00_EDA

October 12, 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

[3]:
```
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.

[4]:
```
#!pip3 install openpyxl
```

[5]:
```
# load the file
df_orig = pd.read_excel(data_dir+"trainset.xlsx", engine='openpyxl')
df_orig.head(2)
```

[5]:
```
   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]
```

[6]:
```
print ("Original data shape:", df_orig.shape)
```

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

[7]:
```
#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
```

[8]:
```python
# check the target classes
df_orig["FRAUD_NONFRAUD"].unique()
```

[8]:
```
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.

[9]:
```python
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"])
```

[10]:
```python
df = df_train0.copy()
```

[11]:
```python
df.head(2).T
```

[11]:

```
                                         2413                 1003
TRAN_AMT                               487.93                 4.84
ACCT_PRE_TRAN_AVAIL_BAL               3714.91                  0.0
CUST_AGE                                   43                   53
OPEN_ACCT_CT                                5                    5
WF_dvc_age                               1037                  305
PWD_UPDT_TS                               NaN   4/12/2017 15:54:53
CARR_NAME                                 NaN                  NaN
RGN_NAME                                  NaN                  NaN
STATE_PRVNC_TXT                           NaN                  NaN
ALERT_TRGR_CD                            MOBL                 MOBL
DVC_TYPE_TXT                              NaN               MOBILE
AUTHC_PRIM_TYPE_CD                     UN_PWD               AFA_PL
AUTHC_SCNDRY_STAT_TXT                   ALLOW                ALLOW
CUST_ZIP                                80234                75232
CUST_STATE                                 CO                   TX
PH_NUM_UPDT_TS             5/0/2020 12:33:41                   NaN
CUST_SINCE_DT            1988-01-11 00:00:00  1987-04-05 00:00:00
TRAN_TS                      4/13/2021 5:2:29   4/29/2021 22:54:53
TRAN_DT                             4/13/2021            4/29/2021
ACTN_CD                                SCHPMT               SCHPMT
ACTN_INTNL_TXT                     P2P_COMMIT           P2P_COMMIT
TRAN_TYPE_CD                              P2P                  P2P
ACTVY_DT                            4/13/2021            4/29/2021
FRAUD_NONFRAUD                          Fraud            Non-Fraud
```

[ ]:

```

```

[12]:

```
df["ACTN_CD"].value_counts()
```

[12]:

```
SCHPMT    11200
Name: ACTN_CD, dtype: int64
```

[13]:

```
df["DVC_TYPE_TXT"].value_counts()
```

[13]:

```
MOBILE     7022
DESKTOP    2413
TABLET      191
PHONE       179
Name: DVC_TYPE_TXT, dtype: int64
```

[14]:

```
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
```

1.3 Feature Engineering

- `PH_NUM_UPDT_TS` : if null replace by open account date
- length of the account (find duration since the customer_since_date)
- `TRAN_TS` - `PH_NUM_UPDT_TS` : time since the phone number was updated
- `TRAN_TS` - `PWD_UPDT_TS` : time since password was updated
- `PH_NUM_UPDT_TS` - `PWD_UPDT_TS` : time difference between phone update and password update

[15]:
```
cols_TS = [c for c in df.columns if "_TS" in c]
cols_DT = [c for c in df.columns if "_DT" in c]
cols_TS, cols_DT
```

[15]:
```
(['PWD_UPDT_TS', 'PH_NUM_UPDT_TS', 'TRAN_TS'],
 ['CUST_SINCE_DT', 'TRAN_DT', 'ACTVY_DT'])
```

[16]:
```
(df['TRAN_DT']==df['ACTVY_DT']).sum(), df['TRAN_DT'].shape[0], df['TRAN_DT'].
 ↪isnull().sum()
```

[16]:
```
(11200, 11200, 0)
```

[17]:
```
print ( df[cols_TS].info() )
df[cols_TS].head(5)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11200 entries, 2413 to 114
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   PWD_UPDT_TS     8684 non-null   object
 1   PH_NUM_UPDT_TS  5579 non-null   object
 2   TRAN_TS         11200 non-null  object
dtypes: object(3)
memory usage: 350.0+ KB
None
```

[17]:

	PWD_UPDT_TS	PH_NUM_UPDT_TS	TRAN_TS
2413	NaN	5/0/2020 12:33:41	4/13/2021 5:2:29
1003	4/12/2017 15:54:53	NaN	4/29/2021 22:54:53
8660	7/8/2021 6:28:13	NaN	1/28/2021 10:28:13
6349	11/18/2020 12:31:31	NaN	1/9/2021 23:31:31
1860	2/3/2020 9:23:53	5/29/2018 10:50:13	3/5/2021 10:23:53

[18]:
```
df[cols_DT].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11200 entries, 2413 to 114
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   CUST_SINCE_DT  11200 non-null  datetime64[ns]
 1   TRAN_DT        11200 non-null  object
 2   ACTVY_DT       11200 non-null  object
dtypes: datetime64[ns](1), object(2)
memory usage: 350.0+ KB
```

[19]:
```
def convert_future_info_to_past(df, tran_date, pwd_phn_date, cust_since_date ):
    # for cases where transaction date is after pwd_update/phone_num_update date
    # then we'd get a -ve value
    # so replace the -ve values by tran_date - cust_since_date
    return np.where((df[tran_date]-df[pwd_phn_date]).dt.days <0,
                    (df[tran_date]- df[cust_since_date]).dt.days,
                    (df[tran_date]-df[pwd_phn_date]).dt.days)
```

[20]:
```
def convert_date_format(x):
    #08/13/1989 = > 1989-08-13
    if len(str(x).split("/"))>1:
        m,d,y=str(x).strip().split()[0].split("/")
```

```
        if d=='0':
            d='1'
        elif d=='31':
            d='30'
        return "-".join([y,m,d])
    # 1989-08-13 12:30:00 => 1989-08-13
    else:
        return str(x).split()[0]


def feature_engineering(df):

    # conver the _DT columns to pandas datetime
    cols_DT = [c for c in df.columns if "_DT" in c]
    df[cols_DT] = df[cols_DT].apply(pd.to_datetime)
    # convert the TRAN_Timestamp to only hour
    df["TRAN_HOUR"]=pd.to_datetime(df['TRAN_TS']).dt.strftime("%H")
    # Fill the Nulls for Phone update by the cust_since_date and keep only date
    df["PH_NUM_UPDT_DT"]=pd.to_datetime(df["PH_NUM_UPDT_TS"].
↪fillna(df["CUST_SINCE_DT"]).apply(convert_date_format))
    # Fill the Nulls for pwd update by the cust_since_date and keep only date
    df["PWD_UPDT_DT"]=pd.to_datetime(df["PWD_UPDT_TS"].
↪fillna(df["CUST_SINCE_DT"]).apply(convert_date_format))

    df["PWD_UPDT_DAYS"] = convert_future_info_to_past(df, "TRAN_DT",
                                                       "PWD_UPDT_DT",
                                                       "CUST_SINCE_DT")



    #np.where( (df["TRAN_DT"]-df["PWD_UPDT_DT"]).dt.days <0,
    #                                (df["TRAN_DT"]- df["CUST_SINCE_DT"]).dt.
↪days,
    #                                (df["TRAN_DT"]-df["PWD_UPDT_DT"]).dt.days
    #                               )

    # Num of days between TRAN_DATE and PHONE_NUM_UPDATE_DAYS
    df["PH_NUM_UPDT_DAYS"] = convert_future_info_to_past(df, "TRAN_DT",
                                                          "PH_NUM_UPDT_DT",
                                                          "CUST_SINCE_DT")

    #np.where( (df["TRAN_DT"]-df["PH_NUM_UPDT_DT"]).dt.days <0,
    #                                   (df["TRAN_DT"]- df["CUST_SINCE_DT"]).dt.
↪days,
    #                                   (df["TRAN_DT"]-df["PH_NUM_UPDT_DT"]).dt.
↪days
    #                                  )
```

```
    # Num of days between TRAN_DATE and CUST_SINCE_DATE
    #df["TRAN_DAYS"] = (df["TRAN_DT"] - df["CUST_SINCE_DT"]).dt.days
    # Num of days between PWD update and phone number update
    #df["PH_NUM_PWD_DAYS"]=df["PH_NUM_UPDT_DAYS"] - df["PWD_UPDT_DAYS"]

    #df["RAND"] =np.random.rand(df.shape[0])
    return df
```

[37]:
```
df = feature_engineering(df)
```

[38]:
```
# 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')
```

[39]:
```
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', 'PWD_UPDT_DAYS', 'PH_NUM_UPDT_DAYS']

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', 'TRAN_HOUR', 'PH_NUM_UPDT_DT',
'PWD_UPDT_DT']
```

[40]:
```
df[nume_cols].head(2)
```

[40]:

	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	PWD_UPDT_DAYS	PH_NUM_UPDT_DAYS
2413	80234	330	347
1003	75232	1478	661

[41]:
```
nume_cols.remove('CUST_ZIP')
cate_cols.append('CUST_ZIP')
```

[42]:
```
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', 'PWD_UPDT_DAYS', 'PH_NUM_UPDT_DAYS']

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', 'TRAN_HOUR', 'PH_NUM_UPDT_DT',
'PWD_UPDT_DT', 'CUST_ZIP']
```

[45]:
```python
df[nume_cols].head(5)
```

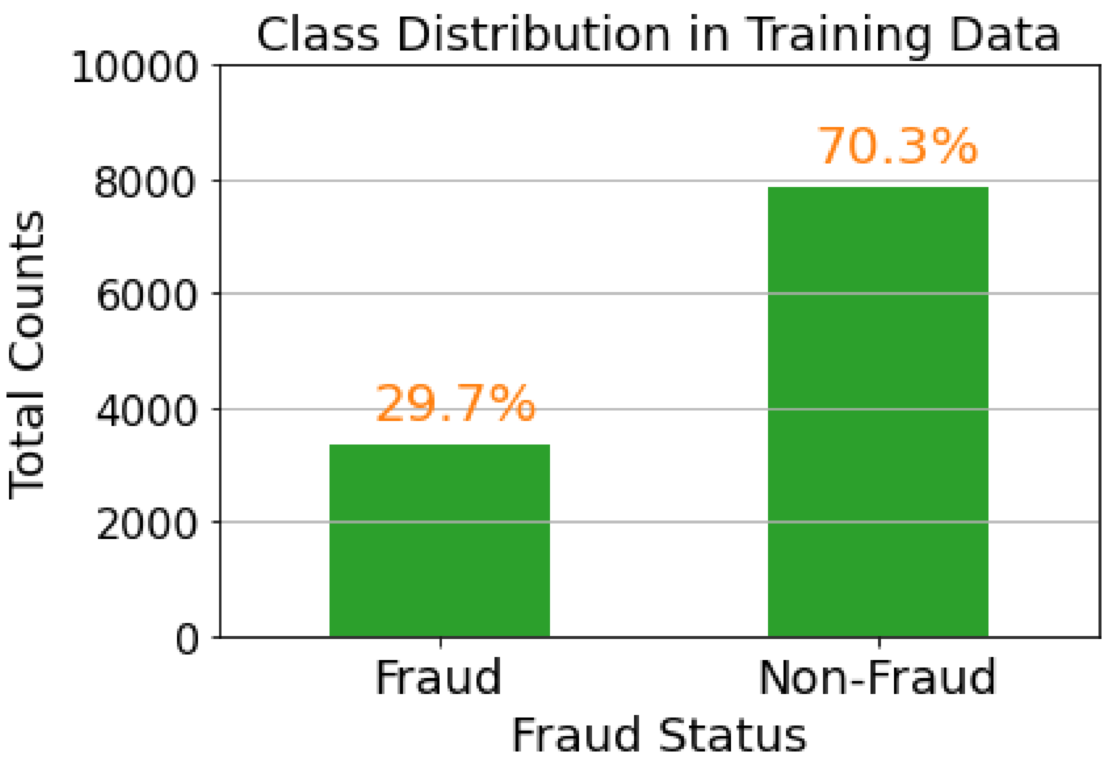
[45]:

	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	

	PWD_UPDT_DAYS	PH_NUM_UPDT_DAYS
2413	330	347
1003	1478	661
8660	9775	570
6349	52	551
1860	396	1011

1.4 Class Distribution

[44]:
```python
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')
```



1.5 Handling missing variables

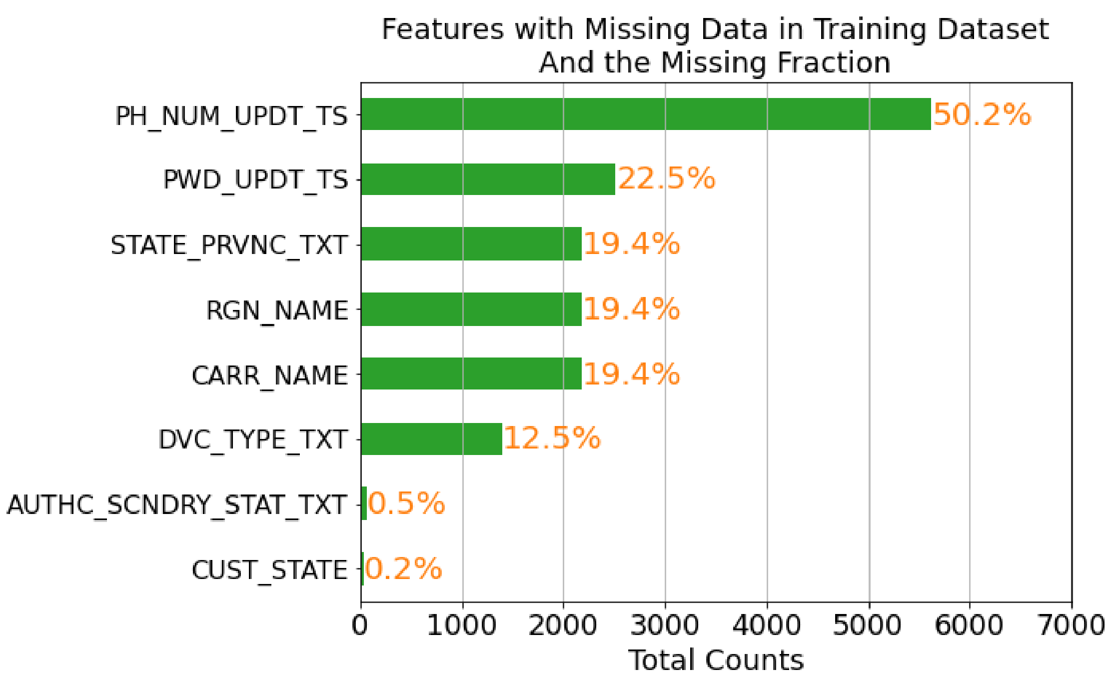
[28]:
```
df.isnull().sum()
```

[28]:
```
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
```

[29]:

```
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.

1.6 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.

[30]:
```
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
```

[30]:
```
{'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'}
```

[31]:
```
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]
```

[32]:
```
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
```

[33]:
```
# impute the columns : cols_to_impute
df=impute_data(df)
```

[34]:
```
df.isnull().sum()
```

[34]:
```
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
```

[35]:
```
df.head(2)
```

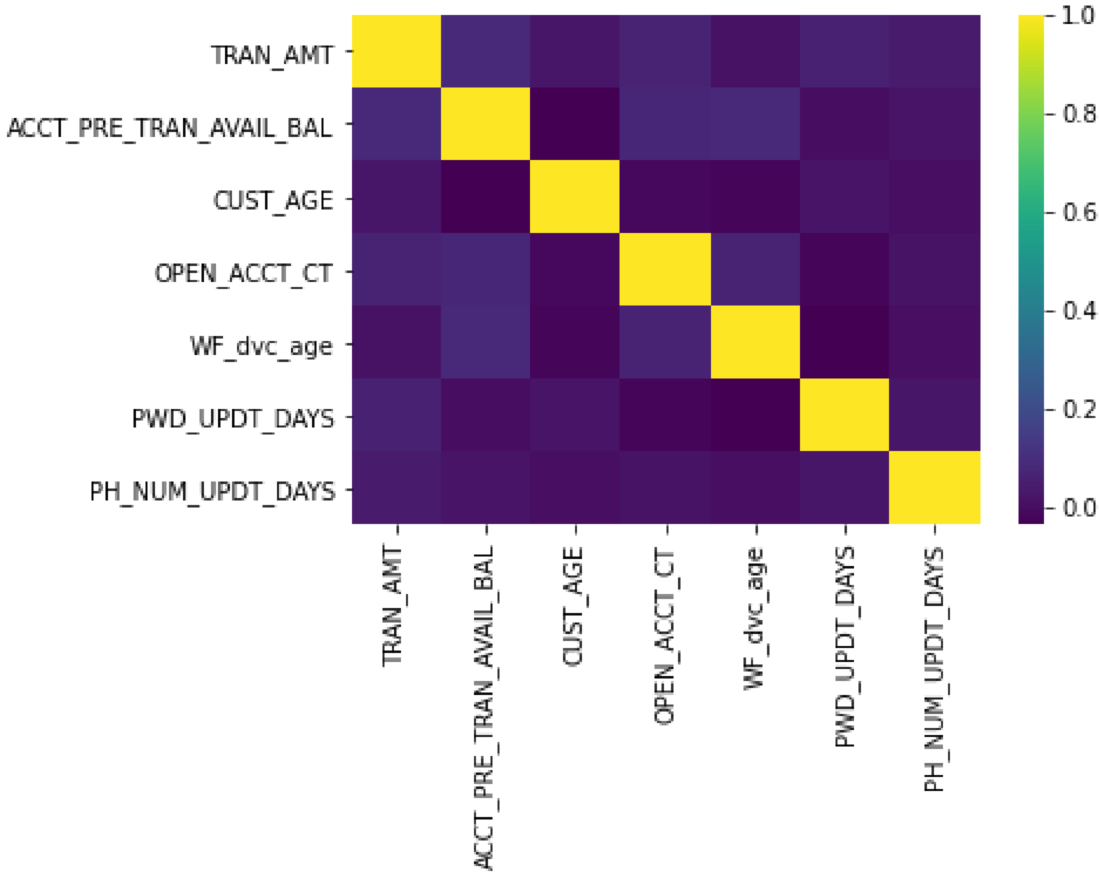
[35]:
```
      TRAN_AMT  ACCT_PRE_TRAN_AVAIL_BAL  CUST_AGE  OPEN_ACCT_CT  WF_dvc_age  \
2413    487.93                  3714.91        43             5        1037
1003      4.84                     0.00        53             5         305

             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]
```

[51]:
```
df[nume_cols].head(2)
```

[51]:
```
      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_DAYS  PH_NUM_UPDT_DAYS
2413            330               347
1003           1478               661
```

1.7 Correlation plot

[50]:
```
sns.heatmap(df[nume_cols].corr(), cmap='viridis', annot_kws={'size':20});
```

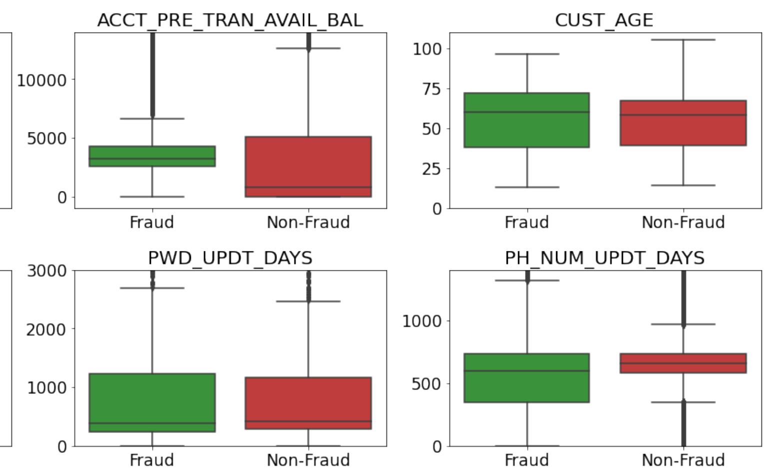


[55]:
```
cols1 = [c for c in nume_cols if c!= 'WF_dvc_age' ]
cols1
```

[55]:
```
['TRAN_AMT',
 'ACCT_PRE_TRAN_AVAIL_BAL',
 'CUST_AGE',
 'OPEN_ACCT_CT',
 'PWD_UPDT_DAYS',
 'PH_NUM_UPDT_DAYS']
```

[93]:
```
def BoxPlots():

    fig, ax = plt.subplots(2,3, figsize=(20,8))
```

```
    ylims = [ [-100, 1500], [-1000, 14000], [0,110], [0,15], [0,3000], [0,1400]␣
↪]

    for ic, col in enumerate(cols1):
        axi = ax[ic//3, ic%3]
        sns.boxplot(x="FRAUD_NONFRAUD", y=col, data=df, palette=["C2", "C3"],␣
↪ax=axi)
        axi.set_ylim(ylims[ic])
        axi.set_title(col, fontsize=20)
        axi.tick_params(axis="both", labelsize='xx-large')
        axi.set_xlabel(''); axi.set_ylabel('')

    plt.subplots_adjust(wspace=0.2, hspace=0.35)

    plt.savefig("images/boxplots.png", dpi=300, bbox_inches='tight')

BoxPlots()
```

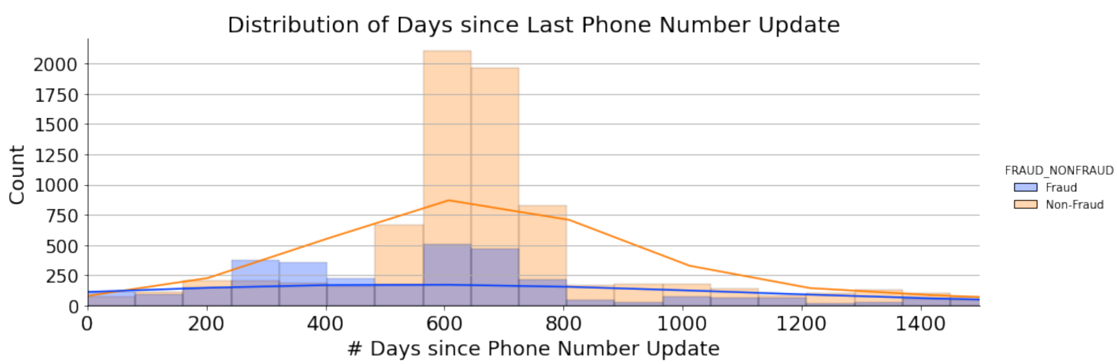


1.8 Distribution Plots

[98]:
```
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')
```

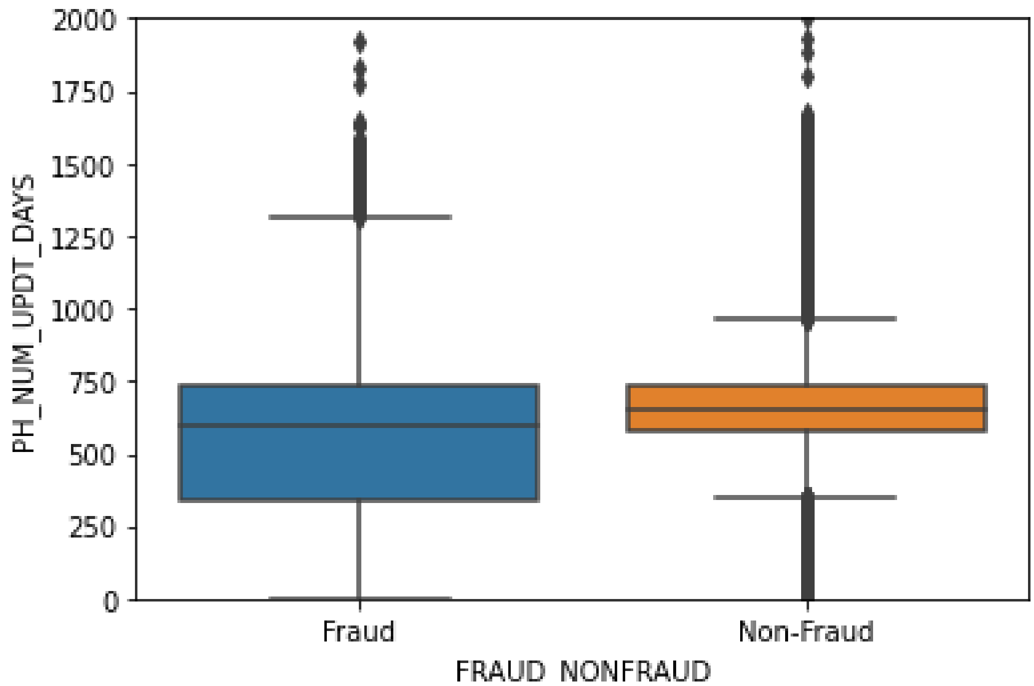
[100]:
```
df = feature_engineering(df)

xcol="PH_NUM_UPDT_DAYS"
xlabel="# Days since Phone Number Update"
title="Distribution of Days since Last Phone Number Update"
savename="dist_phone_update_days.png"
bins=500
xmax=1500
displot(df, xcol, xlabel, title, savename, bins=bins, xmax=xmax)
```



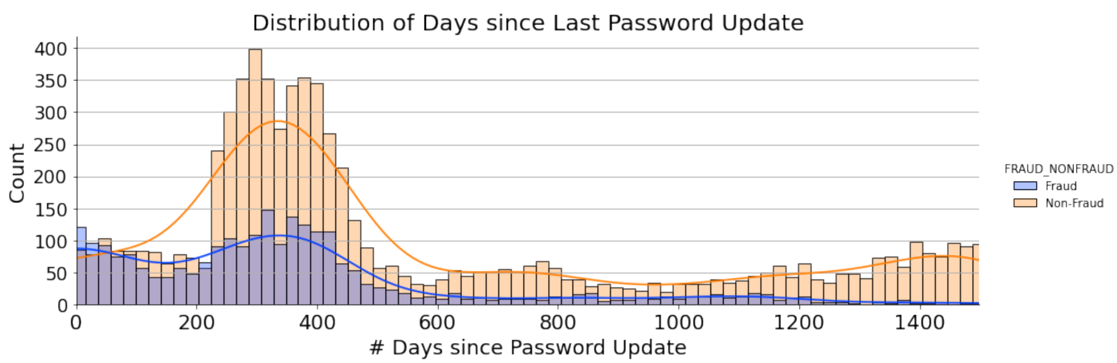
[ ]:
```
df[df["PH_NUM_UPDT_DAYS"]<0][["TRAN_DT", "CUST_SINCE_DT"]]
```

[108]:
```
sns.boxplot(data=df,
            y="PH_NUM_UPDT_DAYS",
            x="FRAUD_NONFRAUD");
plt.ylim([0, 2000])
```

[108]: (0.0, 2000.0)



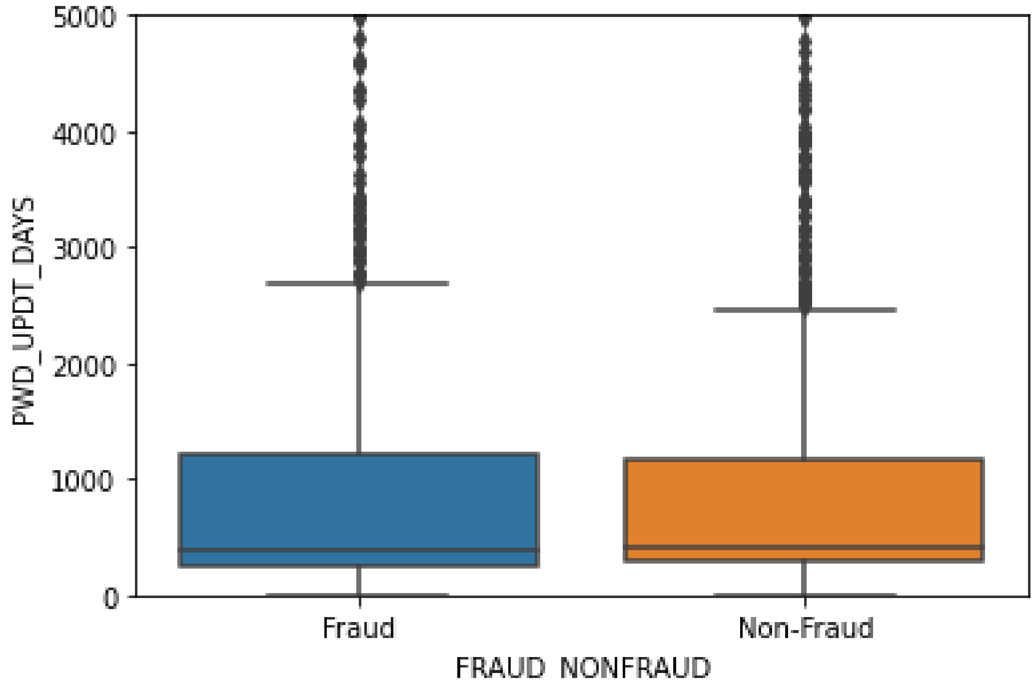
[77]:
```
xcol="PWD_UPDT_DAYS"
xlabel="# Days since Password Update"
title="Distribution of Days since Last Password Update"
savename="dist_pwd_update_days.png"
bins=100
xmax=1500
displot(df, xcol, xlabel, title, savename, bins=bins, xmax=xmax)
```



[113]:
```
sns.boxplot(data=df,
            y="PWD_UPDT_DAYS",
            x="FRAUD_NONFRAUD");
```

```
plt.ylim([0,5000])
```

[113]: (0.0, 5000.0)

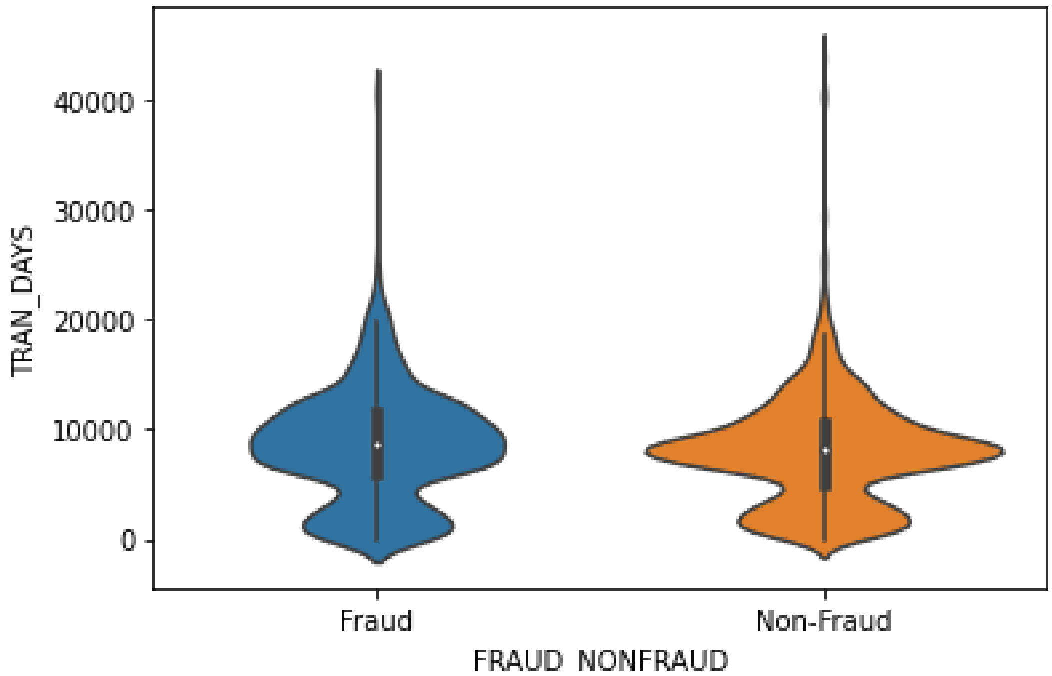


[84]:
```
df[df["TRAN_DAYS"]<0][["TRAN_DT", "CUST_SINCE_DT"]]
```

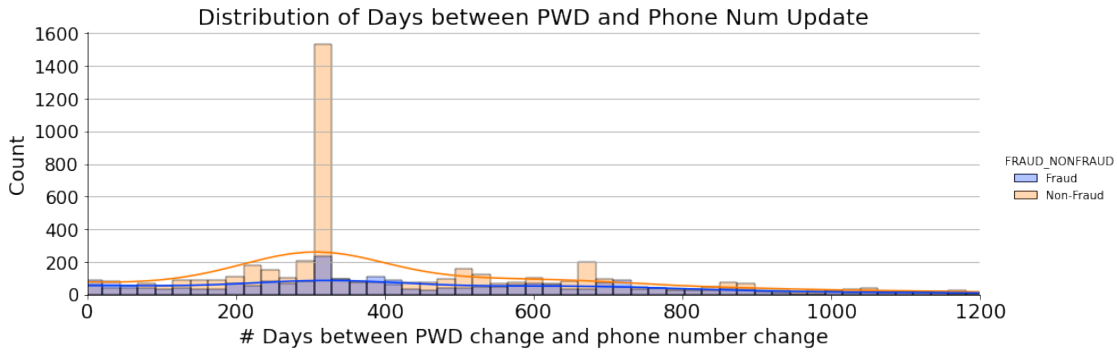
[84]:

	TRAN_DT	CUST_SINCE_DT
5350	2021-01-02	2021-01-15
3530	2021-01-05	2021-01-23

[79]:
```
sns.violinplot(data=df,
            y="TRAN_DAYS",
            x="FRAUD_NONFRAUD");
```

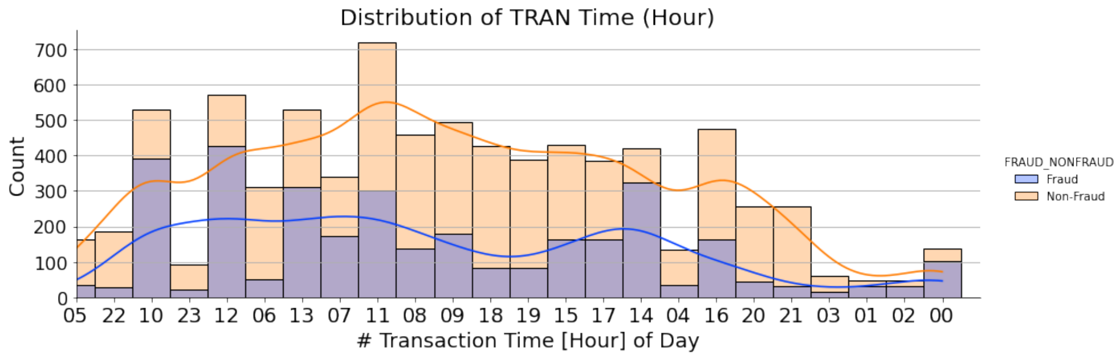


[37]:
```
xcol= "PH_NUM_PWD_DAYS"
xlabel="# Days between PWD change and phone number change"
title="Distribution of Days between PWD and Phone Num Update"
savename="dist_ph_num_pwd_update_days.png"
bins=150
xmax=1200
displot(df, xcol, xlabel, title, savename, bins=bins, xmax=xmax)
```

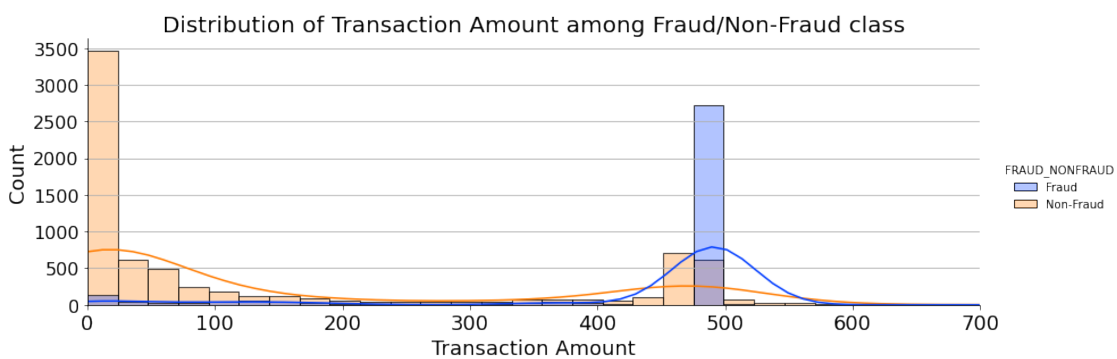


[38]:
```
xcol="TRAN_HOUR"
xlabel="# Transaction Time [Hour] of Day"
title="Distribution of TRAN Time (Hour)"
savename="dist_tran_hour.png"
```

```
bins=100
xmax=24
displot(df, xcol, xlabel, title, savename, bins=bins, xmax=xmax)
```



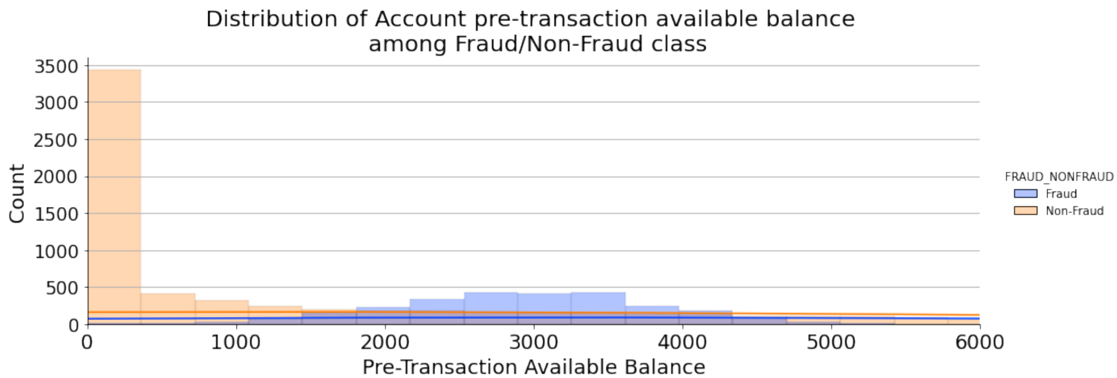
[39]:
```
xcol="TRAN_AMT"
xlabel="Transaction Amount"
title="Distribution of Transaction Amount among Fraud/Non-Fraud class"
savename="dist_trans_amnt.png"
bins=100
xmax=700
displot(df, xcol, xlabel, title, savename, bins=bins, xmax=xmax)
```



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

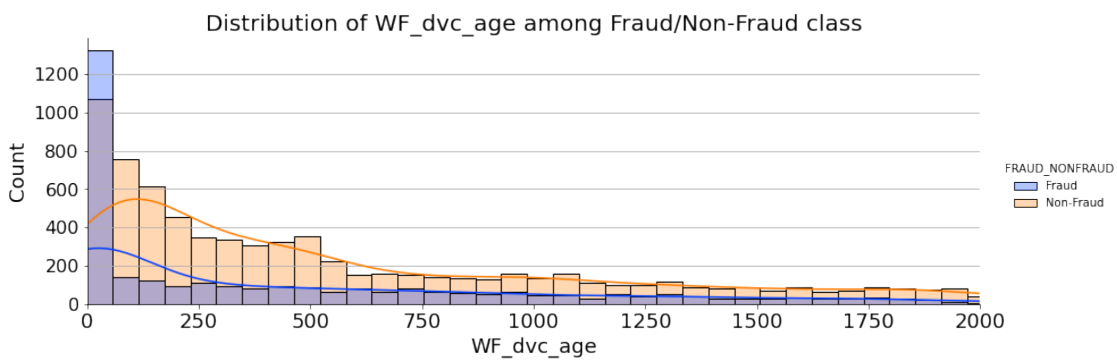
[40]:
```
xcol="ACCT_PRE_TRAN_AVAIL_BAL"
xlabel="Pre-Transaction Available Balance"
title="Distribution of Account pre-transaction available balance \n among Fraud/
↪Non-Fraud class"
savename="dist_pre_trans_blnce.png"
bins=1000
```

```
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.

[41]:
```
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]

[42]:

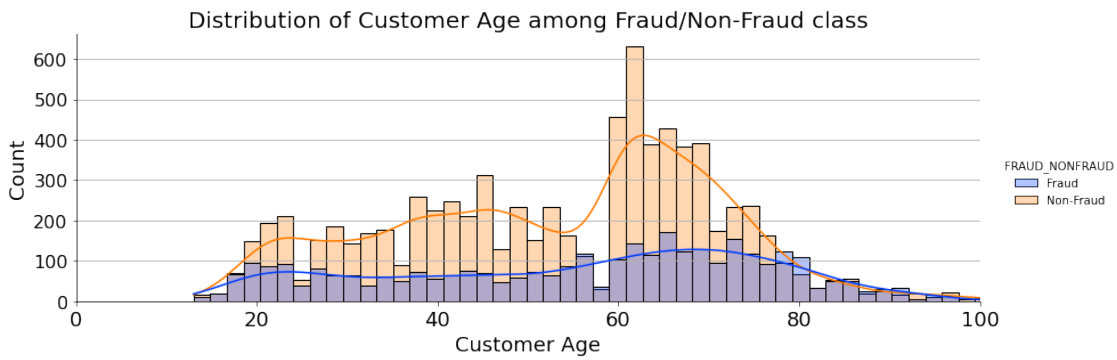
```
nume_cols
```

[42]:

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

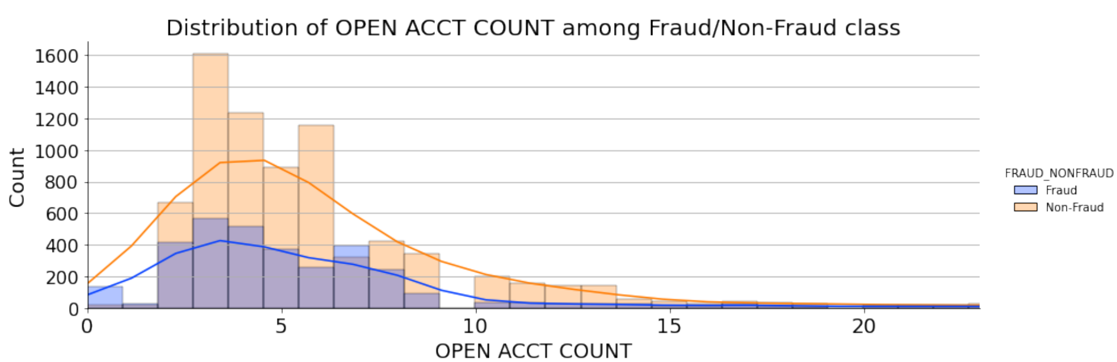
[43]:

```
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)
```



[44]:

```
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)
```



1.9 Categorical Features

[45]:
```
# 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:      283
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:      283
CUST_ZIP        # of Unique values:      3750
```

[46]:
```
# 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': 283, 'ACTVY_DT': 283,
'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.

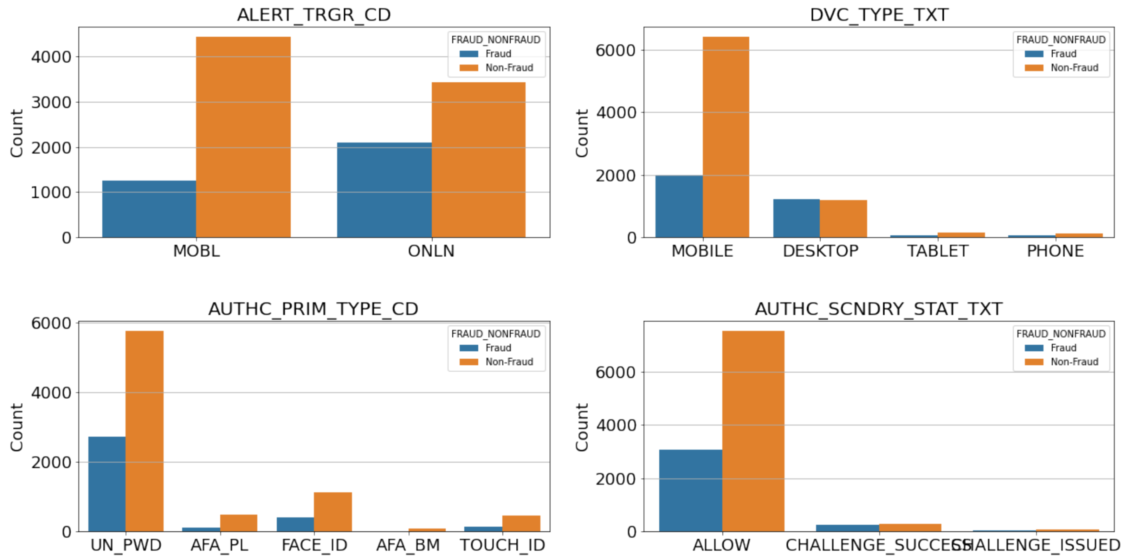
For `TRAN_TS`, do more analysis as in why it’s ogt so may unique values.

[47]:
```
d0
```

[47]:
```
{'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}
```

[48]:
```
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);
```

```python
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.

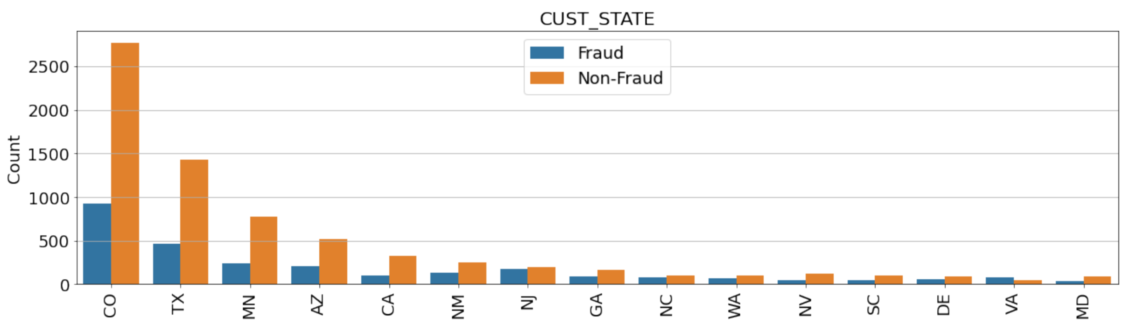
[49]:
```python
d1
```

[49]: {'CUST_STATE': 48}

[50]:
```python
def plot_count_plot(col, df=df, savename=None ):

    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);
    if not savename:
        savename="dist_"+col
    figname = "images/"+savename+".png"
    plt.savefig(figname, dpi=300, bbox_inches='tight')
```

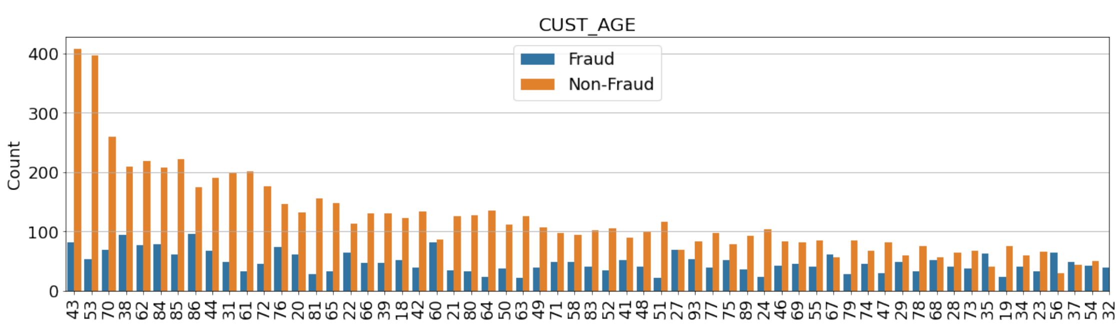
[51]:
```
xcol="CUST_STATE"
plot_count_plot(xcol, df=df, savename="CUST_STATE_before" )
plt.xlim([-.5,14.5]);
```



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

[52]:
```
xcol="CUST_AGE"
plot_count_plot(xcol, df=df )
plt.xlim([-.5,60])
```

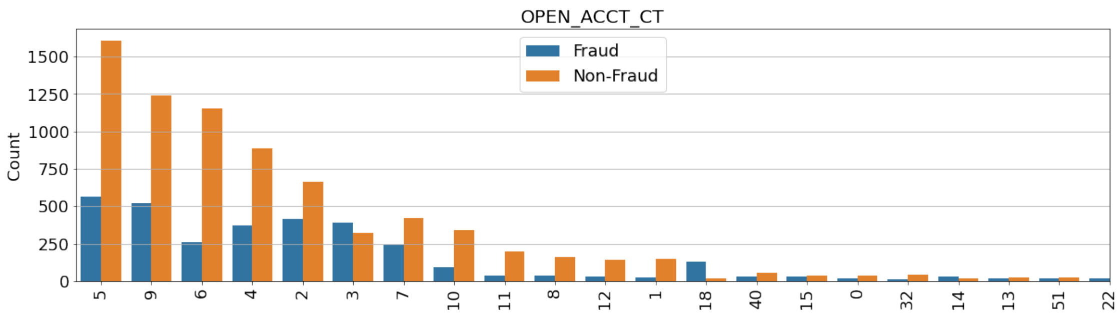
[52]: (-0.5, 60.0)



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

[53]:
```
xcol="OPEN_ACCT_CT"
plot_count_plot(xcol, df=df )
plt.xlim([-.5,20])
```

[53]: (-0.5, 20.0)



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

[54]:
```python
# categories with more than 100 unique values
d2
```

[54]:
```
{'CUST_SINCE_DT': 7431,
 'TRAN_TS': 10871,
 'TRAN_DT': 283,
 'ACTVY_DT': 283,
 'CUST_ZIP': 3750}
```

[55]:
```python
# check whether 'TRAN_DT' and 'ACTVY_DT' are same columns
(df['TRAN_DT']==df['ACTVY_DT']).sum()/df.shape[0]
```

[55]: 1.0

[56]:
```python
from datetime import datetime
```

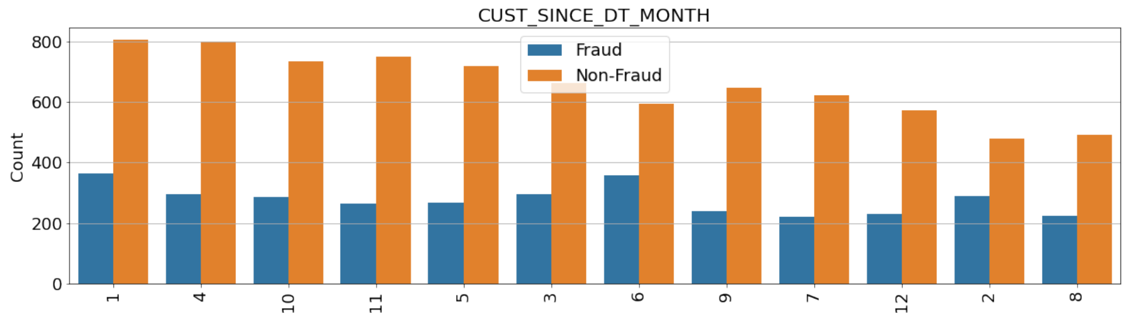
[57]:
```python
df['ACTVY_DT'] = pd.to_datetime(df['ACTVY_DT'])
df['ACTVY_DT_DAY'] = df['ACTVY_DT'].apply(lambda x: x.day)
df['ACTVY_DT_MONTH'] = df['ACTVY_DT'].apply(lambda x: x.month)
df['ACTVY_DT_YEAR'] = df['ACTVY_DT'].apply(lambda x: x.year)
```

[58]:
```python
d2
```

[58]:
```
{'CUST_SINCE_DT': 7431,
 'TRAN_TS': 10871,
 'TRAN_DT': 283,
 'ACTVY_DT': 283,
 'CUST_ZIP': 3750}
```

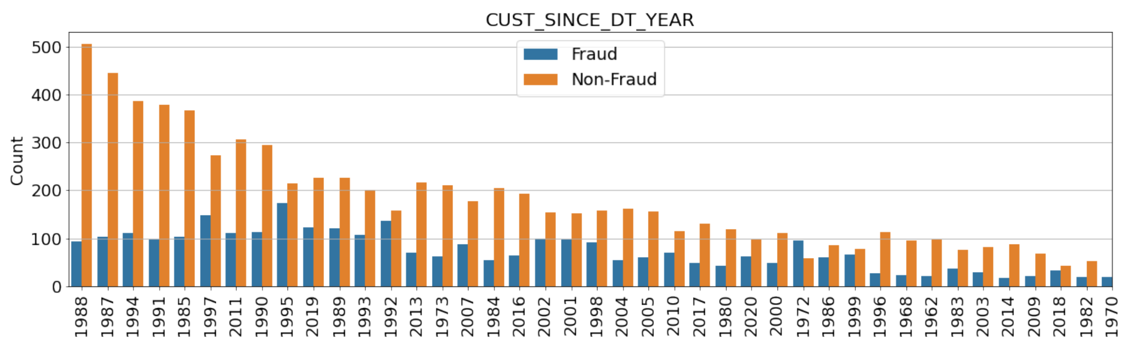
[59]:
```python
df["CUST_SINCE_DT_YEAR"]=df["CUST_SINCE_DT"].apply(lambda x: x.year)
df["CUST_SINCE_DT_MONTH"] = df["CUST_SINCE_DT"].apply(lambda x: x.month)
```

[60]:
```
xcol="CUST_SINCE_DT_MONTH"
plot_count_plot(xcol, df=df )
```



[61]:
```
xcol="CUST_SINCE_DT_YEAR"
plot_count_plot(xcol, df=df )
plt.xlim([-.5,40])
```

[61]: (-0.5, 40.0)



1.10 Feature Transformation

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

[62]:
```
def transform_cate_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
```

[63]:
```python
df=transform_cate_data(df)
```

[64]:
```python
nume_cols
```

[64]:
```
['TRAN_AMT',
 'ACCT_PRE_TRAN_AVAIL_BAL',
 'CUST_AGE',
 'OPEN_ACCT_CT',
 'WF_dvc_age']
```

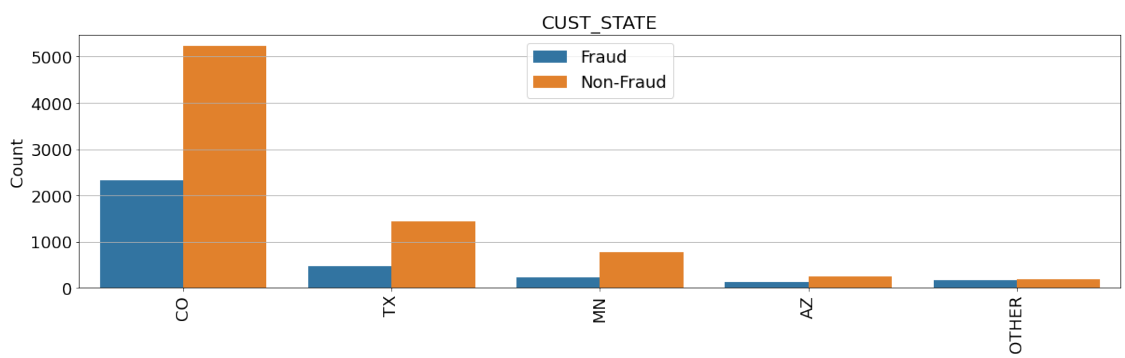
[65]:
```python
cate_cols_to_keep = ['ALERT_TRGR_CD', "CUST_STATE"]
```

[66]:
```python
df[cate_cols_to_keep].head()
```

[66]:

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

[67]:
```python
xcol="CUST_STATE"
plot_count_plot(xcol, df=df, savename="CUST_STATE_after" )

plt.xlim([-.5,4.5]);
```



[68]:
```python
[i for i in range(0,25+1,5)]
```

[68]:
```
[0, 5, 10, 15, 20, 25]
```

[ ]:
```python

```

[ ]:
```python

```