cse519_hw2_?Aditya_Choudhary_112688862

September 26, 2019

1 Homework 2 - IEEE Fraud Detection

[0]:
```
import math
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

[0]:
```
import datetime
import math
from sklearn.preprocessing import StandardScaler,MinMaxScaler
```

[0]:
```
%matplotlib inline
```

[0]:
```
data_dir = ''
# save memory
cols = ['TransactionID','TransactionDT','isFraud',
        'TransactionAmt','ProductCD','card4','card6','P_emaildomain'
        ,'R_emaildomain','addr1','addr2','dist1','dist2']
cols_id = ['TransactionID','DeviceType','DeviceInfo']
```

[5]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id
=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire
ct_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googl
eapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%2
0https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
ûûûûûûûûûû
Mounted at /content/drive
```

[0]:
```
data_dir = '../content/drive/My Drive/data/'
```

[0]:
```
train_id = pd.read_csv(data_dir+'./ieee-fraud-detection/train_identity.
 ↪csv')#,usecols=cols_id)
train_txn = pd.read_csv(data_dir+'./ieee-fraud-detection/train_transaction.
 ↪csv',usecols=cols)
```

[8]:
```
train_txn.head()
```

[8]:
```
   TransactionID  isFraud  TransactionDT  ...  dist2 P_emaildomain R_emaildomain
0        2987000        0          86400  ...    NaN           NaN           NaN
1        2987001        0          86401  ...    NaN     gmail.com           NaN
2        2987002        0          86469  ...    NaN   outlook.com           NaN
3        2987003        0          86499  ...    NaN     yahoo.com           NaN
4        2987004        0          86506  ...    NaN     gmail.com           NaN

[5 rows x 13 columns]
```

1.0.1 Checking Columns

[0]:
```
train_txn.columns
```

[0]:
```
Index(['TransactionID', 'isFraud', 'TransactionDT', 'TransactionAmt',
       'ProductCD', 'card1', 'card2', 'card3', 'card4', 'card5',
       ...
       'V330', 'V331', 'V332', 'V333', 'V334', 'V335', 'V336', 'V337', 'V338',
       'V339'],
      dtype='object', length=394)
```

[0]:
```
train_id.head()
```

[0]:
```
   TransactionID  id_01     id_02  id_03  id_04  id_05  id_06  id_07  id_08  \
0        2987004    0.0   70787.0    NaN    NaN    NaN    NaN    NaN    NaN
1        2987008   -5.0   98945.0    NaN    NaN    0.0   -5.0    NaN    NaN
2        2987010   -5.0  191631.0    0.0    0.0    0.0    0.0    NaN    NaN
3        2987011   -5.0  221832.0    NaN    NaN    0.0   -6.0    NaN    NaN
4        2987016    0.0    7460.0    0.0    0.0    1.0    0.0    NaN    NaN

   id_09  ...                id_31  id_32      id_33           id_34  id_35  \
0    NaN  ...  samsung browser 6.2   32.0  2220x1080  match_status:2      T
1    NaN  ...   mobile safari 11.0   32.0   1334x750  match_status:1      T
2    0.0  ...          chrome 62.0    NaN        NaN             NaN      F
3    NaN  ...          chrome 62.0    NaN        NaN             NaN      F
4    0.0  ...          chrome 62.0   24.0   1280x800  match_status:2      T

  id_36 id_37  id_38  DeviceType                     DeviceInfo
0     F     T      T      mobile  SAMSUNG SM-G892A Build/NRD90M
1     F     F      T      mobile                     iOS Device
2     F     T      T     desktop                        Windows
3     F     T      T     desktop                            NaN
4     F     T      T     desktop                          MacOS
```

```
[5 rows x 41 columns]
```

[0]:
```python
train_id.columns
```

[0]:
```
Index(['TransactionID', 'id_01', 'id_02', 'id_03', 'id_04', 'id_05', 'id_06',
       'id_07', 'id_08', 'id_09', 'id_10', 'id_11', 'id_12', 'id_13', 'id_14',
       'id_15', 'id_16', 'id_17', 'id_18', 'id_19', 'id_20', 'id_21', 'id_22',
       'id_23', 'id_24', 'id_25', 'id_26', 'id_27', 'id_28', 'id_29', 'id_30',
       'id_31', 'id_32', 'id_33', 'id_34', 'id_35', 'id_36', 'id_37', 'id_38',
       'DeviceType', 'DeviceInfo'],
      dtype='object')
```

[0]:
```python
cols = ['TransactionID','DeviceType','DeviceInfo','TransactionDT',
        'TransactionAmt','ProductCD','card4','card6','P_emaildomain'
        ,'R_emaildomain','addr1','addr2','dist1','dist2']
```

[0]:
```python
cols
```

[0]:
```
['TransactionID',
 'DeviceType',
 'DeviceInfo',
 'TransactionDT',
 'TransactionAmt',
 'ProductCD',
 'card4',
 'card6',
 'P_emaildomain',
 'R_emaildomain',
 'addr1',
 'addr2',
 'dist1',
 'dist2']
```

[0]:
```python
len(cols)
```

[0]:
```
14
```

[0]:
```python
train_id = train
```

[0]:
```python
list = set(train_id.columns).intersection(set(cols))
```

[0]:
```
{'DeviceInfo', 'DeviceType', 'TransactionID'}
```

[0]:
```python
set(train_txn.columns).intersection(set(cols))
```

[0]:
```
{'P_emaildomain',
 'ProductCD',
 'R_emaildomain',
 'TransactionAmt',
 'TransactionDT',
 'TransactionID',
 'addr1',
```

```
 'addr2',
 'card4',
 'card6',
 'dist1',
 'dist2'}
```

[0]:
```
train_txn['dist1'].head()
```

[0]:
```
0     19.0
1      NaN
2    287.0
3      NaN
4      NaN
Name: dist1, dtype: float64
```

[0]:
```
train_txn['dist2'].describe()
```

[0]:
```
count    37627.000000
mean       231.855423
std        529.053494
min          0.000000
25%          7.000000
50%         37.000000
75%        206.000000
max      11623.000000
Name: dist2, dtype: float64
```

1.1 Part 1 - Fraudulent vs Non-Fraudulent Transaction

[0]:
```
import gc
gc.collect()
```

[0]:
```
11
```

[0]:
```
train_tot = pd.merge(train_txn,␣
 ↪train_id[['TransactionID','DeviceType','DeviceInfo']],how='left',on='TransactionID')
```

[11]:
```
train_tot.head()
```

[11]:
```
   TransactionID  isFraud  ...  DeviceType                     DeviceInfo
0        2987000        0  ...         NaN                            NaN
1        2987001        0  ...         NaN                            NaN
2        2987002        0  ...         NaN                            NaN
3        2987003        0  ...         NaN                            NaN
4        2987004        0  ...      mobile  SAMSUNG SM-G892A Build/NRD90M

[5 rows x 15 columns]
```

[0]:
```
train_tot.dtypes
```

[0]:
```
TransactionID       int64
isFraud             int64
TransactionDT       int64
TransactionAmt    float64
ProductCD          object
card4              object
card6              object
addr1             float64
addr2             float64
dist1             float64
dist2             float64
P_emaildomain      object
R_emaildomain      object
DeviceType         object
DeviceInfo         object
dtype: object
```

[0]:
```
train_fraud = train_tot[train_tot['isFraud']==1]
train_no_fraud = train_tot[train_tot['isFraud']==0]
```

[96]:
```
train_fraud.head()
```

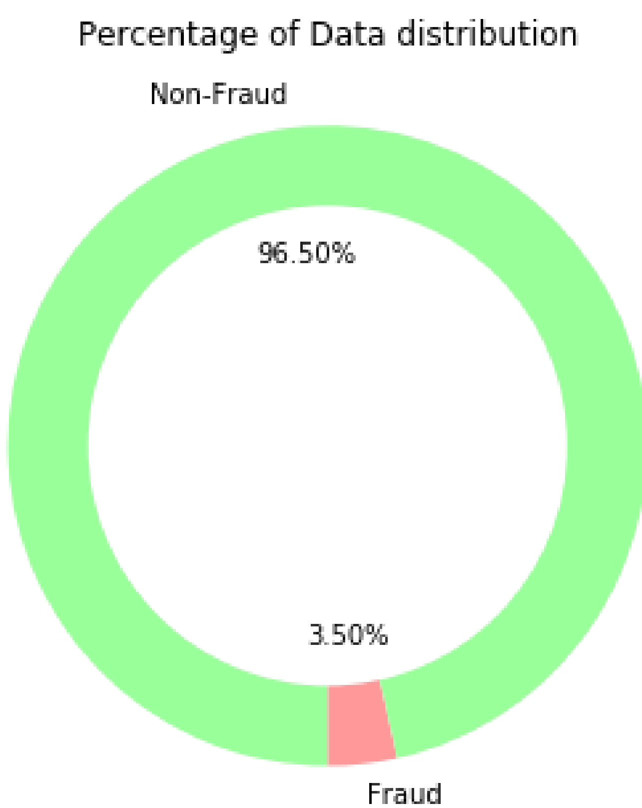
[96]:
```
     TransactionID  isFraud  ...  second  ScaledTransactionAmt
203        2987203        1  ...       0              0.013926
240        2987240        1  ...      13              0.001154
243        2987243        1  ...       6              0.001154
245        2987245        1  ...      55              0.001154
288        2987288        1  ...      26              0.004862

[5 rows x 20 columns]
```

[97]:
```
train_no_fraud.head()
```

[97]:
```
   TransactionID  isFraud  TransactionDT  ...  minute  second
ScaledTransactionAmt
0        2987000        0          86400  ...       0       0
0.002137
1        2987001        0          86401  ...       0       1
0.000900
2        2987002        0          86469  ...       1       9
0.001840
3        2987003        0          86499  ...       1      39
0.001558
4        2987004        0          86506  ...       1      46
0.001558

[5 rows x 20 columns]
```

[0]:
```
# Pie chart
labels = ['Fraud', 'Non-Fraud']
```

```
sizes = [train_fraud.shape[0], train_no_fraud.shape[0]]
colors = ['#ff9999','#99ff99']

fig, ax = plt.subplots()

ax.pie(sizes, colors = colors, labels=labels, autopct='%1.2f%%',␣
 ↪startangle=-90)#draw circle
centre_circle = plt.Circle((0,0),0.75,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
ax.axis('equal')
plt.tight_layout()
plt.title('Percentage of Data distribution\n')
plt.show()
```



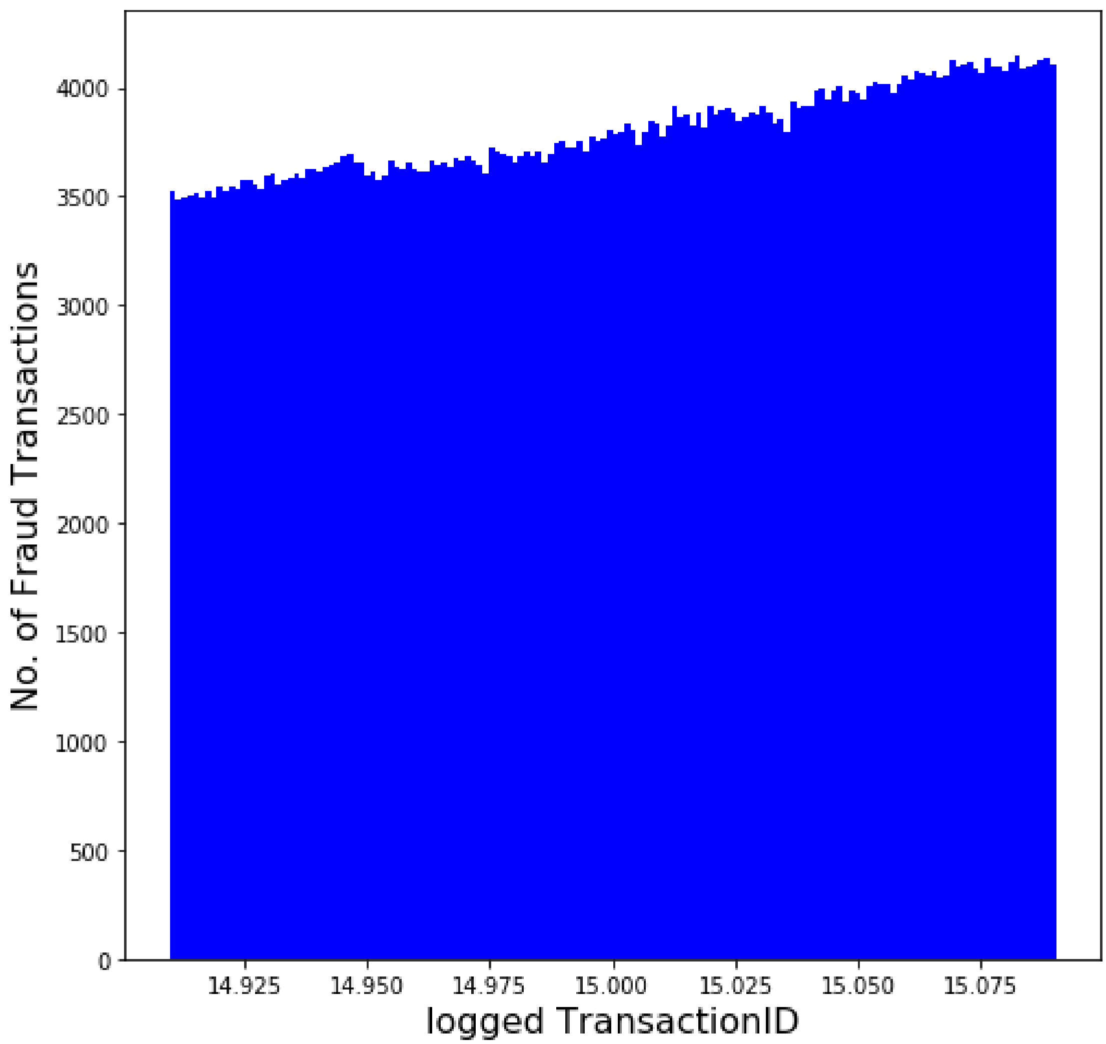
[0]:
```
TransactionID
DeviceType (mobile/desktop/...)
DeviceInfo (Windows/MacOS/...)
TransactionDT (time delta from reference)
TransactionAmt (amount in USD)
ProductCD (product code - W/C/H/R/...)
```

```
card4 (card issuer)
card6 (debit/credit)
P_emaildomain (purchaser email)
R_emaildomain (recipient email)
addr1 / addr2 (billing region / billing country)
dist1 / dist2 (some form of distance - address, zip code, IP, phone, ...)
```

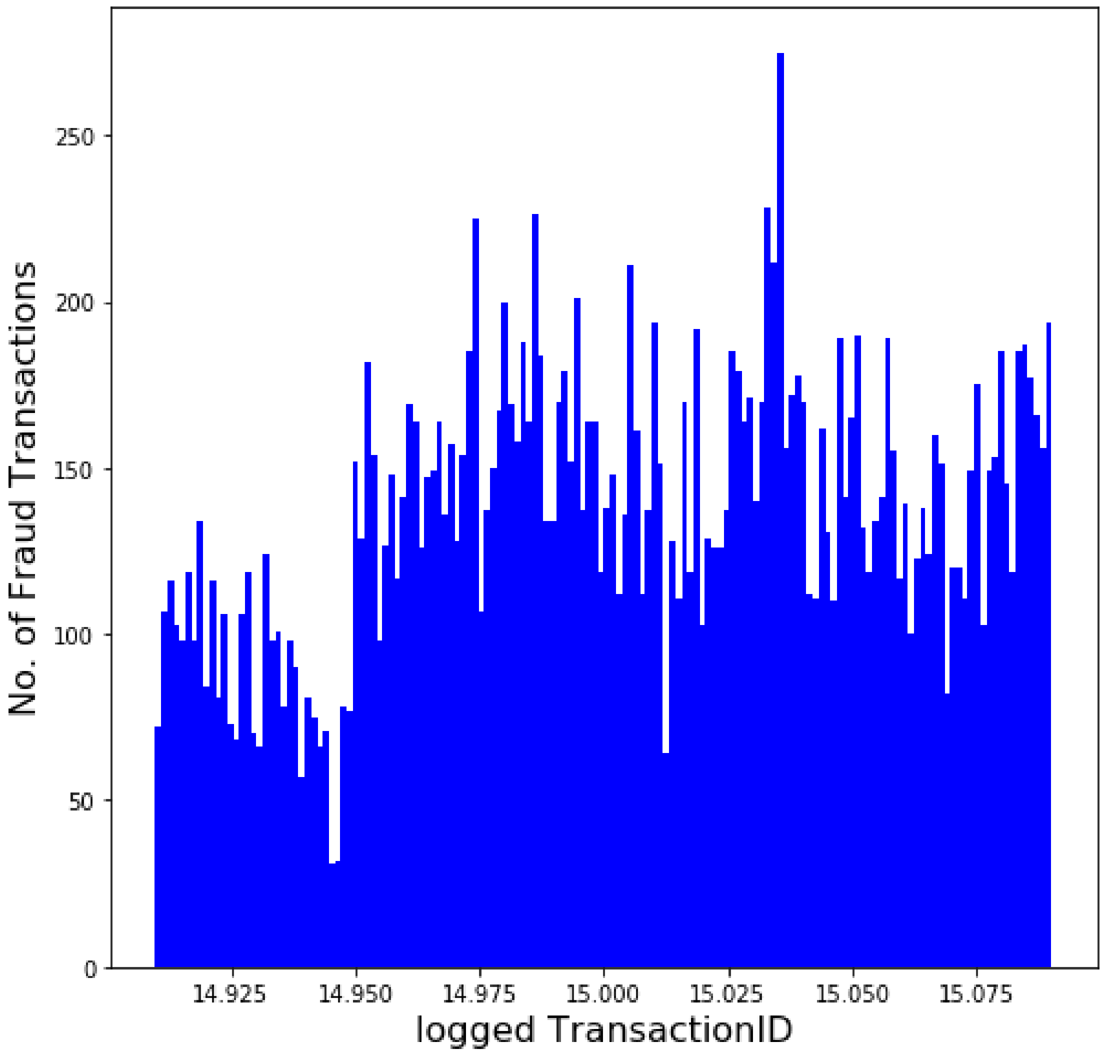
1.1.1 TransactionID

[0]:
```
train_tot['logged_transaction_id'] = np.log(train_tot['TransactionID'])
```

[149]:
```
plt.figure(figsize=(8,8))
train_tot[train_tot['isFraud']==0]['logged_transaction_id'].
 ↪hist(bins=150,grid=False)
plt.xlabel("logged TransactionID", fontsize=16)
plt.ylabel("No. of Fraud Transactions",fontsize=16)
plt.show()
```



[150]:
```
plt.figure(figsize=(8,8))
train_tot[train_tot['isFraud']==1]['logged_transaction_id'].
 ↪hist(bins=150,grid=False)
plt.xlabel("logged TransactionID", fontsize=16)
plt.ylabel("No. of Fraud Transactions",fontsize=16)
plt.show()
```



- in Non-Fraud cases value of TransactionID is increasing as well as the no. of transactions with higher value is also increasing , which is not the case in Fraud cases. **But** this doesn't mean that there is correlation between them

1.1.2 DeviceType

[0]:
```
train_no_fraud.head()
```

[0]:
```
   TransactionID  isFraud  TransactionDT  TransactionAmt ProductCD  \
0        2987000        0          86400            68.5         W
1        2987001        0          86401            29.0         W
2        2987002        0          86469            59.0         W
3        2987003        0          86499            50.0         W
4        2987004        0          86506            50.0         H

        card4   card6  addr1  addr2  dist1  dist2 P_emaildomain R_emaildomain  \
0    discover  credit  315.0   87.0   19.0    NaN           NaN           NaN
1  mastercard  credit  325.0   87.0    NaN    NaN     gmail.com           NaN
2        visa   debit  330.0   87.0  287.0    NaN   outlook.com           NaN
3  mastercard   debit  476.0   87.0    NaN    NaN     yahoo.com           NaN
4  mastercard  credit  420.0   87.0    NaN    NaN     gmail.com           NaN

  DeviceType                     DeviceInfo
0        NaN                            NaN
1        NaN                            NaN
2        NaN                            NaN
3        NaN                            NaN
4     mobile  SAMSUNG SM-G892A Build/NRD90M
```

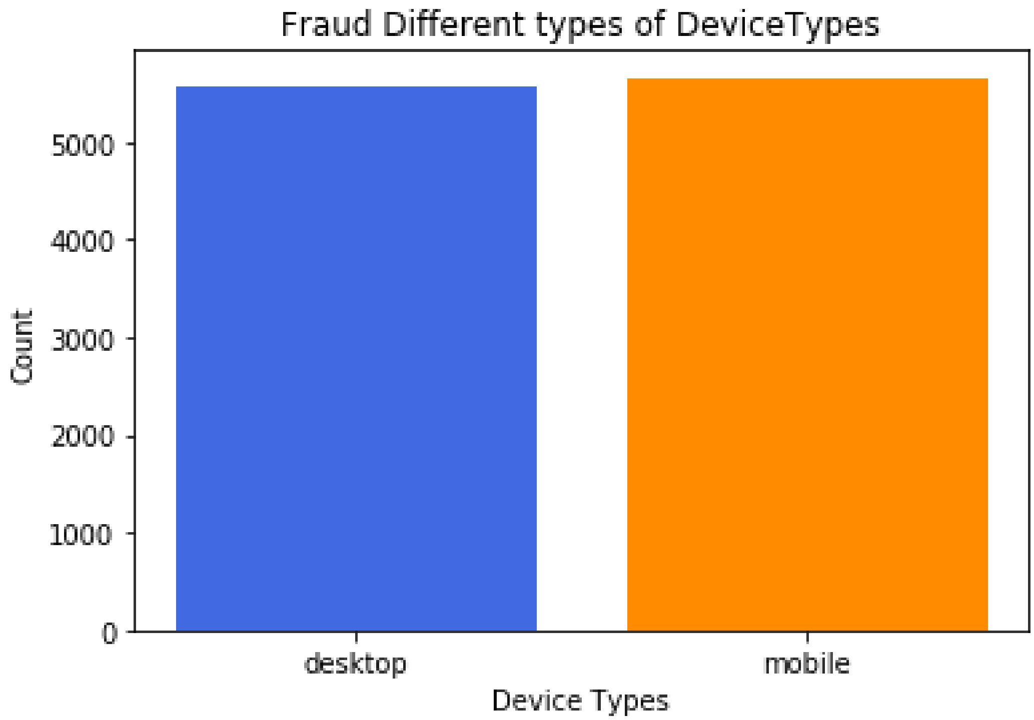
[0]:
```
train_fraud['DeviceType'].value_counts()
```

[0]:
```
mobile     5657
desktop    5554
Name: DeviceType, dtype: int64
```

[0]:
```
fraud_dtype = train_fraud[['isFraud','DeviceType']].groupby(by='DeviceType').
 ↪count().reset_index()
fraud_dtype
```

[0]:
```
  DeviceType  isFraud
0    desktop     5554
1     mobile     5657
```

[0]:
```
plt.bar(fraud_dtype['DeviceType'],fraud_dtype['isFraud'],
        color=['royalblue','darkorange'])
plt.title('Fraud Different types of DeviceTypes')
plt.xlabel('Device Types')
plt.ylabel('Count')
plt.show()
```

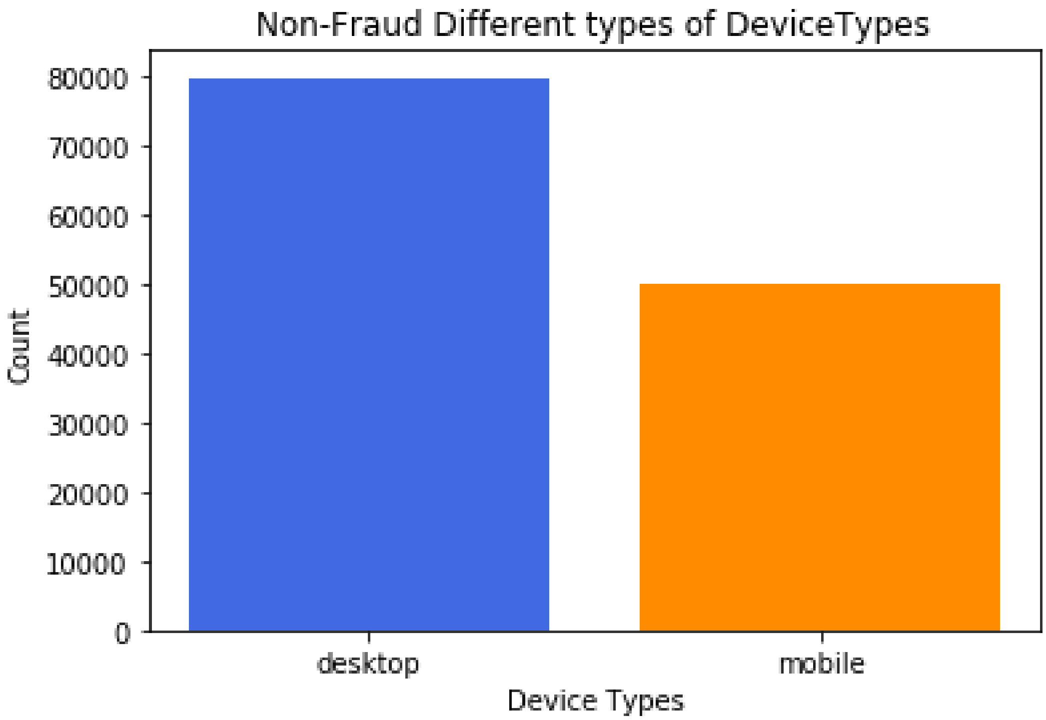


[0]:
```
no_fraud_dtype = train_no_fraud[['isFraud','DeviceType']].
  ↪groupby(by='DeviceType').count().reset_index()
no_fraud_dtype
```

[0]:

	DeviceType	isFraud
0	desktop	79611
1	mobile	49988

[0]:
```
plt.bar(no_fraud_dtype['DeviceType'],no_fraud_dtype['isFraud'],
        color=['royalblue','darkorange'])
plt.title('Non-Fraud Different types of DeviceTypes')
plt.xlabel('Device Types')
plt.ylabel('Count')
plt.colormaps
plt.show()
```



Inference - Fraud transactions are in equal proportion for 'desktop' and 'mobile' but in Non-Fraud case 'mobile' is less. We can infer Mobile is —???

1.1.3 DeviceInfo

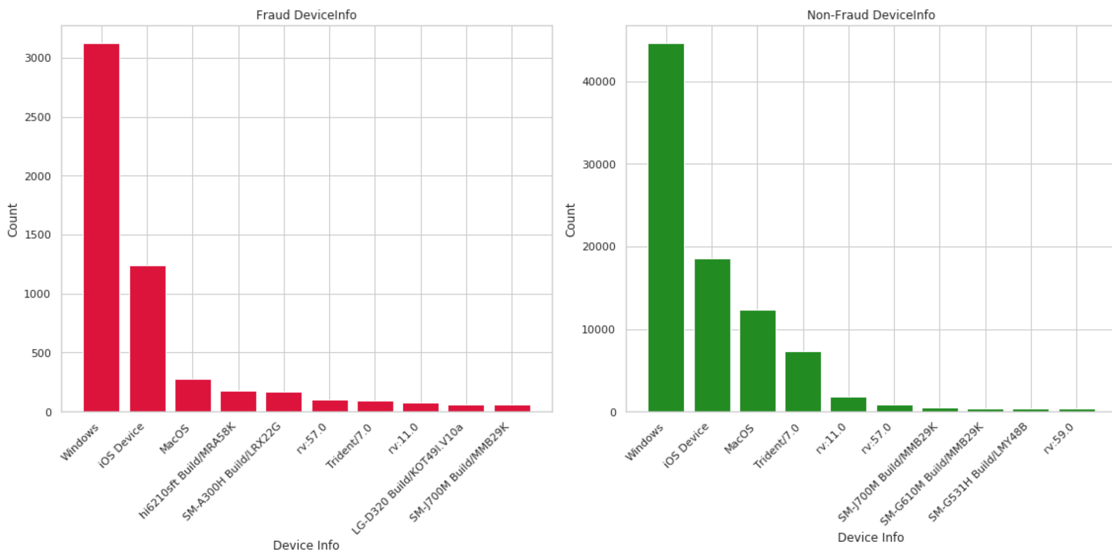
[0]:
```
fraud_dinfo = train_fraud[['isFraud','DeviceInfo']].groupby(by='DeviceInfo').
 ↪count().reset_index().sort_values(by='isFraud',ascending=False)
fraud_dinfo
```

[0]:
```
no_fraud_dinfo = train_no_fraud[['isFraud','DeviceInfo']].
 ↪groupby(by='DeviceInfo').count().reset_index().
 ↪sort_values(by='isFraud',ascending=False)
no_fraud_dinfo
```

Since there are 420 different values in Fraud Cases, plotting only the top 10

[0]:
```
plt.figure(figsize=(16,8))
plt.subplot(1, 2, 1)
plt.bar(fraud_dinfo['DeviceInfo'].head(10),fraud_dinfo['isFraud'].
 ↪head(10),color=['crimson'])
plt.xticks(rotation=45, ha='right')
plt.title('Fraud DeviceInfo')
plt.xlabel('Device Info')
plt.ylabel('Count')
```

```
# plt.show()
plt.subplot(1, 2, 2)
plt.bar(no_fraud_dinfo['DeviceInfo'].head(10),no_fraud_dinfo['isFraud'].
 ↪head(10),color=['forestgreen'])
plt.xticks(rotation=45, ha='right')
plt.title('Non-Fraud DeviceInfo')
plt.xlabel('Device Info')
plt.ylabel('Count')

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



- Inference - 'hi6210sft Build/MRA58K' is in 4th position for Fraud But not in Top-10 for Non-Fraud
- Similar variations are there for other types of Devices

[0]:

```
```

1.1.4 TransactionDT (time delta from reference)

[0]:

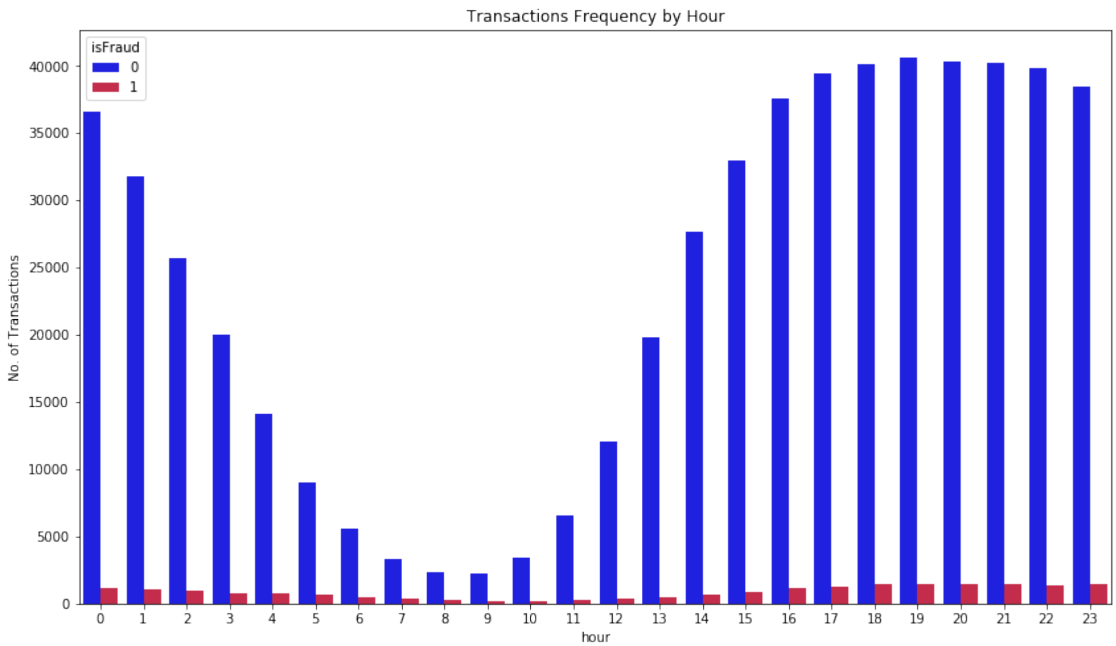
```
train_tot['hour'] = train_tot['TransactionDT'].apply(lambda x: math.floor((x/
 ↪(60*60))%24) )
train_tot['minute'] = train_tot['TransactionDT'].apply(lambda x: math.floor((x/
 ↪(60))%60) )
```

[137]:

```
plt.figure(figsize=(14,8))
sns.countplot(x='hour',data=train_tot,hue='isFraud')
plt.title('Transactions Frequency by Hour')
```

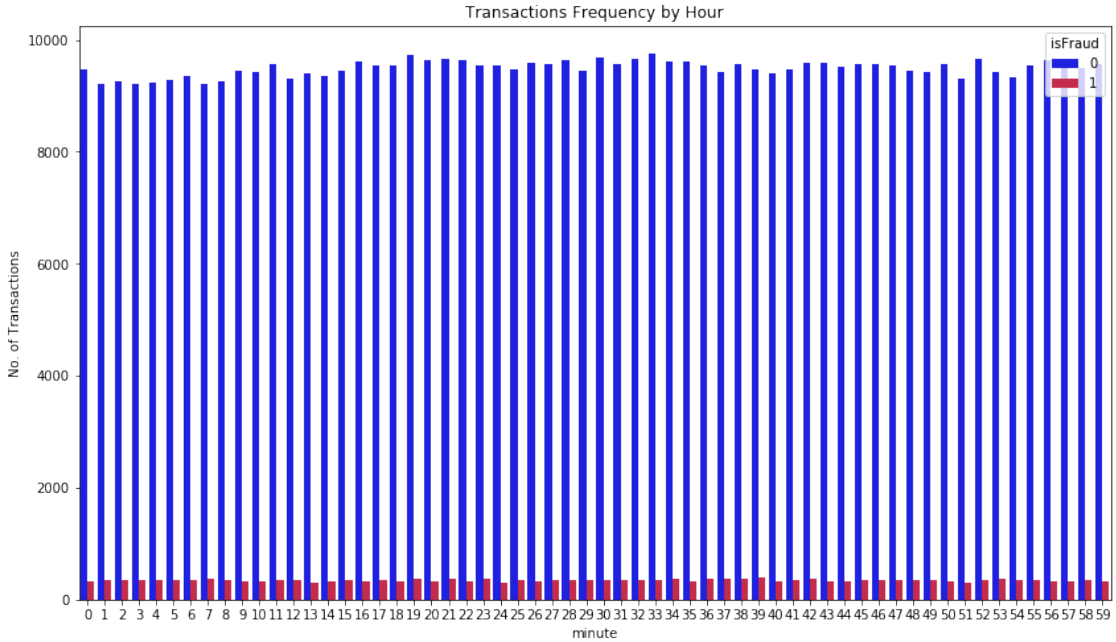
```
plt.ylabel('No. of Transactions')
```

[137]: Text(0, 0.5, 'No. of Transactions')



[138]:
```
plt.figure(figsize=(14,8))
sns.countplot(x='minute',data=train_tot,hue='isFraud')
plt.title('Transactions Frequency by Hour')
plt.ylabel('No. of Transactions')
```

[138]: Text(0, 0.5, 'No. of Transactions')



- Significant variation in No. of transactions is there amongst hours, showing us the sleeping hours of majority population

[0]:
```
gc.collect()
```

[0]:
```
86808
```

1.1.5 TransactionAmt

[0]:
```
train_fraud['TransactionAmt'].describe()
```

[0]:
```
count    20663.000000
mean       149.244779
std        232.212163
min          0.292000
25%         35.044000
50%         75.000000
75%        161.000000
max       5191.000000
Name: TransactionAmt, dtype: float64
```

[0]:
```
train_no_fraud['TransactionAmt'].describe()
```

[0]:
```
count    569877.000000
mean        134.511665
std         239.395078
min           0.251000
25%          43.970000
```

```
50%          68.500000
75%         120.000000
max       31937.391000
Name: TransactionAmt, dtype: float64
```

[0]:
```
trxn_amt = pd.
 ↪concat([train_fraud[['TransactionAmt','isFraud']],train_no_fraud[['TransactionAmt','isFraud']
```

[0]:
```
trxn_amt['TransactionAmt'].values
```

[0]:
```
array([445.    ,  37.098,  37.098, ...,  30.95 , 117.    , 279.95 ])
```

[0]:
```
s = StandardScaler()
trxn_amt['TransactionAmt'] = s.fit_transform(trxn_amt['TransactionAmt'].values.
 ↪reshape(-1,1))
```

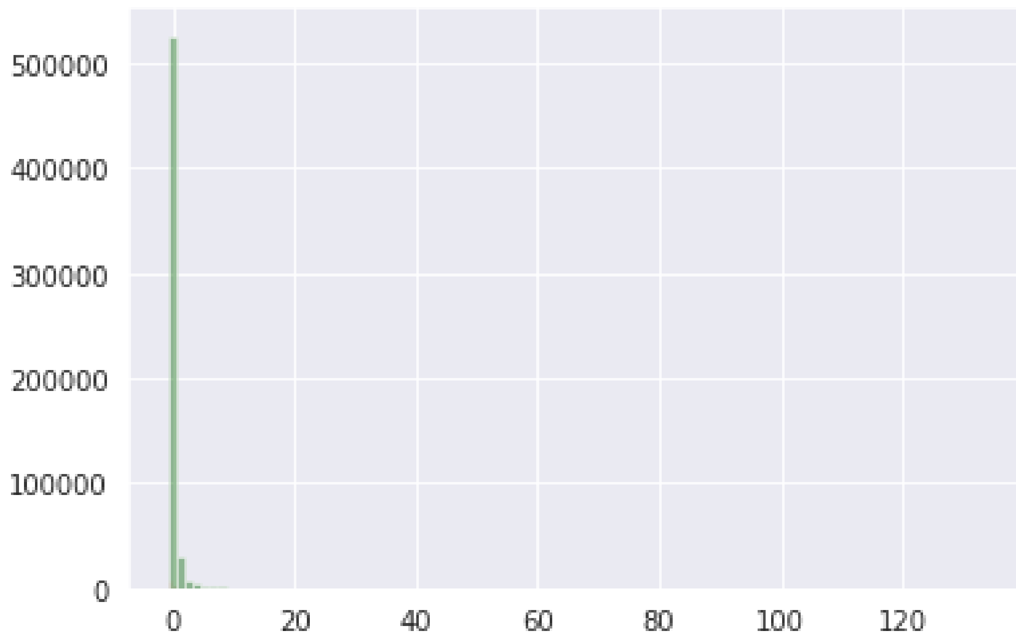
[0]:
```
trxn_amt_fraud = trxn_amt[trxn_amt['isFraud'] == 1]
trxn_amt_no_fraud = trxn_amt[trxn_amt['isFraud'] == 0]
```

[0]:
```
plt.figure(figsize=(16,8))
plt.subplot(1, 2, 1)
sns.distplot(train_fraud['TransactionAmt'], label='Fraud',kde=False,
             vertical=True,color='crimson')
plt.legend(prop={'size': 20})
plt.title('TransactionAmt Distribution')
plt.xlabel('No. of Transactions')
# plt.ylabel('')

plt.subplot(1, 2, 2)

sns.distplot(train_no_fraud['TransactionAmt'], label='Non-Fraud',kde=False,
             vertical=True,color='forestgreen')
plt.legend(prop={'size': 20})
plt.title('TransactionAmt Distribution')
plt.xlabel('No. of Transactions')

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

- In Fraud cases higher Amt. transactions are also in significant proportion

[0]:
```
# sns.distplot(train_fraud['TransactionAmt'], label='Fraud',kde=False,
#               color='crimson')
# plt.legend(prop={'size': 20})
# plt.title('TransactionAmt Distribution')
# plt.xlabel('No. of Transactions')
s = StandardScaler()
dat = s.fit_transform(train_fraud['TransactionAmt'].values.reshape(-1,1))

sns.distplot(dat, label='Non-Fraud',kde=False,
             color='crimson',bins=100)

dat = s.fit_transform(train_no_fraud['TransactionAmt'].values.reshape(-1,1))

sns.distplot(dat, label='Non-Fraud',kde=False,
             color='darkgreen',bins=100)
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5634112e48>
```



[0]:
```
# fig, ax = plt.subplots()
# for a in [train_no_fraud['TransactionAmt'].values,␣
 ↪train_fraud['TransactionAmt'].values]:
#     sns.distplot(a,  ax=ax,bins=range(1, 800, 10) ,kde=False)
# # ax.set_xlim([0, 200])

# fig, ax = plt.subplots()
# for a in [train_no_fraud['TransactionAmt'].values,␣
 ↪train_fraud['TransactionAmt'].values]:
#     sns.distplot(a,  ax=ax,bins=range(1, 200, 10) ,kde=False)
```

[0]:
```
df = pd.melt(trxn_amt, value_vars=['TransactionAmt'], id_vars='isFraud')
```

[0]:
```
df.head()
```

[0]:

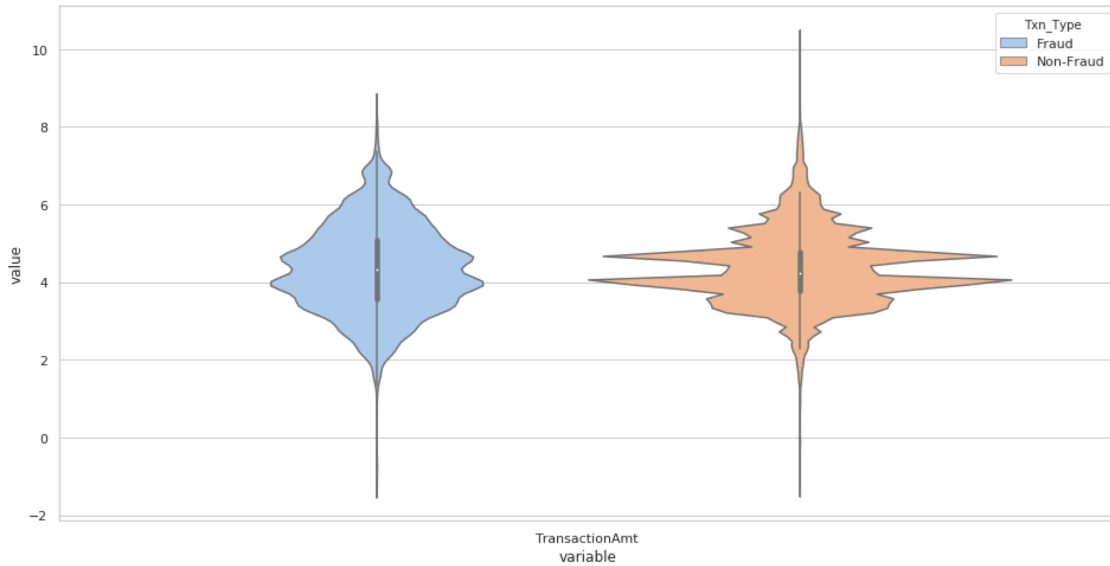
	isFraud	variable	value
0	1	TransactionAmt	1.296077
1	1	TransactionAmt	-0.409467
2	1	TransactionAmt	-0.409467
3	1	TransactionAmt	-0.409467
4	1	TransactionAmt	0.085690

[0]:
```
dicto = {1:'Fraud',0:'Non-Fraud'}
```

[0]:
```
df['value'] = df['value'].apply(lambda x: math.log(x))
df['Txn_Type'] = df['isFraud'].map(dicto)
df.drop(columns=['isFraud'],inplace=True)
```

[0]:
```
# sns.violinplot(x='TransactionAmt', y='TransactionAmt', hue='isFraud',␣
↪data=trxn_amt)
# plt.figure(figsize=(16,8))
# sns.violinplot(x='variable', y='value', hue='Txn_Type', data=df)

# plt.show()
```

[0]:
```
# sns.violinplot(x='TransactionAmt', y='TransactionAmt', hue='isFraud',␣
↪data=trxn_amt)
plt.figure(figsize=(16,8))
sns.violinplot(x='variable', y='value', hue='Txn_Type', data=df)

plt.show()
```



- Both Farud and Non-Fraud has similar distribution for Transaction Amt with 2 peaks but in Non-Fraud cases the peaks are more peculiar

[0]:
```
gc.collect()
```

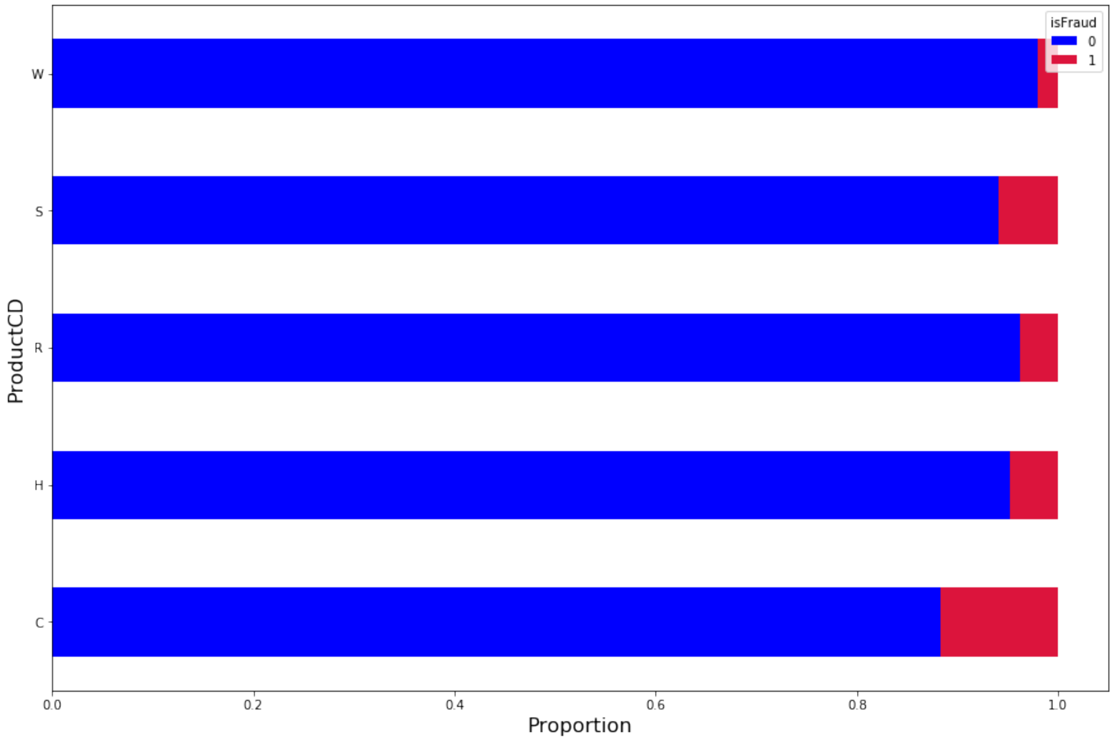
[0]: 691650

1.1.6 ProductCD

[0]:
```
# props.plot??
```

[128]:
```
fig, ax = plt.subplots(figsize=(15, 10))

props = train_tot.groupby("ProductCD")['isFraud'].value_counts(normalize=True).
↪unstack()
```

```
sns.set_palette(['blue', 'crimson'])
props.plot(kind='barh', stacked='True', ax=ax)
plt.xlabel('Proportion',size=16)
plt.ylabel('ProductCD',size=16)
plt.show()
```



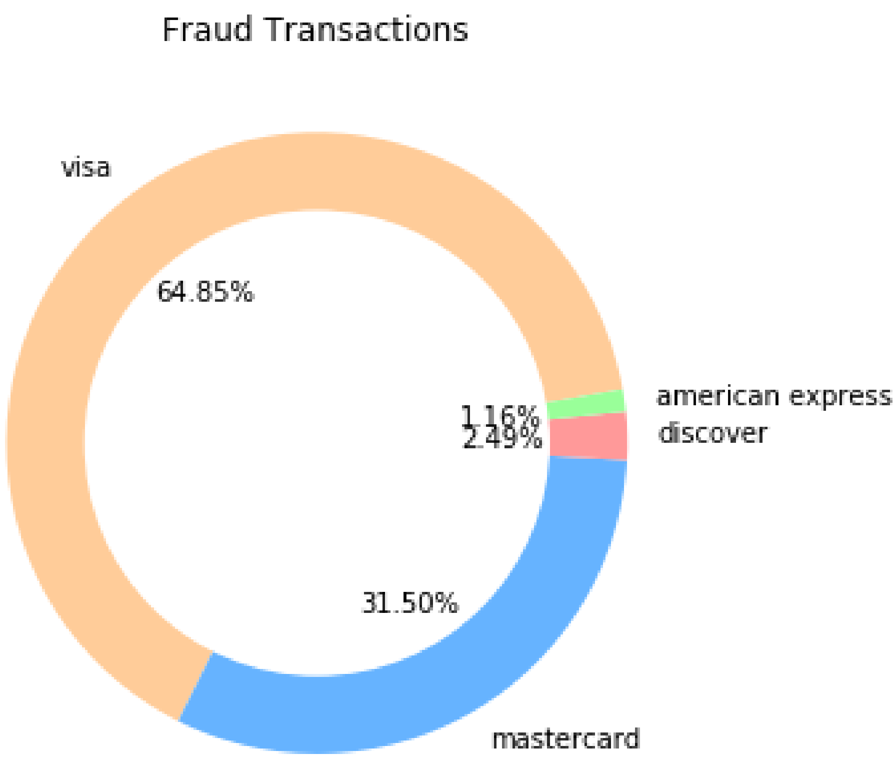
- 'C' type is more frequent in Fraud cases compared to other Product Codes

1.1.7 card4 (card issuer)

[49]:
```
# Pie chart
labels = train_fraud['card4'].value_counts().index
sizes = train_fraud['card4'].value_counts().values
colors = ['#ffcc99','#66b3ff','#ff9999','#99ff99']

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

fig, ax = plt.subplots()

ax.pie(sizes, colors = colors, labels=labels, autopct='%1.2f%%',␣
↪startangle=10)#draw circle
centre_circle = plt.Circle((0,0),0.75,fc='white')
```

```
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
ax.axis('equal')
plt.tight_layout()
plt.title('Fraud Transactions\n')
plt.show()
```

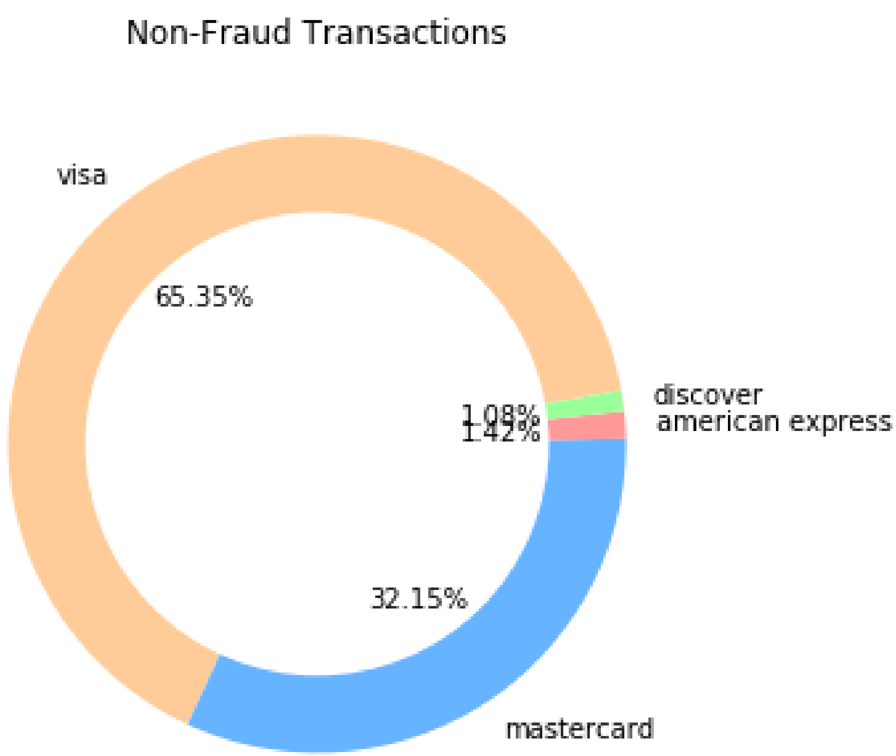
```
<Figure size 432x432 with 0 Axes>
```



[50]:
```
# Pie chart
labels = train_no_fraud['card4'].value_counts().index
sizes = train_no_fraud['card4'].value_counts().values
colors = ['#ffcc99','#66b3ff','#ff9999','#99ff99']

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

fig, ax = plt.subplots()

ax.pie(sizes, colors = colors, labels=labels, autopct='%1.2f%%',␣
 ↪startangle=10)#draw circle
centre_circle = plt.Circle((0,0),0.75,fc='white')
```

```
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
ax.axis('equal')
plt.tight_layout()
plt.title('Non-Fraud Transactions\n')
plt.show()
```

```
<Figure size 432x432 with 0 Axes>
```



- Inference - Card Companies have almost same distribution amongst Fraud and Non-Fraud transactions

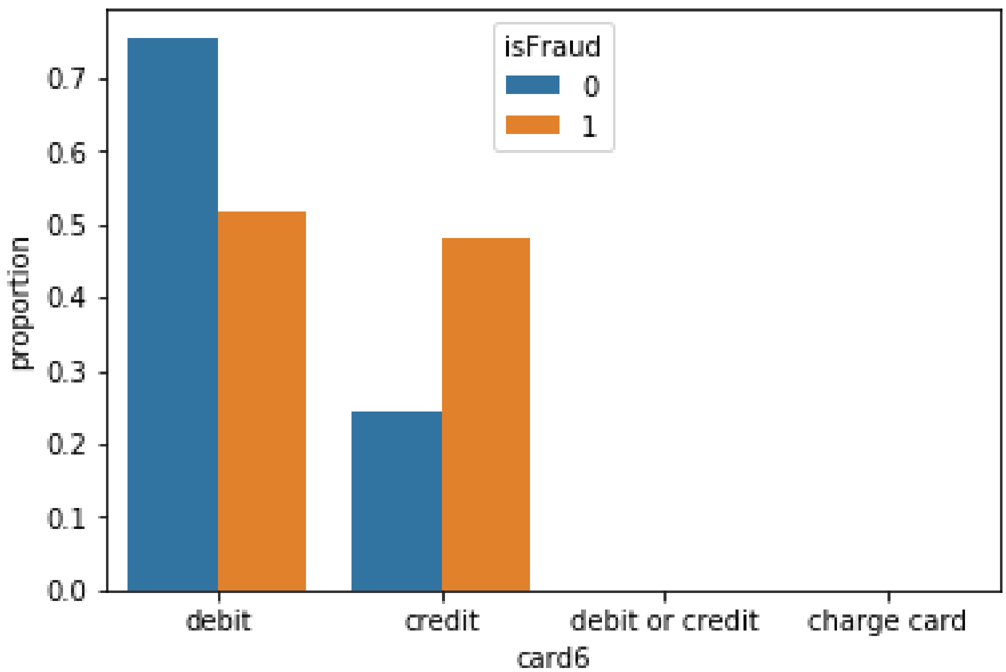
1.1.8 card6 (debit/credit)

[57]:
```
train_tot['card6'].value_counts()
```

[57]:
```
debit              439938
credit             148986
debit or credit        30
charge card            15
Name: card6, dtype: int64
```

[55]:
```
x, y, hue = "card6", "proportion", "isFraud"
hue_order = [0, 1]

(train_tot[x]
 .groupby(train_tot[hue])
 .value_counts(normalize=True)
 .rename(y)
 .reset_index()
 .pipe((sns.barplot, "data"), x=x, y=y, hue=hue))
```

[55]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93f79221d0>

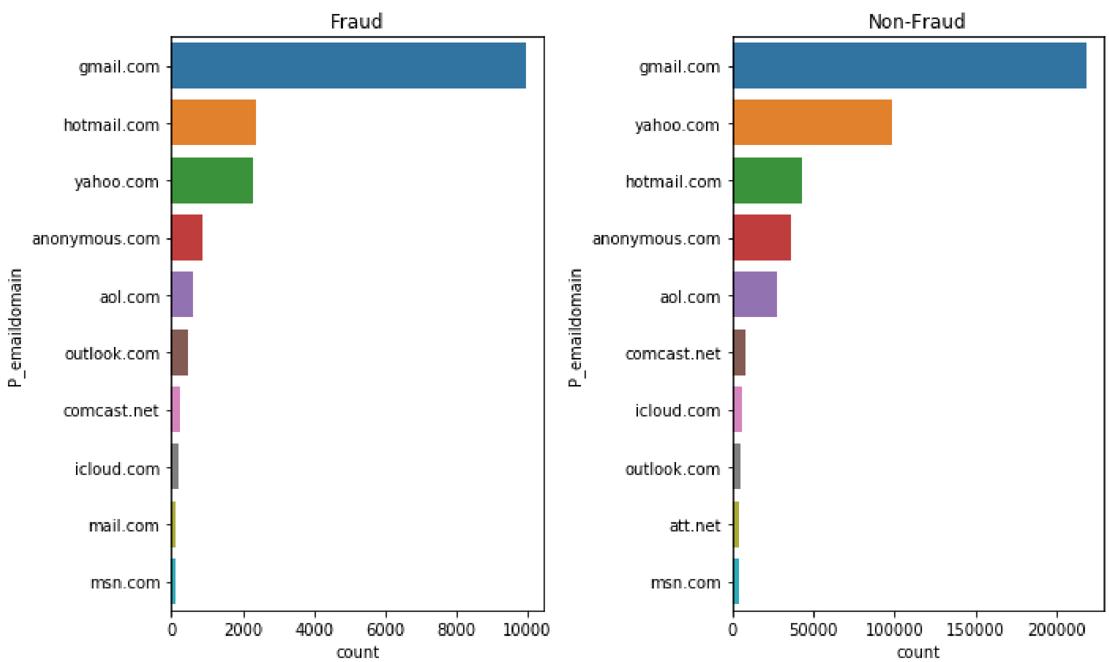


- Fraud Transactions have almost equal percentage of transactions from Credit and debit card.
- Charge Card transactions and 'debit or credit' are minimal

1.1.9 P_emaildomain (purchaser email)

[77]:
```
plt.figure(figsize=(10,6))
plt.subplot(1, 2, 1)
plt.title('Fraud')
sns.countplot(y='P_emaildomain',data=train_fraud,order=train_fraud.P_emaildomain.
 ↪value_counts().iloc[:10].index)

plt.subplot(1, 2, 2)
plt.title('Non-Fraud')
```

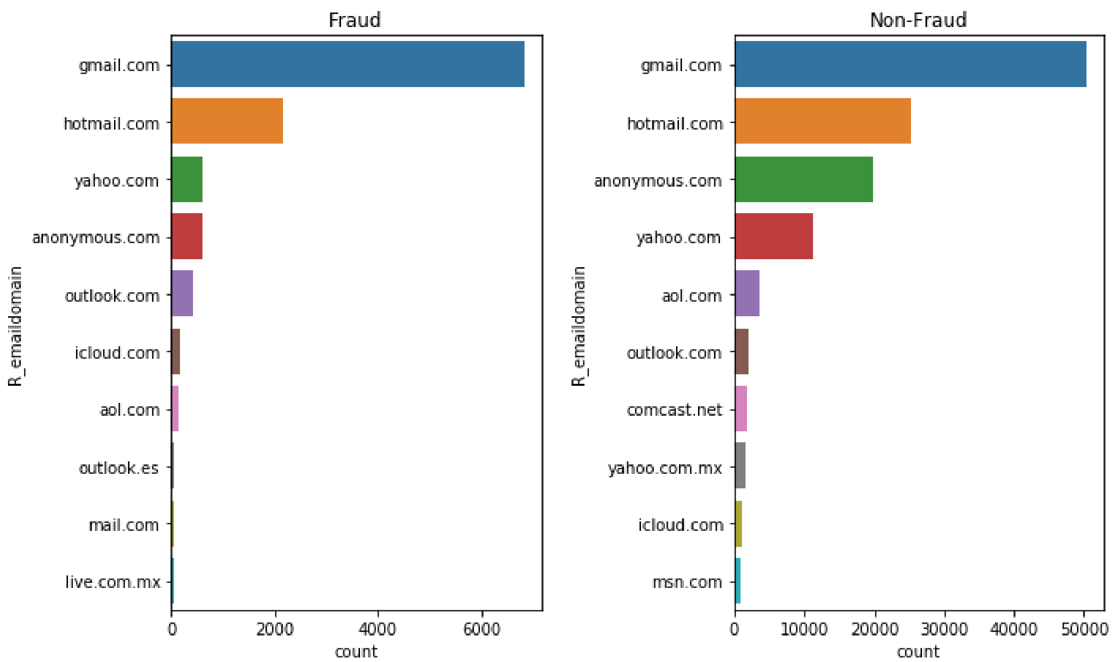
```
sns.countplot(y='P_emaildomain',data=train_no_fraud,order=train_no_fraud.
↪P_emaildomain.value_counts().iloc[:10].index)

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



- 'hotmail.com' domain is used more in Fraud cases compared to 'yahoo.com'. Otherwise general proportions are almost same in both cases

1.1.10 R_emaildomain (recipient email)

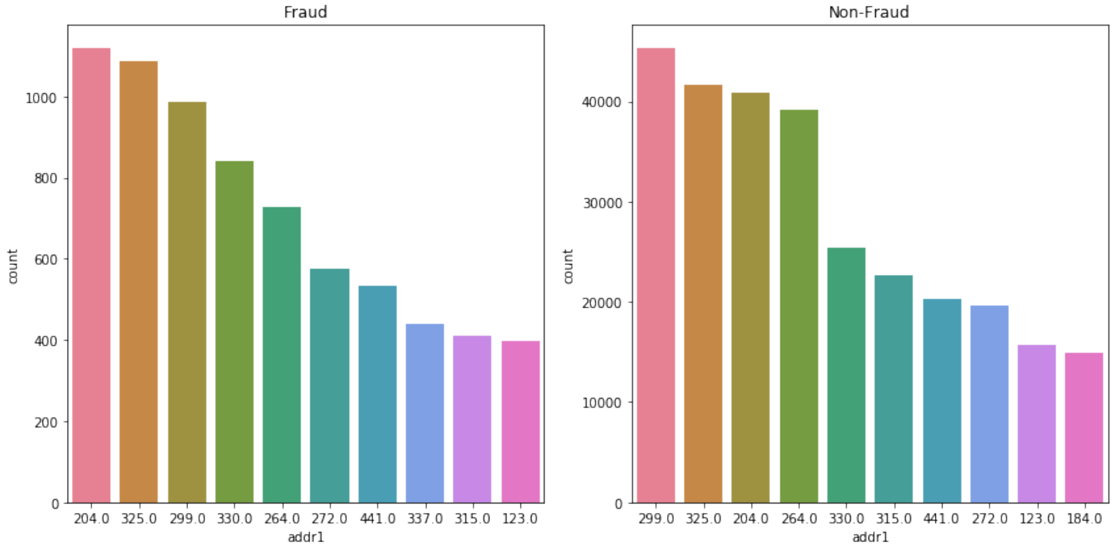
[78]:
```
plt.figure(figsize=(10,6))
plt.subplot(1, 2, 1)
plt.title('Fraud')
sns.countplot(y='R_emaildomain',data=train_fraud,order=train_fraud.R_emaildomain.
↪value_counts().iloc[:10].index)

plt.subplot(1, 2, 2)
plt.title('Non-Fraud')
sns.countplot(y='R_emaildomain',data=train_no_fraud,order=train_no_fraud.
↪R_emaildomain.value_counts().iloc[:10].index)

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



- 'hotmail.com' domain is used more in Fraud cases compared to 'anonymous.com' and 'yahoo.com'. Order of domains is not same in both cases.

1.1.11 addr1 (billing region)

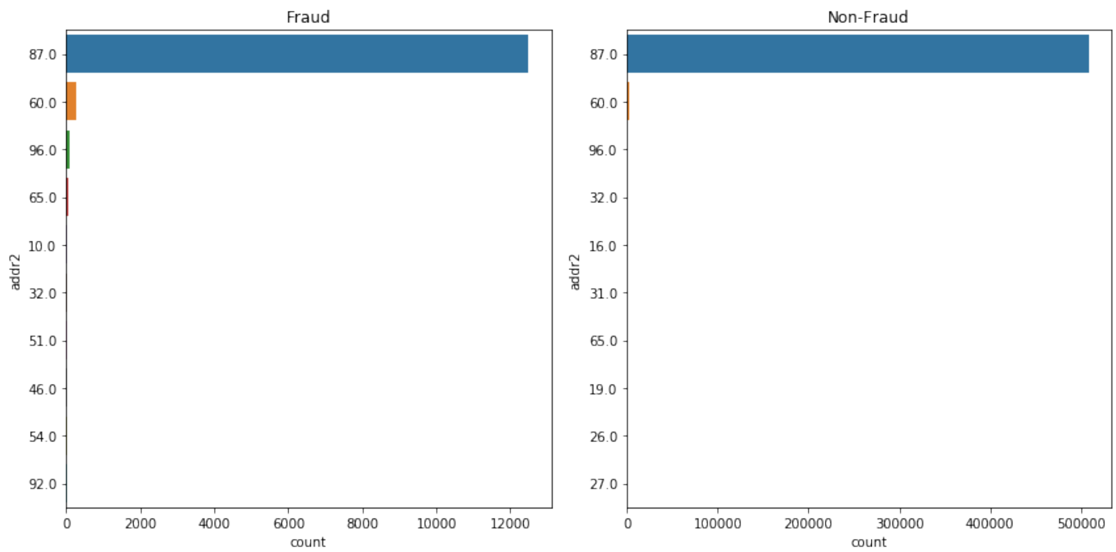
[97]:
```
x = 'addr1'
plt.figure(figsize=(12,6))
plt.subplot(1, 2, 1)
plt.title('Fraud')
sns.countplot(x=x,data=train_fraud,order=train_fraud[x].value_counts().iloc[:10].
 ↪index,palette=sns.color_palette("husl", 10))

plt.subplot(1, 2, 2)
plt.title('Non-Fraud')
sns.countplot(x=x,data=train_no_fraud,order=train_no_fraud[x].value_counts().
 ↪iloc[:10].index,palette=sns.color_palette("husl", 10))

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



- addr1 don't have similar value distribution among Fraud and Non-Fraud

1.1.12 addr2 (billing country)

[85]:
```
x = 'addr2'
plt.figure(figsize=(12,6))
plt.subplot(1, 2, 1)
plt.title('Fraud')
sns.countplot(y=x,data=train_fraud,order=train_fraud[x].value_counts().iloc[:10].
 ↪index)

plt.subplot(1, 2, 2)
plt.title('Non-Fraud')
sns.countplot(y=x,data=train_no_fraud,order=train_no_fraud[x].value_counts().
 ↪iloc[:10].index)

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



- Country code '87' is completely dominating the dataset

1.1.13 dist1

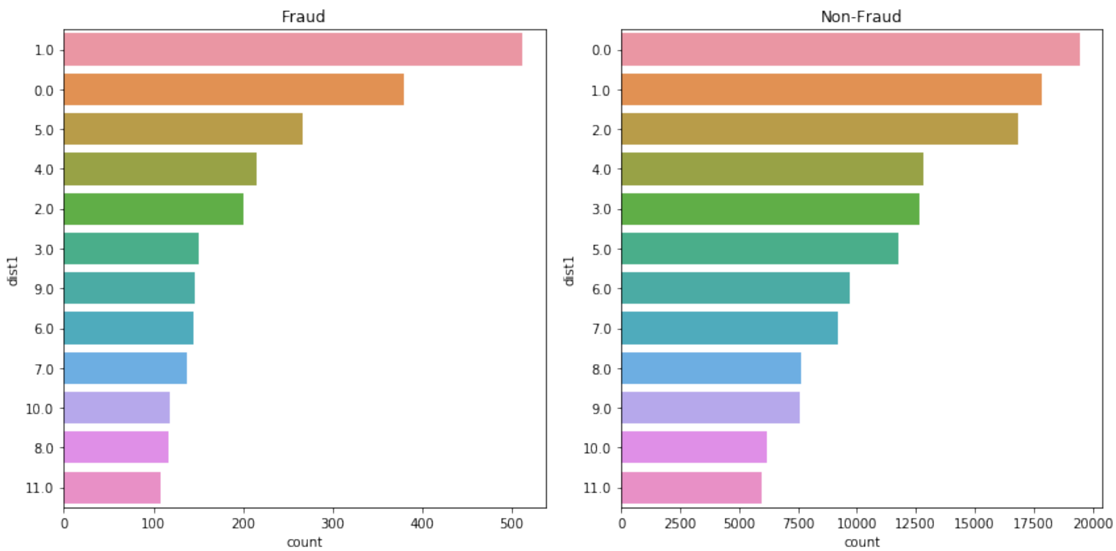
[101]:
```
s = StandardScaler()
train_fraud['scaled_dist1'] = s.fit_transform(train_fraud[['dist1']])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
```

[103]:
```
s = StandardScaler()
train_no_fraud['scaled_dist1'] = s.fit_transform(train_no_fraud[['dist1']])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
```

[110]:
```
x = 'dist1'
plt.figure(figsize=(12,6))
plt.subplot(1, 2, 1)
plt.title('Fraud')
sns.countplot(y=x,data=train_fraud,order=train_fraud[x].value_counts().iloc[:12].
 ↪index)

plt.subplot(1, 2, 2)
plt.title('Non-Fraud')
sns.countplot(y=x,data=train_no_fraud,order=train_no_fraud[x].value_counts().
 ↪iloc[:12].index)

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



- value 1 is more common in Fraud cases

1.1.14 dist2

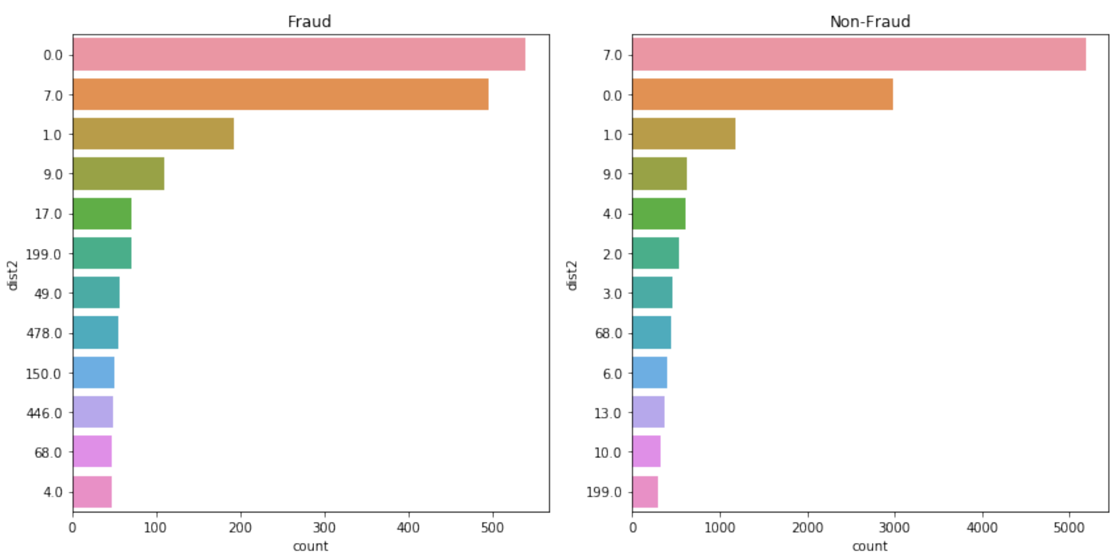
[104]:
```
s = StandardScaler()
train_fraud['scaled_dist2'] = s.fit_transform(train_fraud[['dist2']])

s = StandardScaler()
train_no_fraud['scaled_dist2'] = s.fit_transform(train_no_fraud[['dist2']])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """
```

[112]:

```
x = 'dist2'
plt.figure(figsize=(12,6))
plt.subplot(1, 2, 1)
plt.title('Fraud')
sns.countplot(y=x,data=train_fraud,order=train_fraud[x].value_counts().iloc[:12].
 ↪index)

plt.subplot(1, 2, 2)
plt.title('Non-Fraud')
sns.countplot(y=x,data=train_no_fraud,order=train_no_fraud[x].value_counts().
 ↪iloc[:12].index)

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



- Similar trend like dist1, here '0.0' is occuring more times in Fraud transactions than the most frequent value of Non-Fraud cases ('7.0')

1.2 Part 2 - Transaction Frequency

[0]:
```
train_tot[['addr2','TransactionDT']].groupby(by='addr2').count().reset_index().
↪sort_values(by='TransactionDT',ascending=False).head(10)
```

[0]:

	addr2	TransactionDT
62	87.0	520481
40	60.0	3084
68	96.0	638
20	32.0	91
44	65.0	82
4	16.0	55
19	31.0	47
7	19.0	33
14	26.0	25
15	27.0	20

- Therefore Max Count country Code is 87

[78]:
```
train_c = train_tot[train_tot['addr2'] == 87.0]
train_c.shape
```

[78]:
```
(520481, 20)
```

[85]:
```
train_c['TransactionDT'].describe()
```

[85]:
```
count    5.204810e+05
mean     7.391079e+06
std      4.629410e+06
min      8.640000e+04
25%      3.077831e+06
50%      7.321834e+06
75%      1.131429e+07
max      1.581113e+07
Name: TransactionDT, dtype: float64
```

- Since 86400 = 24*60*60 = seconds in a day

[79]:
```
# since 86400 = 24*60*60
train_c['hour'] = train_c['TransactionDT'].apply(lambda x: math.floor((x/
↪(60*60))%24) )
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

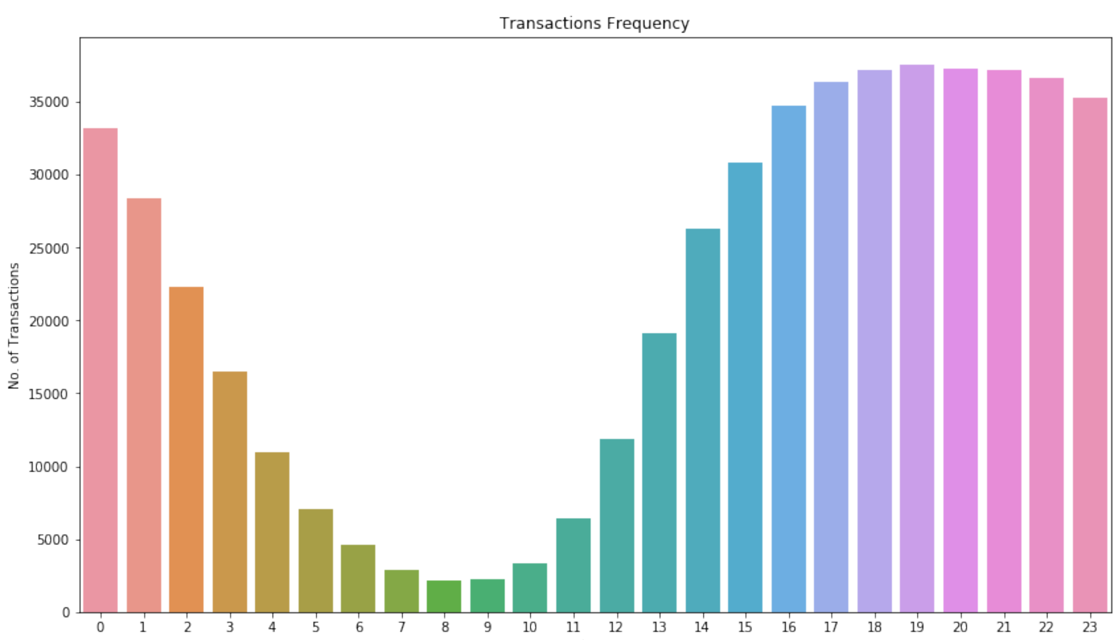
```
See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

[80]:
```python
train_c.head()
```

[80]:
```
   TransactionID  isFraud  TransactionDT  ...  minute second
ScaledTransactionAmt
0        2987000        0          86400  ...       0      0
0.002137
1        2987001        0          86401  ...       0      1
0.000900
2        2987002        0          86469  ...       1      9
0.001840
3        2987003        0          86499  ...       1     39
0.001558
4        2987004        0          86506  ...       1     46
0.001558

[5 rows x 20 columns]
```

[82]:
```python
plt.figure(figsize=(14,8))
gt = sns.countplot(train_c['hour'])
plt.title('Transactions Frequency')
plt.ylabel('No. of Transactions')
```

[82]:
```
Text(0, 0.5, 'No. of Transactions')
```



We can clearly see the **#Transactions dipping down from hour 6-10**. This signifies the sleeping hour of the Country '87'. As we do not know the reference date/time or the timezone of 'TransactionDT', I am assuming that generaly the #Transactions goes down during sleep hours of public. Therefore waking hours for Country '87' according to my calculations are from 11:00 hour till 05:00 hour

[0]:

```

```

1.3 Part 3 - Product Code Pending

[0]:

```
# Only 5 types of Product Code
train_tot.ProductCD.unique()
```

[0]:

```
array(['W', 'H', 'C', 'S', 'R'], dtype=object)
```

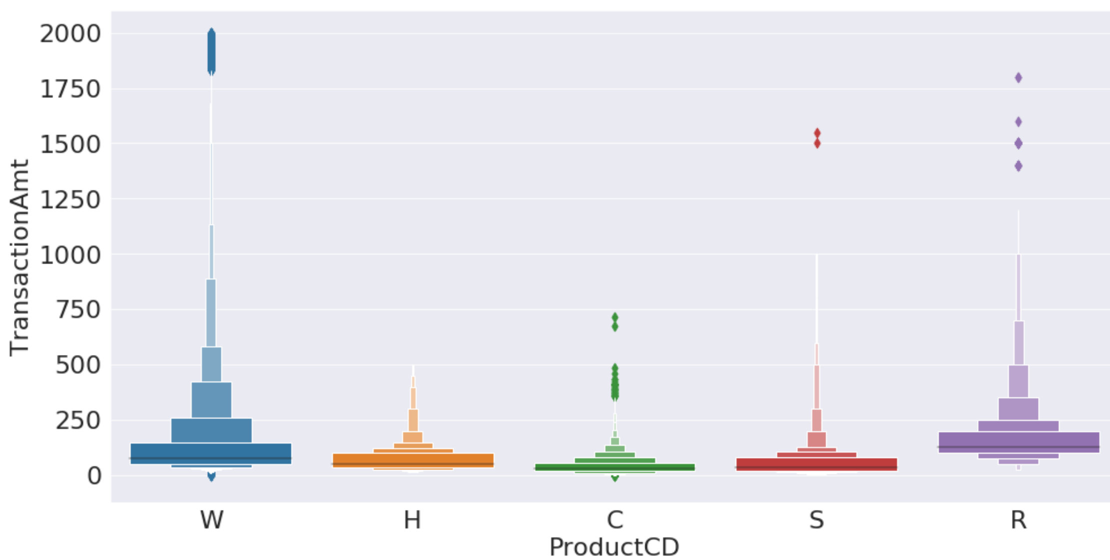
- Observing Transaction Amt distribution with each ProductCD value

[0]:

```
plt.figure(figsize=(16,8))
plt.rcParams.update({'font.size': 22})
sns.boxenplot(x='ProductCD', y='TransactionAmt',
              data=train_tot[train_tot['TransactionAmt'] <= 2000] )
```

[0]:

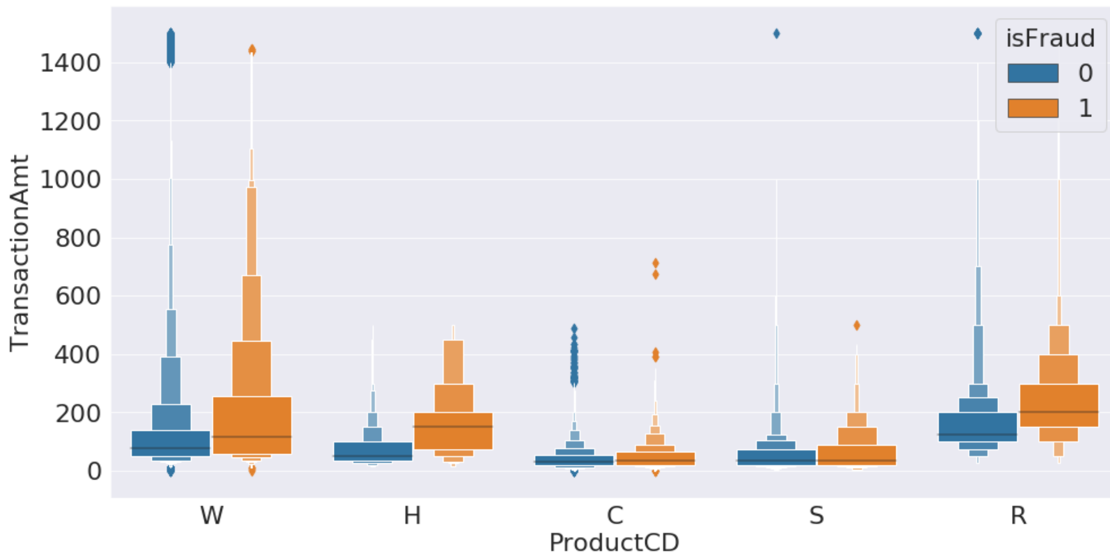
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f56353c3dd8>
```



[0]:

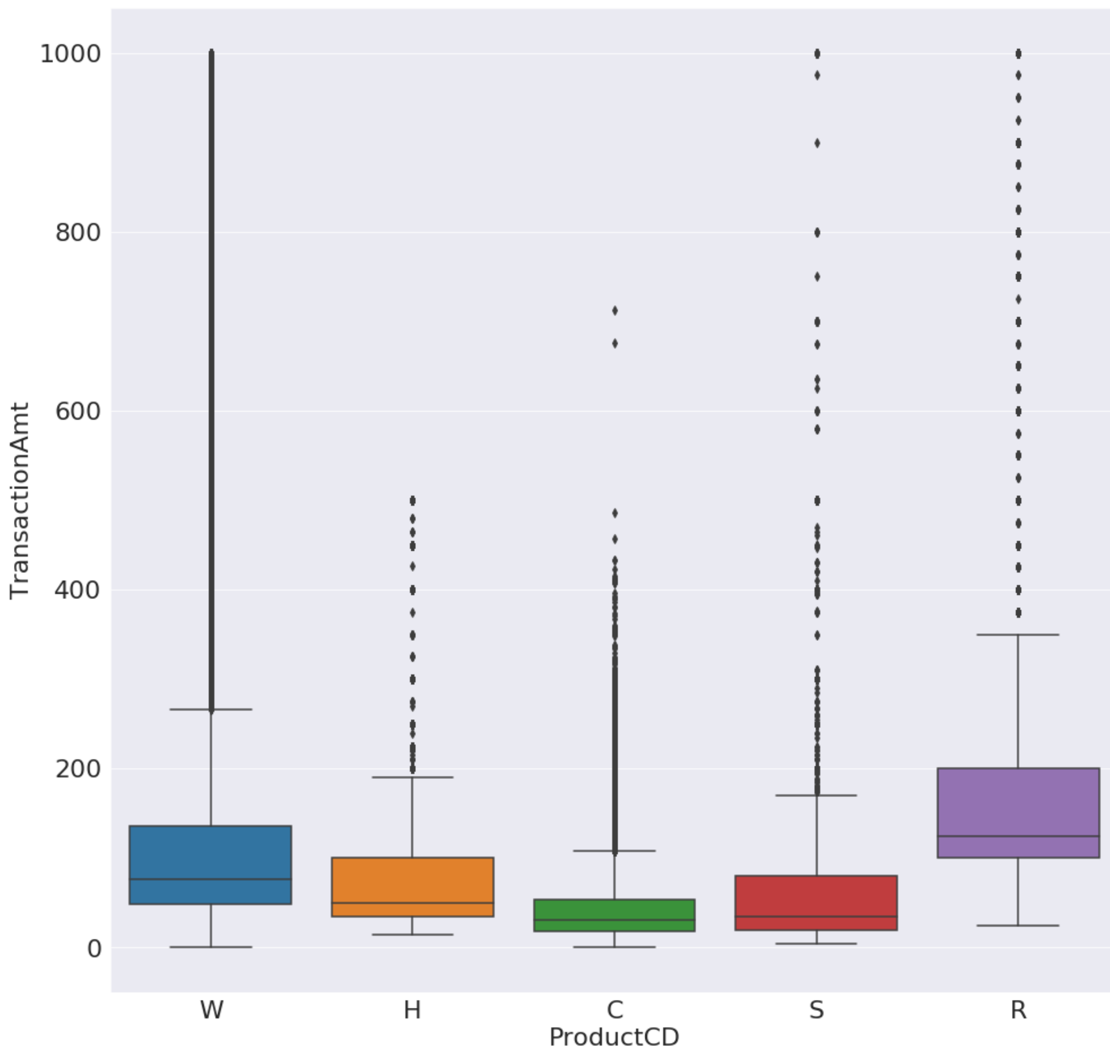
```
# Not needed
plt.figure(figsize=(16,8))
plt.rcParams.update({'font.size': 22})
sns.boxenplot(x='ProductCD', y='TransactionAmt',hue='isFraud',
              data=train_tot[train_tot['TransactionAmt'] <= 1500] )
```

[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f563561b0b8>




```
[0]: plt.figure(figsize=(16,16))
     plt.rcParams.update({'font.size': 22})
     sns.boxplot(x='ProductCD', y='TransactionAmt',
                 data=train_tot[train_tot['TransactionAmt'] <= 1000] )
```

[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f562a803e48>



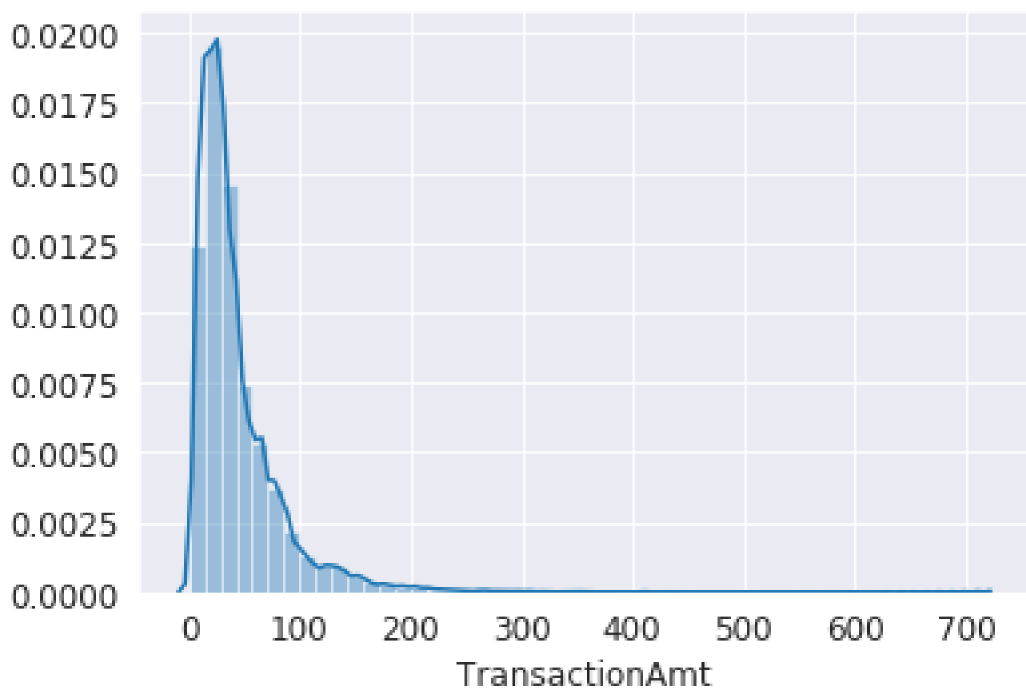
[0]:
```
gb = train_tot.groupby('ProductCD')
grps = [gb.get_group(x) for x in gb.groups]
```

[0]:
```
type(grps)
```

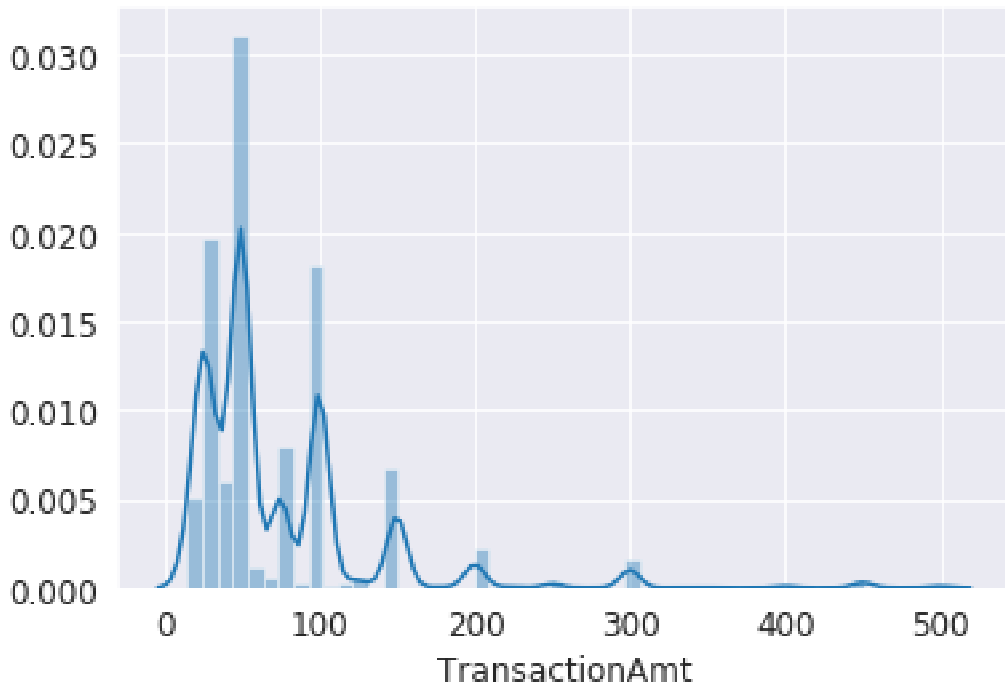
[0]:
```
list
```

[0]:
```
for grp in grps:
    print ("Product - ",grp['ProductCD'].iloc[0])
    print (grp['TransactionAmt'].describe())
    plt.rcParams.update({'font.size': 12})
    print ("\n\t\tProduct - ",grp['ProductCD'].iloc[0],end='')
    sns.distplot(grp[(grp['TransactionAmt'] <= 1500)]['TransactionAmt'],
                 label=str(grp['ProductCD'].iloc[0]),kde=True)
    plt.show()
    print('\n')
```

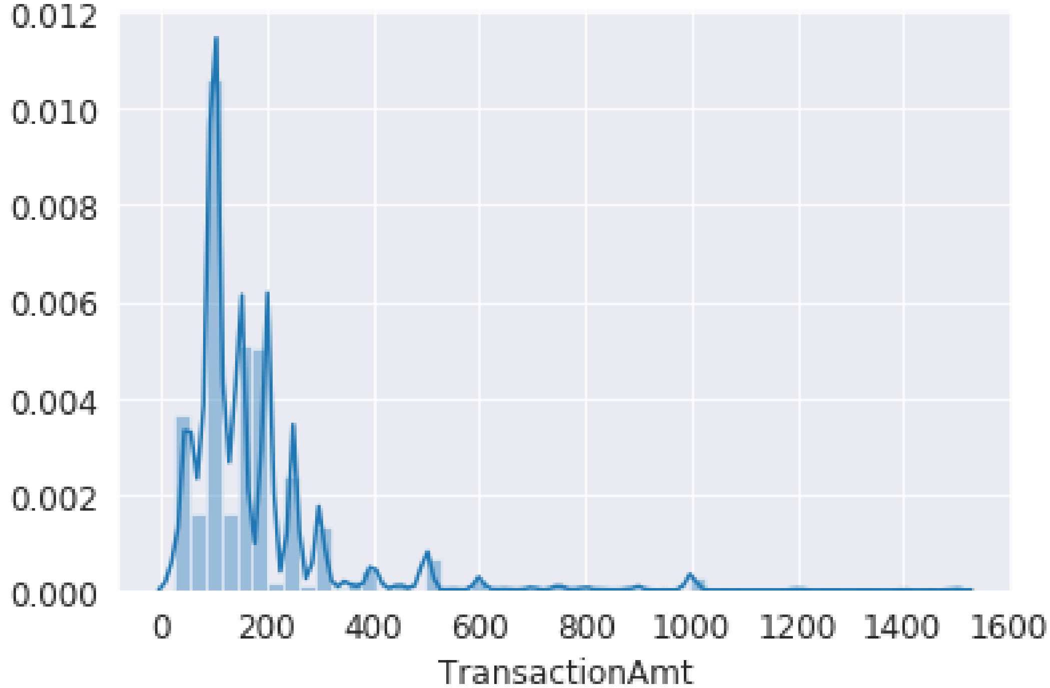
```
Product -  C
count    68519.000000
mean        42.872353
std         38.943070
min          0.251000
25%         18.423000
50%         31.191000
75%         54.102000
max        712.896000
Name: TransactionAmt, dtype: float64
```



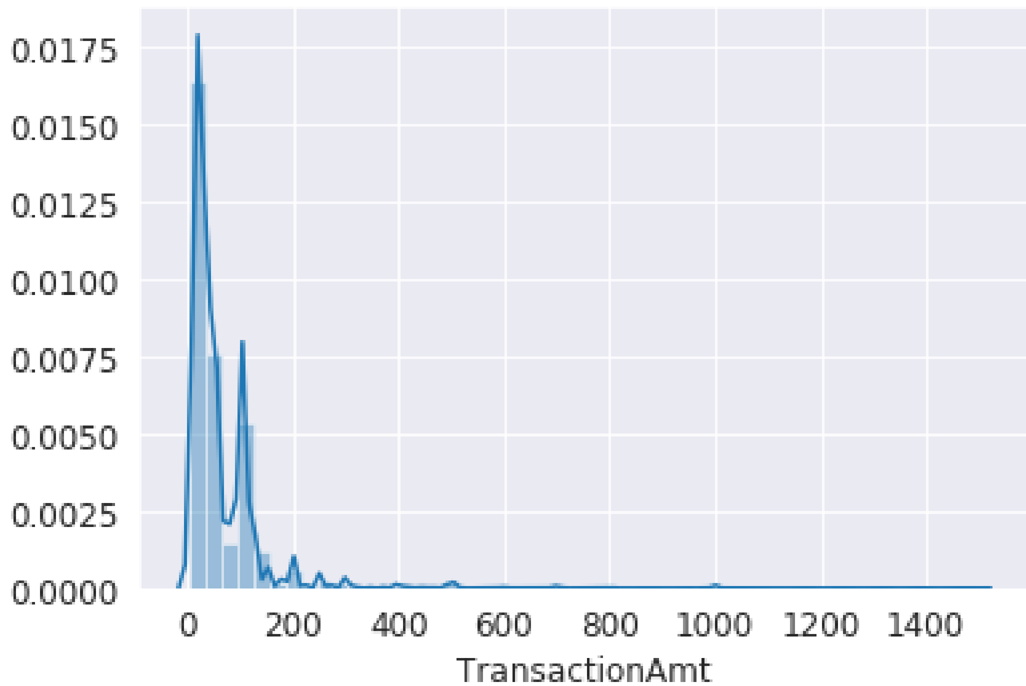

```
Product -  H
count    33024.000000
mean        73.170058
std         61.950955
min         15.000000
25%         35.000000
50%         50.000000
75%        100.000000
max        500.000000
Name: TransactionAmt, dtype: float64
```



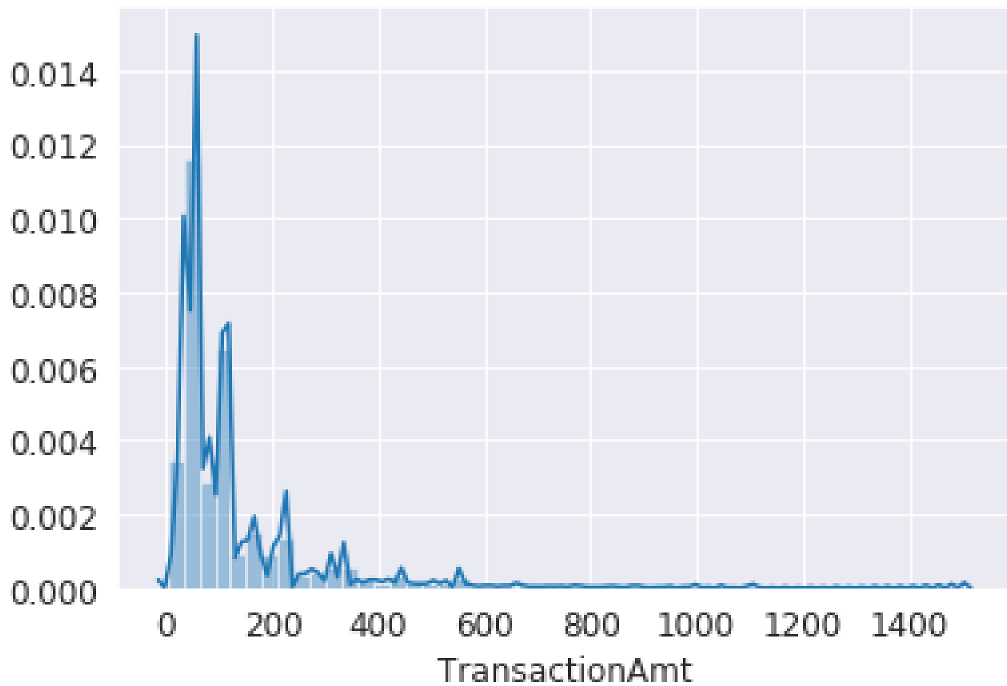

```
Product -  R
count    37699.000000
mean       168.306188
std        142.035568
min         25.000000
25%        100.000000
50%        125.000000
75%        200.000000
max       1800.000000
Name: TransactionAmt, dtype: float64
```




```
Product -  S
count    11628.000000
mean        60.269487
std         80.546775
min          5.000000
25%         20.000000
50%         35.000000
75%         80.000000
max       1550.000000
Name: TransactionAmt, dtype: float64
```

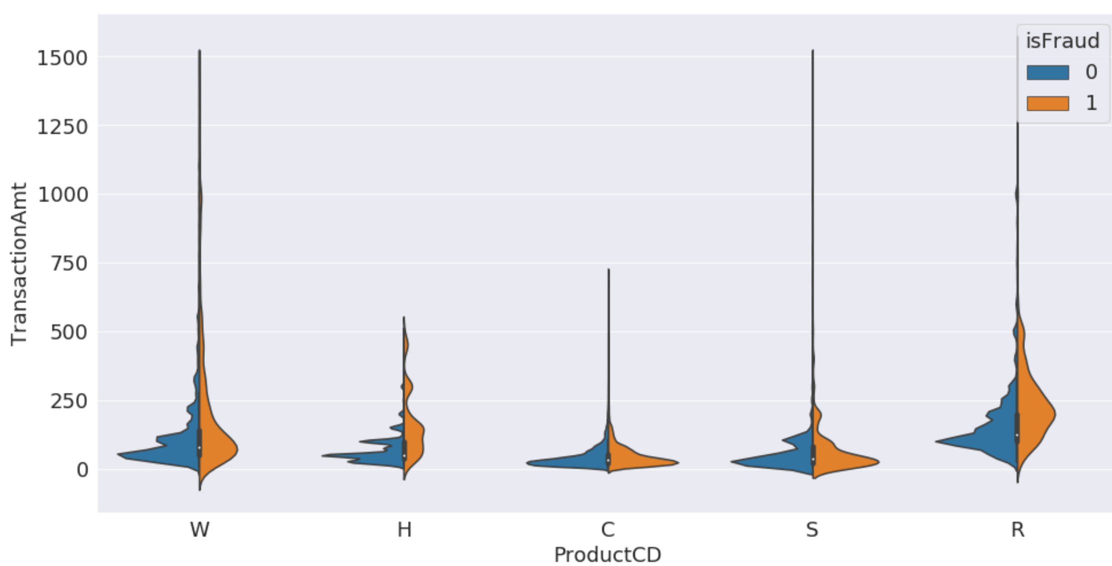



```
Product -  W
count    439670.000000
mean        153.158554
std         268.733692
min           1.000000
25%          49.000000
50%          78.500000
75%         146.000000
max       31937.391000
Name: TransactionAmt, dtype: float64
```



[0]:
```
plt.figure(figsize=(16,8))
plt.rcParams.update({'font.size': 18})
sns.violinplot(x='ProductCD', y='TransactionAmt',hue='isFraud',
              data=train_tot[train_tot['TransactionAmt'] <= 1500],split=True )
```

[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f562abb3be0>



- **Note** - We don't know the Transaction Qty. here, Transaction qty. will give a more clearer picture but this is the most close guess we can make.
- From the above Violin-plot and individual distribution plots, we can infer that Product **'C' is cheapest** because Distribution plot of C is having only one peak skewed to left side and the distribution in Fraud and on-Fraud cases are almost similar. If we see other Categories distribuition in Fraud cases from Violinplot, we notice that most of them have peaks on upper side compared to respective Non-Fraud distributions. Also Min. transaction Amt for 'C' is $0.25
- 'R' is most expensive since it's Min. transaction Amt is $25 and also the distribution of Fraud cases in 'R' category have a peak way above than Non-Fraud distribution
- Order from Cheapest to Expensive C < W < S < H < R

[0]:
```

```

1.4 Part 4 - Correlation Coefficient

[0]:
```
train_tot['day'] = train_tot['TransactionDT'].apply(lambda x: math.floor((x/
↪(60*60*24))) )
```

[0]:
```
train_tot['hour'] = train_tot['TransactionDT'].apply(lambda x: math.floor((x/
↪(60*60))%24) )
```

[0]:
```
train_tot['minute'] = train_tot['TransactionDT'].apply(lambda x: math.floor((x/
↪(60))%60) )
```

[0]:
```
train_tot['second'] = train_tot['TransactionDT'].apply(lambda x: math.floor(x%60␣
↪))
```

[21]:
```
train_tot.head(5)
```

[21]:
```
   TransactionID  isFraud  TransactionDT  ...  hour  minute  second
0        2987000        0          86400  ...     0       0       0
1        2987001        0          86401  ...     0       0       1
2        2987002        0          86469  ...     0       1       9
3        2987003        0          86499  ...     0       1      39
4        2987004        0          86506  ...     0       1      46

[5 rows x 19 columns]
```

[0]:
```
scaler = MinMaxScaler()
train_tot['ScaledTransactionAmt'] = scaler.
↪fit_transform(train_tot[['TransactionAmt']])
```

[23]:
```
train_tot[['day','hour','minute','second','ScaledTransactionAmt','TransactionAmt']].
↪describe()
```

[23]:
```
                 day          hour  ...  ScaledTransactionAmt  TransactionAmt
count  590540.000000  590540.000000  ...         590540.000000   590540.000000
mean       84.729199      13.861923  ...              0.004220      135.027176
std        53.437277       7.607152  ...              0.007489      239.162522
min         1.000000       0.000000  ...              0.000000        0.251000
25%        35.000000       6.000000  ...              0.001349       43.321000
50%        84.000000      16.000000  ...              0.002145       68.769000
75%       130.000000      20.000000  ...              0.003906      125.000000
max       182.000000      23.000000  ...              1.000000    31937.391000

[8 rows x 6 columns]
```

[0]:
```python
## TODO Shift Hours according to Time Zone
train_tot['']
```

- Summing the Transaction Amt by hour of day

[0]:
```python
train_hour = train_tot[['hour','TransactionAmt']].groupby(by=['hour']).sum().
↪reset_index()
```

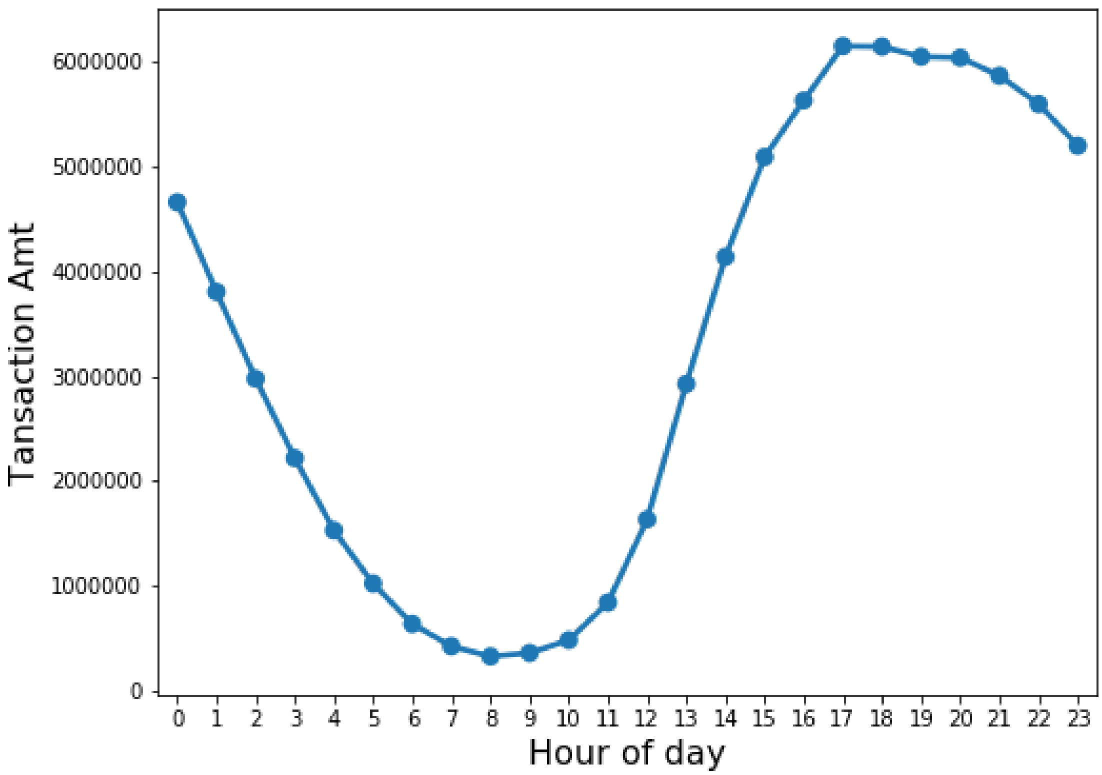
[25]:
```python
train_hour.head()
```

[25]:
```
   hour  TransactionAmt
0     0    4.660496e+06
1     1    3.805385e+06
2     2    2.976132e+06
3     3    2.217529e+06
4     4    1.527839e+06
```

[0]:
```python
train_minute = train_tot[['minute','TransactionAmt']].groupby(by=['minute']).
↪sum().reset_index()
```

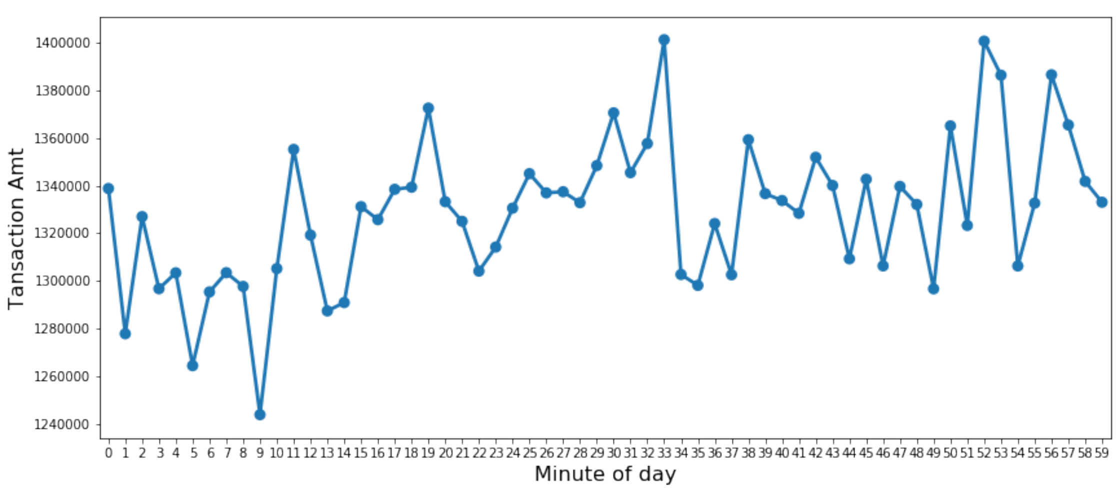
[30]:
```python
plt.figure(figsize=(8,6))
sns.pointplot(x='hour',y='TransactionAmt',data=train_hour)
plt.xlabel('Hour of day', fontsize=16)
plt.ylabel('Tansaction Amt', fontsize=16)
```

[30]: Text(0, 0.5, 'Tansaction Amt')



[41]:
```
plt.figure(figsize=(14,6))
sns.pointplot(x='minute',y='TransactionAmt',data=train_minute)
plt.xlabel('Minute of day', fontsize=16)
plt.ylabel('Tansaction Amt', fontsize=16)
```

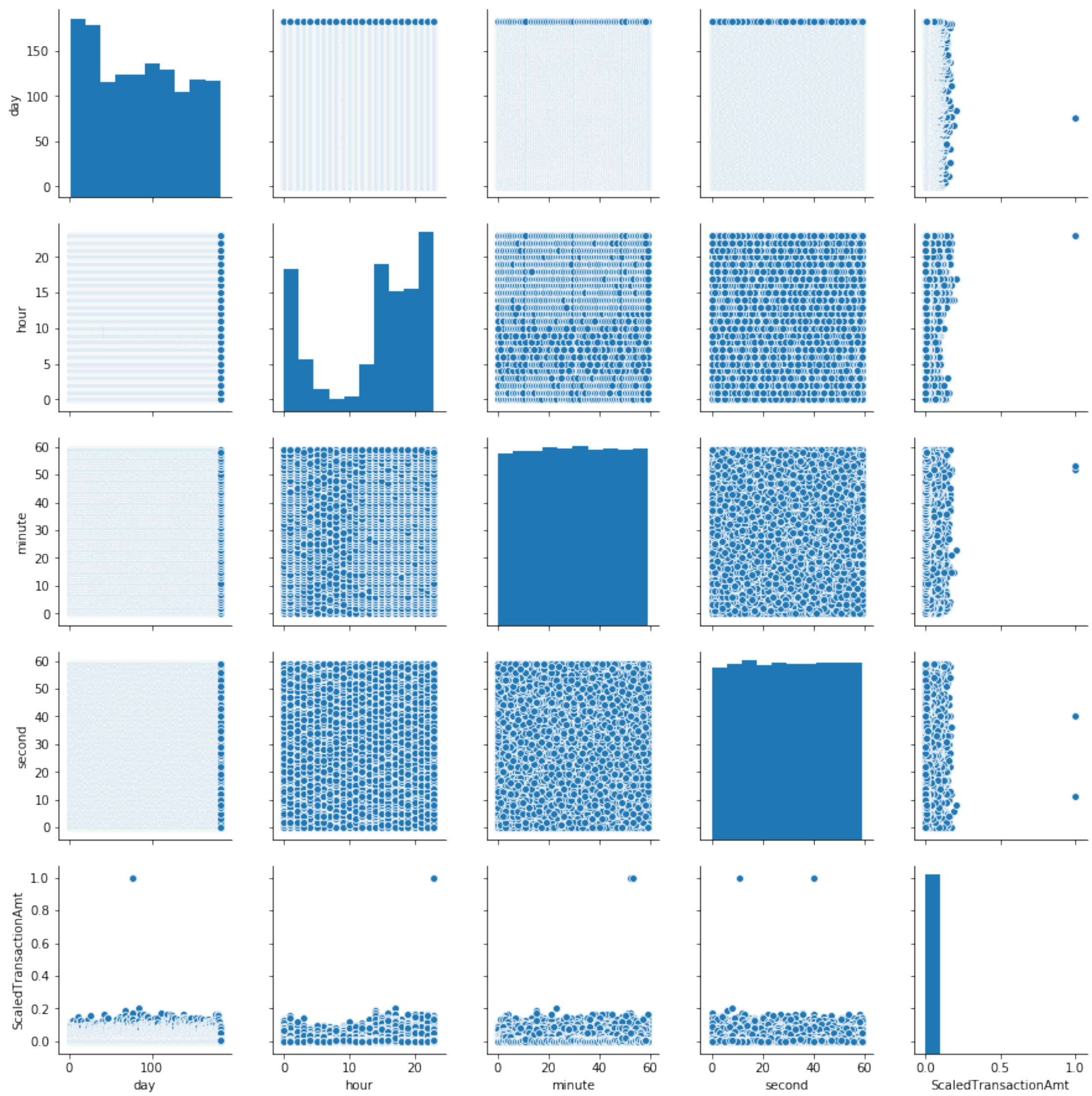
[41]:
```
Text(0, 0.5, 'Tansaction Amt')
```



[0]:
```
# plt.figure(figsize=(16,10))
# sns.boxplot(x='hour',y='TransactionAmt',data=train_hour)
# plt.xlabel('Hour of day', fontsize=16)
# plt.ylabel('Tansaction Amt', fontsize=16)
```

[43]:
```
sns.
 ↪pairplot(train_tot[['day','hour','minute','second','ScaledTransactionAmt']],palette="husl")
```

[43]:
```
<seaborn.axisgrid.PairGrid at 0x7f6e31d00898>
```



[44]:
```
sns.
 ↪pairplot(train_tot[['day','hour','minute','second','ScaledTransactionAmt','isFraud']],hue="is
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:487:
RuntimeWarning: invalid value encountered in true_divide
```

```
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kdetools.py:34:
RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

[44]:
```
<seaborn.axisgrid.PairGrid at 0x7f6e2fb209b0>
```

[45]:
```python
print (train_hour['TransactionAmt'].corr(train_hour['hour']))

print (train_minute['TransactionAmt'].corr(train_minute['minute']))
```

```
0.6421174943084444
0.46884567892333934
```

[0]:
```python
# print (train_tot['TransactionAmt'].corr(train_tot['hour']))

# print (train_tot['TransactionAmt'].corr(train_tot['minute'])

# print (train_tot['TransactionAmt'].corr(train_tot['second'])

# train_tot['TransactionAmt'].corr(train_tot['hour'])
```

This is Pearson Correlation value * Correlation b/w hour and Transaction Amt = 0.642117

- Correlation b/w minute and Transaction Amt = 0.46884

1.4.1 Taking 4 top countries to see their Transaction Amt correlation with Hour of day

[0]:
```python
train_c = train_tot[['addr2','isFraud']].groupby(by=['addr2']).count().
    ↪reset_index().sort_values(by='isFraud',ascending=False)
```

[55]:
```python
train_c.head(10)
```

[55]:

	addr2	isFraud
62	87.0	520481
40	60.0	3084
68	96.0	638
20	32.0	91
44	65.0	82
4	16.0	55
19	31.0	47
7	19.0	33
14	26.0	25
15	27.0	20

[0]:
```python
train_87 = train_tot[train_tot['addr2'] == 87][['hour','TransactionAmt']].
    ↪groupby(by=['hour']).sum().reset_index()
```

[68]:
```python
train_87.shape
```

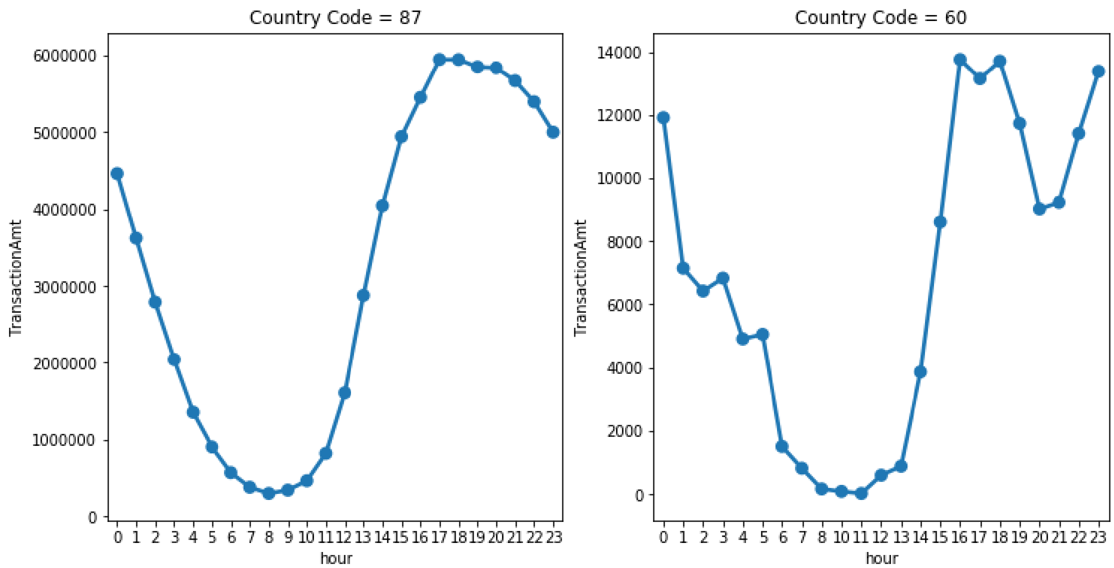
[68]:
```
(24, 2)
```

[0]:
```python
train_60 = train_tot[train_tot['addr2'] == 60][['hour','TransactionAmt']].
    ↪groupby(by=['hour']).sum().reset_index()
train_96 = train_tot[train_tot['addr2'] == 96][['hour','TransactionAmt']].
    ↪groupby(by=['hour']).sum().reset_index()
train_32 = train_tot[train_tot['addr2'] == 32][['hour','TransactionAmt']].
    ↪groupby(by=['hour']).sum().reset_index()
```

[77]:
```python
plt.figure(figsize=(12,6))

plt.subplot(1, 2, 1)
plt.title('Country Code = 87')
# plt.figure(figsize=(8,6))
sns.pointplot(x='hour',y='TransactionAmt',data=train_87)
```

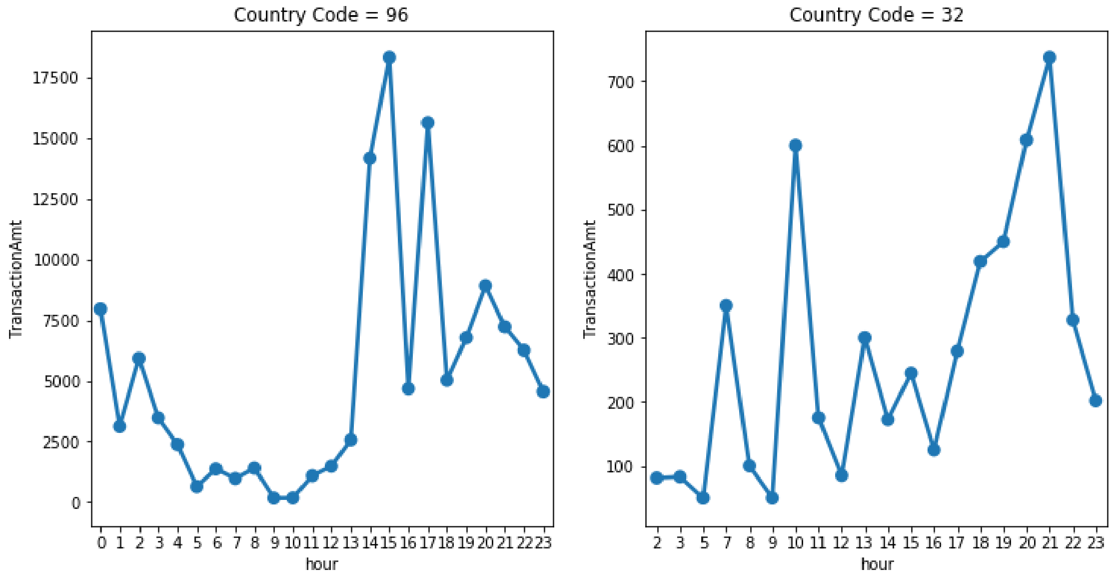
```
plt.subplot(1, 2, 2)
plt.title('Country Code = 60')
# plt.figure(figsize=(8,6))
sns.pointplot(x='hour',y='TransactionAmt',data=train_60)
```

[77]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6e2e1aae80>



[74]:
```
plt.figure(figsize=(12,6))

plt.subplot(1, 2, 1)
plt.title('Country Code = 96')
sns.pointplot(x='hour',y='TransactionAmt',data=train_96)

plt.subplot(1, 2, 2)
plt.title('Country Code = 32')
sns.pointplot(x='hour',y='TransactionAmt',data=train_32)
```

[74]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6e2ecb2940>



[76]:
```
print ('Correlation for Country 87 -', train_87['TransactionAmt'].
    ↪corr(train_87['hour']))
print ('Correlation for Country 60 -', train_60['TransactionAmt'].
    ↪corr(train_60['hour']))
print ('Correlation for Country 96 -', train_96['TransactionAmt'].
    ↪corr(train_96['hour']))
print ('Correlation for Country 32 -', train_32['TransactionAmt'].
    ↪corr(train_32['hour']))
```

```
Correlation for Country 87 - 0.65110479046039
Correlation for Country 60 - 0.45345737174136297
Correlation for Country 96 - 0.3815415447144118
Correlation for Country 32 - 0.5559657154017995
```

1.5 Part 5 - Interesting Plot

Reference - https://www.kaggle.com/c/ieee-fraud-detection/discussion/100400#latest-579480

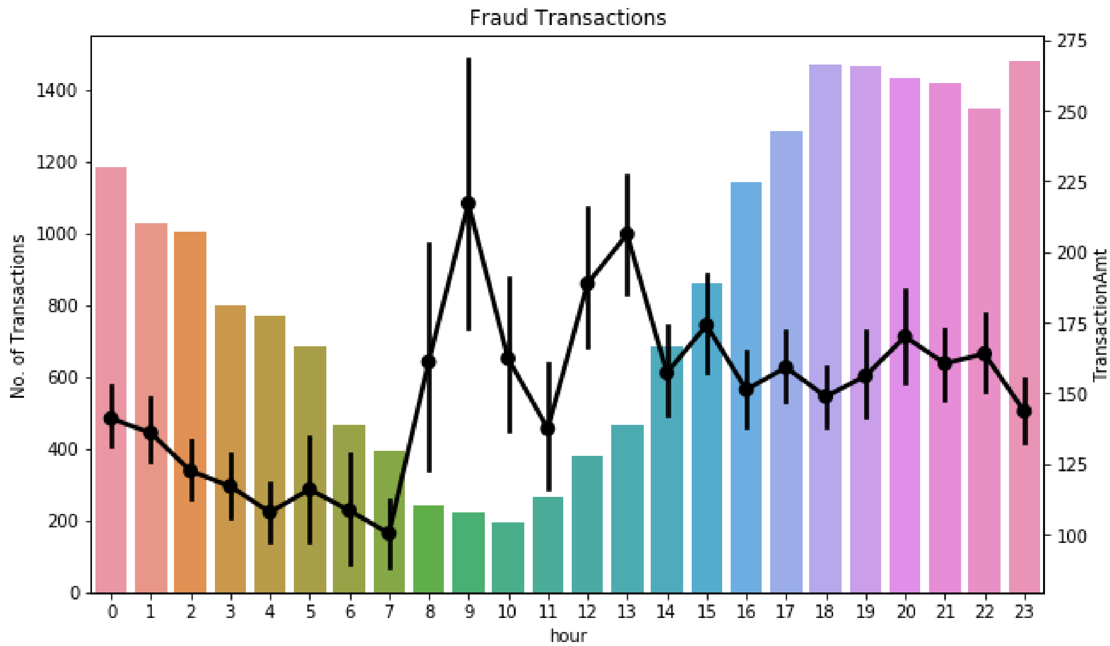
[0]:
```
train_fraudi = train_tot[train_tot['isFraud'] == 1]
```

[0]:
```
train_no_fraudi = train_tot[train_tot['isFraud'] == 0]
gc.collect()
```

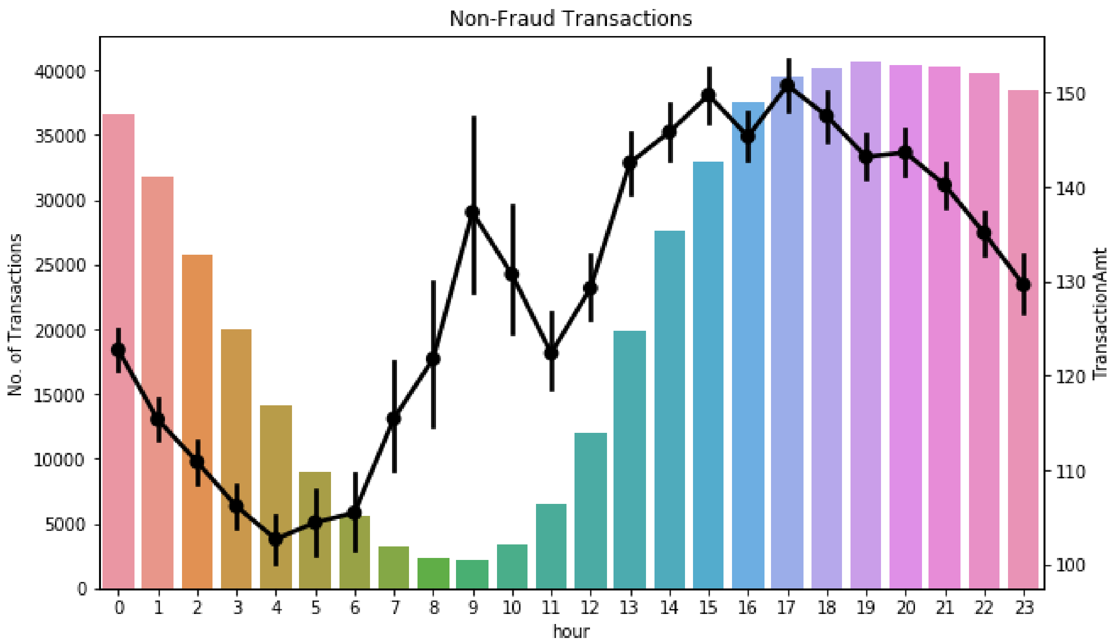
[0]: 32

[0]:
```
plt.figure(figsize=(10,6))
gt = sns.countplot(train_fraudi['hour'])
plt.title('Fraud Transactions')
plt.ylabel('No. of Transactions')
```

```
g1 = gt.twinx()

g1 = sns.pointplot(x='hour', y='TransactionAmt', data=train_fraudi,␣
↪color='black', legend=True)
```



[0]:
```
plt.figure(figsize=(10,6))
gt = sns.countplot(train_no_fraudi['hour'])
plt.title('Non-Fraud Transactions')
plt.ylabel('No. of Transactions')

g1 = gt.twinx()

g1 = sns.pointplot(x='hour', y='TransactionAmt', data=train_no_fraudi,␣
↪color='black', legend=True)
```



Interesting insight - From the above two graphs of Fraud and Non-Fraud transactions we notice that **during the sleeping hours of people** when the no. of transactions goes down than the **Avg. amount of transaction is higher in Fraud cases than in Non-Fraud ones**

This signifies that the fraudsters are carefully doing the high value transactions during sleep time so that people do not get immediate mobile/email alerts of the transaction amount.

1.6 Part 6 - Prediction Model

- Since data had significant amount of Missing values, I replaced them with a particular number -1111, and specifically provided this value to the corresponding algorithm model so that it can utilise this information for classification.
- Basic Feature Engineering - Converted all categorical features from train and test to Label Encoded vectors because of memory constraints on my laptop. Total **31 Columns are with Categorical values with Total of 5508 Unique columns possible.**
- Worked with two ensemble models - 1) Random Forest 2) XgBoost
- Random Forest worked pretty good giving the AUC ROC score of - 0.9140
- XgBoost significant improved this score to - 0.9376
- Both Random Forest and XgBoost are ensemble methods of Decision trees , which works quite good compared to other traditional methods like Logistic Regression and SVM in tabular data and Classification objective.

[0]:
```
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn import preprocessing
import gc
```

[10]:
```
gc.collect()
```

[10]: 0

[0]:
```
train_id = pd.read_csv('./ieee-fraud-detection/train_identity.
↪csv')#,usecols=cols_id)
train_txn = pd.read_csv('./ieee-fraud-detection/train_transaction.csv')
```

[0]:
```
test_id = pd.read_csv(data_dir+'./ieee-fraud-detection/test_identity.
↪csv')#,usecols=cols_id)
test_txn = pd.read_csv(data_dir+'./ieee-fraud-detection/test_transaction.csv')
```

[0]:
```
sample_submission = pd.read_csv(data_dir+
                                './ieee-fraud-detection/sample_submission.csv')
```

[0]:
```
train = train_txn.merge(train_id, how='left',on='TransactionID')
test = test_txn.merge(test_id, how='left', on='TransactionID')
gc.collect()

y_train = train['isFraud'].copy()
```

[16]:
```
X_train = train.drop('isFraud', axis=1)
X_test = test

del train
gc.collect()
```

[16]: 7

[0]:
```
X_train = X_train.fillna(-1111)
X_test = X_test.fillna(-1111)
```

[18]:
```
cnt=0
total_unique = 0
for f in X_train.columns:
    if X_train[f].dtype=='object' or X_test[f].dtype=='object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(X_train[f].values) + list(X_test[f].values))

        print (f,' - ',(X_train[f].nunique()+X_test[f].nunique()))
        cnt+=1
        total_unique+=(X_train[f].nunique()+X_test[f].nunique())
        X_train[f] = lbl.transform(list(X_train[f].values))
        X_test[f] = lbl.transform(list(X_test[f].values))

print ("Columns with Categorical values %f\nTotal Unique columns possible %f"%␣
↪(cnt, total_unique))
```

```
ProductCD  -  10
card4  -  10
card6  -  9
P_emaildomain  -  121
R_emaildomain  -  122
```

```
M1  -  6
M2  -  6
M3  -  6
M4  -  8
M5  -  6
M6  -  6
M7  -  6
M8  -  6
M9  -  6
id_12  -  6
id_15  -  8
id_16  -  6
id_23  -  8
id_27  -  6
id_28  -  6
id_29  -  6
id_30  -  163
id_31  -  267
id_33  -  652
id_34  -  8
id_35  -  6
id_36  -  6
id_37  -  6
id_38  -  6
DeviceType  -  6
DeviceInfo  -  4014
Columns with Categorical values 31.000000
Total Unique columns possible 5508.000000
```

1.6.1 Random Forest

[0]:
```
rf = RandomForestClassifier(max_depth  = 33,
                            max_features = 30,
                            n_estimators =300, n_jobs=-1,
                            min_samples_leaf=200)
```

[33]:
```
rf.fit(X_train.head(10000), y_train.head(10000))
```

[33]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=33, max_features=30, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=200, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=300,
                       n_jobs=-1, oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

[0]:
```
sample_submission['isFraud'] = rf.predict_proba(X_test)
```

[0]:
```
sample_submission.to_csv('rf_simple.csv',index=False)
```

[0]:

```

```

1.6.2 XgBoost

[0]:

```
xgb_clf = xgb.XGBClassifier(
    n_estimators=510,
    max_depth=8,
    learning_rate=0.05,
    subsample=0.91,
    colsample_bytree=0.89,
    missing=-1111,
    random_state=2020,
    tree_method='hist'
)
```

[23]:

```
xgb_clf.fit(X_train.head(1000), y_train.head(1000))
```

[23]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.89, gamma=0,
              learning_rate=0.05, max_delta_step=0, max_depth=8,
              min_child_weight=1, missing=-1111, n_estimators=510, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=2020,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=0.91, tree_method='hist', verbosity=1)
```

[0]:

```
sample_submission['isFraud'] = xgb_clf.predict_proba(X_test)[:,1]
```

[25]:

```
sample_submission.head()
```

[25]:

	TransactionID	isFraud
0	3663549	0.000428
1	3663550	0.000603
2	3663551	0.006466
3	3663552	0.000313
4	3663553	0.000767

[0]:

```
sample_submission.to_csv('xgboost_simple.csv',index=False)
```

1.7 Part 7 - Final Result

Kaggle Link: https://www.kaggle.com/adich23
Highest Rank: 3429
Score: 0.9376
Number of entries: 11

[86]:

```
from IPython.display import Image
Image(filename='../content/drive/My Drive/data/Kaggle_rank.png')
```

[86]:

3428	**SG1**	0.9376	9	1mo
3429	**Aditya Choudhary**	0.9376	11	15h

1.8 References

- stackoverflow - https://stackoverflow.com/questions/35692781/python-plotting-percentage-in-seaborn-bar-plot, https://stackoverflow.com/questions/11854847/how-can-i-display-an-image-from-a-file-in-jupyter-notebook
- pandas-https://pandas.pydata.org/pandas-docs/stable/
- scikit-learn - https://scikit-learn.org/stable/
- xgboost - https://xgboost.readthedocs.io/en/latest/python/index.html
- plot pie code - https://medium.com/@kvnamipara/a-better-visualisation-of-pie-charts-by-matplotlib-935b7667d77f

[0]:

```
```