=  13.6s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.828 total time=  11.2s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.814 total time=  20.6s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.820 total time=  29.7s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.812 total time=  20.2s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.818 total time=  29.5s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.812 total time=  20.3s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.820 total time=  28.7s
[CV 5/5] END classifier__learning_rate=0.001,
```

```
classifier__max_depth=None;, score=0.796 total time= 1.1min
[CV 2/5] END classifier__learning_rate=0.001,
classifier__max_depth=10;, score=0.810 total time= 1.1min
[CV 3/5] END classifier__learning_rate=0.001,
classifier__max_depth=20;, score=0.775 total time= 1.1min
[CV 4/5] END classifier__learning_rate=0.01,
classifier__max_depth=None;, score=0.832 total time= 1.1min
[CV 5/5] END classifier__learning_rate=0.01,
classifier__max_depth=10;, score=0.831 total time= 1.1min
[CV 1/5] END classifier__learning_rate=0.1,
classifier__max_depth=None;, score=0.871 total time=  56.7s
[CV 2/5] END classifier__learning_rate=0.1, classifier__max_depth=10;,
score=0.875 total time=  53.8s
[CV 3/5] END classifier__learning_rate=0.1, classifier__max_depth=20;,
score=0.859 total time=  58.4s
[CV 4/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.723 total time=  17.2s
[CV 4/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.744 total time=  24.6s
[CV 2/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.764 total time=  36.3s
[CV 4/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.753 total time=  11.2s
[CV 5/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.749 total time= 1.3min
[CV 1/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.750 total time=  10.4s
[CV 2/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.759 total time=  10.4s
[CV 3/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.742 total time=  10.7s
[CV 4/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.749 total time=  10.7s
[CV 5/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.742 total time=  10.3s
[CV 1/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.747 total time= 8.7min
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.783 total time=  21.3s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.770 total time=  40.3s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.772 total time=  59.6s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.789 total time=  39.8s
```

```
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.774 total time=  58.5s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.790 total time=  39.1s
[CV 3/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.771 total time=  57.2s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.751 total time=  13.0s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.730 total time=   9.2s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.750 total time=  12.9s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.729 total time=   9.3s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.751 total time=  12.9s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.772 total time=  19.7s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.779 total time=  28.1s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.767 total time=  10.5s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.767 total time=  19.2s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.772 total time=  10.6s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.758 total time=  10.6s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.778 total time=  19.3s
CPU times: user 1min 15s, sys: 869 ms, total: 1min 16s
Wall time: 9min 58s

['rf_grid_auc_roc.pkl']
```

```
%%time
gb_grid.fit(X_train, y_train)
joblib.dump(gb_grid.best_estimator_, "models/gb_grid_auc_roc.pkl")
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
CPU times: user 1min 9s, sys: 6.19 s, total: 1min 15s
Wall time: 8min 15s
```

```
['gb_grid_auc_roc.pkl']
```

```
print("Best Logistic Regression Params:", log_reg_grid.best_params_)
evaluate_pipeline(log_reg_grid.best_estimator_)
```

```
Best Logistic Regression Params: {'classifier__C': 1,
'classifier__max_iter': 100, 'classifier__solver': 'newton-cg'}
Precision:      0.6108
Recall:         0.1977
AUC-ROC:        0.7565
Train Accuracy: 0.8059
Test Accuracy:  0.8018
Classification Report:
              precision    recall  f1-score   support

       False       0.82      0.97      0.88      9097
        True       0.61      0.20      0.30      2469

    accuracy                           0.80     11566
   macro avg       0.71      0.58      0.59     11566
weighted avg       0.77      0.80      0.76     11566

Confusion Matrix:
[[8786  311]
 [1981  488]]
```

```
# Random Forest
print("\n Best Random Forest Params:", rf_grid.best_params_)
evaluate_pipeline(rf_grid.best_estimator_)
```

```

 Best Random Forest Params: {'classifier__max_depth': None,
'classifier__min_samples_split': 5, 'classifier__n_estimators': 240}
Precision:      0.7917
Recall:         0.1539
AUC-ROC:        0.7953
Train Accuracy: 0.9802
Test Accuracy:  0.8107
Classification Report:
              precision    recall  f1-score   support

       False       0.81      0.99      0.89      9097
        True       0.79      0.15      0.26      2469
```

```
    accuracy                           0.81     11566
   macro avg       0.80      0.57      0.57     11566
weighted avg       0.81      0.81      0.76     11566

Confusion Matrix:
[[8997  100]
 [2089  380]]

# GradientBoostingClassifier
print("\n Best GradientBoosting Params:", gb_grid.best_params_)
evaluate_pipeline(gb_grid.best_estimator_)


 Best GradientBoosting Params: {'classifier__learning_rate': 0.1,
'classifier__max_depth': None}
Precision:      0.6809
Recall:         0.2904
AUC-ROC:        0.8067
Train Accuracy: 0.8384
Test Accuracy:  0.8195
Classification Report:
              precision    recall  f1-score   support

       False       0.83      0.96      0.89      9097
        True       0.68      0.29      0.41      2469

    accuracy                           0.82     11566
   macro avg       0.76      0.63      0.65     11566
weighted avg       0.80      0.82      0.79     11566

Confusion Matrix:
[[8761  336]
 [1752  717]]
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.733 total time=  13.1s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.754 total time=   9.4s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.733 total time=  13.1s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.739 total time=   5.6s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.750 total time=  13.1s
[CV 2/5] END classifier__max_depth=20,
```

```
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.772 total time=  10.8s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.758 total time=  19.7s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.769 total time=  28.4s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.758 total time=  19.6s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.771 total time=  28.1s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.761 total time=  19.6s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.770 total time=  27.6s
[CV 5/5] END classifier__learning_rate=0.001,
classifier__max_depth=None;, score=0.723 total time= 1.1min
[CV 5/5] END classifier__learning_rate=0.001,
classifier__max_depth=10;, score=0.723 total time= 1.1min
[CV 2/5] END classifier__learning_rate=0.01,
classifier__max_depth=None;, score=0.761 total time= 1.1min
[CV 3/5] END classifier__learning_rate=0.01,
classifier__max_depth=10;, score=0.743 total time= 1.1min
[CV 4/5] END classifier__learning_rate=0.01,
classifier__max_depth=20;, score=0.759 total time= 1.1min
[CV 4/5] END classifier__learning_rate=0.1,
classifier__max_depth=None;, score=0.794 total time=  51.4s
[CV 4/5] END classifier__learning_rate=0.1, classifier__max_depth=10;,
score=0.792 total time=  49.1s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.748 total time=   9.2s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.740 total time=  12.9s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.748 total time=   9.2s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.743 total time=  13.0s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.777 total time=  19.5s
```

```
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.760 total time=  28.4s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.767 total time=  10.6s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.773 total time=  28.2s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.774 total time=  10.5s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.764 total time=  10.6s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.775 total time=  28.1s
[CV 2/5] END classifier__learning_rate=0.001,
classifier__max_depth=None;, score=0.745 total time= 1.1min
[CV 1/5] END classifier__learning_rate=0.001,
classifier__max_depth=20;, score=0.742 total time= 1.1min
[CV 1/5] END classifier__learning_rate=0.01,
classifier__max_depth=None;, score=0.757 total time= 1.1min
[CV 2/5] END classifier__learning_rate=0.01,
classifier__max_depth=10;, score=0.761 total time= 1.1min
[CV 3/5] END classifier__learning_rate=0.01,
classifier__max_depth=20;, score=0.743 total time= 1.1min
[CV 5/5] END classifier__learning_rate=0.1,
classifier__max_depth=None;, score=0.792 total time=  52.5s
[CV 5/5] END classifier__learning_rate=0.1, classifier__max_depth=10;,
score=0.790 total time=  50.0s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.756 total time=  10.7s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.778 total time=  19.5s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.758 total time=  28.0s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.773 total time=  19.4s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.778 total time=  27.4s
[CV 3/5] END classifier__learning_rate=0.001,
classifier__max_depth=None;, score=0.722 total time= 1.1min
```

```
[CV 2/5] END classifier__learning_rate=0.001,
classifier__max_depth=20;, score=0.745 total time= 1.1min
[CV 3/5] END classifier__learning_rate=0.01,
classifier__max_depth=None;, score=0.743 total time= 1.2min
[CV 4/5] END classifier__learning_rate=0.01,
classifier__max_depth=10;, score=0.759 total time= 1.1min
[CV 5/5] END classifier__learning_rate=0.01,
classifier__max_depth=20;, score=0.745 total time= 1.1min
[CV 1/5] END classifier__learning_rate=0.1, classifier__max_depth=10;,
score=0.796 total time=  46.1s
[CV 2/5] END classifier__learning_rate=0.1, classifier__max_depth=20;,
score=0.805 total time=  55.9s
[CV 3/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.760 total time=  27.7s
[CV 4/5] END classifier__learning_rate=0.001,
classifier__max_depth=None;, score=0.738 total time= 1.0min
[CV 2/5] END classifier__learning_rate=0.001,
classifier__max_depth=10;, score=0.745 total time= 1.1min
[CV 3/5] END classifier__learning_rate=0.001,
classifier__max_depth=20;, score=0.722 total time= 1.1min
[CV 4/5] END classifier__learning_rate=0.01,
classifier__max_depth=None;, score=0.759 total time= 1.1min
[CV 5/5] END classifier__learning_rate=0.01,
classifier__max_depth=10;, score=0.744 total time= 1.1min
[CV 1/5] END classifier__learning_rate=0.1,
classifier__max_depth=None;, score=0.799 total time=  39.1s
[CV 2/5] END classifier__learning_rate=0.1, classifier__max_depth=10;,
score=0.804 total time=  49.8s
[CV 3/5] END classifier__learning_rate=0.1, classifier__max_depth=20;,
score=0.791 total time=  52.1s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.769 total time=  19.1s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.767 total time=  28.0s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.768 total time=  19.0s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.766 total time=  27.2s
[CV 1/5] END classifier__learning_rate=0.001,
classifier__max_depth=10;, score=0.742 total time= 1.1min
[CV 4/5] END classifier__learning_rate=0.001,
classifier__max_depth=10;, score=0.738 total time= 1.1min
[CV 4/5] END classifier__learning_rate=0.001,
classifier__max_depth=20;, score=0.738 total time= 1.1min
```

```
[CV 5/5] END classifier__learning_rate=0.01,
classifier__max_depth=None;, score=0.745 total time= 1.1min
[CV 2/5] END classifier__learning_rate=0.01,
classifier__max_depth=20;, score=0.761 total time= 1.1min
[CV 3/5] END classifier__learning_rate=0.1,
classifier__max_depth=None;, score=0.794 total time=  55.2s
[CV 1/5] END classifier__learning_rate=0.1, classifier__max_depth=20;,
score=0.799 total time=  40.8s
[CV 4/5] END classifier__learning_rate=0.1, classifier__max_depth=20;,
score=0.795 total time=  51.1s
[CV 1/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=160;,
score=0.789 total time=  40.9s
[CV 2/5] END classifier__max_depth=None,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.790 total time=  60.0s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.779 total time=  21.0s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.778 total time=  40.1s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.781 total time=  59.1s
[CV 4/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.785 total time=  39.4s
[CV 5/5] END classifier__max_depth=None,
classifier__min_samples_split=6, classifier__n_estimators=240;,
score=0.780 total time=  57.7s
[CV 3/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.726 total time=   5.5s
[CV 1/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.751 total time=   9.4s
[CV 2/5] END classifier__max_depth=10,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.755 total time=  13.0s
[CV 4/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.748 total time=   5.6s
[CV 5/5] END classifier__max_depth=10,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.740 total time=   9.2s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.768 total time=  10.6s
```

```
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=80;,
score=0.763 total time=  10.6s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=4, classifier__n_estimators=240;,
score=0.773 total time=  28.3s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=80;,
score=0.774 total time=  10.5s
[CV 1/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=160;,
score=0.774 total time=  19.6s
[CV 2/5] END classifier__max_depth=20,
classifier__min_samples_split=5, classifier__n_estimators=240;,
score=0.779 total time=  28.0s
[CV 4/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=80;,
score=0.765 total time=  10.5s
[CV 5/5] END classifier__max_depth=20,
classifier__min_samples_split=6, classifier__n_estimators=160;,
score=0.765 total time=  19.0s
[CV 1/5] END classifier__learning_rate=0.001,
classifier__max_depth=None;, score=0.742 total time= 1.1min
[CV 3/5] END classifier__learning_rate=0.001,
classifier__max_depth=10;, score=0.722 total time= 1.1min
[CV 5/5] END classifier__learning_rate=0.001,
classifier__max_depth=20;, score=0.723 total time= 1.1min
[CV 1/5] END classifier__learning_rate=0.01,
classifier__max_depth=10;, score=0.757 total time= 1.1min
[CV 1/5] END classifier__learning_rate=0.01,
classifier__max_depth=20;, score=0.757 total time= 1.1min
[CV 2/5] END classifier__learning_rate=0.1,
classifier__max_depth=None;, score=0.805 total time=  52.3s
[CV 3/5] END classifier__learning_rate=0.1, classifier__max_depth=10;,
score=0.794 total time=  50.5s
[CV 5/5] END classifier__learning_rate=0.1, classifier__max_depth=20;,
score=0.793 total time=  50.6s
```

The best performing model here is the HistGradientBoostingClassifier with a ROC AUC of 80.61. However, I feel like this can be improved so I'll try adding some more features so that I am more confident when moving on to threshold tuning.

Adding Some More Features

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```python
from sklearn.preprocessing import StandardScaler,
OneHotEncoder,OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_selection import VarianceThreshold
from sklearn.metrics import precision_score, recall_score,
roc_auc_score, accuracy_score
from sklearn.ensemble import
RandomForestClassifier,HistGradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
import joblib



df = pd.read_parquet('dataframes/df_cleaned.parquet')

# Amount ratios
df["amtCreditLimitRatio"] = df["transactionAmountOg"] /
df["creditLimit"]
df["amtAvailableRatio"]   = df["transactionAmountOg"] /
df["availableMoneyOg"]

# Relative transaction amount vs customer average transaction amount
df["custAvgTxnAmt"] = df.groupby("customerId")
["transactionAmountOg"].transform("mean")
df["relativeTxnAmt"] = df["transactionAmountOg"] / df["custAvgTxnAmt"]

# Merchant-category diversity in past 7 days
df =
df.sort_values(["customerId","transactionDateTime"]).set_index("transa
ctionDateTime")
df["merchantCategoryCode_cat"] =
df["merchantCategoryCode"].astype("category").cat.codes
# 7-day distinct merchant-category count
df["catDivLast7d"] = (
    df
      .groupby("customerId")["merchantCategoryCode_cat"]
      .rolling("7d")
      .apply(lambda x: x.nunique(), raw=False)
      .reset_index(level=0, drop=True)
)
df.drop(columns=["merchantCategoryCode_cat"], inplace=True)
df = df.reset_index()

# Customer's typical transaction hour and deviation
typical_hour = (
    df.groupby("customerId")["hour"]
      .agg(lambda x: x.mode().iloc[0] if not x.mode().empty else
np.nan)
```

```
)
df["typicalHour"]    = df["customerId"].map(typical_hour)
df["hourDeviation"]  = (df["hour"] - df["typicalHour"]).abs()

# Cumulative fraud count per customer
df["cumFraudCount"] = df.groupby("customerId")["isFraud"].cumsum()

# Flag for first transaction in past week
df["firstTxnLast7d"] = (
    df.groupby("customerId")["transactionDateTime"]
      .transform(lambda x: x.diff().gt(pd.Timedelta("7d")))
      .astype(int)
)

def custom_sample(input_df):
    #Get all customers who experienced fraud
    fraud_customers = input_df[input_df["isFraud"] == True]
["customerId"].unique()

    # All fraud transactions from those customers
    fraud_df = input_df[(input_df["customerId"].isin(fraud_customers))
& (input_df["isFraud"] == True)]

    # Sample 7.5% of non-fraud transactions from those same customers,
based on trial and error
    nonfraud_df_from_fraud_customers = input_df[
        (input_df["customerId"].isin(fraud_customers)) &
(input_df["isFraud"] == False)
    ]
    nonfraud_sampled =
nonfraud_df_from_fraud_customers.sample(frac=0.075, random_state=42)

    # Sample from customers who never had fraud in 2016
    nonfraud_2016_df = input_df[
        (input_df["transactionDateTime"].dt.year == 2016)
        & (~input_df["customerId"].isin(fraud_customers))
    ]

    # Identify customers who had no fraud at all in 2016
    nonfraud_customers_2016 = nonfraud_2016_df.groupby("customerId")
["isFraud"].sum()
    nonfraud_customers_2016 =
nonfraud_customers_2016[nonfraud_customers_2016 == 0].index

    # Sample transactions from these clean customers to add noise
    noise_sample =
input_df[input_df["customerId"].isin(nonfraud_customers_2016)].sample(
frac=0.01, random_state=42)

    # Combine all three subsets
```

```
    final_sampled_df = pd.concat([fraud_df, nonfraud_sampled,
noise_sample], axis=0).sample(frac=1,
random_state=42).reset_index(drop=True)
    return final_sampled_df

target = "isFraud"

numeric_features = [
    "creditLimit", "availableMoney", "transactionAmount",
"currentBalance",
    "accountAge", "sinceDateOfLastAddressChange", "dayOfMonth",
"month",
    "dayOfYear", "weekOfYear", "dayOfWeek", "quarter", "hour",
"weekday",
    "transactionCountLast1hr", "transactionCountLast24hr",
"transactionCountLast7d",
    "amountSpentLast24hr", "amountSpentLast7d",
"numMerchantsVisitedToday",
    "meanTransactionAmountPastWeek",
    "merchantTransactionFrequency", "timeSinceLastTransaction"
]
categorical_features = [
    "merchantName", "acqCountry", "merchantCountryCode",
"posEntryMode",
    "posConditionCode", "merchantCategoryCode", "transactionType",
    "frequentMerchantCountryCode"
]
ordinal_features = [
    "cardPresent", "expirationDateKeyInMatch", "cvvMatch",
"countryMatch",
    "isFirstTransactionToday", "isNewMerchantForCustomer",
    "isDomesticTransaction", "isFrequentMerchantCountry"
]

final_sampled_df = custom_sample(df)

final_sampled_df = final_sampled_df.drop(
    [
        "currentExpDate",
        "accountOpenDate",
        "dateOfLastAddressChange",
        "cardCVV",
        "enteredCVV",

"cardLast4Digits","currentExpPeriod","date","availableMoneyOg",
"transactionAmountOg"
    ],
    axis=1,
)
```

```
new_numeric_features = numeric_features+ [
"amtCreditLimitRatio","amtAvailableRatio","custAvgTxnAmt","relativeTxn
Amt",

"catDivLast7d","hourDeviation","cumFraudCount"  ]
new_categorical_features = categorical_features
new_ordinal_features = ordinal_features+
["typicalHour","firstTxnLast7d" ]
features =
new_numeric_features+new_categorical_features+new_ordinal_features
final_sampled_df = final_sampled_df[features+[target]].dropna()
X = final_sampled_df[features]
y = final_sampled_df[target]
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
test_size=0.2, random_state=42,stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.2, random_state=42,stratify=y_train_full)

preprocessor = ColumnTransformer(transformers=[
    ("num", StandardScaler(), new_numeric_features),
    ("cat", OneHotEncoder(handle_unknown="ignore",
sparse_output=False), new_categorical_features),
    ("ord", OrdinalEncoder(), new_ordinal_features)
])
```

Grid Search ROC AUC with more features

```
# Logistic Regression
log_reg_params = {
    'classifier__C': [0.001,0.01, 0.1, 1, 10,100],
    'classifier__solver': ['newton-cg', 'lbfgs'],
    'classifier__max_iter': [100]
}

# Random Forest
rf_params = {
    'classifier__n_estimators': [80,160, 240],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [4,5,6]
}

# GradientBoostingClassifier
gb_params = {
    'classifier__learning_rate': [0.001,0.01, 0.1],
    'classifier__max_depth': [None, 10,20],
}

log_reg_pipeline = Pipeline(steps=[
```

```
    ("preprocessor", preprocessor),
    ("selector", VarianceThreshold()),
    ("classifier", LogisticRegression(random_state=42))
])
rf_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("selector", VarianceThreshold()),
    ("classifier", RandomForestClassifier(random_state=42))
])
gb_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("selector", VarianceThreshold()),
    ("classifier", HistGradientBoostingClassifier(random_state=42))
])

# Logistic Regression Grid Search
log_reg_grid = GridSearchCV(log_reg_pipeline, log_reg_params,
cv=5,scoring='roc_auc', n_jobs=-1,verbose=3)

# Random Forest Grid Search
rf_grid = GridSearchCV(rf_pipeline, rf_params, cv=5,
scoring='roc_auc', n_jobs=-1,verbose=3)

# GradientBoosting Grid Search
gb_grid = GridSearchCV(gb_pipeline, gb_params, cv=5,
scoring='roc_auc', n_jobs=-1,verbose=3)

%%time
log_reg_grid.fit(X_train, y_train)
joblib.dump(log_reg_grid.best_estimator_,
"models/log_reg_grid_auc_roc_w_new_feat.pkl")
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

```
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
```

```
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

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

```
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

```
https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
```

```
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

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

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
[CV 1/5] END classifier__C=0.001, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.731 total time=  18.8s
[CV 1/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.753 total time=  28.7s
[CV 4/5] END classifier__C=0.01, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.748 total time=  10.9s
[CV 5/5] END classifier__C=0.1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.755 total time=  45.9s
[CV 5/5] END classifier__C=1, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.757 total time= 1.6min
[CV 5/5] END classifier__C=10, classifier__max_iter=100,
classifier__solver=newton-cg;, score=0.750 total time= 4.4min
[CV 1/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.760 total time=  11.3s
[CV 2/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.769 total time=  11.0s
[CV 3/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.747 total time=  11.1s
[CV 4/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.756 total time=  11.3s
[CV 5/5] END classifier__C=100, classifier__max_iter=100,
classifier__solver=lbfgs;, score=0.754 total time=  11.4s
CPU times: user 2min 38s, sys: 4.71 s, total: 2min 43s
Wall time: 21min 21s
```

```
['log_reg_grid_auc_roc_w_new_feat.pkl']
```

```
%%time
rf_grid.fit(X_train, y_train)
joblib.dump(rf_grid.best_estimator_,
"models/rf_grid_auc_roc_w_new_feat.pkl")
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
CPU times: user 1min 7s, sys: 764 ms, total: 1min 8s
Wall time: 9min 29s
```

```
['rf_grid_auc_roc_w_new_feat.pkl']
```

```
%%time
gb_grid.fit(X_train, y_train)
```

```
joblib.dump(gb_grid.best_estimator_,
"models/gb_grid_auc_roc_w_new_feat.pkl")
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
CPU times: user 1min 15s, sys: 1.45 s, total: 1min 17s
Wall time: 16min 46s

['gb_grid_auc_roc_w_new_feat.pkl']
```

```
print("\n Best Logistic Regression Classifier Params:",
log_reg_grid.best_params_)
evaluate_pipeline(log_reg_grid.best_estimator_)
```

```
 Best Logistic Regression Classifier Params: {'classifier__C': 1,
'classifier__max_iter': 100, 'classifier__solver': 'newton-cg'}
Precision:      0.6092
Recall:         0.2248
AUC-ROC:        0.7650
Train Accuracy: 0.8081
Test Accuracy:  0.8037
Classification Report:
              precision    recall  f1-score   support

       False       0.82      0.96      0.89      9097
        True       0.61      0.22      0.33      2469

    accuracy                           0.80     11566
   macro avg       0.71      0.59      0.61     11566
weighted avg       0.78      0.80      0.77     11566

Confusion Matrix:
[[8741  356]
 [1914  555]]
```

```
print("\n Best Random Forest Classifier Params:",
rf_grid.best_params_)
evaluate_pipeline(rf_grid.best_estimator_)
```

```
 Best Random Forest Classifier Params: {'classifier__max_depth': None,
'classifier__min_samples_split': 4, 'classifier__n_estimators': 240}
Precision:      0.8223
Recall:         0.2155
AUC-ROC:        0.8503
Train Accuracy: 0.9974
Test Accuracy:  0.8226
Classification Report:
              precision    recall  f1-score   support

       False       0.82      0.99      0.90      9097
```

```
        True       0.82      0.22      0.34      2469

    accuracy                           0.82     11566
   macro avg       0.82      0.60      0.62     11566
weighted avg       0.82      0.82      0.78     11566

Confusion Matrix:
[[8982  115]
 [1937  532]]
```

```
print("\n Best GradientBoosting Params:", gb_grid.best_params_)
evaluate_pipeline(gb_grid.best_estimator_)
```

```
 Best GradientBoosting Params: {'classifier__learning_rate': 0.1,
'classifier__max_depth': None, 'classifier__min_samples_leaf': 20}
Precision:      0.7130
Recall:         0.4609
AUC-ROC:        0.8742
Train Accuracy: 0.8685
Test Accuracy:  0.8453
Classification Report:
              precision    recall  f1-score   support

       False       0.87      0.95      0.91      9097
        True       0.71      0.46      0.56      2469

    accuracy                           0.85     11566
   macro avg       0.79      0.71      0.73     11566
weighted avg       0.83      0.85      0.83     11566

Confusion Matrix:
[[8639  458]
 [1331 1138]]
```

I was able to bump up the best ROC AUC score to 87.42 with these new features. Since the HistGradientBoostingClassifier is the best at distinguishing between classes accross different thresholds, I will threshold tune only that. Now I will estimate the best probability threshold for classification using the validation data.

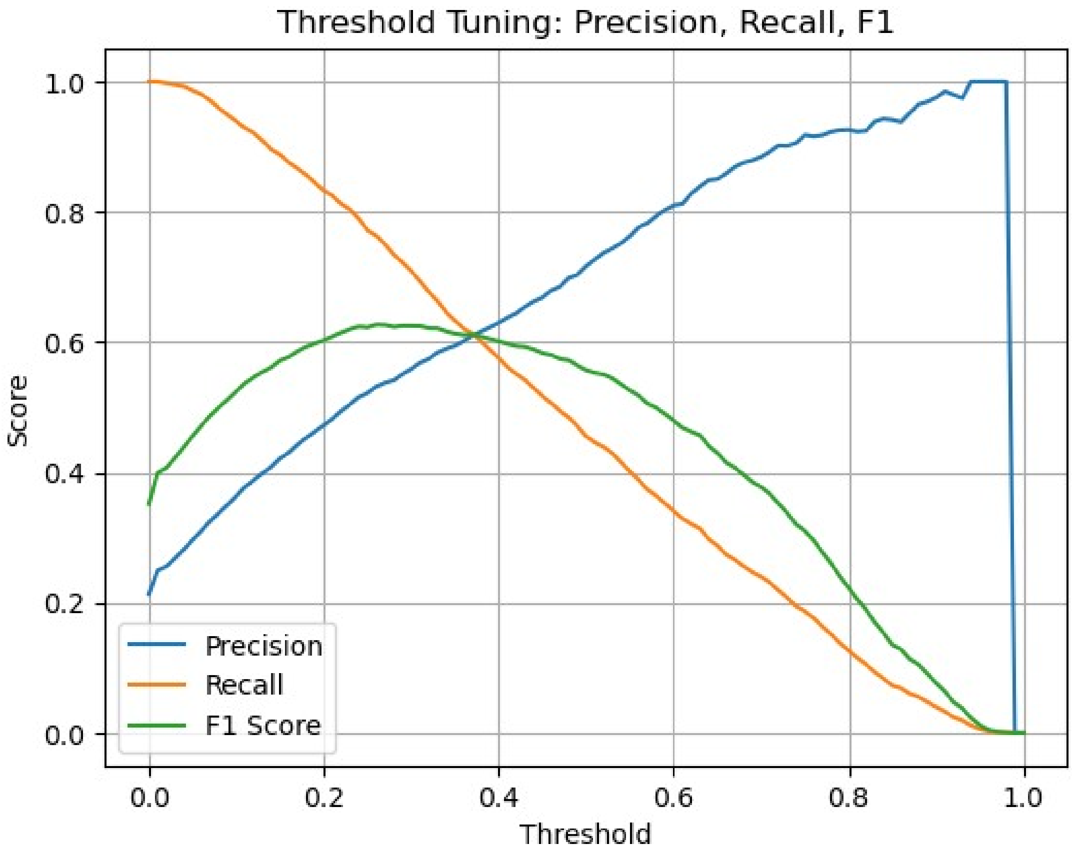
Best Model

```
best_model = joblib.load('models/gb_grid_auc_roc_w_new_feat.pkl')

y_proba = best_model.predict_proba(X_val)[:, 1]  # Probability of the
positive class (fraud)

import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score
```

```python
thresholds = np.linspace(0, 1, 101)  # thresholds from 0.00 to 1.00
results = []

for t in thresholds:
    y_pred = (y_proba >= t).astype(int)
    precision = precision_score(y_val, y_pred)
    recall = recall_score(y_val, y_pred)
    f1 = f1_score(y_val, y_pred)
    results.append((t, precision, recall, f1))
```

```
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
metrics/_classification.py:1344: 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))
/share/apps/anaconda3/2024.02/lib/python3.11/site-packages/sklearn/
metrics/_classification.py:1344: 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))
```

```python
import matplotlib.pyplot as plt

results = np.array(results)
plt.plot(results[:, 0], results[:, 1], label='Precision')
plt.plot(results[:, 0], results[:, 2], label='Recall')
plt.plot(results[:, 0], results[:, 3], label='F1 Score')

plt.xlabel("Threshold")
plt.ylabel("Score")
plt.title("Threshold Tuning: Precision, Recall, F1")
plt.legend()
plt.grid(True)
plt.show()
```




```
best_f1_idx = results[:, 3].argmax()
best_threshold = results[best_f1_idx, 0]
print(f"Best threshold for F1: {best_threshold}")

Best threshold for F1: 0.26
```

Now that I have the ideal threshold for classification as Fraud, I will use this threshold for getting the metrics for the test data.

```
y_proba = best_model.predict_proba(X_test)[:, 1]
final_preds = (y_proba >= best_threshold).astype(int)

precision = precision_score(y_test, final_preds)
recall = recall_score(y_test, final_preds)
f1 = f1_score(y_test, final_preds)
print(f"Final Precision: {precision}")
print(f"Final Recall:    {recall}")
print(f"Final f1-score:  {f1}")

Final Precision: 0.5285754904748365
Final Recall:    0.7529364115026327
Final f1-score:  0.6211159371867692
```

```
nonFraud = len(np.where(final_preds==0)[0])
fraud = len(np.where(final_preds==1)[0])
total = len(final_preds)
print(f"nonFraud preds: {nonFraud}")
print(f"Fraud preds: {fraud}")
print(f"Total preds: {total}")
nonFraudFrac = nonFraud/total
fraudFrac = fraud/total
print(f"Fraction of nonFraud preds: {nonFraudFrac}")
print(f"Fraction of Fraud preds: {fraudFrac}")
false_positives = np.where((final_preds == 1) & (y_test == 0))[0]
fp_count = len(false_positives)
print(f"Number of false positives: {fp_count}")
print(f"False positive rate: {fp_count/len(y_test):.4f}")
```

```
nonFraud preds: 8049
Fraud preds: 3517
Total preds: 11566
Fraction of nonFraud preds: 0.6959190731454262
Fraction of Fraud preds: 0.3040809268545738
Number of false positives: 1658
False positive rate: 0.1434
```

```
y_test.value_counts()
```

```
isFraud
False    9097
True     2469
Name: count, dtype: int64
```

The model now has a balanced performance with a f1 score of 0.62. With a recall of 0.75, 3 out of 4 fraudulent transactions will be identified correctly. With a precision of 0.52, there will still be some false positives. However, as visible from the test data above, False Positive Rate is 14.34%. This is a reasonable trade-off since the costs of fraud transactions is much higher than non fraud transactions.

Feature Importance

Since the HistGradientBoostingClassifier implementation of sklearn does not output feature importance, I attempted to explain the importances through permutation_importance. Source: https://github.com/scikit-learn/scikit-learn/issues/15132

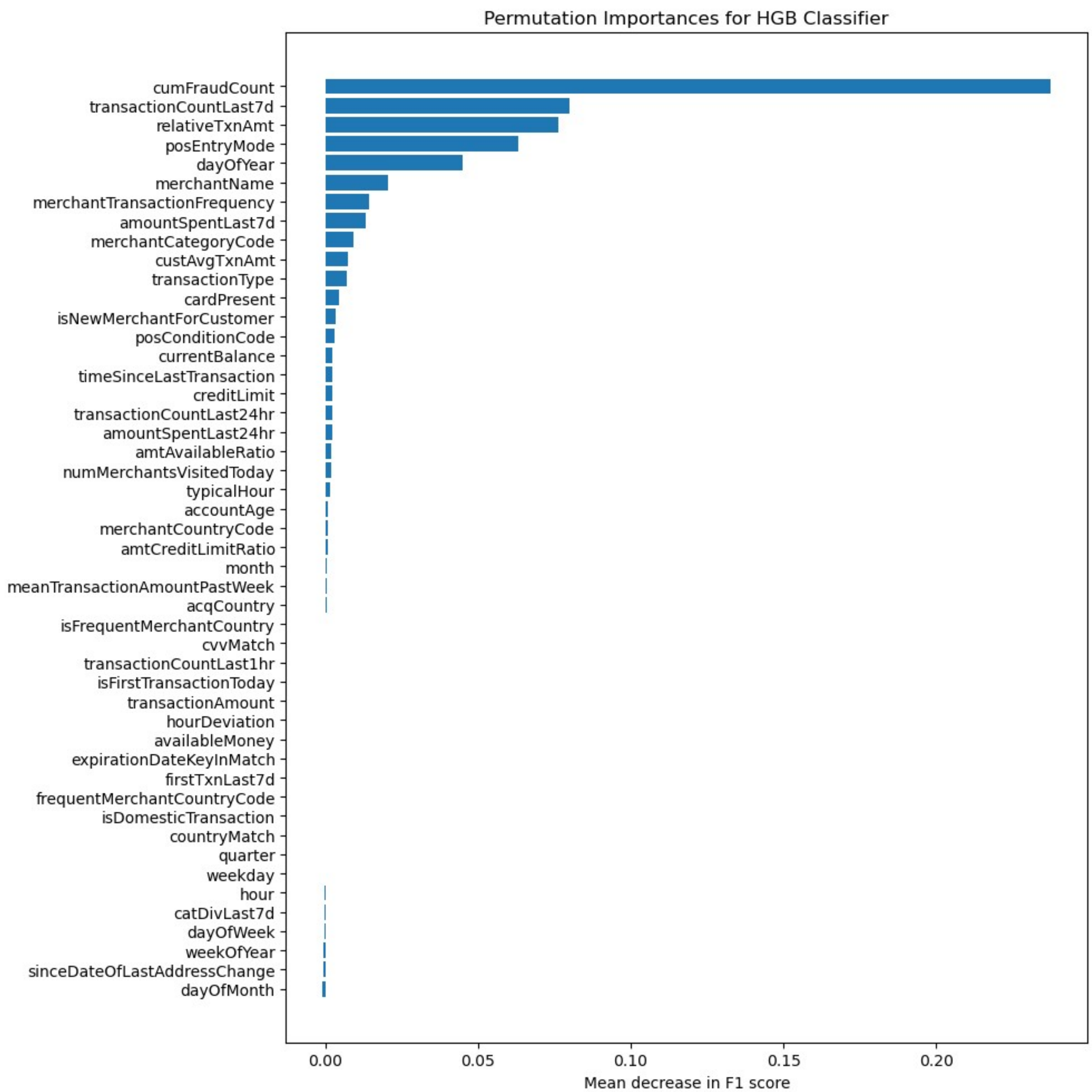
I also wanted to use my fine tuned threshold while evaluating the feature importance, so I had to implement a custom wrapper which will use my threshold, since permutation_importance internally uses model.predict with a 0.5 threshold without an option to customzie the threshold.

```
from sklearn.base import BaseEstimator, ClassifierMixin
```

```python
class ThresholdWrapper(BaseEstimator, ClassifierMixin):
    def __init__(self, model, threshold=0.26):
        self.model = model
        self.threshold = threshold

    def fit(self, X, y):
        self.model.fit(X, y)
        return self

    def predict(self, X):
        proba = self.model.predict_proba(X)[:, 1]
        return (proba >= self.threshold).astype(int)

    def predict_proba(self, X):
        return self.model.predict_proba(X)

    def score(self, X, y):
        from sklearn.metrics import f1_score
        return f1_score(y, self.predict(X))

from sklearn.inspection import permutation_importance

wrapped_model = ThresholdWrapper(model=best_model, threshold=0.26)

result = permutation_importance(
    wrapped_model, X_test, y_test,
    scoring='f1',
    n_repeats=30,
    random_state=42,
    n_jobs=-1
)

importances_mean = result.importances_mean
feature_names    = X_test.columns
sorted_idx       = np.argsort(importances_mean)[::-1]

print("Feature permutation importances (sorted):")
for idx in sorted_idx:
    print(f"{feature_names[idx]:<20}  {importances_mean[idx]:.4f}")
```

```
Feature permutation importances (sorted):
cumFraudCount         0.2375
transactionCountLast7d  0.0800
relativeTxnAmt        0.0762
posEntryMode          0.0632
dayOfYear             0.0449
merchantName          0.0207
merchantTransactionFrequency  0.0146
amountSpentLast7d     0.0131
merchantCategoryCode  0.0092
custAvgTxnAmt         0.0075
```

```
transactionType        0.0069
cardPresent            0.0046
isNewMerchantForCustomer  0.0034
posConditionCode       0.0031
currentBalance         0.0025
timeSinceLastTransaction  0.0024
creditLimit            0.0023
transactionCountLast24hr  0.0022
amountSpentLast24hr    0.0022
amtAvailableRatio      0.0020
numMerchantsVisitedToday  0.0019
typicalHour            0.0016
accountAge             0.0008
merchantCountryCode    0.0008
amtCreditLimitRatio    0.0008
month                  0.0005
meanTransactionAmountPastWeek  0.0004
acqCountry             0.0004
isFrequentMerchantCountry  0.0002
cvvMatch               0.0001
transactionCountLast1hr  0.0001
isFirstTransactionToday  0.0001
transactionAmount      0.0000
hourDeviation          0.0000
availableMoney         0.0000
expirationDateKeyInMatch  0.0000
firstTxnLast7d         0.0000
frequentMerchantCountryCode  0.0000
isDomesticTransaction  0.0000
countryMatch           0.0000
quarter                0.0000
weekday                0.0000
hour                   -0.0001
catDivLast7d           -0.0002
dayOfWeek              -0.0003
weekOfYear             -0.0004
sinceDateOfLastAddressChange  -0.0005
dayOfMonth             -0.0009

plt.figure(figsize=(10,10))
plt.barh(
    feature_names[sorted_idx],
    importances_mean[sorted_idx],
    align='center'
)
plt.xlabel("Mean decrease in F1 score")
plt.title("Permutation Importances for HGB Classifier")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



These feature importances do make logical sense to me. The model values features that capture recent user behavior and account history more than static transaction details isolated to that transaction.

- cumFraudCount stands out by a wide margin — it's the single most predictive feature. This makes sense: accounts with a history of fraud are much more likely to have frauds again, or may be compromised long-term. This could be inherently susceptible customers who repeatedly fall victim to fraud, like senior citizens who are taken advantage of.
- transactionCountLast7d and relativeTxnAmt also rank high, indicating that recent activity spikes and irregular transaction amounts relative to a customer's norm are strong fraud indicators. These features help capture behavioral drift and sudden changes in spending habits.

- posEntryMode and dayOfYear suggest that how and when a transaction occurs matters — certain entry methods (like manually keyed cards) and specific timeframes may carry higher risk.
- Merchant metadata still contribute, but not as much as other features. Some merchant categories (like electronics or luxury goods) could be more commonly targeted in fraud due to higher transaction value.

Answer 4

Data Cleaning & Feature Engineering

I started by removing empty, redundant, and duplicate columns, then engineered a set of behavioral and contextual features. My final features included:

- Match checks (e.g., whether the entered CVV matches the real one, or if the acquiring country matches the merchant's).
- Account-related timing, like days since account opening or last address change.
- Time-based behavior, such as part of day, day of week, hour, and whether the transaction was the first of the day.
- Rolling transaction activity, like transaction counts and total spend over 1-hour, 24-hour, and 7-day windows.
- Merchant patterns, including whether the customer had seen the merchant before, how frequently a merchant appears across the dataset, and how diverse the customer's spending categories were in the past week.
- Behavioral baselines, such as the customer's typical transaction hour, average spend, and deviation from these norms.
- Cumulative fraud history per customer.

These features were designed to capture both short-term spikes in behavior and longer-term patterns, which are often strong indicators in fraud detection.

Sampling Strategy Given the heavy class imbalance in fraud data, I sampled:

- All fraud transactions for each customer who has experienced fraud.
- A 7.5% sample of non-fraud transactions from those same customers, to retain relevant context.
- A 1% sample of non-fraud transactions from customers who have never experienced fraud, to introduce noise and avoid overfitting to a fraud-heavy subset.

This was selected after some trial and error with the 3 models

Modeling Approach

I split the data into train, validation, and test sets (60/20/20). Features were categorized into numeric, ordinal, and categorical, and transformed using a sklearn ColumnTransformer.

I experimented with three models: Logistic Regression, Random Forest, and HistGradientBoostingClassifier.

- Logistic Regression
- Random Forest
- HistGradientBoostingClassifier (HGB)

Logistic Regression was a good starting point because it's simple and easy to interpret. Random Forest helped capture more complex patterns in the data, but was ultimately overfitting. In the end, HistGradientBoostingClassifier worked best — it was fast and was great at picking up subtle fraud signals.

Each pipeline included a VarianceThreshold feature selector, and I ran grid search to optimize ROC AUC. I chose to optimize for ROC AUC because it gives a balanced view of how well the model separates fraud from non-fraud across all classification thresholds. This allowed me to later fine-tune the decision threshold specifically for F1 score, depending on the trade-off between false positives and false negatives.

Threshold Tuning

Since fraud is rare, raw predicted probabilities are usually low, and using a default 0.5 threshold would miss most fraud cases. After selecting the best model (HistGradientBoostingClassifier with a ROC AUC of 87.42), I fine-tuned the threshold using the validation set by sweeping values from 0.0 to 1.0, selecting the threshold that maximized F1 score (to balance precision and recall). This threshold came out to be 0.26, showing that the model is not very confident about fraud classifications simply because of the sheer non fraud samples.

Final Metrics on Test Set

With the selected threshold, the final performance on the test set was:

Precision: 0.53 Recall: 0.75 F1-score: 0.62 FPR: 0.14 The model effectively identifies most fraudulent cases, while keeping false positives at a manageable level.

Feature Importance

Since HistGradientBoostingClassifier doesn't expose importances directly, I used permutation importance and implemented a custom model wrapper to ensure it used my fine-tuned threshold (since the default is 0.5).

Key findings:

- cumFraudCount was the most predictive feature, showing that prior fraud involvement is a strong signal — likely reflecting either high-risk customers or compromised accounts.
- transactionCountLast7d and relativeTxnAmt were also highly ranked, indicating that sudden changes in activity or abnormal spend levels are common red flags.
- posEntryMode and dayOfYear show that how and when a transaction happens can influence risk — e.g., manually entered cards and certain periods may be riskier.
- Merchant metadata like category and frequency contributed to the model but played a supporting role, providing additional context rather than being primary features.

With more time, I would explore a larger hyperparameter grid. I kept the current grid fairly limited due to the computational cost of grid search, but a broader search could help fine-tune the model further.

I’d also be interested in experimenting with attention-based models, which might be better at capturing complex transaction patterns and sequential behavior, especially in customer histories.

Additionally, I’d look into better understanding the nature of false positives and whether certain types of fraud require separate treatment. If there was more context around why a transaction was fraud, it might be useful to be able to incorporate the fraud type.

What didn't work:

At first, I tried sampling the data so that it was perfectly balanced between fraud and non-fraud transactions, by undersampling nonfraud cases to be equal to fraud. The idea was to give the model an equal chance to learn both classes. But in reality, it led to really poor precision. The model started flagging way too many legitimate transactions as fraud.

That kind of tradeoff wouldn’t work in a real banking environment, where false alarms would frustrate customers and overwhelm fraud teams. So I moved away from strict balancing and instead went with a sampling approach that gave the model enough fraud to learn from, while keeping the number of false positives more reasonable.