Importing the necessary libraries

In [202]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import KNNImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import mutual_info_classif
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import *
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from category_encoders import CountEncoder
import category_encoders
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
```

Reading the data

In [4]:
```
transaction_data = pd.read_csv('sample_transaction_data.csv')
transaction_data.dtypes
```

Out[4]:
```
transaction_id         object
targets                 int64
transaction_date       object
account_open_date      object
transaction_amount    float64
                       ...
col_111                  bool
col_112               float64
col_113               float64
col_114                 int64
col_115               float64
Length: 122, dtype: object
```

EDA

In [311]:
```
transaction_data.head()
```

Out[311]:

	transaction_id	targets	transaction_date	account_open_date	transaction_amount	beneficiary	col_0	col_1	col_2	col_3	...	col_106	col_107	col_108	col_
0	TRX00000000	1	2021-10-03	2021-06-28	52092.586207	Manny's Auto Parts	0.0	False	25.0	0	...	0	1	False	
1	TRX00000001	0	2021-10-03	2021-05-16	50042.970326	Zach's Agriculture	0.0	False	25.0	0	...	0	1	False	
2	TRX00000002	0	2021-10-03	2021-04-19	54255.114574	Fiona's Technical Services	0.0	False	25.0	0	...	0	1	False	
3	TRX00000003	0	2021-10-03	2021-04-13	61722.527737	Omar's Exteriors	0.0	True	25.0	0	...	0	1	False	
4	TRX00000004	0	2021-10-03	2021-03-15	54313.312765	Steve's Utilities	0.0	True	25.0	0	...	98	1	False	

5 rows × 122 columns

In [57]:
```
transaction_data.describe()
```

Out[57]:

	targets	transaction_amount	col_0	col_3	col_4	col_8	col_9	col_10	col_13	col_14	...	c
count	15781.000000	1.578100e+04	12186.000000	15781.000000	15781.000000	15781.000000	10898.000000	15781.000000	15781.0	9740.000000	...	5485.
mean	0.047589	1.101420e+05	0.004185	0.266967	2.487358	0.669413	17.052028	334.344718	0.0	0.027002	...	0.
std	0.212901	3.396154e+05	0.064560	1.858497	47.191492	20.223097	180.610973	728.378619	0.0	0.162098	...	0.
min	0.000000	5.000317e+04	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	...	0.
25%	0.000000	5.213120e+04	0.000000	0.000000	0.000000	0.000000	0.000000	43.000000	0.0	0.000000	...	0.
50%	0.000000	5.812264e+04	0.000000	0.000000	0.000000	0.000000	0.000000	123.000000	0.0	0.000000	...	0.
75%	0.000000	8.108830e+04	0.000000	0.000000	0.000000	0.000000	7.000000	334.000000	0.0	0.000000	...	0.
max	1.000000	2.089189e+07	1.000000	117.000000	5054.000000	2441.000000	13567.000000	25927.000000	0.0	1.000000	...	1.

8 rows × 80 columns

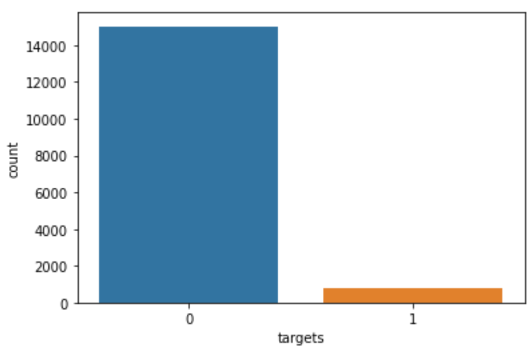
Average transaction amount is 110142.0 which is way higher than the cost of reviewing the false positives. So, it is evident that the cost of false negative is much higher than cost of a false positive(200 Dollars to investigate). Recall is more important than precision for this analysis.

In [54]:
```
transaction_data['targets'].value_counts(normalize=True)*100
```

Out[54]:
```
0    95.241113
1     4.758887
Name: targets, dtype: float64
```

In [50]:
```
sns.countplot(x='targets',data=transaction_data)
```

Out[50]:
```
<AxesSubplot:xlabel='targets', ylabel='count'>
```

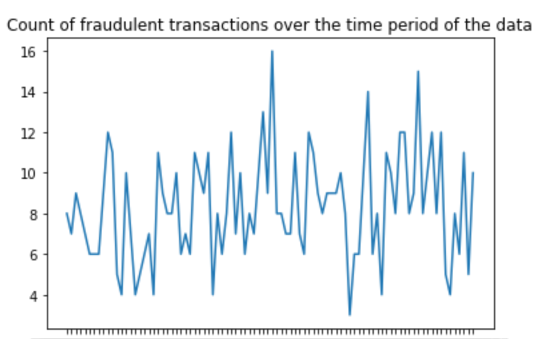


The above plot shows the count of fraud and non-fraud data.

The dataset is clearly imbalanced! To counter this, we may use sampling (oversampling/undersampling) or weigh the minority class higher while defining the loss function, use the right error metrics(precision, recall, f1 or AUC-ROC), stratify samples during train-test split, and carefully deal with outliers.

In [78]:
```
plt.plot(transaction_data.groupby(['transaction_date'])['targets'].sum())
plt.title('Count of fraudulent transactions over the time period of the data')
```

Out[78]:
```
Text(0.5, 1.0, 'Count of fraudulent transactions over the time period of the data')
```



Let's do the analysis by month, day, week-wise and look for patterns.Repeat the same for transaction amount.

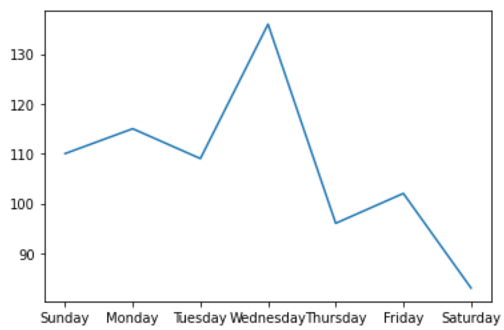
In [80]:
```
transaction_data['transaction_date']=pd.to_datetime(transaction_data['transaction_date'],format="%Y-%m-%d")
transaction_data['transaction_date_day'] = transaction_data.apply(lambda x: x['transaction_date'].day_name(),axis=1)
```

In [83]:
```
fraudulent_transactions_each_day = dict(transaction_data.groupby(['transaction_date_day'])['targets'].sum())
```

In [85]:
```
ord_list = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
fraudulent_transactions_each_day = {key : fraudulent_transactions_each_day[key] for key in ord_list}
```

In [86]:
```
plt.plot(fraudulent_transactions_each_day.keys(), fraudulent_transactions_each_day.values())
```

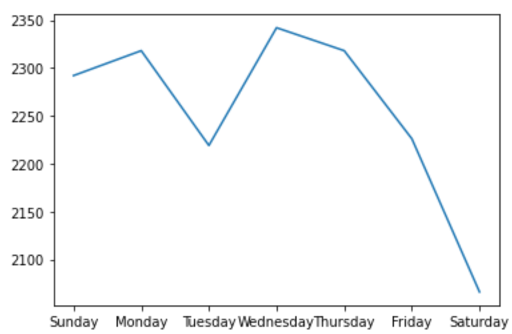
Out[86]: [<matplotlib.lines.Line2D at 0x212079b3c88>]



We see that Wednesday has the highest number of fraudulent transactions across all days of the week. The value may be significant to draw conclusions. So, let's look at the distribution of number of transactions across the days of the week.

In [90]:
```
transactions_each_day = dict(transaction_data.groupby(['transaction_date_day'])['targets'].count())
transactions_each_day = {key : transactions_each_day[key] for key in ord_list}
plt.plot(transactions_each_day.keys(), transactions_each_day.values())
```

Out[90]: [<matplotlib.lines.Line2D at 0x212042e4f08>]



We see that Wednesday also has the highest number of transactions which could be the reason why we have a high number of fraudulent cases. After checking this across months, I found that there were no significant findings.

Feature engineering

Creating a new feature called number of days between account creation and this transaction

In [92]:
```
transaction_data['account_open_date']=pd.to_datetime(transaction_data['account_open_date'],format="%Y-%m-%d")
transaction_data['transaction_date']=pd.to_datetime(transaction_data['transaction_date'],format="%Y-%m-%d")

transaction_data['Days_bw_creation_and_transaction']=transaction_data['transaction_date']-transaction_data['account_open_date']

transaction_data['Days_bw_creation_and_transaction']=transaction_data['Days_bw_creation_and_transaction']/np.timedelta64(1, 'D')

transaction_data[transaction_data['Days_bw_creation_and_transaction']<0]#Checking for inconsistencies
```

Out[92]:

transaction_id	targets	transaction_date	account_open_date	transaction_amount	beneficiary	col_0	col_1	col_2	col_3	...	col_110	col_111	col_112	col_1'

0 rows × 126 columns

In [93]:
```
transaction_data.groupby('targets')['Days_bw_creation_and_transaction'].mean()
```

Out[93]:
```
targets
0    178.728424
1     92.387483
Name: Days_bw_creation_and_transaction, dtype: float64
```

We see that on average, the day count between creation and transaction for fraud is almost half that of regular cases. This could be a useful indicator if we knew the historical transactions of the user.

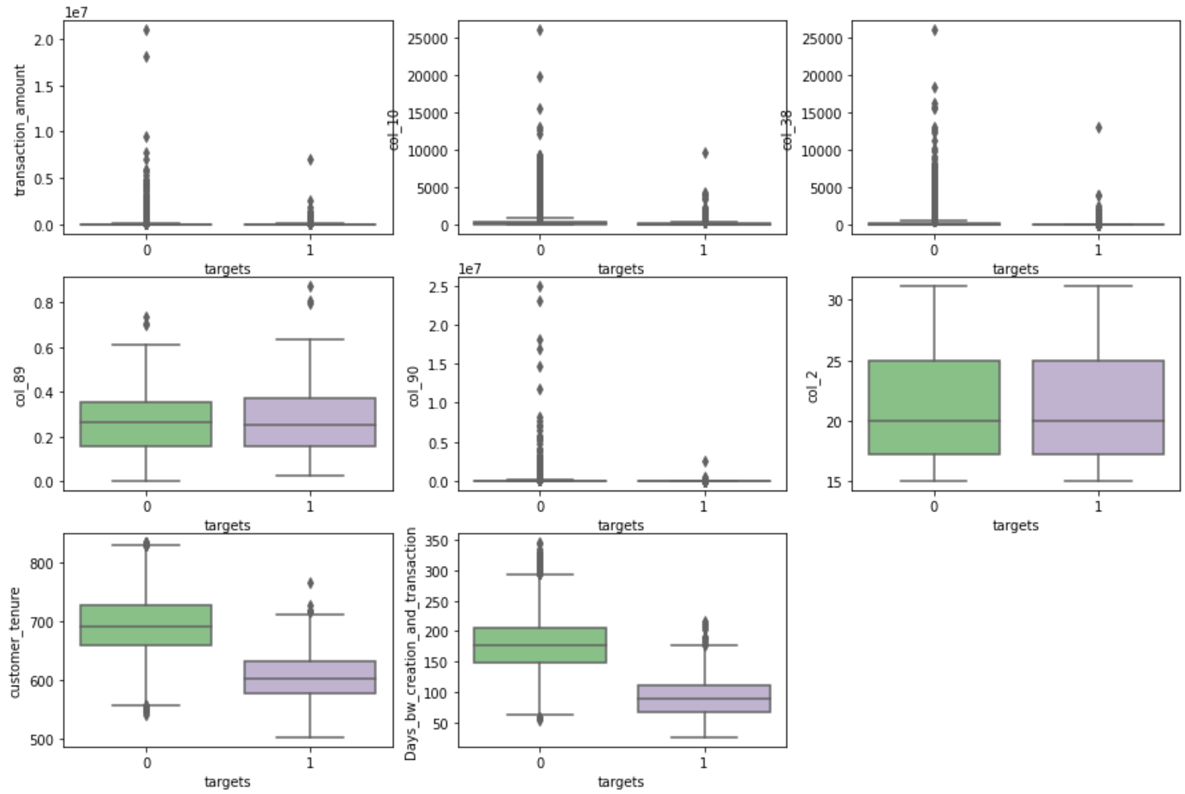
In [19]:
```
#Looking at the count of unique and null values for each column. This will be used to define the variables.
for i in transaction_data.columns:
    print(f'Column {i} of type {transaction_data[i].dtype} has {len(transaction_data[i].unique())} unique values and {transactior
```

...

In [122]:
```
#Continuous numerical variables are the variables that can take a wide range of values. Here, I'm assuming that
#the continuous variable will at least have more unique values than 10 percent of the length of the data.
numeric_columns = []
for i in transaction_data.select_dtypes(np.number).columns:
    if len(transaction_data[i].unique())>len(transaction_data)*0.10:
        numeric_columns.append(i)
numeric_columns
```

Out[122]:
```
['transaction_amount', 'col_10', 'col_38', 'col_89', 'col_90']
```

In [299]:
```
# box plot of numerical features vs Target
fig = plt.figure(figsize = (15,10))
for i in range(len(numeric_columns)):
    column = numeric_columns[i]
    sub= fig.add_subplot(3, 3, i + 1)
    sns.boxplot(x = 'targets', y = column, data = transaction_data, palette = "Accent")
```



In [22]:
```
transaction_data[numeric_columns]
```

Out[22]:

	transaction_amount	col_10	col_38	col_89	col_90
0	52092.586207	11	0	0.538550	0.000000
1	50042.970326	23	0	0.381115	0.000000
2	54255.114574	58	0	NaN	0.000000
3	61722.527737	44	29	0.178724	225.800000
4	54313.312765	1608	3034	0.394032	28343.108225
...	...	...	...	...	...
15776	58000.013393	477	686	0.328141	21179.633803
15777	50356.467423	788	1059	0.168923	45954.358974
15778	51274.840357	108	30	0.253509	172617.923077
15779	51612.014451	167	88	0.479414	323.500000
15780	55834.666756	165	0	NaN	0.000000

15781 rows × 5 columns

In [123]:
```
#This column has a few '.'. We'll convert that to nan and impute it based on K-Nearest Neighbour
numeric_columns.append('col_2')
```

Theoretically, 25 to 30% missing values are permissible, beyond which we might want to drop the variable from analysis. But, when dealing with real-world data, it is quite common to witness more than 50% missing entries. Let's take a look at the missing value distribution!

In [28]:
```
null_percent = dict((transaction_data.isnull().sum()*100)/len(transaction_data))
null_percent
```

...

In [40]:
```
list(filter(lambda x: x > 0, list(null_percent.values())))
```

...

We see that there are quite a few null percent that are greater 50 percent null values. Since, we don't have much information about the features being used. Let's set a threshold of 65 percent as the permissible amount of missing values. This is quite high! but, we can impute and later drop them if they don't add any value.

In [41]:
```
del_cols = ((transaction_data.isnull().sum()*100)/len(transaction_data)>65)
del_cols = del_cols[del_cols==True].index.tolist()
del_cols
```

Out[41]:
```
['col_17', 'col_39', 'col_50', 'col_62', 'col_80', 'col_104', 'col_112']
```

Now, there a few columns which have the same value for all the data points. These columns are not of any value. So, appending them to the list which will later be used to drop the unnecessary columns.

In [43]:
```
del_cols = del_cols + transaction_data.nunique()[transaction_data.nunique() < 2].index.tolist()
del_cols
```

...

In [44]:
```
len(transaction_data['transaction_id'].unique())
```

Out[44]:
```
15781
```

In [45]:
```
len(transaction_data['beneficiary'].unique())
```

Out[45]:
```
1531
```

In [55]:
```
transaction_data['beneficiary'].unique()
```

Out[55]:
```
array(["Manny's Auto Parts", "Zach's Agriculture",
       "Fiona's Technical Services", ..., "Jack's Leasing",
       "Zach's Insurance", "Christ's Retail"], dtype=object)
```

The above feature beneficiary can be split into two valuable columns: Company and Industry

In [64]:
```
transaction_data['company'],transaction_data['industry'] = zip(*transaction_data.apply(lambda x: x['beneficiary'].split("'s"),
                                                                                         axis=1))
```

In [73]:
```
company_grouped = transaction_data.groupby(['company'])['targets'].mean()
company_grouped.sort_values(ascending=False)
```

...

In [76]:
```
industry_grouped = transaction_data.groupby(['industry'])['targets'].mean()
industry_grouped.sort_values(ascending=False)
```

...

This could be a useful column. We see that on an average, Retail, Health care, Entertainment have the highest fraud contribution. whereas gardening, agriculture and utilites have the least.

We may also consider customer tenure as another feature (Number of days between current date and account opening date). But, given the dataset is old, it is unlikely to add value.

In [117]:
```
transaction_data['customer_tenure'] = (pd.Timestamp.now(tz=None) - pd.to_datetime(transaction_data['account_open_date'],
                                                                                   format="%Y-%m-%d")).dt.days
```

In [99]:
```
#Converting '.' to NaN for imputing with KNN
transaction_data.loc[transaction_data['col_2']=='.','col_2']=np.nan

transaction_data['col_2'] = transaction_data['col_2'].astype(float)
```

In [118]:
```
#Dropping the date,id columns and appending the derived features to the numeric columns
transaction_data.drop(columns=del_cols+['transaction_id','account_open_date','transaction_date','beneficiary'],inplace=True)
numeric_columns.extend(['customer_tenure','Days_bw_creation_and_transaction'])
```

Let's check the other numbered columns which are not continuous in nature.

In [131]:
```
#All these columns look like they are ordinal in nature. So, let's use them as categorical
for i in transaction_data.select_dtypes([np.number]).columns:
    if i not in numeric_columns:
        print(i,sorted(pd.unique(transaction_data[i])))
        transaction_data[i] = transaction_data[i].astype('category')
```

...

In [132]:
```
transaction_data[transaction_data.select_dtypes([np.number]).columns]
```

Out[132]:

	transaction_amount	col_2	col_10	col_38	col_89	col_90	Days_bw_creation_and_transaction	customer_tenure
0	52092.586207	25.00	11	0	0.538550	0.000000	97.0	656.0
1	50042.970326	25.00	23	0	0.381115	0.000000	140.0	699.0
2	54255.114574	25.00	58	0	NaN	0.000000	167.0	726.0
3	61722.527737	25.00	44	29	0.178724	225.800000	173.0	732.0
4	54313.312765	25.00	1608	3034	0.394032	28343.108225	202.0	761.0
...	...	...	...	...	...	...	...	...
15776	58000.013393	17.22	477	686	0.328141	21179.633803	125.0	595.0
15777	50356.467423	17.22	788	1059	0.168923	45954.358974	146.0	616.0
15778	51274.840357	17.22	108	30	0.253509	172617.923077	165.0	635.0
15779	51612.014451	17.22	167	88	0.479414	323.500000	183.0	653.0
15780	55834.666756	17.22	165	0	NaN	0.000000	214.0	684.0

15781 rows × 8 columns

In [151]:
```
#None of the dtype objects or bool have any null in them
for i in transaction_data.select_dtypes(['object','bool']).columns:
    print(f'Column {i} has {len(transaction_data[i].unique())} unique values and {transaction_data[i].isnull().sum()} null values
```

...

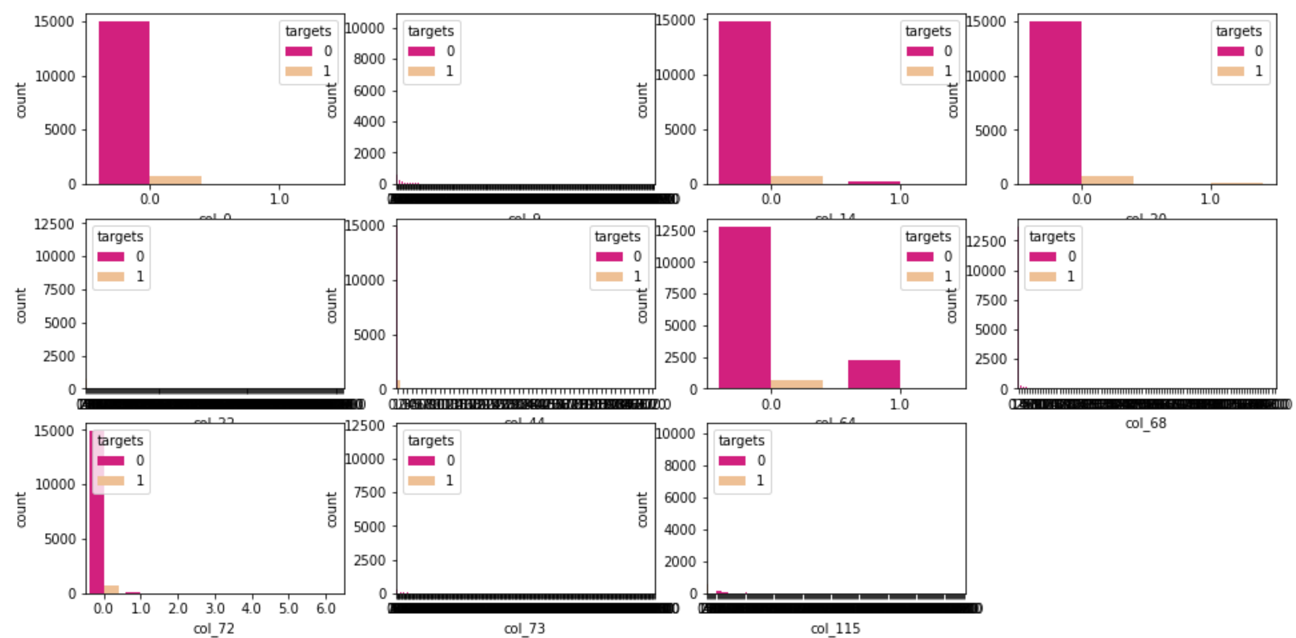
In [170]:
```
# separate categorical and numerical features
cat_nulls = (transaction_data.select_dtypes(include='category').isnull().sum())>0
cat_cols = cat_nulls[cat_nulls==True].index

#perform median imputation for categorical
for i in cat_cols:
    transaction_data[i] = transaction_data[i].fillna(transaction_data[i].astype(str).median())

# perform KNN imputation for numerical
num_nulls = (transaction_data.select_dtypes(include=[np.number]).isnull().sum())>0
num_cols = num_nulls[num_nulls==True].index
X = transaction_data[num_cols]
imputer = KNNImputer(n_neighbors=3)
X_imputed = imputer.fit_transform(X)
num_imputed = pd.DataFrame(X_imputed[:, :len(num_cols)], columns=num_cols)
transaction_data[num_cols] = num_imputed
transaction_data
```

Out[170]:

	targets	transaction_amount	col_0	col_1	col_2	col_3	col_4	col_5	col_6	col_7	...	col_109	col_110	col_111	col_114	col_115	company	industr
0	1	52092.586207	0.0	False	25.00	0	0	False	False	E	...	0	0	True	0	0.0	Manny	Auto Part
1	0	50042.970326	0.0	False	25.00	0	0	False	True	B	...	0	0	False	0	0.0	Zach	Agricultur
2	0	54255.114574	0.0	False	25.00	0	0	False	False	A	...	0	0	False	0	0.0	Fiona	Technica Service
3	0	61722.527737	0.0	True	25.00	0	9	False	False	C	...	0	0	False	0	0.0	Omar	Exterior
4	0	54313.312765	0.0	True	25.00	0	0	False	True	C	...	14	0	False	0	29.0	Steve	Utilitie
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
15776	0	58000.013393	0.0	True	17.22	0	0	False	True	A	...	9	0	False	0	5.0	Laura	Utilitie
15777	0	50356.467423	0.0	True	17.22	1	0	False	False	B	...	98	5	False	0	1.0	Kathy	Foo Processin
15778	0	51274.840357	0.0	True	17.22	0	0	False	True	C	...	8	0	False	0	0.0	Xavier	Technica Service
15779	0	51612.014451	0.0	True	17.22	2	0	False	False	A	...	0	0	False	0	0.0	David	Tutorin
15780	0	55834.666756	0.0	False	17.22	0	0	False	True	C	...	0	0	False	0	0.0	Romeo	Foo Processin

15781 rows × 101 columns

In [306]:
```
# Grouped bar plot of categorical features
fig = plt.figure(figsize = (16,8))
for i in range(len(cat_cols)):
    column =cat_cols[i]
    sub= fig.add_subplot(3, 4, i + 1)
    chart = sns.countplot(data = transaction_data, x= column, hue= 'targets', palette = 'Accent_r')
```



Feature engineering- Dropping unnecessary features

In [172]:
```
transaction_data
```

Out[172]:

	targets	transaction_amount	col_0	col_1	col_2	col_3	col_4	col_5	col_6	col_7	...	col_109	col_110	col_111	col_114	col_115	company	industry
0	1	52092.586207	0.0	False	25.00	0	0	False	False	E	...	0	0	True	0	0.0	Manny	Auto Parts
1	0	50042.970326	0.0	False	25.00	0	0	False	True	B	...	0	0	False	0	0.0	Zach	Agriculture
2	0	54255.114574	0.0	False	25.00	0	0	False	False	A	...	0	0	False	0	0.0	Fiona	Technical Services
3	0	61722.527737	0.0	True	25.00	0	9	False	False	C	...	0	0	False	0	0.0	Omar	Exteriors
4	0	54313.312765	0.0	True	25.00	0	0	False	True	C	...	14	0	False	0	29.0	Steve	Utilities
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15776	0	58000.013393	0.0	True	17.22	0	0	False	True	A	...	9	0	False	0	5.0	Laura	Utilities
15777	0	50356.467423	0.0	True	17.22	1	0	False	False	B	...	98	5	False	0	1.0	Kathy	Food Processing
15778	0	51274.840357	0.0	True	17.22	0	0	False	True	C	...	8	0	False	0	0.0	Xavier	Technical Services
15779	0	51612.014451	0.0	True	17.22	2	0	False	False	A	...	0	0	False	0	0.0	David	Tutoring
15780	0	55834.666756	0.0	False	17.22	0	0	False	True	C	...	0	0	False	0	0.0	Romeo	Food Processing

15781 rows × 101 columns

In [174]:
```
data = transaction_data.copy()

# Convert any categorical variables to numerical using label encoding
le = LabelEncoder()
for column in data.columns:
    if data[column].dtype == np.object or column=='targets':
        data[column] = le.fit_transform(data[column])

# Calculate the correlation matrix and select highly correlated features
corr_matrix = data.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
to_drop = [column for column in upper.columns if any(upper[column] > 0.75)]
# data.drop(to_drop, axis=1, inplace=True)

# Remove low variance features
selector = VarianceThreshold(threshold=0.1)
selected_data = selector.fit_transform(data)
dropped_columns = data.columns[~selector.get_support()]

# Calculate mutual information and select highly informative features
mutual_info = mutual_info_classif(data.iloc[:, 1:], data.iloc[:, 0], random_state=0)
mutual_info = pd.Series(mutual_info)
mutual_info.index = data.iloc[:, :-1].columns
mutual_info.sort_values(ascending=False,inplace=True)

# Print the results
print("Correlated features to drop: ", to_drop)
print("Mutual Information: \n", mutual_info)
print("Variance Threshold: \n", dropped_columns)
```

```
C:\Users\rmanoger\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.7_qbz5n2kfra8p0\LocalCache\local-packages\Python37\s
ite-packages\ipykernel_launcher.py:6: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silenc
e this warning, use `object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations (http
s://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)

C:\Users\rmanoger\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.7_qbz5n2kfra8p0\LocalCache\local-packages\Python37\s
ite-packages\ipykernel_launcher.py:11: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence t
his warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy sc
alar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations (http
s://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
  # This is added back by InteractiveShellApp.init_path()
C:\Users\rmanoger\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.7_qbz5n2kfra8p0\LocalCache\local-packages\Python37\s
ite-packages\sklearn\utils\validation.py:976: FutureWarning: Arrays of bytes/strings is being converted to decimal numbers if d
type='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your data to
numeric values explicitly instead.
  estimator=estimator,

Correlated features to drop:  ['col_38', 'customer_tenure']
Mutual Information: 
 company                             0.105550
transaction_date_day                0.099401
Days_bw_creation_and_transaction    0.072735
col_110                             0.043755
col_40                              0.042540
                                      ...
col_66                              0.000000
col_96                              0.000000
col_16                              0.000000
col_71                              0.000000
col_56                              0.000000
Length: 100, dtype: float64
Variance Threshold: 
 Index(['targets', 'col_0', 'col_5', 'col_12', 'col_14', 'col_20', 'col_31',
       'col_33', 'col_34', 'col_42', 'col_47', 'col_48', 'col_54', 'col_57',
       'col_60', 'col_61', 'col_66', 'col_70', 'col_72', 'col_75', 'col_82',
       'col_83', 'col_84', 'col_86', 'col_88', 'col_89', 'col_95', 'col_96',
       'col_97', 'col_100', 'col_102', 'col_107', 'col_108', 'col_111'],
      dtype='object')
```

In [195]:
```
#Removing columns with high correlation between features, low variance features and features with least mutual information
#The previous approach involved removing features based on random forest feature importance
cols_to_remove = set(list(mutual_info[mutual_info.values==0].keys())+to_drop+list(dropped_columns))-{'targets'}
```

Modelling

In [223]:
```
transaction_data_ftr_rmd = transaction_data.copy()
```

In [224]:
```
#Removing the unimportant features
transaction_data_ftr_rmd.drop(cols_to_remove,axis=1,inplace=True)
```

In [225]:
```
#Creating a count encoder for the categorical columns
transaction_data_ftr_rmd = CountEncoder(cols=['col_7','col_15','col_30','col_40','company','transaction_date_day'], normalize=Tru
```

In [227]:
```
transaction_data_ftr_rmd
```

Out[227]:

	targets	transaction_amount	col_1	col_2	col_4	col_6	col_7	col_8	col_9	col_10	...	col_99	col_105	col_106	col_109	col_110	col_114	col_115
0	1	52092.586207	False	25.00	0	False	0.059946	0	0.0	11	...	0	0	0	0	0	0	0.0
1	0	50042.970326	False	25.00	0	True	0.153412	0	0.0	23	...	0	0	0	0	0	0	0.0
2	0	54255.114574	False	25.00	0	False	0.304163	0	0.0	58	...	0	0	0	0	0	0	0.0
3	0	61722.527737	True	25.00	9	False	0.428807	0	1.0	44	...	0	0	0	0	0	0	0.0
4	0	54313.312765	True	25.00	0	True	0.428807	1	0.0	1608	...	1	1	98	14	0	0	29.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15776	0	58000.013393	True	17.22	0	True	0.304163	2	50.0	477	...	4	0	34	9	0	0	5.0
15777	0	50356.467423	True	17.22	0	False	0.153412	0	60.0	788	...	0	0	56	98	5	0	1.0
15778	0	51274.840357	True	17.22	0	True	0.428807	0	6.0	108	...	0	0	4	8	0	0	0.0
15779	0	51612.014451	True	17.22	0	False	0.304163	0	2.0	167	...	0	0	0	0	0	0	0.0
15780	0	55834.666756	False	17.22	0	True	0.428807	0	0.0	165	...	0	0	0	0	0	0	0.0

15781 rows × 56 columns

In [228]:
```
#Using a minmaxscaler because of the presence of ordinal variables which will be normalized without getting affected
slr = MinMaxScaler()
num_cols_scaled = slr.fit_transform(transaction_data_ftr_rmd.select_dtypes(np.number))
transaction_data_ftr_rmd[transaction_data_ftr_rmd.select_dtypes(np.number).columns] = num_cols_scaled
```

In [ ]:
```
#Stratify is made true so that train_test_split method returns training and test subsets that have the same proportions
#of class labels as the input dataset
X_train, X_test, y_train, y_test = train_test_split(transaction_data_ftr_rmd.drop('targets',axis=1),
                                                    transaction_data_ftr_rmd['targets'], test_size=0.2,
                                                    random_state=42,stratify=transaction_data_ftr_rmd['targets'])
```

In [229]:
```
transaction_data_ftr_rmd
```

Out[229]:

	targets	transaction_amount	col_1	col_2	col_4	col_6	col_7	col_8	col_9	col_10	...	col_99	col_105	col_106	col_109	col_110	col_114
0	1	0.000100	False	0.620112	0	False	0.016723	0	0.0	0.000424	...	0	0	0	0	0	0
1	0	0.000002	False	0.620112	0	True	0.265878	0	0.0	0.000887	...	0	0	0	0	0	0
2	0	0.000204	False	0.620112	0	False	0.667736	0	0.0	0.002237	...	0	0	0	0	0	0
3	0	0.000562	True	0.620112	9	False	1.000000	0	1.0	0.001697	...	0	0	0	0	0	0
4	0	0.000207	True	0.620112	0	True	1.000000	1	0.0	0.062020	...	1	1	98	14	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15776	0	0.000384	True	0.137182	0	True	0.667736	2	50.0	0.018398	...	4	0	34	9	0	0
15777	0	0.000017	True	0.137182	0	False	0.265878	0	60.0	0.030393	...	0	0	56	98	5	0
15778	0	0.000061	True	0.137182	0	True	1.000000	0	6.0	0.004166	...	0	0	4	8	0	0
15779	0	0.000077	True	0.137182	0	False	0.667736	0	2.0	0.006441	...	0	0	0	0	0	0
15780	0	0.000280	False	0.137182	0	True	1.000000	0	0.0	0.006364	...	0	0	0	0	0	0

15781 rows × 56 columns

GridSearchCV

In [286]:
```
clf1 = RandomForestClassifier(random_state=42)
clf2 = SVC(probability=True, random_state=42)
clf3 = LogisticRegression(random_state=42)
clf4 = DecisionTreeClassifier(random_state=42)
clf5 = KNeighborsClassifier()
clf6 = MultinomialNB()
clf7 = GradientBoostingClassifier(random_state=42)
```

In [287]:
```
param1 = {}
param1['classifier__n_estimators'] = [10, 50, 100, 250]
param1['classifier__max_depth'] = [5, 10, 20]
param1['classifier__class_weight'] = ['balanced']
param1['classifier'] = [clf1]

param2 = {}
param2['classifier__C'] = [10**-2, 10**-1, 10**0, 10**1, 10**2]
param2['classifier__class_weight'] = ['balanced']
param2['classifier'] = [clf2]

param3 = {}
param3['classifier__C'] = [10**-2, 10**-1, 10**0, 10**1, 10**2]
param3['classifier__penalty'] = ['l1', 'l2']
param3['classifier__class_weight'] = ['balanced']
param3['classifier'] = [clf3]

param4 = {}
param4['classifier__max_depth'] = [5,10,25,None]
param4['classifier__min_samples_split'] = [2,5,10]
param4['classifier__class_weight'] = ['balanced']
param4['classifier'] = [clf4]

param5 = {}
param5['classifier__n_neighbors'] = [2,5,10,25,50]
param5['classifier'] = [clf5]

param6 = {}
param6['classifier__alpha'] = [10**0, 10**1, 10**2]
param6['classifier'] = [clf6]

param7 = {}
param7['classifier__n_estimators'] = [10, 50, 100, 250]
param7['classifier__max_depth'] = [5, 10, 20]
param7['classifier'] = [clf7]
```

In [288]:
```
pipeline = Pipeline([('classifier', clf1)])
params = [param1, param2, param3, param4, param5, param6, param7]
```

In [292]:
```
gs = GridSearchCV(pipeline, params, cv=3, n_jobs=-1, scoring='roc_auc').fit(X_train, y_train)
gs.best_params_
```

Out[292]:
```
{'classifier': RandomForestClassifier(class_weight='balanced', max_depth=20, n_estimators=250,
                        random_state=42),
 'classifier__class_weight': 'balanced',
 'classifier__max_depth': 20,
 'classifier__n_estimators': 250}
```

In [290]:
```
gs.best_score_
```

Out[290]:
```
0.9921814444319997
```

In [291]:
```
print("Test Precision:",precision_score(gs.predict(X_test), y_test))
print("Test Recall:",recall_score(gs.predict(X_test), y_test))
print("Test ROC AUC Score:",roc_auc_score(gs.predict(X_test), y_test))
```

```
Test Precision: 0.6666666666666666
Test Recall: 0.9259259259259259
Test ROC AUC Score: 0.9547635533204573
```