modeling-marketing-offers

August 21, 2025

[ ]:
```
#!pip install xgboost
#!pip install scipy
#!pip install Jinja2
#!pip install category-encoders
!pip install Boruta
```

```
Collecting Boruta
  Downloading Boruta-0.4.3-py3-none-any.whl.metadata (8.8 kB)
Requirement already satisfied: numpy>=1.10.4 in
/databricks/python3/lib/python3.12/site-packages (from Boruta) (1.26.4)
Requirement already satisfied: scikit-learn>=0.17.1 in /local_disk0/.ephemeral_n
fs/envs/pythonEnv-2454007c-f378-419f-abbe-746737c3a54b/lib/python3.12/site-
packages (from Boruta) (1.7.1)
Requirement already satisfied: scipy>=0.17.0 in
/databricks/python3/lib/python3.12/site-packages (from Boruta) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in
/databricks/python3/lib/python3.12/site-packages (from scikit-
learn>=0.17.1->Boruta) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /local_disk0/.ephemeral_n
fs/envs/pythonEnv-2454007c-f378-419f-abbe-746737c3a54b/lib/python3.12/site-
packages (from scikit-learn>=0.17.1->Boruta) (3.6.0)
Downloading Boruta-0.4.3-py3-none-any.whl (57 kB)
Installing collected packages: Boruta
Successfully installed Boruta-0.4.3
Note: you may need to restart the kernel using %restart_python ordbutils.library.restartPython
```

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import glob
from xgboost import XGBClassifier
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score,␣
 ↪average_precision_score, f1_score, roc_auc_score, roc_curve,␣
 ↪precision_recall_curve, matthews_corrcoef
from scipy.stats import mannwhitneyu
```

```python
import category_encoders as ce
from boruta import BorutaPy
```

Modeling logic

At first glance, the task appears to be about creating a dataset and training a predictive model such that, given a user profile and the characteristics of an offer, the model can estimate the probability that a transaction will occur. In other words, we want to model:

$$P(Transaction \quad User\ Profile, Offer)$$

However, this is not sufficient for understanding the **true effectiveness of offers**. What we really care about is the **causal impact** of giving an offer, i.e., whether the offer actually changes user behavior compared to what they would have done without the offer.

This brings us to the concept of **uplift modeling**. Instead of just predicting the probability of a transaction under treatment (receiving the offer), we want to measure the **difference** between:

1. The probability that a transaction occurs if the user receives the offer.

2. The probability that a transaction occurs if the same user does **not** receive the offer.

The second probability depends only on the user's intrinsic profile and past behavior (their "baseline" propensity to transact).

Formally, the **uplift** can be written as:

$$\mathbf{Uplift(x)=P(Transaction\ x,Offer)\ -\ P(Transaction\ x,No\ Offer)}$$

Thus, the problem is not just predictive, but **causal**: we want to isolate the incremental effect of the offer on transactions. A positive uplift indicates that the offer increases the likelihood of a transaction, while a negative uplift means the offer may actually discourage transactions or simply attract users who would have transacted anyway (cannibalization).

Here I first start with some data analysis.

Plan: - Data Analysis for both datasets: **labeled_data_with_offer.csv** and **user_profiles_transaction_classes.csv** - Feature engineering if needed - Treating missing values - Categorized to numerical data convertion - Data split and modeling (take care of data leakage if any) - Metric choice (Recall, precision, AUC, AVG.precision) - scale_pos_weight setting if needed - Important feature analysis using Boruta

[ ]:
```python
## user_transaction_profile_df: this

user_files = glob.glob("/Volumes/workspace/default/data/
 ↪user_profiles_transaction_classes.csv/part-*.csv")
user_transaction_profile_df = pd.concat((pd.read_csv(f) for f in user_files),␣
 ↪ignore_index=True)

offer_files = glob.glob("/Volumes/workspace/default/data/
 ↪labeled_data_with_offer.csv/part-*.csv")
offer_related_transaction_profile_df = pd.concat((pd.read_csv(f) for f in␣
 ↪offer_files), ignore_index=True)
```

```
user_transaction_profile_df.to_csv("data/user_profiles_transaction_classes.csv")
offer_related_transaction_profile_df.to_csv("data/
↪offer_user_profiles_transaction_classes.csv")
```

0.1 User profile <> Transaction analysis

[ ]:
```
user_transaction_profile_df
```

[ ]:
```
                             account_id  age  …  registered_on class
0      2d49e5a5886c4b9fb82c62a420dd2e85   72  …       20180709     0
1      382199dc87f34bb2b01b2d3deea9d9b3   74  …       20171112     0
2      882a3db453b941f98e91fbac42d39b72  118  …       20170728     0
3      aee8cae3f1a345128f3b3612e4d529dd   73  …       20171225     0
4      86044a1798d646e18b43cd813f7a79c5   69  …       20170828     0
…                                   …  …  …            …   …
16683  281d18c6603f43beb05270eb41d8c2f0   56  …       20180502     1
16684  2a070c1a63e348fda6f9772df48f4c85   73  …       20171104     1
16685  a7a0d8c4d2644519bd8f5dffcb7a7efb   35  …       20171213     1
16686  4202baa282014408a665fcbb58941620   83  …       20170422     1
16687  8a504d8980764110bd0a6ce89213e097   77  …       20151224     1

[16688 rows x 6 columns]
```

[ ]:
```
offer_related_transaction_profile_df.shape
```

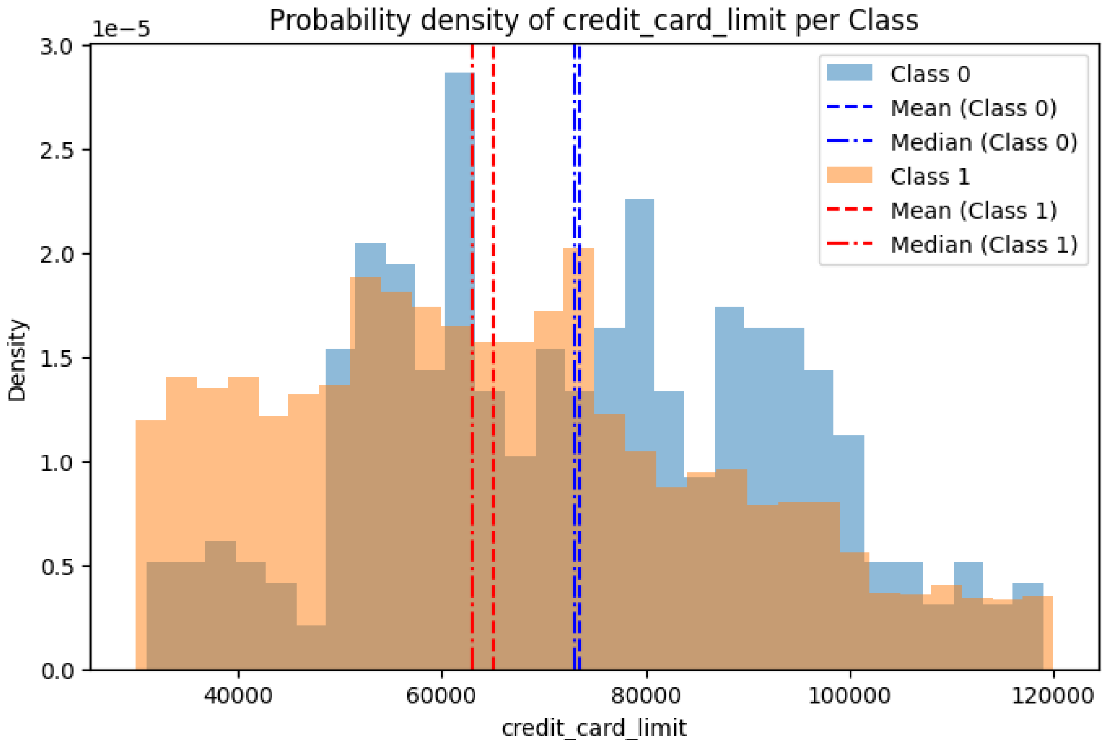
[ ]:
```
(63288, 12)
```

[ ]:
```
counts = (
    user_transaction_profile_df
    .groupby(["gender", "class"])
    .size()
    .reset_index(name="count")
)

counts["proportion"] = counts.groupby("gender")["count"].transform(lambda x: x /
 ↪ x.sum())

print(counts)
```

```
  gender  class  count  proportion
0      F      0    136    0.022629
1      F      1   5874    0.977371
2      M      0    189    0.022618
3      M      1   8167    0.977382
4      O      0      8    0.038462
5      O      1    200    0.961538
```

[ ]:
```python
def plot_distribution_per_class(df, feature, class_name):
    plt.figure(figsize=(8,5))

    for c in df[class_name].unique():
        subset = df[df[class_name] == c]
        plt.hist(
            subset[feature].dropna(),
            bins=30,
            density=True,
            alpha=0.5,
            label=f"Class {c}"
        )

        mean_val = subset[feature].mean()
        plt.axvline(mean_val, color="blue" if c==0 else "red", linestyle="--",␣
 ↪linewidth=1.5, label=f"Mean (Class {c})")

        median_val = subset[feature].median()
        plt.axvline(median_val, color="blue" if c==0 else "red", linestyle="-.
 ↪", linewidth=1.5, label=f"Median (Class {c})")

    plt.xlabel(feature)
    plt.ylabel("Density")
    plt.title(f"Probability density of {feature} per Class")
    plt.legend()
    plt.show()
```

[ ]:
```python
plot_distribution_per_class(user_transaction_profile_df, "credit_card_limit", ␣
 ↪"class")
```



[ ]:
```
group0 = user_transaction_profile_df.loc[user_transaction_profile_df["class"]␣
    ↪== 0, "credit_card_limit"].dropna()
group1 = user_transaction_profile_df.loc[user_transaction_profile_df["class"]␣
    ↪== 1, "credit_card_limit"].dropna()

print("Group0 size:", len(group0))
print("Group1 size:", len(group1))
print("Group0 unique values:", group0.unique())
print("Group1 unique values:", group1.unique())
print("Group0 dtypes:", group0.dtype)
print("Group1 dtypes:", group1.dtype)
print("Group0 has inf:", np.isinf(group0).any())
print("Group1 has inf:", np.isinf(group1).any())
```

```
Group0 size: 333
Group1 size: 14241
Group0 unique values: [101000.  89000.  92000.  75000.  70000.  73000.  74000.
63000.  61000.
  51000.  88000.  52000. 107000.  82000.  91000.  94000.  57000.  39000.
  59000.  84000.  47000.  79000.  54000.  65000.  86000. 119000. 108000.
  93000. 115000.  98000.  78000.  56000.  64000.  62000.  71000.  53000.
  44000. 118000.  37000.  69000.  60000.  55000.  68000.  95000. 106000.
  80000.  66000. 103000. 111000.  97000.  96000.  49000.  31000.  48000.
```

```
  87000. 100000.  81000.  67000.  35000.  72000.  40000.  58000. 113000.
 114000. 110000.  77000.  36000.  50000.  99000.  83000.  90000.  41000.
  38000.  85000. 112000.  76000.  32000.  42000.  43000. 116000.  34000.
  45000.]
Group1 unique values: [ 34000.  32000.  72000.  89000.  61000.  57000.  52000.
59000.  94000.
  36000. 115000.  38000.  64000.  35000.  95000.  68000.  69000.  58000.
  55000.  30000.  82000.  40000.  44000.  50000.  62000.  84000.  77000.
  75000. 111000.  67000.  70000.  73000.  54000. 117000.  51000. 116000.
  63000.  87000.  85000.  33000.  86000.  45000.  96000.  99000.  43000.
  65000.  79000.  49000.  31000.  60000.  83000.  42000.  56000.  41000.
  78000.  53000.  48000.  47000.  37000.  76000.  80000. 102000.  92000.
  39000.  71000.  88000.  74000.  66000. 114000. 100000.  81000.  93000.
 118000. 113000. 112000.  98000. 109000. 107000.  90000.  91000. 108000.
  97000.  46000. 105000. 120000. 106000. 119000. 110000. 101000. 103000.
 104000.]
Group0 dtypes: float64
Group1 dtypes: float64
Group0 has inf: False
Group1 has inf: False
```

–>This shows that the data are very likely manufactured data and are not real. Same values repeats over and over

[ ]:
```
result = mannwhitneyu(group0, group1, alternative="two-sided")
print(result.statistic, result.pvalue)
```

```
2931903.5 1.4712766087752866e-13
```

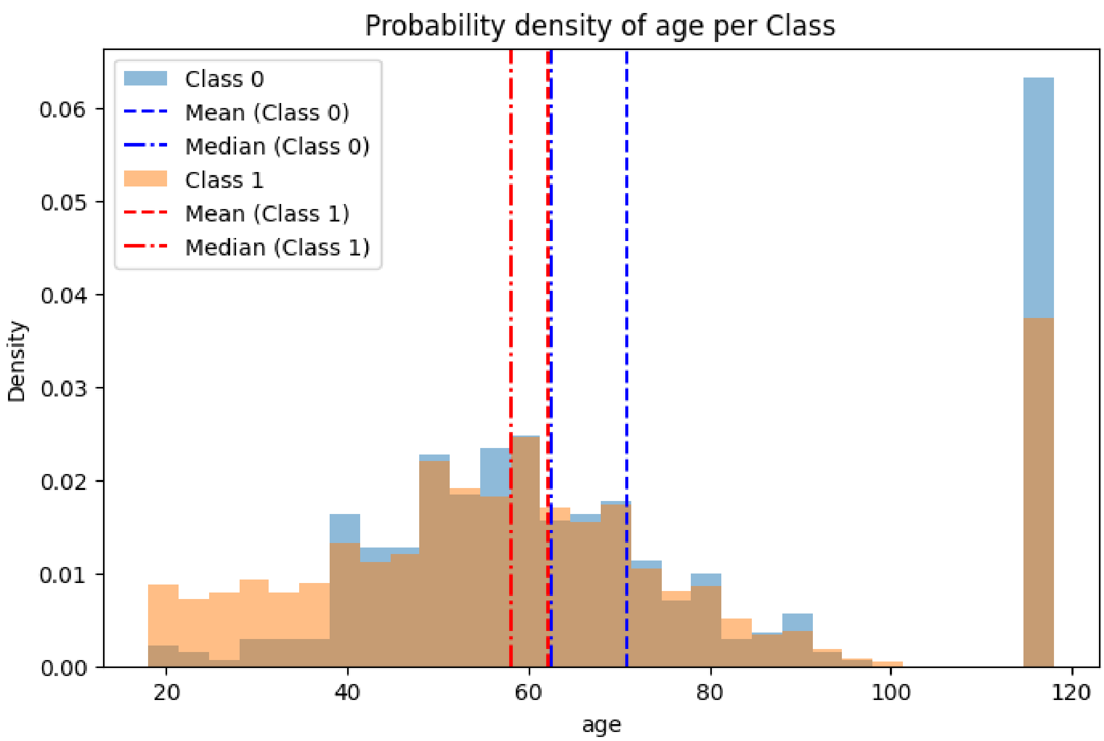
–> This p-value shows that the data rank for credit limit are infact different and statistically significant

[ ]:
```
group0 = user_transaction_profile_df.loc[user_transaction_profile_df["class"]␣
 ↪== 0, "age"].dropna()
group1 = user_transaction_profile_df.loc[user_transaction_profile_df["class"]␣
 ↪== 1, "age"].dropna()

print("Group0 size:", len(group0))
print("Group1 size:", len(group1))
print("Group0 unique values:", group0.unique())
print("Group1 unique values:", group1.unique())
print("Group0 dtypes:", group0.dtype)
print("Group1 dtypes:", group1.dtype)
print("Group0 has inf:", np.isinf(group0).any())
print("Group1 has inf:", np.isinf(group1).any())
```

```
Group0 size: 422
Group1 size: 16266
Group0 unique values: [ 72  74 118  73  69  56  38  77  51  45  42  64  61  57
```

```
70  53  47  50
  41  40  75  85  31  81  54  88  86  76  55  49  58  68  66  20  28  27
  44  43  78  97  79  65  39  71  80  91  52  60  48  59  36  63  62  46
  35  22  82  67  83  37  34  29  18  92  90  33  93  32  23]
Group1 unique values: [ 61  68  22  69  53  75  37  54  59  51 118  71  55  70
29  66  60  24
  77  36  31  78  64  49  80  74  45  65  57  43  58  39  88  81  52  41
  28  72  27  42  73  48  62  30  50  63  47  40  20  56  83  23  67  26
  21  82  38  18  87  98  35  33  95  92  44  93  79  25  94  76  32  86
  46  34  19 100  89  84  85  91  90  97  96  99 101]
Group0 dtypes: int64
Group1 dtypes: int64
Group0 has inf: False
Group1 has inf: False
```

[ ]:
```python
plot_distribution_per_class(user_transaction_profile_df, "age",  "class")
```



[ ]:
```python
result = mannwhitneyu(group0, group1, alternative="two-sided")
print(result.statistic, result.pvalue)
```

```
4011267.5 2.9559462589289866e-09
```

–> This p-value shows that the data rank for age are infact different and statistically significant

Note: These features are very abnormal. Very high credit limits, people with age of 120. In real world these values dont make sense

[ ]:
```
# Lets evaluate the variable correlation:
corr = user_transaction_profile_df.drop(["class", "registered_on"], axis = 1).
  ↪corr()
corr.style.background_gradient(cmap='coolwarm')
```

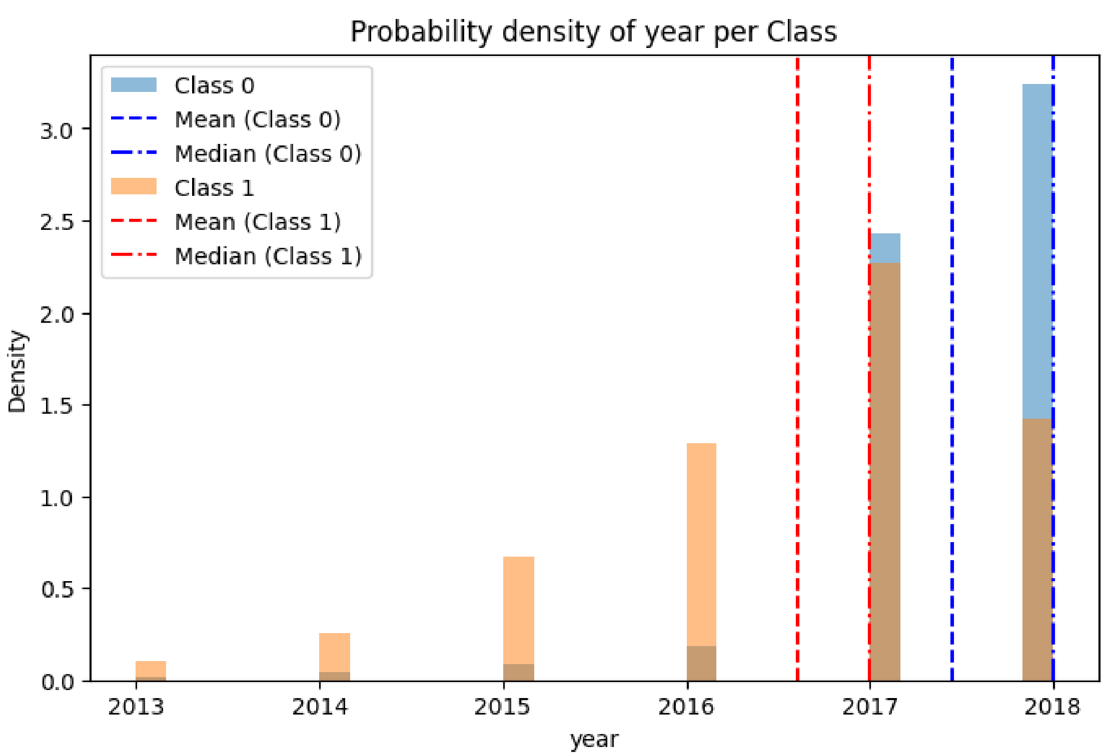
```
/home/spark-ff69a62a-7422-447b-9397-42/.ipykernel/2943/command-6391446602122633-
2730879993:2: FutureWarning: The default value of numeric_only in DataFrame.corr
is deprecated. In a future version, it will default to False. Select only valid
columns or specify the value of numeric_only to silence this warning.
  corr = user_transaction_profile_df.drop(["class", "registered_on"], axis =
1).corr()
```

[ ]: <pandas.io.formats.style.Styler at 0x7f637648fe90>

Lets analyze the registration years as well. We want to answer: do people who registered earlier have a higher probability of having a transaction?

[ ]:
```
user_transaction_profile_df["registered_on"] = pd.
  ↪to_datetime(user_transaction_profile_df["registered_on"], format="%Y%m%d")
user_transaction_profile_df["year"] =␣
  ↪user_transaction_profile_df["registered_on"].dt.year
```

[ ]:
```
plot_distribution_per_class(user_transaction_profile_df, "year", "class")
```



This is interesting because it shows that those erlier users are more faithful to the company than those who are newer and they buy more

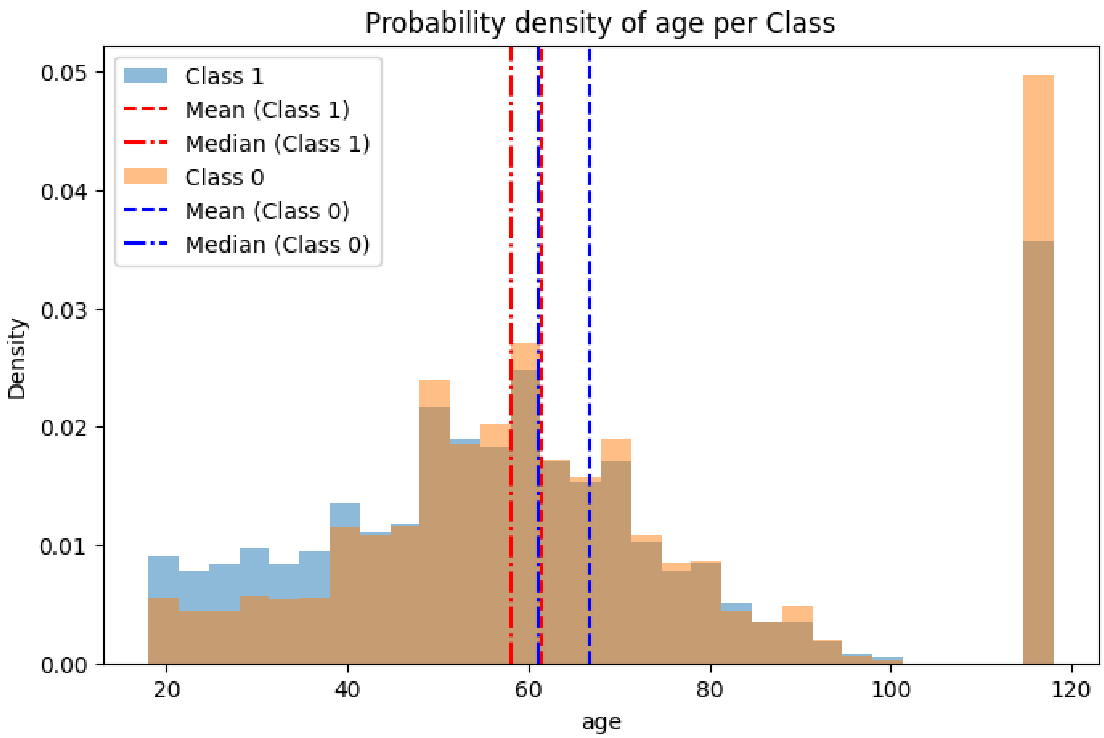
0.2 User profile <> Offer <> Transaction analysis

User profile analysis per class

```
[ ]: offer_related_transaction_profile_df.head()
```

```
[ ]:                              offer_id  ...     offer_type
     0  2298d6c36e964ae4a3e7e9706d1fb8c2  ...       discount
     1  0b1e1539f2cc45b7b9fa7c272da2e1d7  ...       discount
     2  fafdcd668e3743c1bb461111dcafc2a4  ...       discount
     3  3f207df678b143eea3cee63160fa8bed  ...  informational
     4  9b98b8c7a33c4b65b9aebfe6a799e6d9  ...           bogo

     [5 rows x 12 columns]
```

```
[ ]: plot_distribution_per_class(offer_related_transaction_profile_df, "age",␣
       ↪"offer_led_to_transaction")
```



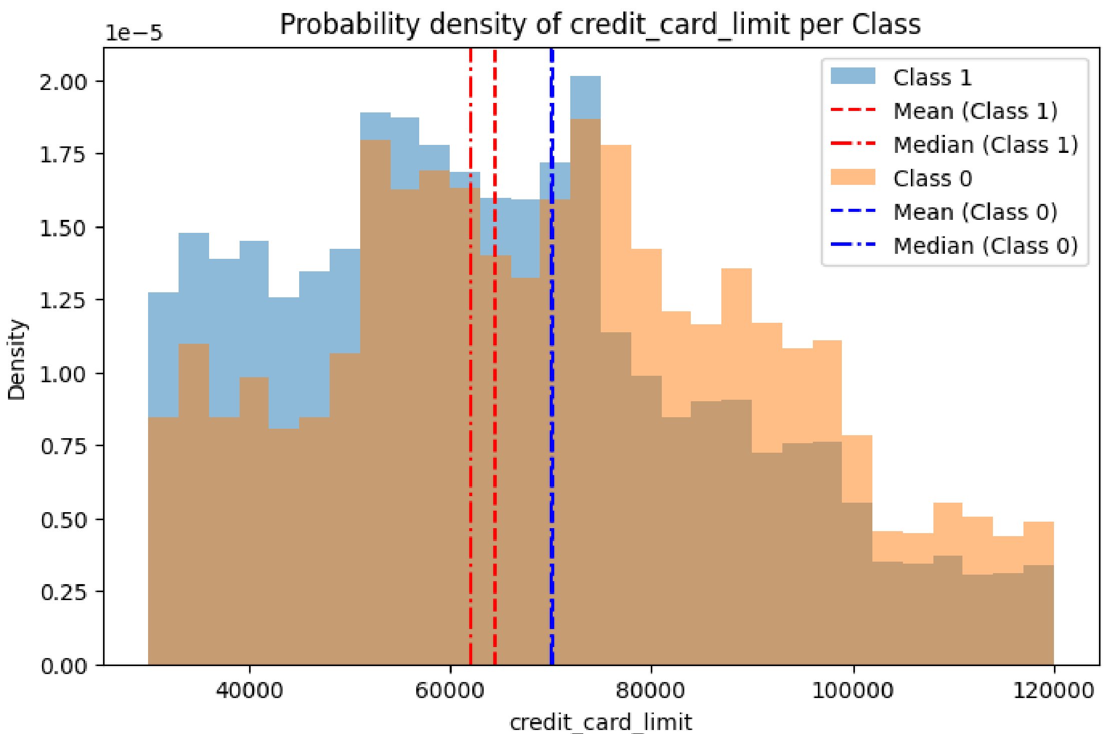
[ ]:
```
group0 = offer_related_transaction_profile_df.
 ↪loc[offer_related_transaction_profile_df["offer_led_to_transaction"] == 0,␣
 ↪"age"].dropna()
group1 = offer_related_transaction_profile_df.
 ↪loc[offer_related_transaction_profile_df["offer_led_to_transaction"] == 1,␣
 ↪"age"].dropna()

print("Group0 size:", len(group0))
print("Group1 size:", len(group1))
print("Group0 unique values:", group0.unique())
print("Group1 unique values:", group1.unique())
print("Group0 dtypes:", group0.dtype)
print("Group1 dtypes:", group1.dtype)
print("Group0 has inf:", np.isinf(group0).any())
print("Group1 has inf:", np.isinf(group1).any())
```

```
Group0 size: 11383
Group1 size: 51905
Group0 unique values: [ 55  61  75  53  59  68  51 118  54  24  31  77  36  78
80  64  49  39
  65  45  43  70  41  81  42  72  58  20  56  47  57  67  71  62  40  50
  22  82  60  98  87  48  33  28  63  52  95  34  44  37  79  76  29  73
  46  88  66 100  21  69  92  74  84  89  94  27  18  23  38  93  19  86
  85  25  90  32  26  35  91  83  30  97  99 101  96]
Group1 unique values: [ 69 118  68  37  54  59  22  51  53  55  71  61  70  29
36  66  77  24
  60  31  75  74  49  80  64  78  45  43  57  58  65  39  81  52  88  41
  28  27  72  42  30  73  48  62  50  63  40  47  20  56  83  23  67  26
  82  21  38  18  87  98  35  33  95  44  92  93  79  25  94  76  32  86
  46  34  19 100  89  84  85  91  90  97  96  99 101]
Group0 dtypes: int64
Group1 dtypes: int64
Group0 has inf: False
Group1 has inf: False
```

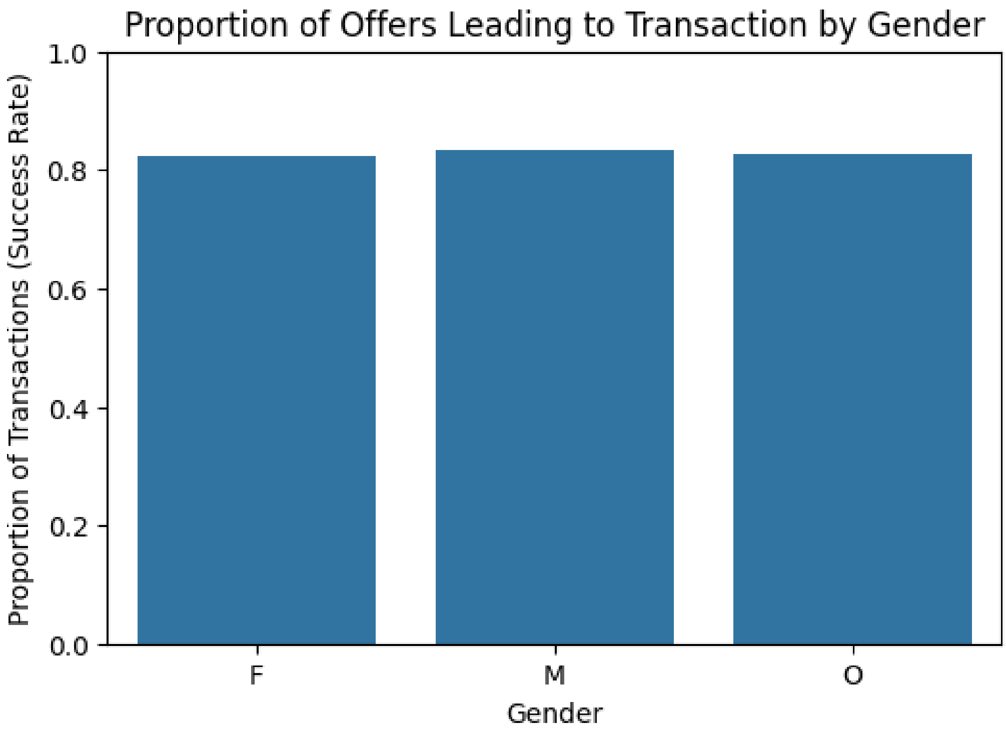
[ ]:
```
result = mannwhitneyu(group0, group1, alternative="two-sided")
print(result.statistic, result.pvalue)
```

```
328319419.0 1.0470080101360145e-77
```

[ ]:
```
plot_distribution_per_class(offer_related_transaction_profile_df,␣
 ↪"credit_card_limit", "offer_led_to_transaction")
```



[ ]:
```
# Here I want to know if there is a gender difference in the offer led to␣
↪transaction
gender_success_rate = (
    offer_related_transaction_profile_df
    .dropna(subset=["gender"])
    .groupby("gender")["offer_led_to_transaction"]
    .mean()
    .reset_index(name="success_rate")
)

plt.figure(figsize=(6,4))
sns.barplot(
    data=gender_success_rate,
    x="gender",
    y="success_rate"
)
plt.title("Proportion of Offers Leading to Transaction by Gender")
plt.xlabel("Gender")
plt.ylabel("Proportion of Transactions (Success Rate)")
plt.ylim(0,1)
plt.show()
```

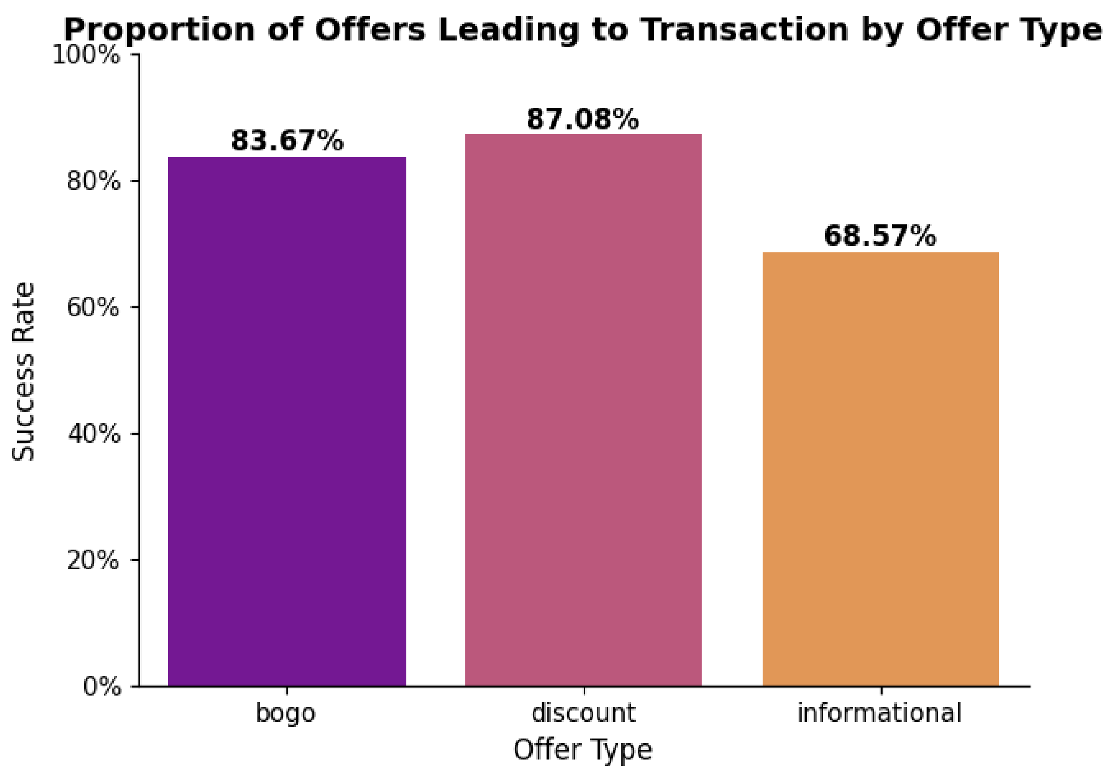


There is no gender difference in accepting or not accepting an offer. Both genders used the offers more than not using

[ ]:
```
offer_success_rate = (
    offer_related_transaction_profile_df
    .groupby("offer_type")["offer_led_to_transaction"]
    .mean()
    .reset_index(name="success_rate")
)

plt.figure(figsize=(7,5))
ax = sns.barplot(
    data=offer_success_rate,
    x="offer_type",
    y="success_rate",
    palette="plasma"
)

for p in ax.patches:
    ax.annotate(
        f"{p.get_height():.2%}",
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha="center", va="bottom",
```

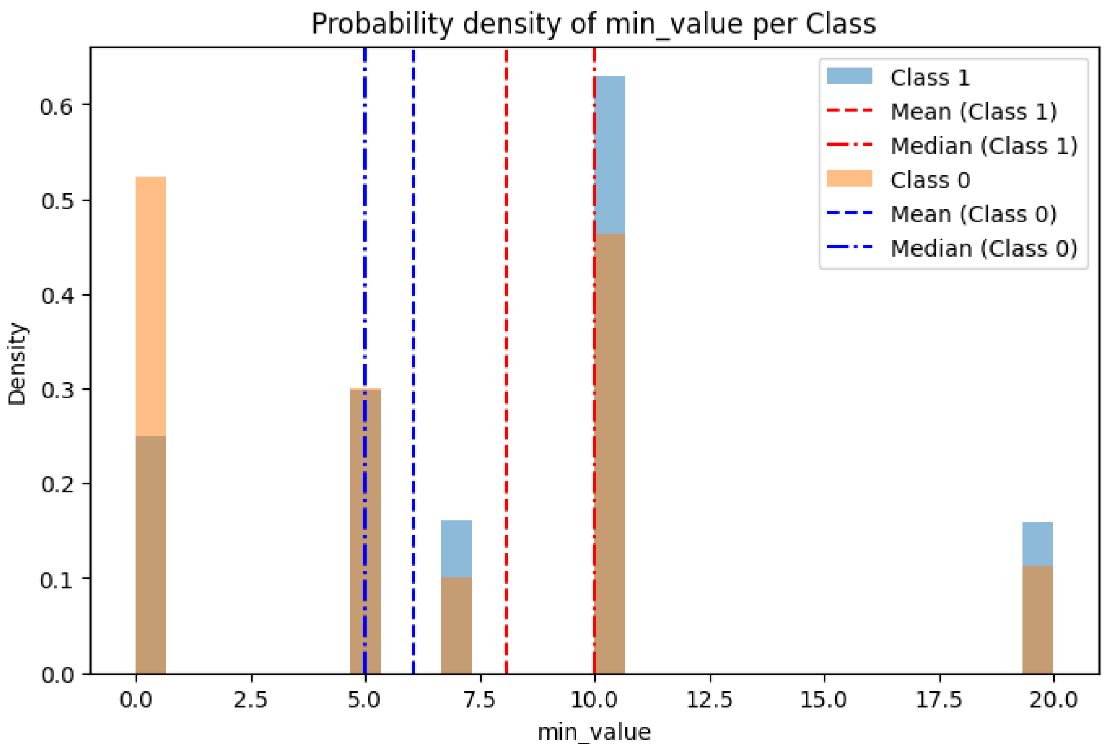
```
        fontsize=12, color="black", weight="bold"
    )

plt.title("Proportion of Offers Leading to Transaction by Offer Type",␣
 ↪fontsize=14, weight="bold")
plt.xlabel("Offer Type", fontsize=12)
plt.ylabel("Success Rate", fontsize=12)
plt.ylim(0, 1)
plt.xticks(fontsize=11)
plt.yticks(np.linspace(0,1,6), [f"{x:.0%}" for x in np.linspace(0,1,6)],␣
 ↪fontsize=11)  # show % on y-axis
sns.despine()

plt.show()
```

```
/home/spark-ff69a62a-7422-447b-9397-42/.ipykernel/2943/command-6391446602122645-
3255890933:9: 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.

  ax = sns.barplot(
```

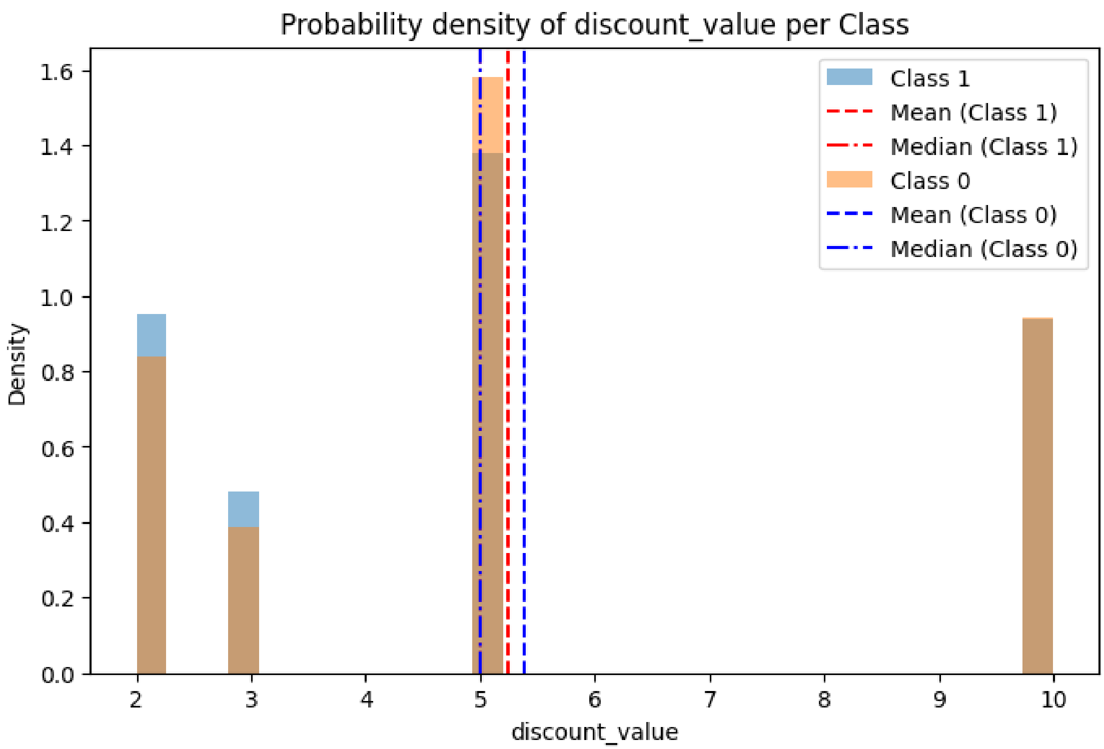


This is interesting. The offers which have a discount or those of Buy One, Get One (BOGO) has higher conversion rate to a transaction.

[ ]:
```
plot_distribution_per_class(offer_related_transaction_profile_df, "min_value",␣
  ↪"offer_led_to_transaction")
```



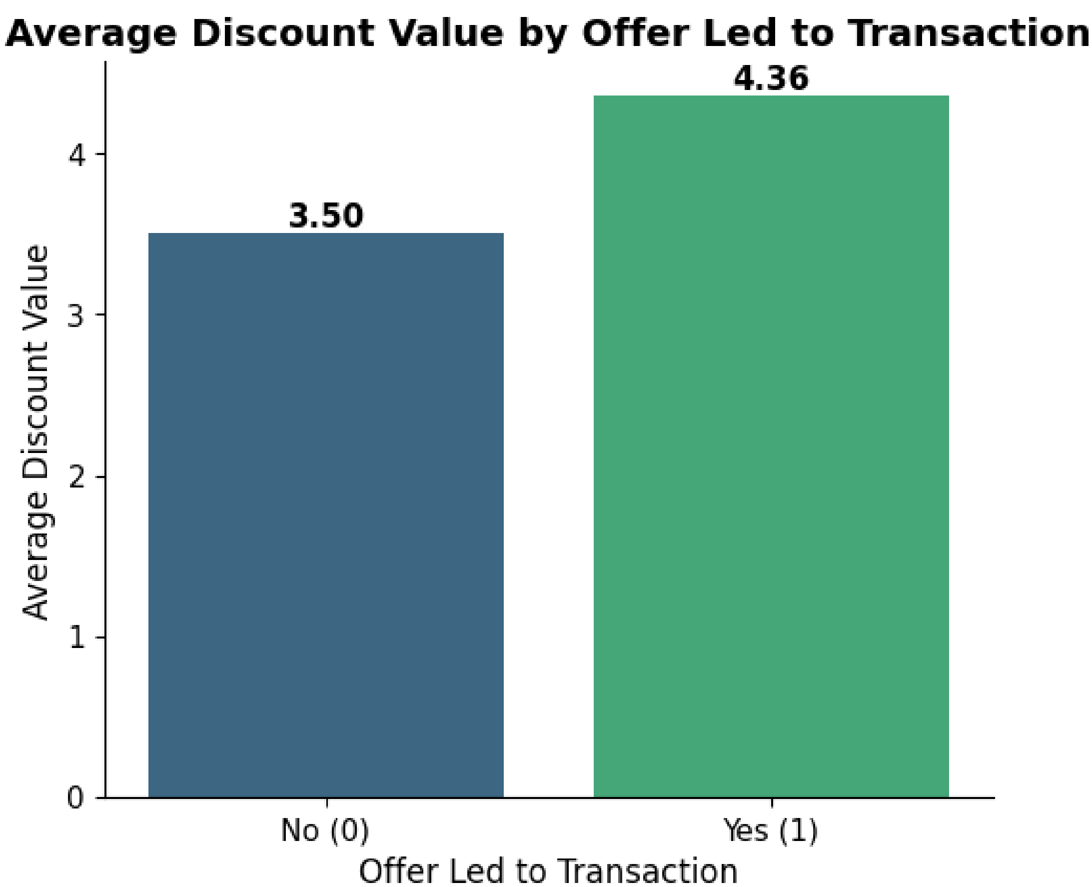
[ ]:
```
```



[ ]:
```
discount_means = (
    offer_related_transaction_profile_df
    .groupby("offer_led_to_transaction")["discount_value"]
    .mean()
    .reset_index()
)

plt.figure(figsize=(6,5))
ax = sns.barplot(
    data=discount_means,
    x="offer_led_to_transaction",
    y="discount_value",
    palette="viridis"
)

for p in ax.patches:
    ax.annotate(
        f"{p.get_height():.2f}",
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha="center", va="bottom",
        fontsize=12, color="black", weight="bold"
    )
```

```
plt.title("Average Discount Value by Offer Led to Transaction", fontsize=14,␣
    ↪weight="bold")
plt.xlabel("Offer Led to Transaction", fontsize=12)
plt.ylabel("Average Discount Value", fontsize=12)
plt.xticks([0,1], ["No (0)", "Yes (1)"], fontsize=11)
plt.yticks(fontsize=11)
sns.despine()

plt.show()
```

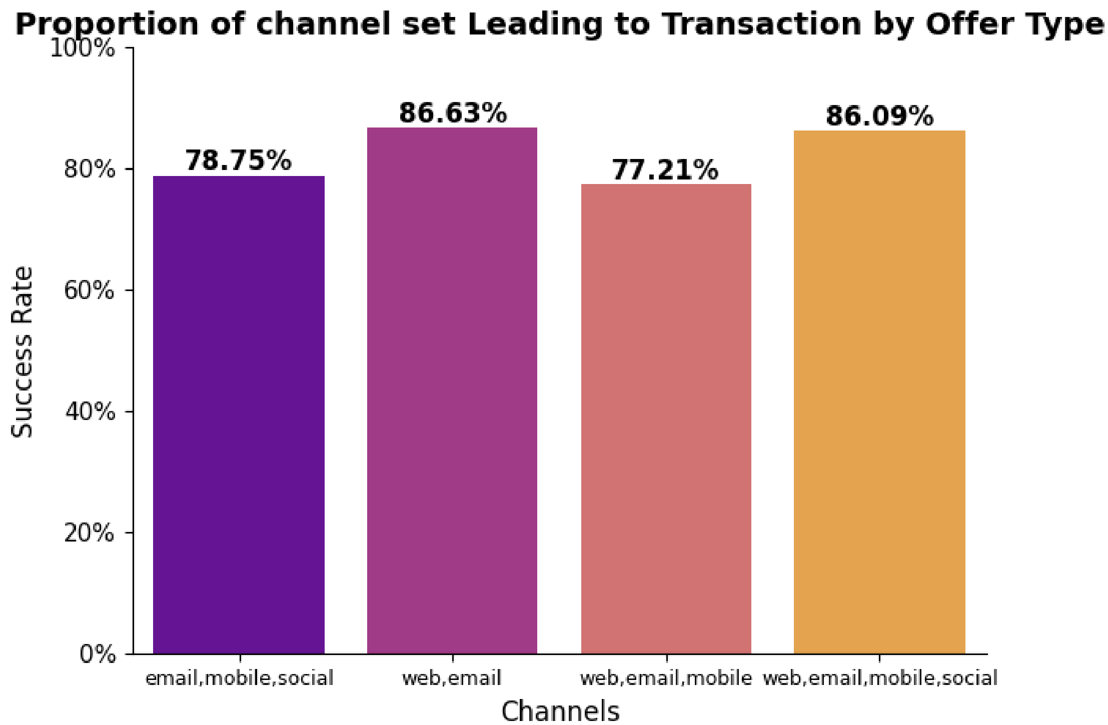
```
/home/spark-ff69a62a-7422-447b-9397-42/.ipykernel/2943/command-6391446602122650-
2626658147:9: FutureWarning:

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.

  ax = sns.barplot(
```



Interstingly, those offers which led to transaction have higher vdiscount values.

[ ]:
```python
offer_success_rate = (
    offer_related_transaction_profile_df
    .groupby("channels")["offer_led_to_transaction"]
    .mean()
    .reset_index(name="success_rate")
)

plt.figure(figsize=(7,5))
ax = sns.barplot(
    data=offer_success_rate,
    x="channels",
    y="success_rate",
    palette="plasma"
)

for p in ax.patches:
    ax.annotate(
        f"{p.get_height():.2%}",
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha="center", va="bottom",
        fontsize=12, color="black", weight="bold"
    )

plt.title("Proportion of channel set Leading to Transaction by Offer Type",␣
 ↪fontsize=14, weight="bold")
plt.xlabel("Channels", fontsize=12)
plt.ylabel("Success Rate", fontsize=12)
plt.ylim(0, 1)
plt.xticks(fontsize=9)
plt.yticks(np.linspace(0,1,6), [f"{x:.0%}" for x in np.linspace(0,1,6)],␣
 ↪fontsize=11)
sns.despine()

plt.show()
```

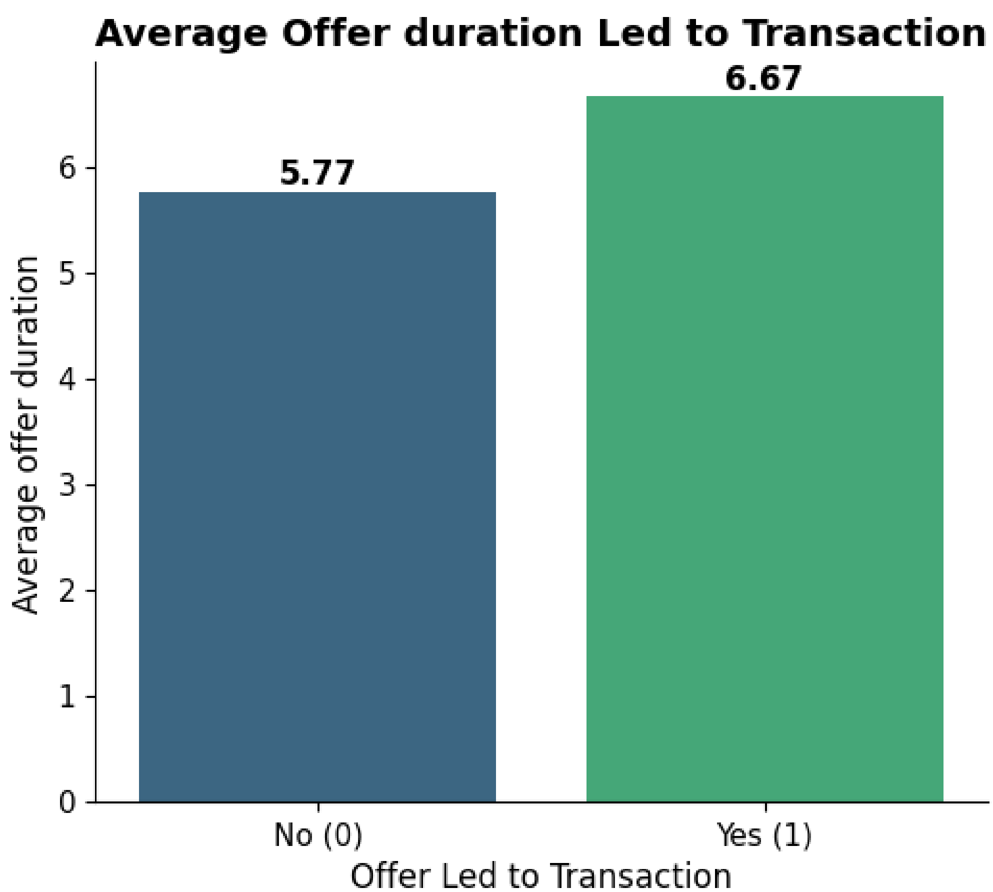
```
/home/spark-2454007c-f378-419f-abbe-74/.ipykernel/2941/command-4911024392514238-
2149953904:9: FutureWarning:

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.

  ax = sns.barplot(
```



[ ]:
```
discount_means = (
    offer_related_transaction_profile_df
    .groupby("offer_led_to_transaction")["duration"]
    .mean()
    .reset_index()
)

plt.figure(figsize=(6,5))
ax = sns.barplot(
    data=discount_means,
    x="offer_led_to_transaction",
    y="duration",
    palette="viridis"
)

for p in ax.patches:
    ax.annotate(
        f"{p.get_height():.2f}",
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha="center", va="bottom",
        fontsize=12, color="black", weight="bold"
    )
```

```
plt.title("Average Offer duration Led to Transaction", fontsize=14,␣
↪weight="bold")
plt.xlabel("Offer Led to Transaction", fontsize=12)
plt.ylabel("Average offer duration", fontsize=12)
plt.xticks([0,1], ["No (0)", "Yes (1)"], fontsize=11)
plt.yticks(fontsize=11)
sns.despine()

plt.show()
```

```
/home/spark-ff69a62a-7422-447b-9397-42/.ipykernel/2943/command-6391446602122651-
1910241514:9: 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.

  ax = sns.barplot(
```



Those offers that have higher duration more likely lead to a transaction.

```
[ ]: corr = offer_related_transaction_profile_df.drop(["offer_led_to_transaction",␣
     ↪"offer_id", "account_id", "gender", "channels", "offer_type", ], axis = 1).
     ↪corr()
     corr.style.background_gradient(cmap='coolwarm')
```

```
[ ]: <pandas.io.formats.style.Styler at 0x7f3e81b10b30>
```

This is mostly a good news because we do not have the risc of singularity and model unstability. For trees, we dont have the effect of biased featurte importance, or redundant random splits

0.3 Modeling

I will train two models, one for offer-userprofile transaction probability and the other one which accounts for no-offer probability using the user profile characteristics.

```
[ ]: ## I will keep the X_holdout and y_holdout for my final uplift test

     y = offer_related_transaction_profile_df["offer_led_to_transaction"]
     X = offer_related_transaction_profile_df.
     ↪drop(columns=["offer_led_to_transaction", "offer_id", "account_id"])

     ## ---- separating a test dataset for uplift calculation -----------------

     X_train_full, X_holdout, y_train_full, y_holdout = train_test_split(
         X, y, test_size=0.2, random_state=42, stratify=y
     )

     ## ----------------------------------------------------------------------
```

1. Offer - transaction predictor   Categorical to numeric data conversion

Here I have several choices: 1. Simple category to number conversion * **Pros**: simple * **Cons**: the numbers have no meaning at all, and no order exist.

2. one-hot encoding

- **Pros**: preserves information about categories
- **Cons**: a) adds high dimensionality to the data; b) makes data sparse; c) when used with trees, it cause the tree to grow in the direction of zeros, to split redundantly, and to overfit due to tree complexity.

3. using target-encoding

- **Pros**: when there is a strong relation between categories and the target, it helps the model to learn better and faster.
- **Cons**: Data leackage: if not handled correctly, it might leak target information and overestimate the model performance.

I will choose target-encoding due to the analysis results I had earlier that shows some categories are infact related to a specific target. For features like gender no encoding type will help due to the lack of correlation with the target.

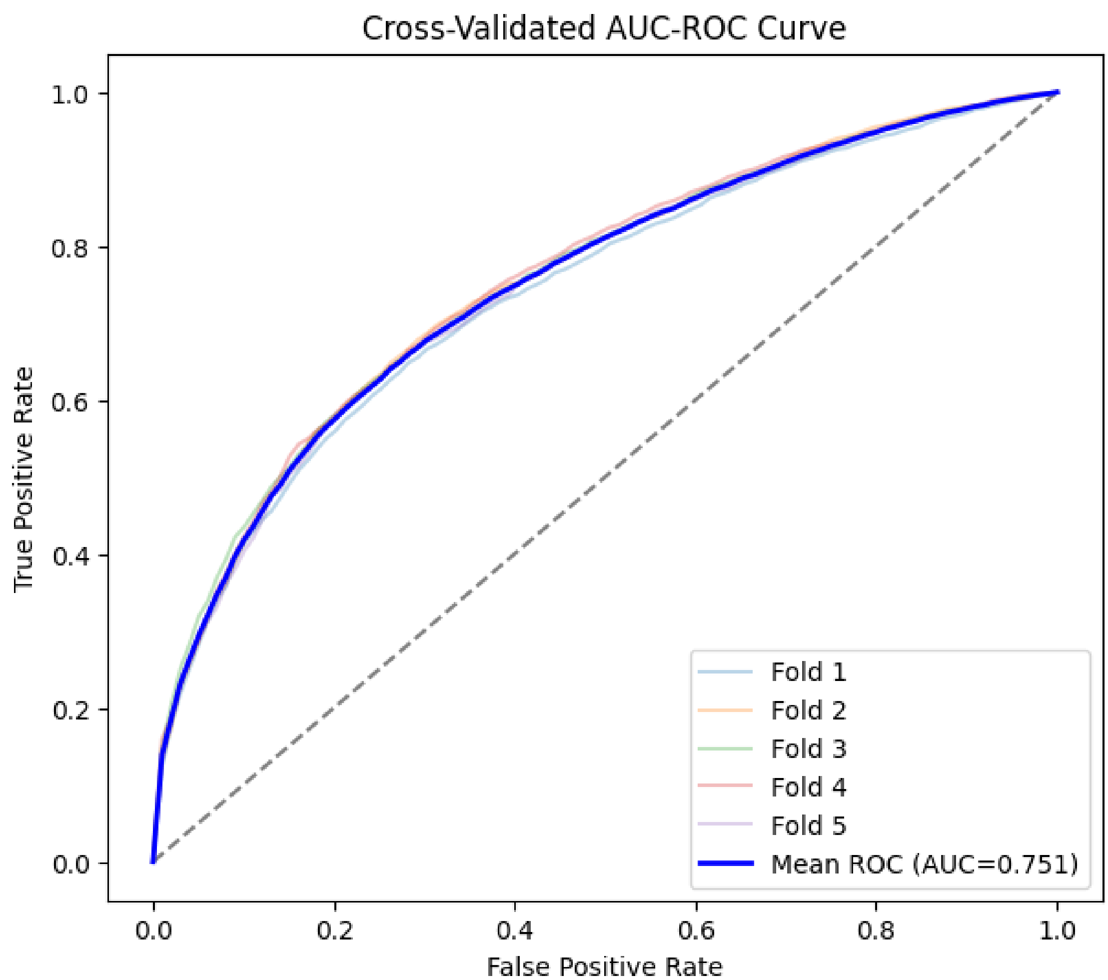
```
[ ]: def train_and_test(X, y, categorical_cols, threshold = 0.5):
         """
         Train and evaluate an XGBoost classifier with stratified 5-fold CV.

         Parameters
         ----------
         X : pd.DataFrame
             Feature matrix
         y : pd.Series
             Target variable (binary)
         categorical_cols : list
             List of categorical column names to be target-encoded
         threshold : float
             Classification threshold (default 0.5)
         """

         skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
         avg_precision, precisions, recalls, f1s, aucs, tprs, pr_aucs, mccs  = [],␣
     ↪[], [], [], [], [], [], []
         mean_fpr = np.linspace(0, 1, 100)
         mean_recall = np.linspace(0, 1, 100)
         precisions_interp = []

         for fold, (train_index, test_index) in enumerate(skf.split(X, y)):
             X_train_kfold, X_test_kfold = X.iloc[train_index].copy(), X.
     ↪iloc[test_index].copy()
             y_train_kfold, y_test_kfold = y.iloc[train_index], y.iloc[test_index]
             neg, pos = np.bincount(y_train_kfold)
             scale_pos_weight_param = neg/pos

             print(f"\nFold {fold+1}:")
             print(f"  Train set class distribution: {np.bincount(y_train_kfold)}")
             print(f"  Test set class distribution: {np.bincount(y_test_kfold)}")

             # Target encoding (fit only on training fold) so that I can avoid␣
     ↪target leackage
             target_encoder = ce.TargetEncoder(cols=categorical_cols)
             X_train_enc = target_encoder.fit_transform(X_train_kfold, y_train_kfold)
             X_test_enc = target_encoder.transform(X_test_kfold)


             ## I will use binary logistic because it is naturally callibrated loss
             model = XGBClassifier(
```

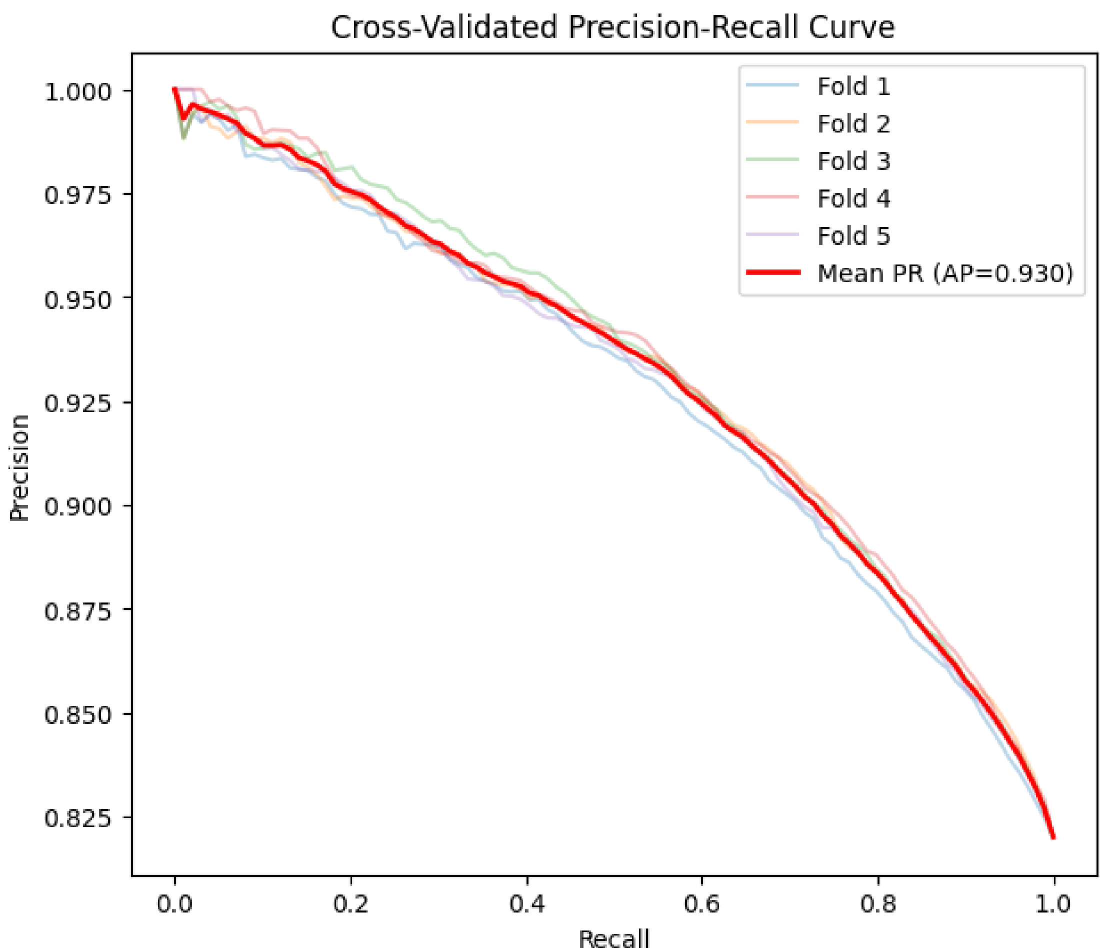
```
            random_state=42,
            eval_metric="logloss",
            reg_alpha=1,
            reg_lambda=1,
            max_depth=4,
            n_estimators=200,
            scale_pos_weight = scale_pos_weight_param
        )
        model.fit(X_train_enc, y_train_kfold)


        y_pred_proba = model.predict_proba(X_test_enc)[:, 1]
        y_pred = (y_pred_proba > threshold).astype(int)

        precisions.append(precision_score(y_test_kfold, y_pred))
        recalls.append(recall_score(y_test_kfold, y_pred))
        f1s.append(f1_score(y_test_kfold, y_pred))
        mccs.append(matthews_corrcoef(y_test_kfold, y_pred))
        avg_precision.append(average_precision_score(y_test_kfold,␣
↪y_pred_proba))
        auc = roc_auc_score(y_test_kfold, y_pred_proba)
        aucs.append(auc)

        fpr, tpr, _ = roc_curve(y_test_kfold, y_pred_proba)
        tprs.append(np.interp(mean_fpr, fpr, tpr))
        tprs[-1][0] = 0.0

        prec, rec, _ = precision_recall_curve(y_test_kfold, y_pred_proba)
        pr_auc = average_precision_score(y_test_kfold, y_pred_proba)
        pr_aucs.append(pr_auc)

        precisions_interp.append(np.interp(mean_recall, rec[::-1], prec[::-1]))


    plt.figure(figsize=(7, 6))
    for i, tpr in enumerate(tprs):
        plt.plot(mean_fpr, tpr, alpha=0.3, label=f"Fold {i+1}")
    plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
    plt.plot(mean_fpr, np.mean(tprs, axis=0), color="b", lw=2,
            label=f"Mean ROC (AUC={np.mean(aucs):.3f})")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Cross-Validated AUC-ROC Curve")
    plt.legend()
    plt.show()

    plt.figure(figsize=(7, 6))
```

```
    for i, prec in enumerate(precisions_interp):
        plt.plot(mean_recall, prec, alpha=0.3, label=f"Fold {i+1}")
    plt.plot(mean_recall, np.mean(precisions_interp, axis=0), color="r", lw=2,
            label=f"Mean PR (AP={np.mean(pr_aucs):.3f})")
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title("Cross-Validated Precision-Recall Curve")
    plt.legend()
    plt.show()

    print("Average ROC AUC:", np.mean(aucs))
    print("Average PR AUC:", np.mean(pr_aucs))
    print("Average precision:", np.mean(precisions))
    print("Average recall:", np.mean(recalls))
    print("Average f1:", np.mean(f1s))
    print("Mathews Correlation Coefficient (MCC):", np.mean(mccs))
```

[ ]:
```
train_and_test(X_train_full, y_train_full, ["gender", "channels",␣
 ↪"offer_type"], threshold= 0.3)
```

```
Fold 1:
  Train set class distribution: [ 7285 33219]
  Test set class distribution: [1821 8305]

Fold 2:
  Train set class distribution: [ 7285 33219]
  Test set class distribution: [1821 8305]

Fold 3:
  Train set class distribution: [ 7285 33219]
  Test set class distribution: [1821 8305]

Fold 4:
  Train set class distribution: [ 7285 33219]
  Test set class distribution: [1821 8305]

Fold 5:
  Train set class distribution: [ 7284 33220]
  Test set class distribution: [1822 8304]
```


Cross-Validated AUC-ROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Mean ROC (AUC=0.751)




```
Average ROC AUC: 0.75061153469968
Average PR AUC: 0.9302599513713954
Average precision: 0.8598988563185488
Average recall: 0.8927368051374673
Average f1: 0.8760058336404712
Mathews Correlation Coefficient (MCC): 0.24777545547572388
```

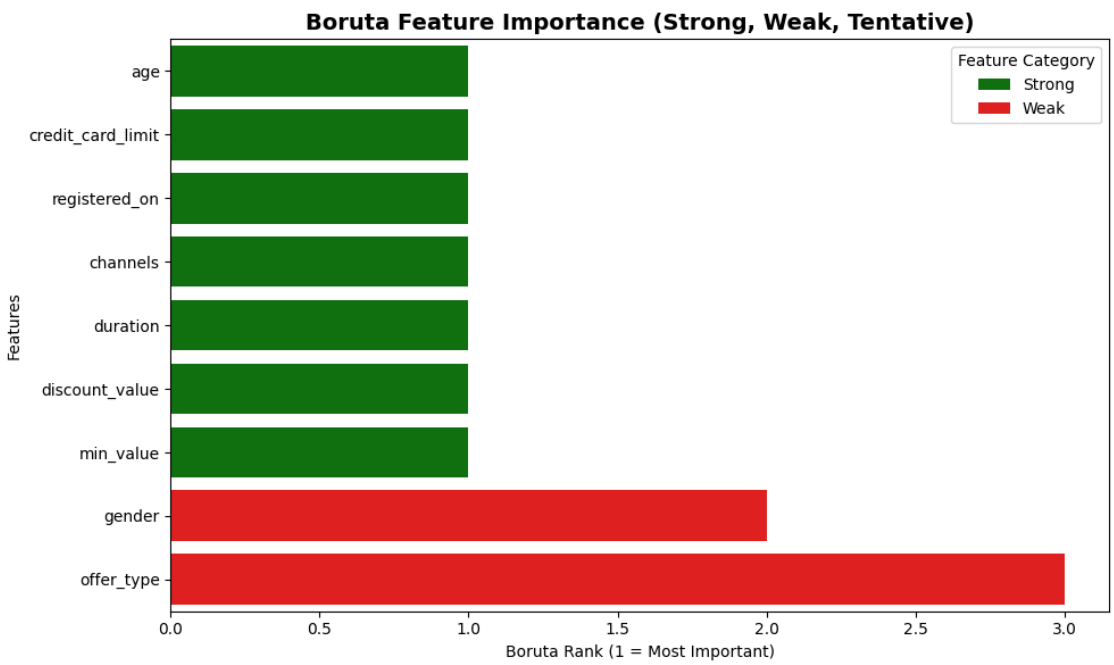
[ ]:

```
## Feature importance
def boruta_feature_importance(X: pd.DataFrame, y: pd.Series, max_iter: int =
↪100, random_state: int = 42):
    """
    Run Boruta for feature importance and plot results.

    Parameters
    ----------
    X : pd.DataFrame
        Feature matrix
```

```
    y : pd.Series
        Target variable (binary/multi-class)
    max_iter : int
        Maximum number of iterations for Boruta
    random_state : int
        Random seed for reproducibility

    Returns
    -------
    feature_ranks : pd.DataFrame
        DataFrame with features and their Boruta ranks
    """

    rf = RandomForestClassifier(n_estimators=100, random_state=random_state,␣
↪n_jobs=-1, class_weight="balanced")
    boruta = BorutaPy(rf, n_estimators="auto", verbose=0,␣
↪random_state=random_state, max_iter=max_iter)

    boruta.fit(X.values, y.values)

    feature_ranks = pd.DataFrame({
        "feature": X.columns,
        "rank": boruta.ranking_,
        "support": boruta.support_,
        "tentative": boruta.support_weak_
    }).sort_values(by="rank")

    def categorize(row):
        if row["support"]:
            return "Strong"
        elif row["tentative"]:
            return "Tentative"
        else:
            return "Weak"

    feature_ranks["category"] = feature_ranks.apply(categorize, axis=1)

    plt.figure(figsize=(10,6))
    sns.barplot(
        data=feature_ranks,
        x="rank", y="feature", hue="category",
        dodge=False, palette={"Strong":"green", "Weak":"red", "Tentative":
↪"orange"}
    )

    plt.title("Boruta Feature Importance (Strong, Weak, Tentative)",␣
↪fontsize=14, weight="bold")
```

```
    plt.xlabel("Boruta Rank (1 = Most Important)")
    plt.ylabel("Features")
    plt.legend(title="Feature Category")
    plt.tight_layout()
    plt.show()

    return feature_ranks
```

[ ]:
```
target_encoder = ce.TargetEncoder(cols=categorical_cols)
X_train_enc = target_encoder.fit_transform(X_train_full, y_train_full)
X_train_enc_clean = X_train_enc.dropna()
y_train_enc_clean = y_train_full.loc[X_train_enc_clean.index]

boruta_feature_importance(X_train_enc_clean, y_train_enc_clean)
```



[ ]:

	feature	rank	support	tentative	category
0	age	1	True	False	Strong
1	credit_card_limit	1	True	False	Strong
3	registered_on	1	True	False	Strong
4	channels	1	True	False	Strong
6	duration	1	True	False	Strong
5	discount_value	1	True	False	Strong
7	min_value	1	True	False	Strong
2	gender	2	False	False	Weak
8	offer_type	3	False	False	Weak

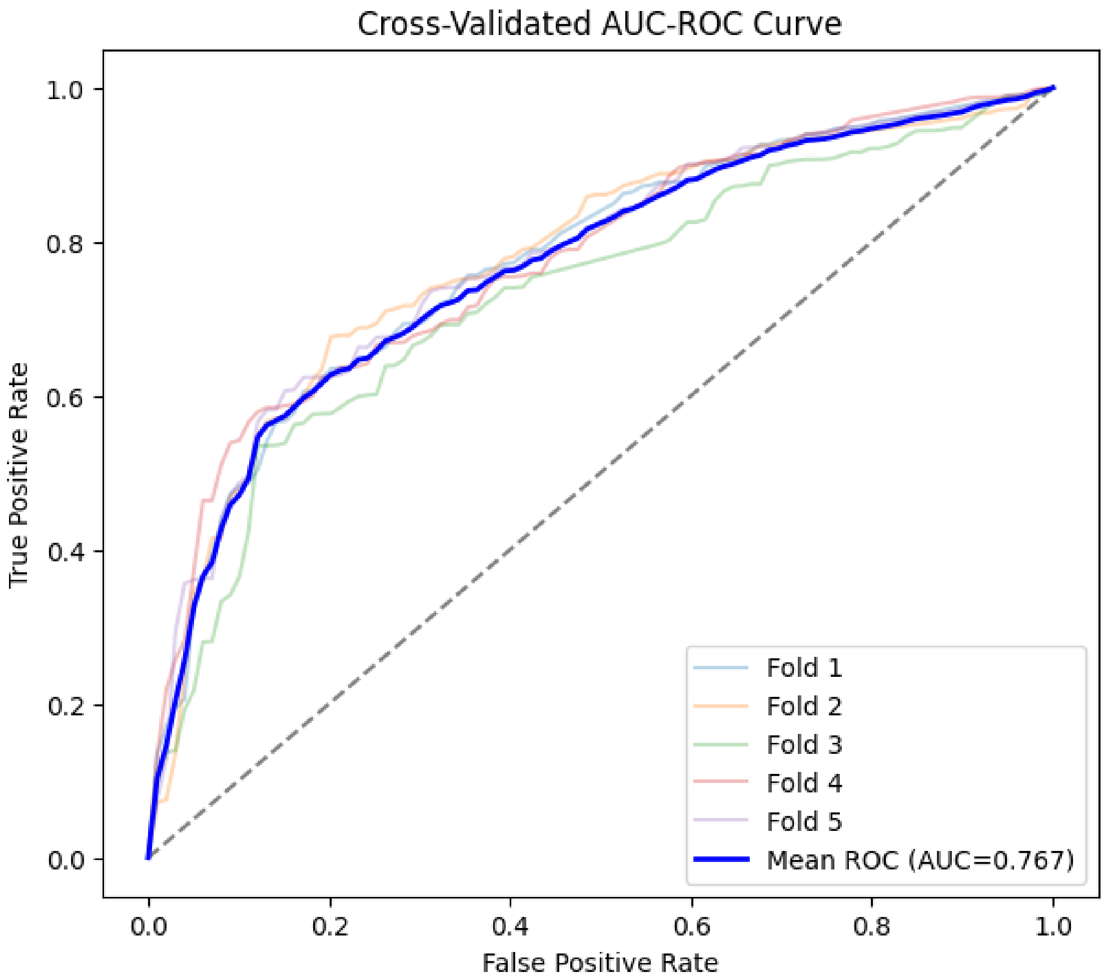
2. Profile - transaction predictor

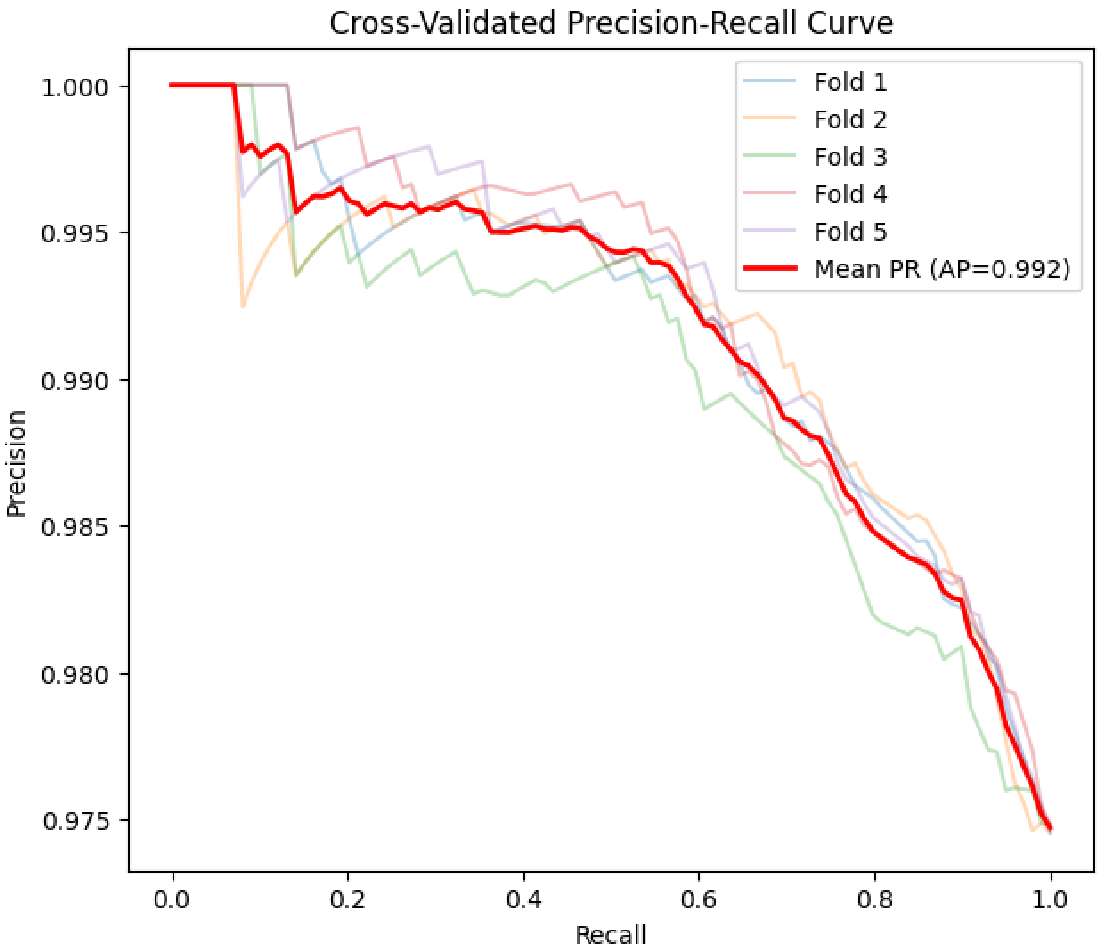
```
[ ]: user_transaction_profile_df["registered_on"] = pd.
      ↪to_datetime(user_transaction_profile_df["registered_on"], format="%Y%m%d")
     user_transaction_profile_df["year"] =␣
      ↪user_transaction_profile_df["registered_on"].dt.year

     user_transaction_profile_df.columns
```

```
[ ]: Index(['account_id', 'age', 'credit_card_limit', 'gender', 'registered_on',
            'class', 'year'],
           dtype='object')
```

```
[ ]: X = user_transaction_profile_df.drop(["account_id", "registered_on", 'class'],␣
      ↪axis = 1)
     y = user_transaction_profile_df['class']
     train_and_test(X, y, ["gender"], threshold = 0.3)
```

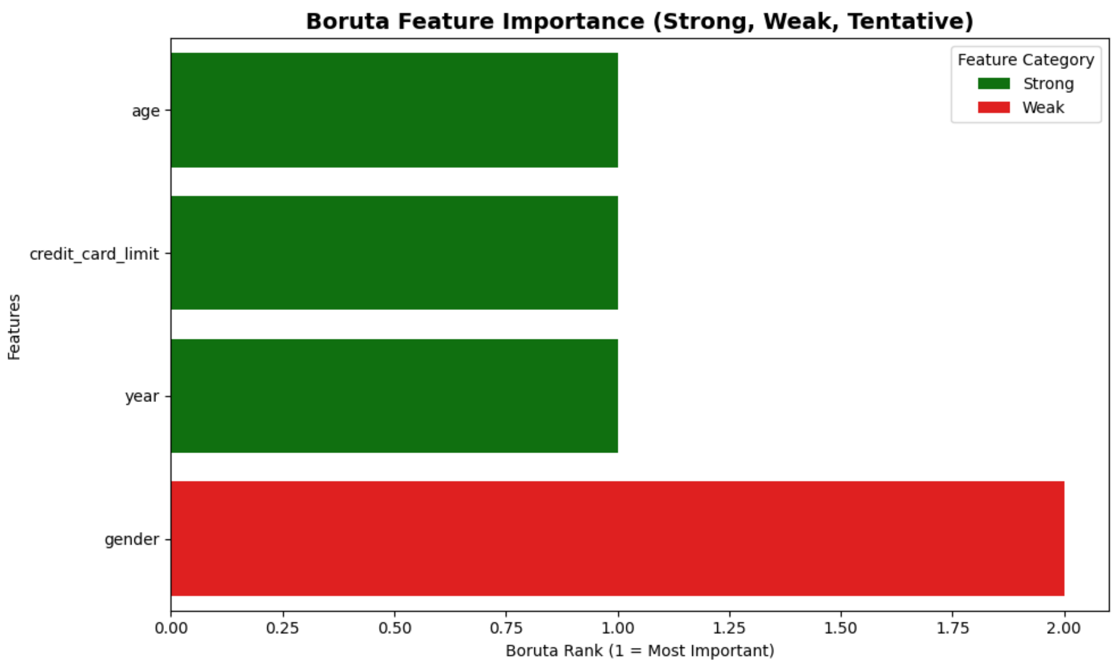
```
Fold 1:
  Train set class distribution: [  337 13013]
  Test set class distribution: [  85 3253]

Fold 2:
  Train set class distribution: [  337 13013]
  Test set class distribution: [  85 3253]

Fold 3:
  Train set class distribution: [  338 13012]
  Test set class distribution: [  84 3254]

Fold 4:
  Train set class distribution: [  338 13013]
  Test set class distribution: [  84 3253]

Fold 5:
  Train set class distribution: [  338 13013]
  Test set class distribution: [  84 3253]
```


Cross-Validated AUC-ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Mean ROC (AUC=0.767)




```
Average ROC AUC: 0.7673912915951652
Average PR AUC: 0.9915045307411999
Average precision: 0.982572659299394
Average recall: 0.8937046055166136
Average f1: 0.9359978832013315
Mathews Correlation Coefficient (MCC): 0.1396220210804054
```

[ ]:
```
target_encoder = ce.TargetEncoder(cols=["gender"])
X_train_enc = target_encoder.fit_transform(X, y)
X_train_enc_clean = X_train_enc.dropna()
y_train_enc_clean = y.loc[X_train_enc_clean.index]

boruta_feature_importance(X_train_enc_clean, y_train_enc_clean)
```



[ ]:

	feature	rank	support	tentative	category
0	age	1	True	False	Strong
1	credit_card_limit	1	True	False	Strong
3	year	1	True	False	Strong
2	gender	2	False	False	Weak