# Credit Card Fraud Detection

**Goal:**

Predict the probability of an online credit card transaction being fraudulent, based on different properties of the transactions.

# 1. Setup Environment

In [1]:
```
# Data Manipulation
import numpy as np
import pandas as pd

# Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.lines as mlines

# Time
import time
import datetime

# Machine Learning
from   sklearn.preprocessing import LabelEncoder, minmax_scale
from   sklearn.ensemble import RandomForestClassifier
from   sklearn.decomposition import PCA
from   sklearn.model_selection import train_test_split, GridSearchCV
from   sklearn.metrics import (confusion_matrix , classification_report, accur
                               plot_roc_curve, precision_recall_curve, plot_pr
from   sklearn.calibration import calibration_curve
from   sklearn.calibration import CalibratedClassifierCV

from   xgboost import XGBClassifier
from   lightgbm import LGBMClassifier

from   imblearn.over_sampling import RandomOverSampler
from   scipy.stats import chi2_contingency,  f_oneway

import gc
import warnings
from   tqdm import tqdm


# Set Options
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', 500)
%matplotlib inline
warnings.filterwarnings("ignore")
```

# 2. Data Overview

In [2]:
```
%%time
df_id   = pd.read_csv(r'C:\Users\admin\Desktop\PYTHON LEARNING\Data_Projects_0
df_tran = pd.read_csv(r'C:\Users\admin\Desktop\PYTHON LEARNING\Data_Projects_0
```

```
Wall time: 26.5 s
```

In [3]:
```
df_id.head()
```

Out[3]:

| | TransactionID | id_01 | id_02 | id_03 | id_04 | id_05 | id_06 | id_07 | id_08 | id_09 | id_10 | id_11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2987004 | 0.0 | 70787.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 100.0 | Nc |
| **1** | 2987008 | -5.0 | 98945.0 | NaN | NaN | 0.0 | -5.0 | NaN | NaN | NaN | NaN | 100.0 | Nc |
| **2** | 2987010 | -5.0 | 191631.0 | 0.0 | 0.0 | 0.0 | 0.0 | NaN | NaN | 0.0 | 0.0 | 100.0 | Nc |
| **3** | 2987011 | -5.0 | 221832.0 | NaN | NaN | 0.0 | -6.0 | NaN | NaN | NaN | NaN | 100.0 | Nc |
| **4** | 2987016 | 0.0 | 7460.0 | 0.0 | 0.0 | 1.0 | 0.0 | NaN | NaN | 0.0 | 0.0 | 100.0 | Nc |

## Identity Data Description

Variables in this table are identity information – network connection information (IP, ISP, Proxy, etc) and digital signature (UA/browser/os/version, etc) associated with transactions. They're collected by Vesta's fraud protection system and digital security partners. (The field names are masked and pairwise dictionary will not be provided for privacy protection and contract agreement)

Categorical Features:

- DeviceType
- DeviceInfo
- id_12 - id_38

In [4]:
```
df_tran.head()
```

Out[4]:

| | TransactionID | isFraud | TransactionDT | TransactionAmt | ProductCD | card1 | card2 | card3 | ca |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2987000 | 0 | 86400 | 68.5 | W | 13926 | NaN | 150.0 | disc |
| **1** | 2987001 | 0 | 86401 | 29.0 | W | 2755 | 404.0 | 150.0 | master |
| **2** | 2987002 | 0 | 86469 | 59.0 | W | 4663 | 490.0 | 150.0 | |
| **3** | 2987003 | 0 | 86499 | 50.0 | W | 18132 | 567.0 | 150.0 | master |
| **4** | 2987004 | 0 | 86506 | 50.0 | H | 4497 | 514.0 | 150.0 | master |

## Transaction Data Description

- **TransactionDT**: timedelta from a given reference datetime (not an actual timestamp)
- **TransactionAMT**: transaction payment amount in USD
- **ProductCD**: product code, the product for each transaction
- **card1 - card6**: payment card information, such as card type, card category, issue bank, country, etc.
- **addr**: address
- **dist**: distance
- **P_ and (R__) emaildomain**: purchaser and recipient email domain
- **C1-C14**: counting, such as how many addresses are found to be associated with the payment card, etc. The actual meaning is masked.
- **D1-D15**: timedelta, such as days between previous transaction, etc.
- **M1-M9**: match, such as names on card and address, etc.
- **Vxxx**: Vesta engineered rich features, including ranking, counting, and other entity relations.

# 3. Optimize Memory Used by Data

Memory occupied by the dataframe (in mb)

In [5]:
```
df_id.memory_usage(deep=True).sum() / 1024**2
```

Out[5]:
```
157.63398933410645
```

In [6]:
```
df_tran.memory_usage(deep=True).sum() / 1024**2
```

Out[6]:
```
2100.701406478882
```

In [7]:
```
print('int64 min: ', np.iinfo(np.int64).min)
print('int64 max: ', np.iinfo(np.int64).max)
```

```
int64 min:  -9223372036854775808
int64 max:  9223372036854775807
```

In [8]:
```
print('int8 min: ', np.iinfo(np.int8).min)
print('int8 max: ', np.iinfo(np.int8).max)
```

```
int8 min:  -128
int8 max:  127
```

In [9]:
```
# Reduce memory usage
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage(deep=True).sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8)
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.in
                    df[col] = df[col].astype(np.int16)
```

```
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.in
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.in
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.fl
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage(deep=True).sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'
    return df
```

In [10]:
```
# Reduce the memory size of the dataframe
df_id   = reduce_mem_usage(df_id)
df_tran = reduce_mem_usage(df_tran)
```

```
Mem. usage decreased to 138.38 Mb (12.2% reduction)
Mem. usage decreased to 867.89 Mb (58.7% reduction)
```

# 4. Basic Data Stats

## Shape of dataframe

In [11]:
```
df_id.shape
```

Out[11]:
```
(144233, 41)
```

In [12]:
```
df_tran.shape
```

Out[12]:
```
(590540, 394)
```

### Check how many transactions has ID info

In [13]:
```
df_tran.TransactionID.isin(df_id.TransactionID).sum()
```

Out[13]:
```
144233
```

## Summary of dataframe

In [14]:
```
from pandas_summary import DataFrameSummary
df_id_summary = DataFrameSummary(df_id)
df_id_summary.summary()
```

Out[14]:

| | TransactionID | id_01 | id_02 | id_03 | id_04 | id_05 | id_06 | id |
|---|---|---|---|---|---|---|---|---|
| **count** | 144233.0 | 144233.0 | 140872.0 | 66324.0 | 66324.0 | 136865.0 | 136865.0 | 51 |
| **mean** | 3236329.311288 | NaN | 174716.59375 | 0.0 | -0.0 | NaN | NaN | |
| **std** | 178849.571186 | 0.0 | 159651.8125 | 0.0 | 0.0 | 0.0 | 0.0 | 11.382 |
| **min** | 2987004.0 | -100.0 | 1.0 | -13.0 | -28.0 | -72.0 | -100.0 | -4 |
| **25%** | 3077142.0 | -10.0 | 67992.0 | 0.0 | 0.0 | 0.0 | -6.0 | |

| | TransactionID | id_01 | id_02 | id_03 | id_04 | id_05 | id_06 | id |
|---|---|---|---|---|---|---|---|---|
| **50%** | 3198818.0 | -5.0 | 125800.5 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **75%** | 3392923.0 | -5.0 | 228749.0 | 0.0 | 0.0 | 1.0 | 0.0 | |
| **max** | 3577534.0 | 0.0 | 999595.0 | 10.0 | 0.0 | 52.0 | 0.0 | |
| **counts** | 144233 | 144233 | 140872 | 66324 | 66324 | 136865 | 136865 | 5 |
| **uniques** | 144233 | 77 | 115655 | 24 | 15 | 93 | 101 | |
| **missing** | 0 | 0 | 3361 | 77909 | 77909 | 7368 | 7368 | 139 |
| **missing_perc** | 0% | 0% | 2.33% | 54.02% | 54.02% | 5.11% | 5.11% | 96.4 |
| **types** | numeric | numeric | numeric | numeric | numeric | numeric | numeric | num |

In [15]:
```
df_tran.describe()
```

Out[15]:

| | TransactionID | isFraud | TransactionDT | TransactionAmt | card1 | card2 | c |
|---|---|---|---|---|---|---|---|
| **count** | 5.905400e+05 | 590540.000000 | 5.905400e+05 | 590540.000000 | 590540.000000 | 581607.0 | 5889 |
| **mean** | 3.282270e+06 | 0.034990 | 7.372311e+06 | NaN | 9898.734658 | NaN | |
| **std** | 1.704744e+05 | 0.183755 | 4.617224e+06 | NaN | 4901.170153 | NaN | |
| **min** | 2.987000e+06 | 0.000000 | 8.640000e+04 | 0.250977 | 1000.000000 | 100.0 | 1 |
| **25%** | 3.134635e+06 | 0.000000 | 3.027058e+06 | 43.312500 | 6019.000000 | 214.0 | 1 |
| **50%** | 3.282270e+06 | 0.000000 | 7.306528e+06 | 68.750000 | 9678.000000 | 361.0 | 1 |
| **75%** | 3.429904e+06 | 0.000000 | 1.124662e+07 | 125.000000 | 14184.000000 | 512.0 | 1 |
| **max** | 3.577539e+06 | 1.000000 | 1.581113e+07 | 31936.000000 | 18396.000000 | 600.0 | 2 |

### Check class imbalance

In [16]:
```
df_tran.loc[:, 'isFraud'].value_counts()
```

Out[16]:
```
0    569877
1     20663
Name: isFraud, dtype: int64
```

In [17]:
```
df_tran.loc[:, 'isFraud'].value_counts(normalize=True)*100
```

Out[17]:
```
0    96.500999
1     3.499001
Name: isFraud, dtype: float64
```

Lot of interesting things can be observed here:

- Rows in identity dataset are less than transaction dataset, that means only a subset of transactions in transactions dataset has identity data
- Both datasets have the common and unique key as TransactionID, both can be joined at this unique key
- id_24, id_25, dist2, D7 and many more columns have 90%+ missing values, which means that these columns are probably useless so need to drop it for now

- Columns from V1 to V339 in transaction dataset are numeric whereas columns from id_01 to id_39 are of mixed datatype
- TransactionDT column is a timedelta from a given reference datetime (not an actual timestamp). But reference datetime is not known, so need to assume it and convert it to date format
- Target class is imbalanced. So no need to drop the columns where one category contains the majority of rows

# 5. Data Preprocessing for EDA

## Merge the datasets

In [18]:
```
df = df_tran.merge(df_id, how='left', on='TransactionID')

del df_tran, df_id

gc.collect()
```

Out[18]:
```
0
```

In [19]:
```
df.shape
```

Out[19]:
```
(590540, 434)
```

## Add missing flag

In [20]:
```
for col in df.columns:
    df[col+"_missing_flag"] = df[col].isnull()

df.head()
```

Out[20]:

| | TransactionID | isFraud | TransactionDT | TransactionAmt | ProductCD | card1 | card2 | card3 | ca |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2987000 | 0 | 86400 | 68.5 | W | 13926 | NaN | 150.0 | disc |
| **1** | 2987001 | 0 | 86401 | 29.0 | W | 2755 | 404.0 | 150.0 | master |
| **2** | 2987002 | 0 | 86469 | 59.0 | W | 4663 | 490.0 | 150.0 | |
| **3** | 2987003 | 0 | 86499 | 50.0 | W | 18132 | 567.0 | 150.0 | master |
| **4** | 2987004 | 0 | 86506 | 50.0 | H | 4497 | 514.0 | 150.0 | master |

5 rows × 868 columns

## Clean Data

Let's drop the columns which may not be useful for our analysis

Create a missing value flag column for the columns we are dropping which have more than 90% missing values, there might be some specific pattern associated with missing values and transaction being fraud

In [21]:
```
drop_cols = []

for col in df.columns:
    missing_share = df[col].isnull().sum()/df.shape[0]
    if missing_share > 0.9:
        drop_cols.append(col)
        print(col)
        # df[col + "_missing_flag"] = df[col].isnull()

good_cols = [col for col in df.columns if col not in drop_cols]
```

```
dist2
D7
id_07
id_08
id_18
id_21
id_22
id_23
id_24
id_25
id_26
id_27
```

In [22]:
```
drop_cols = []
for col in good_cols:
    unique_value = df[col].nunique()
    if unique_value == 1:
        drop_cols.append(col)
        print(col)
good_cols = [col for col in good_cols if col not in drop_cols]
```

```
TransactionID_missing_flag
isFraud_missing_flag
TransactionDT_missing_flag
TransactionAmt_missing_flag
ProductCD_missing_flag
card1_missing_flag
C1_missing_flag
C2_missing_flag
C3_missing_flag
C4_missing_flag
C5_missing_flag
C6_missing_flag
C7_missing_flag
C8_missing_flag
C9_missing_flag
C10_missing_flag
C11_missing_flag
C12_missing_flag
C13_missing_flag
C14_missing_flag
```

In [23]:
```
# Filter the data for relevant columns only
df = df[good_cols]
```

In [24]:
```
df.shape
```

Out[24]:
```
(590540, 836)
```

### Create date features

In [25]:
```
START_DATE         = '2017-12-01'
startdate          = datetime.datetime.strptime(START_DATE, "%Y-%m-%d")
df["Date"]         = df['TransactionDT'].apply(lambda x: (startdate + datetime

df['_Weekdays']    = df['Date'].dt.dayofweek
df['_Hours']       = df['Date'].dt.hour
df['_Days']        = df['Date'].dt.day
```

In [26]:
```
df = reduce_mem_usage(df)
```

```
Mem. usage decreased to 1453.88 Mb (0.8% reduction)
```

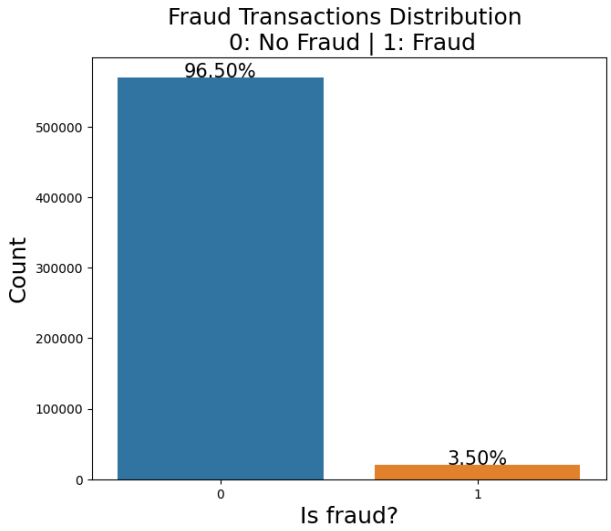
# 6. Exploratory Data Analysis

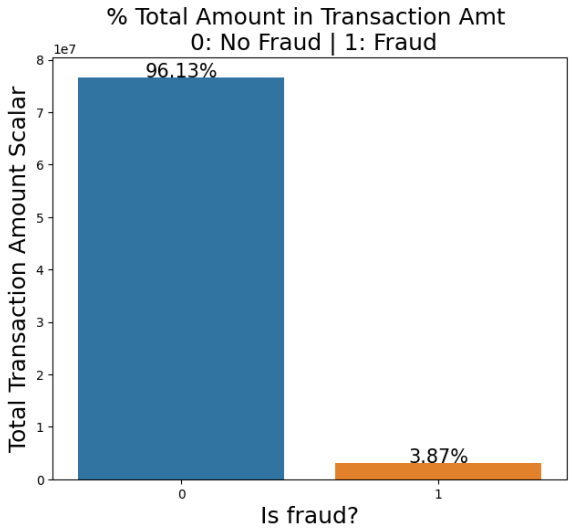
## Check distribution of target variable

In [27]:
```
df['isFraud'].value_counts()
```

Out[27]:
```
0    569877
1     20663
Name: isFraud, dtype: int64
```

In [28]:
```
df['TransactionAmt'] = df['TransactionAmt'].astype(float)
total = len(df)
total_amt = df.groupby(['isFraud'])['TransactionAmt'].sum().sum()
plt.figure(figsize=(16,6))

plt.subplot(121)
g = sns.countplot(x='isFraud', data=df )
g.set_title("Fraud Transactions Distribution \n 0: No Fraud | 1: Fraud", fonts
g.set_xlabel("Is fraud?", fontsize=18)
g.set_ylabel('Count', fontsize=18)
for p in g.patches:
    height = p.get_height()
    g.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/total*100),
            ha="center", fontsize=15)

perc_amt = (df.groupby(['isFraud'])['TransactionAmt'].sum())
perc_amt = perc_amt.reset_index()

plt.subplot(122)
g1 = sns.barplot(x='isFraud', y='TransactionAmt',  dodge=True, data=perc_amt)
g1.set_title("% Total Amount in Transaction Amt \n 0: No Fraud | 1: Fraud", fo
g1.set_xlabel("Is fraud?", fontsize=18)
g1.set_ylabel('Total Transaction Amount Scalar', fontsize=18)
for p in g1.patches:
    height = p.get_height()
    g1.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/total_amt * 100),
            ha="center", fontsize=15)

plt.show()
```




In [29]:
```
df.groupby('isFraud')['TransactionAmt'].mean()
```

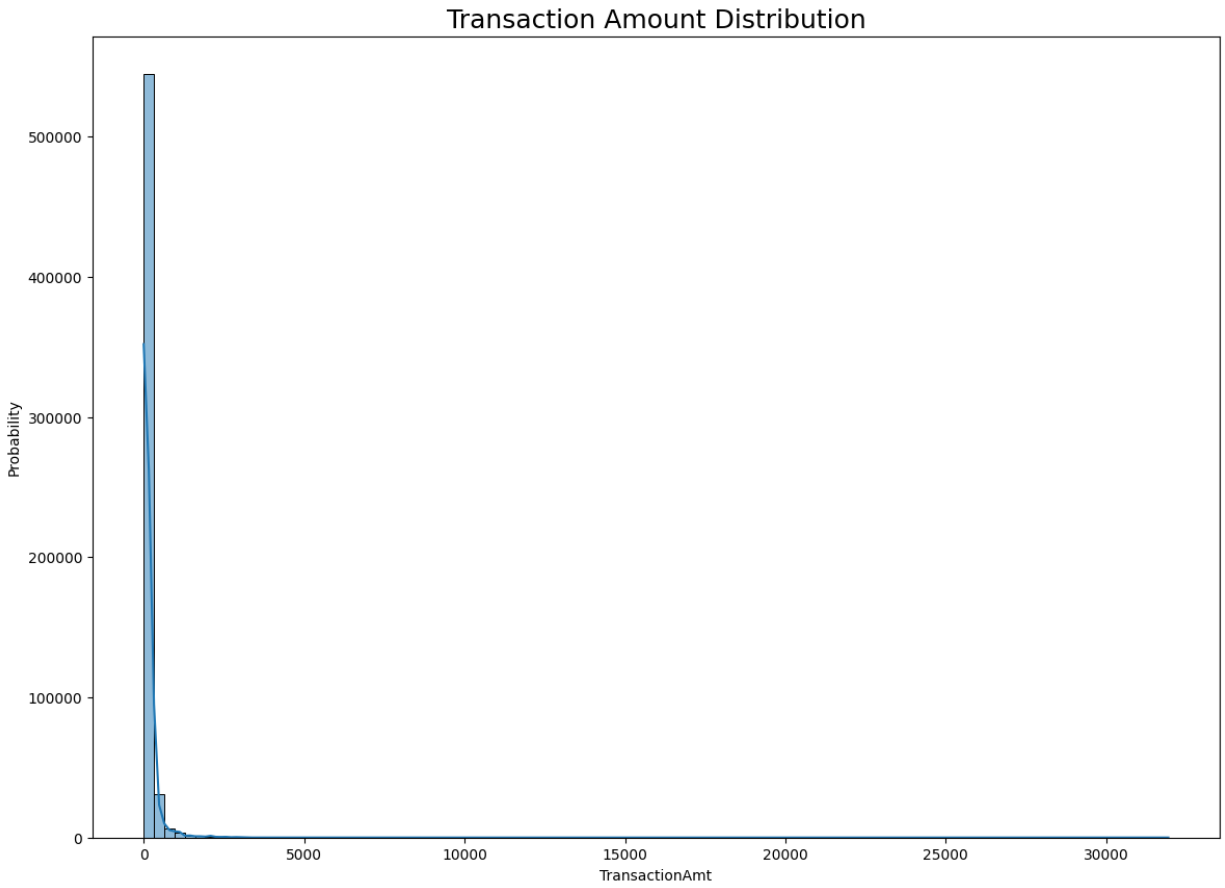
Out[29]:
```
isFraud
0    134.511857
1    149.244353
Name: TransactionAmt, dtype: float64
```

- The target variable is **imbalanced**. 3.5% transactions are Fraud
- Around same % of transaction amounts are fraud

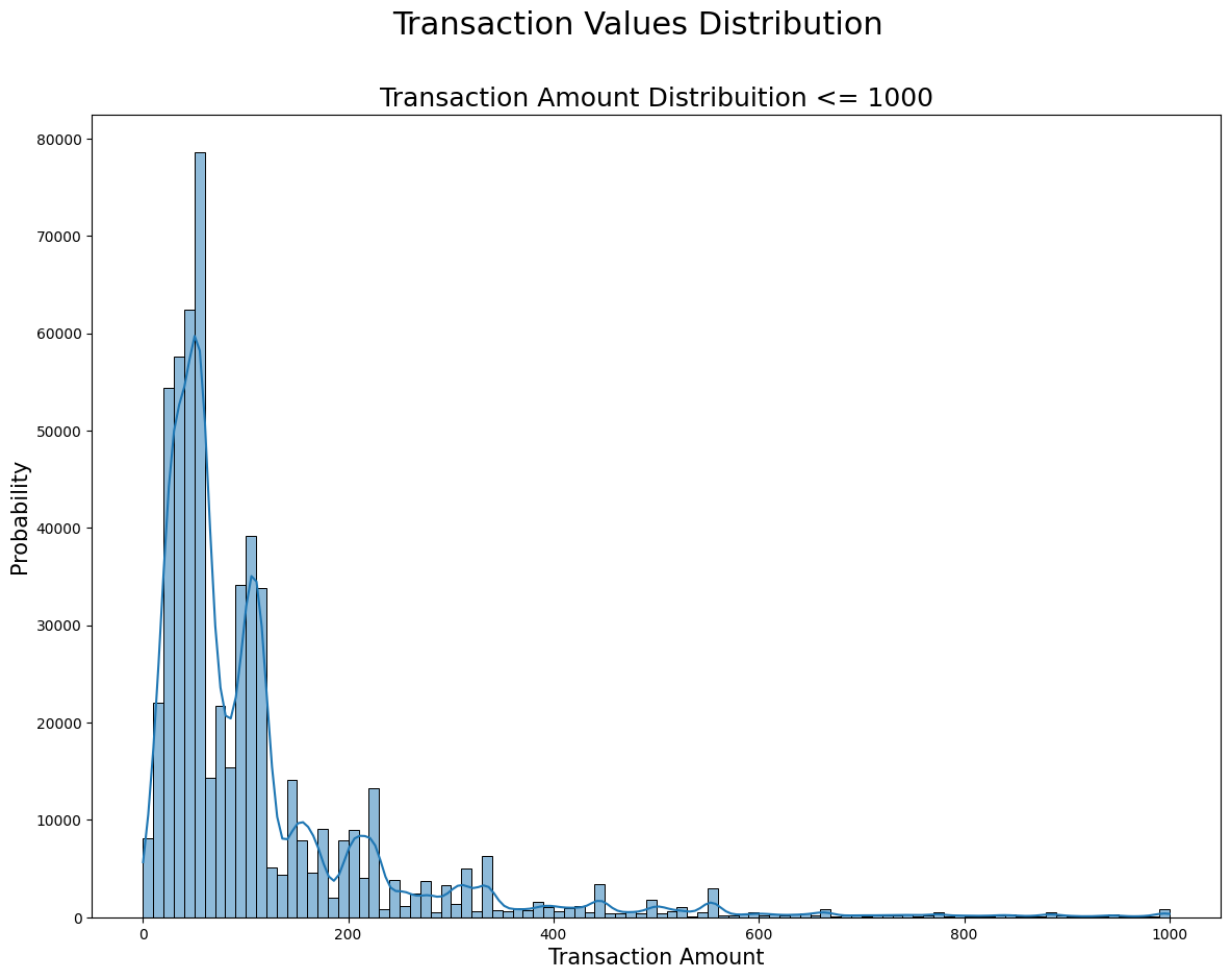
Let's explore the Transaction amount further

## Check distribution of Transaction Amount

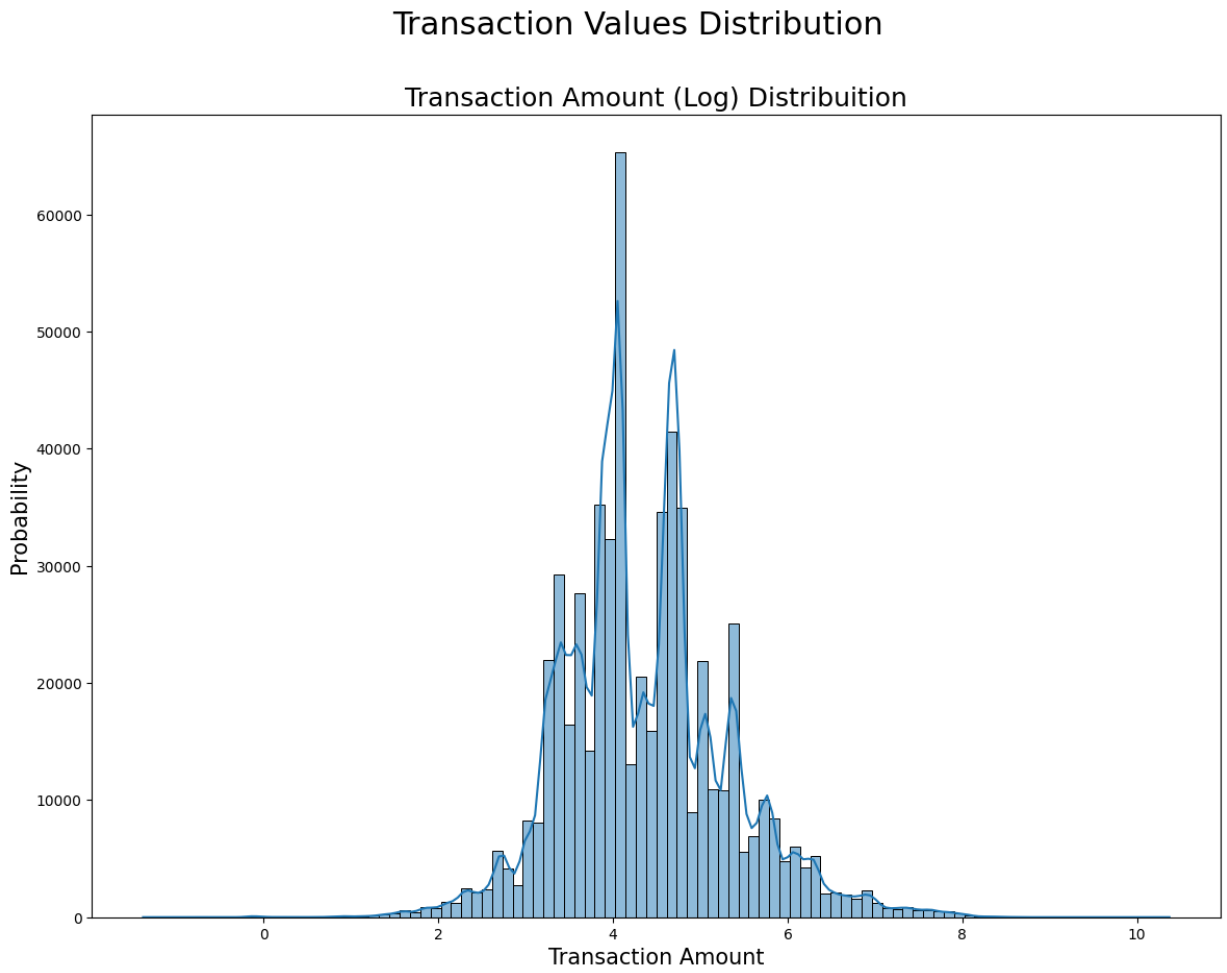
In [30]:
```
plt.figure(figsize=(14,10))
sns.histplot(df['TransactionAmt'],bins=100,kde=True)
plt.title("Transaction Amount Distribution",fontsize=18)
plt.ylabel("Probability")
plt.show()
```




In [31]:
```
# Distribution plot of Transaction Amount less than 1000
plt.figure(figsize=(14,10))
plt.suptitle('Transaction Values Distribution', fontsize=22)
sns.histplot(df[df['TransactionAmt'] <= 1000]['TransactionAmt'],bins=100,kde=T
plt.title("Transaction Amount Distribuition <= 1000", fontsize=18)
plt.xlabel("Transaction Amount", fontsize=15)
plt.ylabel("Probability", fontsize=15)
plt.show()
```




In [32]:
```
# Distribution plot of Transaction Amount less than 1000
plt.figure(figsize=(14,10))
plt.suptitle('Transaction Values Distribution', fontsize=22)
sns.histplot(np.log(df['TransactionAmt']),kde=True,bins=100)
plt.title("Transaction Amount (Log) Distribuition", fontsize=18)
plt.xlabel("Transaction Amount", fontsize=15)
plt.ylabel("Probability", fontsize=15)
plt.show()
```




- Transaction Amount is right skewed.
- Log of transaction amount is almost normally distributed, so use log of transaction amount while building the model

## Product Features

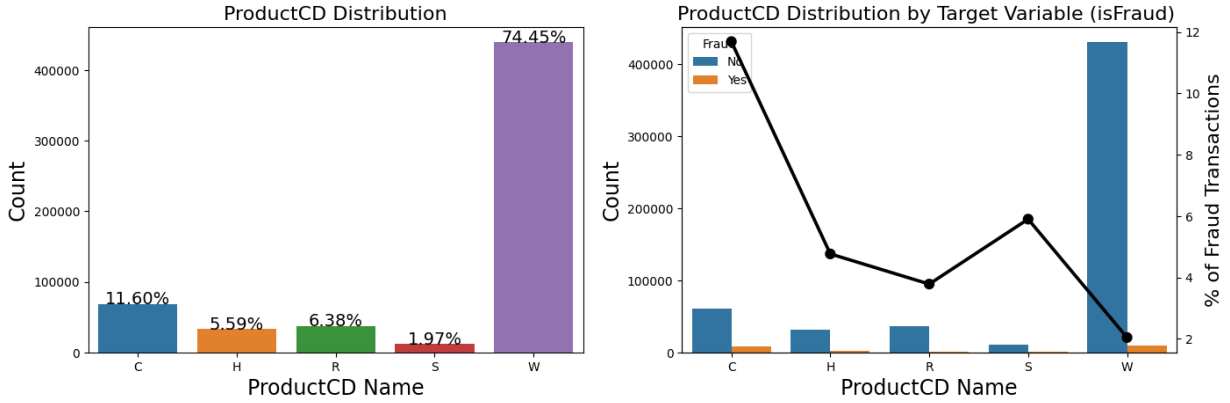
- Distribution of ProductCD
- Distribution of Frauds by Product

In [33]:
```
def plot_cat_feat_dist(df, col):
    tmp = pd.crosstab(df[col], df['isFraud'], normalize='index') * 100
    tmp = tmp.reset_index()
    tmp.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)

    plt.figure(figsize=(16,12))
    plt.suptitle(f'{col} Distributions', fontsize=22)

    plt.subplot(221)
    g = sns.countplot(x=col, data=df, order=tmp[col].values)

    g.set_title(f"{col} Distribution", fontsize=16)
    g.set_xlabel(f"{col} Name", fontsize=17)
    g.set_ylabel("Count", fontsize=17)
    for p in g.patches:
        height = p.get_height()
        g.text(p.get_x()+p.get_width()/2.,
               height + 3,
```

```
                '{:1.2f}%'.format(height/total*100),
                ha="center", fontsize=14)

    plt.subplot(222)
    g1 = sns.countplot(x=col, hue='isFraud', data=df, order=tmp[col].values)
    plt.legend(title='Fraud', loc='best', labels=['No', 'Yes'])
    gt = g1.twinx()
    gt = sns.pointplot(x=col, y='Fraud', data=tmp, color='black', order=tmp[co
    gt.set_ylabel("% of Fraud Transactions", fontsize=16)

    g1.set_title(f"{col} Distribution by Target Variable (isFraud) ", fontsize
    g1.set_xlabel(f"{col} Name", fontsize=17)
    g1.set_ylabel("Count", fontsize=17)

    plt.subplots_adjust(hspace = 0.4, top = 0.85)
    plt.show()
```

In [34]:
```
plot_cat_feat_dist(df, "ProductCD")
```

ProductCD Distributions




In [35]:
```
df.groupby('ProductCD')['isFraud'].mean()
```

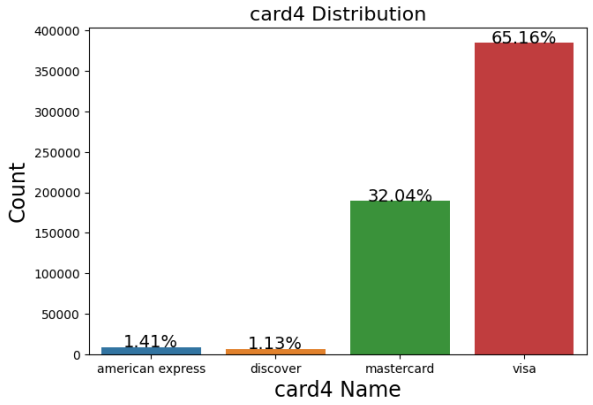
Out[35]:
```
ProductCD
C    0.116873
H    0.047662
R    0.037826
S    0.058996
W    0.020399
Name: isFraud, dtype: float64
```

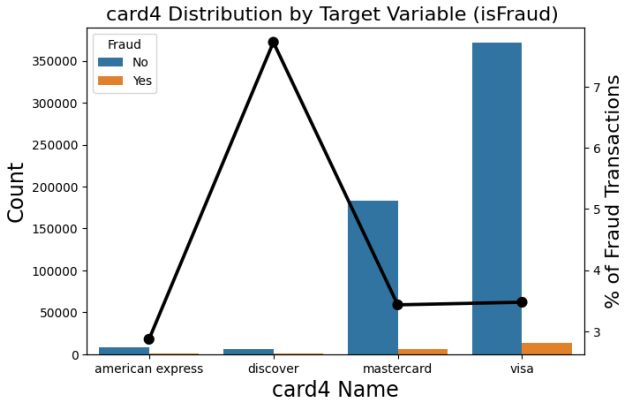
- 75% of the transactions are for Product Catergory W
- 11.6% of the transactions are for Product Category C
- Fraud Transaction rate is maximum for Product Category C and minimum for Product Category W

## Card Features

In [36]:
```
# Card 4
plot_cat_feat_dist(df, "card4")
```

card4 Distributions



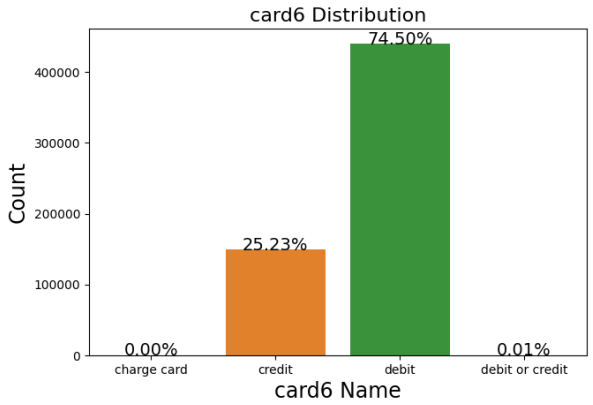

In [37]:
```
df.groupby('card4')['isFraud'].mean()
```

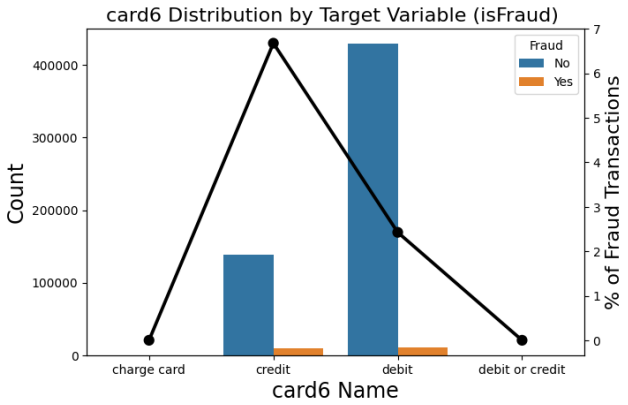
Out[37]:
```
card4
american express    0.028698
discover            0.077282
mastercard          0.034331
visa                0.034756
Name: isFraud, dtype: float64
```

- 97% of transactions are from Mastercard(32%) and Visa(65%
- Fraud transaction rate is highest for discover cards(~8%) against ~3.5% of Mastercard and Visa and 2.87% in American Express

In [38]:
```
# Card 6
plot_cat_feat_dist(df, "card6")
```

card6 Distributions




In [39]:
```
df.groupby('card6')['isFraud'].mean()
```

Out[39]:
```
card6
charge card        0.000000
credit             0.066785
debit              0.024263
debit or credit    0.000000
Name: isFraud, dtype: float64
```

- Almost all the transactions are from Credit and Debit cards.
- Debit card transactions are almost 3 times as compared to credit card transactions.
- Fraud transaction rate is high for Credit cards as compared to Debit cards.

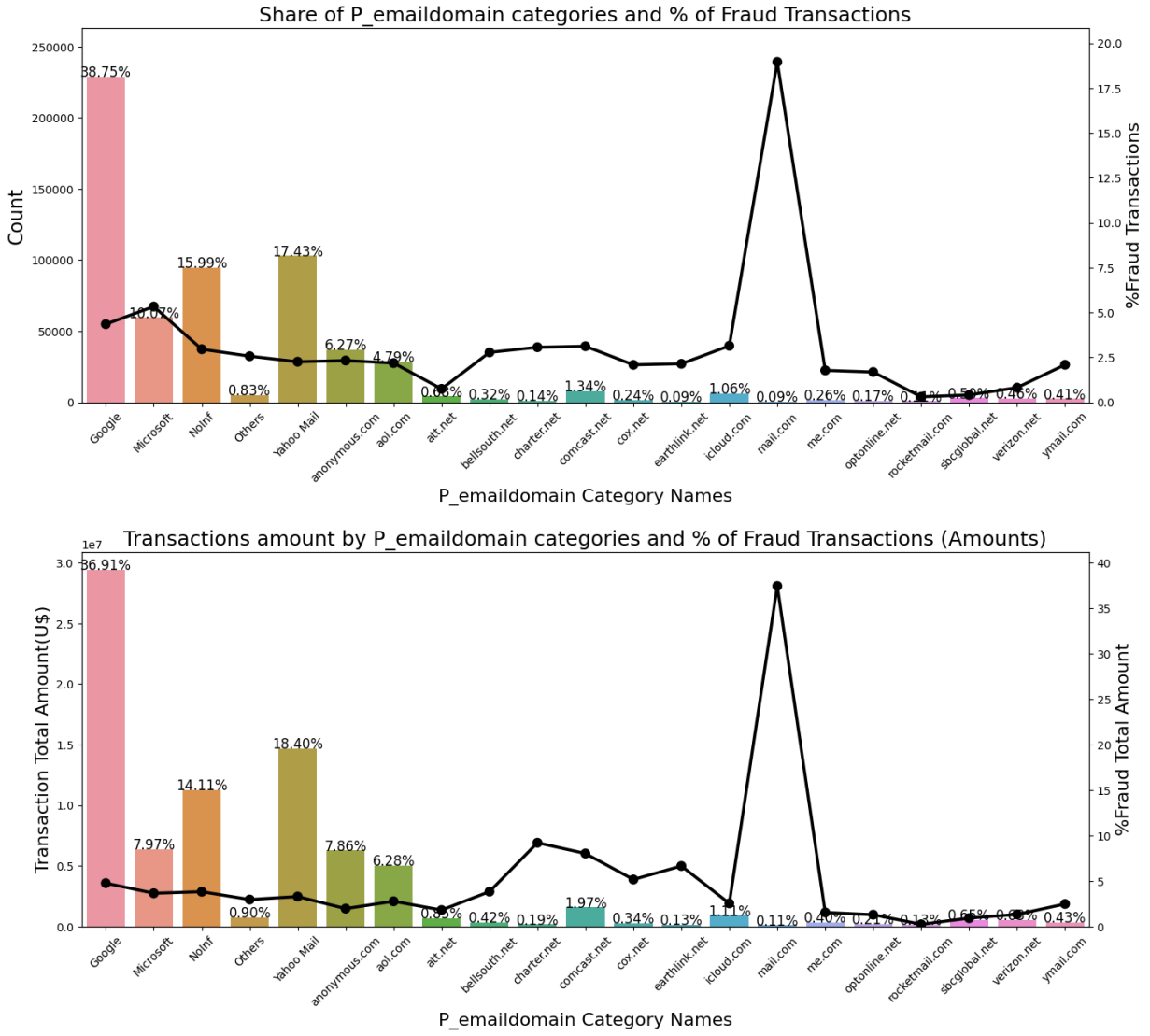
## P emaildomain

In [40]:
```
df.loc[df['P_emaildomain'].isin(['gmail.com', 'gmail']),'P_emaildomain'] = 'Go

df.loc[df['P_emaildomain'].isin(['yahoo.com', 'yahoo.com.mx',  'yahoo.co.uk',
                                 'yahoo.co.jp', 'yahoo.de', 'yahoo.fr'
                                 'yahoo.es']), 'P_emaildomain'] = 'Yah
df.loc[df['P_emaildomain'].isin(['hotmail.com','outlook.com','msn.com', 'live.
                                 'hotmail.es','hotmail.co.uk', 'hotmai
                                 'outlook.es', 'live.com', 'live.fr',
                                 'hotmail.fr']), 'P_emaildomain'] = 'M
df.loc[df.P_emaildomain.isin(df.P_emaildomain\
                                 .value_counts()[df.P_emaildomain.valu
                                 .index), 'P_emaildomain'] = "Others"
df.P_emaildomain.fillna("NoInf", inplace=True)
```

In [41]:
```
def plot_cat_with_amt(df, col, lim=2000):
    tmp = pd.crosstab(df[col], df['isFraud'], normalize='index') * 100
    tmp = tmp.reset_index()
    tmp.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)

    plt.figure(figsize=(16,14))
    plt.suptitle(f'{col} Distributions ', fontsize=24)

    plt.subplot(211)
    g = sns.countplot( x=col,  data=df, order=list(tmp[col].values))
    gt = g.twinx()
    gt = sns.pointplot(x=col, y='Fraud', data=tmp, order=list(tmp[col].values)
                       color='black', legend=False, )
    gt.set_ylim(0,tmp['Fraud'].max()*1.1)
    gt.set_ylabel("%Fraud Transactions", fontsize=16)
    g.set_title(f"Share of {col} categories and % of Fraud Transactions", font
    g.set_xlabel(f"{col} Category Names", fontsize=16)
    g.set_ylabel("Count", fontsize=17)
    g.set_xticklabels(g.get_xticklabels(),rotation=45)
    sizes = []
    for p in g.patches:
        height = p.get_height()
        sizes.append(height)
        g.text(p.get_x()+p.get_width()/2.,
               height + 3,
               '{:1.2f}%'.format(height/total*100),
               ha="center",fontsize=12)

    g.set_ylim(0,max(sizes)*1.15)

    ########################################################################
    perc_amt = (df.groupby(['isFraud',col])['TransactionAmt'].sum() \
                / df.groupby([col])['TransactionAmt'].sum() * 100).unstack('is
    perc_amt = perc_amt.reset_index()
    perc_amt.rename(columns={0:'NoFraud', 1:'Fraud'}, inplace=True)
    amt = df.groupby([col])['TransactionAmt'].sum().reset_index()
    perc_amt = perc_amt.fillna(0)
```

```
    plt.subplot(212)
    g1 = sns.barplot(x=col, y='TransactionAmt',
                     data=amt,
                     order=list(tmp[col].values))
    g1t = g1.twinx()
    g1t = sns.pointplot(x=col, y='Fraud', data=perc_amt,
                        order=list(tmp[col].values),
                     color='black', legend=False, )
    g1t.set_ylim(0,perc_amt['Fraud'].max()*1.1)
    g1t.set_ylabel("%Fraud Total Amount", fontsize=16)
    g.set_xticklabels(g.get_xticklabels(),rotation=45)
    g1.set_title(f"Transactions amount by {col} categories and % of Fraud Tran
    g1.set_xlabel(f"{col} Category Names", fontsize=16)
    g1.set_ylabel("Transaction Total Amount(U$)", fontsize=16)
    g1.set_xticklabels(g.get_xticklabels(),rotation=45)

    for p in g1.patches:
        height = p.get_height()
        g1.text(p.get_x()+p.get_width()/2.,
                height + 3,
                '{:1.2f}%'.format(height/total_amt*100),
                ha="center",fontsize=12)

    plt.subplots_adjust(hspace=.4, top = 0.9)
    plt.show()
```

In [42]:
```
plot_cat_with_amt(df, 'P_emaildomain')
```




In [43]:
```
df.groupby('P_emaildomain')['isFraud'].mean()
```

Out[43]:
```
P_emaildomain
Google            0.043496
Microsoft         0.053298
NoInf             0.029538
Others            0.025646
Yahoo Mail        0.022544
anonymous.com     0.023217
aol.com           0.021811
att.net           0.007439
bellsouth.net     0.027763
charter.net       0.030637
comcast.net       0.031187
cox.net           0.020818
earthlink.net     0.021401
icloud.com        0.031434
mail.com          0.189624
me.com            0.017740
optonline.net     0.016815
rocketmail.com    0.003012
sbcglobal.net     0.004040
verizon.net       0.008133
```

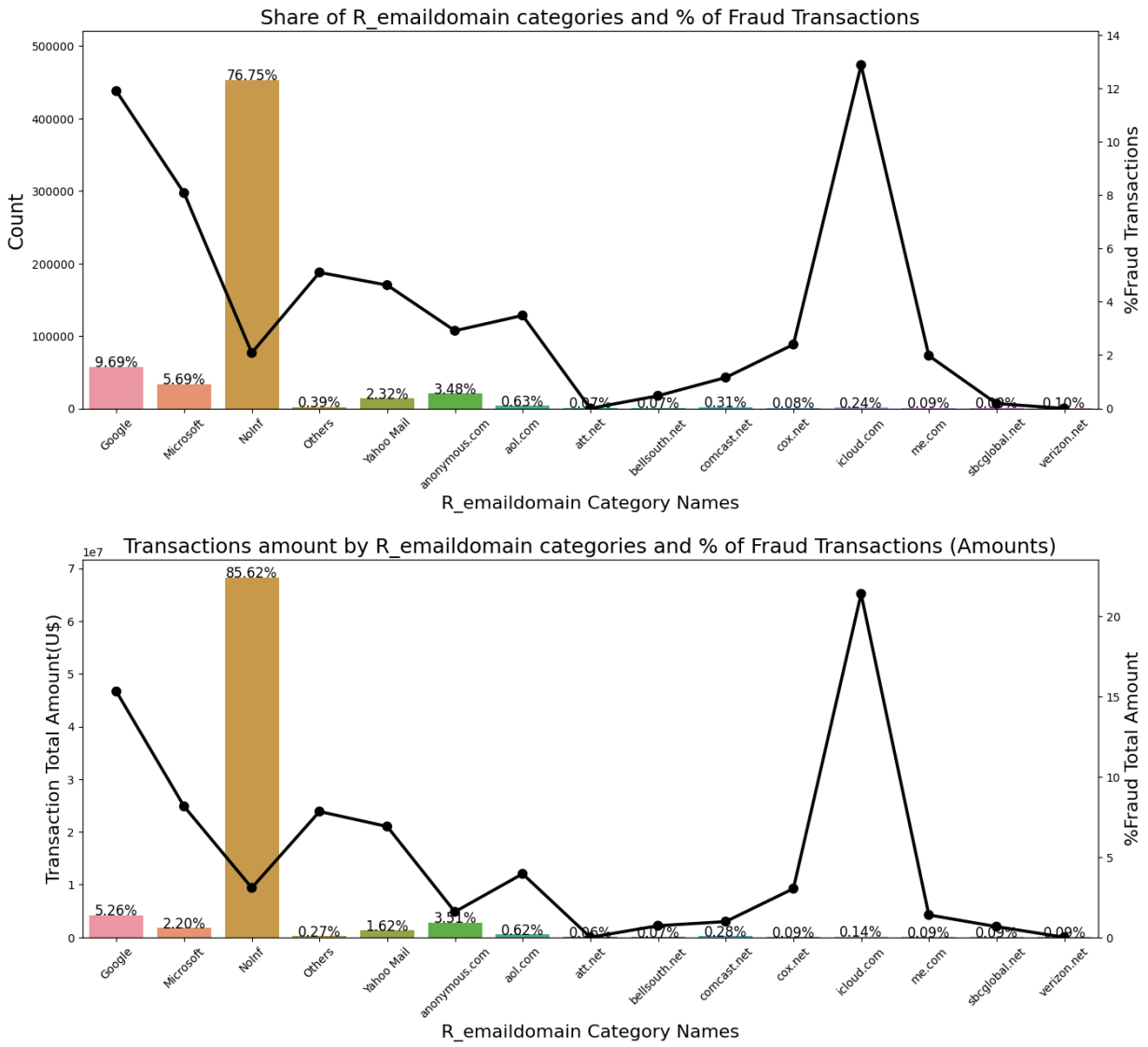
```
ymail.com         0.020868
Name: isFraud, dtype: float64
```

- Majority of transactions are with P_emaildomain as Google, Microsoft and Yahoo Mail
- There isn't any information about P_emaildomain of around 16% transactions in terms of count and 14.11% in terms of amount
- Fraud transaction rate for Microsoft is high as compared to Google and Yahoo mail
- Fraud transaction rate (amount) for Google is high as comapred to Microsoft and Yahoo mail

## R-Email Domain

In [44]:
```
df.loc[df['R_emaildomain'].isin(['gmail.com', 'gmail']),'R_emaildomain'] = 'Go

df.loc[df['R_emaildomain'].isin(['yahoo.com', 'yahoo.com.mx',  'yahoo.co.uk',
                                 'yahoo.co.jp', 'yahoo.de', 'yahoo
                                 'yahoo.es']), 'R_emaildomain'] =
df.loc[df['R_emaildomain'].isin(['hotmail.com','outlook.com','msn.com', 'live.
                                 'hotmail.es','hotmail.co.uk', 'ho
                                 'outlook.es', 'live.com', 'live.f
                                 'hotmail.fr']), 'R_emaildomain']
df.loc[df.R_emaildomain.isin(df.R_emaildomain\
                                 .value_counts()[df.R_emaildomain.valu
                                 .index), 'R_emaildomain'] = "Others"
df.R_emaildomain.fillna("NoInf", inplace=True)
```

In [45]:
```
plot_cat_with_amt(df, 'R_emaildomain')
```




In [46]:
```
df.groupby('R_emaildomain')['isFraud'].mean()
```

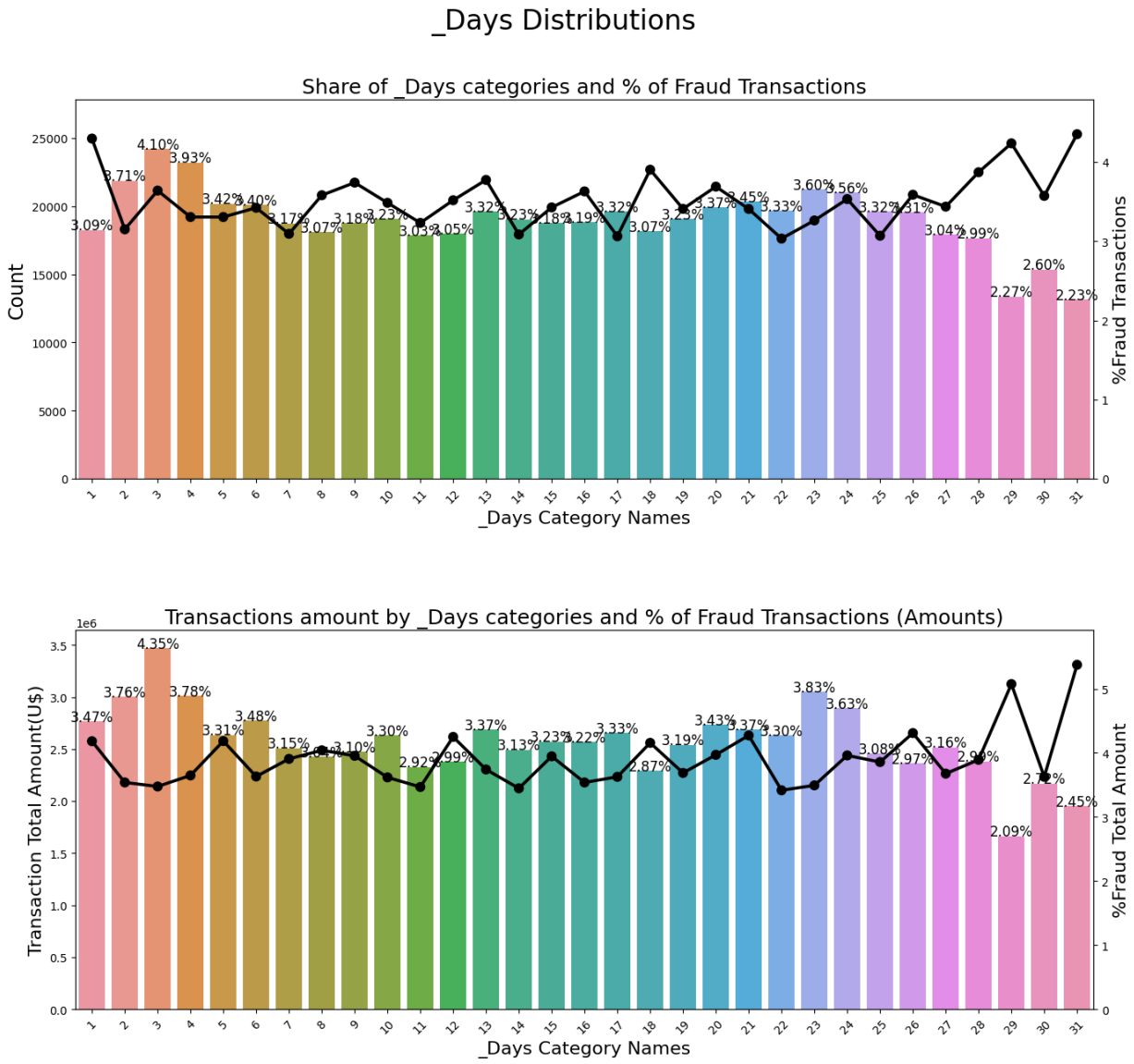
Out[46]:
```
R_emaildomain
Google           0.118986
Microsoft        0.080764
NoInf            0.020819
Others           0.050989
Yahoo Mail       0.046235
anonymous.com    0.029130
aol.com          0.034855
att.net          0.000000
bellsouth.net    0.004739
comcast.net      0.011589
cox.net          0.023965
icloud.com       0.128755
me.com           0.019784
sbcglobal.net    0.001812
verizon.net      0.000000
Name: isFraud, dtype: float64
```

- There isn't any information about R_emaildomain for Majority of transactions (76.75% count , 85.62% amount)

- Fraud transaction rate for Google is high as compared to Yahoo, anaonymous.com and Microsoft

## Days of the Month

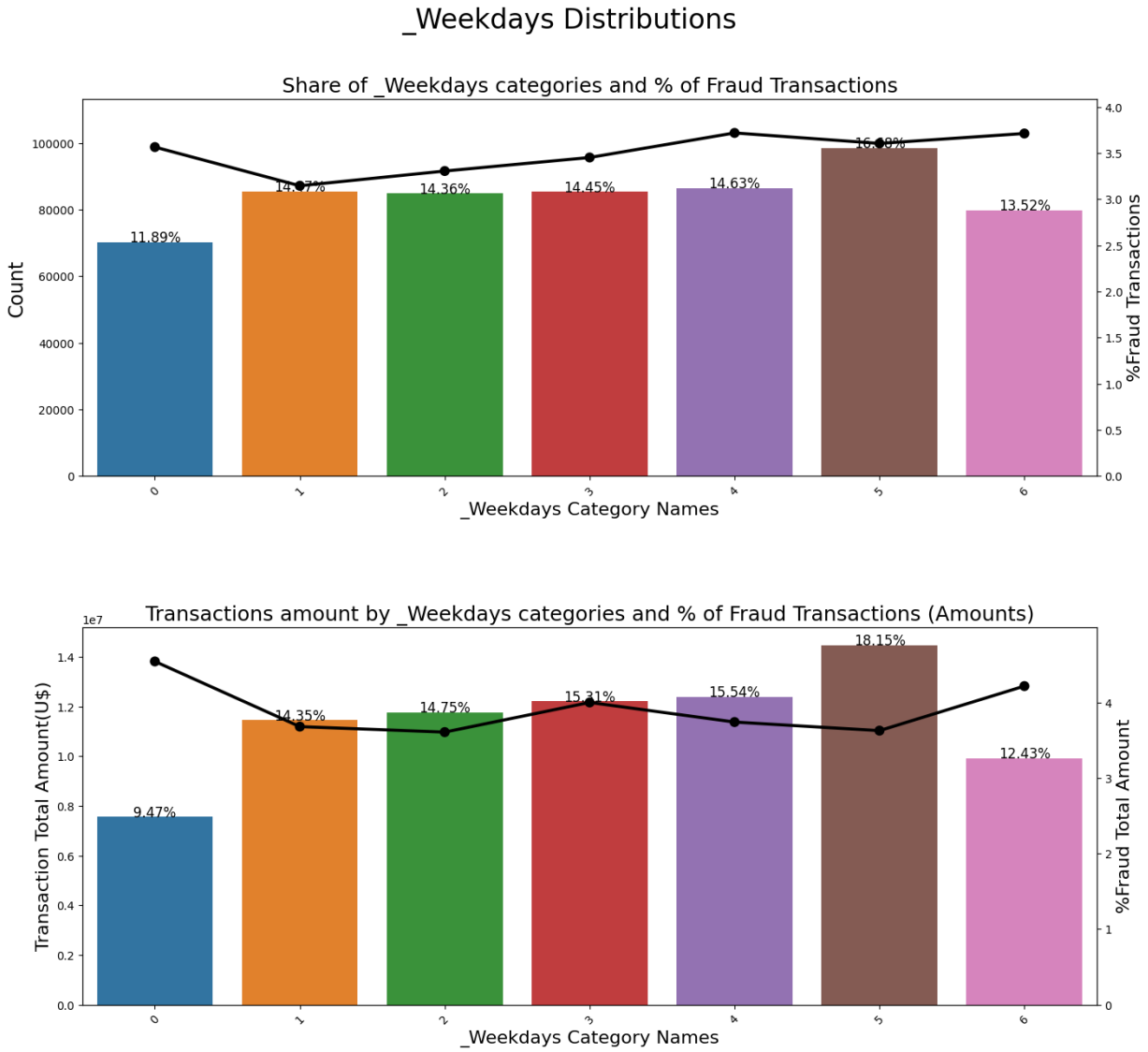
In [47]:
```
plot_cat_with_amt(df, '_Days')
```




- The perc of fraud transactions is highest towards the beginning and the end of the month. Might be accelerated at the time of receiving pay-checks.
- Incidentally, fraud transaction rate is high on the days when number of transactions are less
- Day 29,30 and 31 are having less transactions, looks like people are cautious with spending in those times.

## Days of the week

In [48]:
```
plot_cat_with_amt(df, '_Weekdays')
```

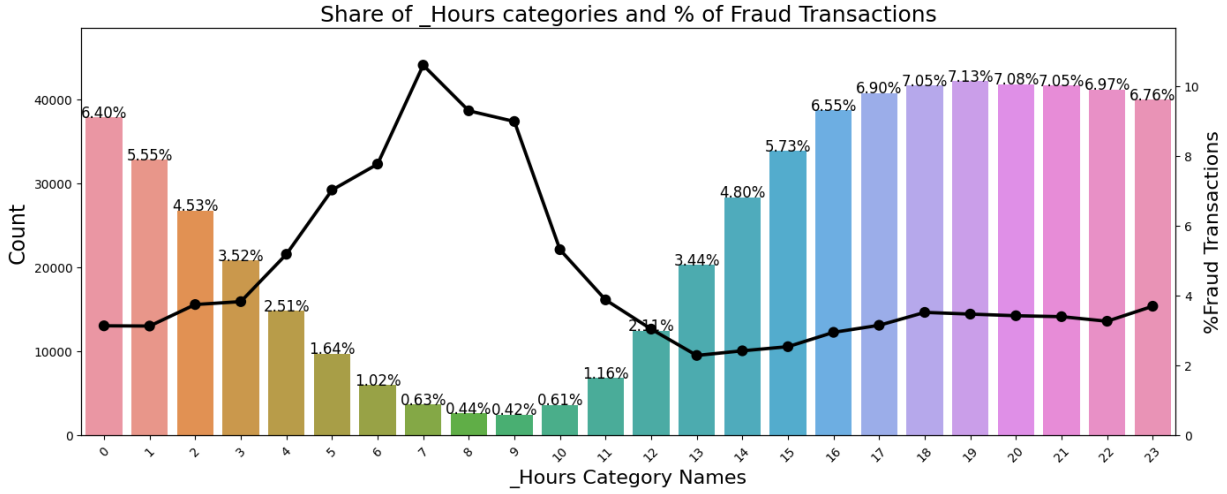


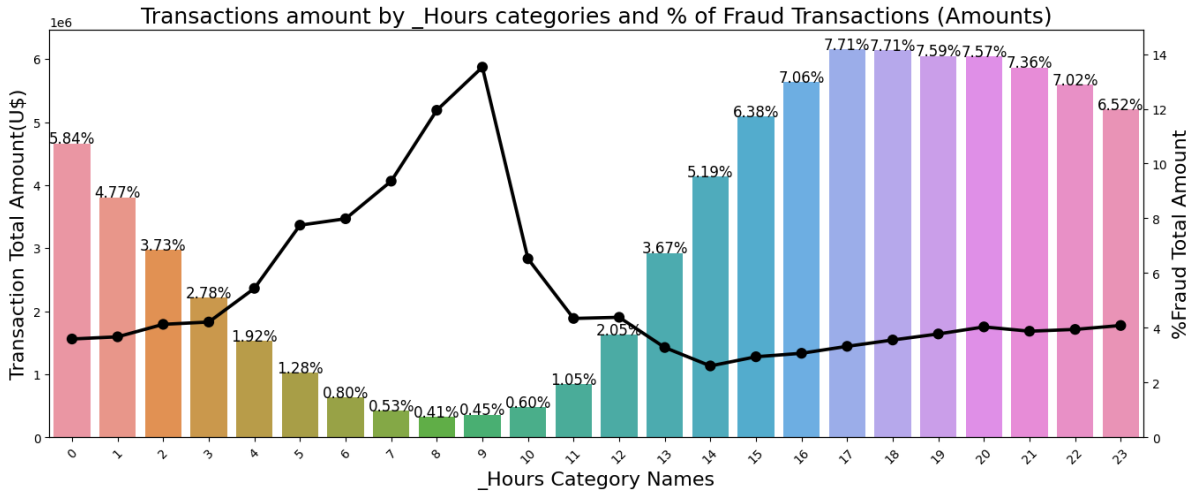

- Surprisingly fraud transaction rate is high on the days when number of transactions and transaction amounts are less. Day 0 and 6
- Day 0 and 6 have less transactions, these might be weekend days

## Hour of the Day

In [49]:
```
plot_cat_with_amt(df, '_Hours')
```

_Hours Distributions



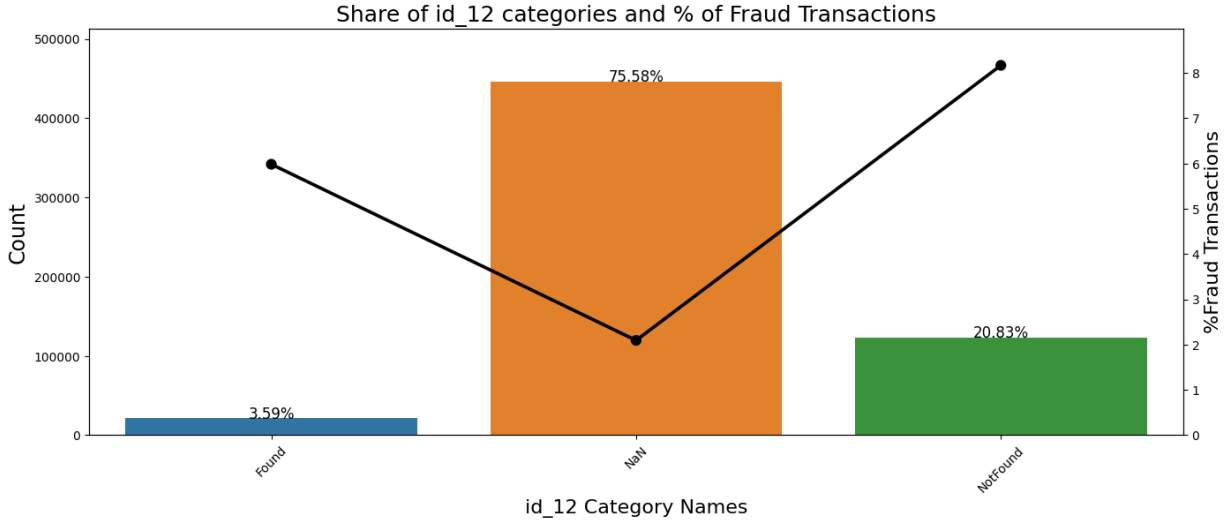



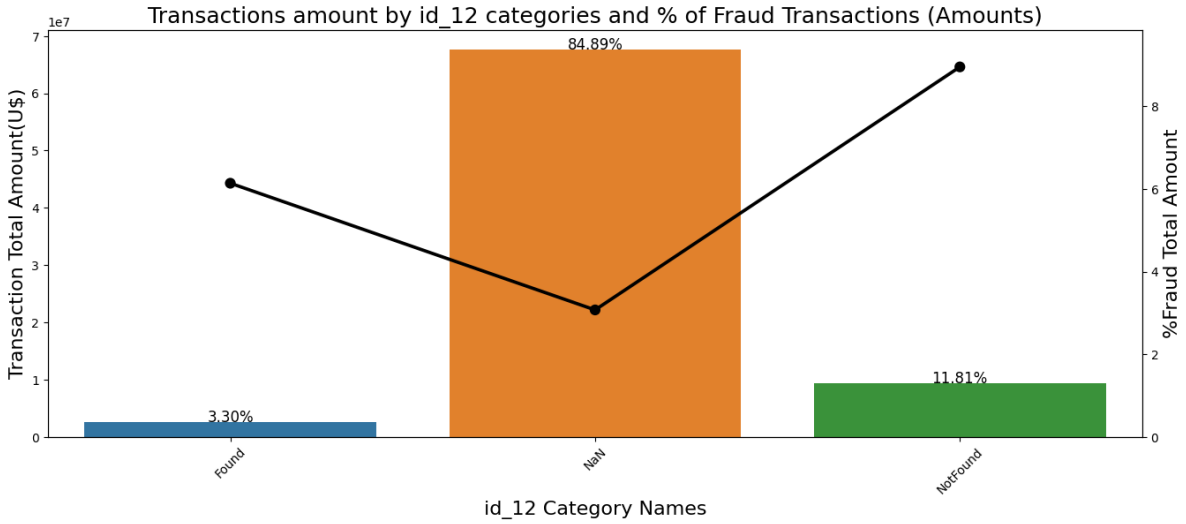

- Transactions start decreasing mid night but the fraud rate starts increasing
- Transactions from 3 AM to 12 PM needs to monitored very closely

## Columns from identity data

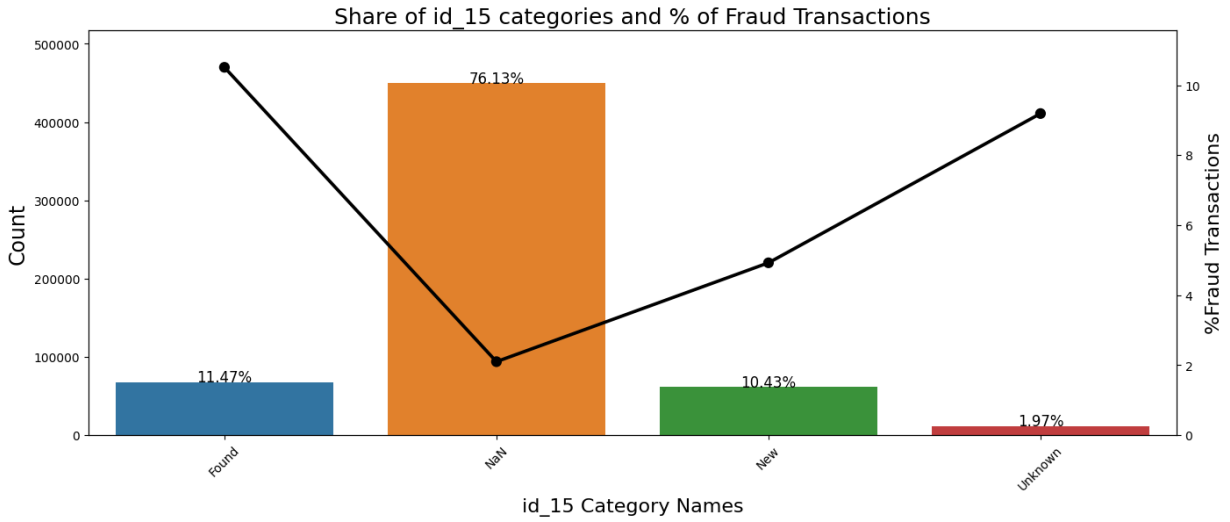
In [50]:
```
for col in ['id_12', 'id_15', 'id_16', 'id_28', 'id_29']:
    df[col] = df[col].fillna('NaN')
    plot_cat_with_amt(df, col)
```

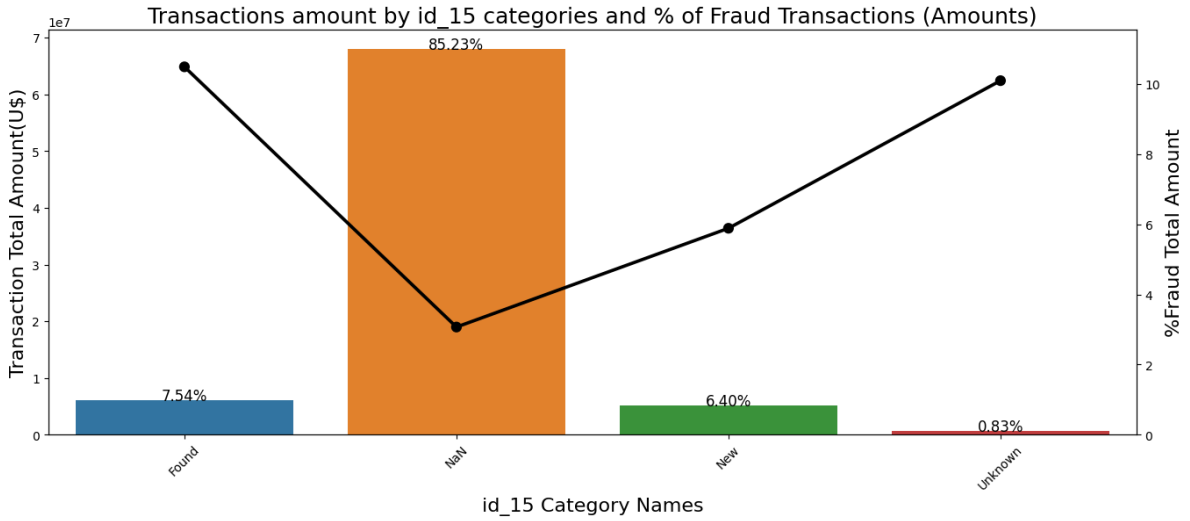
# id_12 Distributions



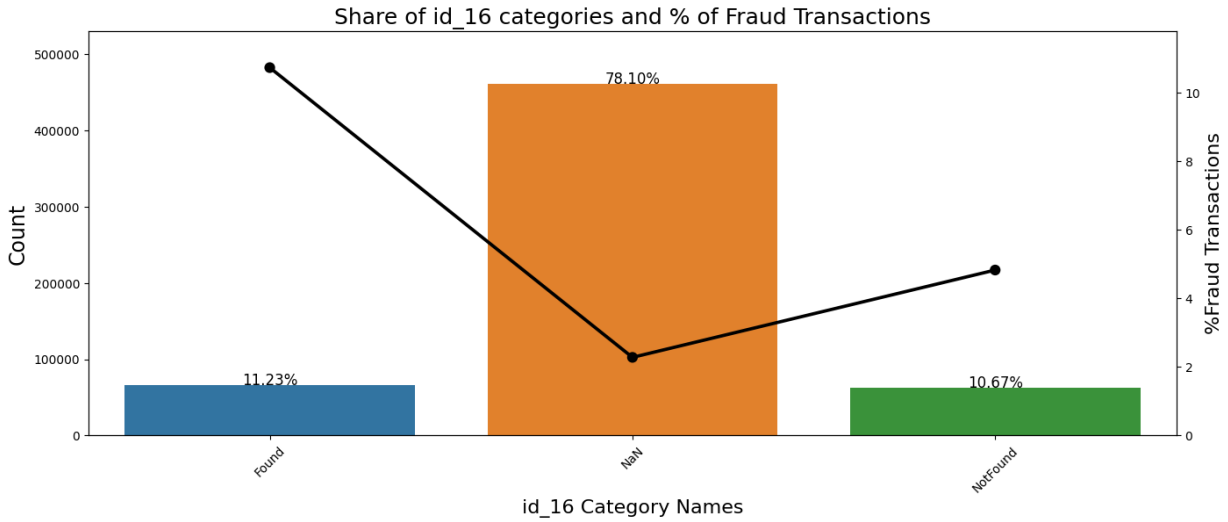



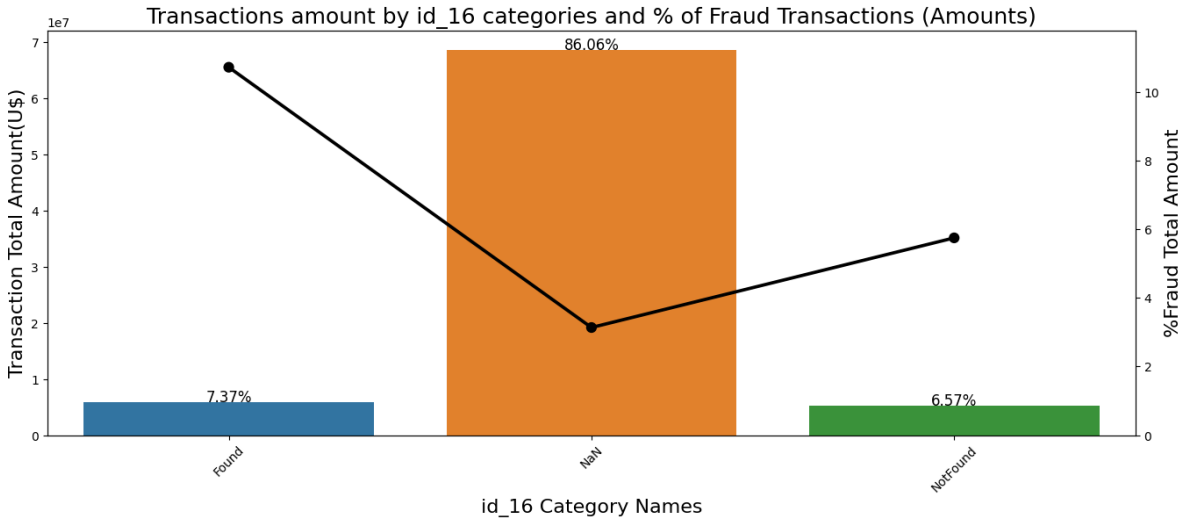
# id_15 Distributions



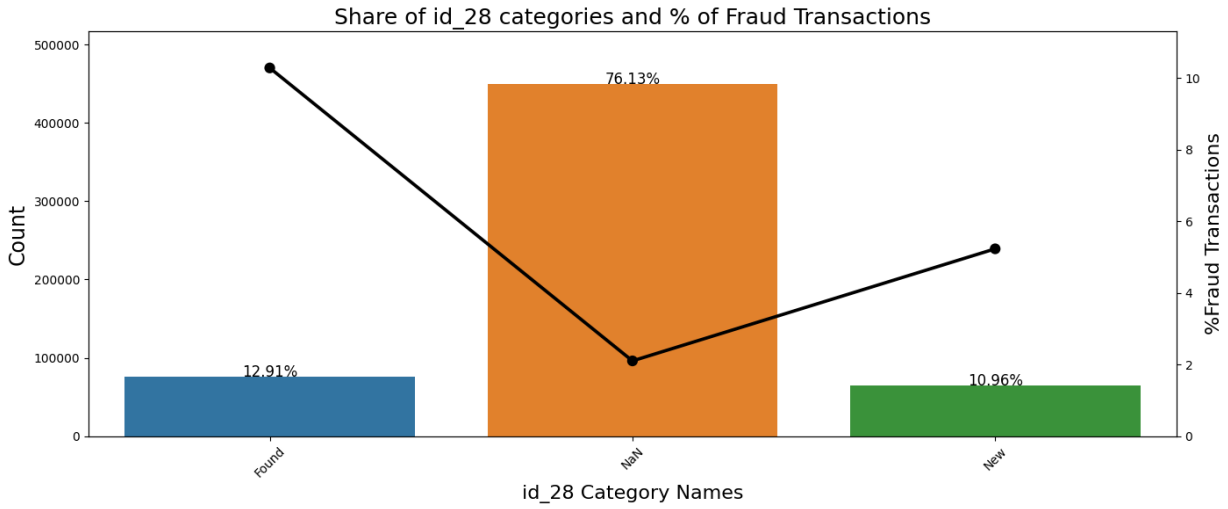



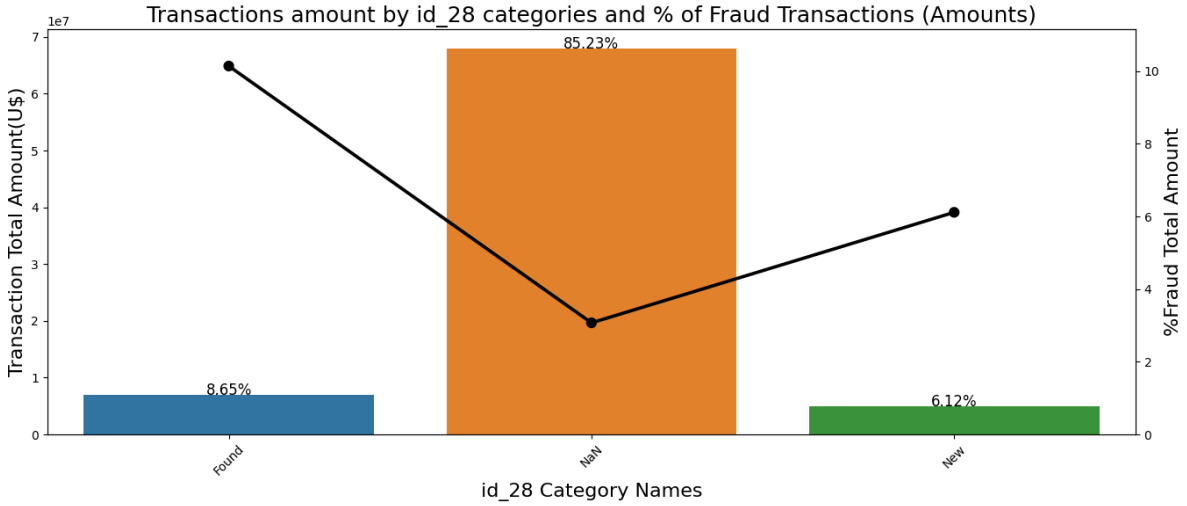
# id_16 Distributions



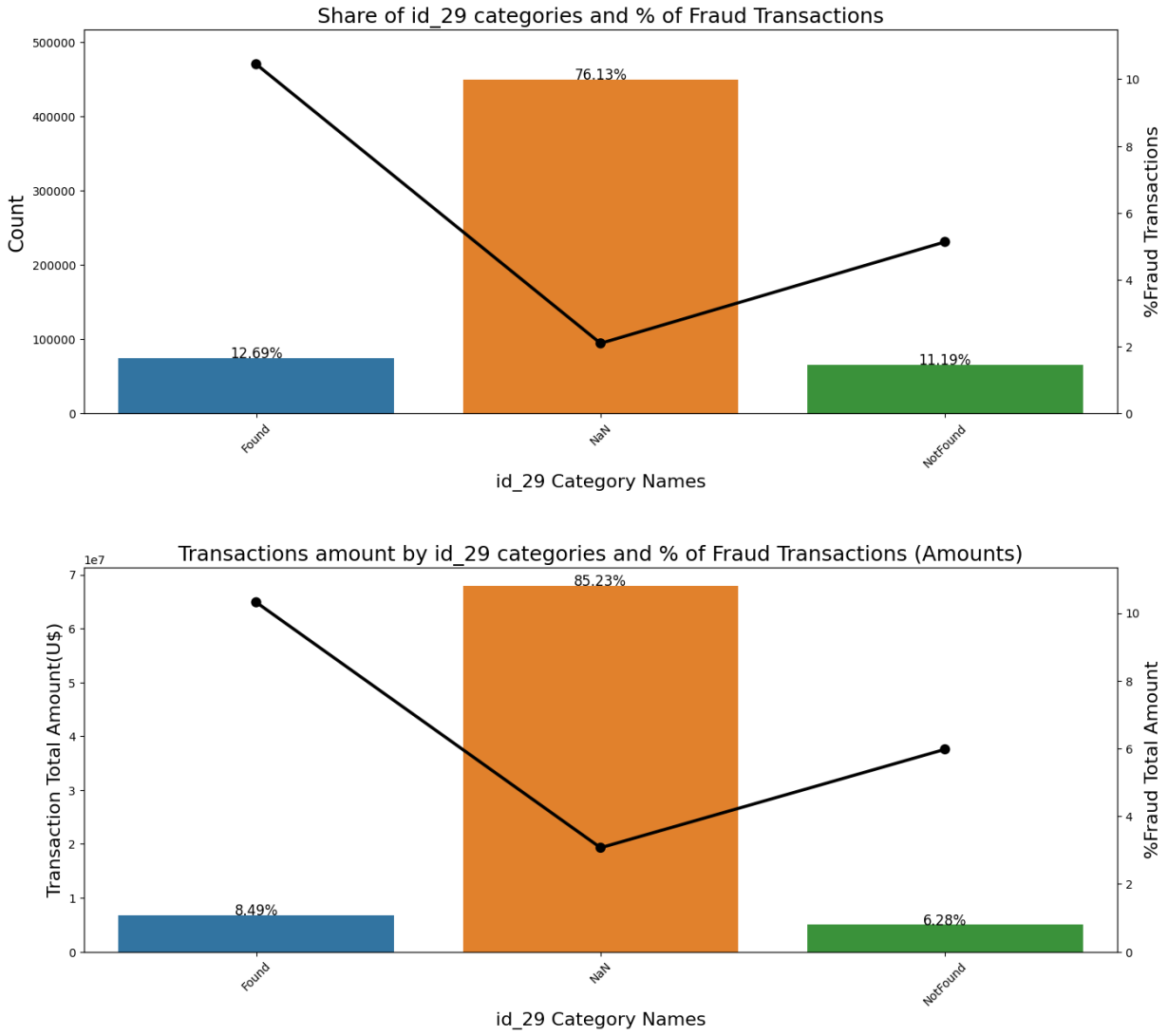



# id_28 Distributions

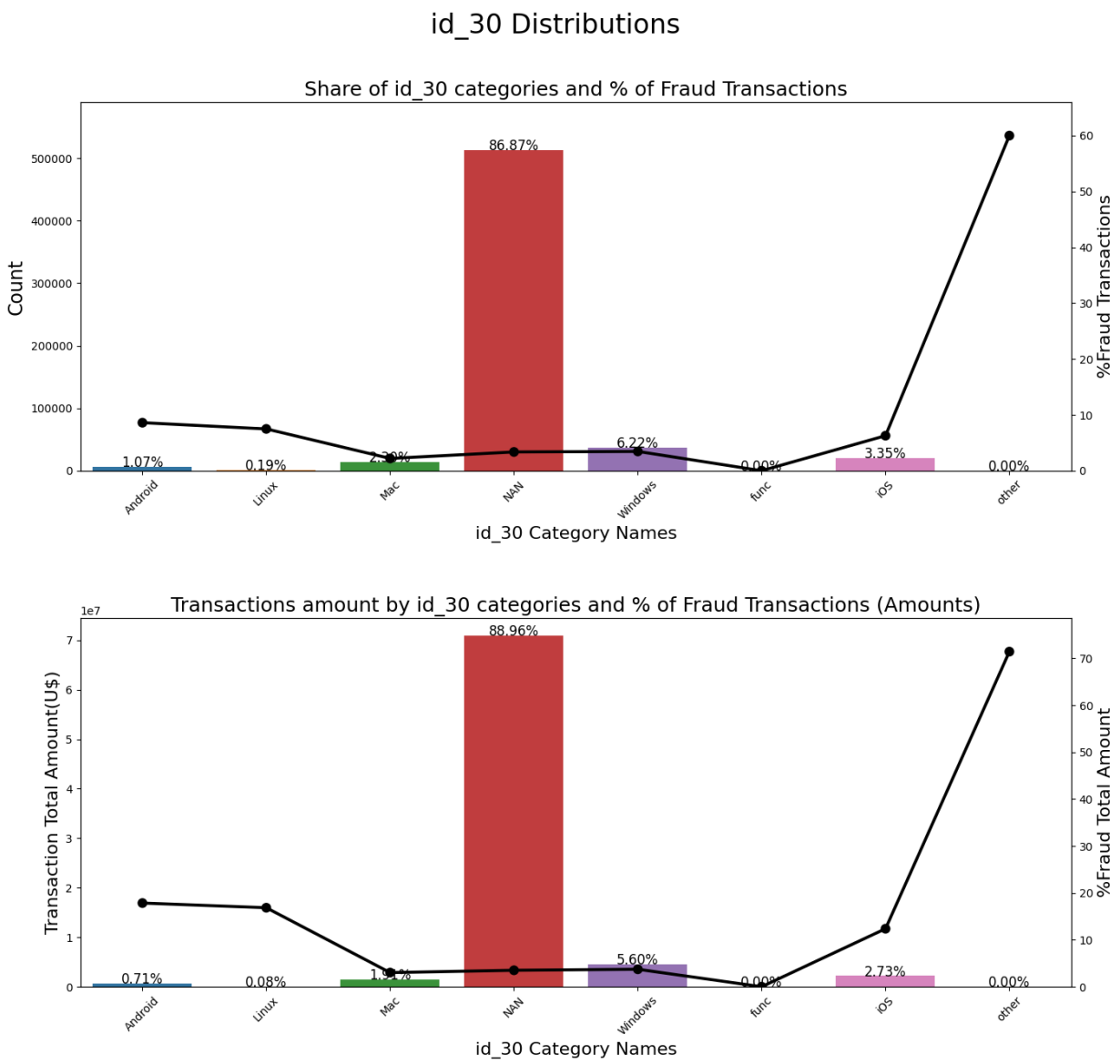


## id_29 Distributions




In [51]:
```
df.loc[df['id_30'].str.contains('Windows', na=False), 'id_30'] = 'Windows'
df.loc[df['id_30'].str.contains('iOS', na=False), 'id_30'] = 'iOS'
df.loc[df['id_30'].str.contains('Mac OS', na=False), 'id_30'] = 'Mac'
df.loc[df['id_30'].str.contains('Android', na=False), 'id_30'] = 'Android'
df['id_30'].fillna("NAN", inplace=True)

plot_cat_with_amt(df, "id_30")
```




In [52]:
```
cat_columns = df.select_dtypes(include=['object']).columns
len(cat_columns)
```

Out[52]:
```
29
```

In [53]:
```
binary_columns = [col for col in df.columns if df[col].nunique() == 2]
len(binary_columns)
```

Out[53]:
```
435
```

In [54]:
```
num_columns = [col for col in df.columns if (col not in cat_columns) & (col no
len(num_columns)
```

Out[54]:
```
389
```

In [55]:
```
cat_columns = cat_columns.to_list() + binary_columns
```

# 7. Feature Engineering

In [56]:
```
df.head()
```

Out[56]:

| | TransactionID | isFraud | TransactionDT | TransactionAmt | ProductCD | card1 | card2 | card3 | ca |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2987000 | 0 | 86400 | 68.5 | W | 13926 | NaN | 150.0 | disc |
| **1** | 2987001 | 0 | 86401 | 29.0 | W | 2755 | 404.0 | 150.0 | master |
| **2** | 2987002 | 0 | 86469 | 59.0 | W | 4663 | 490.0 | 150.0 | |
| **3** | 2987003 | 0 | 86499 | 50.0 | W | 18132 | 567.0 | 150.0 | master |
| **4** | 2987004 | 0 | 86506 | 50.0 | H | 4497 | 514.0 | 150.0 | master |

5 rows × 840 columns

## Domain Specific Features

In [57]:
```
df['Trans_min_mean'] = df['TransactionAmt'] - np.nanmean(df['TransactionAmt'],
df['Trans_min_std']  = df['Trans_min_mean'] / np.nanstd(df['TransactionAmt'].a
```

In [58]:
```
df['TransactionAmt_to_mean_card1'] = df['TransactionAmt'] / df.groupby(['card1
df['TransactionAmt_to_mean_card4'] = df['TransactionAmt'] / df.groupby(['card4
df['TransactionAmt_to_std_card1']  = df['TransactionAmt'] / df.groupby(['card1
df['TransactionAmt_to_std_card4']  = df['TransactionAmt'] / df.groupby(['card4
```

In [59]:
```
df['TransactionAmt'] = np.log(df['TransactionAmt'])
```

In [60]:
```
df.head()
```

Out[60]:

| | TransactionID | isFraud | TransactionDT | TransactionAmt | ProductCD | card1 | card2 | card3 | ca |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2987000 | 0 | 86400 | 4.226834 | W | 13926 | NaN | 150.0 | disc |
| **1** | 2987001 | 0 | 86401 | 3.367296 | W | 2755 | 404.0 | 150.0 | master |
| **2** | 2987002 | 0 | 86469 | 4.077537 | W | 4663 | 490.0 | 150.0 | |
| **3** | 2987003 | 0 | 86499 | 3.912023 | W | 18132 | 567.0 | 150.0 | master |
| **4** | 2987004 | 0 | 86506 | 3.912023 | H | 4497 | 514.0 | 150.0 | master |

5 rows × 846 columns

# 8. Dimensionality Reduction - PCA

In [61]:
```
def perform_PCA(df, cols, n_components, prefix='PCA_', rand_seed=4):
    pca = PCA(n_components=n_components, random_state=rand_seed)
    principalComponents = pca.fit_transform(df[cols])
    principalDf = pd.DataFrame(principalComponents)
    df.drop(cols, axis=1, inplace=True)

    principalDf.rename(columns=lambda x: str(prefix)+str(x), inplace=True)
    df = pd.concat([df, principalDf], axis=1)
    return df
```

In [62]:
```
# Columns starting from V1 to V339
filter_col = df.columns[53:392]
```

Impute missing values in the mas_v columns, later use minmax_scale function to scale the values in these columns

In [63]:
```
from   sklearn.preprocessing import minmax_scale

# Fill na values and scale V columns
for col in filter_col:
    df[col] = df[col].fillna((df[col].min() - 2))
    df[col] = (minmax_scale(df[col], feature_range=(0,1)))

# Perform PCA
df = perform_PCA(df, filter_col, prefix='PCA_V_', n_components=30)
```

In [64]:
```
df = reduce_mem_usage(df)
```

```
Mem. usage decreased to 1170.85 Mb (4.4% reduction)
```

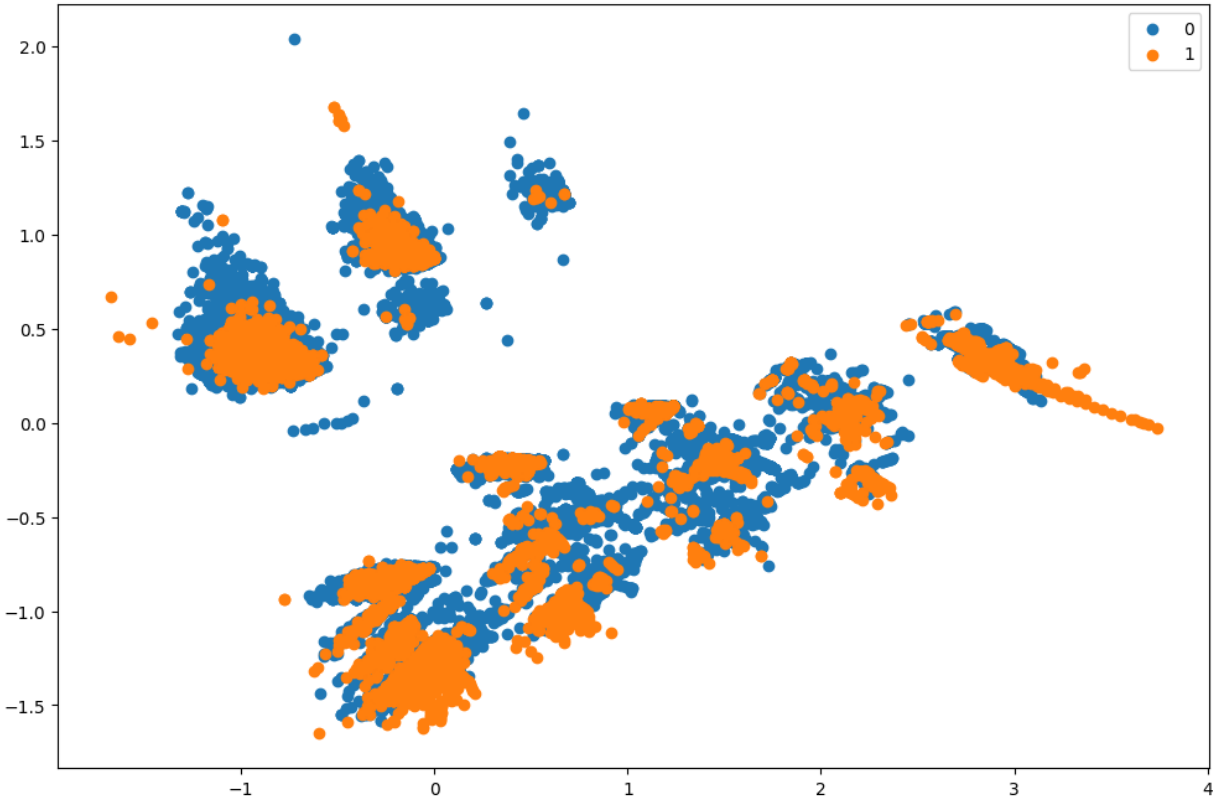
In [65]:
```
df.head()
```

Out[65]:

| | TransactionID | isFraud | TransactionDT | TransactionAmt | ProductCD | card1 | card2 | card3 | ca |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2987000 | 0 | 86400 | 4.226562 | W | 13926 | NaN | 150.0 | disc |
| **1** | 2987001 | 0 | 86401 | 3.367188 | W | 2755 | 404.0 | 150.0 | master |
| **2** | 2987002 | 0 | 86469 | 4.078125 | W | 4663 | 490.0 | 150.0 | |
| **3** | 2987003 | 0 | 86499 | 3.912109 | W | 18132 | 567.0 | 150.0 | master |
| **4** | 2987004 | 0 | 86506 | 3.912109 | H | 4497 | 514.0 | 150.0 | master |

5 rows × 537 columns

In [66]:
```
# Plot first 2 PCA features and colour by target variable
plt.figure(figsize=(12, 8));
groups = df.groupby("isFraud")
for name, group in groups:
    plt.scatter(group["PCA_V_0"], group["PCA_V_1"], label=name)
plt.legend()
plt.show()
```




# 9. Feature Encoding

In [67]:
```
cat_cols = df.select_dtypes(include=['object','category']).columns.tolist()
```

In [68]:
```
binary_cols = [col for col in df.columns if df[col].dropna().nunique() ==2 and
```

In [69]:
```
num_cols = [col for col in df.select_dtypes(include=['int64','int32','int16','
```

In [70]:
```
len(cat_cols)
```

Out[70]:
```
29
```

In [71]:
```
len(binary_cols)
```

Out[71]:
```
415
```

In [72]:
```
len(num_cols)
```

Out[72]:
```
92
```

In [73]:
```
cat_cols = cat_cols + binary_cols
len(cat_cols)
```

Out[73]:
```
444
```

In [ ]:
```
```

In [74]:
```
# Frequecny encoding variables
frequency_encoded_variables = []
for col in cat_cols:
    if df[col].nunique() > 30:
        print(col, df[col].nunique())
        frequency_encoded_variables.append(col)
```

```
id_31 130
id_33 260
DeviceInfo 1786
```

In [75]:
```
for variable in tqdm(frequency_encoded_variables):
    # group by frequency
    fq = df.groupby(variable).size()/len(df)
    # mapping values to dataframe
    df.loc[:, "{}".format(variable)] = df[variable].map(fq)
    cat_columns.remove(variable)
```

```
100%|██████████| 3/3 [00:00<00:00,  4.14it/s]
```

In [76]:
```
df.head()
```

Out[76]:

| | TransactionID | isFraud | TransactionDT | TransactionAmt | ProductCD | card1 | card2 | card3 | ca |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2987000 | 0 | 86400 | 4.226562 | W | 13926 | NaN | 150.0 | disc |
| **1** | 2987001 | 0 | 86401 | 3.367188 | W | 2755 | 404.0 | 150.0 | master |
| **2** | 2987002 | 0 | 86469 | 4.078125 | W | 4663 | 490.0 | 150.0 | |
| **3** | 2987003 | 0 | 86499 | 3.912109 | W | 18132 | 567.0 | 150.0 | master |
| **4** | 2987004 | 0 | 86506 | 3.912109 | H | 4497 | 514.0 | 150.0 | master |

5 rows × 537 columns

In [77]:
```
# Label encode the variables
for col in cat_cols:
    lbl        = LabelEncoder()
    lbl.fit(list(df[col].values))
    df[col] = lbl.transform(list(df[col].values))
```

In [78]:
```
df = reduce_mem_usage(df)
```

```
Mem. usage decreased to 384.45 Mb (81.5% reduction)
```

# 10. Data Preprocessing for Model Building

In [79]:
```
df.loc[:, 'isFraud'].value_counts()
```

Out[79]:
```
0    569877
1     20663
Name: isFraud, dtype: int64
```

In [80]:
```
df = df.drop(['TransactionID','TransactionDT','Date'], axis=1)
```

In [81]:
```
# Split the y variable series and x variables dataset
X = df.drop(['isFraud'],axis=1)
y = df.isFraud.astype(bool)

# Delete train df
del df

# Collect garbage
gc.collect()
```

Out[81]:
```
0
```

In [82]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, ran
```

In [83]:
```
X_train.head()
```

Out[83]:

| | TransactionAmt | ProductCD | card1 | card2 | card3 | card4 | card5 | card6 | addr1 | addr2 | dist1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **448539** | 4.679688 | 4 | 6598 | 111.0 | 150.0 | 2 | 195.0 | 2 | 264.0 | 87.0 | 6.0 |
| **321311** | 4.355469 | 4 | 12839 | 321.0 | 150.0 | 4 | 226.0 | 2 | 264.0 | 87.0 | 0.0 |
| **497320** | 3.892578 | 4 | 14649 | 548.0 | 150.0 | 4 | 226.0 | 2 | 441.0 | 87.0 | 86.0 |
| **350951** | 4.058594 | 4 | 6489 | 295.0 | 150.0 | 4 | 226.0 | 2 | 184.0 | 87.0 | NaN |
| **98132** | 5.296875 | 2 | 5714 | 170.0 | 150.0 | 4 | 195.0 | 1 | 498.0 | 87.0 | NaN |

5 rows × 533 columns

# 11. Model Building

# XGBoost Classifier

In [ ]:
```
```

In [84]:
```
X_train = X_train.replace([np.inf, -np.inf], np.nan)
X_test = X_test.replace([np.inf, -np.inf], np.nan)

median_values = X_train.median()
X_train = X_train.fillna(median_values)
X_test = X_test.fillna(median_values)
```

In [85]:
```
%%time
xgb = XGBClassifier(nthread=-1, random_state=0)
xgb.fit(X_train, y_train)
xgb
```

```
[13:34:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation met
ric used with the objective 'binary:logistic' was changed from 'error' to 'log
loss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Wall time: 7min 4s
```

Out[85]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=8, nthread=-1, num_parallel_tree=1,
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

In [86]:
```
y_pred_xgb = xgb.predict(X_test)
y_prob_pred_xgb = xgb.predict_proba(X_test)
y_prob_pred_xgb = [x[1] for x in y_prob_pred_xgb]
print("Y predicted : ",y_pred_xgb)
print("Y probability predicted : ",y_prob_pred_xgb[:5])
```

```
Y predicted :  [False False False ... False False False]
Y probability predicted :  [0.0032196122, 0.0040678284, 0.007803707, 0.0231334
03, 0.0024363862]
```

# 12. Evaluation Metrics

- Accuracy Score
- Confusion Matrix
- Classification Report
- AUC Score
- Concordance Index
- ROC curve
- PR curve

In [87]:
```
from bisect import bisect_left, bisect_right

def concordance(actuals, preds):
    ones_preds  = [p for a,p in zip(actuals, preds) if a == 1]
    zeros_preds = [p for a,p in zip(actuals, preds) if a == 0]
    n_ones      = len([x for x in actuals if x == 1])
    n_total_pairs =  float(n_ones) * float(len(actuals) - n_ones)
    # print("Total Pairs: ", n_total_pairs)

    zeros_sorted = sorted(zeros_preds)

    conc = 0; disc = 0; ties = 0;
    for i, one_pred in enumerate(ones_preds):
        cur_conc = bisect_left(zeros_sorted, one_pred)
        cur_ties = bisect_right(zeros_sorted, one_pred) - cur_conc
        conc += cur_conc
        ties += cur_ties
```

```
        disc += float(len(zeros_sorted)) - cur_ties - cur_conc

    concordance = conc/n_total_pairs
    discordance = disc/n_total_pairs
    ties_perc = ties/n_total_pairs
    return concordance
```

In [88]:
```
def compute_evaluation_metric(model, x_test, y_actual, y_predicted, y_predicte
    print("\nAccuracy Score : ", accuracy_score(y_actual, y_predicted))
    print("\nAUC Score : ", roc_auc_score(y_actual, y_predicted_prob))
    print("\nConfusion Matrix : \n", confusion_matrix(y_actual, y_predicted))
    print("\nClassification Report : \n", classification_report(y_actual, y_pr

    # Concordance index if function exists
    try:
        print("\nConcordance Index : ", concordance(y_actual, y_predicted_prob
    except:
        print("\nConcordance Index : Function 'concordance' not defined")

    # ROC curve
    print("\nROC curve : \n")
    plot_roc_curve(model, x_test, y_actual)
    plt.show()

    # PR curve
    print("\nPR curve : \n")
    plot_precision_recall_curve(model, x_test, y_actual)
    plt.show()

    # --- Compute TPR, FPR, TNR, FNR ---
    tn, fp, fn, tp = confusion_matrix(y_actual, y_predicted).ravel()

    tpr = tp / (tp + fn)  # True Positive Rate / Recall
    fpr = fp / (fp + tn)  # False Positive Rate
    tnr = tn / (tn + fp)  # True Negative Rate / Specificity
    fnr = fn / (fn + tp)  # False Negative Rate

    print("\nAdditional Metrics:")
    print(f"TPR (Recall) : {tpr:.4f}")
    print(f"FPR : {fpr:.4f}")
    print(f"TNR (Specificity) : {tnr:.4f}")
    print(f"FNR : {fnr:.4f}")
```

In [89]:
```
concordance(y_test.values, y_prob_pred_xgb)
```

Out[89]:
```
0.9363753273299149
```

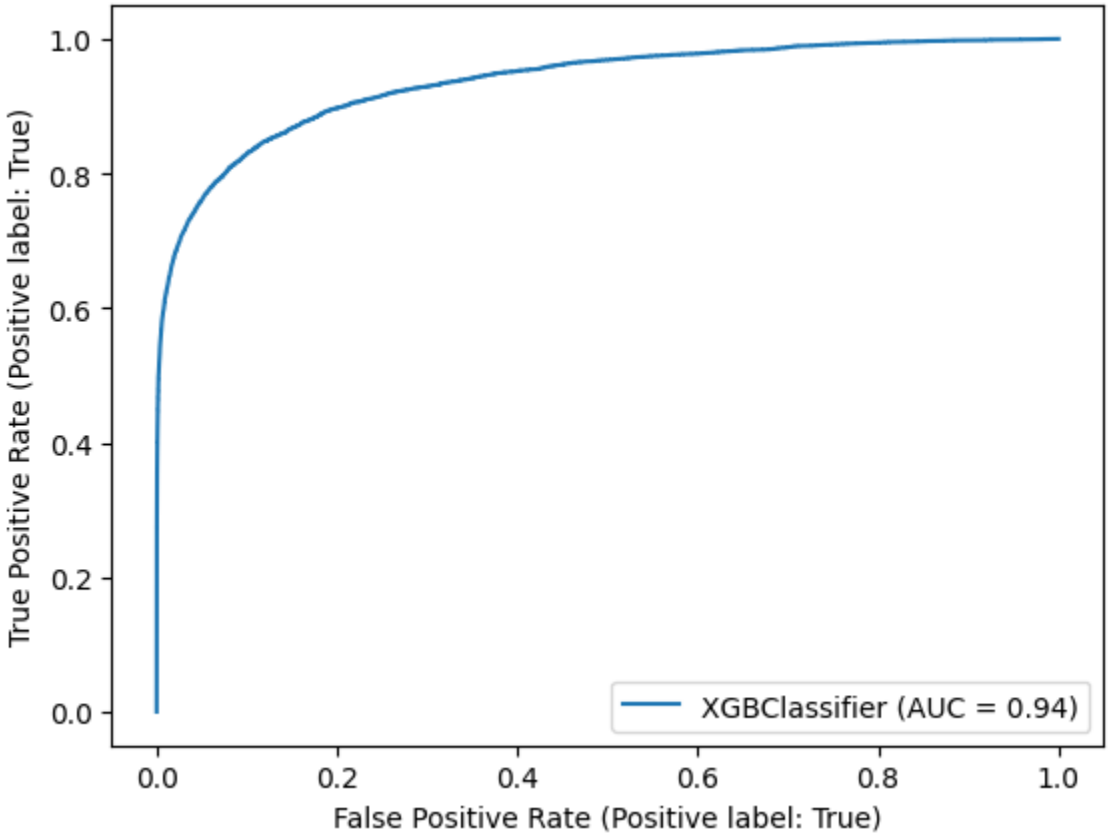
In [90]:
```
compute_evaluation_metric(xgb, X_test, y_test, y_pred_xgb, y_prob_pred_xgb)
```

```
Accuracy Score :  0.9801086011672933

AUC Score :  0.9363753419553209

Confusion Matrix :
 [[170646    317]
 [  3207   2992]]

Classification Report :
               precision    recall  f1-score   support

       False       0.98      1.00      0.99    170963
```

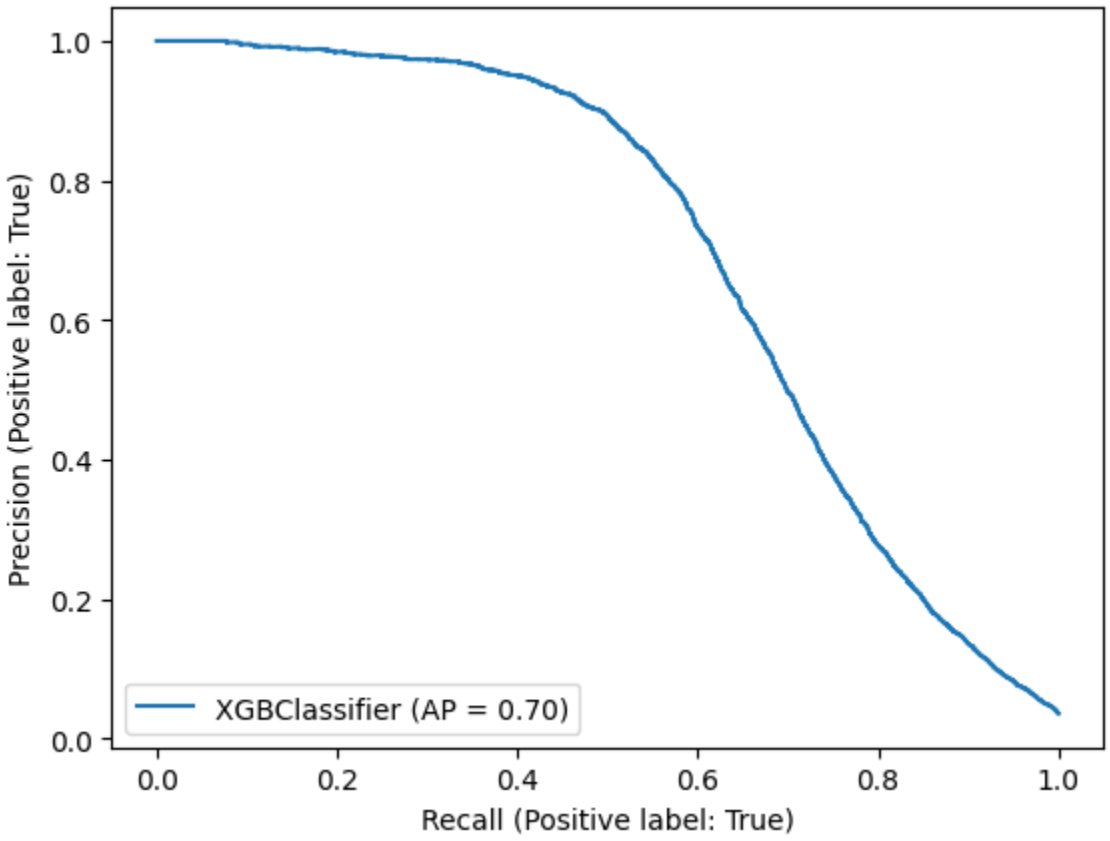
```
        True       0.90      0.48      0.63      6199

    accuracy                           0.98    177162
   macro avg       0.94      0.74      0.81    177162
weighted avg       0.98      0.98      0.98    177162
```

```
Concordance Index :  0.9363753273299149
```

ROC curve :




PR curve :




```
Additional Metrics:
TPR (Recall) : 0.4827
FPR : 0.0019
TNR (Specificity) : 0.9981
FNR : 0.5173
```

# 13. Capture Rates and Calibration Curve

Divide the data in 10 equal bins as per predicted probability scores. Then, compute the percentage of the total target class 1 captured in every bin.

Ideally the proportion should be decreasing as we go down ever bin. Let's check it out

In [91]:
```
# Create Validation set
validation_df = {'y_test' : y_test, 'y_pred' : y_pred_xgb, 'y_pred_prob' : y_p
validation_df = pd.DataFrame(data = validation_df)

# Add binning column to the dataframe
validation_df['bin_y_pred_prob'] = pd.qcut(validation_df['y_pred_prob'], q=10)
validation_df.head()
```

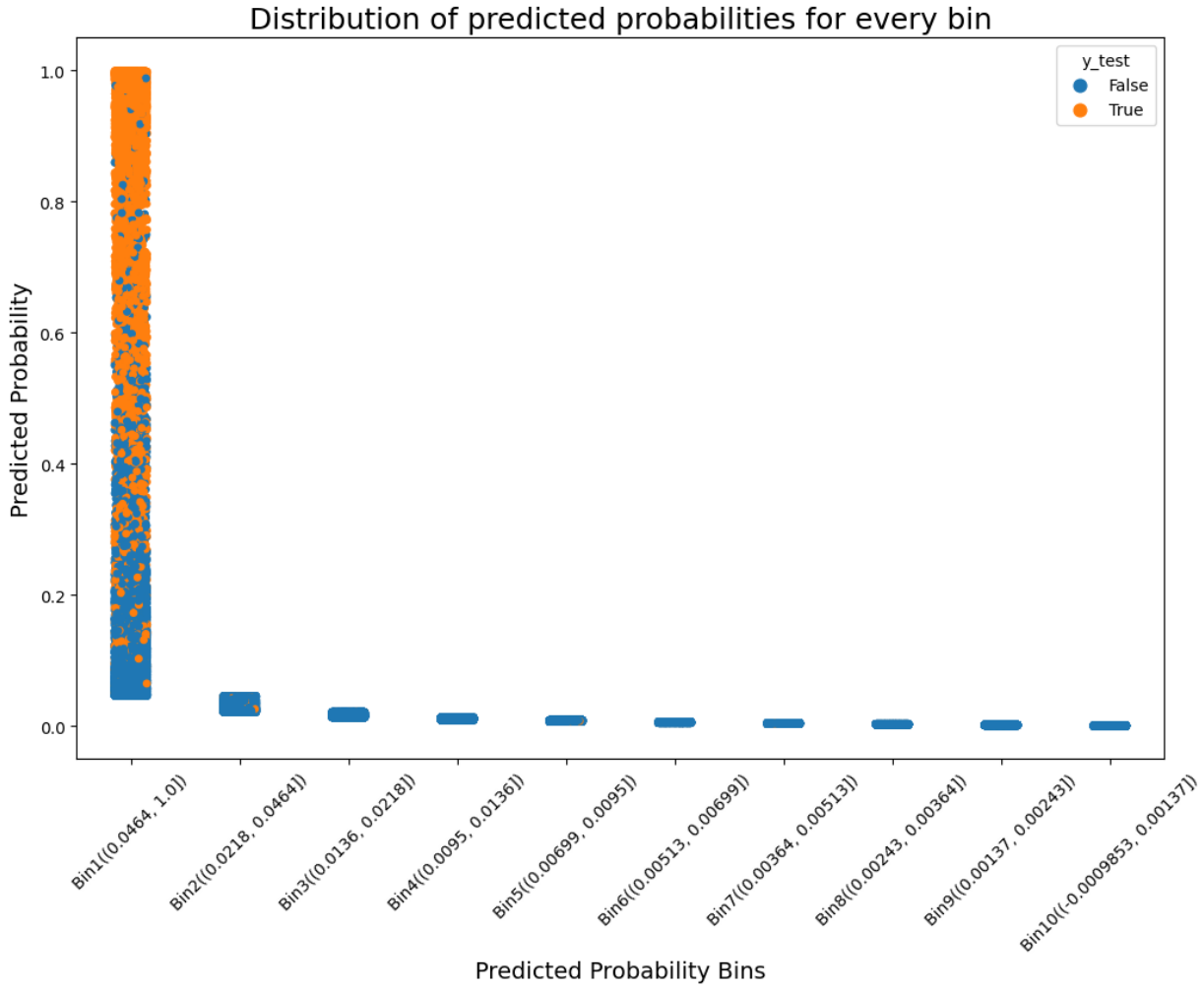
Out[91]:

| | y_test | y_pred | y_pred_prob | bin_y_pred_prob |
|---|---|---|---|---|
| **274635** | False | False | 0.003220 | (0.00243, 0.00364] |
| **115501** | False | False | 0.004068 | (0.00364, 0.00513] |
| **387765** | False | False | 0.007804 | (0.00699, 0.0095] |
| **123591** | False | False | 0.023133 | (0.0218, 0.0464] |
| **462833** | False | False | 0.002436 | (0.00243, 0.00364] |

In [92]:
```
x_label = []
for i in range(len(validation_df['bin_y_pred_prob'].cat.categories[::-1].astyp
    x_label.append("Bin" + str(i + 1)+ "(" + validation_df['bin_y_pred_prob'].
```

## Capture Rates Plot

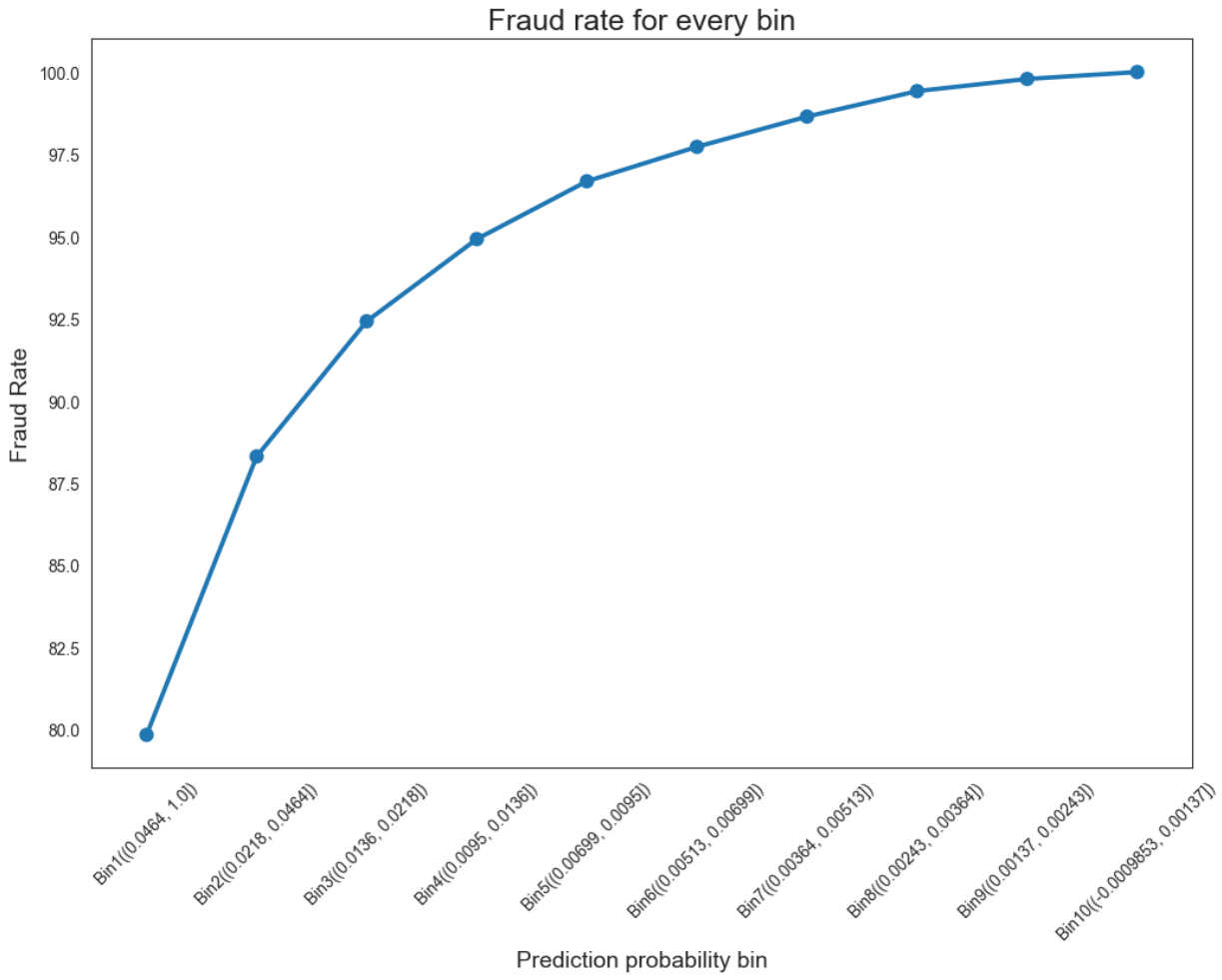
In [93]:
```
# Plot Distribution of predicted probabilities for every bin
plt.figure(figsize=(12, 8));
sns.stripplot(validation_df.bin_y_pred_prob, validation_df.y_pred_prob, jitter
plt.title("Distribution of predicted probabilities for every bin", fontsize=18
plt.xlabel("Predicted Probability Bins", fontsize=14);
plt.ylabel("Predicted Probability", fontsize=14);
plt.xticks(np.arange(10), x_label, rotation=45);
plt.show()
```




## Gains Table

In [94]:
```
# Aggregate the data
gains_df             = validation_df.groupby(["bin_y_pred_prob","y_test"]).agg
gains_df.columns     = gains_df.columns.map(''.join)
gains_df['prob_bin'] = gains_df.index.get_level_values(0)
gains_df['y_test']   = gains_df.index.get_level_values(1)
gains_df.reset_index(drop = True, inplace = True)
gains_df

# Get infection rate and percentage infections
gains_table = gains_df.pivot(index='prob_bin', columns='y_test', values='y_tes
```

```
gains_table['prob_bin'] = gains_table.index
gains_table = gains_table.iloc[::-1]
gains_table['prob_bin'] = x_label
gains_table.reset_index(drop = True, inplace = True)
gains_table = gains_table[['prob_bin', 0, 1]]
gains_table.columns = ['prob_bin', "not_fraud", "fraud"]
gains_table['perc_fraud'] = gains_table['fraud']/gains_table['fraud'].sum()
gains_table['perc_not_fraud'] = gains_table['not_fraud']/gains_table['not_frau
gains_table['cum_perc_fraud'] = 100*(gains_table.fraud.cumsum() / gains_table.
gains_table['cum_perc_not_fraud'] = 100*(gains_table.not_fraud.cumsum() / gain
gains_table


# Plot
plt.figure(figsize=(12, 8));
sns.set_style("white")
sns.pointplot(x = "prob_bin", y = "cum_perc_fraud", data = gains_table, legend
plt.xticks(rotation=45);
plt.ylabel("Fraud Rate", fontsize=14)
plt.xlabel("Prediction probability bin", fontsize=14)
plt.title("Fraud rate for every bin", fontsize=18)
plt.show()
```

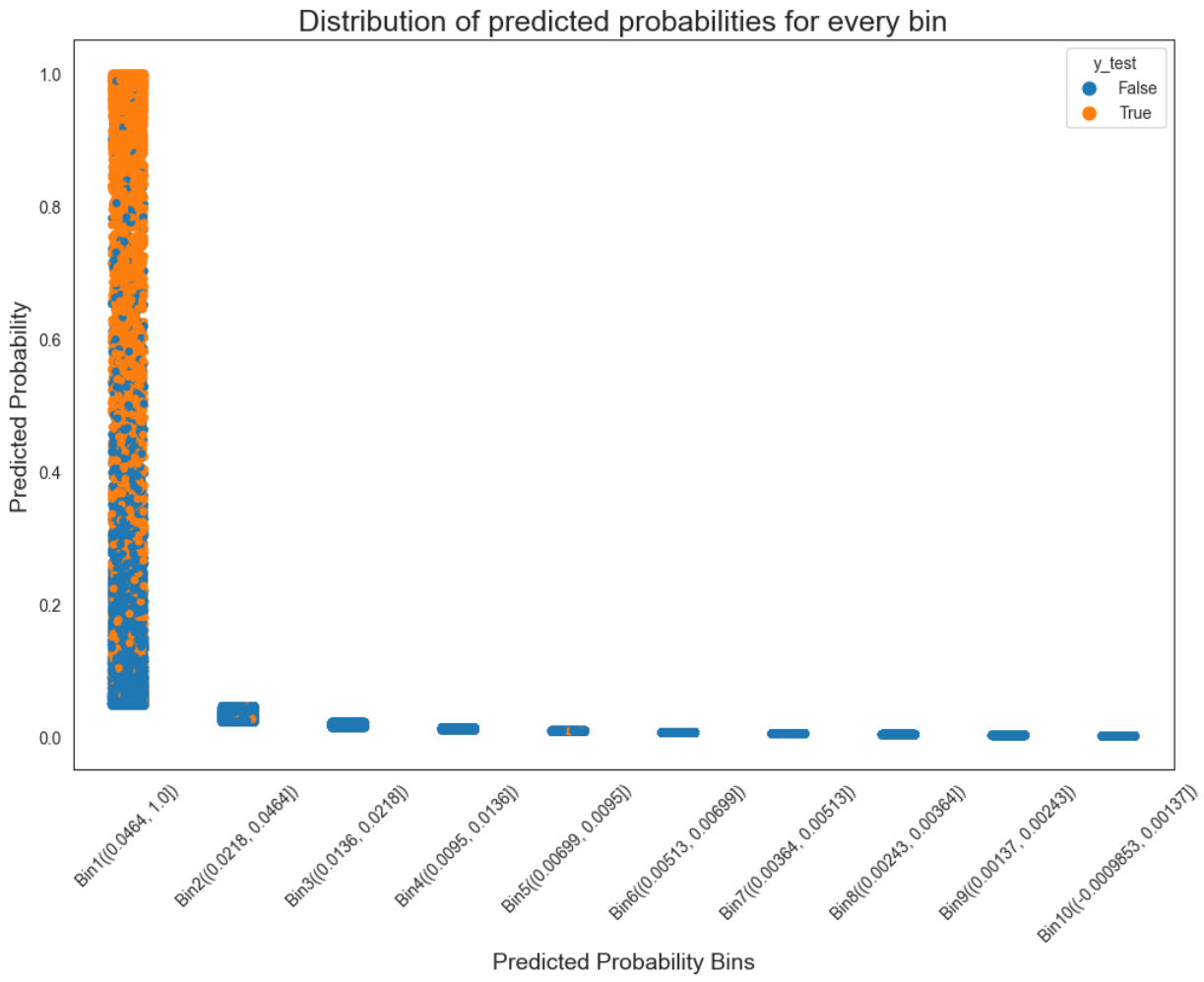


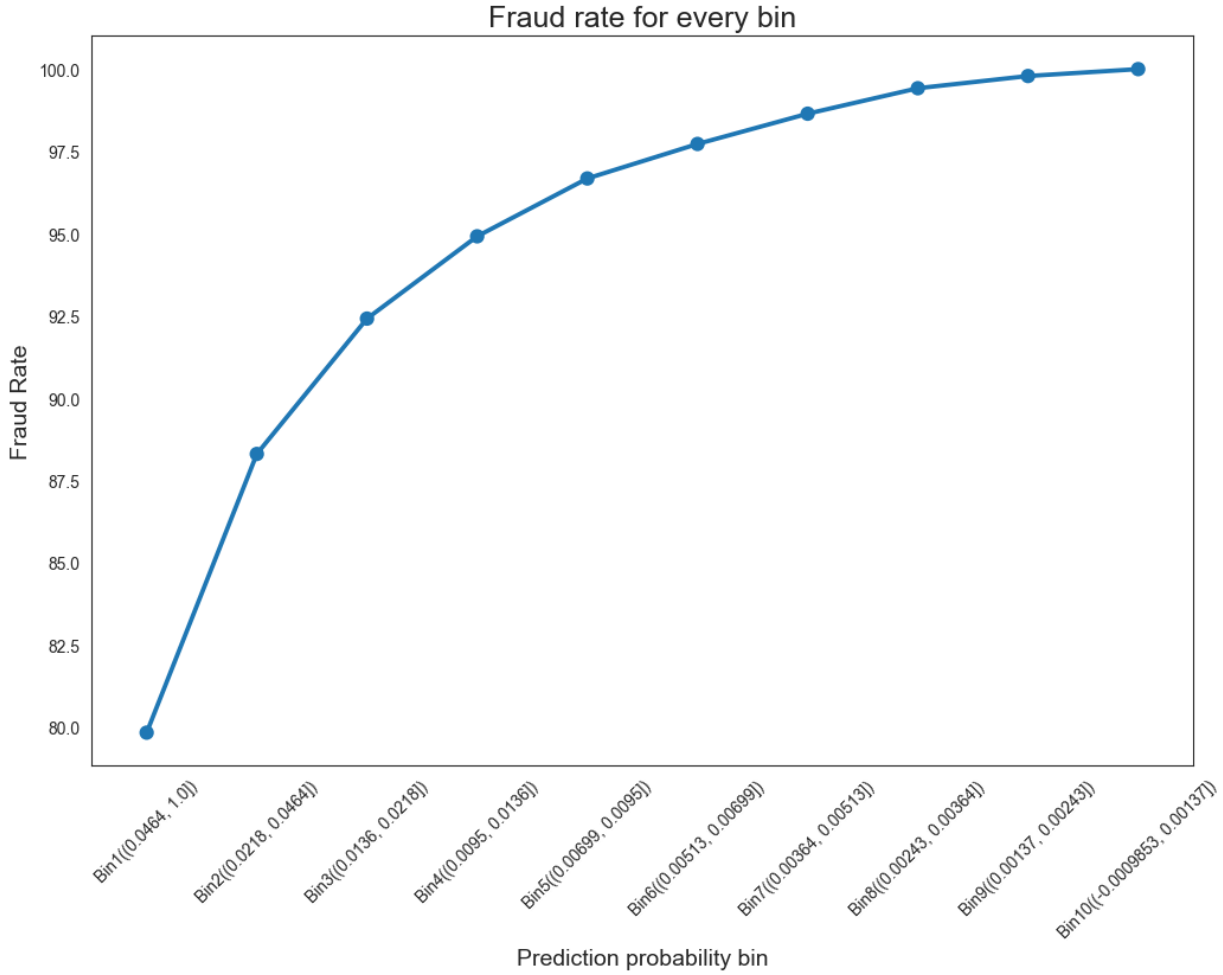

Ideally the slope should be high initially and should decrease as we move further to the right. This is not really a good model.

In [95]:
```
def captures(y_test, y_pred, y_pred_prob):
    # Create Validation set
    validation_df = {'y_test' : y_test, 'y_pred' : y_pred, 'y_pred_prob' : y_p
```

```
    validation_df = pd.DataFrame(data = validation_df)

    # Add binning column to the dataframe
    try:
        validation_df['bin_y_pred_prob'] = pd.qcut(validation_df['y_pred_prob'
    except:
        validation_df['bin_y_pred_prob'] = pd.qcut(validation_df['y_pred_prob'

    # Change x label and column names
    x_label = []
    for i in range(len(validation_df['bin_y_pred_prob'].cat.categories[::-1].a
        x_label.append("Bin" + str(i + 1)+ "(" + validation_df['bin_y_pred_pro

    # Plot Distribution of predicted probabilities for every bin
    plt.figure(figsize=(12, 8));
    sns.stripplot(validation_df.bin_y_pred_prob, validation_df.y_pred_prob, ji
    plt.title("Distribution of predicted probabilities for every bin", fontsiz
    plt.xlabel("Predicted Probability Bins", fontsize=14);
    plt.ylabel("Predicted Probability", fontsize=14);
    try:
        plt.xticks(np.arange(10), x_label, rotation=45);
    except:
        pass
    plt.show()

    # Aggregate the data
    gains_df             = validation_df.groupby(["bin_y_pred_prob","y_test"])
    gains_df.columns     = gains_df.columns.map(''.join)
    gains_df['prob_bin'] = gains_df.index.get_level_values(0)
    gains_df['y_test']   = gains_df.index.get_level_values(1)
    gains_df.reset_index(drop = True, inplace = True)
    gains_df

    # Get infection rate and percentage infections
    gains_table = gains_df.pivot(index='prob_bin', columns='y_test', values='y
    gains_table['prob_bin'] = gains_table.index
    gains_table = gains_table.iloc[::-1]
    gains_table['prob_bin'] = x_label
    gains_table.reset_index(drop = True, inplace = True)
    gains_table = gains_table[['prob_bin', 0, 1]]
    gains_table.columns = ['prob_bin', "not_fraud", "fraud"]
    gains_table['perc_fraud'] = gains_table['fraud']/gains_table['fraud'].sum(
    gains_table['perc_not_fraud'] = gains_table['not_fraud']/gains_table['not_
    gains_table['cum_perc_fraud'] = 100*(gains_table.fraud.cumsum() / gains_ta
    gains_table['cum_perc_not_fraud'] = 100*(gains_table.not_fraud.cumsum() /
    gains_table


    # Plot
    plt.figure(figsize=(12, 8));
    sns.set_style("white")
    sns.pointplot(x = "prob_bin", y = "cum_perc_fraud", data = gains_table, le
    plt.xticks(rotation=45);
    plt.ylabel("Fraud Rate", fontsize=14)
    plt.xlabel("Prediction probability bin", fontsize=14)
    plt.title("Fraud rate for every bin", fontsize=18)
    plt.show()
    return gains_table
```

In [96]:
```
captures(y_test, y_pred_xgb, y_prob_pred_xgb)
```



Out[96]:

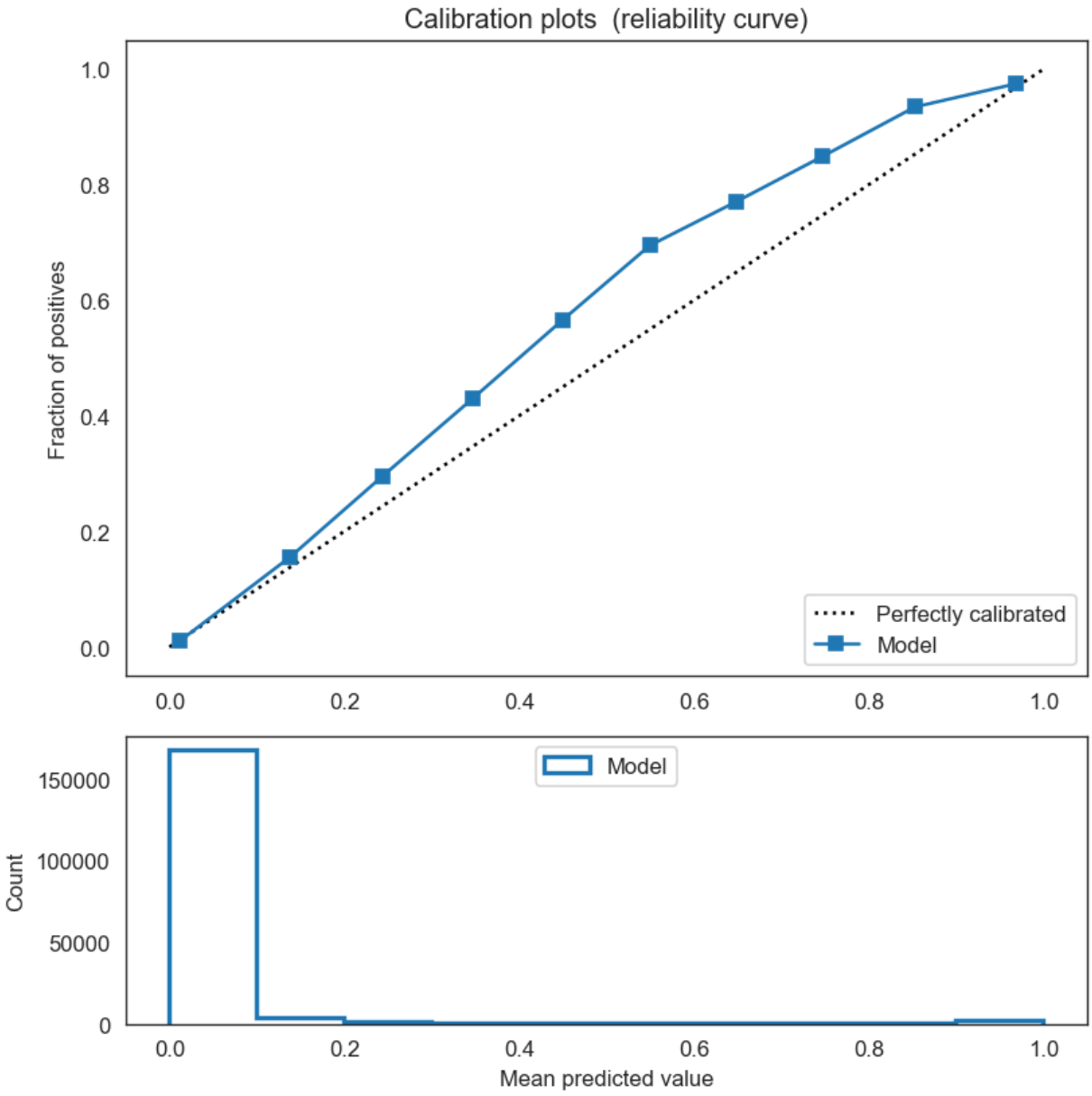
| | prob_bin | not_fraud | fraud | perc_fraud | perc_not_fraud | cum_perc_fraud | cum_perc_not_ |
|---|---|---|---|---|---|---|---|
| **0** | Bin1((0.0464, 1.0]) | 12768 | 4949 | 0.798355 | 0.074683 | 79.835457 | 7.46 |
| **1** | Bin2((0.0218, 0.0464]) | 17191 | 525 | 0.084691 | 0.100554 | 88.304565 | 17.52 |
| **2** | Bin3((0.0136, 0.0218]) | 17461 | 255 | 0.041136 | 0.102133 | 92.418132 | 27.73 |
| **3** | Bin4((0.0095, 0.0136]) | 17561 | 155 | 0.025004 | 0.102718 | 94.918535 | 38.00 |
| **4** | Bin5((0.00699, 0.0095]) | 17607 | 109 | 0.017583 | 0.102987 | 96.676883 | 48.30 |
| **5** | Bin6((0.00513, 0.00699]) | 17651 | 65 | 0.010486 | 0.103245 | 97.725440 | 58.63 |
| **6** | Bin7((0.00364, 0.00513]) | 17659 | 57 | 0.009195 | 0.103291 | 98.644943 | 68.96 |
| **7** | Bin8((0.00243, 0.00364]) | 17668 | 48 | 0.007743 | 0.103344 | 99.419261 | 79.29 |
| **8** | Bin9((0.00137, 0.00243]) | 17693 | 23 | 0.003710 | 0.103490 | 99.790289 | 89.64 |
| **9** | Bin10((-0.0009853, 0.00137]) | 17704 | 13 | 0.002097 | 0.103555 | 100.000000 | 100.00 |

# Calibration Curve

In [97]:
```
from sklearn.calibration import calibration_curve
import matplotlib.pyplot as plt
```

In [98]:
```
def draw_calibration_curve(y_test, y_prob, n_bins=10):
    plt.figure(figsize=(7, 7), dpi=120)
    ax1 = plt.subplot2grid((3, 1), (0, 0), rowspan=2)
    ax2 = plt.subplot2grid((3, 1), (2, 0))
    ax1.plot([0, 1], [0, 1], "k:", label="Perfectly calibrated")


    fraction_of_positives, mean_predicted_value = calibration_curve(y_test, y_

    ax1.plot(mean_predicted_value, fraction_of_positives, "s-", label="%s" % (
    ax2.hist(y_prob, range=(0, 1), bins=10, label="Model", histtype="step", lw

    # Labels
    ax1.set_ylabel("Fraction of positives")
    ax1.set_ylim([-0.05, 1.05])
    ax1.legend(loc="lower right")
    ax1.set_title('Calibration plots  (reliability curve)')

    ax2.set_xlabel("Mean predicted value")
    ax2.set_ylabel("Count")
    ax2.legend(loc="upper center", ncol=2)

    plt.tight_layout()
    plt.show()
```

In [99]:
```
draw_calibration_curve(y_test, y_prob_pred_xgb, n_bins=10)
```




# 14. Model Comparison and Evaluation

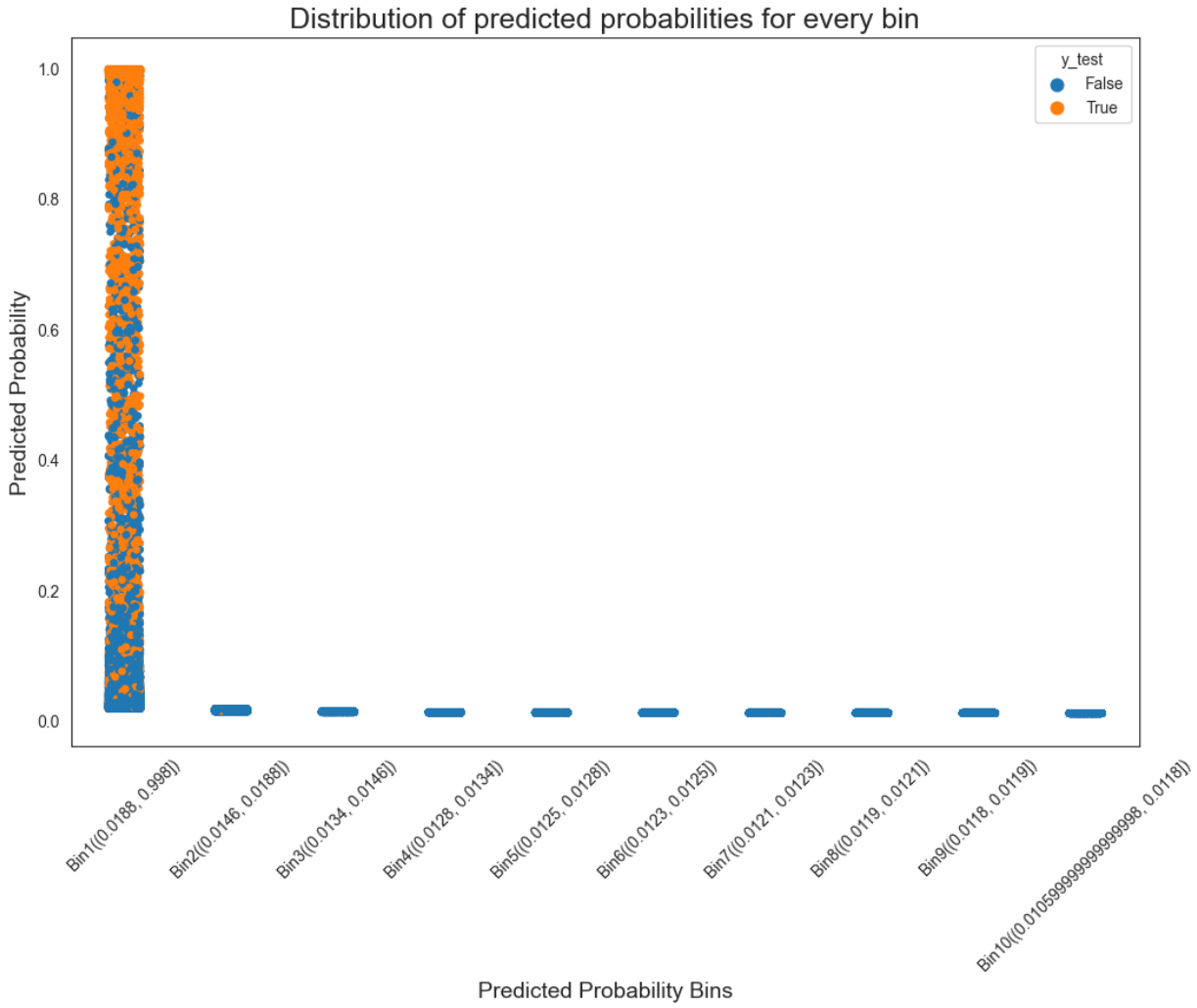
## A) Logistic regression

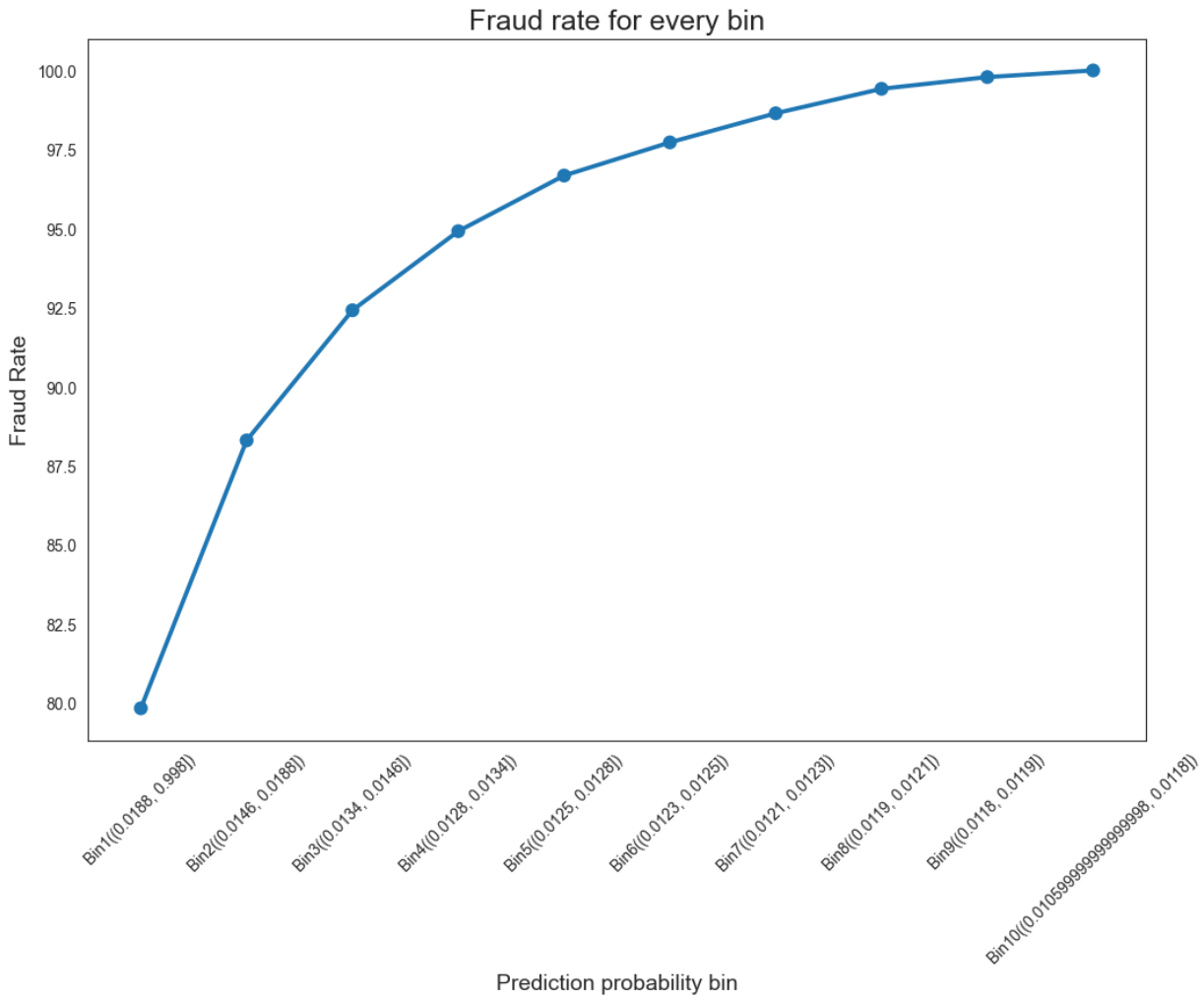
In [100…
```
# Prediction
y_pred_xgb_test = xgb.predict(X_test)
y_prob_pred_xgb_test = xgb.predict_proba(X_test)[:, 1]
```

In [101…
```
from sklearn.linear_model import LogisticRegression
X = np.array(y_prob_pred_xgb_test)
clf = LogisticRegression(random_state=0).fit(X.reshape(-1, 1), y_test)
```

In [102…
```
y_prob_pred_calib = clf.predict_proba(X.reshape(-1, 1))[:, 1]
y_pred_calib      = clf.predict(X.reshape(-1, 1))
```

In [103…
```
captures(y_test, y_pred_calib, y_prob_pred_calib)
```



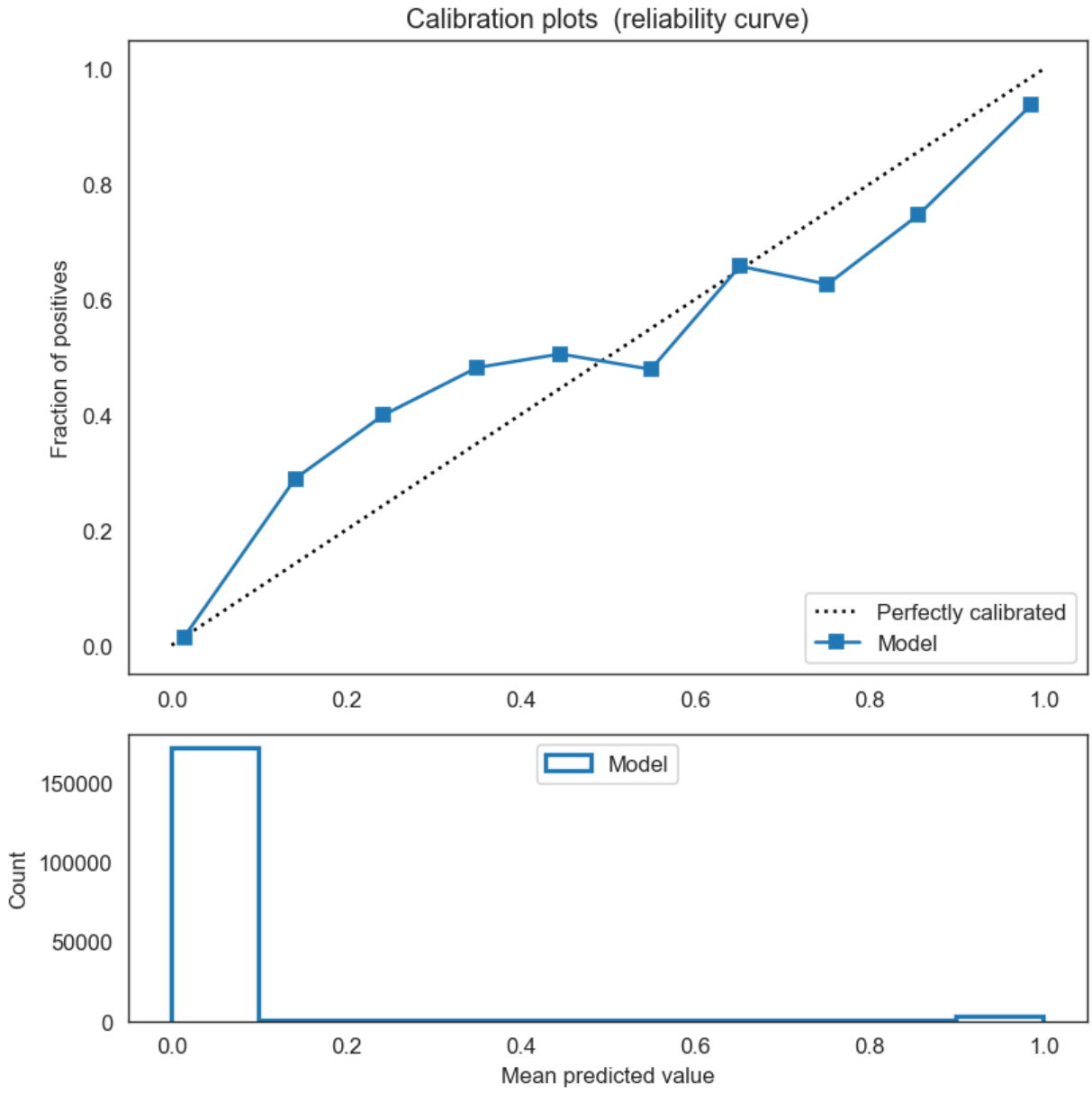



Out[103]:

| | prob_bin | not_fraud | fraud | perc_fraud | perc_not_fraud | cum_perc_fraud | cu |
|---|---|---|---|---|---|---|---|
| **0** | Bin1((0.0188, 0.998]) | 12768 | 4949 | 0.798355 | 0.074683 | 79.835457 | |
| **1** | Bin2((0.0146, 0.0188]) | 17191 | 525 | 0.084691 | 0.100554 | 88.304565 | |
| **2** | Bin3((0.0134, 0.0146]) | 17461 | 255 | 0.041136 | 0.102133 | 92.418132 | |
| **3** | Bin4((0.0128, 0.0134]) | 17561 | 155 | 0.025004 | 0.102718 | 94.918535 | |
| **4** | Bin5((0.0125, 0.0128]) | 17607 | 109 | 0.017583 | 0.102987 | 96.676883 | |
| **5** | Bin6((0.0123, 0.0125]) | 17651 | 65 | 0.010486 | 0.103245 | 97.725440 | |
| **6** | Bin7((0.0121, 0.0123]) | 17659 | 57 | 0.009195 | 0.103291 | 98.644943 | |
| **7** | Bin8((0.0119, 0.0121]) | 17668 | 48 | 0.007743 | 0.103344 | 99.419261 | |
| **8** | Bin9((0.0118, 0.0119]) | 17693 | 23 | 0.003710 | 0.103490 | 99.790289 | |
| **9** | Bin10((0.010599999999999998, 0.0118]) | 17704 | 13 | 0.002097 | 0.103555 | 100.000000 | |

In [104…

```
draw_calibration_curve(y_test, y_prob_pred_calib, n_bins=10)
```




# B) LightGBM

In [107…
```
from lightgbm import LGBMClassifier
```

In [108…
```
%%time
lgbc = LGBMClassifier(random_state=0, n_jobs = -1)
lgbc.fit(X_train,y_train)
lgbc
```

```
Wall time: 29.5 s
```

Out[108]:
```
LGBMClassifier(random_state=0)
```

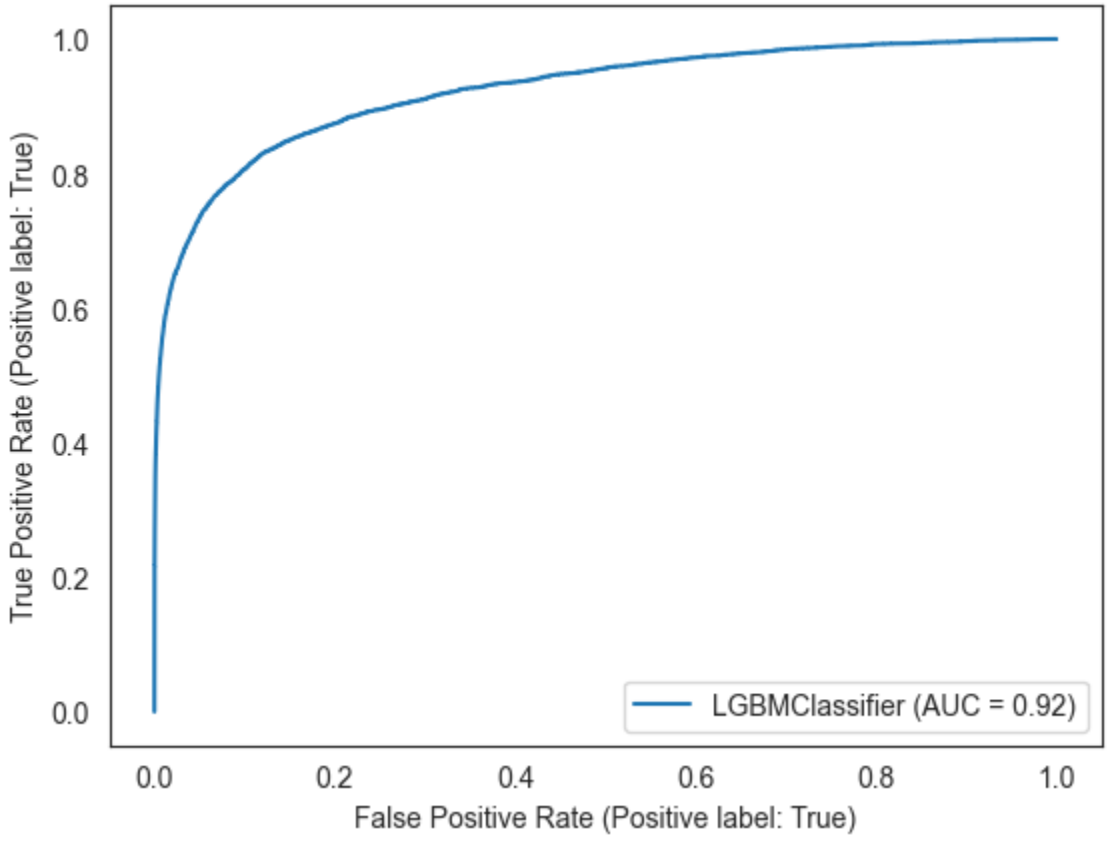
In [109…
```
y_pred_lgbc = lgbc.predict(X_test)
y_prob_pred_lgbc = lgbc.predict_proba(X_test)
y_prob_pred_lgbc = [x[1] for x in y_prob_pred_lgbc]
print("Y predicted : ",y_pred_lgbc)
print("Y probability predicted : ",y_prob_pred_lgbc[:5])
```

```
Y predicted :  [False False False ... False False False]
Y probability predicted :  [0.016160027443867835, 0.0080773782284121, 0.008255
```

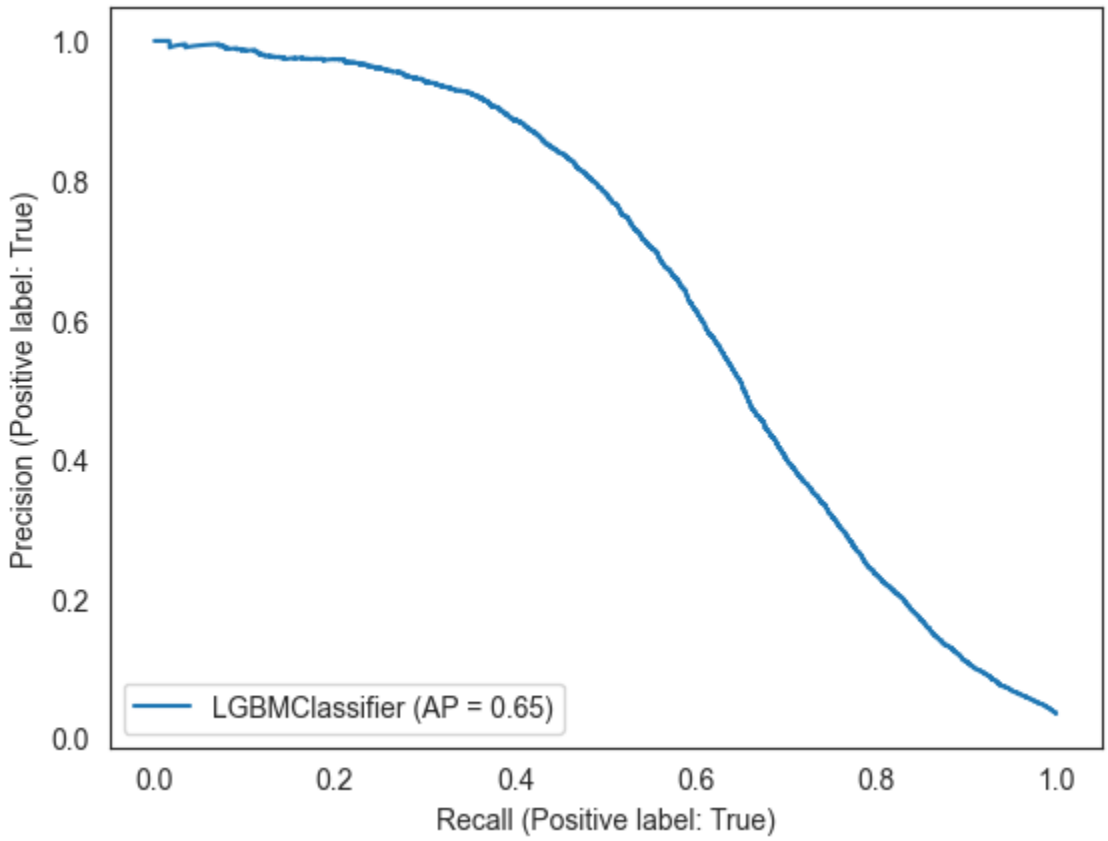
```
09576482752, 0.06708271917846045, 0.007287115458060899]
```

In [110…
```
compute_evaluation_metric(lgbc, X_test, y_test, y_pred_lgbc, y_prob_pred_lgbc)
```

```
Accuracy Score :  0.9774613066007383

AUC Score :  0.9239690737882372

Confusion Matrix :
 [[170585    378]
 [  3615   2584]]

Classification Report :
               precision    recall  f1-score   support

       False       0.98      1.00      0.99    170963
        True       0.87      0.42      0.56      6199

    accuracy                           0.98    177162
   macro avg       0.93      0.71      0.78    177162
weighted avg       0.98      0.98      0.97    177162


Concordance Index :  0.9239690501988727

ROC curve :
```



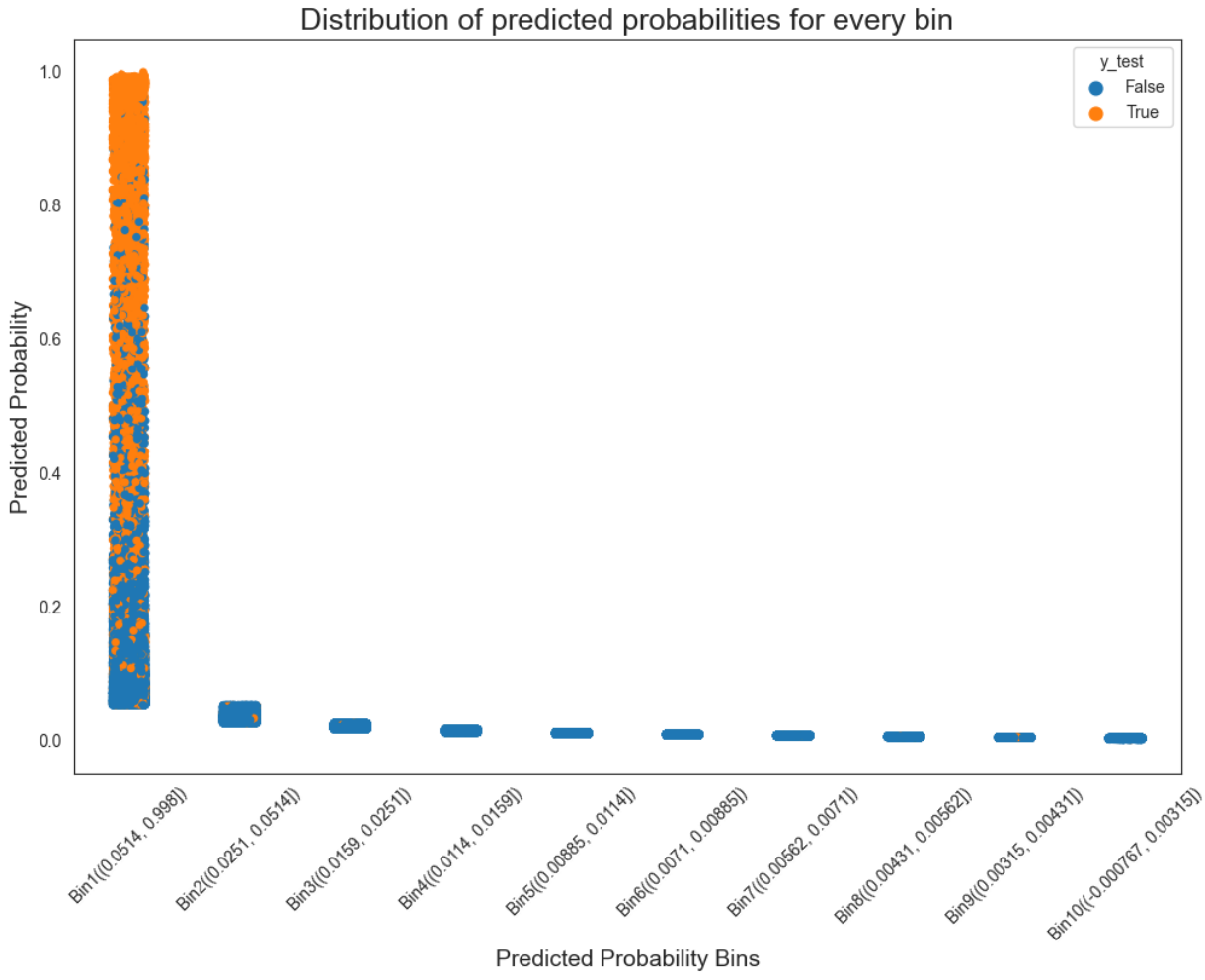

```
PR curve :
```

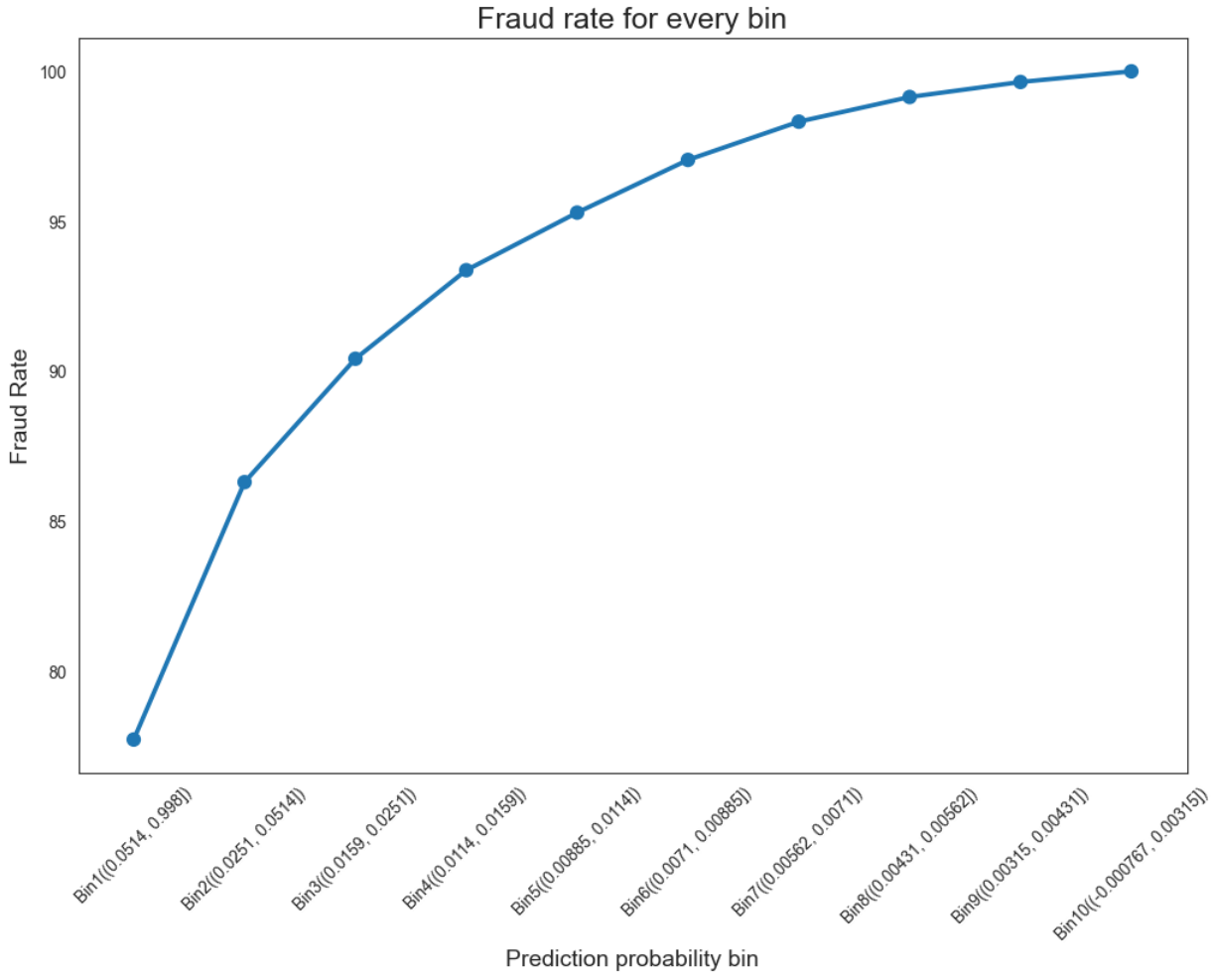



```
Additional Metrics:
TPR (Recall) : 0.4168
FPR : 0.0022
TNR (Specificity) : 0.9978
FNR : 0.5832
```

In [111…

```
captures(y_test, y_pred_lgbc, y_prob_pred_lgbc)
```



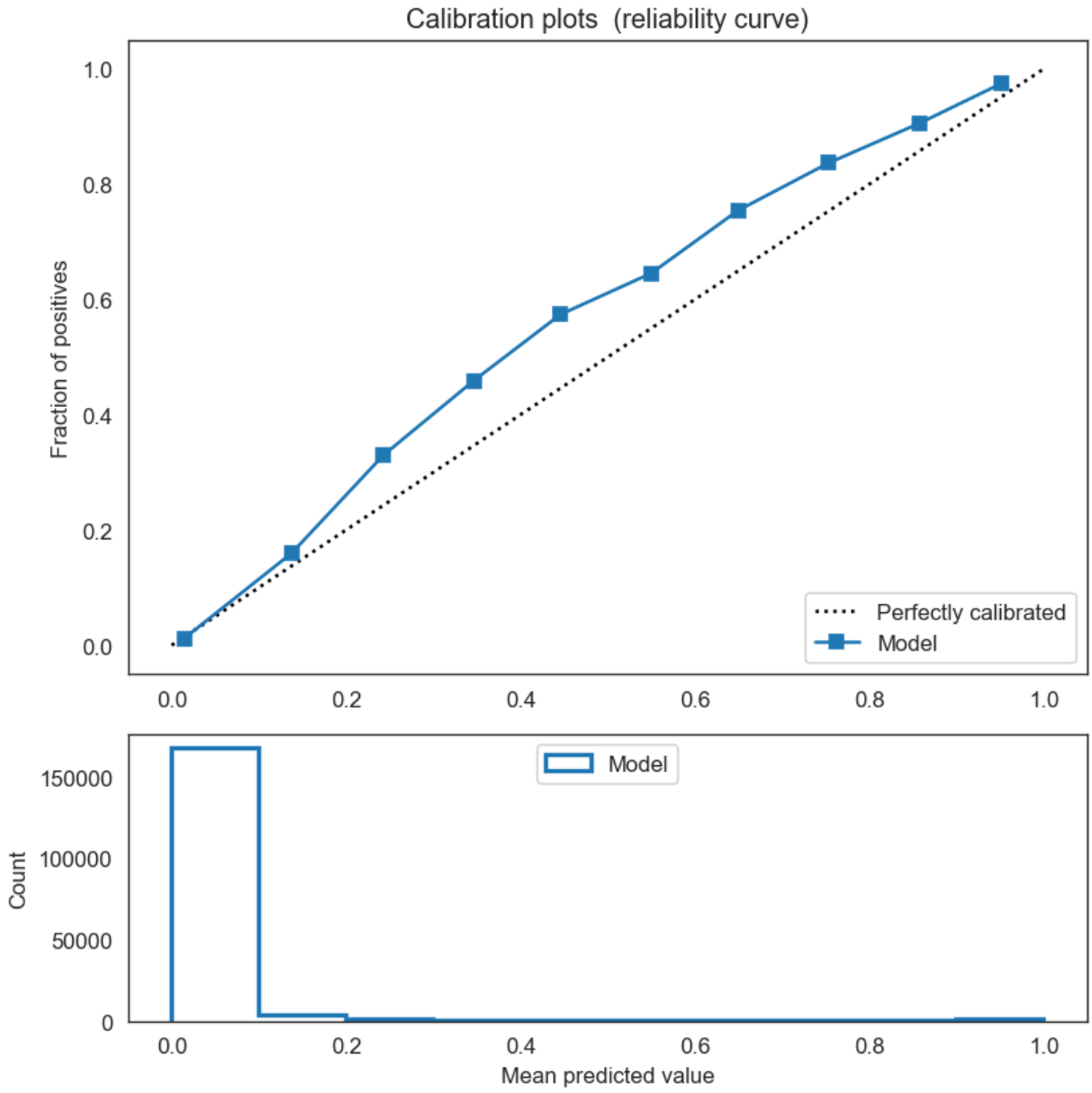



Out[111]:

| | prob_bin | not_fraud | fraud | perc_fraud | perc_not_fraud | cum_perc_fraud | cum_perc_not_ |
|---|---|---|---|---|---|---|---|
| **0** | Bin1((0.0514, 0.998]) | 12899 | 4818 | 0.777222 | 0.075449 | 77.722213 | 7.54 |
| **1** | Bin2((0.0251, 0.0514]) | 17184 | 532 | 0.085820 | 0.100513 | 86.304243 | 17.59 |
| **2** | Bin3((0.0159, 0.0251]) | 17461 | 255 | 0.041136 | 0.102133 | 90.417809 | 27.80 |
| **3** | Bin4((0.0114, 0.0159]) | 17533 | 183 | 0.029521 | 0.102554 | 93.369898 | 38.06 |
| **4** | Bin5((0.00885, 0.0114]) | 17597 | 119 | 0.019197 | 0.102929 | 95.289563 | 48.35 |
| **5** | Bin6((0.0071, 0.00885]) | 17607 | 109 | 0.017583 | 0.102987 | 97.047911 | 58.65 |
| **6** | Bin7((0.00562, 0.0071]) | 17637 | 79 | 0.012744 | 0.103163 | 98.322310 | 68.97 |
| **7** | Bin8((0.00431, 0.00562]) | 17665 | 51 | 0.008227 | 0.103326 | 99.145023 | 79.30 |
| **8** | Bin9((0.00315, 0.00431]) | 17685 | 31 | 0.005001 | 0.103443 | 99.645104 | 89.64 |
| **9** | Bin10((-0.000767, 0.00315]) | 17695 | 22 | 0.003549 | 0.103502 | 100.000000 | 100.00 |

In [112…

```
draw_calibration_curve(y_test, y_prob_pred_lgbc, n_bins=10)
```




- With LGBM, Accuracy score is 97.7%. It's almost similar to XGBoost model
- AUC score has imporved to 92.6 from 88.5
- Recall and f-1 score have also improved, but it's still not upto the mark

# C) Random Forest Classifier

In [113…
```
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, Gra
```

In [114…
```
X_train.head()
```

Out[114]:

| | TransactionAmt | ProductCD | card1 | card2 | card3 | card4 | card5 | card6 | addr1 | addr2 | dist |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **448539** | 4.679688 | 4 | 6598 | 111.0 | 150.0 | 2 | 195.0 | 2 | 264.0 | 87.0 | 6. |
| **321311** | 4.355469 | 4 | 12839 | 321.0 | 150.0 | 4 | 226.0 | 2 | 264.0 | 87.0 | 0. |
| **497320** | 3.892578 | 4 | 14649 | 548.0 | 150.0 | 4 | 226.0 | 2 | 441.0 | 87.0 | 86. |
| **350951** | 4.058594 | 4 | 6489 | 295.0 | 150.0 | 4 | 226.0 | 2 | 184.0 | 87.0 | 8. |

| | TransactionAmt | ProductCD | card1 | card2 | card3 | card4 | card5 | card6 | addr1 | addr2 | dist |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **98132** | 5.296875 | 2 | 5714 | 170.0 | 150.0 | 4 | 195.0 | 1 | 498.0 | 87.0 | 8. |

5 rows × 533 columns

In [118…
```
X_train.isnull().sum()
```

Out[118]:
```
TransactionAmt    0
ProductCD         0
card1             0
card2             0
card3             0
                 ..
PCA_V_25          0
PCA_V_26          0
PCA_V_27          0
PCA_V_28          0
PCA_V_29          0
Length: 533, dtype: int64
```

In [120…
```
%%time
rfc = RandomForestClassifier(random_state=0, n_jobs = -1)
rfc.fit(X_train, y_train)
rfc
```

```
Wall time: 2min 45s
```

Out[120]:
```
RandomForestClassifier(n_jobs=-1, random_state=0)
```

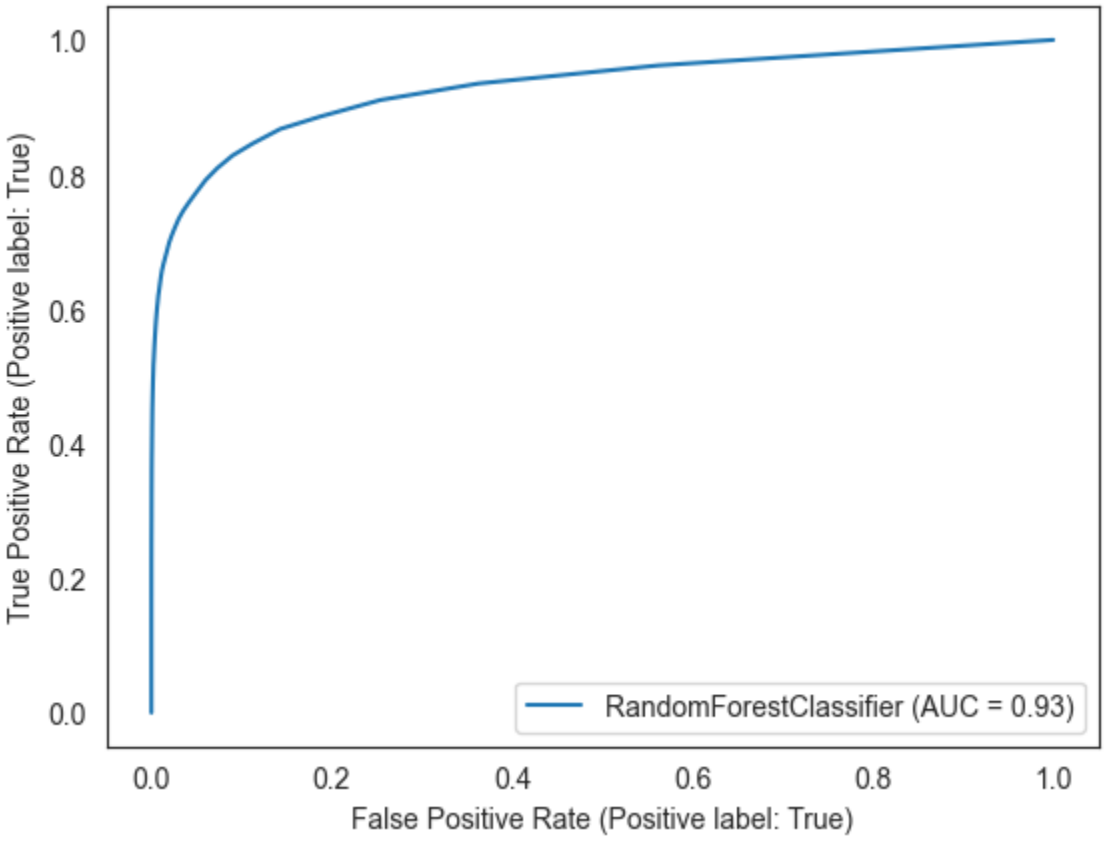
In [121…
```
y_pred_rfc = rfc.predict(X_test)
y_prob_pred_rfc = rfc.predict_proba(X_test)[:, 1]

print("Y predicted : ",y_pred_rfc)
print("Y probability predicted : ",y_prob_pred_rfc[:5])
```

```
Y predicted :  [False False False ... False False False]
Y probability predicted :  [0.02 0.   0.   0.06 0.02]
```

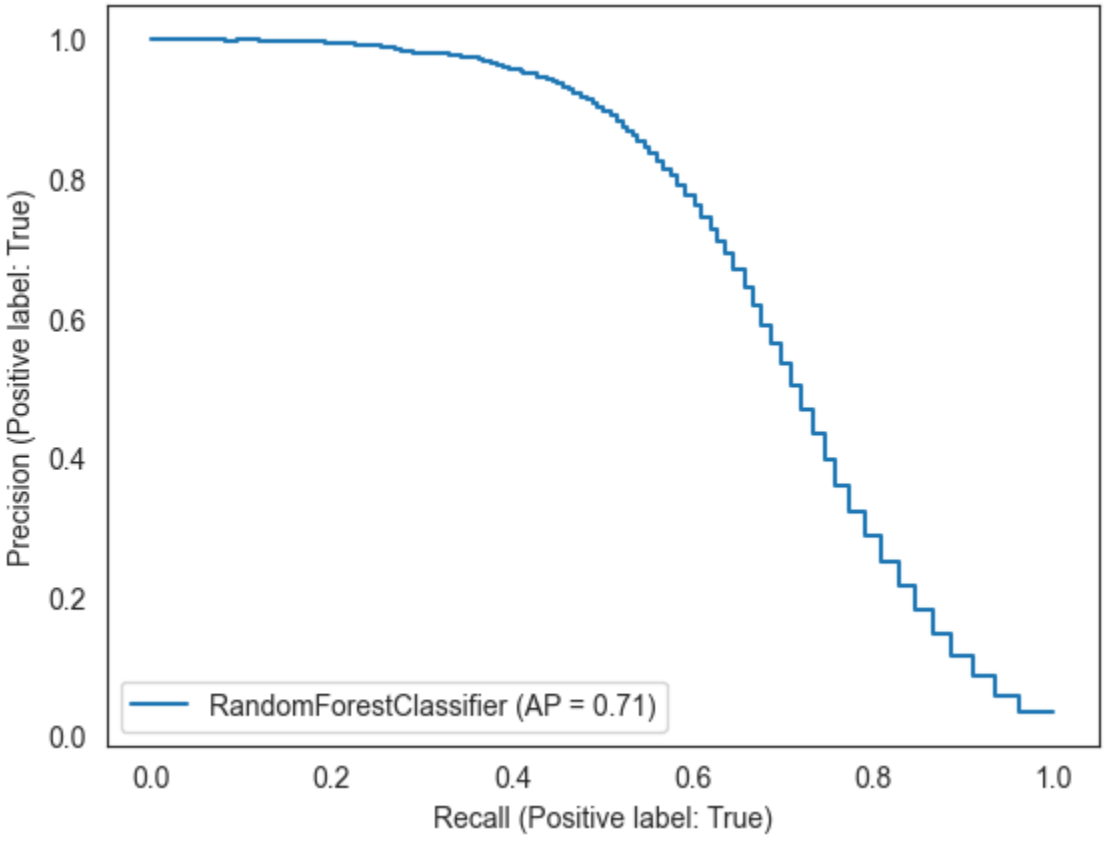
In [123…
```
compute_evaluation_metric(rfc, X_test, y_test, y_pred_rfc, y_prob_pred_rfc)
```

```
Accuracy Score :  0.9792901412266739

AUC Score :  0.9295181538168427

Confusion Matrix :
 [[170811    152]
 [  3517   2682]]

Classification Report :
               precision    recall  f1-score   support

       False       0.98      1.00      0.99    170963
        True       0.95      0.43      0.59      6199

    accuracy                           0.98    177162
   macro avg       0.96      0.72      0.79    177162
weighted avg       0.98      0.98      0.98    177162


Concordance Index :  0.9147900963094 97
```

ROC curve :




PR curve :



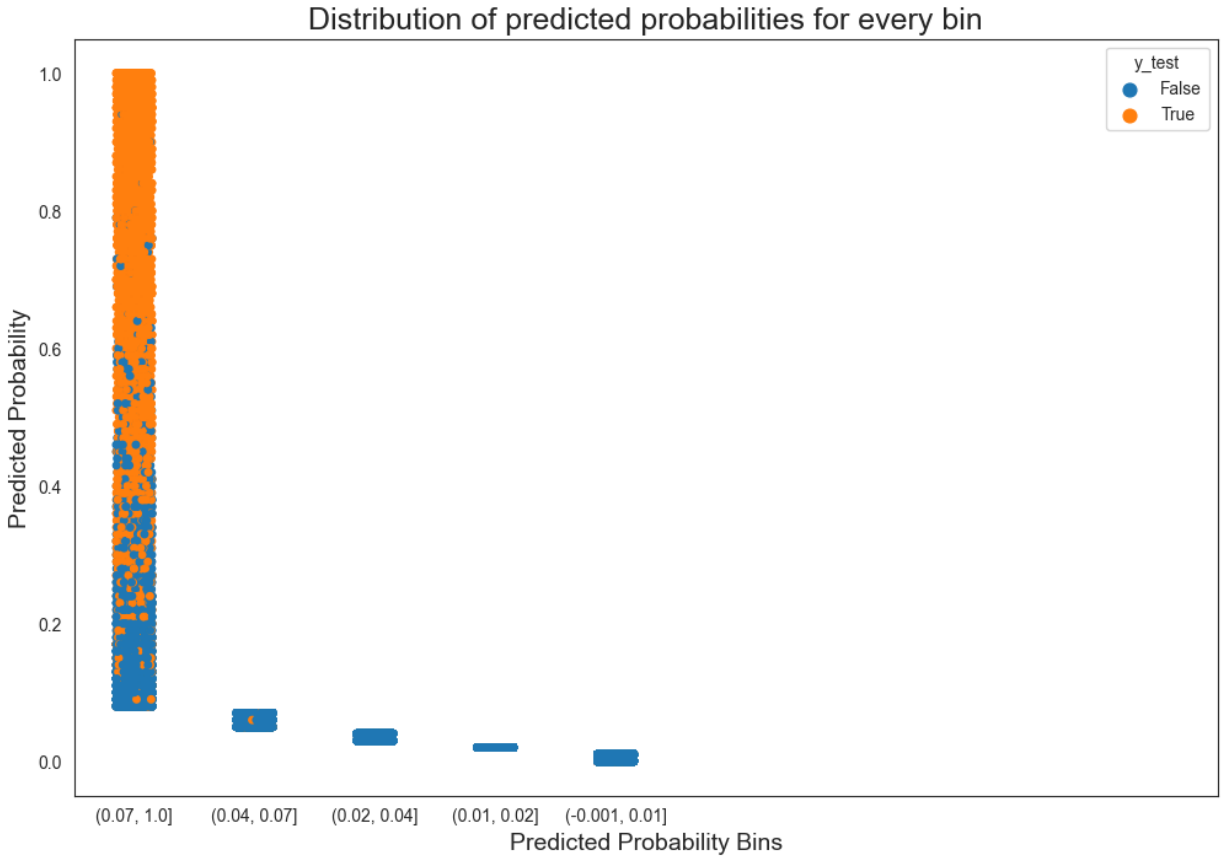

Additional Metrics:
TPR (Recall) : 0.4327
FPR : 0.0009

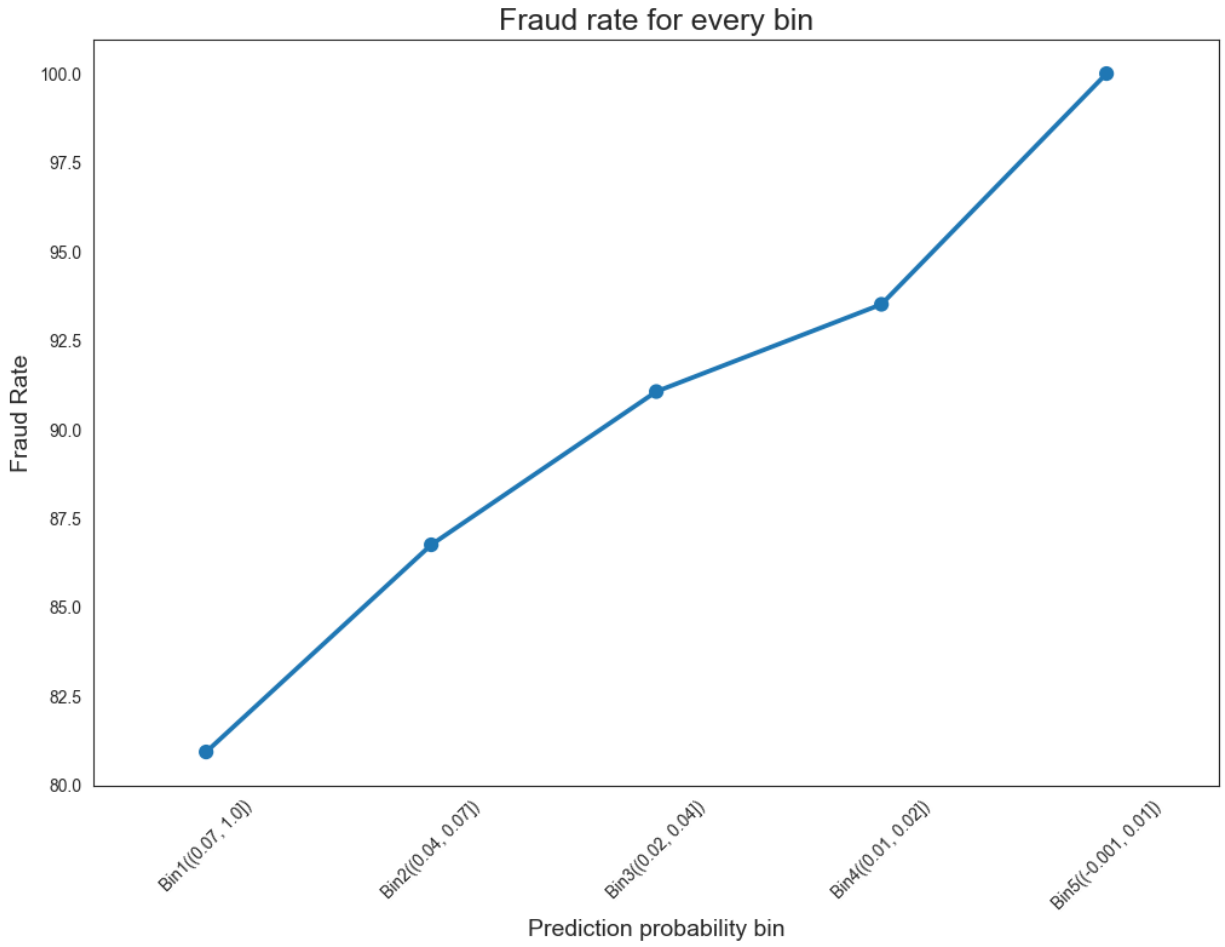
```
TNR (Specificity) : 0.9991
FNR : 0.5673
```

In [124…]
```
concordance(y_test.values, y_prob_pred_rfc)
```

Out[124]:
```
0.914790096309497
```

In [125…]
```
captures(y_test, y_pred_rfc, y_prob_pred_rfc)
```



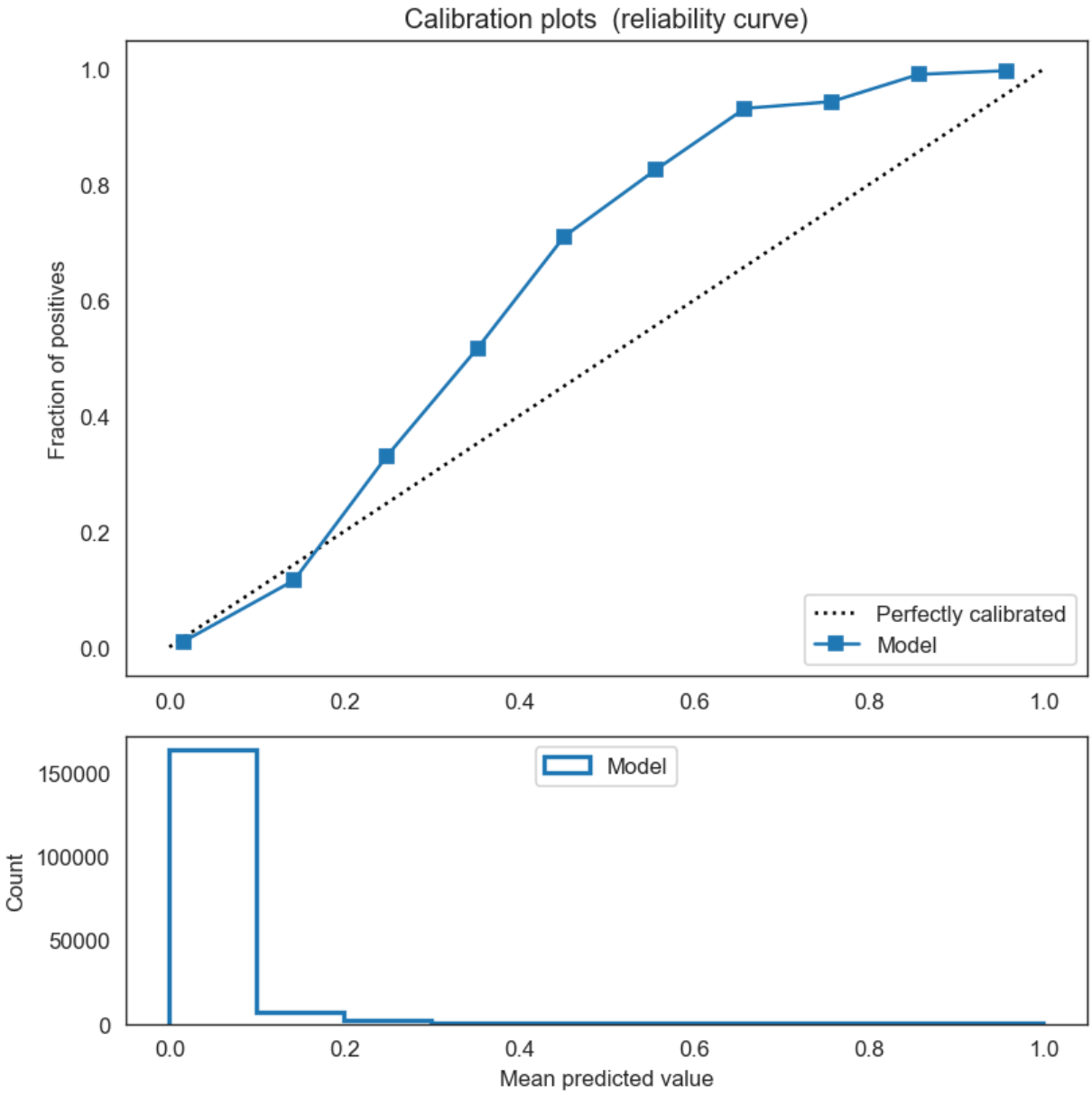



Out[125]:

| | prob_bin | not_fraud | fraud | perc_fraud | perc_not_fraud | cum_perc_fraud | cum_perc_not_fraud |
|---|---|---|---|---|---|---|---|
| **0** | Bin1((0.07, 1.0]) | 12482 | 5017 | 0.809324 | 0.073010 | 80.932408 | 7.300995 |
| **1** | Bin2((0.04, 0.07]) | 11939 | 361 | 0.058235 | 0.069834 | 86.755928 | 14.284377 |
| **2** | Bin3((0.02, 0.04]) | 18984 | 267 | 0.043071 | 0.111042 | 91.063075 | 25.388534 |
| **3** | Bin4((0.01, 0.02]) | 18595 | 152 | 0.024520 | 0.108766 | 93.515083 | 36.265157 |
| **4** | Bin5((-0.001, 0.01]) | 108963 | 402 | 0.064849 | 0.637348 | 100.000000 | 100.000000 |

In [126…

```
draw_calibration_curve(y_test, y_prob_pred_rfc, n_bins=10)
```




# 15. Handling Class Imbalance

## Handle Class Imbalance with Random Oversampler

In [130…
```
from   imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler()
X_train_ros, y_train_ros = ros.fit_resample(X_train, y_train)
y_train_ros.value_counts()
```

Out[130]:
```
False    398914
True     398914
Name: isFraud, dtype: int64
```

In [131…
```
%%time
lgbc_ros = LGBMClassifier(random_state=0)
lgbc_ros.fit(X_train_ros,y_train_ros)
lgbc_ros
```

```
Wall time: 56.9 s
```

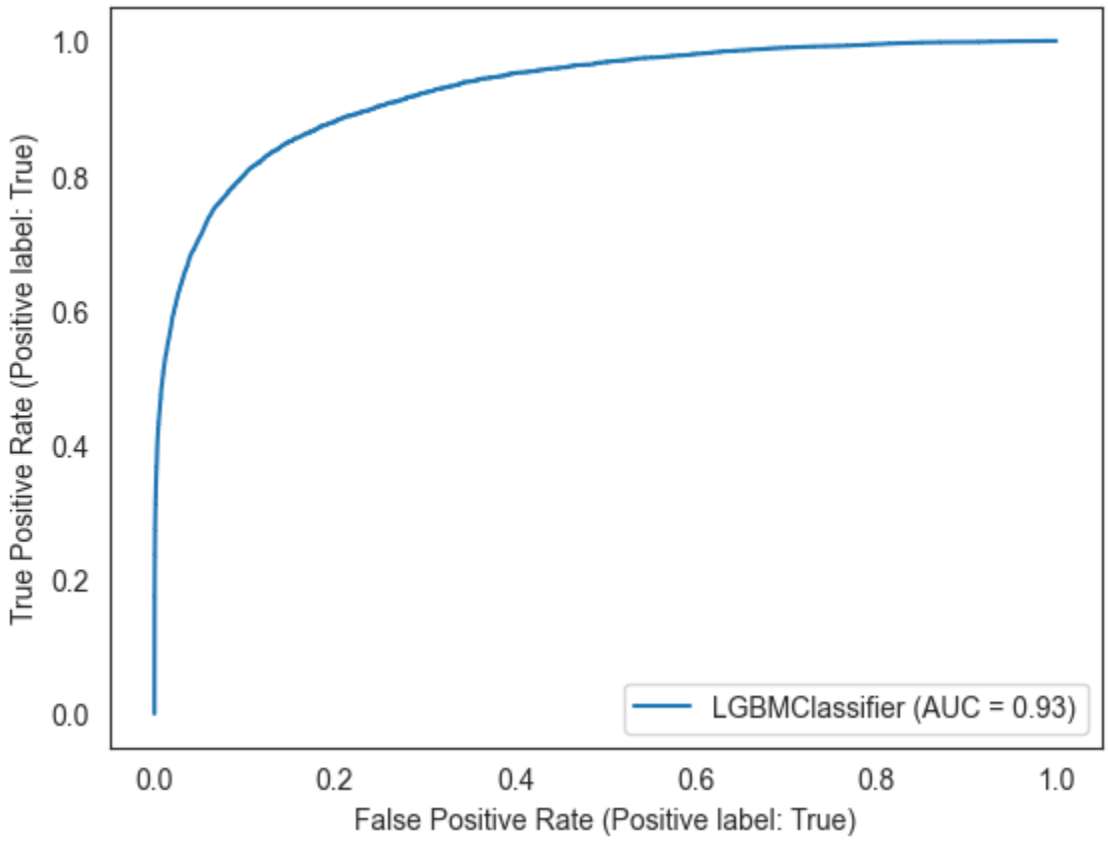
Out[131]:
```
LGBMClassifier(random_state=0)
```

In [133…
```
y_pred_lgbcros = lgbc_ros.predict(X_test)
y_prob_pred_lgbcros = lgbc_ros.predict_proba(X_test)[:, 1]

print("Y predicted : ",y_pred_lgbcros)
print("Y probability predicted : ",y_prob_pred_lgbcros[:5])
```

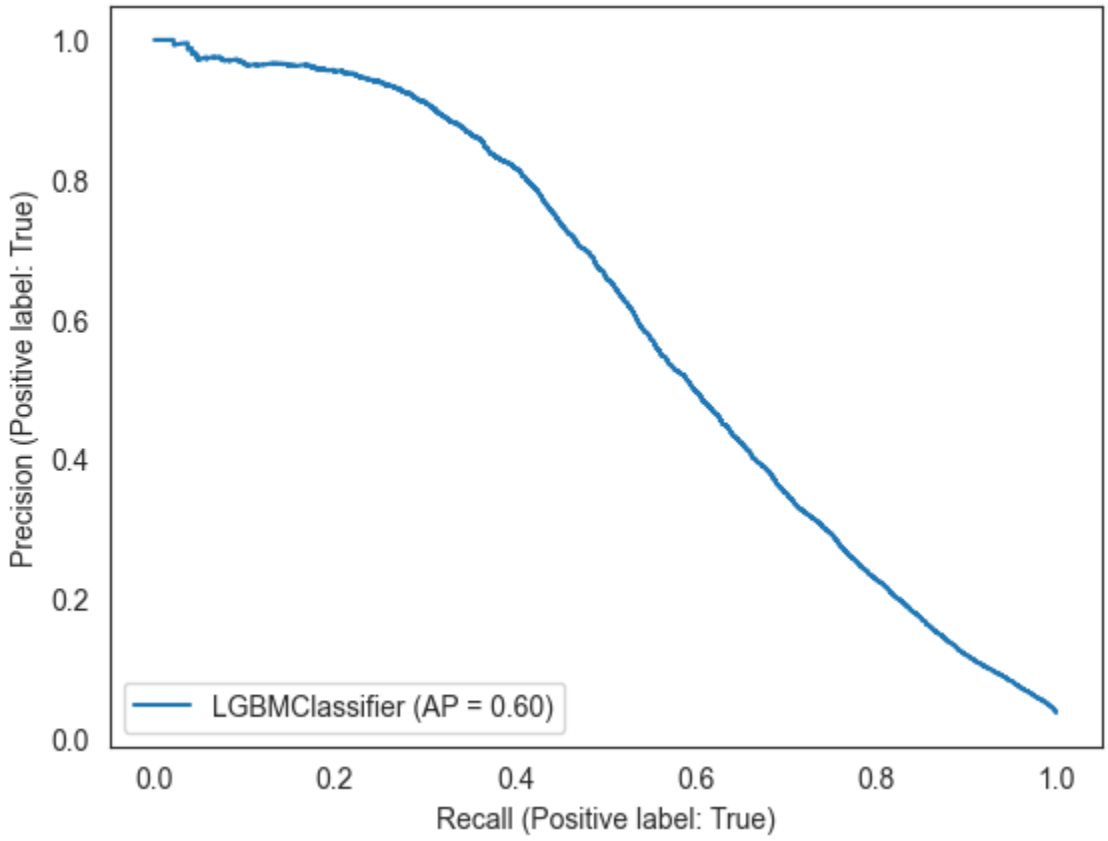
```
Y predicted :  [False False False ... False False False]
Y probability predicted :  [0.23093612 0.12247041 0.13222139 0.56481875 0.1046
0942]
```

In [135…
```
compute_evaluation_metric(lgbc_ros, X_test, y_test, y_pred_lgbcros, y_prob_pre
```

```
Accuracy Score :  0.886696921461713

AUC Score :  0.9271778128567145

Confusion Matrix :
 [[152037  18926]
 [  1147   5052]]

Classification Report :
               precision    recall  f1-score   support

       False       0.99      0.89      0.94    170963
        True       0.21      0.81      0.33      6199

    accuracy                           0.89    177162
   macro avg       0.60      0.85      0.64    177162
weighted avg       0.97      0.89      0.92    177162


Concordance Index :  0.9271778029491814

ROC curve :
```



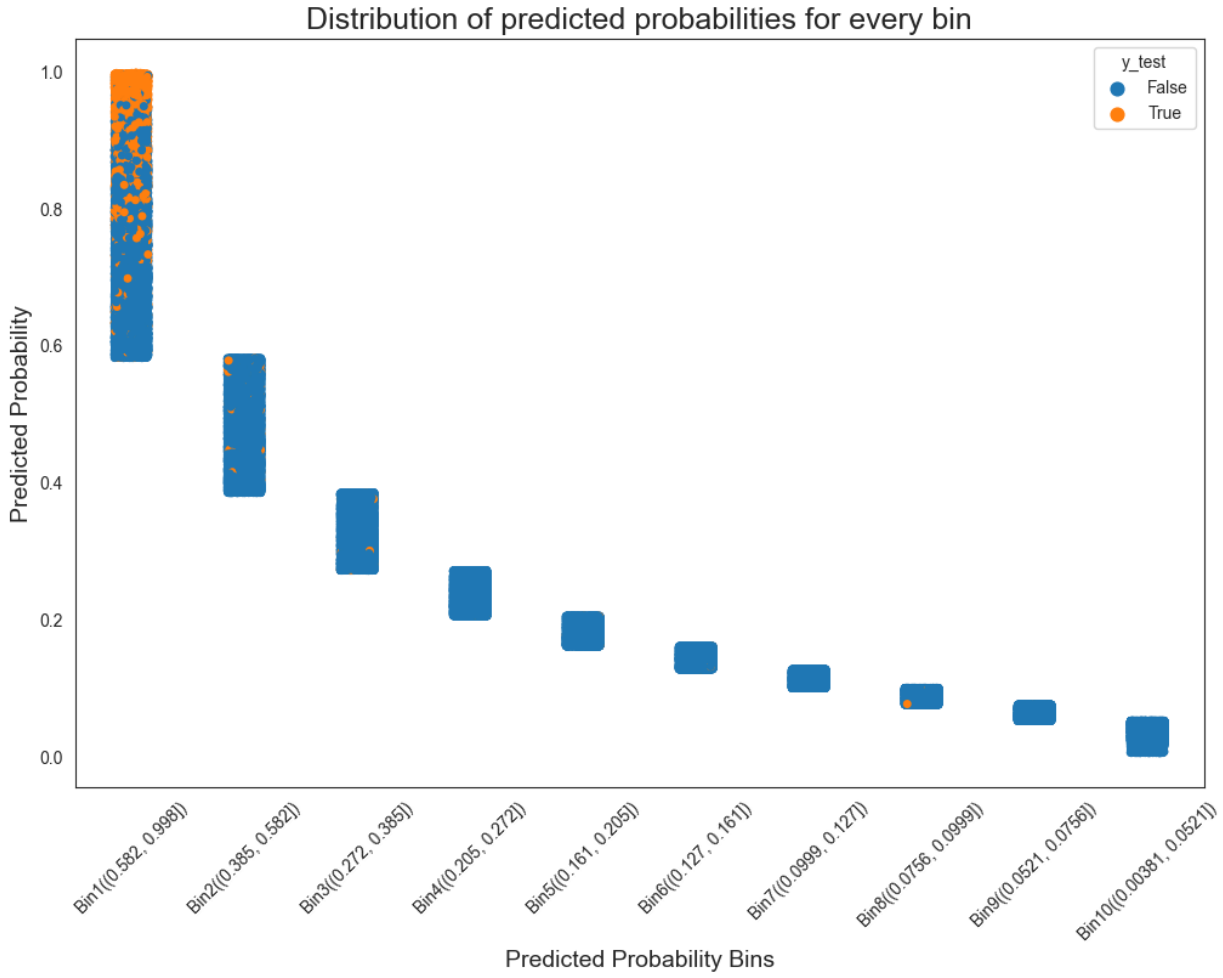

PR curve :

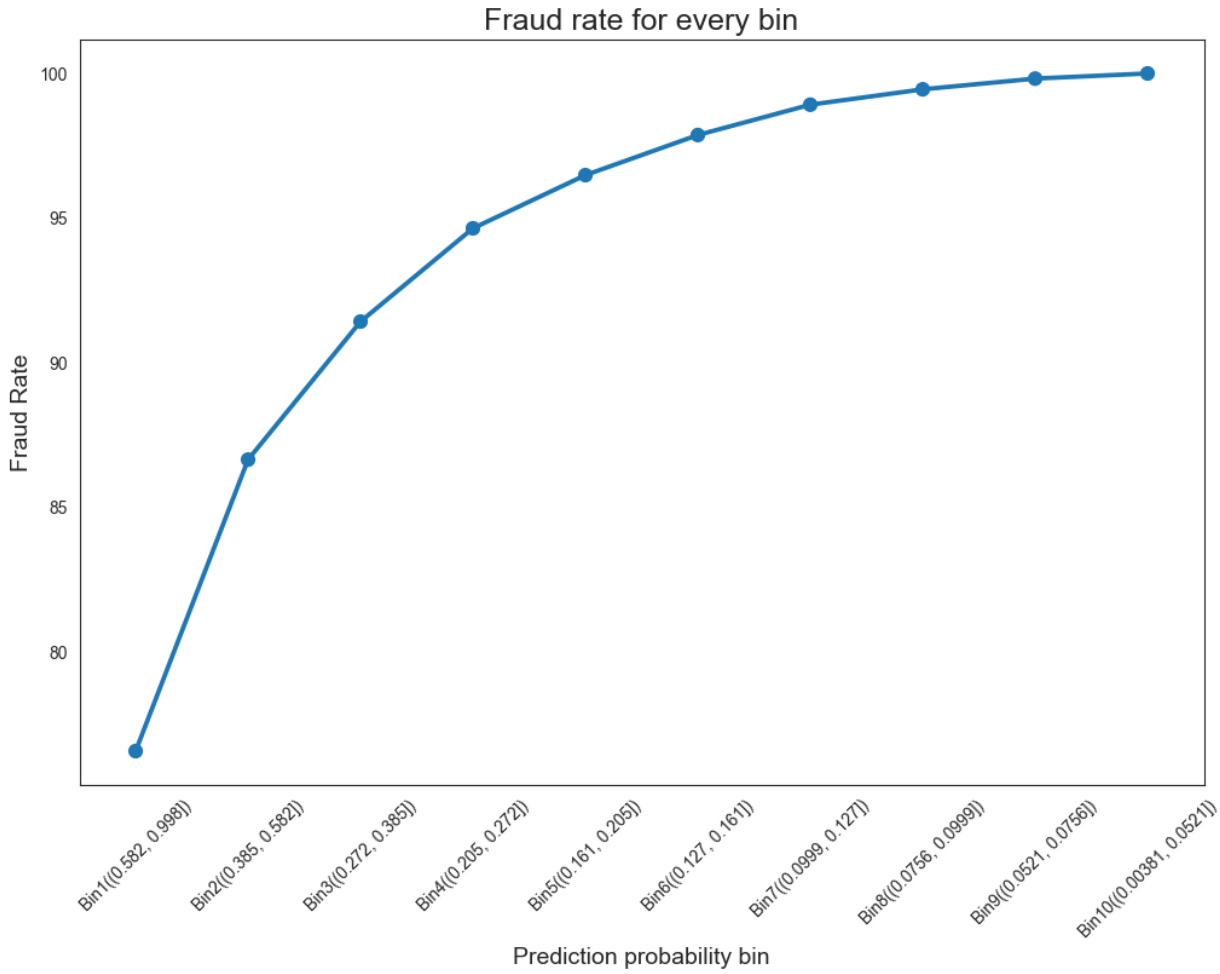



```
Additional Metrics:
TPR (Recall) : 0.8150
FPR : 0.1107
TNR (Specificity) : 0.8893
FNR : 0.1850
```

In [136…

```
captures(y_test, y_pred_lgbcros, y_prob_pred_lgbcros)
```



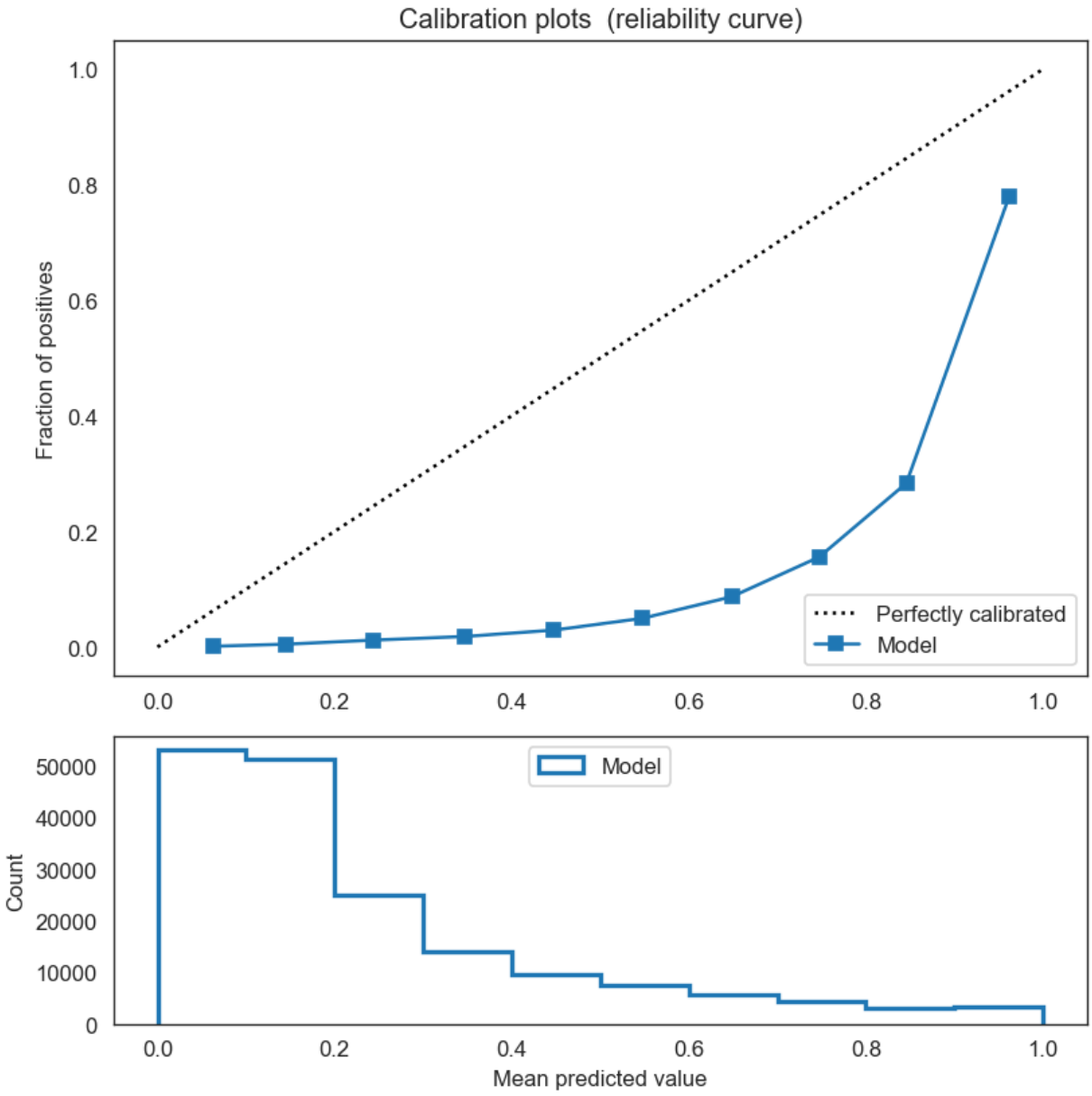



Out[136]:

| | prob_bin | not_fraud | fraud | perc_fraud | perc_not_fraud | cum_perc_fraud | cum_perc_not_fra |
|---|---|---|---|---|---|---|---|
| **0** | Bin1((0.582, 0.998]) | 12971 | 4746 | 0.765607 | 0.075870 | 76.560736 | 7.587 |
| **1** | Bin2((0.385, 0.582]) | 17091 | 625 | 0.100823 | 0.099969 | 86.643007 | 17.583 |
| **2** | Bin3((0.272, 0.385]) | 17420 | 296 | 0.047750 | 0.101893 | 91.417971 | 27.773 |
| **3** | Bin4((0.205, 0.272]) | 17516 | 200 | 0.032263 | 0.102455 | 94.644297 | 38.018 |
| **4** | Bin5((0.161, 0.205]) | 17602 | 114 | 0.018390 | 0.102958 | 96.483304 | 48.314 |
| **5** | Bin6((0.127, 0.161]) | 17630 | 86 | 0.013873 | 0.103122 | 97.870624 | 58.626 |
| **6** | Bin7((0.0999, 0.127]) | 17651 | 65 | 0.010486 | 0.103245 | 98.919181 | 68.951 |
| **7** | Bin8((0.0756, 0.0999]) | 17683 | 33 | 0.005323 | 0.103432 | 99.451524 | 79.294 |
| **8** | Bin9((0.0521, 0.0756]) | 17693 | 23 | 0.003710 | 0.103490 | 99.822552 | 89.643 |
| **9** | Bin10((0.00381, 0.0521]) | 17706 | 11 | 0.001774 | 0.103566 | 100.000000 | 100.000 |

In [137…

```
draw_calibration_curve(y_test, y_prob_pred_lgbcros, n_bins=10)
```




- After balancing the class, accuracy score is 0.88 and AUC score is 92.5%
- Accuracy has decreased as compared to the previos model, but AUC has improved
- Additionally the recall has improved significantly at the cost of precision.

# 16. Cost Sensitive Learning with Class weights

In [139…
```
%%time
lgbc_bal = LGBMClassifier(random_state=0, class_weight='balanced')
lgbc_bal.fit(X_train, y_train)
lgbc_bal
```

```
Wall time: 32 s
```

Out[139]:
```
LGBMClassifier(class_weight='balanced', random_state=0)
```

In [140…
```
y_pred_lgbcbal = lgbc_bal.predict(X_test)
y_prob_pred_lgbcbal = lgbc_bal.predict_proba(X_test)[:, 1]
```

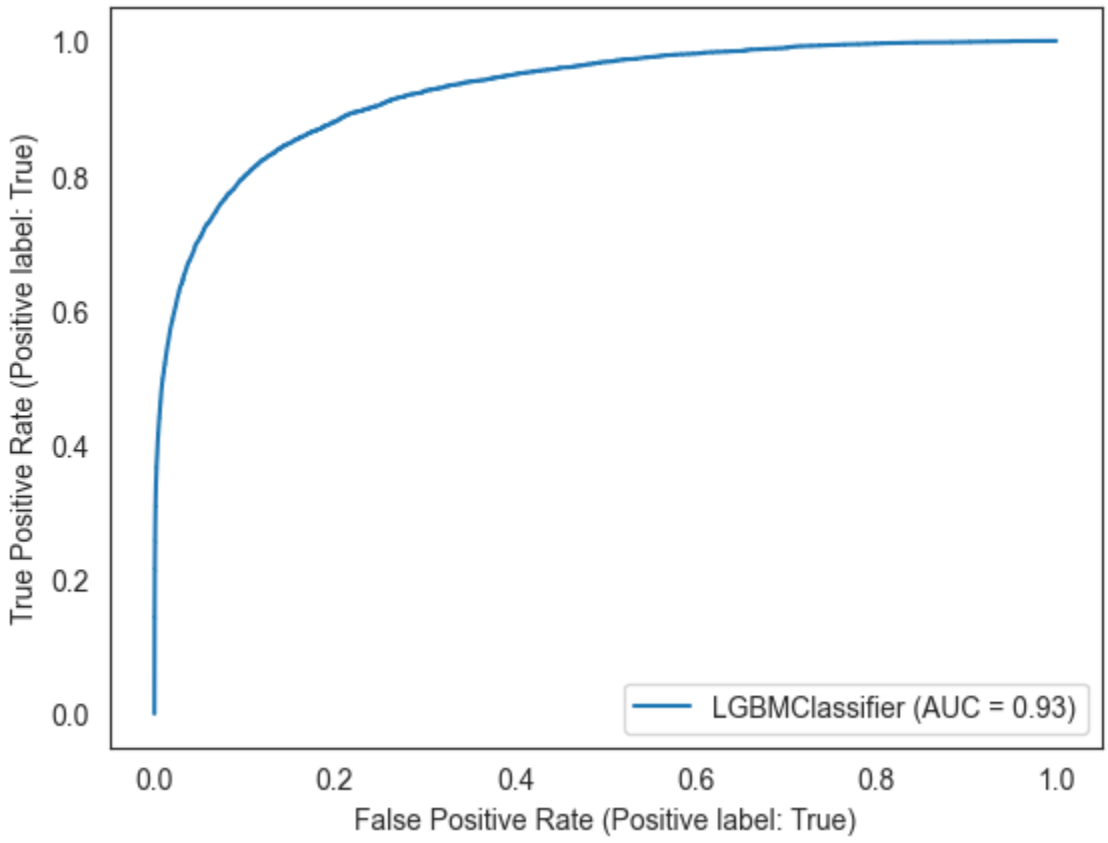
```
print("Y predicted : ",y_pred_lgbcbal)
print("Y probability predicted : ",y_prob_pred_lgbcbal[:5])
```

```
Y predicted :  [False False False ... False False False]
Y probability predicted :  [0.22700362 0.15390235 0.13087413 0.52349433 0.1403
2882]
```

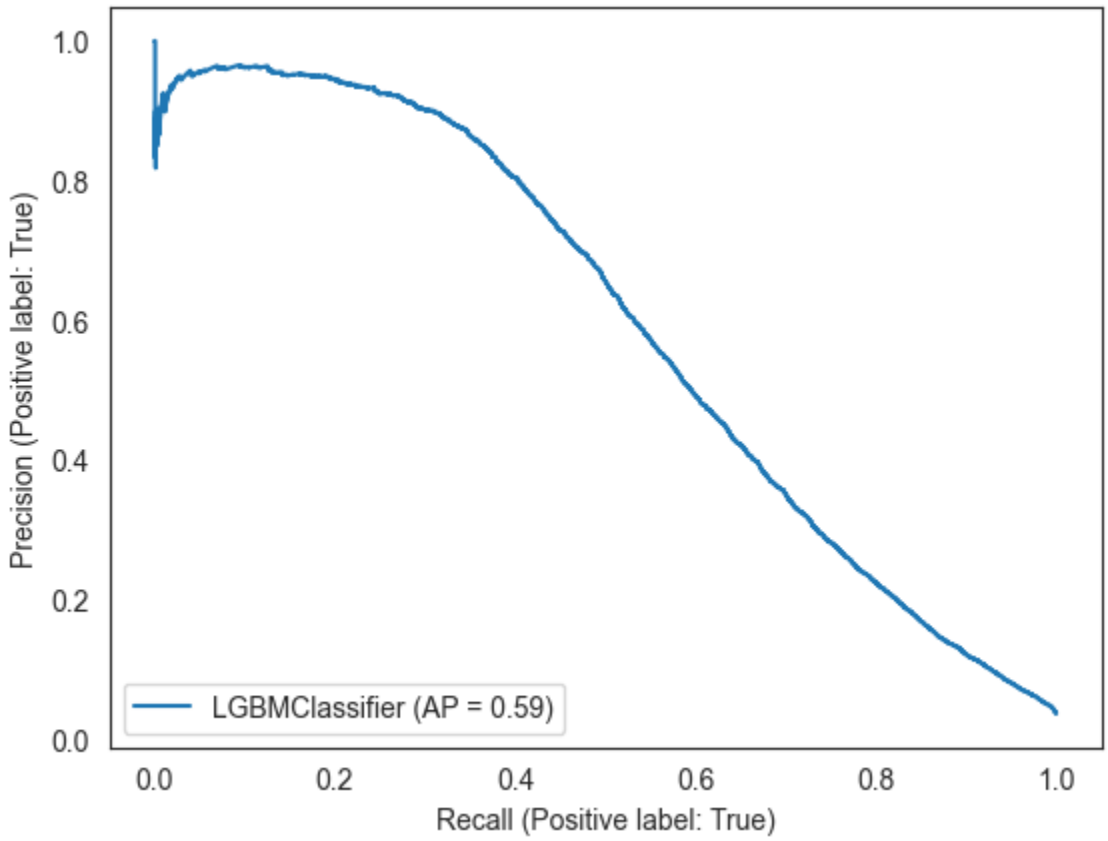
In [141…

```
compute_evaluation_metric(lgbc_bal, X_test, y_test, y_pred_lgbcbal, y_prob_pre
```

```
Accuracy Score :  0.8862284237025999

AUC Score :  0.927124006459723

Confusion Matrix :
 [[151969  18994]
 [  1162   5037]]

Classification Report :
               precision    recall  f1-score   support

       False       0.99      0.89      0.94    170963
        True       0.21      0.81      0.33      6199

    accuracy                           0.89    177162
   macro avg       0.60      0.85      0.64    177162
weighted avg       0.97      0.89      0.92    177162


Concordance Index :  0.9271239946650406

ROC curve :
```



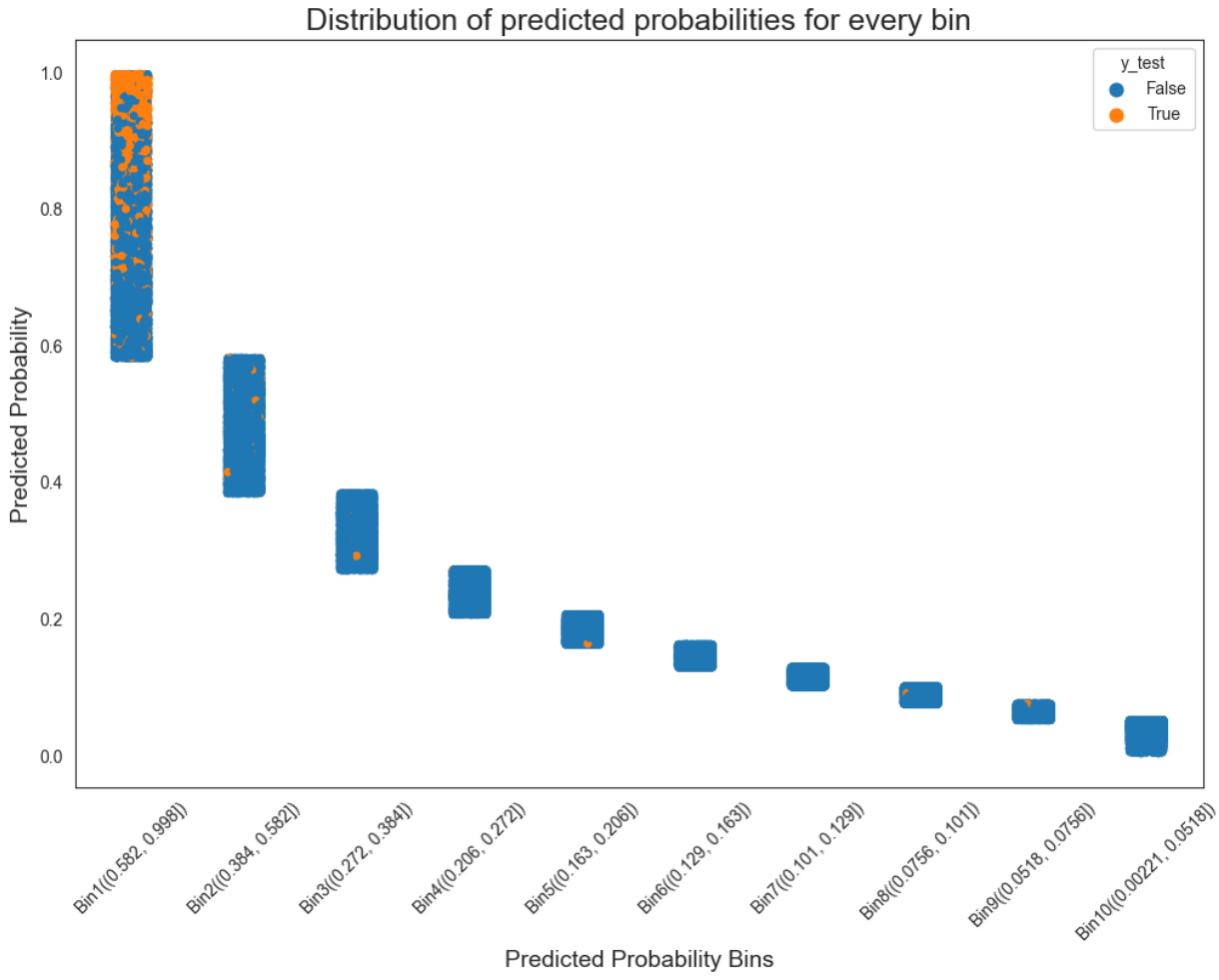

```
PR curve :
```

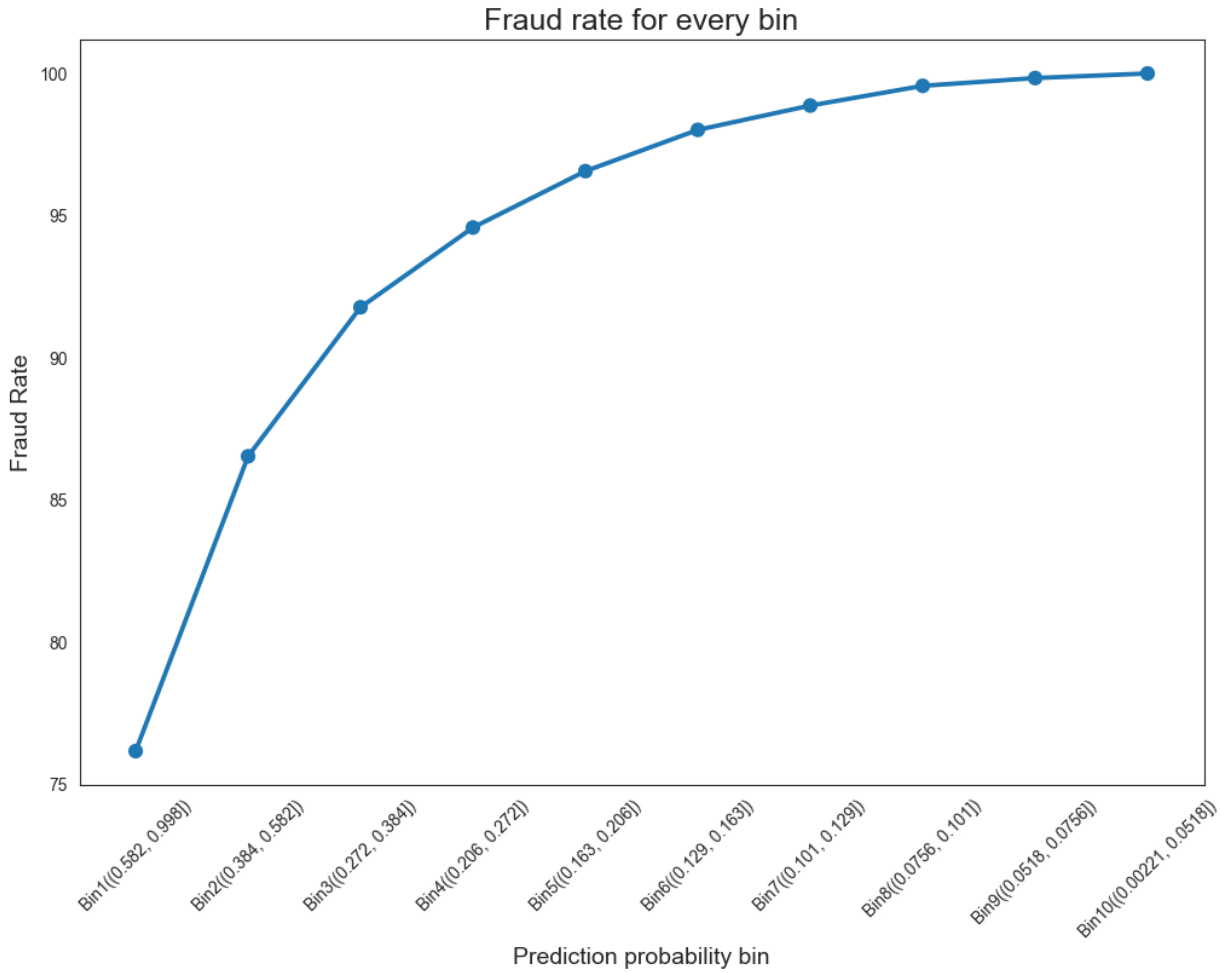



```
Additional Metrics:
TPR (Recall) : 0.8126
FPR : 0.1111
TNR (Specificity) : 0.8889
FNR : 0.1874
```

In [142…

```
captures(y_test, y_pred_lgbcbal, y_prob_pred_lgbcbal)
```



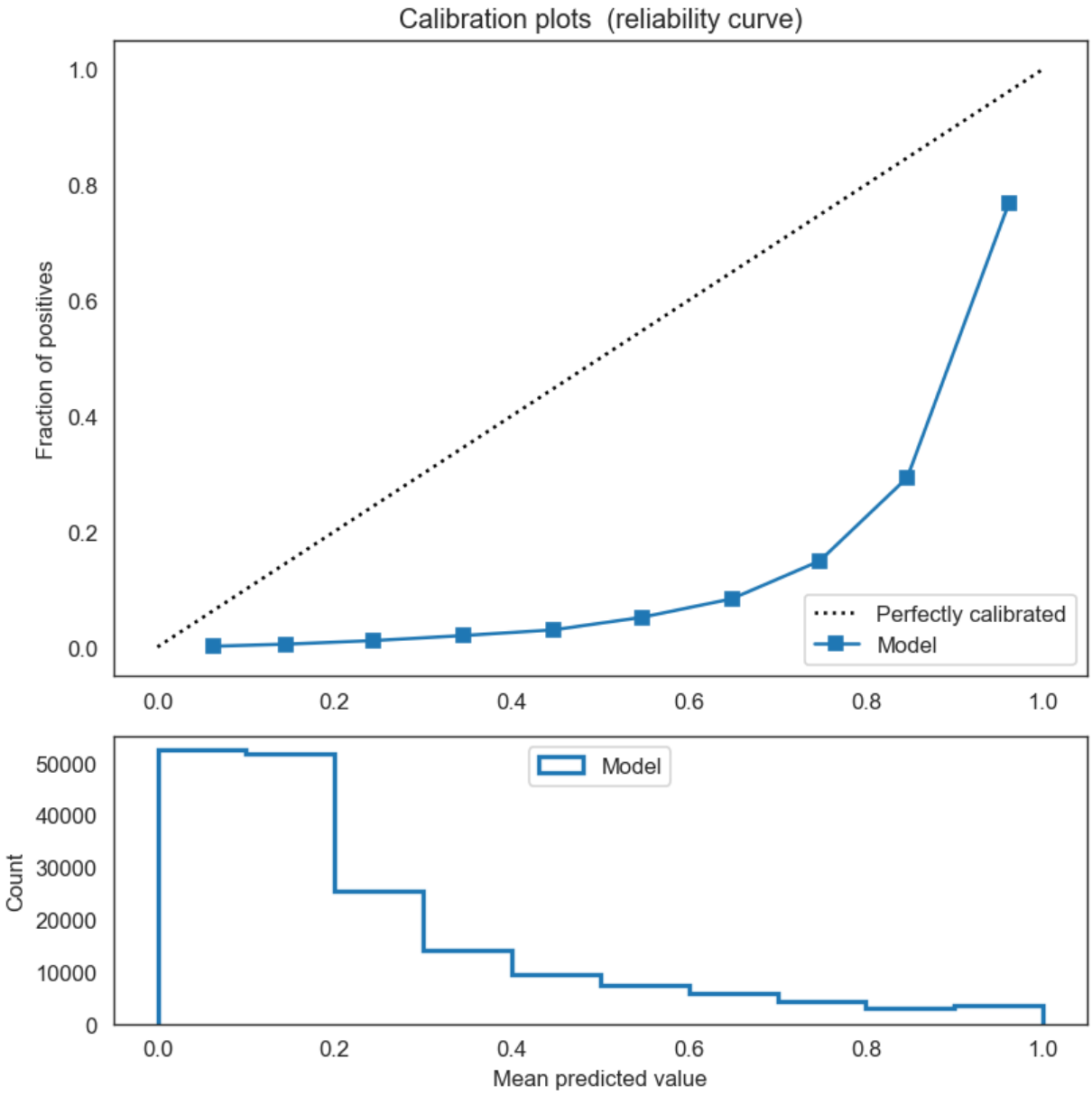



Out[142]:

| | prob_bin | not_fraud | fraud | perc_fraud | perc_not_fraud | cum_perc_fraud | cum_perc_not_fra |
|---|---|---|---|---|---|---|---|
| **0** | Bin1((0.582, 0.998]) | 12996 | 4721 | 0.761574 | 0.076016 | 76.157445 | 7.601 |
| **1** | Bin2((0.384, 0.582]) | 17073 | 643 | 0.103726 | 0.099864 | 86.530085 | 17.588 |
| **2** | Bin3((0.272, 0.384]) | 17391 | 325 | 0.052428 | 0.101724 | 91.772867 | 27.760 |
| **3** | Bin4((0.206, 0.272]) | 17542 | 174 | 0.028069 | 0.102607 | 94.579771 | 38.021 |
| **4** | Bin5((0.163, 0.206]) | 17593 | 123 | 0.019842 | 0.102905 | 96.563962 | 48.311 |
| **5** | Bin6((0.129, 0.163]) | 17626 | 90 | 0.014518 | 0.103098 | 98.015809 | 58.621 |
| **6** | Bin7((0.101, 0.129]) | 17663 | 53 | 0.008550 | 0.103315 | 98.870786 | 68.952 |
| **7** | Bin8((0.0756, 0.101]) | 17673 | 43 | 0.006937 | 0.103373 | 99.564446 | 79.290 |
| **8** | Bin9((0.0518, 0.0756]) | 17699 | 17 | 0.002742 | 0.103525 | 99.838684 | 89.642 |
| **9** | Bin10((0.00221, 0.0518]) | 17707 | 10 | 0.001613 | 0.103572 | 100.000000 | 100.000 |

In [143…

```
draw_calibration_curve(y_test, y_prob_pred_lgbcbal, n_bins=10)
```




# 17. Model Calibration

In [144…
```
from sklearn.calibration import CalibratedClassifierCV
```

In [145…
```
lgbc_bal = LGBMClassifier(random_state=0)
calibrated_clf = CalibratedClassifierCV(base_estimator=lgbc_bal, cv=3, method=
calibrated_clf.fit(X_train, y_train)
```

Out[145]:
```
CalibratedClassifierCV(base_estimator=LGBMClassifier(random_state=0), cv=3)
```

In [146…
```
y_pred_calib = calibrated_clf.predict(X_test)
y_prob_pred_calib = calibrated_clf.predict_proba(X_test)[:, 1]
```

In [147…
```
len(calibrated_clf.calibrated_classifiers_)
```

Out[147]:
```
3
```

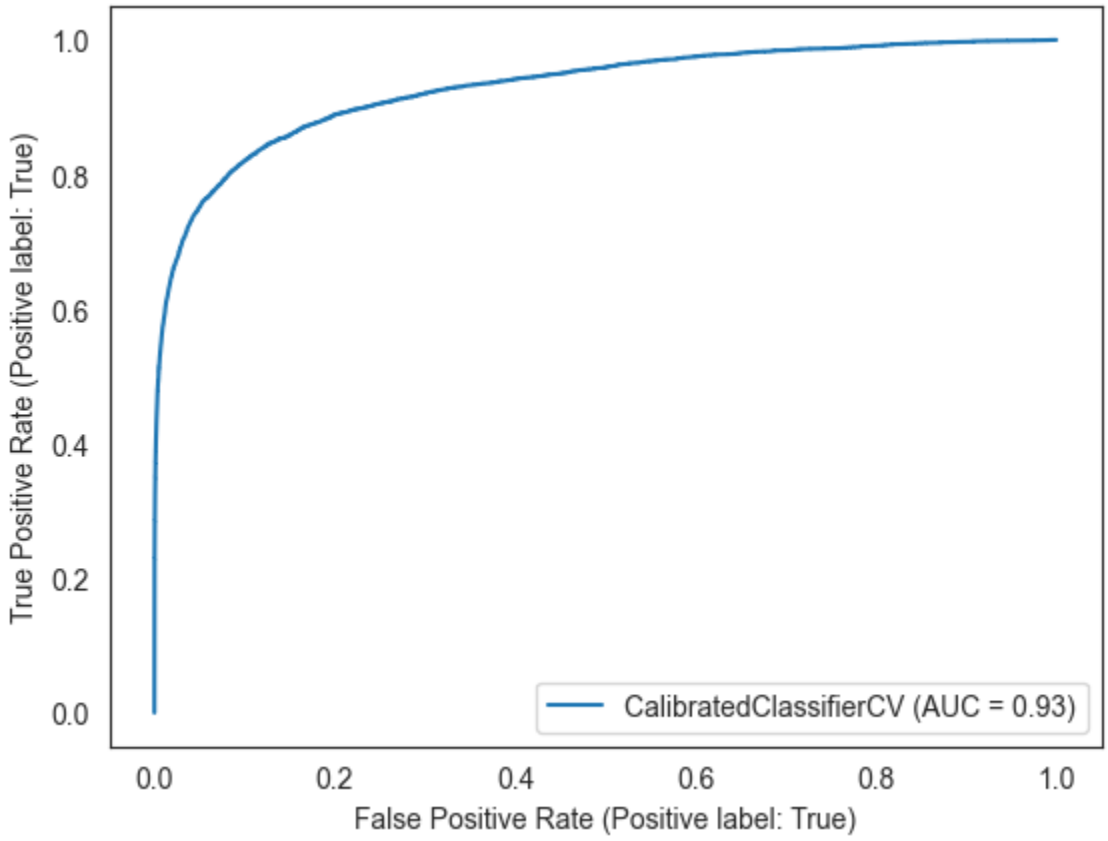
In [148…
```
print("Y predicted : ", y_pred_calib)
print("Y probability predicted : ", y_prob_pred_calib[:5])
```

```
Y predicted :  [False False False ... False False False]
Y probability predicted :  [0.01510527 0.013539   0.01360709 0.0235388  0.0134
2325]
```

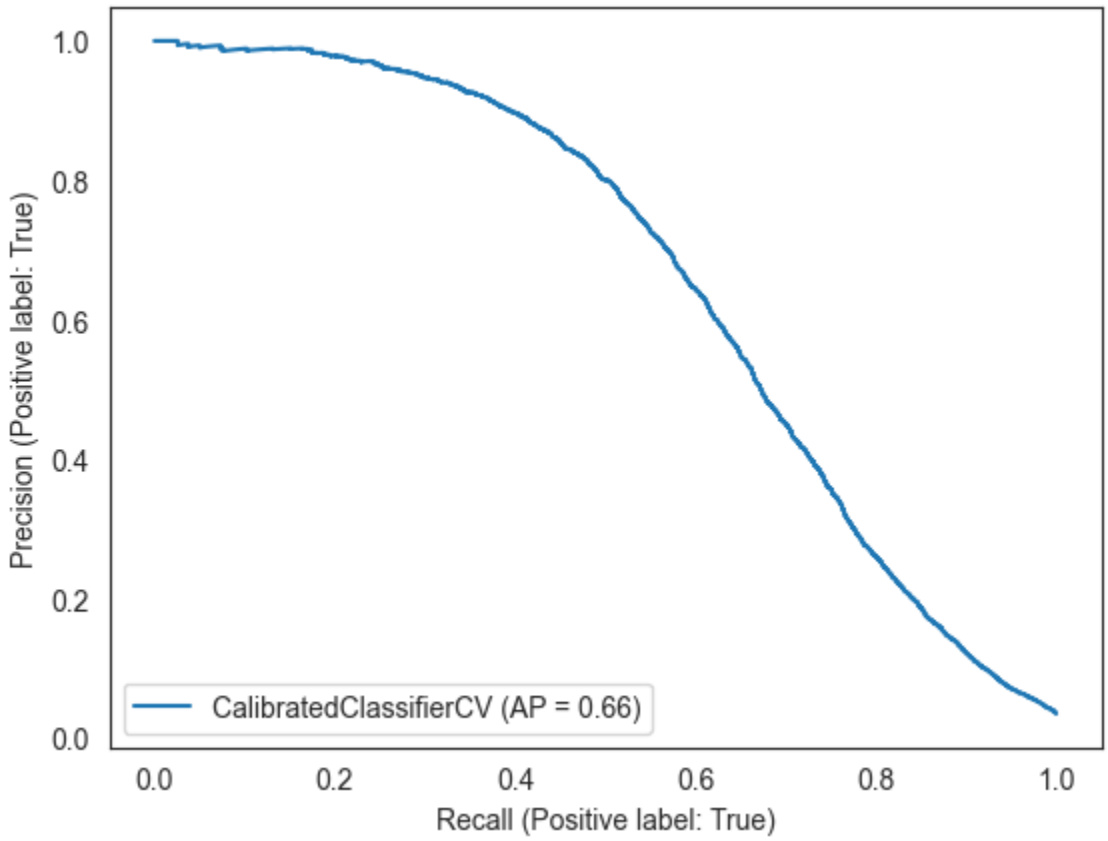
In [150…

```
compute_evaluation_metric(calibrated_clf, X_test, y_test, y_pred_calib, y_prob
```

```
Accuracy Score :  0.9781160745532338

AUC Score :  0.9292560410642979

Confusion Matrix :
 [[170502    461]
 [  3416   2783]]

Classification Report :
               precision    recall  f1-score   support

       False       0.98      1.00      0.99    170963
        True       0.86      0.45      0.59      6199

    accuracy                           0.98    177162
   macro avg       0.92      0.72      0.79    177162
weighted avg       0.98      0.98      0.97    177162


Concordance Index :  0.9292560391771487

ROC curve :
```



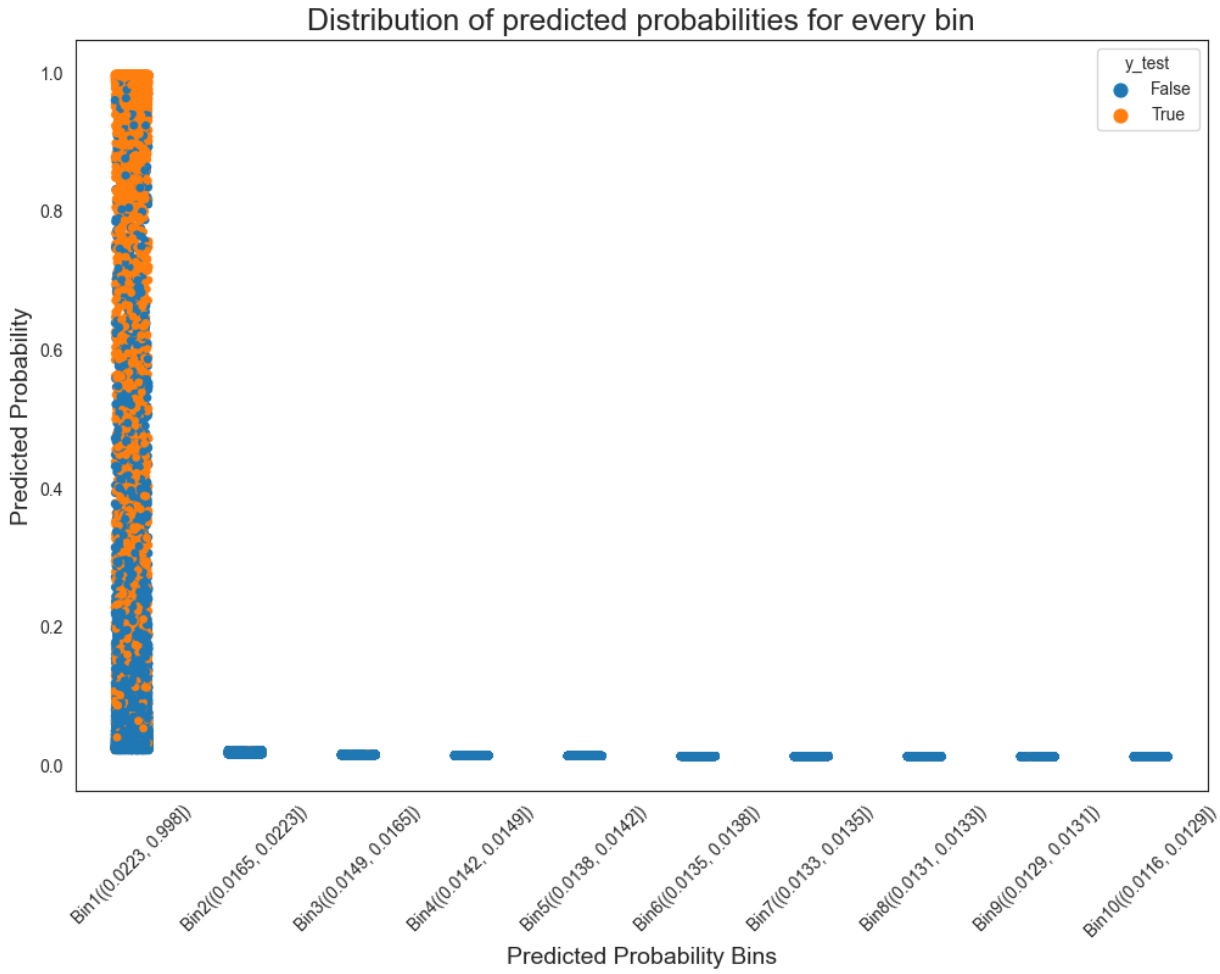

```
PR curve :
```

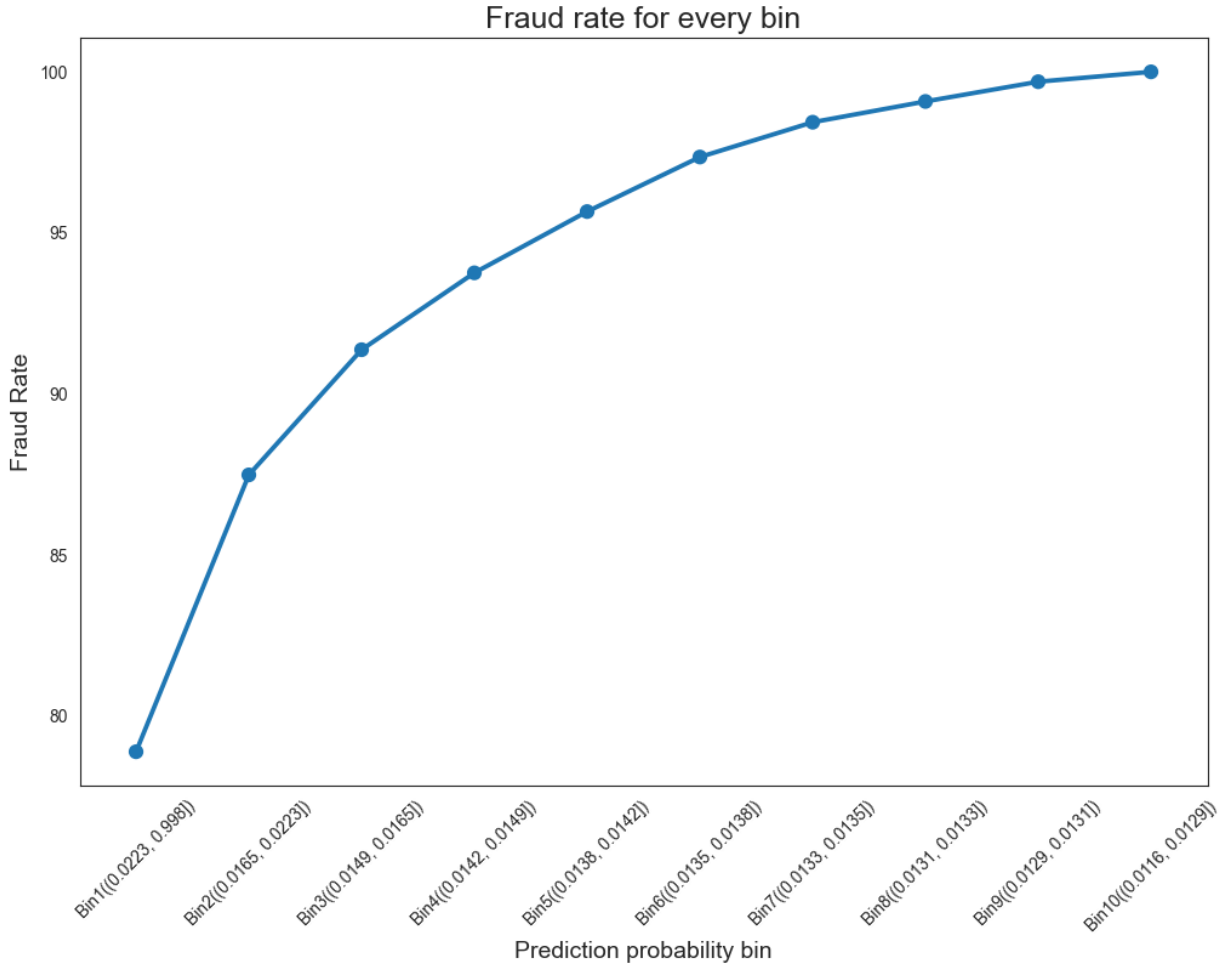



```
Additional Metrics:
TPR (Recall) : 0.4489
FPR : 0.0027
TNR (Specificity) : 0.9973
FNR : 0.5511
```

In [151…

```
captures(y_test, y_pred_calib, y_prob_pred_calib)
```



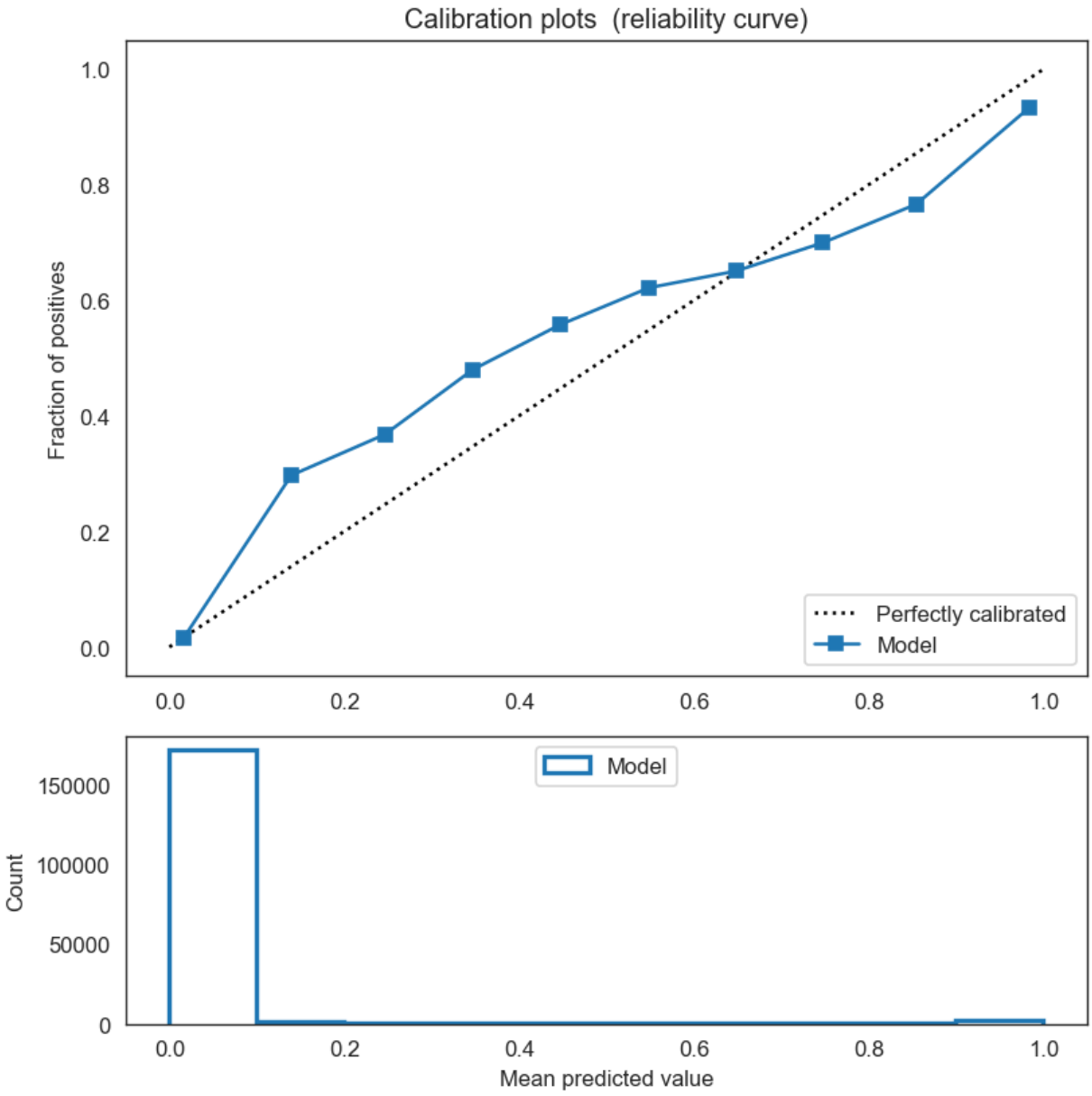



Out[151]:

| | prob_bin | not_fraud | fraud | perc_fraud | perc_not_fraud | cum_perc_fraud | cum_perc_not_frau |
|---|---|---|---|---|---|---|---|
| **0** | Bin1((0.0223, 0.998]) | 12827 | 4890 | 0.788837 | 0.075028 | 78.883691 | 7.50279 |
| **1** | Bin2((0.0165, 0.0223]) | 17183 | 533 | 0.085982 | 0.100507 | 87.481852 | 17.55350 |
| **2** | Bin3((0.0149, 0.0165]) | 17475 | 241 | 0.038877 | 0.102215 | 91.369576 | 27.77501 |
| **3** | Bin4((0.0142, 0.0149]) | 17568 | 148 | 0.023875 | 0.102759 | 93.757058 | 38.05092 |
| **4** | Bin5((0.0138, 0.0142]) | 17598 | 118 | 0.019035 | 0.102935 | 95.660590 | 48.34437 |
| **5** | Bin6((0.0135, 0.0138]) | 17611 | 105 | 0.016938 | 0.103011 | 97.354412 | 58.64543 |
| **6** | Bin7((0.0133, 0.0135]) | 17649 | 67 | 0.010808 | 0.103233 | 98.435231 | 68.96872 |
| **7** | Bin8((0.0131, 0.0133]) | 17676 | 40 | 0.006453 | 0.103391 | 99.080497 | 79.30780 |
| **8** | Bin9((0.0129, 0.0131]) | 17678 | 38 | 0.006130 | 0.103402 | 99.693499 | 89.64805 |
| **9** | Bin10((0.0116, 0.0129]) | 17698 | 19 | 0.003065 | 0.103519 | 100.000000 | 100.00000 |

In [152…

```
draw_calibration_curve(y_test, y_prob_pred_calib, n_bins=10)
```




# 18. Model Tuning

In [153…

```
%%time

lgbmclassifier = LGBMClassifier(
    class_weight=None,
    learning_rate=0.1,
    max_depth=8,
    n_estimators=100,
    num_leaves=256,
    reg_alpha=0.5,
    random_state=0
)

lgbmclassifier.fit(X_train, y_train)
```

```
Wall time: 34.7 s
```

Out[153]:

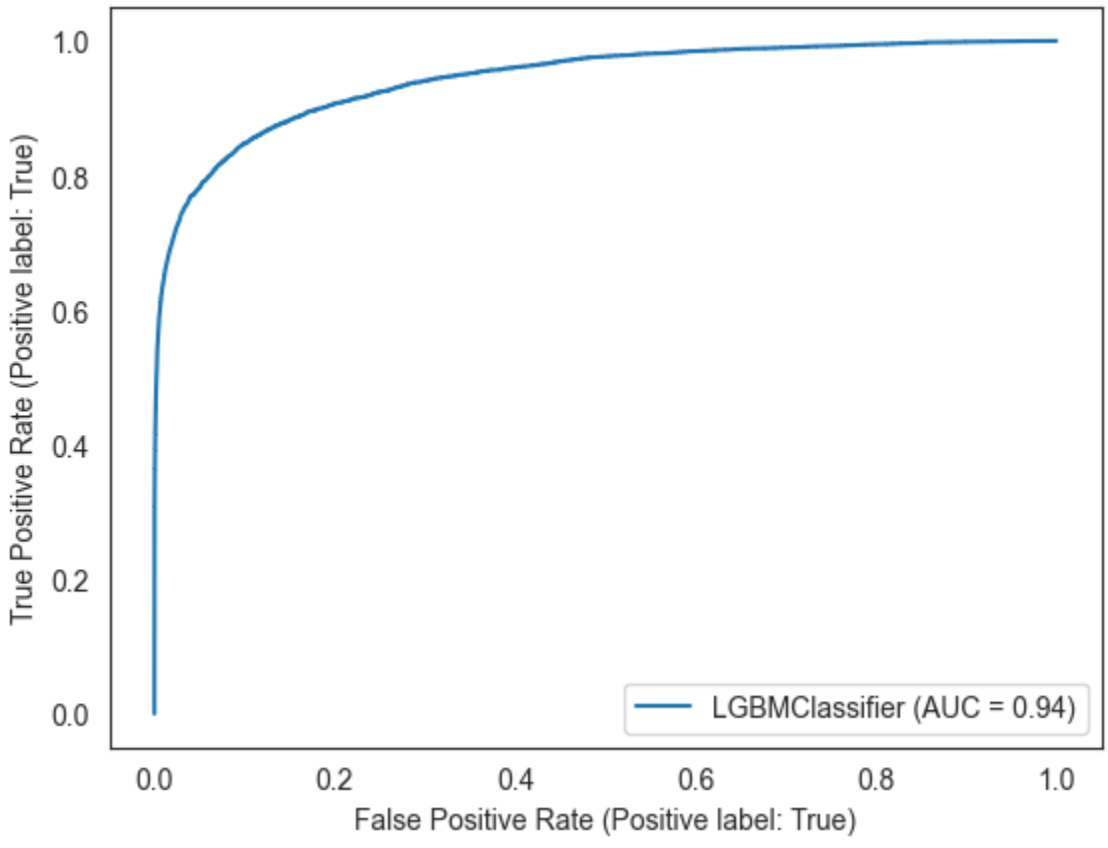
```
LGBMClassifier(max_depth=8, num_leaves=256, random_state=0, reg_alpha=0.5)
```

In [154…
```
y_g_pred = lgbmclassifier.predict(X_test)
y_prob_g_pred = lgbmclassifier.predict_proba(X_test)[:, 1]
print("Y predicted : ",y_g_pred)
print("Y probability predicted : ",y_prob_g_pred[:5])
```

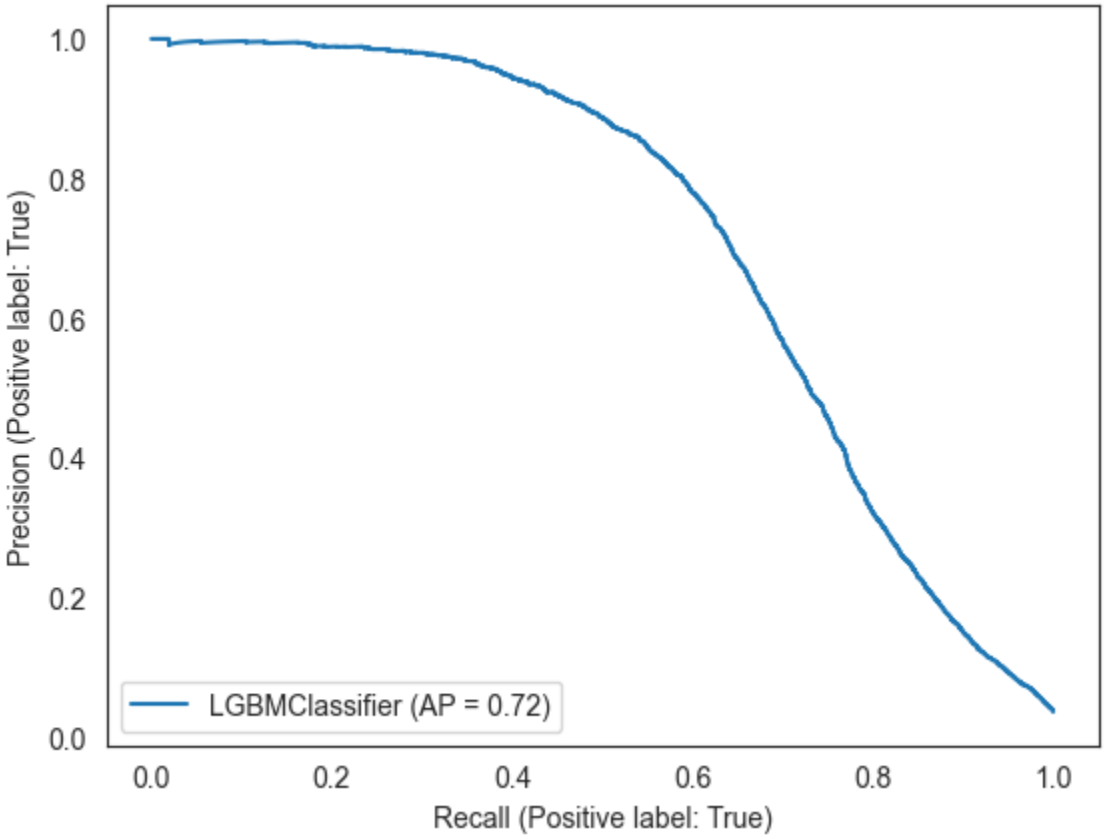
```
Y predicted :  [False False False ... False False False]
Y probability predicted :  [0.00549227 0.00615294 0.007173   0.09576845 0.0073
4595]
```

In [155…
```
compute_evaluation_metric(lgbmclassifier, X_test, y_test, y_g_pred, y_prob_g_p
```

```
Accuracy Score :  0.9796909043700116

AUC Score :  0.9439534102237099

Confusion Matrix :
 [[170676    287]
 [  3311   2888]]

Classification Report :
               precision    recall  f1-score   support

       False       0.98      1.00      0.99    170963
        True       0.91      0.47      0.62      6199

    accuracy                           0.98    177162
   macro avg       0.95      0.73      0.80    177162
weighted avg       0.98      0.98      0.98    177162


Concordance Index :  0.9439534040904752

ROC curve :
```



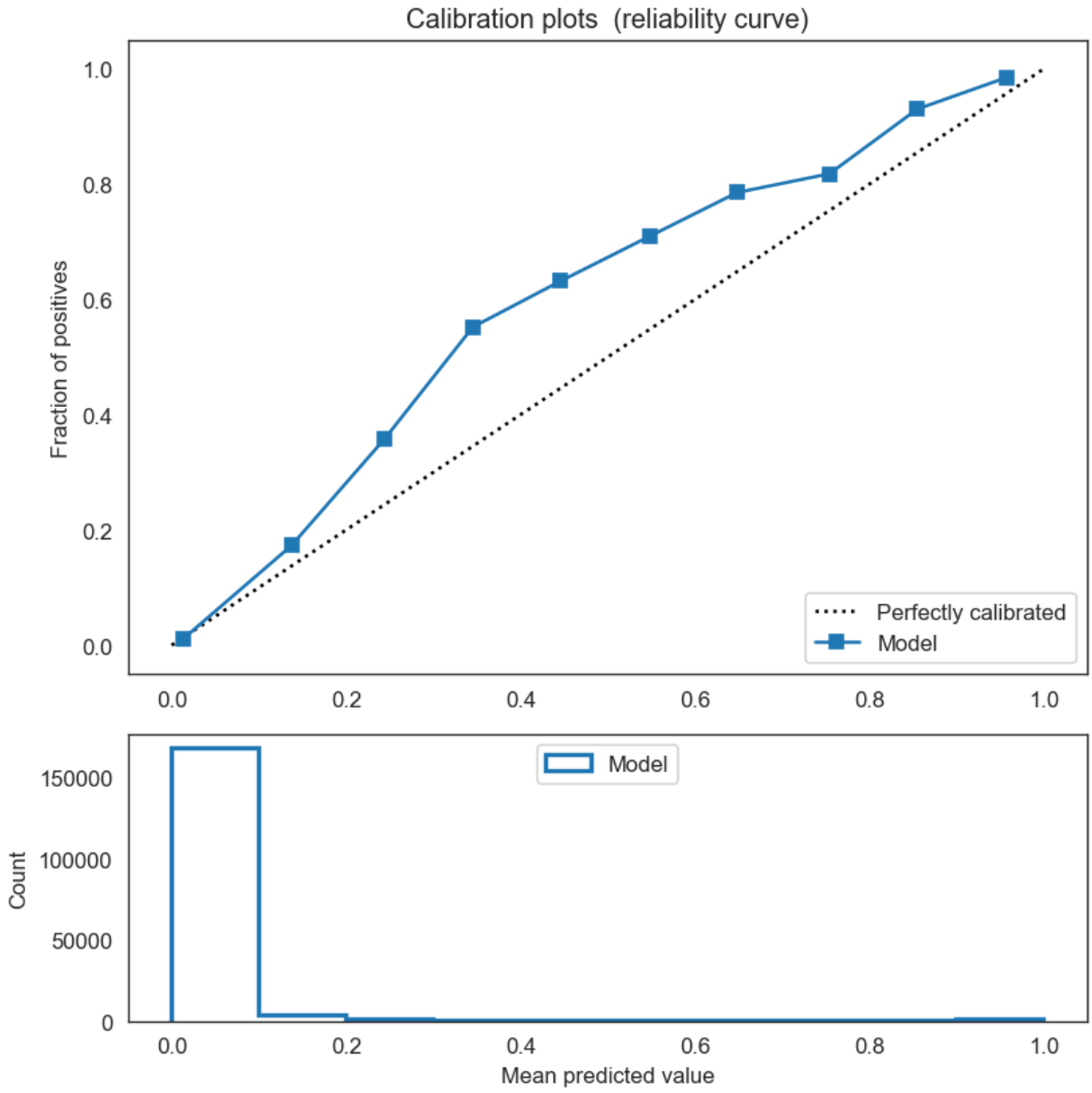
PR curve :




```
Additional Metrics:
TPR (Recall) : 0.4659
FPR : 0.0017
TNR (Specificity) : 0.9983
FNR : 0.5341
```

In [156…

```
draw_calibration_curve(y_test, y_prob_g_pred, n_bins=10)
```




In [157…
```
# Calibrate
calibrated_clf = CalibratedClassifierCV(base_estimator=lgbmclassifier, cv=3)
calibrated_clf.fit(X_train, y_train)
y_pred_calib = calibrated_clf.predict(X_test)
y_prob_pred_calib = calibrated_clf.predict_proba(X_test)[:, 1]
```

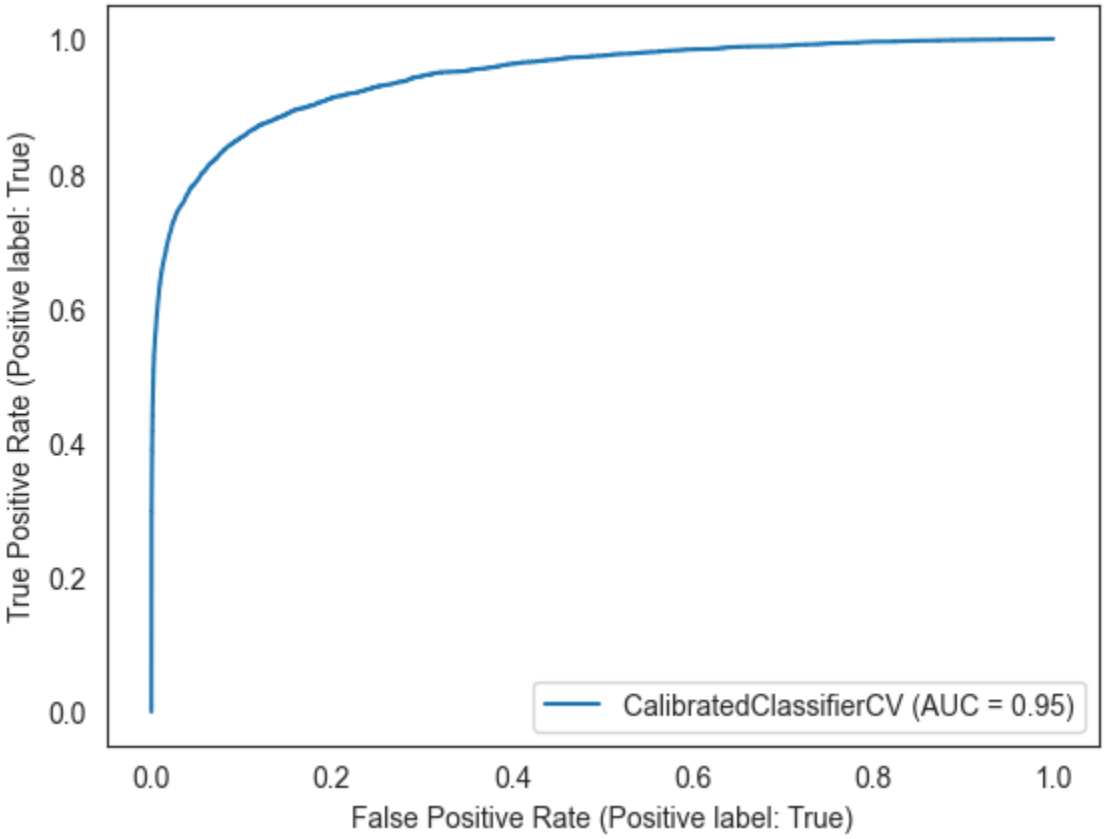
In [158…
```
compute_evaluation_metric(calibrated_clf, X_test, y_test, y_pred_calib, y_prob
```

```
Accuracy Score :  0.9804416296948556

AUC Score :  0.9456479526988175

Confusion Matrix :
 [[170572    391]
 [  3074   3125]]

Classification Report :
              precision    recall  f1-score   support

       False       0.98      1.00      0.99    170963
        True       0.89      0.50      0.64      6199
```

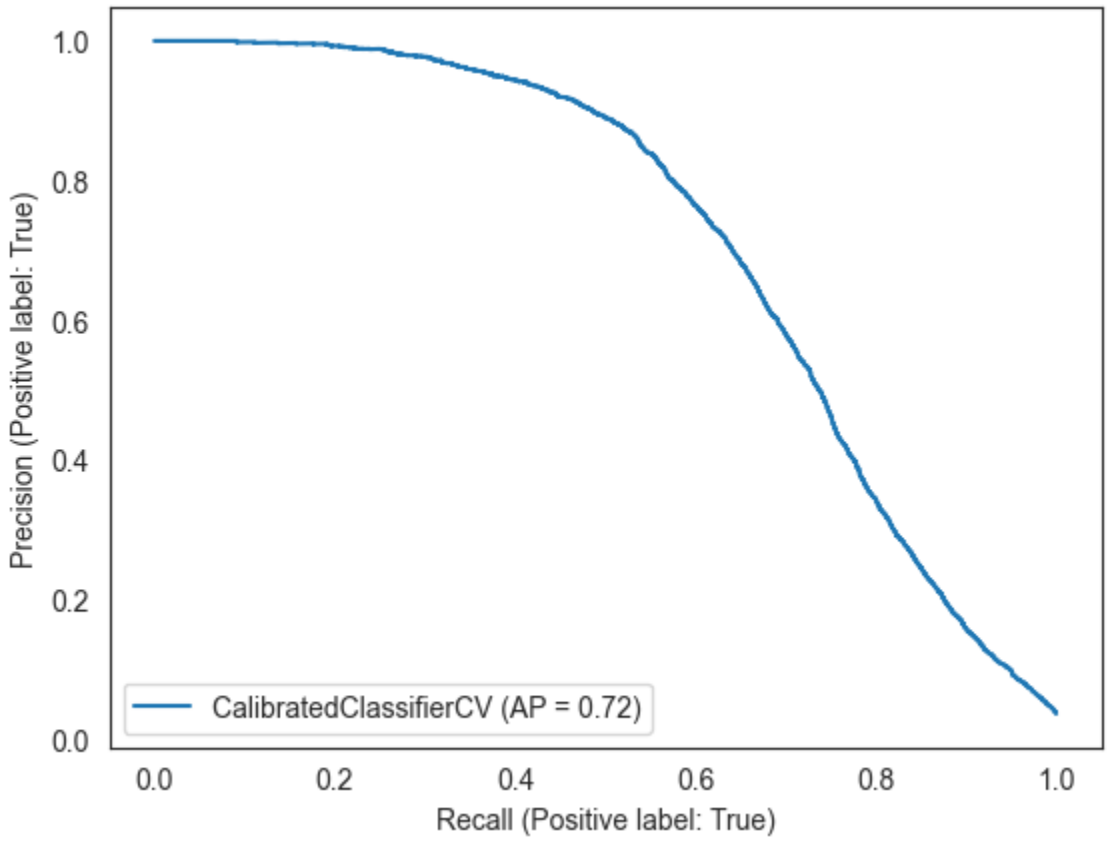
```
    accuracy                           0.98    177162
   macro avg       0.94      0.75      0.82    177162
weighted avg       0.98      0.98      0.98    177162
```

```
Concordance Index :  0.9456479526988175
```

```
ROC curve :
```



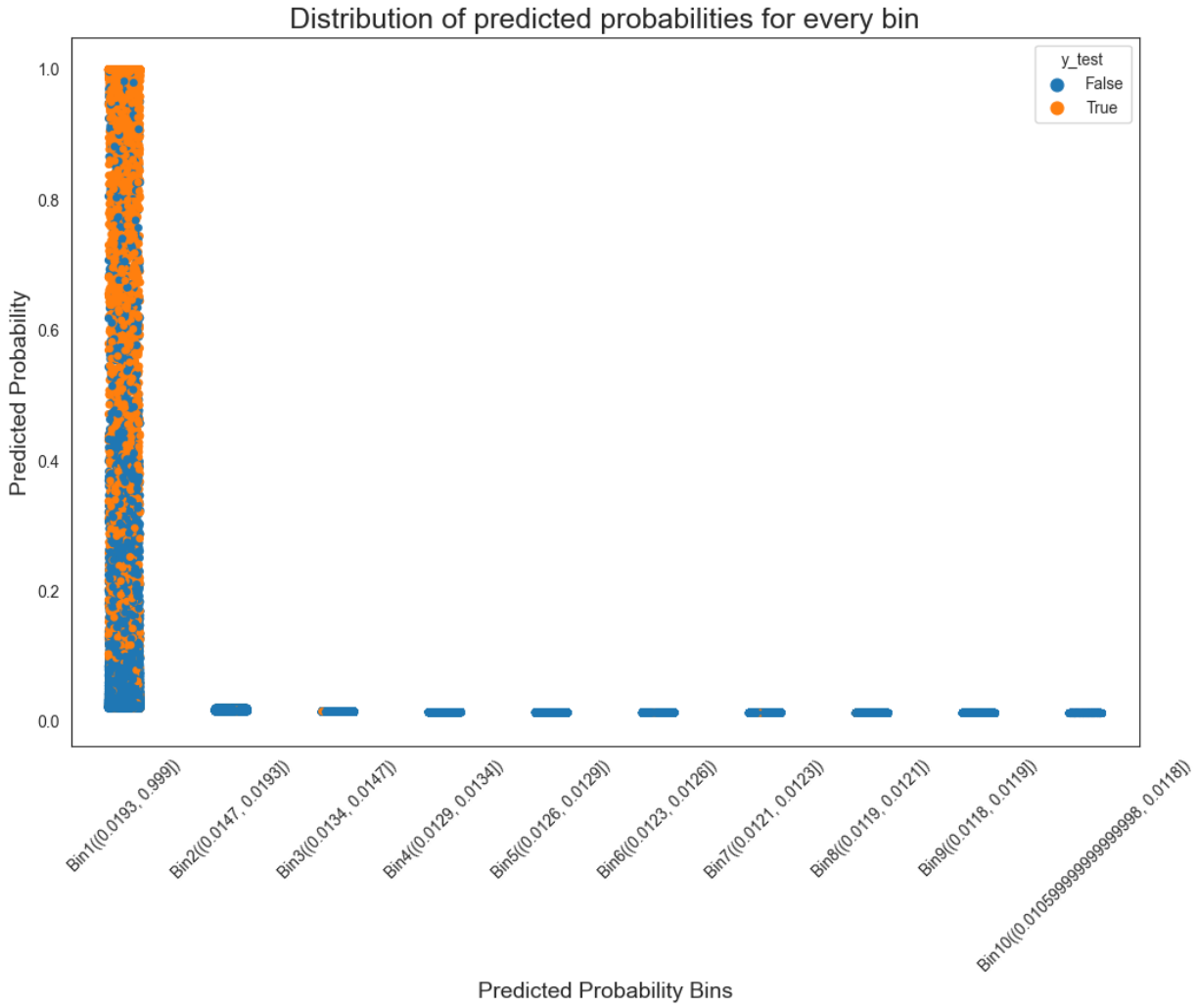

```
PR curve :
```

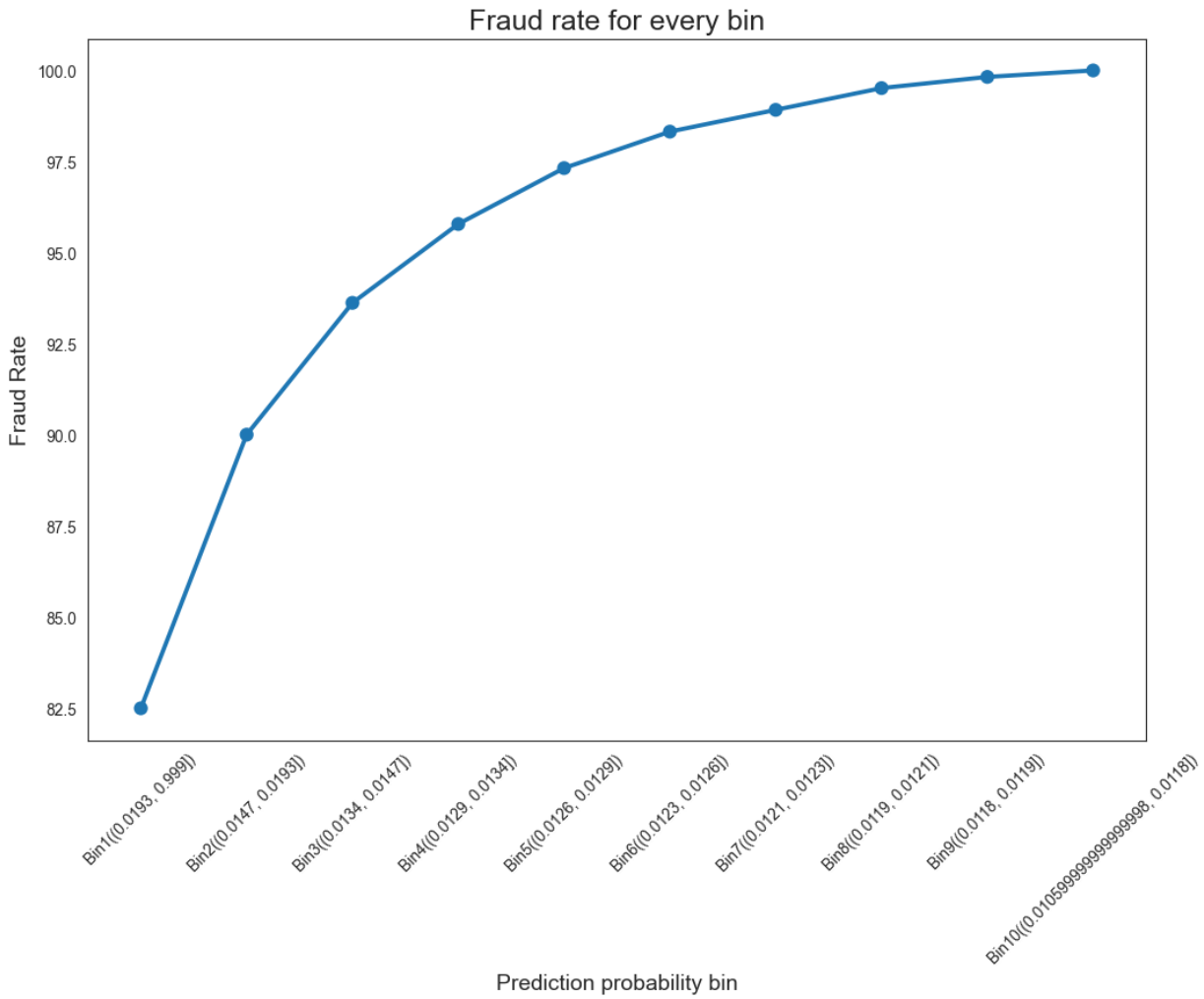



```
Additional Metrics:
TPR (Recall) : 0.5041
FPR : 0.0023
TNR (Specificity) : 0.9977
FNR : 0.4959
```

In [160…

```
captures(y_test, y_pred_calib, y_prob_pred_calib)
```



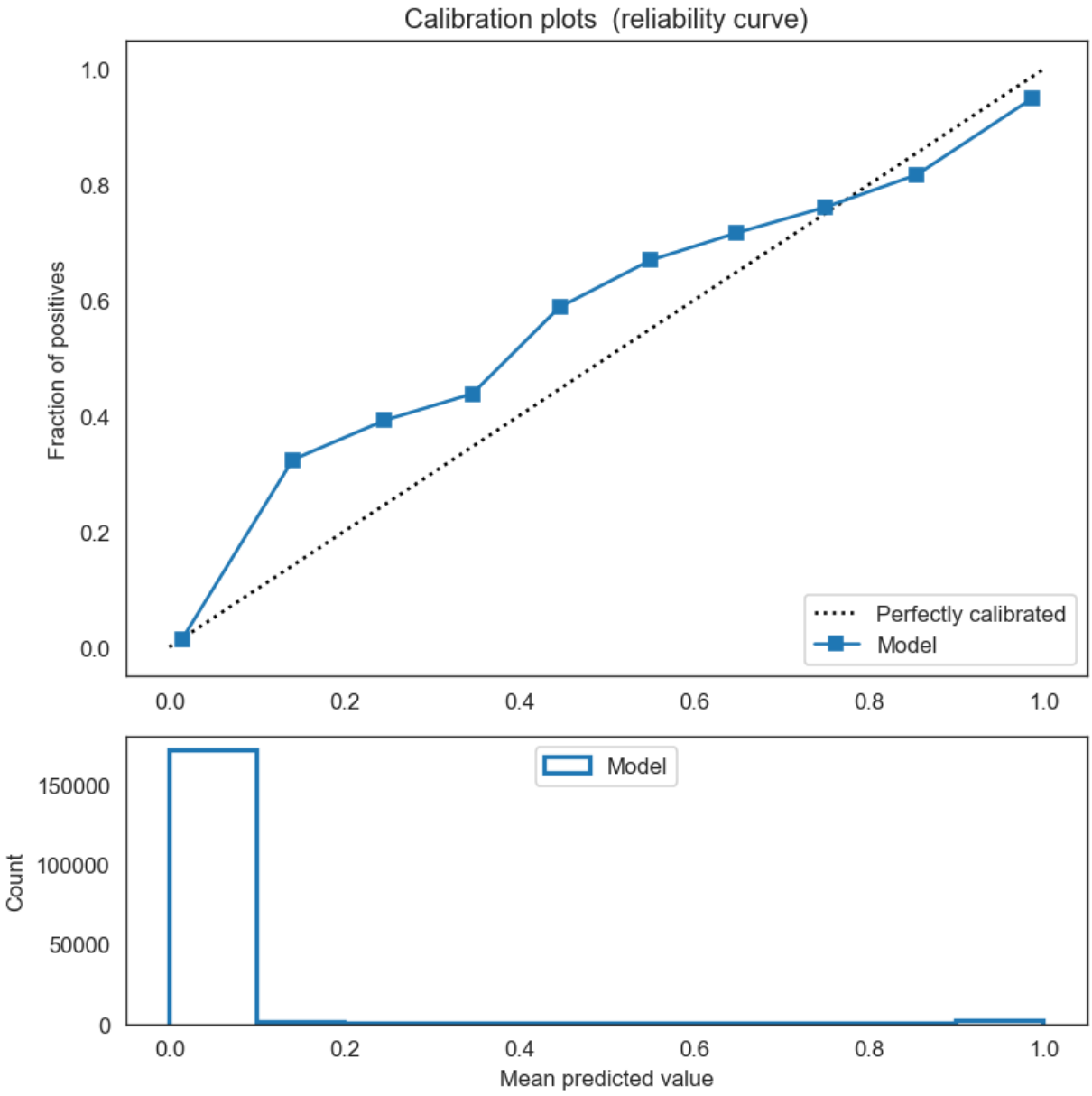



Out[160]:

| | prob_bin | not_fraud | fraud | perc_fraud | perc_not_fraud | cum_perc_fraud | cu |
|---|---|---|---|---|---|---|---|
| **0** | Bin1((0.0193, 0.999]) | 12602 | 5115 | 0.825133 | 0.073712 | 82.513309 | |
| **1** | Bin2((0.0147, 0.0193]) | 17251 | 465 | 0.075012 | 0.100905 | 90.014518 | |
| **2** | Bin3((0.0134, 0.0147]) | 17492 | 224 | 0.036135 | 0.102315 | 93.628005 | |
| **3** | Bin4((0.0129, 0.0134]) | 17582 | 134 | 0.021616 | 0.102841 | 95.789643 | |
| **4** | Bin5((0.0126, 0.0129]) | 17621 | 95 | 0.015325 | 0.103069 | 97.322149 | |
| **5** | Bin6((0.0123, 0.0126]) | 17654 | 62 | 0.010002 | 0.103262 | 98.322310 | |
| **6** | Bin7((0.0121, 0.0123]) | 17679 | 37 | 0.005969 | 0.103408 | 98.919181 | |
| **7** | Bin8((0.0119, 0.0121]) | 17679 | 37 | 0.005969 | 0.103408 | 99.516051 | |
| **8** | Bin9((0.0118, 0.0119]) | 17697 | 19 | 0.003065 | 0.103514 | 99.822552 | |
| **9** | Bin10((0.010599999999999998, 0.0118]) | 17706 | 11 | 0.001774 | 0.103566 | 100.000000 | |

In [159…

```
draw_calibration_curve(y_test, y_prob_pred_calib, n_bins=10)
```




- Accuracy score is 0.98
- AUC score and Concordance index are 0.95, which are the best so far
- Classfication report is also balanced between both the classes
- ROC curve and PR are the best so far

# 19. Feature Importance

In [161…
```
feature_importance_df = pd.DataFrame({'feature' : X_train.columns, 'importance
```

In [162…
```
feature_importance_df = feature_importance_df.sort_values(by="importance", asc
feature_importance_df = feature_importance_df.iloc[:30,:]
feature_importance_df
```

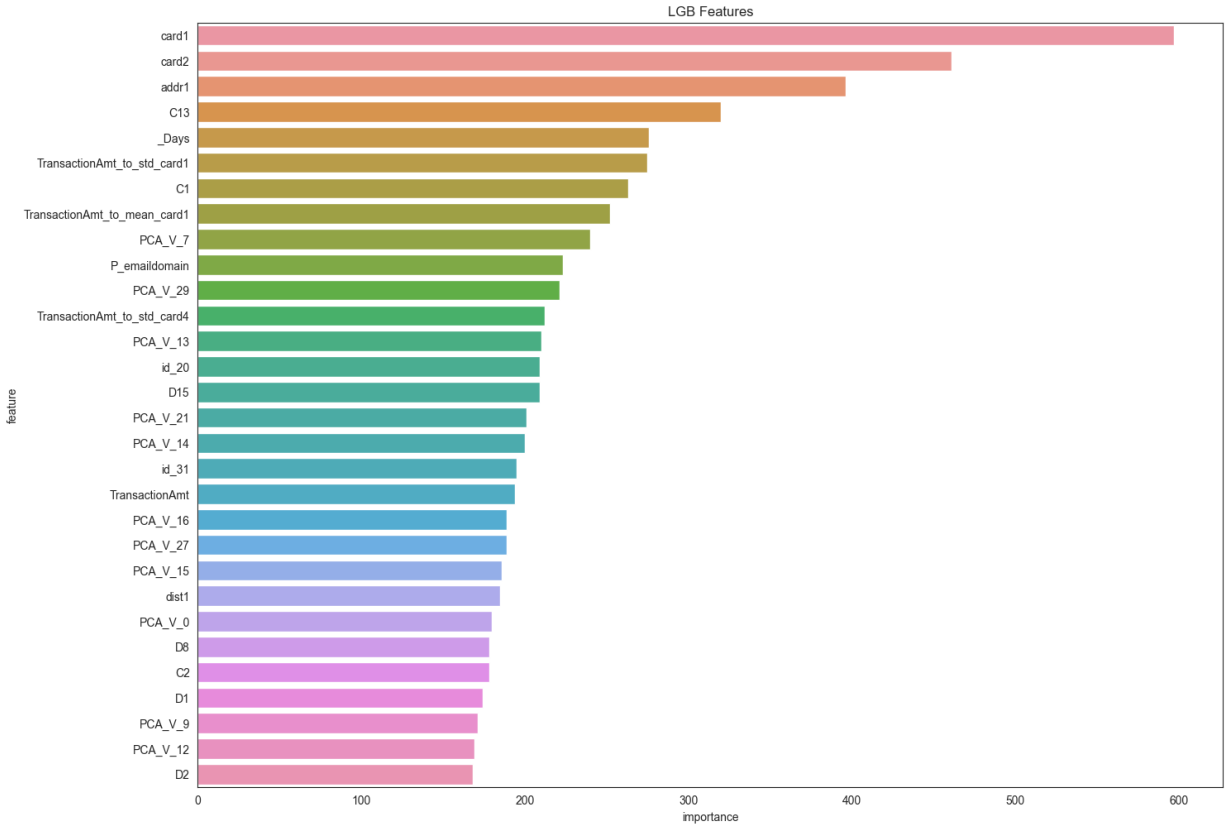
Out[162]:

| | feature | importance |
|---|---|---|
| **2** | card1 | 597 |
| **3** | card2 | 461 |

| | feature | importance |
|---|---|---|
| **8** | addr1 | 396 |
| **25** | C13 | 320 |
| **496** | _Days | 276 |
| **501** | TransactionAmt_to_std_card1 | 275 |
| **13** | C1 | 263 |
| **499** | TransactionAmt_to_mean_card1 | 252 |
| **510** | PCA_V_7 | 240 |
| **11** | P_emaildomain | 223 |
| **532** | PCA_V_29 | 221 |
| **502** | TransactionAmt_to_std_card4 | 212 |
| **516** | PCA_V_13 | 210 |
| **66** | id_20 | 209 |
| **40** | D15 | 209 |
| **524** | PCA_V_21 | 201 |
| **517** | PCA_V_14 | 200 |
| **70** | id_31 | 195 |
| **0** | TransactionAmt | 194 |
| **519** | PCA_V_16 | 189 |
| **530** | PCA_V_27 | 189 |
| **518** | PCA_V_15 | 186 |
| **10** | dist1 | 185 |
| **503** | PCA_V_0 | 180 |
| **33** | D8 | 178 |
| **14** | C2 | 178 |
| **27** | D1 | 174 |
| **512** | PCA_V_9 | 171 |
| **515** | PCA_V_12 | 169 |
| **28** | D2 | 168 |

In [163…

```
plt.figure(figsize=(16, 12));
sns.barplot(x="importance", y="feature", data=feature_importance_df.sort_value
plt.title('LGB Features');
```



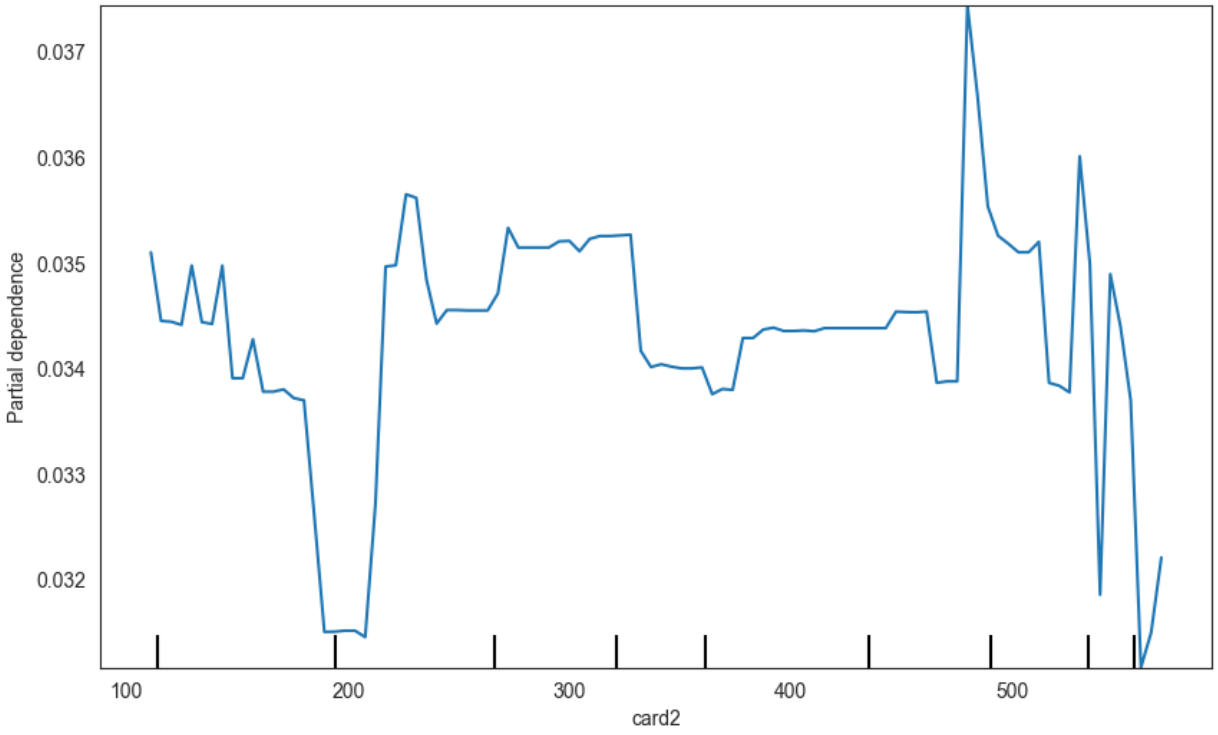

- card1 is contributing the most in predicting if a transaction is fraud or not
- card2, addr1, C13, P_emaildomain, C1,_days etc are some of the most important features in predicting the fraud
- Certain card types, addresses and emails are at high risk of fraud, so there is a need to monitor these carefully

# 20. Partial Dependence and Individual Conditional Expectations (ICE)

**Plot Partial Dependence**

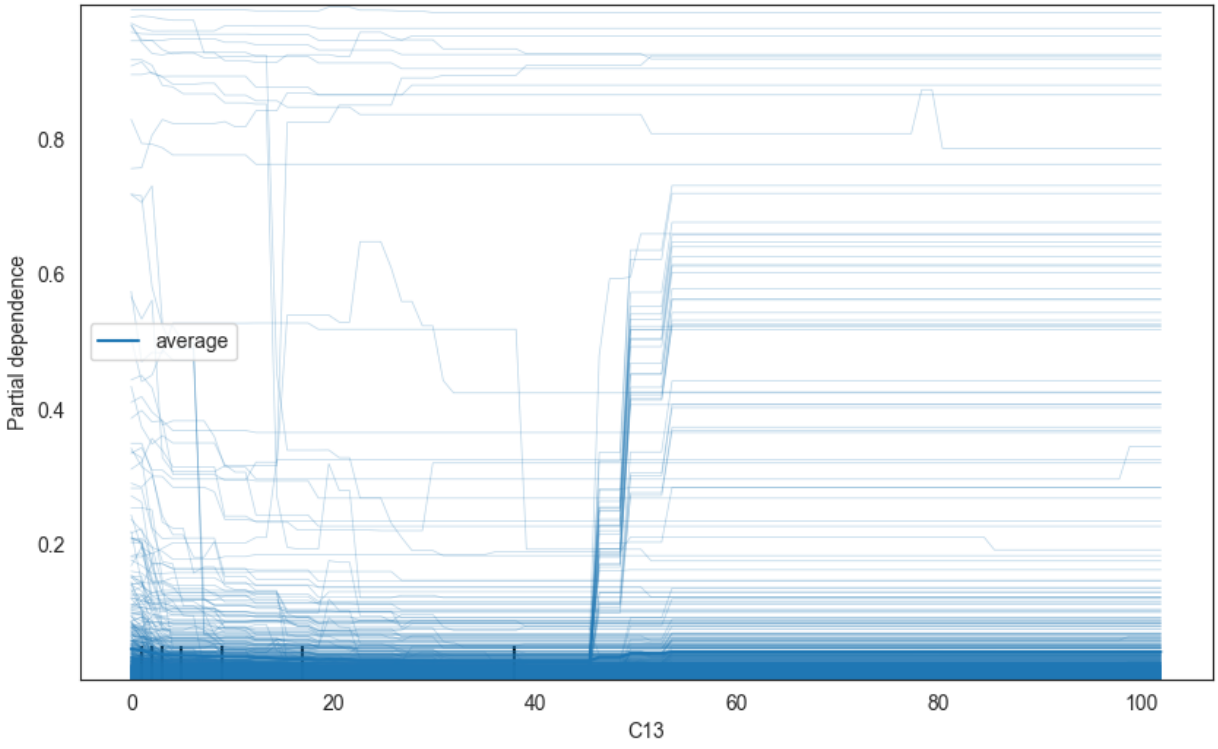
In [168…

```
from sklearn.inspection import PartialDependenceDisplay
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 6))
PartialDependenceDisplay.from_estimator(
    lgbmclassifier,        # trained LightGBM model
    X_train,               # your training data
    features=['card2'],    # feature name as a list
    ax=ax
)
plt.show()
```




In [170…

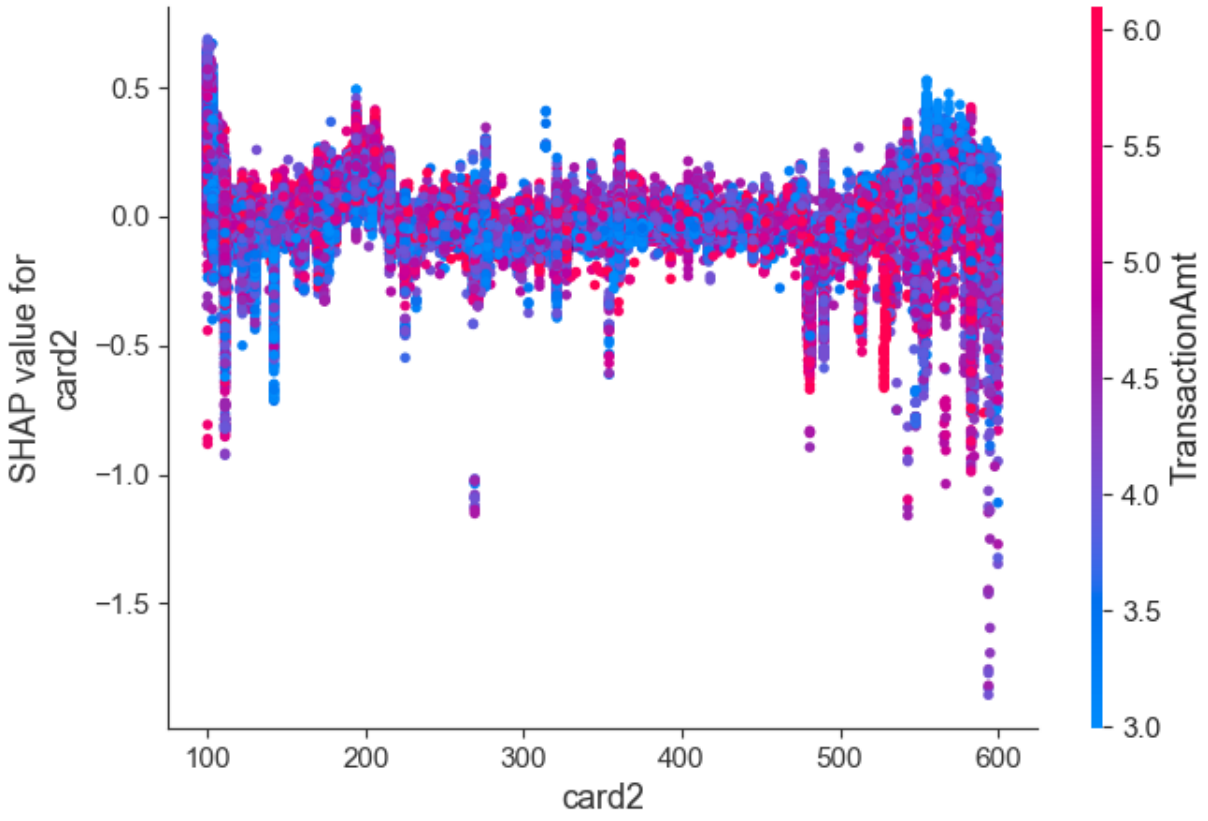
```
fig, ax = plt.subplots(figsize=(10, 6))

PartialDependenceDisplay.from_estimator(
    lgbmclassifier,
    X_train,
    features=['C13'],
    kind='both',
    ax=ax
)

plt.show()
```



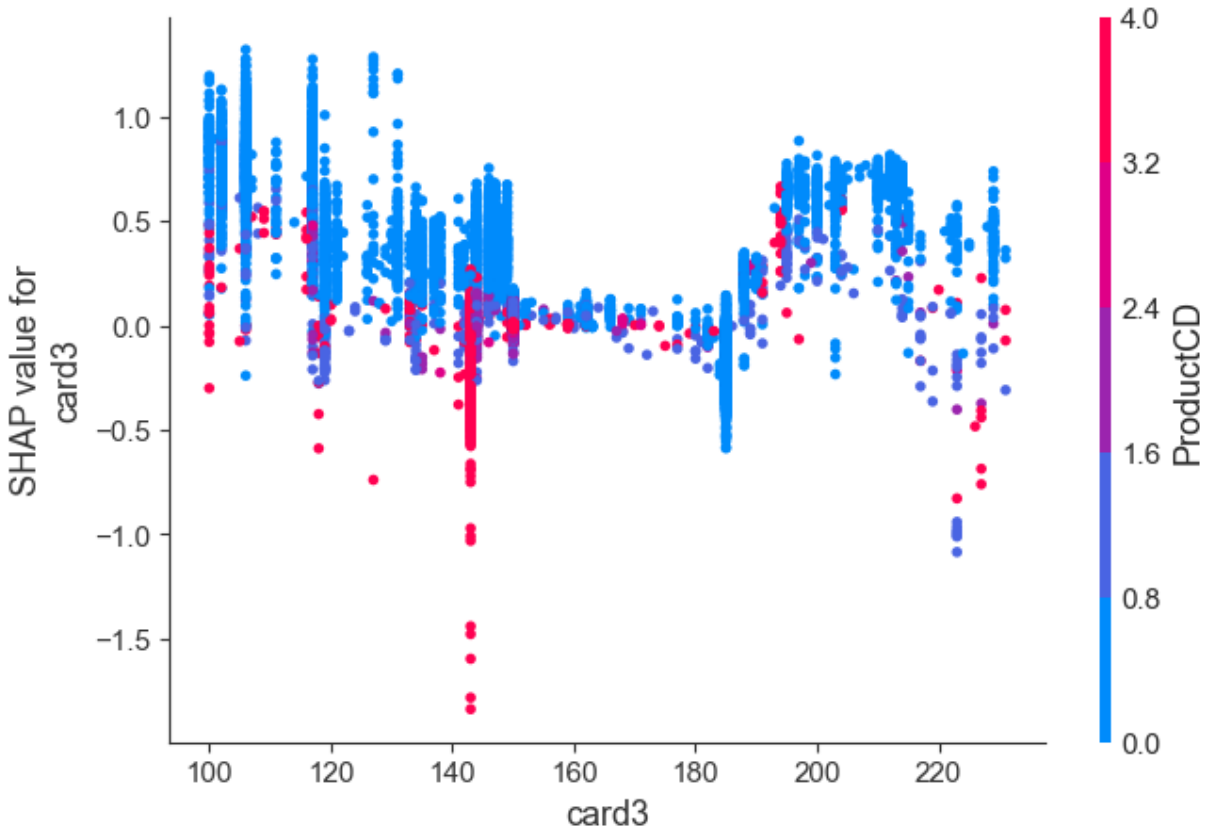
# 21. SHAP Values

In [172…]
```
import shap
shap_model = shap.TreeExplainer(lgbmclassifier)
shap_values = shap_model.shap_values(X_train)
```

In [173…]
```
shap.dependence_plot("card2", shap_values[0], X_train)
```

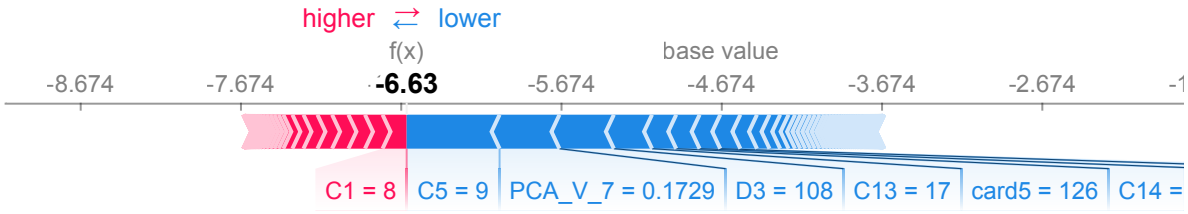



In [174…]
```
shap.dependence_plot("card3", shap_values[0], X_train)
```



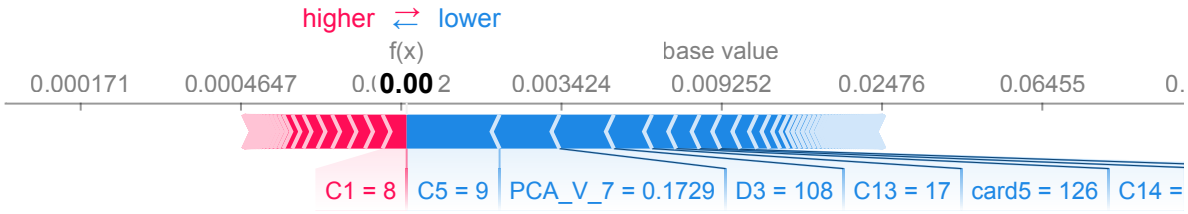

In [175…
```
shap.initjs()
shap.force_plot(shap_model.expected_value[1], shap_values[1][14], X_train.iloc
```

Out[175]:



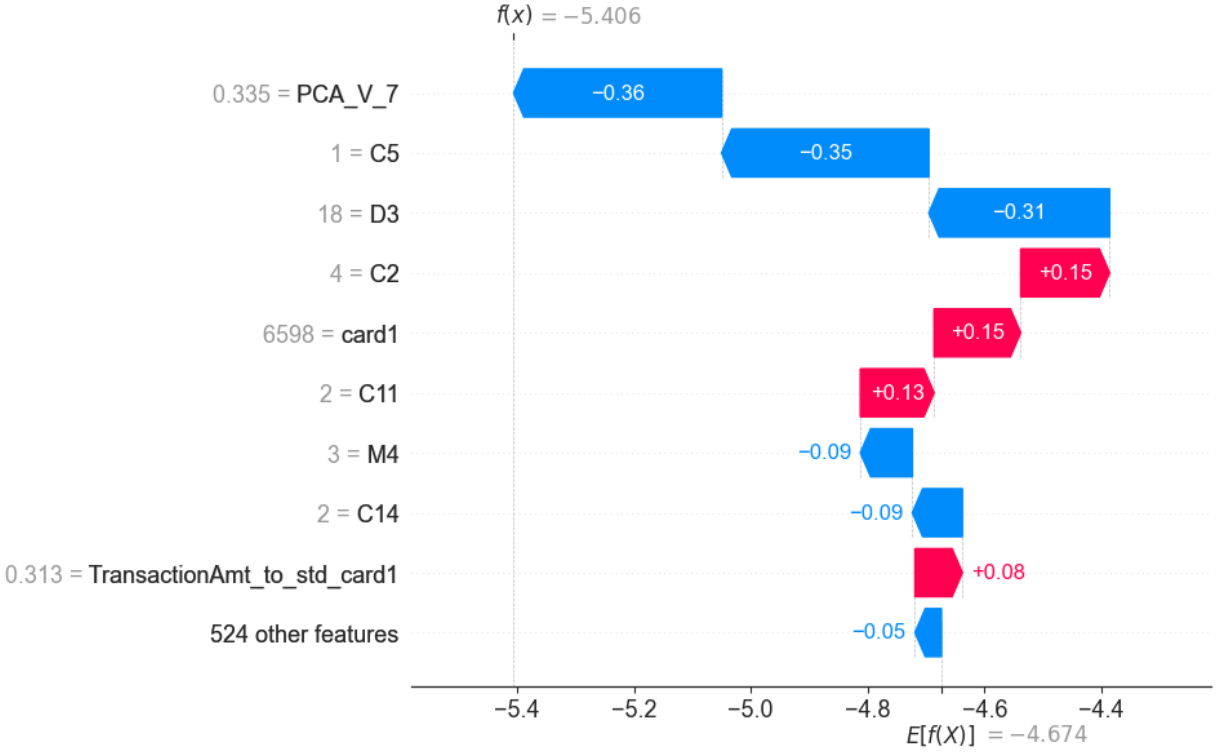

In [176…
```
shap.initjs()  # needed to show viz
shap.force_plot(shap_model.expected_value[1], shap_values[1][14], X_train.iloc
```

Out[176]:




In [183…
```
index = 0

shap.plots.waterfall(
```

```
    shap.Explanation(
        values=shap_values[1][index],
        base_values=shap_model.expected_value[1],
        data=X_train.iloc[index],
        feature_names=X_train.columns
    )
)
```




In [ ]:

# 👩‍💻 Author

**MIT TRIVEDI**

# 📂 Project

**Credit Card Fraud Detection**

# 📅 Date

**October 2025**

# 📬 Contact

LinkedIn · GitHub · Email

# 📖 Summary

This project presents an **end-to-end credit card fraud detection pipeline** featuring:

- **Models:** LightGBM and XGBoost
- **Feature Engineering:** Handling missing values, encoding, new features, etc.
- **Model Evaluation:** AUC, confusion matrix, precision, recall
- **Explainability:** SHAP visualizations for model interpretability

All **code and notebooks are available in this repository** for reproducibility and experimentation.

In [ ]: