01_predictive_modelling_0

September 25, 2021

1 Wells Fargo Challenge

- https://www.mindsumo.com/contests/campus-analytics-challenge-2021

1.0.1 To Complete a Submission:

Build a classification model for predicting elder fraud in the digital payments space as described in Rule 4, which:

- Handles missing variables
- Maximizes the F1 score
- Uses the given data set
- Includes suitable encoding schemes
- Has the least set of feature variables

1.0.2 Resources

- https://github.com/pdglenn/WellsFargoAnalyticsChallenge

[1]:
```
import pandas as pd
import numpy as np
import pylab as plt
import seaborn as sns

data_dir = "./dataset/"

# following few lines are to supress the pandas warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)

pd.options.mode.chained_assignment = None
pd.options.display.max_columns = 20
np.set_printoptions(suppress=True)

data_dir = "./dataset/"
image_dir = "./images/"
```

1.1 Loading the data

Note `pd.read_excel` gave me an errow while reading the `xlsx` file so had to install `openpyxl` using `pip3 install openpyxl` and give `engine=openpyxl` as an extra arguement.

[2]:
```
#!pip3 install openpyxl
```

[3]:
```
# load the file
df_orig = pd.read_excel(data_dir+"trainset.xlsx", engine='openpyxl')
df_orig.head(2)
```

[3]:
```
   TRAN_AMT  ACCT_PRE_TRAN_AVAIL_BAL  CUST_AGE  OPEN_ACCT_CT  WF_dvc_age  \
0      5.38                 23619.91        47             4        2777
1     65.19                     0.00        45             5        2721

         PWD_UPDT_TS                CARR_NAME   RGN_NAME STATE_PRVNC_TXT  \
0  1/16/2018 11:3:58  cox communications inc.  southwest          nevada
1                NaN   charter communications  southwest      california

  ALERT_TRGR_CD  … CUST_STATE      PH_NUM_UPDT_TS CUST_SINCE_DT  \
0          MOBL  …         NV  2/24/2021 15:55:10    1993-01-06
1          MOBL  …         CA                 NaN    1971-01-07

             TRAN_TS    TRAN_DT ACTN_CD ACTN_INTNL_TXT TRAN_TYPE_CD  \
0   5/3/2021 18:3:58   5/3/2021  SCHPMT     P2P_COMMIT          P2P
1  1/13/2021 19:19:37  1/13/2021  SCHPMT     P2P_COMMIT          P2P

    ACTVY_DT FRAUD_NONFRAUD
0   5/3/2021      Non-Fraud
1  1/13/2021      Non-Fraud

[2 rows x 24 columns]
```

[4]:
```
print ("Original data shape:", df_orig.shape)
```

```
Original data shape: (14000, 24)
```

[5]:
```
#information of the dataset
df_orig.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14000 entries, 0 to 13999
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   TRAN_AMT                 14000 non-null  float64
 1   ACCT_PRE_TRAN_AVAIL_BAL  14000 non-null  float64
 2   CUST_AGE                 14000 non-null  int64
```

```
 3   OPEN_ACCT_CT             14000 non-null  int64
 4   WF_dvc_age               14000 non-null  int64
 5   PWD_UPDT_TS              10875 non-null  object
 6   CARR_NAME                11291 non-null  object
 7   RGN_NAME                 11291 non-null  object
 8   STATE_PRVNC_TXT          11291 non-null  object
 9   ALERT_TRGR_CD            14000 non-null  object
 10  DVC_TYPE_TXT             12239 non-null  object
 11  AUTHC_PRIM_TYPE_CD       14000 non-null  object
 12  AUTHC_SCNDRY_STAT_TXT    13926 non-null  object
 13  CUST_ZIP                 14000 non-null  int64
 14  CUST_STATE               13964 non-null  object
 15  PH_NUM_UPDT_TS           6939 non-null   object
 16  CUST_SINCE_DT            14000 non-null  datetime64[ns]
 17  TRAN_TS                  14000 non-null  object
 18  TRAN_DT                  14000 non-null  object
 19  ACTN_CD                  14000 non-null  object
 20  ACTN_INTNL_TXT           14000 non-null  object
 21  TRAN_TYPE_CD             14000 non-null  object
 22  ACTVY_DT                 14000 non-null  object
 23  FRAUD_NONFRAUD           14000 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(4), object(17)
memory usage: 2.6+ MB
```

[6]:
```
# check the target classes
df_orig["FRAUD_NONFRAUD"].unique()
```

[6]:
```
array(['Non-Fraud', 'Fraud'], dtype=object)
```

1.2 Train test split

Before doing any data visualization let's set some test data aside and use them to score the model later on.

[7]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# stratify the target column so that the distribution look similar in the train␣
↪and test data
df_train0, df_test0 = train_test_split(df_orig,
                                       test_size = .2,
                                       random_state = 8848,
                                       shuffle = True,
                                       stratify = df_orig["FRAUD_NONFRAUD"])
```

[8]:
```
df = df_train0.copy()
```

[9]:
```
df.head(2)
```

[9]:
```
        TRAN_AMT  ACCT_PRE_TRAN_AVAIL_BAL  CUST_AGE  OPEN_ACCT_CT  WF_dvc_age  \
2413    487.93                  3714.91        43             5        1037
1003      4.84                     0.00        53             5         305

             PWD_UPDT_TS CARR_NAME RGN_NAME STATE_PRVNC_TXT ALERT_TRGR_CD  \
2413                 NaN       NaN      NaN             NaN          MOBL
1003  4/12/2017 15:54:53       NaN      NaN             NaN          MOBL

      … CUST_STATE     PH_NUM_UPDT_TS CUST_SINCE_DT            TRAN_TS  \
2413  …         CO  5/0/2020 12:33:41    1988-01-11   4/13/2021 5:2:29
1003  …         TX                NaN    1987-04-05  4/29/2021 22:54:53

        TRAN_DT ACTN_CD ACTN_INTNL_TXT TRAN_TYPE_CD   ACTVY_DT FRAUD_NONFRAUD
2413  4/13/2021  SCHPMT     P2P_COMMIT          P2P  4/13/2021          Fraud
1003  4/29/2021  SCHPMT     P2P_COMMIT          P2P  4/29/2021      Non-Fraud

[2 rows x 24 columns]
```

[10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11200 entries, 2413 to 114
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   TRAN_AMT                 11200 non-null  float64
 1   ACCT_PRE_TRAN_AVAIL_BAL  11200 non-null  float64
 2   CUST_AGE                 11200 non-null  int64
 3   OPEN_ACCT_CT             11200 non-null  int64
 4   WF_dvc_age               11200 non-null  int64
 5   PWD_UPDT_TS              8684 non-null   object
 6   CARR_NAME                9022 non-null   object
 7   RGN_NAME                 9022 non-null   object
 8   STATE_PRVNC_TXT          9022 non-null   object
 9   ALERT_TRGR_CD            11200 non-null  object
 10  DVC_TYPE_TXT             9805 non-null   object
 11  AUTHC_PRIM_TYPE_CD       11200 non-null  object
 12  AUTHC_SCNDRY_STAT_TXT    11140 non-null  object
 13  CUST_ZIP                 11200 non-null  int64
 14  CUST_STATE               11172 non-null  object
 15  PH_NUM_UPDT_TS           5579 non-null   object
 16  CUST_SINCE_DT            11200 non-null  datetime64[ns]
 17  TRAN_TS                  11200 non-null  object
 18  TRAN_DT                  11200 non-null  object
 19  ACTN_CD                  11200 non-null  object
 20  ACTN_INTNL_TXT           11200 non-null  object
 21  TRAN_TYPE_CD             11200 non-null  object
```

```
 22  ACTVY_DT                  11200 non-null  object
 23  FRAUD_NONFRAUD            11200 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(4), object(17)
memory usage: 2.1+ MB
```

[11]:
```
df.shape
```

[11]:
```
(11200, 24)
```

[12]:
```
# find numerical and categorical columns
nume_cols = list(df.select_dtypes(include="number").columns)
cate_cols = list(df.select_dtypes(exclude="number").columns)
cate_cols.remove('FRAUD_NONFRAUD')
```

[13]:
```
print ("Numeric Columns:\n", nume_cols)
print ("")
print ("Categorical Columns:\n", cate_cols)
```

```
Numeric Columns:
 ['TRAN_AMT', 'ACCT_PRE_TRAN_AVAIL_BAL', 'CUST_AGE', 'OPEN_ACCT_CT',
'WF_dvc_age', 'CUST_ZIP']

Categorical Columns:
 ['PWD_UPDT_TS', 'CARR_NAME', 'RGN_NAME', 'STATE_PRVNC_TXT', 'ALERT_TRGR_CD',
'DVC_TYPE_TXT', 'AUTHC_PRIM_TYPE_CD', 'AUTHC_SCNDRY_STAT_TXT', 'CUST_STATE',
'PH_NUM_UPDT_TS', 'CUST_SINCE_DT', 'TRAN_TS', 'TRAN_DT', 'ACTN_CD',
'ACTN_INTNL_TXT', 'TRAN_TYPE_CD', 'ACTVY_DT']
```

[14]:
```
df[nume_cols].head(2)
```

[14]:
```
      TRAN_AMT  ACCT_PRE_TRAN_AVAIL_BAL  CUST_AGE  OPEN_ACCT_CT  WF_dvc_age  \
2413    487.93                  3714.91        43             5        1037
1003      4.84                     0.00        53             5         305

      CUST_ZIP
2413     80234
1003     75232
```

[15]:
```
nume_cols.remove('CUST_ZIP')
cate_cols.append('CUST_ZIP')
```

[16]:
```
print ("Numeric Columns:\n", nume_cols)
print ("")
print ("Categorical Columns:\n", cate_cols)
```

```
Numeric Columns:
 ['TRAN_AMT', 'ACCT_PRE_TRAN_AVAIL_BAL', 'CUST_AGE', 'OPEN_ACCT_CT',
'WF_dvc_age']
```

```
Categorical Columns:
 ['PWD_UPDT_TS', 'CARR_NAME', 'RGN_NAME', 'STATE_PRVNC_TXT', 'ALERT_TRGR_CD',
'DVC_TYPE_TXT', 'AUTHC_PRIM_TYPE_CD', 'AUTHC_SCNDRY_STAT_TXT', 'CUST_STATE',
'PH_NUM_UPDT_TS', 'CUST_SINCE_DT', 'TRAN_TS', 'TRAN_DT', 'ACTN_CD',
'ACTN_INTNL_TXT', 'TRAN_TYPE_CD', 'ACTVY_DT', 'CUST_ZIP']
```

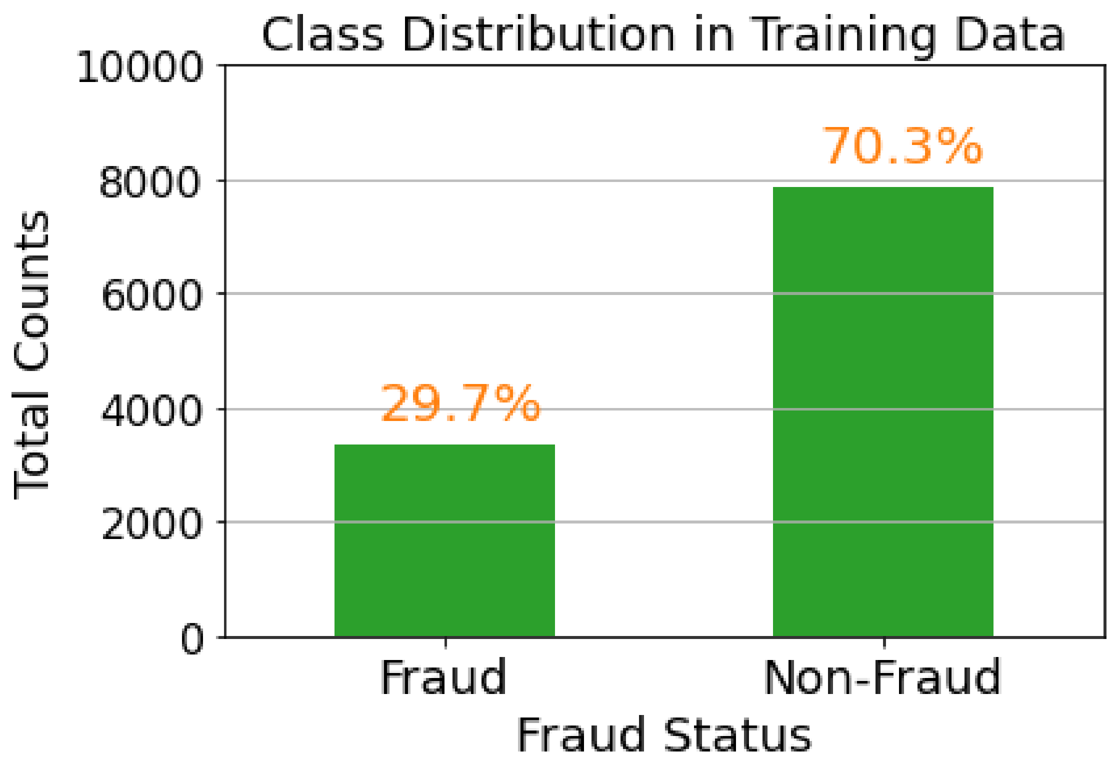
[17]:
```
df[nume_cols].head(5)
```

[17]:

	TRAN_AMT	ACCT_PRE_TRAN_AVAIL_BAL	CUST_AGE	OPEN_ACCT_CT	WF_dvc_age
2413	487.93	3714.91	43	5	1037
1003	4.84	0.00	53	5	305
8660	494.94	2525.50	70	9	583
6349	0.01	0.00	70	6	467
1860	488.36	4344.55	38	4	0

1.3 Class Distribution

[18]:
```
dfs=df.groupby("FRAUD_NONFRAUD")["CUST_ZIP"].count()
dfs.plot(kind='bar', color="C2")
plt.grid(axis='y')
plt.xticks(rotation=0, fontsize=18);
plt.xlabel("Fraud Status", fontsize=18);
plt.ylabel("Total Counts", fontsize=18);
plt.yticks( fontsize=16);
plt.title("Class Distribution in Training Data", fontsize=18);
pcts = np.round(100*dfs.values/df.shape[0], 1)
plt.text(0-0.15, 3800, str(pcts[0])+"%", fontsize=20, color="C1");
plt.text(1-0.15, 8300, str(pcts[1])+"%", fontsize=20, color="C1");
plt.ylim([0, 10000]);
plt.savefig("images/class_distribution.png", dpi=300, bbox_inches='tight')
```



1.4 Handling missing variables

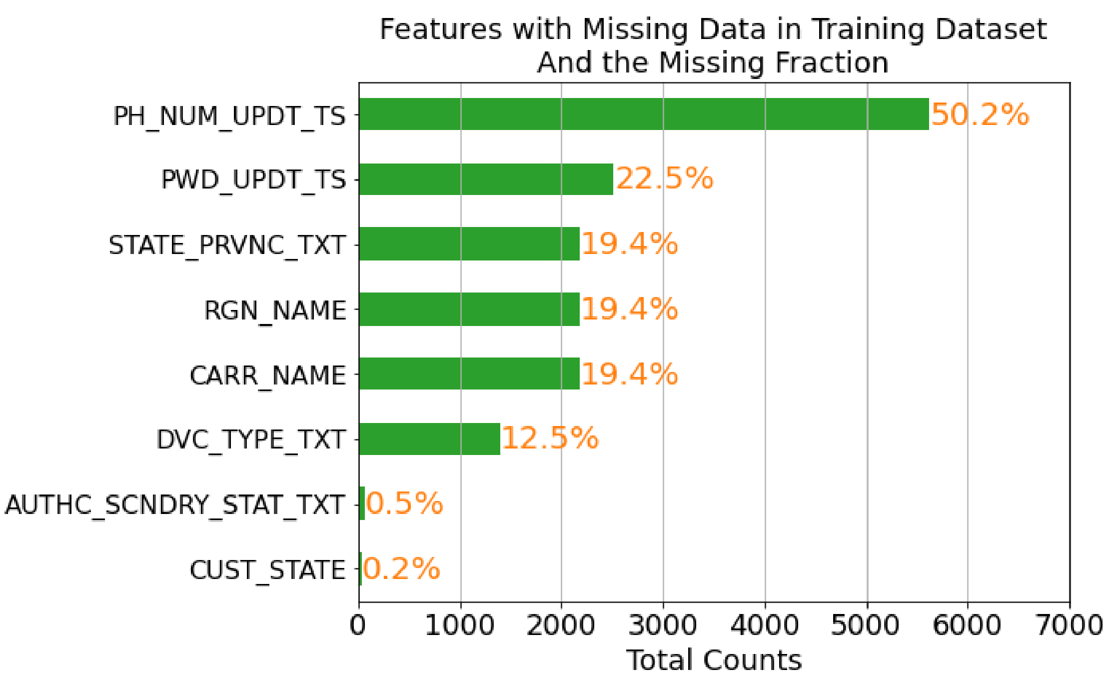
[19]:
```
df.isnull().sum()
```

[19]:
```
TRAN_AMT                       0
ACCT_PRE_TRAN_AVAIL_BAL        0
CUST_AGE                       0
OPEN_ACCT_CT                   0
WF_dvc_age                     0
PWD_UPDT_TS                 2516
CARR_NAME                   2178
RGN_NAME                    2178
STATE_PRVNC_TXT             2178
ALERT_TRGR_CD                  0
DVC_TYPE_TXT                1395
AUTHC_PRIM_TYPE_CD             0
AUTHC_SCNDRY_STAT_TXT         60
CUST_ZIP                       0
CUST_STATE                    28
PH_NUM_UPDT_TS              5621
CUST_SINCE_DT                  0
TRAN_TS                        0
```

```
TRAN_DT                        0
ACTN_CD                        0
ACTN_INTNL_TXT                 0
TRAN_TYPE_CD                   0
ACTVY_DT                       0
FRAUD_NONFRAUD                 0
dtype: int64
```

[20]:

```
dfnull=df.isnull().sum()[df.isnull().sum()>0].sort_values(ascending=True)

dfnull.plot(kind='barh', color="C2", figsize=(8,6))
plt.grid(axis='x')
plt.yticks(rotation=0, fontsize=16);
#plt.ylabel("Features with Missing values", fontsize=18);
plt.xlabel("Total Counts", fontsize=18);
plt.xticks( fontsize=18);
plt.title("Features with Missing Data in Training Dataset\nAnd the Missing␣
↪Fraction",
          fontsize=18);
pcts = np.round(100*dfnull.values/df.shape[0], 1)
vals = dfnull.values

for i in range(len(pcts)):
    plt.text(vals[i]*1, i-0.15, str(pcts[i])+"%", fontsize=20, color="C1");

plt.xlim([0, 7000]);
plt.savefig("images/missing_data.png", dpi=300, bbox_inches='tight')
```



- From the above figure, we can see that 5 features: PH_NUM_UPDT_TS, PWD_UPDT_TS, CARR_NAME, RGN_NAME, STATE_PRVNC_TXT have almost one fifth of their total training data missing. Imputing these features is doable but the model might not be able to learn much from these features, so I believe dropping these features from the model is a good idea.
- There are 3 features: DVC_TYPE_TXT, AUTHC_SCNDRY_STAT_TXT, CUST_STATE which have less than one fifth missing data. In particular AUTHC_SCNDRY_STAT_TXT, CUST_STATE have less than 1 % of the missing data, which is completely normal in real world data. And we are going to impute the missing values in these three features.
- In order to impute the missing data we are using following two methods:
  - If the feature is `numerical`, we are going to impute the values by the `median` of the entire feature values.
  - If the feature is `categorical`, we are going to impute the values by the `mode` of the entire feature values.

1.5 Case when real-world data has missing data in a new features

- This can totally happen when the model is deployed for production. To avoid our model from failing we have to make sure our code has a way to impute the missing data for any features that the model uses.
- Create a dictionary with all the column names as keys and the imputation value as the value.

[21]:
```python
impute_vals={}

for col in df.columns:
    if col in nume_cols:
        impute_vals[col] = df[col].median()
    elif col in cate_cols:
        impute_vals[col] = df[col].mode()[0]

impute_vals
```

[21]:
```
{'TRAN_AMT': 162.07,
 'ACCT_PRE_TRAN_AVAIL_BAL': 2396.1549999999997,
 'CUST_AGE': 59.0,
 'OPEN_ACCT_CT': 5.0,
 'WF_dvc_age': 366.5,
 'PWD_UPDT_TS': '5/18/2020 4:7:20',
 'CARR_NAME': 'cox communications inc.',
 'RGN_NAME': 'southwest',
 'STATE_PRVNC_TXT': 'california',
 'ALERT_TRGR_CD': 'MOBL',
 'DVC_TYPE_TXT': 'MOBILE',
 'AUTHC_PRIM_TYPE_CD': 'UN_PWD',
```

```
 'AUTHC_SCNDRY_STAT_TXT': 'ALLOW',
 'CUST_ZIP': 77459,
 'CUST_STATE': 'CA',
 'PH_NUM_UPDT_TS': '7/8/2019 6:45:37',
 'CUST_SINCE_DT': Timestamp('1997-08-01 00:00:00'),
 'TRAN_TS': datetime.datetime(2021, 10, 1, 0, 0),
 'TRAN_DT': '2/28/2021',
 'ACTN_CD': 'SCHPMT',
 'ACTN_INTNL_TXT': 'P2P_COMMIT',
 'TRAN_TYPE_CD': 'P2P',
 'ACTVY_DT': '2/28/2021'}
```

[22]:
```
cols_to_drop = ['PH_NUM_UPDT_TS', 'PWD_UPDT_TS', 'CARR_NAME', 'RGN_NAME',␣
 ↪'STATE_PRVNC_TXT']
nume_cols     = [c for c in nume_cols if c not in cols_to_drop]
cate_cols     = [c for c in cate_cols if c not in cols_to_drop]
```

[23]:
```
def impute_data(df, impute_dict=impute_vals):
    """
    this function takes in a dataframe and list of columns which have missing␣
 ↪values
    then imputes those columns using the precomputed values.
    """
    for col in list(impute_dict.keys()):
        df[col] = df[col].fillna(impute_dict[col])
    return df
```

[24]:
```
# impute the columns : cols_to_impute
df=impute_data(df)
```

[25]:
```
df.isnull().sum()
```

[25]:
```
TRAN_AMT                  0
ACCT_PRE_TRAN_AVAIL_BAL   0
CUST_AGE                  0
OPEN_ACCT_CT              0
WF_dvc_age                0
PWD_UPDT_TS               0
CARR_NAME                 0
RGN_NAME                  0
STATE_PRVNC_TXT           0
ALERT_TRGR_CD             0
DVC_TYPE_TXT              0
AUTHC_PRIM_TYPE_CD        0
AUTHC_SCNDRY_STAT_TXT     0
CUST_ZIP                  0
CUST_STATE                0
```

```
PH_NUM_UPDT_TS              0
CUST_SINCE_DT               0
TRAN_TS                     0
TRAN_DT                     0
ACTN_CD                     0
ACTN_INTNL_TXT              0
TRAN_TYPE_CD                0
ACTVY_DT                    0
FRAUD_NONFRAUD              0
dtype: int64
```

[26]:
```
df.head(2)
```

[26]:
```
      TRAN_AMT  ACCT_PRE_TRAN_AVAIL_BAL  CUST_AGE  OPEN_ACCT_CT  WF_dvc_age  \
2413    487.93                  3714.91        43             5        1037
1003      4.84                     0.00        53             5         305

             PWD_UPDT_TS                CARR_NAME   RGN_NAME STATE_PRVNC_TXT  \
2413    5/18/2020 4:7:20  cox communications inc.  southwest      california
1003  4/12/2017 15:54:53  cox communications inc.  southwest      california

     ALERT_TRGR_CD  … CUST_STATE     PH_NUM_UPDT_TS CUST_SINCE_DT  \
2413          MOBL  …         CO  5/0/2020 12:33:41    1988-01-11
1003          MOBL  …         TX   7/8/2019 6:45:37    1987-04-05

                TRAN_TS    TRAN_DT ACTN_CD ACTN_INTNL_TXT TRAN_TYPE_CD  \
2413    4/13/2021 5:2:29  4/13/2021  SCHPMT     P2P_COMMIT          P2P
1003  4/29/2021 22:54:53  4/29/2021  SCHPMT     P2P_COMMIT          P2P

       ACTVY_DT FRAUD_NONFRAUD
2413  4/13/2021          Fraud
1003  4/29/2021      Non-Fraud

[2 rows x 24 columns]
```

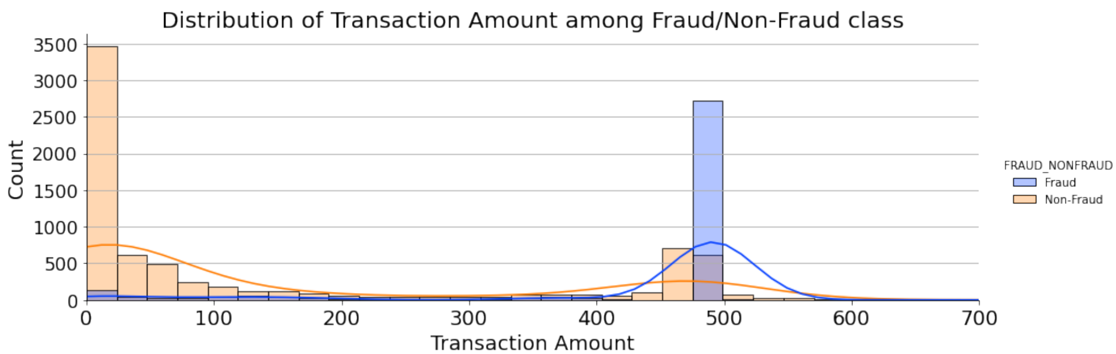
[27]:
```
df[nume_cols].head(2)
```

[27]:
```
      TRAN_AMT  ACCT_PRE_TRAN_AVAIL_BAL  CUST_AGE  OPEN_ACCT_CT  WF_dvc_age
2413    487.93                  3714.91        43             5        1037
1003      4.84                     0.00        53             5         305
```

[28]:
```
def displot(df, xcol, xlabel, title, savename, xmax=None, bins=100):
    sns.displot(data=df,
            x=xcol,
            alpha=0.3,
            hue="FRAUD_NONFRAUD",
            height=4,
```

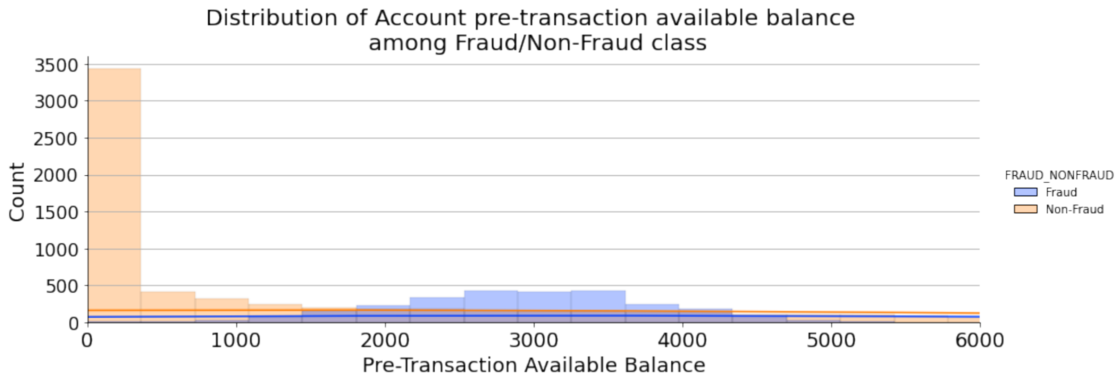
```
                aspect=3,
                kde=True,
                palette="bright",
                bins=bins,
               )
    plt.yticks(rotation=0, fontsize=16);
    plt.ylabel("Count", fontsize=18);
    plt.xlabel(xlabel, fontsize=18);
    plt.xticks(fontsize=18);
    plt.title(title,fontsize=20);
    if xmax:
        plt.xlim([0, xmax]);
    plt.grid(axis='y')
    plt.savefig("images/"+savename+".png", dpi=300, bbox_inches='tight')
```

[29]:
```
xcol="TRAN_AMT"
xlabel="Transaction Amount"
title="Distribution of Transaction Amount among Fraud/Non-Fraud class"
savename="dist_trans_amnt.png"
bins=100
xmax=700
displot(df, xcol, xlabel, title, savename, bins=bins, xmax=xmax)
```



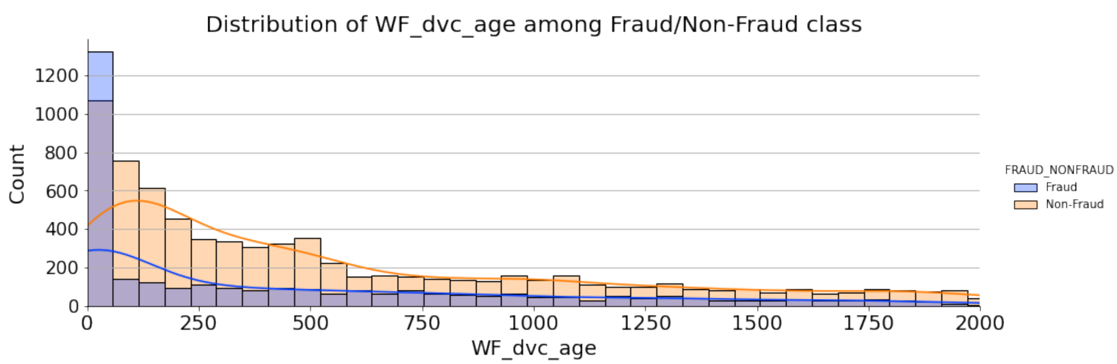
- This plot shows that most of the Fraudulent transactions have been around $500.
- So this clearly is an important feature in the model.

[30]:
```
xcol="ACCT_PRE_TRAN_AVAIL_BAL"
xlabel="Pre-Transaction Available Balance"
title="Distribution of Account pre-transaction available balance \n among Fraud/
 ↪Non-Fraud class"
savename="dist_pre_trans_blnce.png"
bins=1000
xmax=6000
displot(df, xcol, xlabel, title, savename, bins=bins, xmax=xmax)
```



- This plot shows us that there is a clear peak of distribution for fraudalant transaction in the range 2000-4000, compared to non-fraudalant class which peaks near $100.
- So, this feature is an important one for the model.

[31]:
```
xcol="WF_dvc_age"
xlabel="WF_dvc_age"
title="Distribution of WF_dvc_age among Fraud/Non-Fraud class"
savename="dist_wf_dvc_age.png"
bins=50
xmax=2000
displot(df, xcol, xlabel, title, savename, bins=bins, xmax=xmax)
```

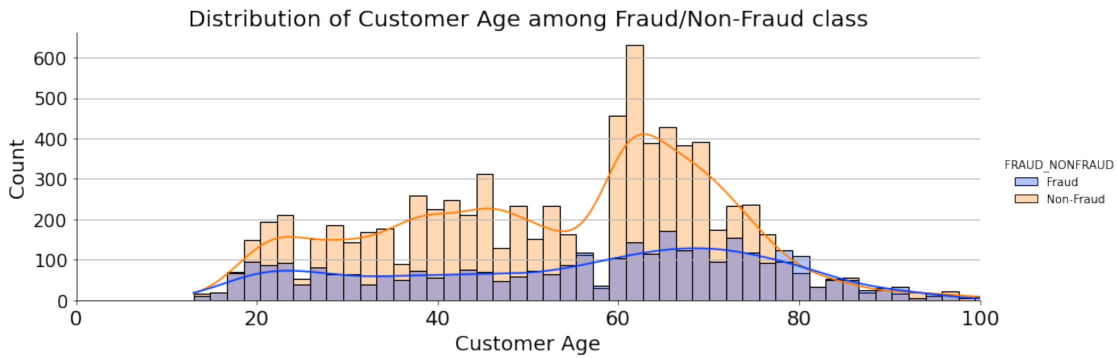


- Unlike the previous features, this plot doesn't tell us much about the distinction between fraud vs non-fraud class.
- So, this feature may not be greatly important but we should still keep this features as there is some distribution in the range [0,500]

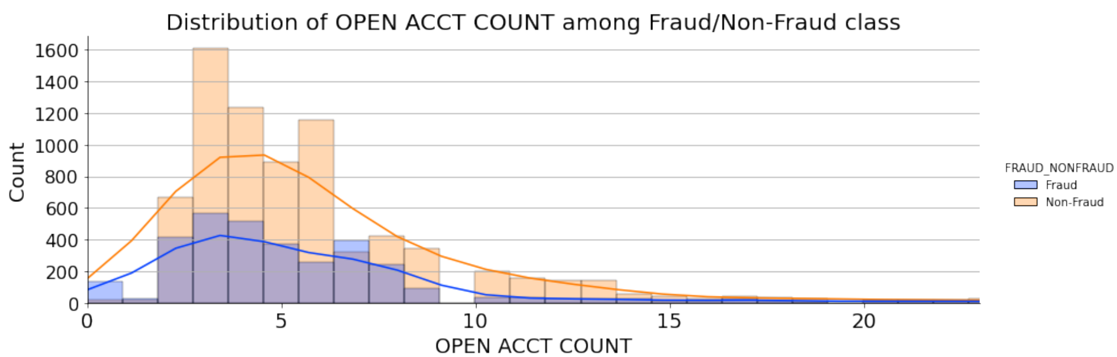
[32]:
```
nume_cols
```

[32]: ['TRAN_AMT',
 'ACCT_PRE_TRAN_AVAIL_BAL',
 'CUST_AGE',
 'OPEN_ACCT_CT',
 'WF_dvc_age']

[33]:
```
xcol = "CUST_AGE"
xlabel="Customer Age"
title="Distribution of Customer Age among Fraud/Non-Fraud class"
savename="dist_cust_age.png"
bins=50
xmax=100
displot(df, xcol, xlabel, title, savename, bins=bins, xmax=xmax)
```



[34]:
```
xcol='OPEN_ACCT_CT'
xlabel="OPEN ACCT COUNT"
title="Distribution of OPEN ACCT COUNT among Fraud/Non-Fraud class"
savename="dist_open_Acct_ct.png"
bins=250
xmax=23
displot(df, xcol, xlabel, title, savename, bins=bins, xmax=xmax)
```



1.6 Categorical Features

[35]:
```
# find # of unique features in all categorical features
for col in cate_cols:
    print (col, "\t# of Unique values:\t",df[col].nunique() )
```

```
ALERT_TRGR_CD   # of Unique values:     2
DVC_TYPE_TXT    # of Unique values:     4
AUTHC_PRIM_TYPE_CD      # of Unique values:     5
AUTHC_SCNDRY_STAT_TXT   # of Unique values:     3
CUST_STATE      # of Unique values:     48
CUST_SINCE_DT   # of Unique values:     7431
TRAN_TS         # of Unique values:     10871
TRAN_DT         # of Unique values:     333
ACTN_CD         # of Unique values:     1
ACTN_INTNL_TXT  # of Unique values:     1
TRAN_TYPE_CD    # of Unique values:     1
ACTVY_DT        # of Unique values:     333
CUST_ZIP        # of Unique values:     3750
```

[36]:
```
# find # of unique features in all categorical features
d0={c:df[c].nunique() for c in cate_cols if (df[c].nunique()<=10) }
d1={c:df[c].nunique() for c in cate_cols if (df[c].nunique()>10) & (df[c].
 ↪nunique()<=100) }
d2={c:df[c].nunique() for c in cate_cols if (df[c].nunique()>100) }

print ("Features with unique value in the range [1,10]:\n",d0)
print ("\nFeatures with unique value in the range [10,100]:\n",d1)
print ("\nFeatures with unique value in the range [100,10000]:\n",d2)
```

```
Features with unique value in the range [1,10]:
 {'ALERT_TRGR_CD': 2, 'DVC_TYPE_TXT': 4, 'AUTHC_PRIM_TYPE_CD': 5,
'AUTHC_SCNDRY_STAT_TXT': 3, 'ACTN_CD': 1, 'ACTN_INTNL_TXT': 1, 'TRAN_TYPE_CD':
1}

Features with unique value in the range [10,100]:
 {'CUST_STATE': 48}

Features with unique value in the range [100,10000]:
 {'CUST_SINCE_DT': 7431, 'TRAN_TS': 10871, 'TRAN_DT': 333, 'ACTVY_DT': 333,
'CUST_ZIP': 3750}
```

Observations Broadly we can group the categorical features into 3 categories.

Features that have #unique values [**1,10**] - 'ALERT_TRGR_CD' : 2 - 'DVC_TYPE_TXT' :4 - 'AUTHC_PRIM_TYPE_CD' : 5 - 'AUTHC_SCNDRY_STAT_TXT' : 3 - 'ACTN_CD' : 1 - 'ACTN_INTNL_TXT' : 1 - 'TRAN_TYPE_CD' : 1

We can safely delete features `ACTN_CD`, `ACTN_INTNL_TXT`,`TRAN_TYPE_CD` as they have constant value all across.

For others do some analysis on the distribution.

Features that have #unique values [10,100] - 'CUST_STATE' 48 - 'CUST_AGE' 90 - 'OPEN_ACCT_CT' 50

We can't use all of these unique values so find a way to cut these sort

Features that have #unique values > 100

- 'CUST_SINCE_DT' 7431
- 'TRAN_TS' 10871
- 'TRAN_DT' 333
- 'ACTVY_DT' 333
- 'CUST_ZIP' 3750

For these the `TRAN_DT` and `ACTVY_DT` have same unique numbers so they must be same value. Remove one. For the date, it may not tell much to use all of it so may be break it up into year/month/day/time and so on.

For `CUST_ZIP` do some distribution analysis and see how it's distributed.

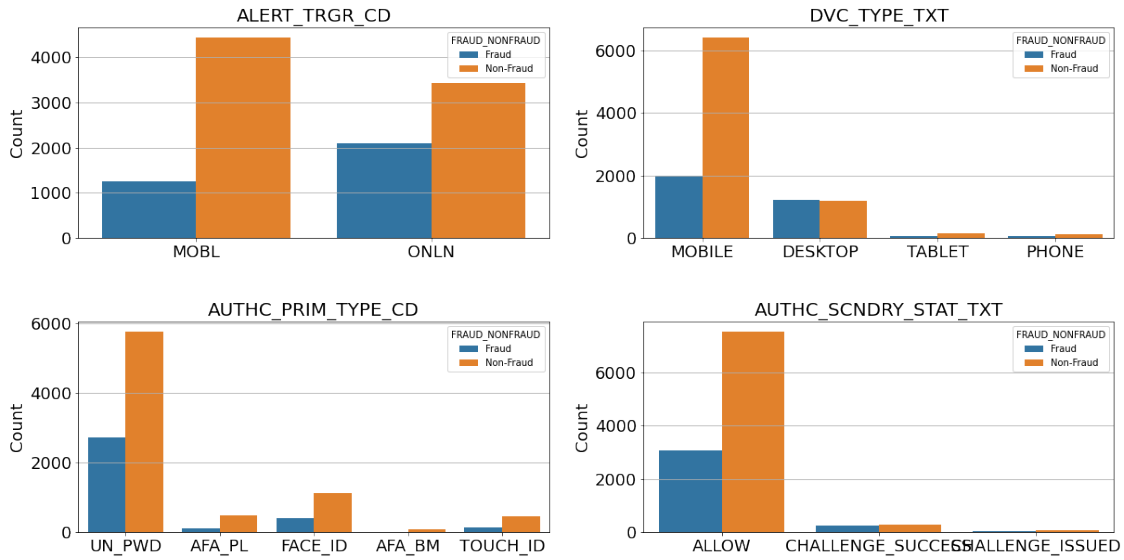
For `CUST_SINCE_DT` we might break up the date into year only. as month might not matter much.

For `TRAN_TS`, do more analysis as in why it's ogt so may unique values.

[37]:
```
d0
```

[37]:
```
{'ALERT_TRGR_CD': 2,
 'DVC_TYPE_TXT': 4,
 'AUTHC_PRIM_TYPE_CD': 5,
 'AUTHC_SCNDRY_STAT_TXT': 3,
 'ACTN_CD': 1,
 'ACTN_INTNL_TXT': 1,
 'TRAN_TYPE_CD': 1}
```

[38]:
```
fig, ax = plt.subplots(2,2, figsize=(20,10))
feats = [c for c in list(d0.keys()) if d0[c]>1]
for ic, col in enumerate(feats):
    axi=ax[ic//2, ic%2]
    sns.countplot(x=col, hue="FRAUD_NONFRAUD", data=df, ax=axi)
    axi.set_title(col, fontsize=20)
    axi.grid(axis='y')
    plt.subplots_adjust(wspace=.2, hspace=.4)
    axi.tick_params(axis='both', labelsize=18)

    axi.set_xlabel(None, fontsize=18);
    axi.set_ylabel("Count", fontsize=18);
figname = "images/distribution_cate_features0.png"
plt.savefig(figname, dpi=300, bbox_inches='tight')
```



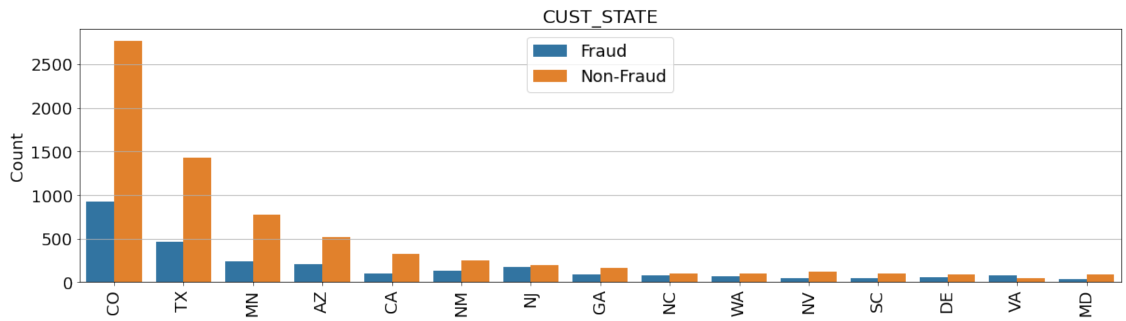
From these plots we can drop a few more columns `DVC_TYPE_TXT`, `AUTHC_PRIM_TYPE_CD`, `AUTHC_SCNDRY_STAT_TXT` as there is a very small number of data for categories other than one particular category.

[39]:
```
d1
```

[39]:
```
{'CUST_STATE': 48}
```

[56]:
```
def plot_count_plot(col, df=df ):

    fig, axi = plt.subplots(1,1, figsize=(20,5))
    sns.countplot(x=col, hue="FRAUD_NONFRAUD",
                  data=df, ax=axi,
                  order = df[col].value_counts().index,
                  #order =  df[col].value_counts().sort_index(ascending=False).keys()

                  #df[xcol].value_counts().sort_index().keys()
                  #df[xcol].value_counts().sort_index(ascending=False)
                 )
    axi.set_title(col, fontsize=20)
    axi.grid(axis='y')
    plt.subplots_adjust(wspace=.2, hspace=.4)
    axi.tick_params(axis='both', labelsize=18)
    axi.set_xticklabels(labels=df[col].unique(), rotation=90)
    axi.set_xlabel(None, fontsize=18);
    axi.set_ylabel("Count", fontsize=18);
    axi.legend(loc='upper center', fontsize=18);
    figname = "images/dist_"+col+".png"
    plt.savefig(figname, dpi=300, bbox_inches='tight')
```

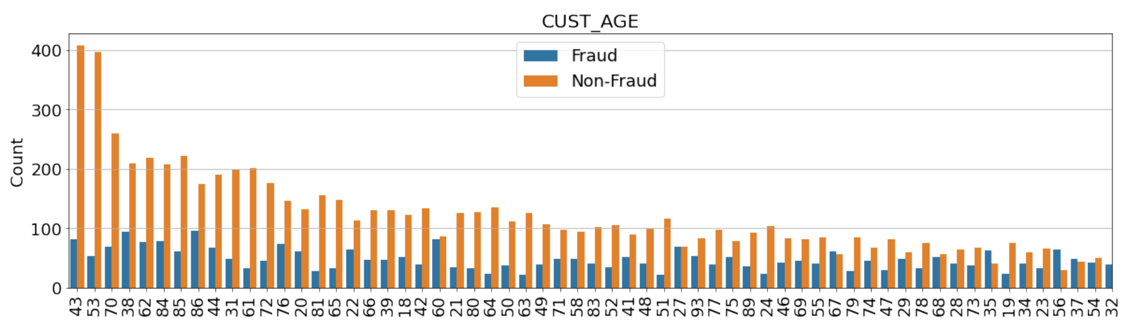
```
[57]: xcol="CUST_STATE"
plot_count_plot(xcol, df=df )
plt.xlim([-.5,14.5]);
```



for this feature keep only a few states: `CO`, `TX`, `MN`, `AZ` and convert rest into `OTHER`

```
[58]: xcol="CUST_AGE"
plot_count_plot(xcol, df=df )
plt.xlim([-.5,60])
```

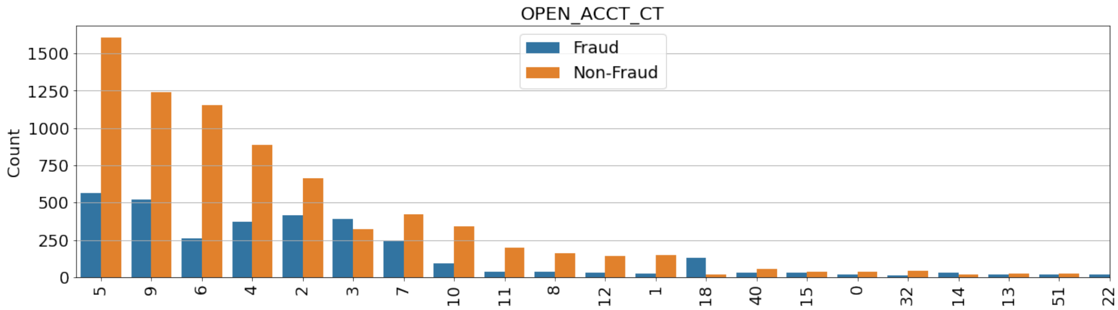
```
[58]: (-0.5, 60.0)
```



Since there is almost no distribution, remove this feature altogether.

```
[59]: xcol="OPEN_ACCT_CT"
plot_count_plot(xcol, df=df )
plt.xlim([-.5,20])
```

```
[59]: (-0.5, 20.0)
```



From this keep only the [1,12] `OPEN_ACCT_CT` and convert others to `13`

[60]:
```python
# categories with more than 100 unique values
d2
```

[60]:
```
{'CUST_SINCE_DT': 7431,
 'TRAN_TS': 10871,
 'TRAN_DT': 333,
 'ACTVY_DT': 333,
 'CUST_ZIP': 3750}
```

[61]:
```python
# check whether 'TRAN_DT' and 'ACTVY_DT' are same columns
(df['TRAN_DT']==df['ACTVY_DT']).sum()/df.shape[0]
```

[61]: `0.0`

[62]:
```python
from datetime import datetime
```

[63]:
```python
df['ACTVY_DT'] = pd.to_datetime(df['ACTVY_DT'].str.strip())
df['ACTVY_DT_DAY'] = df['ACTVY_DT'].apply(lambda x: x.day)
df['ACTVY_DT_MONTH'] = df['ACTVY_DT'].apply(lambda x: x.month)
df['ACTVY_DT_YEAR'] = df['ACTVY_DT'].apply(lambda x: x.year)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-63-ee8a81d38559> in <module>
----> 1 df['ACTVY_DT'] = pd.to_datetime(df['ACTVY_DT'].str.strip())
      2 df['ACTVY_DT_DAY'] = df['ACTVY_DT'].apply(lambda x: x.day)
      3 df['ACTVY_DT_MONTH'] = df['ACTVY_DT'].apply(lambda x: x.month)
      4 df['ACTVY_DT_YEAR'] = df['ACTVY_DT'].apply(lambda x: x.year)

/usr/local/lib/python3.9/site-packages/pandas/core/generic.py in
 ↪__getattr__(self, name)
   5459             or name in self._accessors
   5460         ):
-> 5461             return object.__getattribute__(self, name)
```

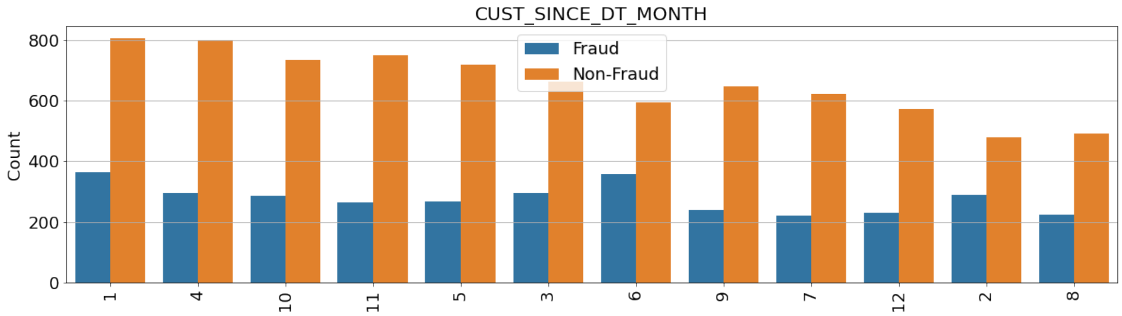
```
   5462         else:
   5463             if self._info_axis.
 ↪_can_hold_identifiers_and_holds_name(name):

/usr/local/lib/python3.9/site-packages/pandas/core/accessor.py in __get__(self,␣
 ↪obj, cls)
    178             # we're accessing the attribute of the class, i.e., Dataset
 ↪geo
    179             return self._accessor
--> 180         accessor_obj = self._accessor(obj)
    181         # Replace the property with the accessor object. Inspired by:
    182         # https://www.pydanny.com/cached-property.html

/usr/local/lib/python3.9/site-packages/pandas/core/strings/accessor.py in␣
 ↪__init__(self, data)
    152         from pandas.core.arrays.string_ import StringDtype
    153
--> 154         self._inferred_dtype = self._validate(data)
    155         self._is_categorical = is_categorical_dtype(data.dtype)
    156         self._is_string = isinstance(data.dtype, StringDtype)

/usr/local/lib/python3.9/site-packages/pandas/core/strings/accessor.py in␣
 ↪_validate(data)
    215
    216         if inferred_dtype not in allowed_types:
--> 217             raise AttributeError("Can only use .str accessor with string␣
 ↪values!")
    218         return inferred_dtype
    219

AttributeError: Can only use .str accessor with string values!
```

[64]:
```
d2
```

[64]:
```
{'CUST_SINCE_DT': 7431,
 'TRAN_TS': 10871,
 'TRAN_DT': 333,
 'ACTVY_DT': 333,
 'CUST_ZIP': 3750}
```

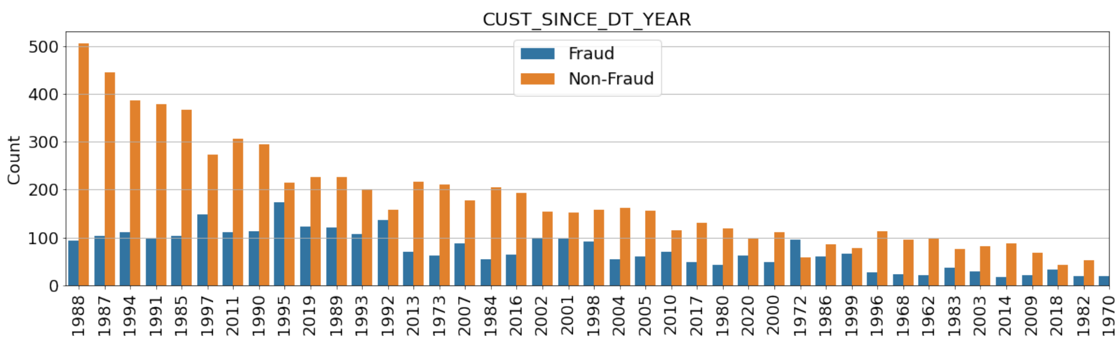
[65]:
```
df["CUST_SINCE_DT_YEAR"]=df["CUST_SINCE_DT"].apply(lambda x: x.year)
df["CUST_SINCE_DT_MONTH"] = df["CUST_SINCE_DT"].apply(lambda x: x.month)
```

[66]:
```
xcol="CUST_SINCE_DT_MONTH"
plot_count_plot(xcol, df=df )
```



[67]:
```
xcol="CUST_SINCE_DT_YEAR"
plot_count_plot(xcol, df=df )
plt.xlim([-.5,40])
```

[67]: (-0.5, 40.0)



1.7 Data Wrangling

- Convert the categorical featurs into small number of categories when possible

[68]:
```
def wrangle_column_data(df):
    #CUST_STATE
    # keep only CO, TX, MN, AZ and convert rest into OTHER
    df["CUST_STATE"] = df["CUST_STATE"].apply(lambda x: x if x in ["CO", "TX",␣
↪"MN", "AZ"] else "OTHER")
    #OPEN_ACCT_CT
    #keep only the [2,9] and convert others to 10
    df["OPEN_ACCT_CT"] = df["OPEN_ACCT_CT"].apply(lambda x: x if x in␣
↪range(1,13) else 13)
    return df
```

[69]:
```
df=wrangle_column_data(df)
```

[71]:
```
nume_cols
```

[71]:
```
['TRAN_AMT',
 'ACCT_PRE_TRAN_AVAIL_BAL',
 'CUST_AGE',
 'OPEN_ACCT_CT',
 'WF_dvc_age']
```

[73]:
```
cate_cols_to_keep = ['ALERT_TRGR_CD', "CUST_STATE"]
```

[74]:
```
df[cate_cols_to_keep].head()
```

[74]:

	ALERT_TRGR_CD	CUST_STATE
2413	MOBL	CO
1003	MOBL	TX
8660	MOBL	TX
6349	ONLN	MN
1860	MOBL	AZ

1.8 Build a model with only Numerical features

[75]:
```
# map Fraud to 1 and Non-Fraud to 0 in the dataframe for both train and test␣
↪data
df["FRAUD_NONFRAUD"] = df["FRAUD_NONFRAUD"].map({"Fraud":1, "Non-Fraud":0})
```

[76]:
```
X_train1, y_train1 = df[nume_cols], df["FRAUD_NONFRAUD"]
```

[77]:
```
# prepare test data
df_test = df_test0.copy()
print ("missing values in test data:\n", df_test[nume_cols].isnull().sum() )
df_test["FRAUD_NONFRAUD"] = df_test["FRAUD_NONFRAUD"].map({"Fraud":
↪1,"Non-Fraud":0})
X_test1,  y_test1  = df_test[nume_cols], df_test["FRAUD_NONFRAUD"]
```

```
missing values in test data:
 TRAN_AMT                   0
ACCT_PRE_TRAN_AVAIL_BAL    0
CUST_AGE                   0
OPEN_ACCT_CT               0
WF_dvc_age                 0
dtype: int64
```

[78]:
```
X_train1.shape, y_train1.shape, X_test1.shape, y_test1.shape
```

[78]:
```
((11200, 5), (11200,), (2800, 5), (2800,))
```

1.8.1 Base Model: Logistic Regression, Random Forest, XGBoost

[99]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
 ↪recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score, plot_roc_curve
```

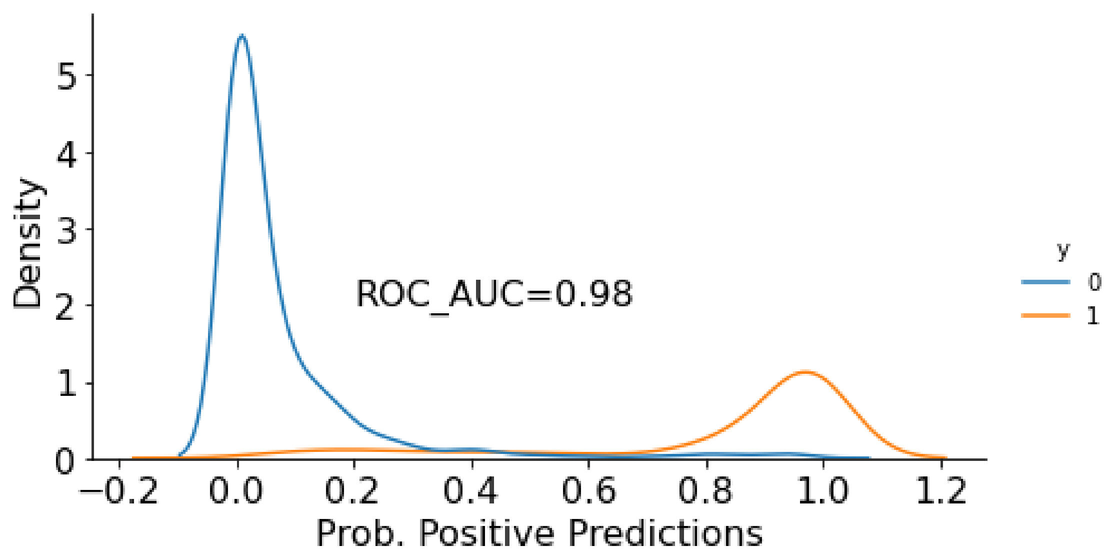
[133]:
```
class Model_training:
    def __init__(self, model, X_train, y_train, X_test, y_test, savename="Fig"):
        self.model = model
        self.X_train = X_train
        self.y_train = y_train
        self.X_test  = X_test
        self.y_test  = y_test
        self.savename = savename

        self.model.fit(self.X_train, self.y_train)

    def print_metrics(self):
        round_to_pct = lambda x: np.round(100*x, 2)
        y_pred = self.model.predict(self.X_test)
        ac = round_to_pct(accuracy_score(self.y_test, y_pred))
        f1 = round_to_pct(f1_score(self.y_test, y_pred))
        pr = round_to_pct(precision_score(self.y_test, y_pred))
        re = round_to_pct(recall_score(self.y_test, y_pred))
        print (f"Accuracy = {ac}% F1 Score= {f1}% \nPrecision={pr}% Recall=␣
 ↪{re}%")
        print (classification_report(self.y_test, y_pred))
        return (self.model, (ac, f1, pr, re))

    def displot(self):
        pr=self.model.predict_proba(self.X_test)
        roc_auc = np.round(roc_auc_score(self.y_test,
                                         self.model.predict_proba(self.X_test)[:
 ↪, 1]), 2)

        pr_df = pd.DataFrame({'pred_0':pr[:,0],
                              'pred_1':pr[:,1],
                              'y': self.y_test})

        ax=sns.displot(data=pr_df,
                       x='pred_1',
                       hue='y',
```

```
                        alpha=0.8,
                        kind="kde",
                        height = 3.5,
                        aspect=1.8);

        plt.xlabel("Prob. Positive Predictions", fontsize=16)
        plt.text(0.2, 2, "ROC_AUC="+str(roc_auc), fontsize=16)
        plt.ylabel("Density", fontsize=16)

        plt.yticks(fontsize=16);
        plt.xticks(fontsize=16);
        figname = "images/displot_"+self.savename+".png"
        plt.savefig(figname, dpi=300, bbox_inches='tight')

    def feature_importance(self):

        df_imp = pd.DataFrame({"Feature":self.X_train.columns,
                               "Feature Importance":self.model.
↪feature_importances_})

        df_imp = df_imp.sort_values(by=['Feature Importance'],
                                    axis=0,
                                    ascending=True)

        df_imp.plot(kind='barh',
                    x='Feature',
                    y='Feature Importance',
                    color="C2", figsize=(8,5));

        plt.grid(axis='x')
        plt.yticks(fontsize=16);
        plt.ylabel('');
        plt.xticks(fontsize=16);
        plt.legend(loc='best',fontsize=16);

        figname = "images/feat_imp_"+self.savename+".png"
        plt.savefig(figname, dpi=300, bbox_inches='tight')

    def plot_roc_curve(self):
        roc_auc = np.round(roc_auc_score(self.y_test,
                                         self.model.predict_proba(self.X_test)[:
↪, 1]), 2)
        label_name = self.savename +"\n AUC = "+str(roc_auc)

        plot_roc_curve(self.model, self.X_test, self.y_test,
                       lw=3., color='C2', label=label_name)
        plt.xlabel("False Positive Rate", fontsize=16)
```

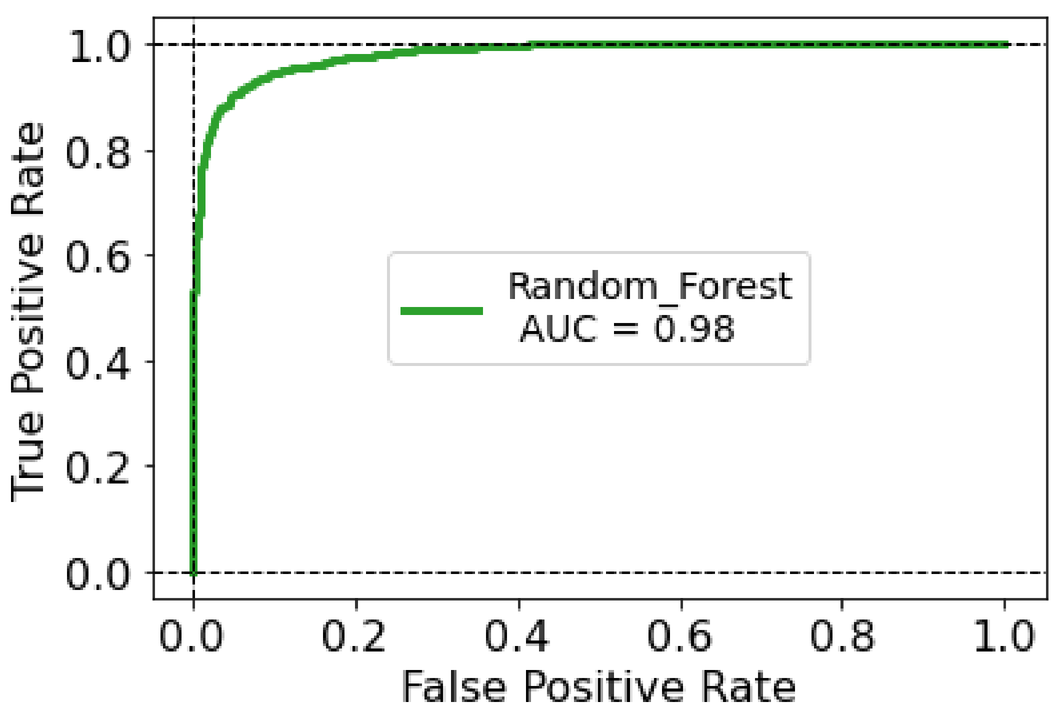
```
        plt.ylabel("True Positive Rate", fontsize=16)
        plt.xticks(fontsize=16);
        plt.yticks(fontsize=16);
        plt.legend(loc="center", fontsize=14);
        plt.axvline(x=0, color='k', ls='--', lw=1)
        plt.axhline(y=0, color='k', ls='--', lw=1)
        plt.axhline(y=1, color='k', ls='--', lw=1)

        figname = "images/roc_curve_"+self.savename+".png"
        plt.savefig(figname, dpi=300, bbox_inches='tight')
```

[134]:
```
model_rf = RandomForestClassifier(max_depth=10,
                                  random_state=8848)

mod = Model_training(model_rf,
                     X_train1, y_train1,
                     X_test1, y_test1,
                     "Random_Forest")
mod_tr, _= mod.print_metrics()
mod.displot()
mod.plot_roc_curve()
mod.feature_importance()
```

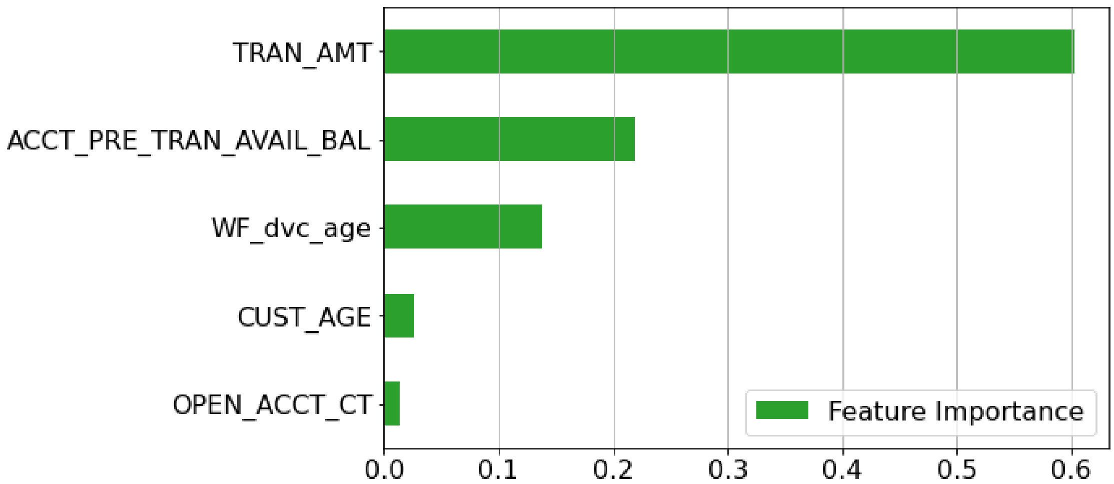
```
Accuracy = 93.64% F1 Score= 88.81%
Precision=93.26% Recall= 84.75%
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      1967
           1       0.93      0.85      0.89       833

    accuracy                           0.94      2800
   macro avg       0.94      0.91      0.92      2800
weighted avg       0.94      0.94      0.94      2800
```



ROC_AUC=0.98
Density
Prob. Positive Predictions
y
0
1



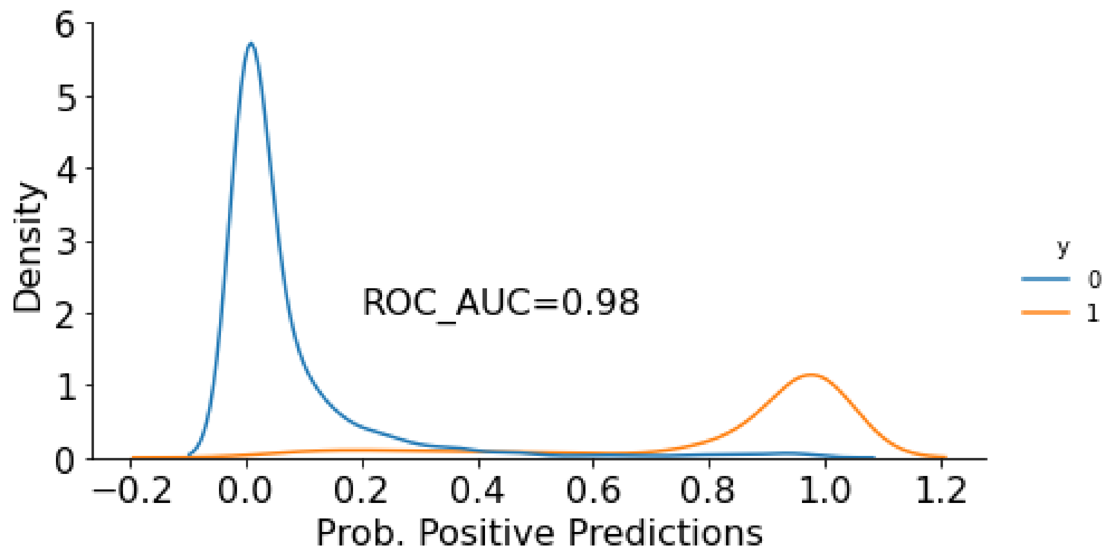
Random_Forest
AUC = 0.98
True Positive Rate
False Positive Rate



For a base model: F1 score ~ 88% is a great result.

[82]:
```
model_rf_gs = GridSearchCV(RandomForestClassifier(),
                           param_grid={'max_depth':[8, 9, 10, 11, 12, 14]},
                           scoring ='f1',
                           verbose=1)

mod2 = Model_training(model_rf_gs,
                      X_train1, y_train1, X_test1, y_test1,
                      "random_forest_grid_search")
mod_tr, _= mod2.print_metrics()
mod2.displot()
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:   23.1s finished

Accuracy = 93.71% F1 Score= 88.94%
Precision=93.28% Recall= 84.99%
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      1967
           1       0.93      0.85      0.89       833

    accuracy                           0.94      2800
   macro avg       0.94      0.91      0.92      2800
weighted avg       0.94      0.94      0.94      2800
```

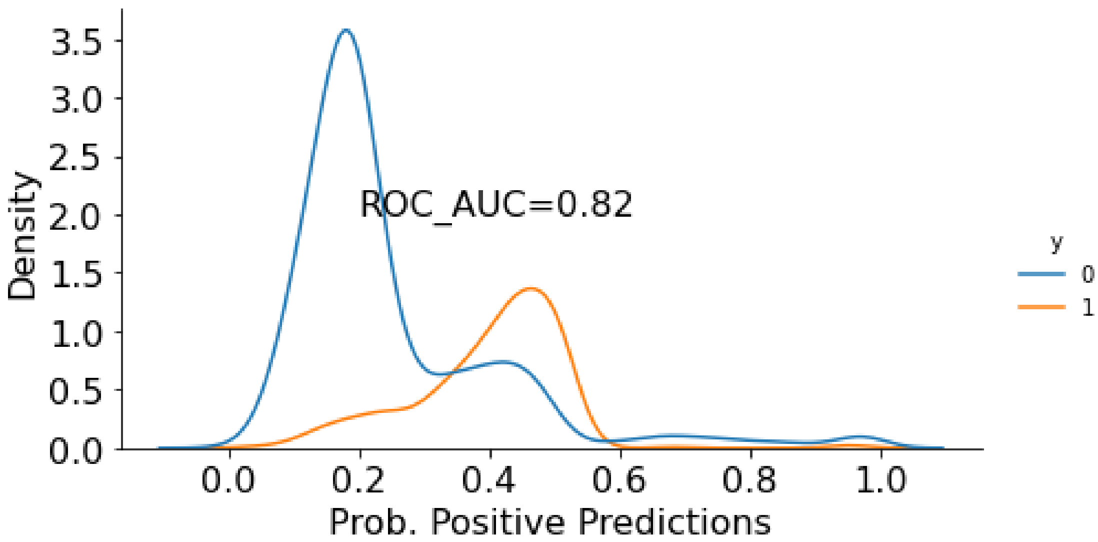



```
[83]: mod_tr.best_params_
```

```
[83]: {'max_depth': 12}
```

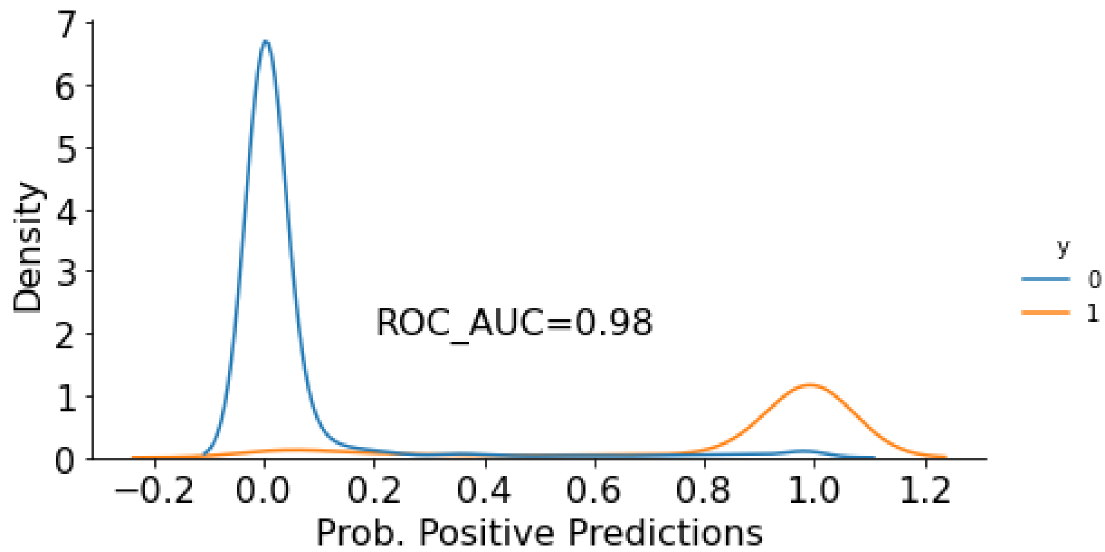
```
[84]: model_lr = LogisticRegression(max_iter=5000)
      mod3 = Model_training(model_lr, X_train1, y_train1, X_test1, y_test1,␣
       ↪"logistic_regression")
      mod_tr, _= mod3.print_metrics()
      mod3.displot()
```

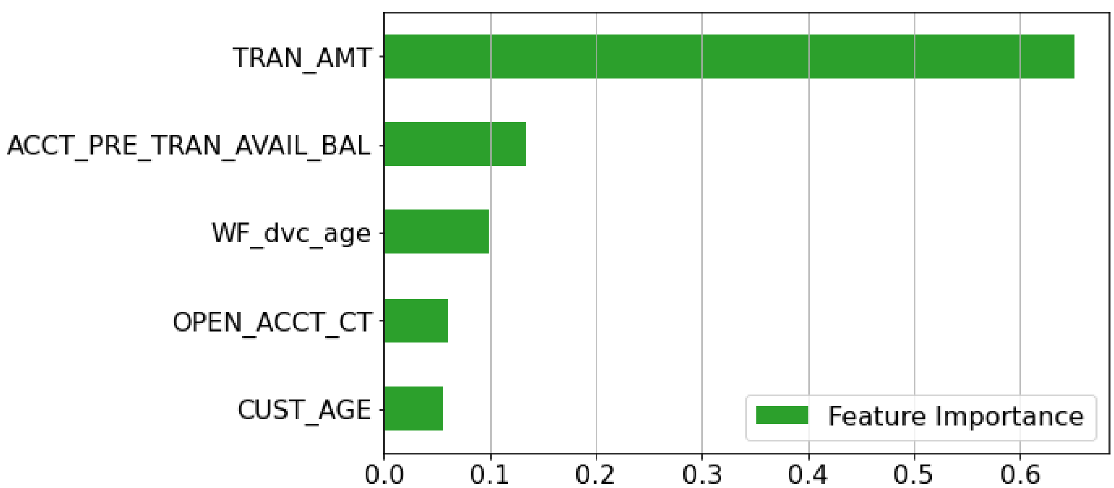
```
Accuracy = 70.86% F1 Score= 23.02%
Precision=53.74% Recall= 14.65%
              precision    recall  f1-score   support

           0       0.72      0.95      0.82      1967
           1       0.54      0.15      0.23       833

    accuracy                           0.71      2800
   macro avg       0.63      0.55      0.53      2800
weighted avg       0.67      0.71      0.64      2800
```



[85]:
```
xgb = XGBClassifier(verbosity=1,
                    use_label_encoder=False,
                    max_depth=10,
                    eval_metric = "logloss")

mod4 = Model_training(xgb, X_train1, y_train1, X_test1, y_test1, "xgb")
mod_tr, _= mod4.print_metrics()
mod4.displot()
mod4.feature_importance()
```

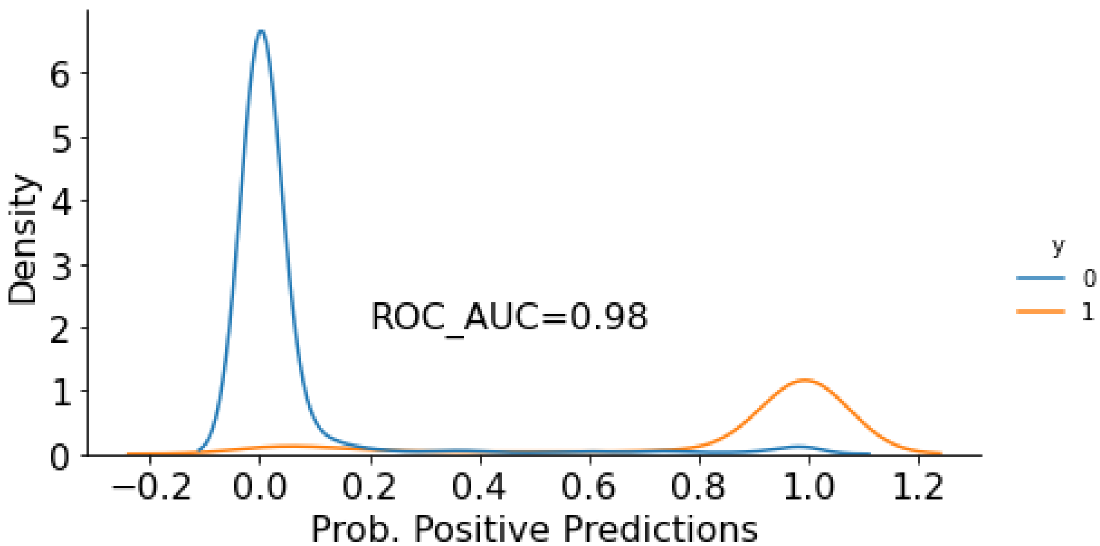
```
Accuracy = 93.14% F1 Score= 88.13%
Precision=90.83% Recall= 85.59%
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      1967
           1       0.91      0.86      0.88       833

    accuracy                           0.93      2800
   macro avg       0.92      0.91      0.92      2800
weighted avg       0.93      0.93      0.93      2800
```





[86]:
```
xgb_gs = GridSearchCV(XGBClassifier(),
                      param_grid={'max_depth':[8, 9, 10, 11, 12, 14],
                                  'eval_metric' :["logloss"],
                                  'reg_alpha':[0.1, 0.5]},
                      scoring = 'f1',
                      verbose = 0 )

mod5 = Model_training(xgb_gs, X_train1, y_train1, X_test1, y_test1,␣
 ↪"xgb_grid_search")
mod_tr, _= mod5.print_metrics()
mod5.displot()
```

```
Accuracy = 92.93% F1 Score= 87.78%
Precision=90.34% Recall= 85.35%
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      1967
           1       0.90      0.85      0.88       833

    accuracy                           0.93      2800
   macro avg       0.92      0.91      0.91      2800
weighted avg       0.93      0.93      0.93      2800
```



[87]:
```
mod_tr.best_params_
```

[87]: {'eval_metric': 'logloss', 'max_depth': 11, 'reg_alpha': 0.1}

[ ]:

[ ]:

[ ]:

[ ]:

1.9 Deep learning models

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalMaxPool1D,␣
↪BatchNormalization
from tensorflow.keras import backend as K
```

[ ]:
```
# this piece of code copied from
#https://neptune.ai/blog/implementing-the-macro-f1-score-in-keras
def custom_f1(y_true, y_pred):
    def recall_m(y_true, y_pred):
        TP = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        Positives = K.sum(K.round(K.clip(y_true, 0, 1)))

        recall = TP / (Positives+K.epsilon())
        return recall


    def precision_m(y_true, y_pred):
        TP = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        Pred_Positives = K.sum(K.round(K.clip(y_pred, 0, 1)))

        precision = TP / (Pred_Positives+K.epsilon())
        return precision

    precision, recall = precision_m(y_true, y_pred), recall_m(y_true, y_pred)

    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

[ ]:
```
def make_model_dense(X_train, y_train):
    model = Sequential()
    model.add(Dense(1, input_shape=(X_train.shape[1],), activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(1, activation='relu'))
    opt = Adam(learning_rate=0.001)
    model.compile(loss='binary_crossentropy',
                  optimizer=opt,
                  metrics=custom_f1)
    return model
```

[ ]:
```
model_dnn = make_model_dense(X_train1, y_train1)
model_dnn.summary()
```

[ ]:
```
history_dnn = model_dnn.fit(X_train1, y_train1,
                            validation_data=(X_test1, y_test1),
```

```
                             epochs=20,
                             batch_size=32,
                             verbose=0)
```

[ ]:
```
plt.plot(history_dnn.history['val_custom_f1'])
plt.plot(history_dnn.history['val_loss'])
```

1.10 Modeling including categorical features

[88]:
```
import category_encoders as ce
```

[89]:
```
df2 = df_train0.copy()
df2["FRAUD_NONFRAUD"] =  df2["FRAUD_NONFRAUD"].map({"Fraud":1, "Non-Fraud":0})
df2 = impute_data(df2)
df2 = wrangle_column_data(df2)
```

[90]:
```
encoder = ce.OneHotEncoder()
df2_tr_cat = encoder.fit_transform(df[cate_cols_to_keep])
df2_tr_join = pd.concat( [df2[nume_cols], df2_tr_cat], axis=1)
X_train2 = df2_tr_join.values
y_train2 = df2["FRAUD_NONFRAUD"].values
```

[91]:
```
# test data

df2_te = df_test0.copy()
df2_te["FRAUD_NONFRAUD"] = df2_te["FRAUD_NONFRAUD"].map({"Fraud":1, "Non-Fraud":
 ↪0})
df2_te = impute_data(df2_te)
df2_te = wrangle_column_data(df2_te)
```

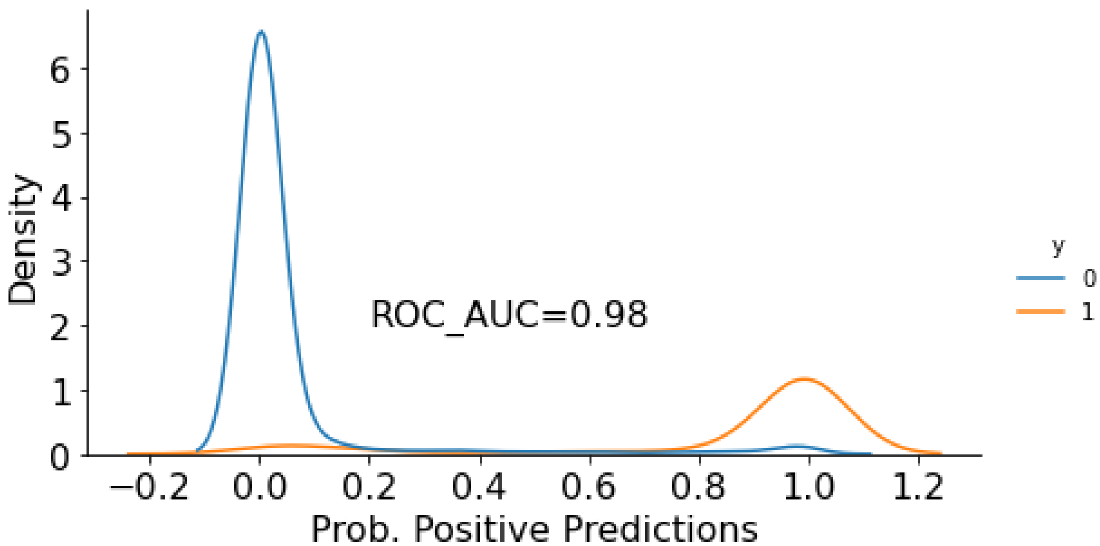
[92]:
```
df2_te_cat = encoder.transform(df2_te[cate_cols_to_keep])
df2_te_join = pd.concat( [df2_te[nume_cols], df2_te_cat], axis=1)
X_test2 = df2_te_join.values
y_test2 = df2_te["FRAUD_NONFRAUD"].values
```

[93]:
```
X_train2.shape, y_train2.shape, X_test2.shape, y_test2.shape
```

[93]:
```
((11200, 12), (11200,), (2800, 12), (2800,))
```

[94]:
```
xgb = XGBClassifier(verbosity=1,
                    max_depth=10,
                    eval_metric = "logloss")

mod6 = Model_training(xgb, X_train2, y_train2, X_test2, y_test2)
mod_trained, _ = mod5.print_metrics()
mod6.displot()
```

```
Accuracy = 92.93% F1 Score= 87.78%
Precision=90.34% Recall= 85.35%
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      1967
           1       0.90      0.85      0.88       833

    accuracy                           0.93      2800
   macro avg       0.92      0.91      0.91      2800
weighted avg       0.93      0.93      0.93      2800
```



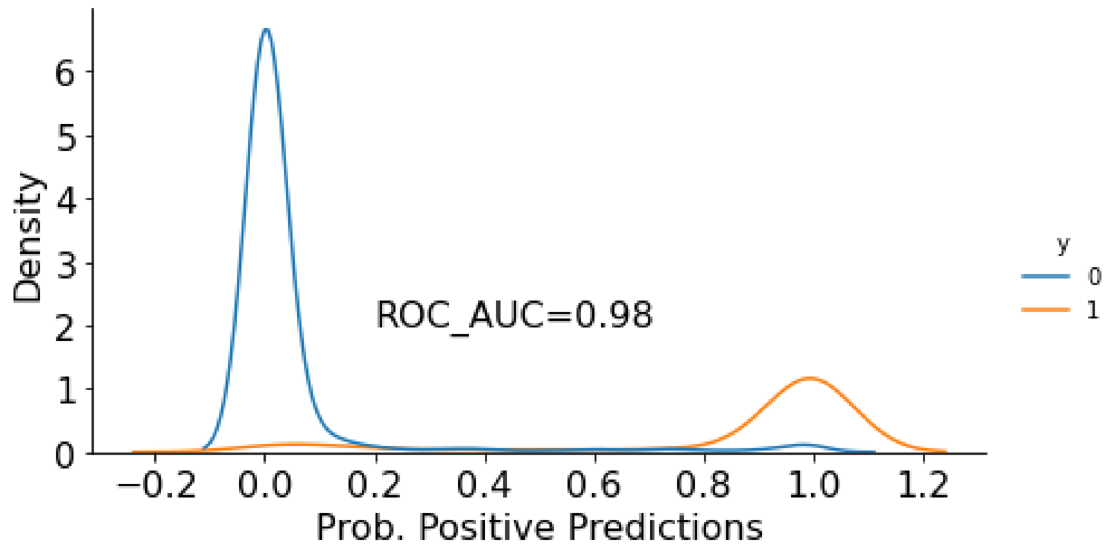
[ ]:

[96]:
```python
from sklearn.svm import SVC
```

[97]:
```python
svm = SVC(gamma='auto')
mod = Model_training(xgb_gs, X_train1, y_train1, X_test1, y_test1)
mod_tr, _= mod.print_metrics()
mod.displot()
```

```
Accuracy = 92.93% F1 Score= 87.78%
Precision=90.34% Recall= 85.35%
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      1967
           1       0.90      0.85      0.88       833

    accuracy                           0.93      2800
```

```
   macro avg       0.92      0.91      0.91      2800
weighted avg       0.93      0.93      0.93      2800
```



[98]:
```python
def train_svm_gs():
    svm_gs = GridSearchCV(SVC(),
                          param_grid={'C':[0.1, 0.5, 1.0],
                                      'kernel':['poly', 'rbf'],
                                      'gamma': ['scale', 'auto']},
                          scoring = 'f1',
                          verbose = 1 )

    mod = Model_training(svm_gs, X_train1, y_train1, X_test1, y_test1)
    mod_tr, _= mod.print_metrics()
    mod.displot()

#this takes a little long time so think before running
#train_svm_gs()
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: