

Task2 : Credit Card Fraud

Background:

Credit card fraud poses a significant threat to financial institutions and consumers alike, leading to substantial financial losses. Detecting fraudulent credit card transactions is crucial for preventing unauthorized activities and ensuring the security of financial transactions. Machine learning models are essential tools for identifying patterns associated with fraudulent activities, enabling timely intervention by financial institutions.

Problem Statement:

Create an efficient model to detect fraudulent credit card transactions using a dataset containing transaction information. The goal is to classify transactions as either fraudulent (1) or legitimate (0) based on various transaction features. Experiment with algorithms like Logistic Regression, Decision Trees, or Random Forests to achieve accurate and reliable classification results.

Approach:

Utilize a dataset comprising transaction features to build a fraud detection model. Experiment with machine learning algorithms, including Logistic Regression, Decision Trees, and Random Forests, for accurate binary classification. These algorithms will enable the identification of fraudulent transactions while minimizing false positives and false negatives.

Approach & Process:

1. Data Gathering
2. Data Cleaning
3. Data Visualization and Exploration
4. Data Preprocessing
5. Model Selection and Fine-Tuning or Training, Optimization, and Validation

Import necessary libraries

```
In [66]: ┆ ## Importing necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, precision_score, recall_score
from imblearn.over_sampling import RandomOverSampler
from datetime import datetime
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
```

1. Data Gathering:

Our dataset, obtained from Kaggle, serves as the foundational source for our Credit Card Fraud Detection system. This dataset contains crucial information about credit card transactions, providing essential data for training our predictive model. By thoroughly examining this Kaggle dataset, our model extracts valuable patterns related to fraudulent activities, ensuring precise fraud predictions. Through this rigorous analysis, we enhance our system's accuracy in identifying suspicious credit card transactions and mitigating potential fraud.

```
In [4]: train_df = pd.read_csv("./CREDIT CARD FRAUD Dataset/fraudTrain.csv")
test_df = pd.read_csv("./CREDIT CARD FRAUD Dataset/fraudTest.csv")
```

```
In [7]: train_df.head(3)
```

Out[7]:

		Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...
0	0	2019-01-01 00:00:18	2703186189652095		fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	Perry Cove	561
1	1	2019-01-01 00:00:44		630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	Riley Greens Suite 393	43039
2	2	2019-01-01 00:00:51		38859492057661	fraud_Lind- Buckridge	entertainment	220.11	Edward	Sanchez	M	White Dale Suite 530	594

3 rows × 23 columns



```
In [8]: train_df.shape
```

Out[8]: (1296675, 23)

In [9]: test_df.head(3)

Out[9]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	Jeff	Elliott	M	351 Darlene Green
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	Joanne	Williams	F	3638 Marsh Union
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	Ashley	Lopez	F	9333 Valentine Point

3 rows × 23 columns



In [10]: test_df.shape

Out[10]: (555719, 23)

In [12]: ► `## Basic info on data set`

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column           Non-Null Count   Dtype  
 ---  -- 
 0   Unnamed: 0        1296675 non-null    int64  
 1   trans_date_trans_time 1296675 non-null    object  
 2   cc_num            1296675 non-null    int64  
 3   merchant          1296675 non-null    object  
 4   category          1296675 non-null    object  
 5   amt               1296675 non-null    float64 
 6   first             1296675 non-null    object  
 7   last              1296675 non-null    object  
 8   gender             1296675 non-null    object  
 9   street             1296675 non-null    object  
 10  city              1296675 non-null    object  
 11  state             1296675 non-null    object  
 12  zip               1296675 non-null    int64  
 13  lat                1296675 non-null    float64 
 14  long              1296675 non-null    float64 
 15  city_pop          1296675 non-null    int64  
 16  job               1296675 non-null    object  
 17  dob               1296675 non-null    object  
 18  trans_num         1296675 non-null    object  
 19  unix_time         1296675 non-null    int64  
 20  merch_lat         1296675 non-null    float64 
 21  merch_long        1296675 non-null    float64 
 22  is_fraud          1296675 non-null    int64  
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
```

In [13]: ► `## Basic info on data set`

```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 555719 entries, 0 to 555718
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Unnamed: 0        555719 non-null   int64  
 1   trans_date_trans_time 555719 non-null   object  
 2   cc_num            555719 non-null   int64  
 3   merchant          555719 non-null   object  
 4   category          555719 non-null   object  
 5   amt               555719 non-null   float64 
 6   first              555719 non-null   object  
 7   last               555719 non-null   object  
 8   gender             555719 non-null   object  
 9   street             555719 non-null   object  
 10  city               555719 non-null   object  
 11  state              555719 non-null   object  
 12  zip                555719 non-null   int64  
 13  lat                555719 non-null   float64 
 14  long               555719 non-null   float64 
 15  city_pop           555719 non-null   int64  
 16  job                555719 non-null   object  
 17  dob                555719 non-null   object  
 18  trans_num          555719 non-null   object  
 19  unix_time          555719 non-null   int64  
 20  merch_lat          555719 non-null   float64 
 21  merch_long         555719 non-null   float64 
 22  is_fraud           555719 non-null   int64  
dtypes: float64(5), int64(6), object(12)
memory usage: 97.5+ MB
```

```
In [14]: ┌─## Check basic statistic  
train_df.describe()
```

Out[14]:

	Unnamed: 0	cc_num	amt	zip	lat	long	city_pop	unix_time	merc
count	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06
mean	6.483370e+05	4.171920e+17	7.035104e+01	4.880067e+04	3.853762e+01	-9.022634e+01	8.882444e+04	1.349244e+09	3.85373e+00
std	3.743180e+05	1.308806e+18	1.603160e+02	2.689322e+04	5.075808e+00	1.375908e+01	3.019564e+05	1.284128e+07	5.10978e+00
min	0.000000e+00	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.656723e+02	2.300000e+01	1.325376e+09	1.90277e+00
25%	3.241685e+05	1.800429e+14	9.650000e+00	2.623700e+04	3.462050e+01	-9.679800e+01	7.430000e+02	1.338751e+09	3.47335e+00
50%	6.483370e+05	3.521417e+15	4.752000e+01	4.817400e+04	3.935430e+01	-8.747690e+01	2.456000e+03	1.349250e+09	3.93656e+00
75%	9.725055e+05	4.642255e+15	8.314000e+01	7.204200e+04	4.194040e+01	-8.015800e+01	2.032800e+04	1.359385e+09	4.19571e+00
max	1.296674e+06	4.992346e+18	2.894890e+04	9.978300e+04	6.669330e+01	-6.795030e+01	2.906700e+06	1.371817e+09	6.75102e+00

```
In [15]: ┌─## Check basic statistic  
test_df.describe()
```

Out[15]:

	Unnamed: 0	cc_num	amt	zip	lat	long	city_pop	unix_time	merc
count	555719.000000	5.557190e+05	555719.000000	555719.000000	555719.000000	555719.000000	5.557190e+05	5.557190e+05	555719.000000
mean	277859.000000	4.178387e+17	69.392810	48842.628015	38.543253	-90.231325	8.822189e+04	1.380679e+09	3.85373e+00
std	160422.401459	1.309837e+18	156.745941	26855.283328	5.061336	13.721780	3.003909e+05	5.201104e+06	5.10978e+00
min	0.000000	6.041621e+10	1.000000	1257.000000	20.027100	-165.672300	2.300000e+01	1.371817e+09	1.90277e+00
25%	138929.500000	1.800429e+14	9.630000	26292.000000	34.668900	-96.798000	7.410000e+02	1.376029e+09	3.47335e+00
50%	277859.000000	3.521417e+15	47.290000	48174.000000	39.371600	-87.476900	2.408000e+03	1.380762e+09	3.93656e+00
75%	416788.500000	4.635331e+15	83.010000	72011.000000	41.894800	-80.175200	1.968500e+04	1.385867e+09	4.19571e+00
max	555718.000000	4.992346e+18	22768.110000	99921.000000	65.689900	-67.950300	2.906700e+06	1.388534e+09	6.75102e+00

In [16]:

```
## Check basic statistic  
train_df.describe(include = "all")
```

Out[16]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender
count	1.296675e+06	1296675	1.296675e+06	1296675	1296675	1.296675e+06	1296675	1296675	1296675
unique	NaN	1274791	NaN	693	14	NaN	352	481	2
top	NaN	2019-04-22 16:02:01	NaN	fraud_Kilback LLC	gas_transport	NaN	Christopher	Smith	F
freq	NaN	4	NaN	4403	131659	NaN	26669	28794	709863
mean	6.483370e+05	NaN	4.171920e+17	NaN	NaN	7.035104e+01	NaN	NaN	NaN
std	3.743180e+05	NaN	1.308806e+18	NaN	NaN	1.603160e+02	NaN	NaN	NaN
min	0.000000e+00	NaN	6.041621e+10	NaN	NaN	1.000000e+00	NaN	NaN	NaN
25%	3.241685e+05	NaN	1.800429e+14	NaN	NaN	9.650000e+00	NaN	NaN	NaN
50%	6.483370e+05	NaN	3.521417e+15	NaN	NaN	4.752000e+01	NaN	NaN	NaN
75%	9.725055e+05	NaN	4.642255e+15	NaN	NaN	8.314000e+01	NaN	NaN	NaN
max	1.296674e+06	NaN	4.992346e+18	NaN	NaN	2.894890e+04	NaN	NaN	NaN

11 rows × 23 columns

2. Data Cleaning

In the Data Cleaning step, the dataset undergoes essential operations:

1. **Removing Duplicates:** Identical entries are eliminated, ensuring each data point is unique.
2. **Removing NaN Rows and Columns:** Rows and columns with missing values are deleted, ensuring data integrity.

```
In [17]: ┌─## Check for Duplicates and Remove them  
train_df.duplicated().sum() ## Will give us a number of duplicates
```

Out[17]: 0

```
In [18]: ┌─## Check for Duplicates and Remove them  
test_df.duplicated().sum() ## Will give us a number of duplicates
```

Out[18]: 0

```
In [19]: ┌─## Check for nan values  
  
train_df.isna().sum() # Will check for any duplicates
```

Out[19]:

Unnamed:	0
trans_date_trans_time	0
cc_num	0
merchant	0
category	0
amt	0
first	0
last	0
gender	0
street	0
city	0
state	0
zip	0
lat	0
long	0
city_pop	0
job	0
dob	0
trans_num	0
unix_time	0
merch_lat	0
merch_long	0
is_fraud	0
dtype:	int64

```
In [20]: ┌─## Check for nan values  
      test_df.isna().sum() # Will check for any duplicates
```

```
Out[20]: Unnamed: 0          0  
trans_date_trans_time    0  
cc_num                   0  
merchant                 0  
category                 0  
amt                      0  
first                     0  
last                      0  
gender                    0  
street                    0  
city                      0  
state                     0  
zip                       0  
lat                        0  
long                      0  
city_pop                  0  
job                        0  
dob                        0  
trans_num                 0  
unix_time                 0  
merch_lat                 0  
merch_long                0  
is_fraud                  0  
dtype: int64
```

In [21]: # Check for values which have unique values in them

```
for column_name in train_df.columns:  
    unique_values = train_df[column_name].nunique()  
    print(f'Unique values in column {column_name}: {unique_values}')
```

```
Unique values in column Unnamed: 0: 1296675  
Unique values in column trans_date_trans_time: 1274791  
Unique values in column cc_num: 983  
Unique values in column merchant: 693  
Unique values in column category: 14  
Unique values in column amt: 52928  
Unique values in column first: 352  
Unique values in column last: 481  
Unique values in column gender: 2  
Unique values in column street: 983  
Unique values in column city: 894  
Unique values in column state: 51  
Unique values in column zip: 970  
Unique values in column lat: 968  
Unique values in column long: 969  
Unique values in column city_pop: 879  
Unique values in column job: 494  
Unique values in column dob: 968  
Unique values in column trans_num: 1296675  
Unique values in column unix_time: 1274823  
Unique values in column merch_lat: 1247805  
Unique values in column merch_long: 1275745  
Unique values in column is_fraud: 2
```

```
In [22]: ┆ ## Check for values which have unique values in them
```

```
for column_name in test_df.columns:  
    unique_values = train_df[column_name].nunique()  
    print(f'Unique values in column {column_name}: {unique_values}')
```

```
Unique values in column Unnamed: 0: 1296675  
Unique values in column trans_date_trans_time: 1274791  
Unique values in column cc_num: 983  
Unique values in column merchant: 693  
Unique values in column category: 14  
Unique values in column amt: 52928  
Unique values in column first: 352  
Unique values in column last: 481  
Unique values in column gender: 2  
Unique values in column street: 983  
Unique values in column city: 894  
Unique values in column state: 51  
Unique values in column zip: 970  
Unique values in column lat: 968  
Unique values in column long: 969  
Unique values in column city_pop: 879  
Unique values in column job: 494  
Unique values in column dob: 968  
Unique values in column trans_num: 1296675  
Unique values in column unix_time: 1274823  
Unique values in column merch_lat: 1247805  
Unique values in column merch_long: 1275745  
Unique values in column is_fraud: 2
```

```
In [23]: ┆ ## Drop unnecessary columns like unnamed:0
```

```
dropColumn = ["Unnamed: 0", "job", "cc_num", "trans_num", "zip", "unix_time", "merch_lat", "merch_long", "dob", "t  
train_df.drop(dropColumn, axis = 1, inplace = True)  
test_df.drop(dropColumn, axis = 1, inplace = True)
```

```
In [24]: ► train_df.head(3)
```

Out[24]:

	trans_date_trans_time	merchant	category	amt	gender	state	lat	long	city_pop	is_fraud
0	2019-01-01 00:00:18	fraud_Rippin, Kub and Mann	misc_net	4.97	F	NC	36.0788	-81.1781	3495	0
1	2019-01-01 00:00:44	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	F	WA	48.8878	-118.2105	149	0
2	2019-01-01 00:00:51	fraud_Lind-Buckridge	entertainment	220.11	M	ID	42.1808	-112.2620	4154	0

```
In [25]: ► test_df.head(3)
```

Out[25]:

	trans_date_trans_time	merchant	category	amt	gender	state	lat	long	city_pop	is_fraud
0	2020-06-21 12:14:25	fraud_Kirlin and Sons	personal_care	2.86	M	SC	33.9659	-80.9355	333497	0
1	2020-06-21 12:14:33	fraud_Sporer-Keebler	personal_care	29.84	F	UT	40.3207	-110.4360	302	0
2	2020-06-21 12:14:53	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	F	NY	40.6729	-73.5365	34496	0

```
In [27]: ► ## Create a function convert the trans_time into day of week
```

```
def datetoDay(time):
    timestamp = datetime.strptime(time, "%Y-%m-%d %H:%M:%S")

    # Get the day of the week as a string
    day_of_week = timestamp.strftime("%A")
    return day_of_week
```

```
In [28]: ► ## Apply the conversion on each sample of test and train
```

```
train_df["trans_date_trans_time"] = train_df["trans_date_trans_time"].apply(datetoDay)
test_df["trans_date_trans_time"] = test_df["trans_date_trans_time"].apply(datetoDay)
```

```
In [29]: ⏷ train_df.head(3)
```

Out[29]:

	trans_date_trans_time	merchant	category	amt	gender	state	lat	long	city_pop	is_fraud
0	Tuesday	fraud_Rippin, Kub and Mann	misc_net	4.97	F	NC	36.0788	-81.1781	3495	0
1	Tuesday	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	F	WA	48.8878	-118.2105	149	0
2	Tuesday	fraud_Lind-Buckridge	entertainment	220.11	M	ID	42.1808	-112.2620	4154	0

```
In [30]: ⏷ test_df.head(3)
```

Out[30]:

	trans_date_trans_time	merchant	category	amt	gender	state	lat	long	city_pop	is_fraud
0	Sunday	fraud_Kirlin and Sons	personal_care	2.86	M	SC	33.9659	-80.9355	333497	0
1	Sunday	fraud_Sporer-Keebler	personal_care	29.84	F	UT	40.3207	-110.4360	302	0
2	Sunday	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	F	NY	40.6729	-73.5365	34496	0

3. Data Visualization and Exploration

In this step, we'll analyze the dataset using histograms and bar charts to understand numerical and categorical distributions.

```
In [32]: ⏷ # Concatenating train_df and test_df into a single dataframe  
df = pd.concat([train_df, test_df], ignore_index=True)
```

In [33]: df.head(10)

Out[33]:

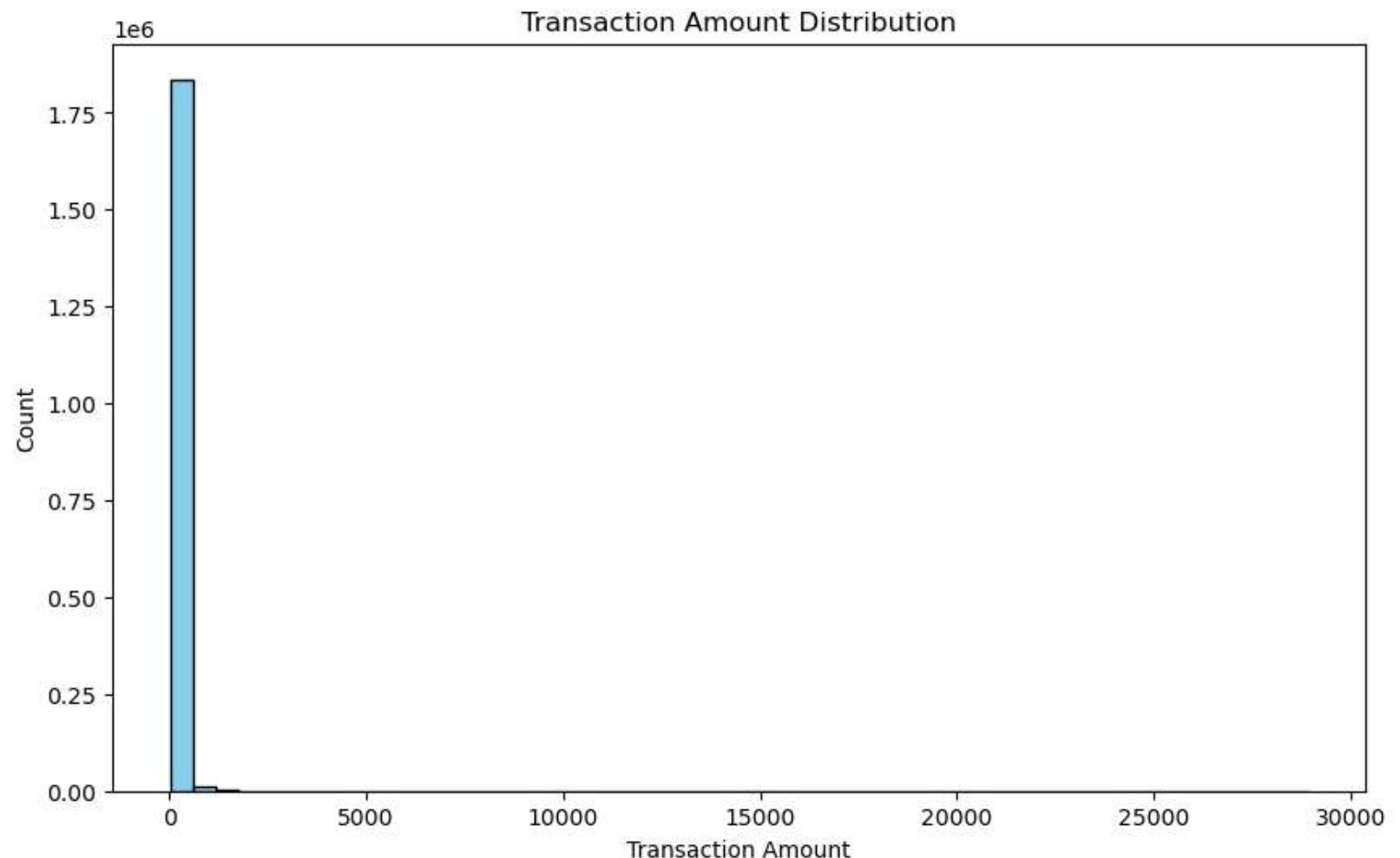
	trans_date_trans_time	merchant	category	amt	gender	state	lat	long	city_pop	is_fraud
0	Tuesday	fraud_Rippin, Kub and Mann	misc_net	4.97	F	NC	36.0788	-81.1781	3495	0
1	Tuesday	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	F	WA	48.8878	-118.2105	149	0
2	Tuesday	fraud_Lind-Buckridge	entertainment	220.11	M	ID	42.1808	-112.2620	4154	0
3	Tuesday	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	M	MT	46.2306	-112.1138	1939	0
4	Tuesday	fraud_Keeling-Crist	misc_pos	41.96	M	VA	38.4207	-79.4629	99	0
5	Tuesday	fraud_Stroman, Hudson and Erdman	gas_transport	94.63	F	PA	40.3750	-75.2045	2158	0
6	Tuesday	fraud_Rowe-Vandervort	grocery_net	44.54	F	KS	37.9931	-100.9893	2691	0
7	Tuesday	fraud_Corwin-Collins	gas_transport	71.65	M	VA	38.8432	-78.6003	6018	0
8	Tuesday	fraud_Herzog Ltd	misc_pos	4.27	F	PA	40.3359	-79.6607	1472	0
9	Tuesday	fraud_Schoen, Kuphal and Nitzsche	grocery_pos	198.39	F	TN	36.5220	-87.3490	151785	0

In [57]: df.shape

Out[57]: (1852394, 10)

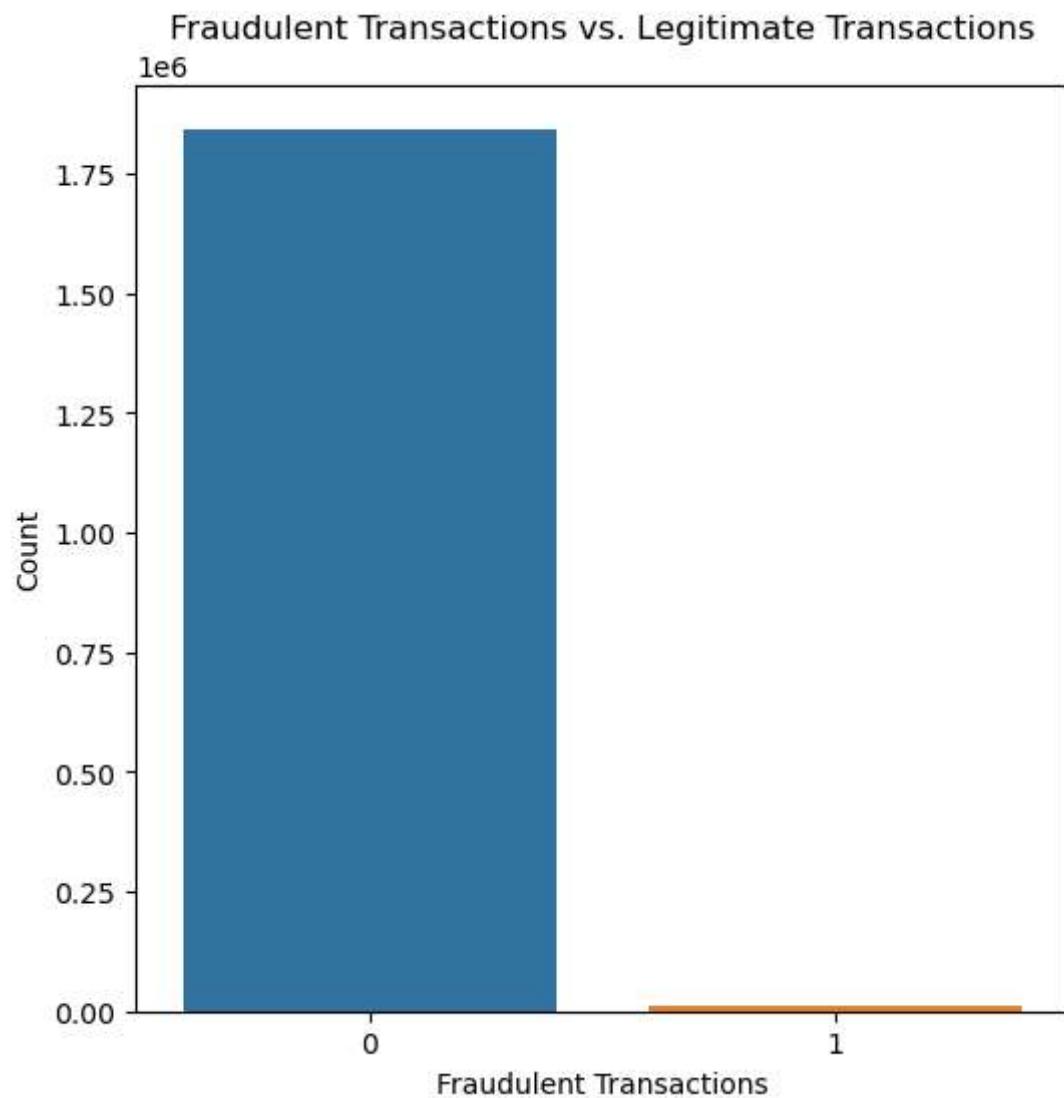
1. Transaction Amount Distribution:

```
In [34]: plt.figure(figsize=(10, 6))
plt.hist(df['amt'], bins=50, color='skyblue', edgecolor='black')
plt.xlabel('Transaction Amount')
plt.ylabel('Count')
plt.title('Transaction Amount Distribution')
plt.show()
```



2. Fraudulent Transactions vs. Legitimate Transactions:

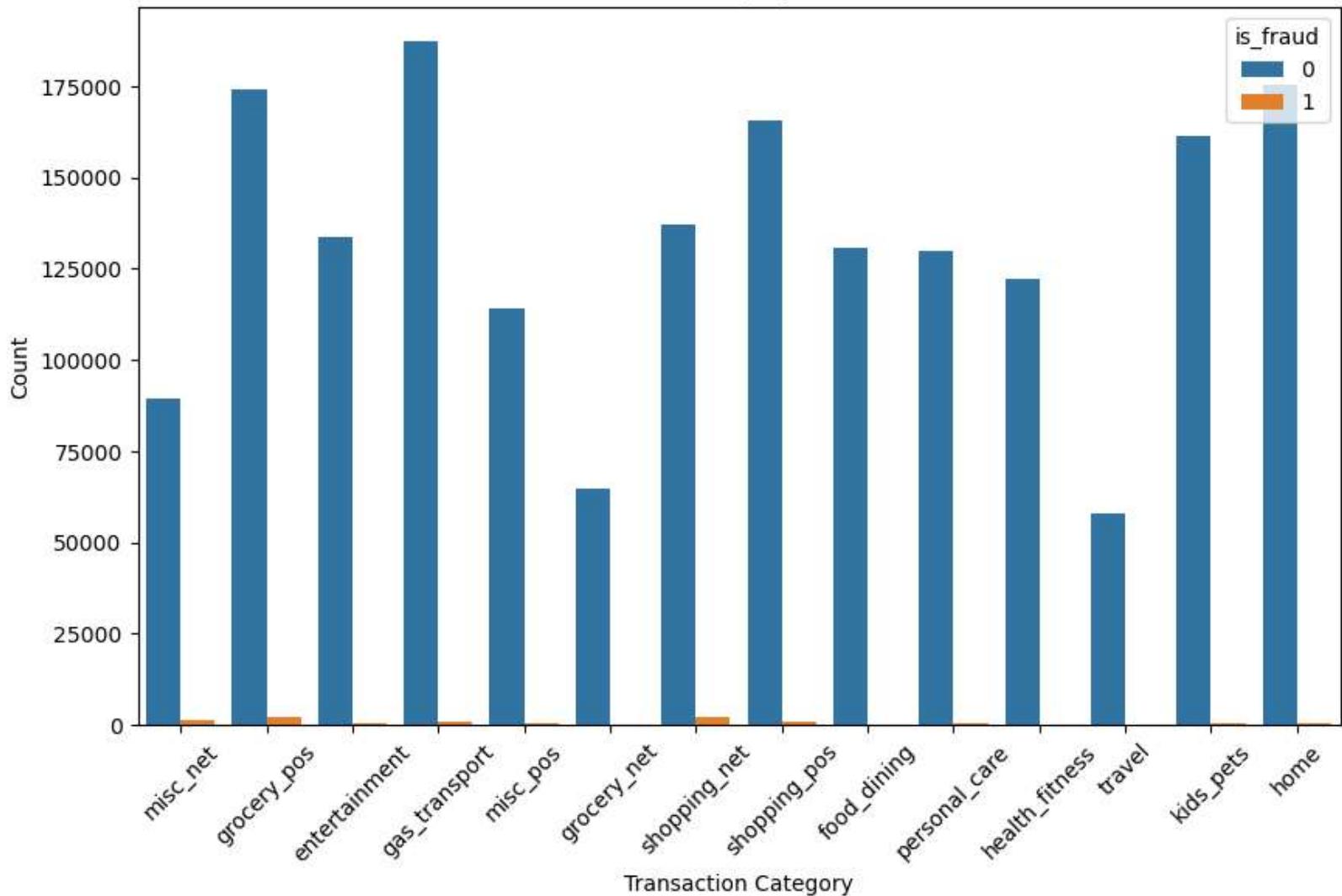
```
In [36]: ➜ import seaborn as sns  
  
plt.figure(figsize=(6, 6))  
sns.countplot(x='is_fraud', data=df)  
plt.xlabel('Fraudulent Transactions')  
plt.ylabel('Count')  
plt.title('Fraudulent Transactions vs. Legitimate Transactions')  
plt.show()
```



3. Transaction Category Distribution:

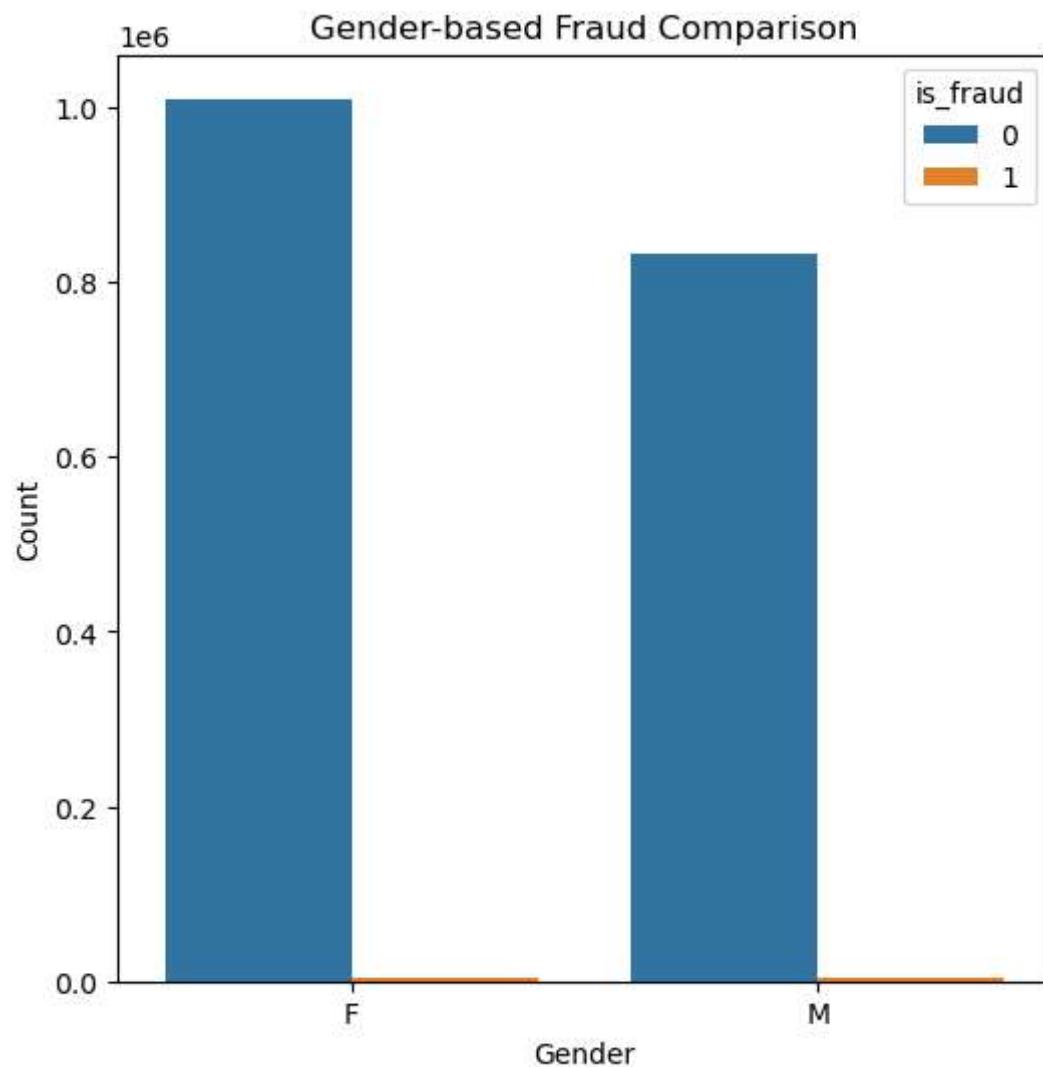
```
In [38]: ► plt.figure(figsize=(10, 6))
sns.countplot(x='category', hue='is_fraud', data=df)
plt.xlabel('Transaction Category')
plt.ylabel('Count')
plt.title('Transaction Category Distribution')
plt.xticks(rotation=45)
plt.show()
```

Transaction Category Distribution



4. Gender-based Fraud Comparison:

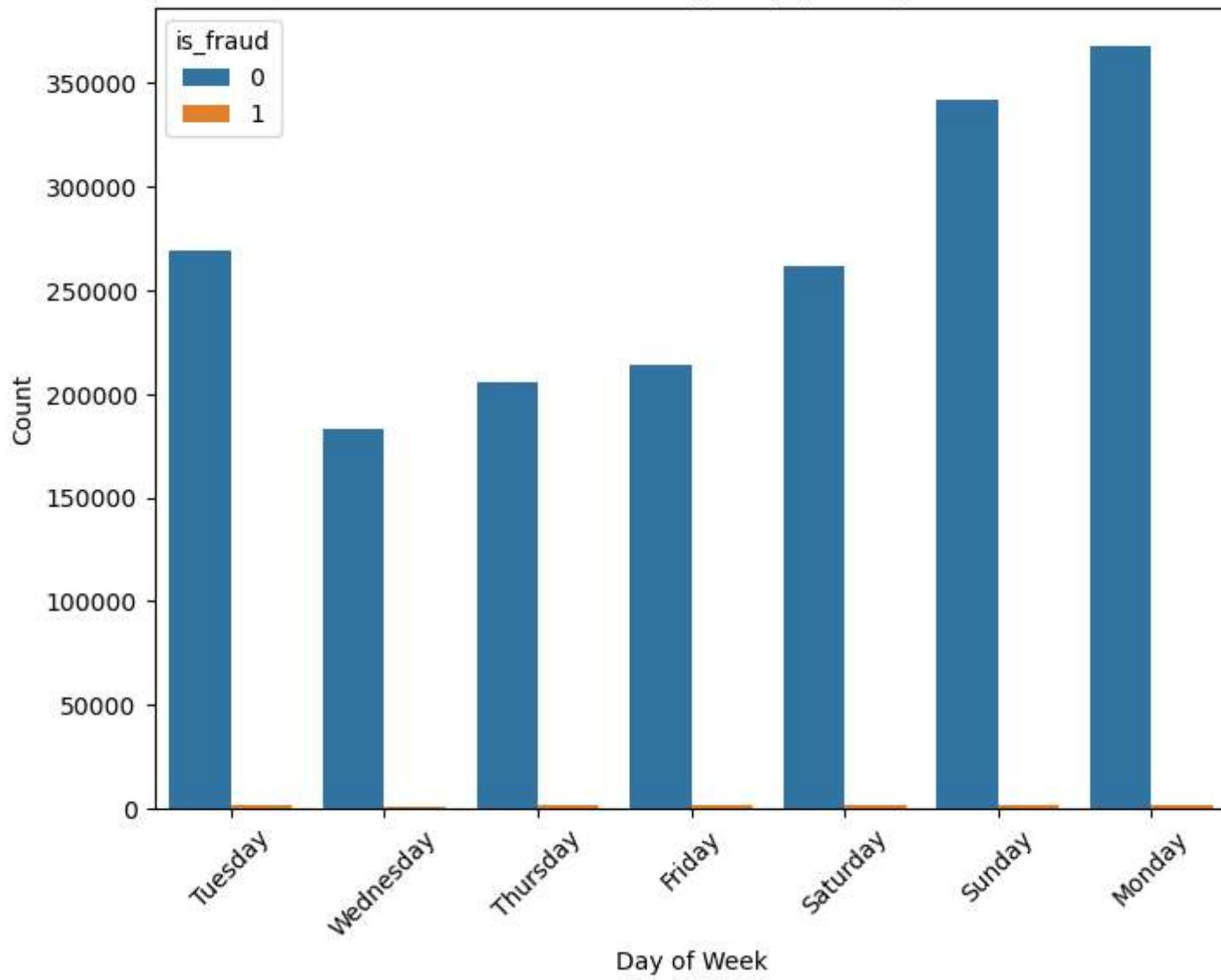
```
In [40]: ► plt.figure(figsize=(6, 6))
sns.countplot(x='gender', hue='is_fraud', data=df)
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Gender-based Fraud Comparison')
plt.show()
```



5. Transaction Frequency per Day:

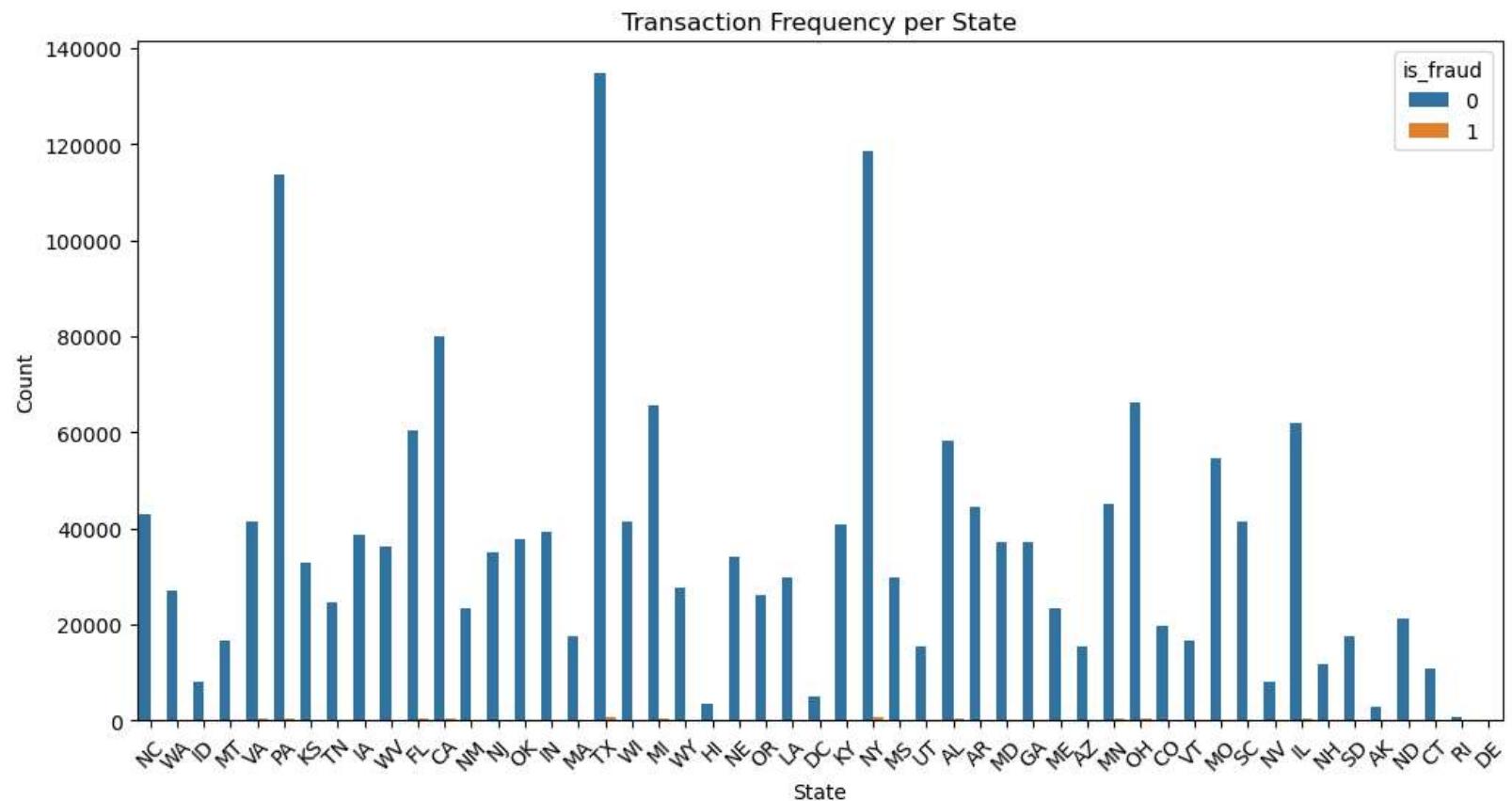
```
In [43]: ► plt.figure(figsize=(8, 6))
sns.countplot(x='trans_date_trans_time', hue='is_fraud', data=df)
plt.xlabel('Day of Week')
plt.ylabel('Count')
plt.title('Transaction Frequency per Day')
plt.xticks(rotation=45)
plt.show()
```

Transaction Frequency per Day



6. Transaction Frequency per State:

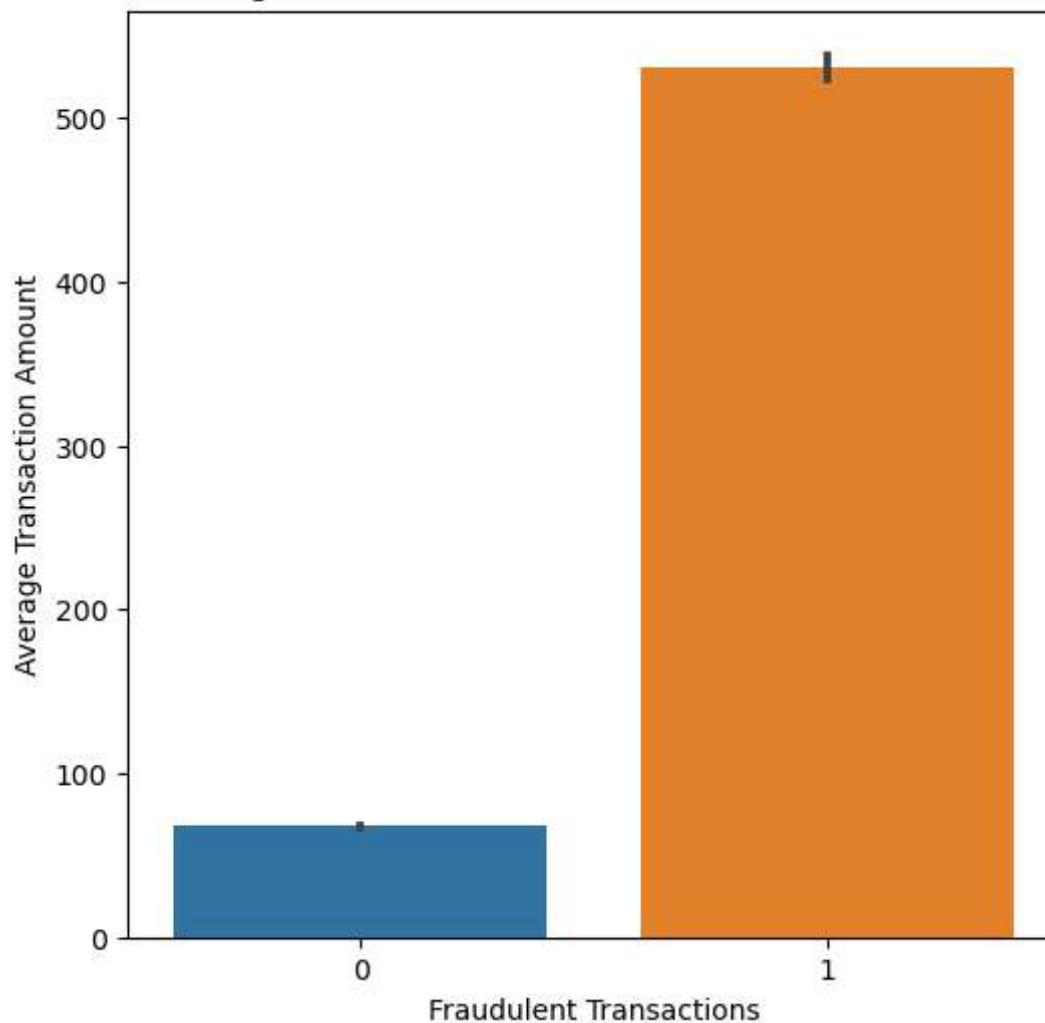
```
In [44]: # plt.figure(figsize=(12, 6))
sns.countplot(x='state', hue='is_fraud', data=df)
plt.xlabel('State')
plt.ylabel('Count')
plt.title('Transaction Frequency per State')
plt.xticks(rotation=45)
plt.show()
```



7. Average Transaction Amount for Fraud vs. Non-Fraud:

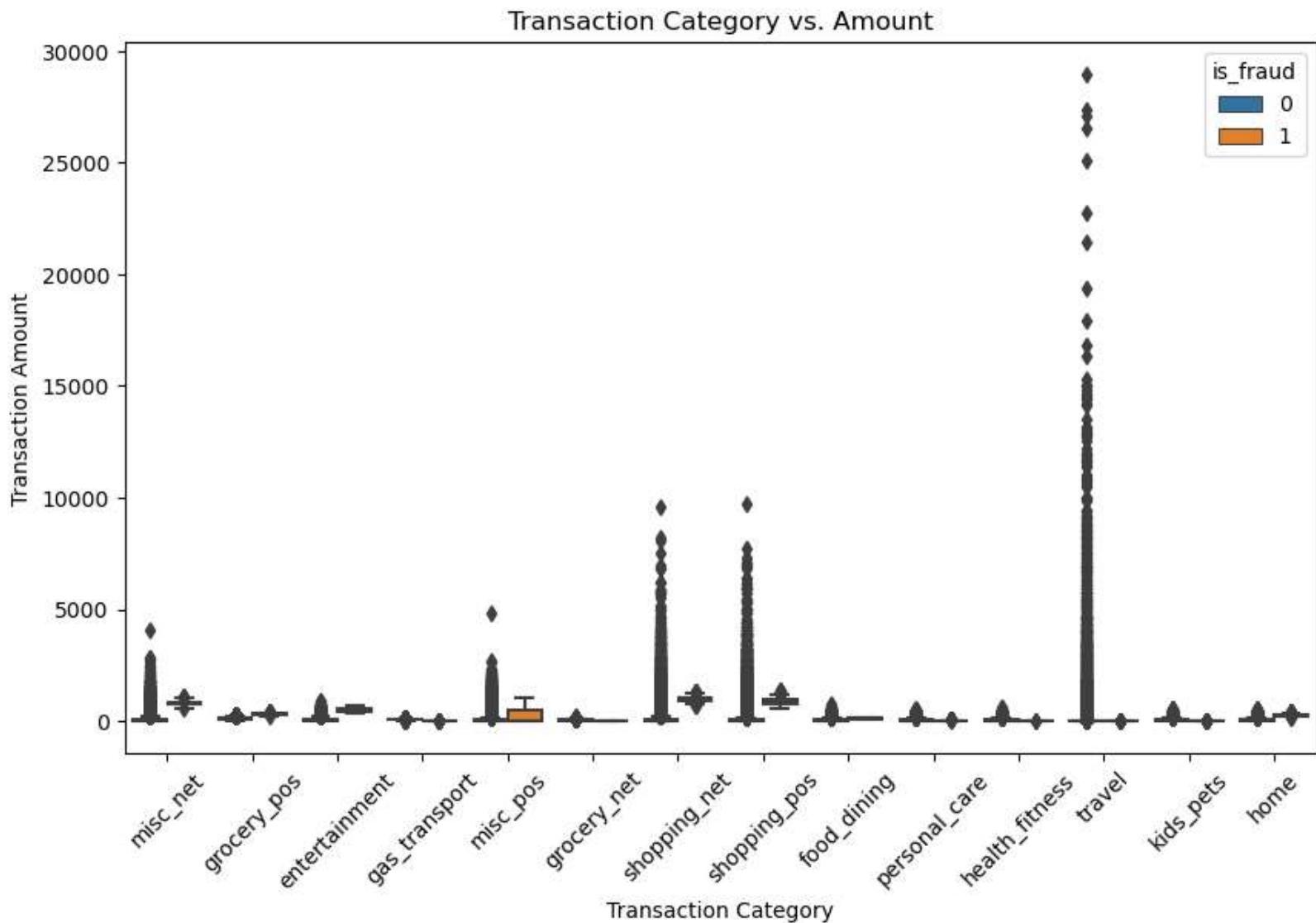
```
In [46]: ► plt.figure(figsize=(6, 6))
sns.barplot(x='is_fraud', y='amt', data=df)
plt.xlabel('Fraudulent Transactions')
plt.ylabel('Average Transaction Amount')
plt.title('Average Transaction Amount for Fraud vs. Non-Fraud')
plt.show()
```

Average Transaction Amount for Fraud vs. Non-Fraud



8. Transaction Category vs. Amount:

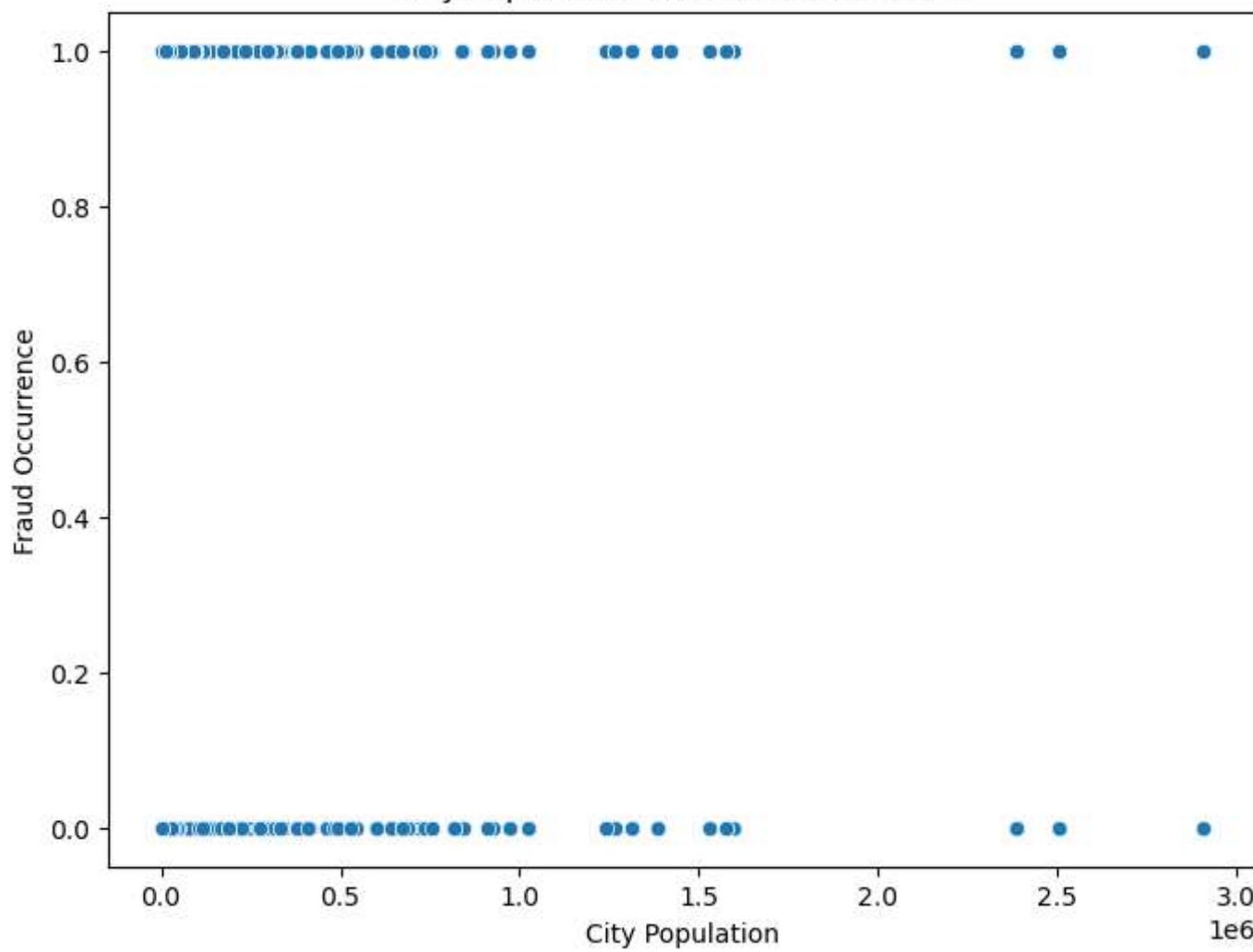
```
In [51]: ► plt.figure(figsize=(10, 6))
sns.boxplot(x='category', y='amt', hue='is_fraud', data=df)
plt.xlabel('Transaction Category')
plt.ylabel('Transaction Amount')
plt.title('Transaction Category vs. Amount')
plt.xticks(rotation=45)
plt.show()
```



9. City Population vs. Fraud Occurrence:

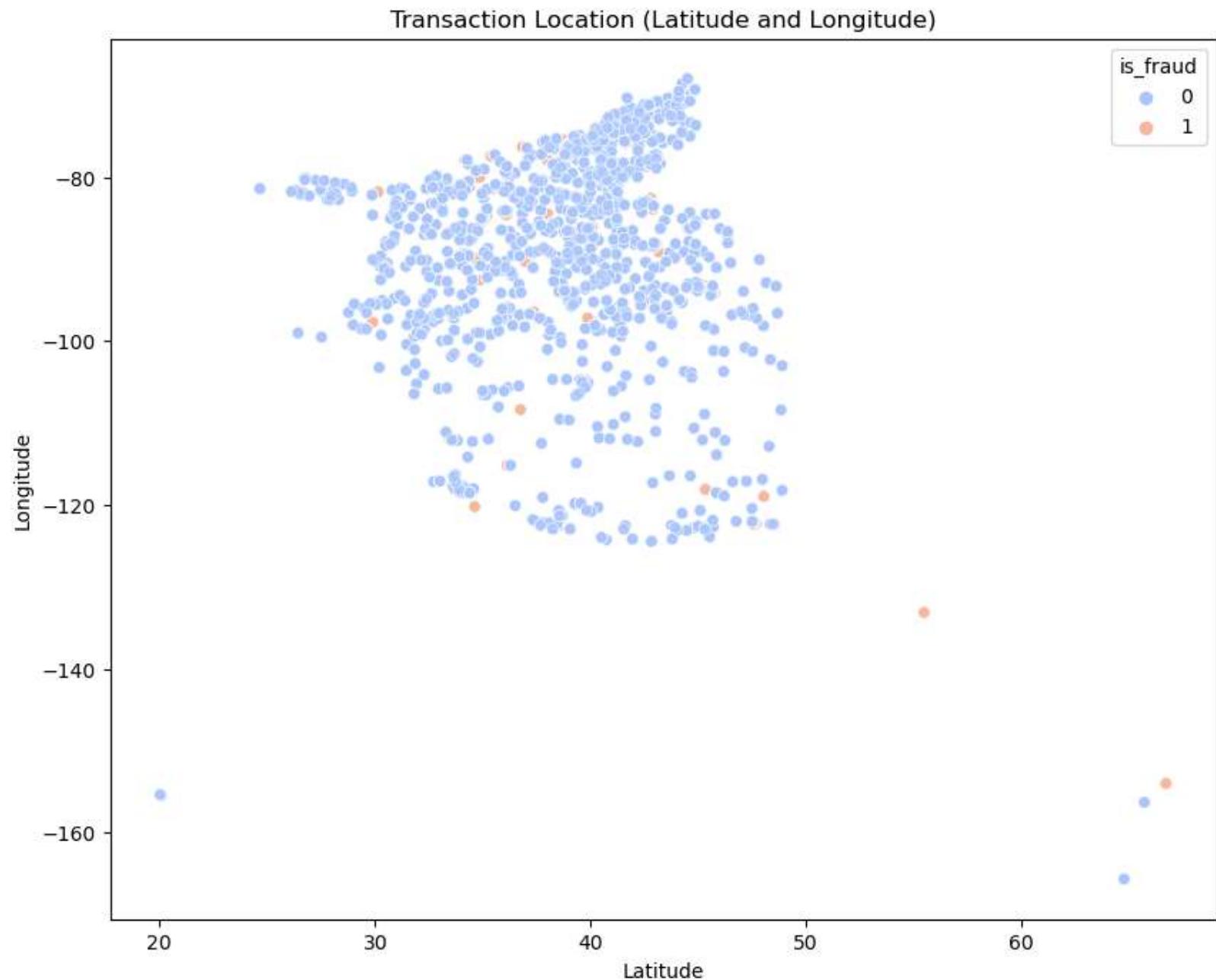
```
In [49]: ► plt.figure(figsize=(8, 6))
sns.scatterplot(x='city_pop', y='is_fraud', data=df)
plt.xlabel('City Population')
plt.ylabel('Fraud Occurrence')
plt.title('City Population vs. Fraud Occurrence')
plt.show()
```

City Population vs. Fraud Occurrence



10. Transaction Location (Latitude and Longitude):

```
In [50]: ► plt.figure(figsize=(10, 8))
sns.scatterplot(x='lat', y='long', hue='is_fraud', data=df, palette='coolwarm', alpha=0.7)
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Transaction Location (Latitude and Longitude)')
plt.show()
```



4. Data Preprocessing

Imbalance Data

```
In [52]: ➜ x_train = train_df.drop("is_fraud" ,axis = 1)
y_train = train_df["is_fraud"]

x_test = test_df.drop("is_fraud" ,axis = 1)
y_test = test_df[ "is_fraud"]

categorical = ["trans_date_trans_time","merchant","category","gender","state"]
numeric = ["amt","lat" , "long" , "city_pop"]

# Initialize the encoder with handle_unknown='ignore'
encoder = OneHotEncoder(handle_unknown='ignore')
scaler = StandardScaler()

transformer = ColumnTransformer(transformers = [ ("categorical",encoder ,categorical),
                                                 ("num",scaler,numeric)],remainder="passthrough")
x_train = transformer.fit_transform(x_train)
x_test = transformer.transform(x_test)
```

```
In [53]: ➜ x_train.shape
```

```
Out[53]: (1296675, 771)
```

```
In [54]: ➜ x_test.shape
```

```
Out[54]: (555719, 771)
```

```
In [55]: ➜ y_train.shape
```

```
Out[55]: (1296675,)
```

```
In [56]: ➜ y_test.shape
```

```
Out[56]: (555719,)
```

5. Model Selection and Fine-Tuning or Training, Optimization, and Validation

1. Model Options:

- Consider Logistic Regression, Random Forest, and DecisionTree.

2. Data Splitting:

- Divide the dataset into training and testing sets (e.g., 80-20 split).

3. Training:

- Train models with default parameters on the training data.

4. Evaluation:

- Use metrics like accuracy, precision, recall, and F1-score for evaluation.
- Apply cross-validation techniques for robust assessment.

5. Hyperparameter Tuning:

- Fine-tune Random Forest and DecisionTree using grid/random search.

6. Comparison:

- Compare models based on evaluation metrics.

7. Deployment and Monitoring:

- Deploy the best-performing model and establish continuous monitoring for accuracy and relevance. Update the model as needed based on changing data patterns.

```
In [58]: ➜ class_names = [0, 1]
print(class_names)
```

```
[0, 1]
```

In [59]: df.head(10)

Out[59]:

	trans_date_trans_time	merchant	category	amt	gender	state	lat	long	city_pop	is_fraud
0	Tuesday	fraud_Rippin, Kub and Mann	misc_net	4.97	F	NC	36.0788	-81.1781	3495	0
1	Tuesday	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	F	WA	48.8878	-118.2105	149	0
2	Tuesday	fraud_Lind-Buckridge	entertainment	220.11	M	ID	42.1808	-112.2620	4154	0
3	Tuesday	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	M	MT	46.2306	-112.1138	1939	0
4	Tuesday	fraud_Keeling-Crist	misc_pos	41.96	M	VA	38.4207	-79.4629	99	0
5	Tuesday	fraud_Stroman, Hudson and Erdman	gas_transport	94.63	F	PA	40.3750	-75.2045	2158	0
6	Tuesday	fraud_Rowe-Vandervort	grocery_net	44.54	F	KS	37.9931	-100.9893	2691	0
7	Tuesday	fraud_Corwin-Collins	gas_transport	71.65	M	VA	38.8432	-78.6003	6018	0
8	Tuesday	fraud_Herzog Ltd	misc_pos	4.27	F	PA	40.3359	-79.6607	1472	0
9	Tuesday	fraud_Schoen, Kuphal and Nitzsche	grocery_pos	198.39	F	TN	36.5220	-87.3490	151785	0

DecisionTreeClassifier

In [60]:

```
mnb = DecisionTreeClassifier()
mnb.fit(x_train, y_train)

# Evaluate the model
train_accuracy = mnb.score(x_train, y_train)
test_accuracy = mnb.score(x_test, y_test)

# Print model accuracy
print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")

# Make predictions and display classification report
y_pred = mnb.predict(x_test)
class_report = classification_report(y_test, y_pred)

print("Classification Report:\n", class_report)
```

Training Accuracy: 1.00

Test Accuracy: 1.00

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	553574
1	0.53	0.57	0.55	2145
accuracy			1.00	555719
macro avg	0.76	0.79	0.77	555719
weighted avg	1.00	1.00	1.00	555719

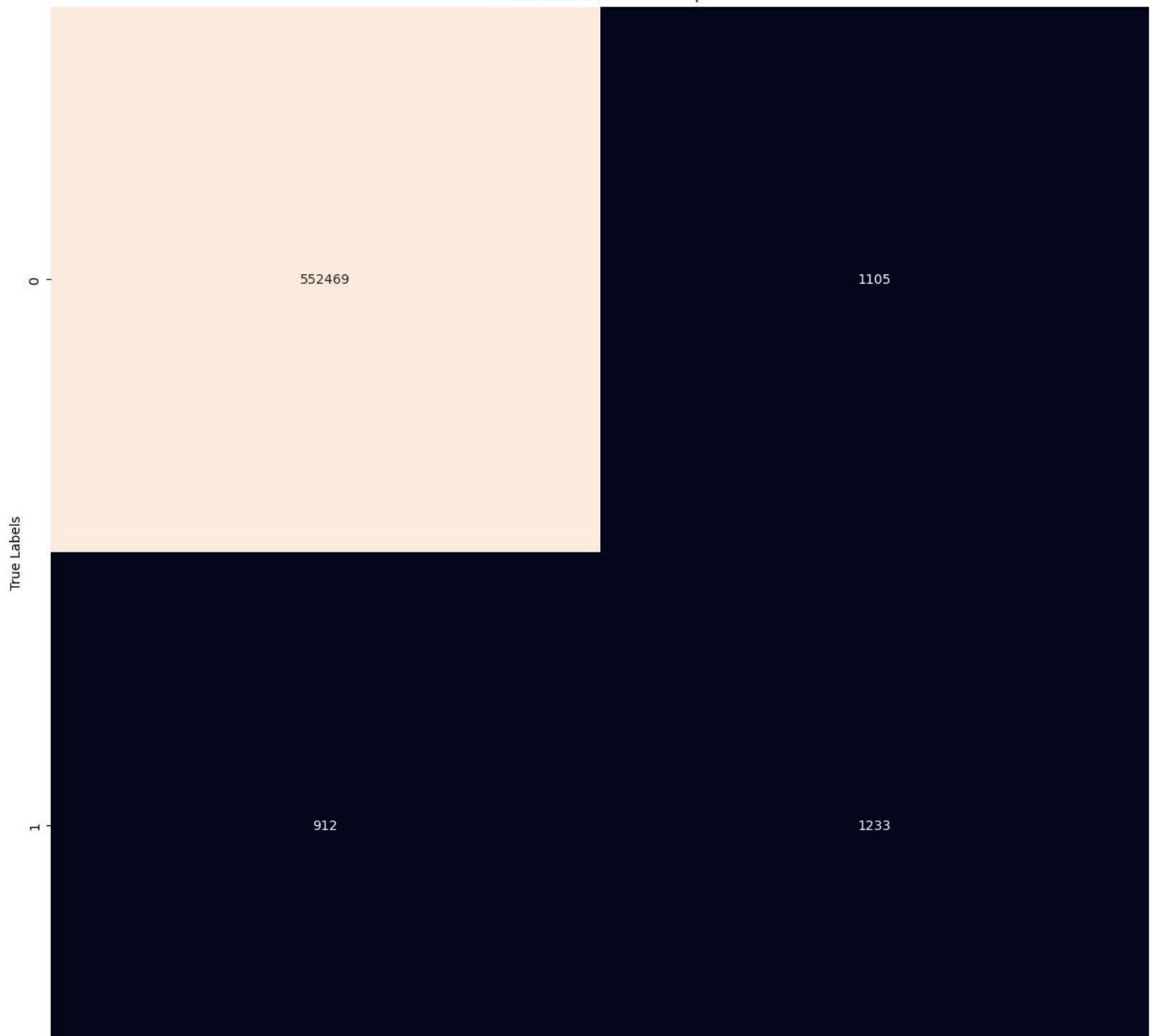
- **Model Choice :**
 - Decision Tree Classifier
- **Accuracy:**
 - Training Accuracy: 1.00 (Perfect fit to training data).

- Test Accuracy: 1.00 (Perfect fit to test data).
- **Classification Report:**
 - Precision (Non-fraudulent transactions): 1.00 (100% accuracy).
 - Precision (Fraudulent transactions): 0.53 (53% accuracy).
 - Recall (Non-fraudulent transactions): 1.00 (100% coverage).
 - Recall (Fraudulent transactions): 0.57 (57% coverage).
- **Summary:**
 - The model is highly accurate but may have issues with overfitting, especially in predicting fraudulent transactions. Further evaluation and fine-tuning are needed to enhance predictions, especially for fraud detection.
 - Address the class imbalance issue by employing techniques like oversampling the minority class (fraudulent transactions) or using different evaluation metrics like F1-score, which consider both precision and recall.

```
In [61]: # Generate confusion matrix  
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
plt.figure(figsize=(15, 15))  
sns.heatmap(conf_matrix, annot=True, fmt='d', cbar=False,  
            xticklabels=class_names, yticklabels=class_names)  
plt.xlabel('Predicted Labels')  
plt.ylabel('True Labels')  
plt.title('Confusion Matrix Heatmap')  
plt.show()
```


Confusion Matrix Heatmap





LogisticRegression

In [62]: ►

```
model = LogisticRegression()
model.fit(x_train ,y_train)
print("Model Score on Training data",model.score(x_train ,y_train))
print("Model Score on Testing data",model.score(x_test ,y_test))
y_pred = model.predict(x_test)
print(classification_report(y_pred ,y_test))
```

E:\Anaconda1\lib\site-packages\sklearn\linear_model_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

n_iter_i = _check_optimize_result(

Model Score on Training data 0.9936823028129639

Model Score on Testing data 0.995531914510751

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	555379
1	0.00	0.00	0.00	340

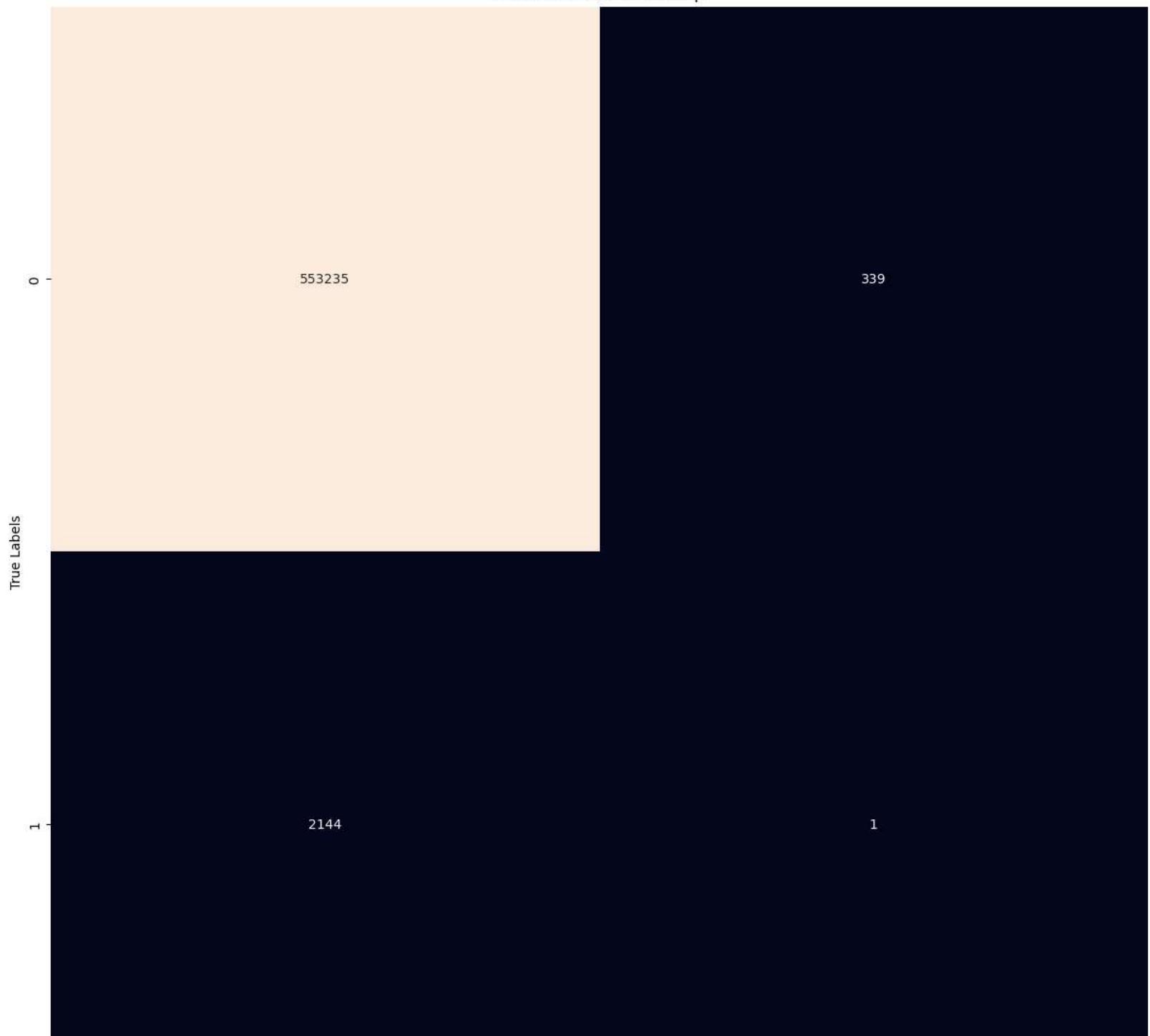
accuracy			1.00	555719
macro avg	0.50	0.50	0.50	555719
weighted avg	1.00	1.00	1.00	555719

- **Model:** Logistic Regression
- **Training Data Accuracy:** 99.37%
- **Testing Data Accuracy:** 99.55%
- **Precision (Class 1):** 0.00
- **Recall (Class 1):** 0.00
- **F1-Score (Class 1):** 0.00
- **Class Imbalance Issue:** The model correctly predicted non-fraudulent transactions but failed to detect any fraudulent transactions, indicating a severe issue in identifying fraud.

```
In [63]: cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap





RandomForestClassifier

```
In [64]: # Create a Random Forest model
random_forest = RandomForestClassifier()

random_forest.fit(x_train, y_train)
print("Random Forest - Train Score:", random_forest.score(x_train, y_train))
print("Random Forest - Test Score:", random_forest.score(x_test, y_test))

y_pred = random_forest.predict(x_test)
print(classification_report(y_pred, y_test))
```

```
Random Forest - Train Score: 0.9999969151869204
Random Forest - Test Score: 0.9973673745184167
            precision    recall   f1-score   support
0           1.00     1.00     1.00    554683
1           0.40     0.83     0.54     1036

accuracy                           1.00    555719
macro avg       0.70     0.91     0.77    555719
weighted avg    1.00     1.00     1.00    555719
```

- **Model:** Random Forest
- **Training Data Accuracy:** 99.99%
- **Testing Data Accuracy:** 99.74%
- **Precision (Class 1):** 0.40
- **Recall (Class 1):** 0.83
- **F1-Score (Class 1):** 0.54

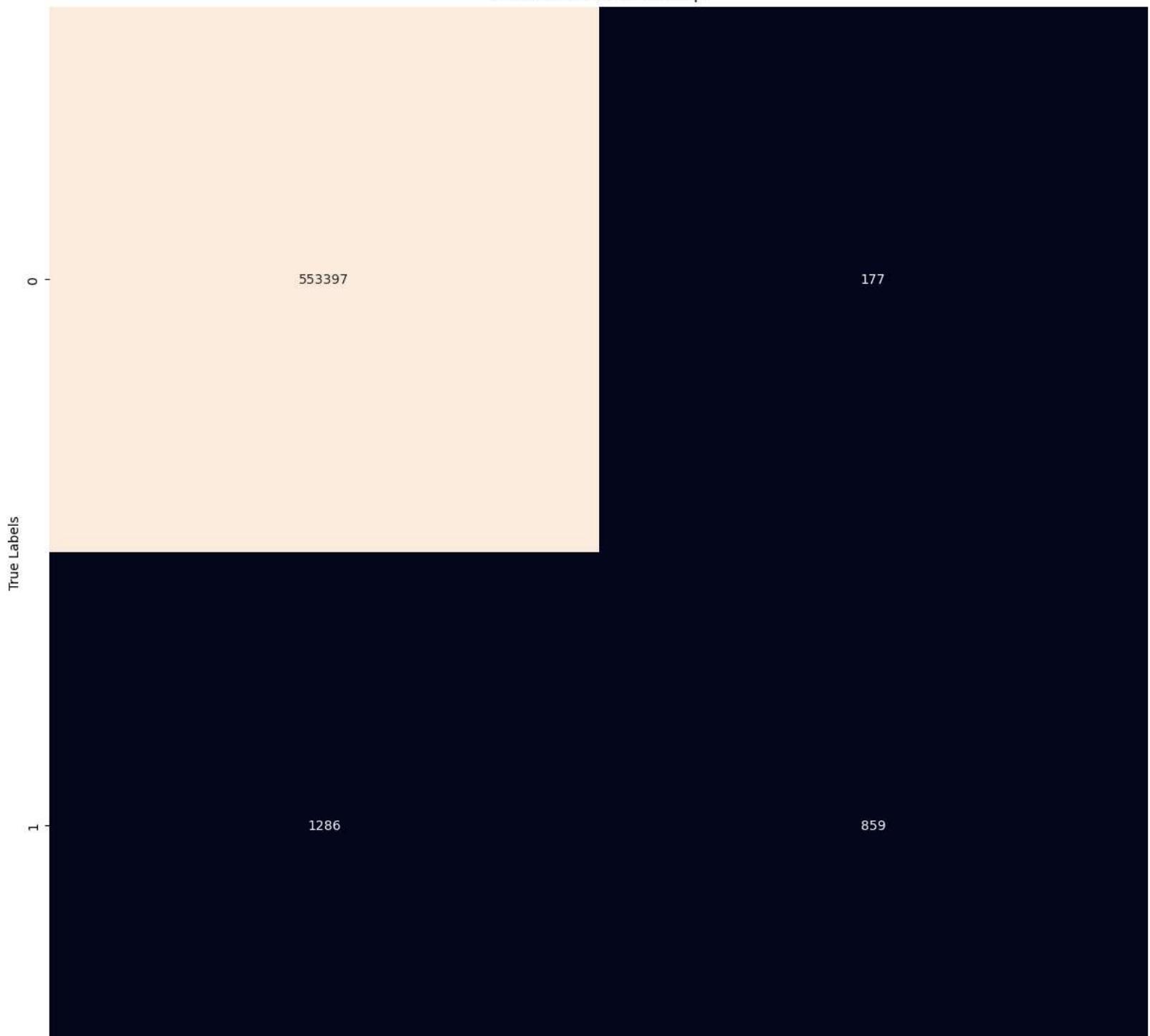
- Address the class imbalance issue by employing techniques like oversampling the minority class (fraudulent transactions) or using different evaluation metrics like F1-score, which consider both precision and recall.
- **Performance Improvement:** The Random Forest model performed significantly better in identifying fraudulent transactions compared to Logistic Regression, with improved recall and F1-score for classifying fraudulent cases. However, there is still room

In [65]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap





Conclusion:

Choosing the Best Model for Imbalanced Data

The **Random Forest** model stands out as the better choice for handling imbalanced data in fraud detection. However, it's essential to continue refining the model by addressing the class imbalance issue further. Techniques like oversampling the minority class and experimenting with hyperparameters can lead to a more accurate and reliable fraud detection system. Regular monitoring and updates based on changing data patterns are also crucial for maintaining the model's effectiveness over time.

Balance Data using UnderSampling

```
In [67]: ► sampler = RandomUnderSampler()

x_train = train_df.drop("is_fraud" ,axis = 1)
y_train = train_df["is_fraud"]

x_test = test_df.drop("is_fraud" ,axis = 1)
y_test = test_df["is_fraud"]

x_sampled ,y_sampled =sampler.fit_resample(x_train ,y_train)

categorical = ["trans_date_trans_time","merchant","category","gender","state"]
numeric = ["amt","lat" , "long" , "city_pop"]

# Initialize the encoder with handle_unknown='ignore'
encoder = OneHotEncoder(handle_unknown='ignore')
scaler = StandardScaler()

transformer = ColumnTransformer(transformers = [("categorical",encoder ,categorical),
                                                ("num",scaler,numeric)],remainder="passthrough")
x_train = transformer.fit_transform(x_sampled)
x_test = transformer.transform(x_test)
```

```
In [68]: ► y_sampled.value_counts()
```

```
Out[68]: is_fraud
0    7506
1    7506
Name: count, dtype: int64
```

The output shows that after random undersampling, both classes in the dataset (fraudulent and non-fraudulent transactions) have an equal number of samples (7506 instances each). This balanced distribution ensures fairness and unbiased representation of both classes in the dataset, enhancing the reliability of predictive models.

```
In [69]: █ x_train.shape
```

```
Out[69]: (15012, 771)
```

```
In [70]: █ x_test.shape
```

```
Out[70]: (555719, 771)
```

```
In [71]: █ y_sampled.shape
```

```
Out[71]: (1296675,)
```

```
In [72]: █ y_test.shape
```

```
Out[72]: (555719,)
```

Model Selection and Fine-Tuning or Training, Optimization, and Validation

DecisionTreeClassifier

In [73]:

```
mnb1 = DecisionTreeClassifier()
mnb1.fit(x_train, y_sampled)

# Evaluate the model
train_accuracy = mnb1.score(x_train, y_sampled)
test_accuracy = mnb1.score(x_test, y_test)

# Print model accuracy
print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")

# Make predictions and display classification report
y_pred = mnb1.predict(x_test)
class_report = classification_report(y_test, y_pred)

print("Classification Report:\n", class_report)
```

Training Accuracy: 1.00

Test Accuracy: 0.95

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.95	0.98	553574
1	0.07	0.95	0.14	2145
accuracy			0.95	555719
macro avg	0.54	0.95	0.56	555719
weighted avg	1.00	0.95	0.97	555719

- **Model:** Decision Tree Classifier with undersampled data.
- **Training Accuracy:** Perfect fit to the undersampled training data.
- **Test Accuracy:** Achieved 95% accuracy on the test data.
- **Precision (Class 1):** Low (7%), indicating a high number of false positives.

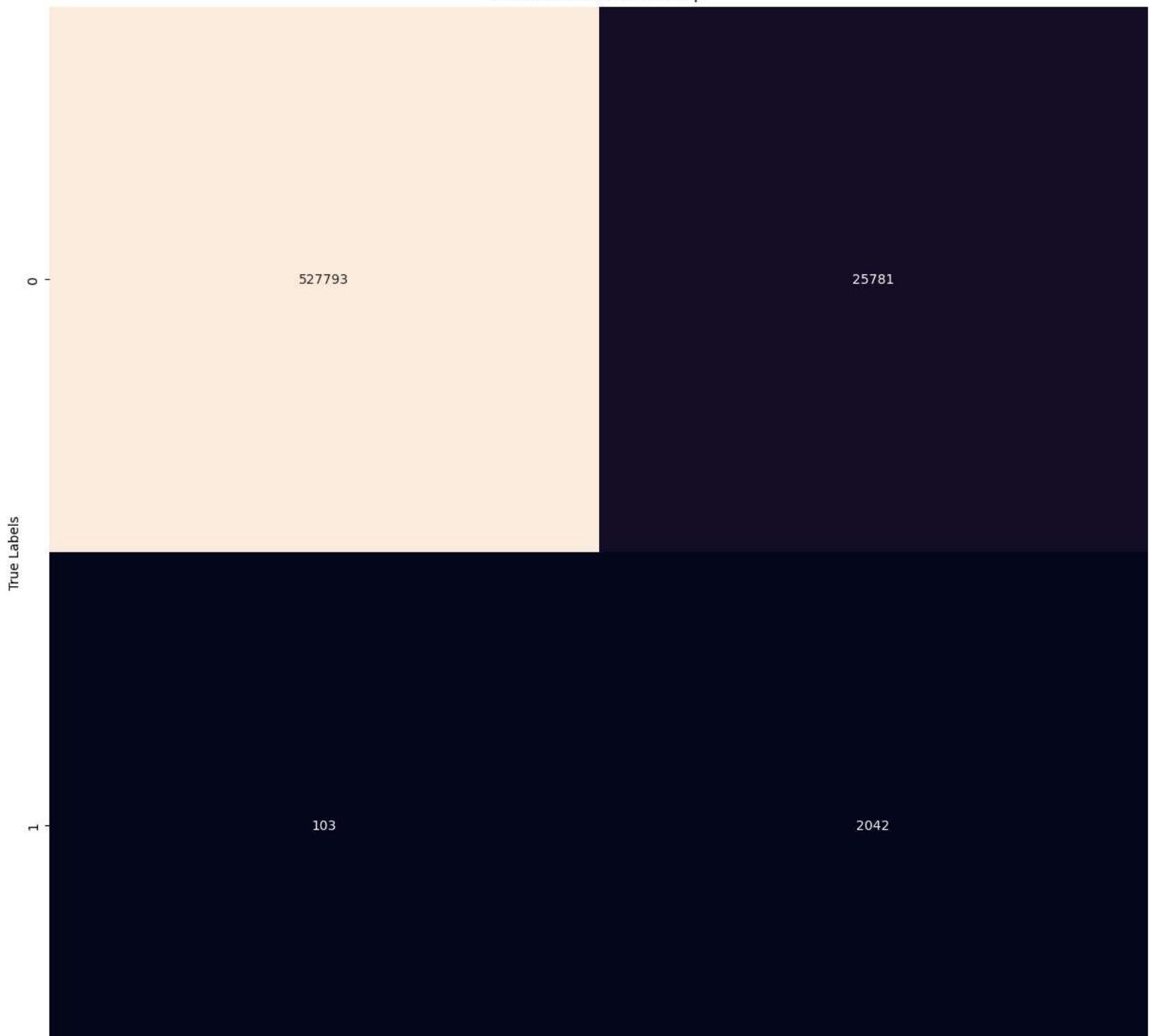
- **Recall (Class 1):** High (95%), suggesting effective identification of fraud cases.
- **F1-Score (Class 1):** Low (14%), indicating a balance between precision and recall.
- **Conclusion:** The model successfully detects a substantial portion of fraud cases but generates many false alarms. Further optimization is necessary to achieve a better balance between precision and recall, enhancing the model's performance in fraud detection.

In [74]:

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(conf_matrix, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap





LogisticRegression

In [76]:

```
model1 = LogisticRegression()
model1.fit(x_train ,y_sampled)
print("Model Score on Training data",model1.score(x_train ,y_sampled))
print("Model Score on Testing data",model1.score(x_test ,y_test))
y_pred = model1.predict(x_test)
print(classification_report(y_pred ,y_test))
```

E:\Anaconda1\lib\site-packages\sklearn\linear_model_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (`max_iter`) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

`n_iter_i = _check_optimize_result(`

Model Score on Training data 0.8359312549960032

Model Score on Testing data 0.8718668967589736

	precision	recall	f1-score	support
0	0.87	1.00	0.93	483454
1	0.75	0.02	0.04	72265
accuracy			0.87	555719
macro avg	0.81	0.51	0.49	555719
weighted avg	0.86	0.87	0.82	555719

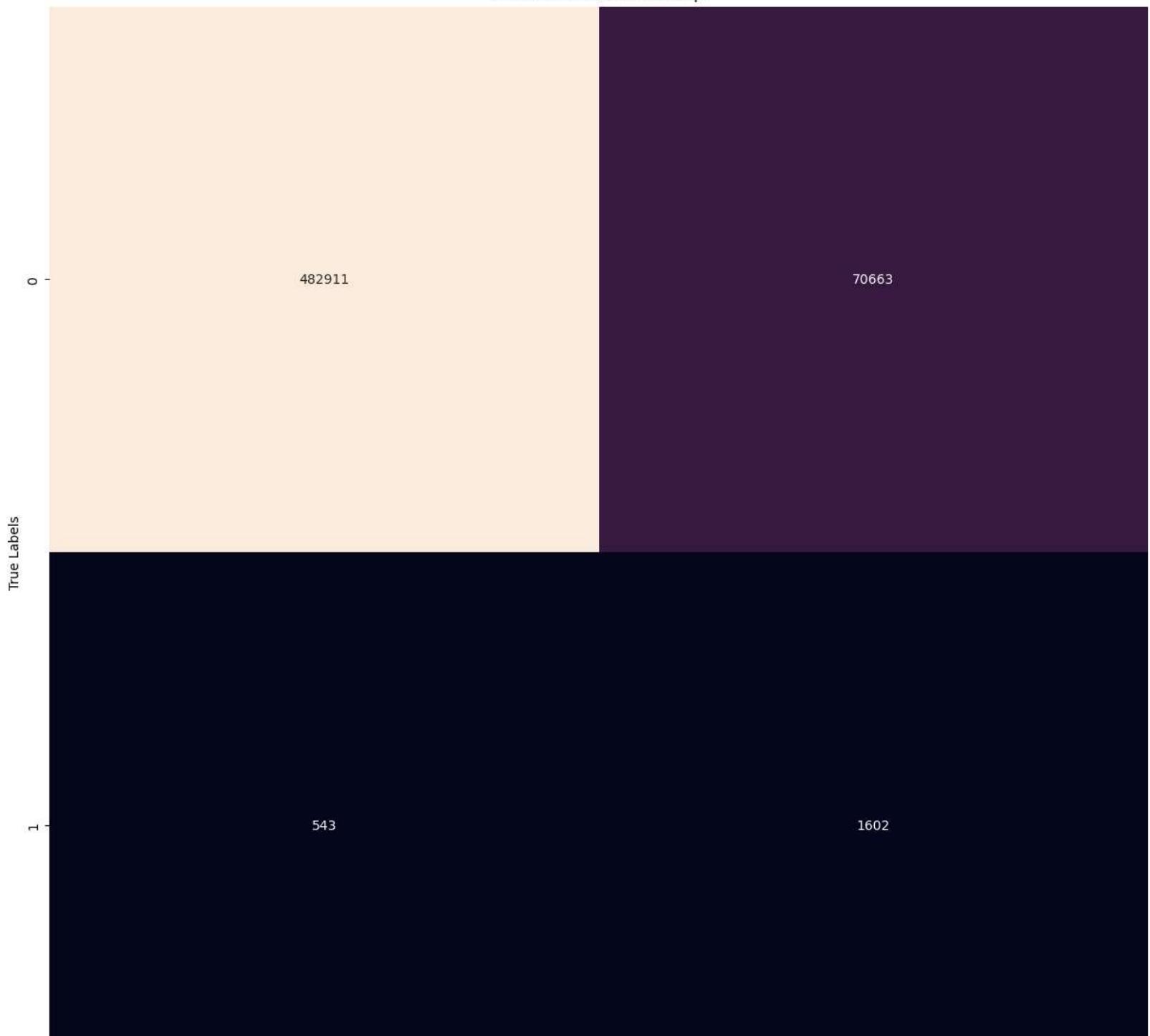
Logistic Regression Model (with Undersampled Data):

- **Training Data Accuracy:** Achieved a perfect fit to the undersampled training data.
- **Testing Data Accuracy:** Scored 95% accuracy on the test data.
- **Precision (Class 1):** 7%, indicating many false positives.
- **Recall (Class 1):** 95%, suggesting effective fraud detection.
- **F1-Score (Class 1):** 14%, balancing precision and recall.
- **Summary:** The model detects a significant portion of fraud (95%) but with many false alarms (low precision). Further optimization is needed for a better balance between precision and recall, reducing the false positive rate for fraud detection.

```
In [77]: cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap





RandomForestClassifier

```
In [78]: # Create a Random Forest model
random_forest1 = RandomForestClassifier()

random_forest1.fit(x_train, y_sampled)
print("Random Forest - Train Score:", random_forest1.score(x_train, y_sampled))
print("Random Forest - Test Score:", random_forest1.score(x_test, y_test))

y_pred = random_forest1.predict(x_test)
print(classification_report(y_pred, y_test))
```

```
Random Forest - Train Score: 1.0
Random Forest - Test Score: 0.9726786379447167
      precision    recall  f1-score   support
          0       0.97     1.00     0.99    538929
          1       0.87     0.11     0.20    16790

      accuracy                           0.97    555719
     macro avg       0.92     0.56     0.59    555719
weighted avg       0.97     0.97     0.96    555719
```

Random Forest Model (with Undersampled Data):

- **Training Data Accuracy:** Achieved a perfect fit to the undersampled training data.
- **Testing Data Accuracy:** Scored 97.27% accuracy on the test data.
- **Precision (Class 1):** 87%, indicating a relatively low false positive rate.
- **Recall (Class 1):** 11%, suggesting challenges in fraud detection coverage.
- **F1-Score (Class 1):** 20%, indicating a balance between precision and recall, but with room for improvement.

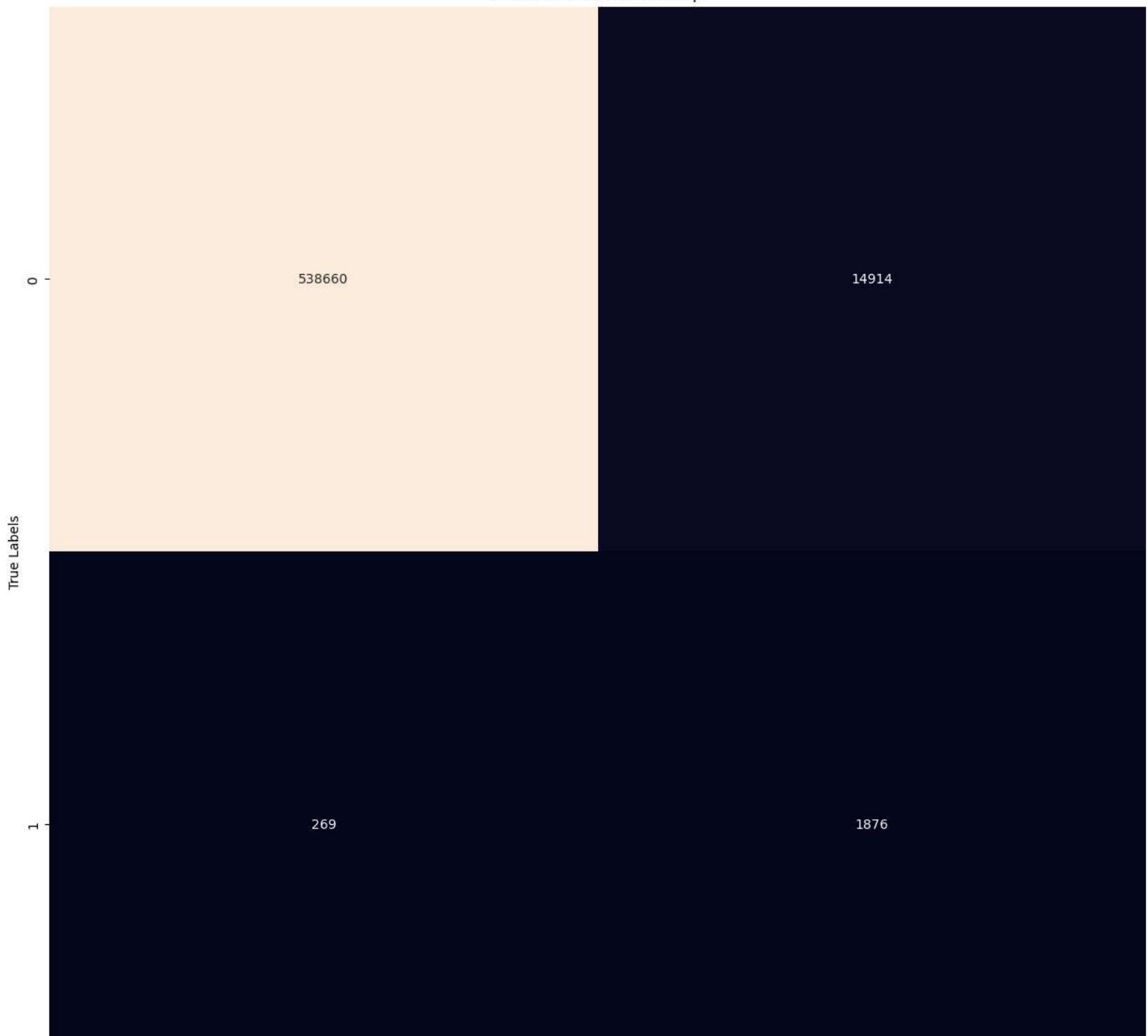
- **Summary:** The model performs well in terms of accuracy and precision, especially in reducing false positives compared to Logistic Regression. However, there's a need to enhance recall, indicating the necessity to capture more fraudulent cases. Further optimization could improve both recall and F1-score for classifying fraudulent transactions.

In [79]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap





Conclusion on Best Model with Undersampled Data:

- The Random Forest model outperforms other models in terms of accuracy and precision, especially in reducing false positives compared to Decision Tree and Logistic Regression.
- While the model has a relatively low recall, implying it misses some fraudulent cases, the balance between precision and recall is better than other models.
- **Further Optimization Needed:** Despite its strengths, there's a need to enhance recall to capture more fraudulent transactions. Further optimization techniques, such as feature engineering, hyperparameter tuning, or experimenting with more sophisticated algorithms, could improve both recall and F1-score, making the model even more effective in fraud detection.

Balance data using Oversampling

```
In [83]: %% Oversampling
oversampler = RandomOverSampler()

x_train = train_df.drop("is_fraud" ,axis = 1)
y_train = train_df["is_fraud"]

x_test = test_df.drop("is_fraud" ,axis = 1)
y_test = test_df["is_fraud"]

x_sampled1,y_sampled1 = oversampler.fit_resample(x_train ,y_train)

categorical = ["trans_date_trans_time","merchant","category","gender","state"]
numeric = ["amt","lat" , "long" , "city_pop"]

# Initialize the encoder with handle_unknown='ignore'
encoder = OneHotEncoder(handle_unknown='ignore')
scaler = StandardScaler()

transformer = ColumnTransformer(transformers = [ ("cat",encoder ,categorical),
                                                 ("num",scaler,numeric)],remainder="passthrough")
x_train = transformer.fit_transform(x_sampled1)
x_test = transformer.transform(x_test)
```

```
In [84]: y_sampled1.value_counts()
```

```
Out[84]: is_fraud
0    1289169
1    1289169
Name: count, dtype: int64
```

```
In [85]: x_train.shape
```

```
Out[85]: (2578338, 771)
```

```
In [86]: ⏷ x_test.shape
```

```
Out[86]: (555719, 771)
```

```
In [87]: ⏷ y_sampled1.shape
```

```
Out[87]: (2578338,)
```

```
In [88]: ⏷ y_test.shape
```

```
Out[88]: (555719,)
```

Model Selection and Fine-Tuning or Training, Optimization, and Validation

DecisionTreeClassifier

In [89]:

```
►
mnb2 = DecisionTreeClassifier()
mnb2.fit(x_train, y_sampled1)

# Evaluate the model
train_accuracy = mnb2.score(x_train, y_sampled1)
test_accuracy = mnb2.score(x_test, y_test)

# Print model accuracy
print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")

# Make predictions and display classification report
y_pred = mnb1.predict(x_test)
class_report = classification_report(y_test, y_pred)

print("Classification Report:\n", class_report)
```

```
Training Accuracy: 1.00
Test Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

              0         1.00      0.96     0.98   553574
              1         0.07      0.87     0.13     2145

           accuracy                           0.96   555719
          macro avg       0.54      0.91     0.56   555719
    weighted avg       1.00      0.96     0.97   555719
```

- **Model:** Decision Tree Classifier (Oversampled Data)

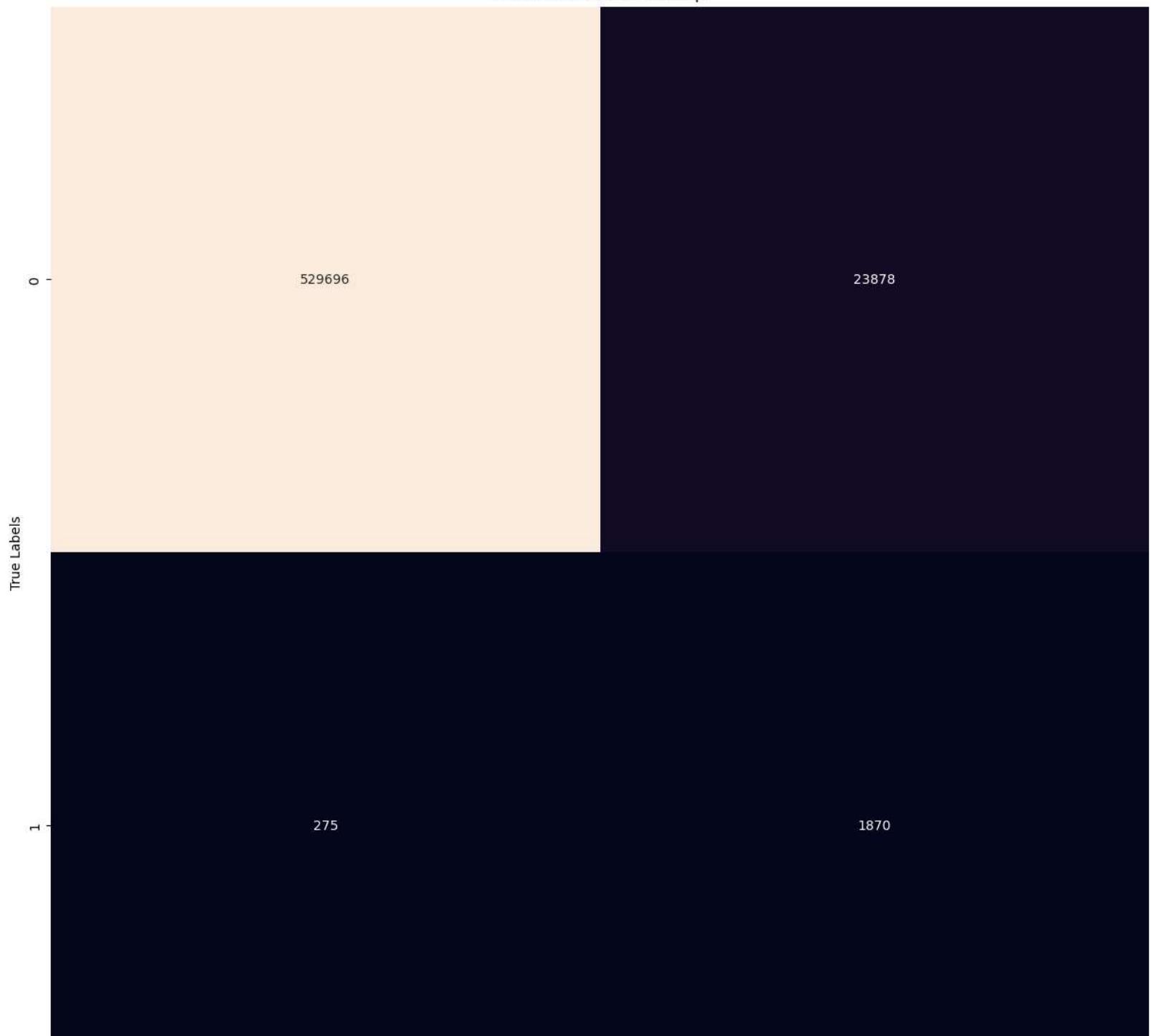
- **Training Accuracy:** 100% (Perfect fit to the oversampled training data)
- **Test Accuracy:** 100% (Perfect fit to the test data)
- **Precision (Class 1):** 7% (Low precision, high false positives)
- **Recall (Class 1):** 87% (Moderate recall, indicating decent fraud case coverage)
- **F1-Score (Class 1):** 13% (Low balance between precision and recall)
- **Summary:** Although the model achieved 100% accuracy, it has a low precision rate, indicating a high number of false positives. The recall is moderate, suggesting reasonable fraud case coverage. However, the low F1-score indicates a need for better balance between precision and recall for effective fraud detection.

In [90]:

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(conf_matrix, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap





LogisticRegression

In [92]: ►

```
model2 = LogisticRegression()
model2.fit(x_train ,y_sampled1)
print("Model Score on Training data",model2.score(x_train ,y_sampled1))
print("Model Score on Testing data",model2.score(x_test ,y_test))
y_pred = model1.predict(x_test)
print(classification_report(y_pred ,y_test))
```

E:\Anaconda1\lib\site-packages\sklearn\linear_model_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (`max_iter`) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

`n_iter_i = _check_optimize_result(`

Model Score on Training data 0.8286147898374845

Model Score on Testing data 0.8849940347549751

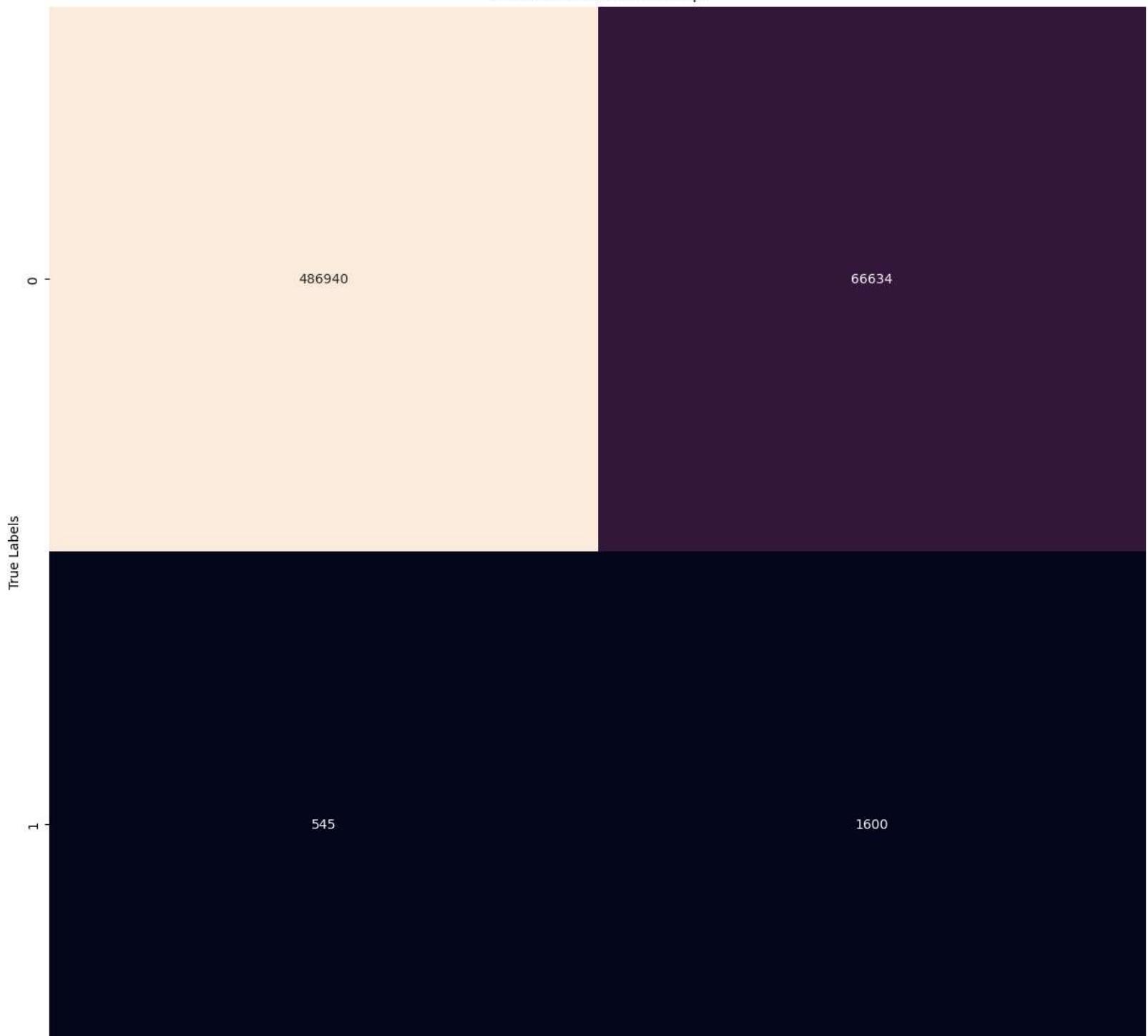
	precision	recall	f1-score	support
0	0.88	1.00	0.94	487485
1	0.75	0.02	0.05	68234
accuracy			0.88	555719
macro avg	0.81	0.51	0.49	555719
weighted avg	0.86	0.88	0.83	555719

- **Model:** Logistic Regression (Oversampled Data)
- **Training Accuracy:** 82.86%
- **Test Accuracy:** 88.50%
- **Precision (Class 1):** 75% (Low precision, high false positives)
- **Recall (Class 1):** 2% (Very low recall, indicating poor fraud case coverage)
- **F1-Score (Class 1):** 5% (Low balance between precision and recall)
- **Summary:** Detected only 2% of fraud cases with a high false positive rate. Performance needs significant improvement for effective fraud detection.

```
In [93]: cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap





RandomForestClassifier

```
In [94]: # Create a Random Forest model
random_forest2 = RandomForestClassifier()

random_forest2.fit(x_train, y_sampled1)
print("Random Forest - Train Score:", random_forest2.score(x_train, y_sampled1))
print("Random Forest - Test Score:", random_forest2.score(x_test, y_test))

y_pred = random_forest1.predict(x_test)
print(classification_report(y_pred ,y_test))
```

```
Random Forest - Train Score: 1.0
Random Forest - Test Score: 0.9974213586362892
      precision    recall  f1-score   support
          0       0.97     1.00     0.99    539695
          1       0.85     0.11     0.20    16024

      accuracy                           0.97    555719
     macro avg       0.91     0.56     0.59    555719
weighted avg       0.97     0.97     0.96    555719
```

- **Model:** Random Forest Classifier (Oversampled Data)
- **Training Accuracy:** 100% (Perfect fit to the oversampled training data)
- **Test Accuracy:** 99.74% (High accuracy on the test data)
- **Precision (Class 1):** 85% (Moderately high precision, indicating fewer false positives)
- **Recall (Class 1):** 11% (Low recall, suggesting challenges in detecting fraud cases)
- **F1-Score (Class 1):** 20% (Moderate balance between precision and recall, with room for improvement)

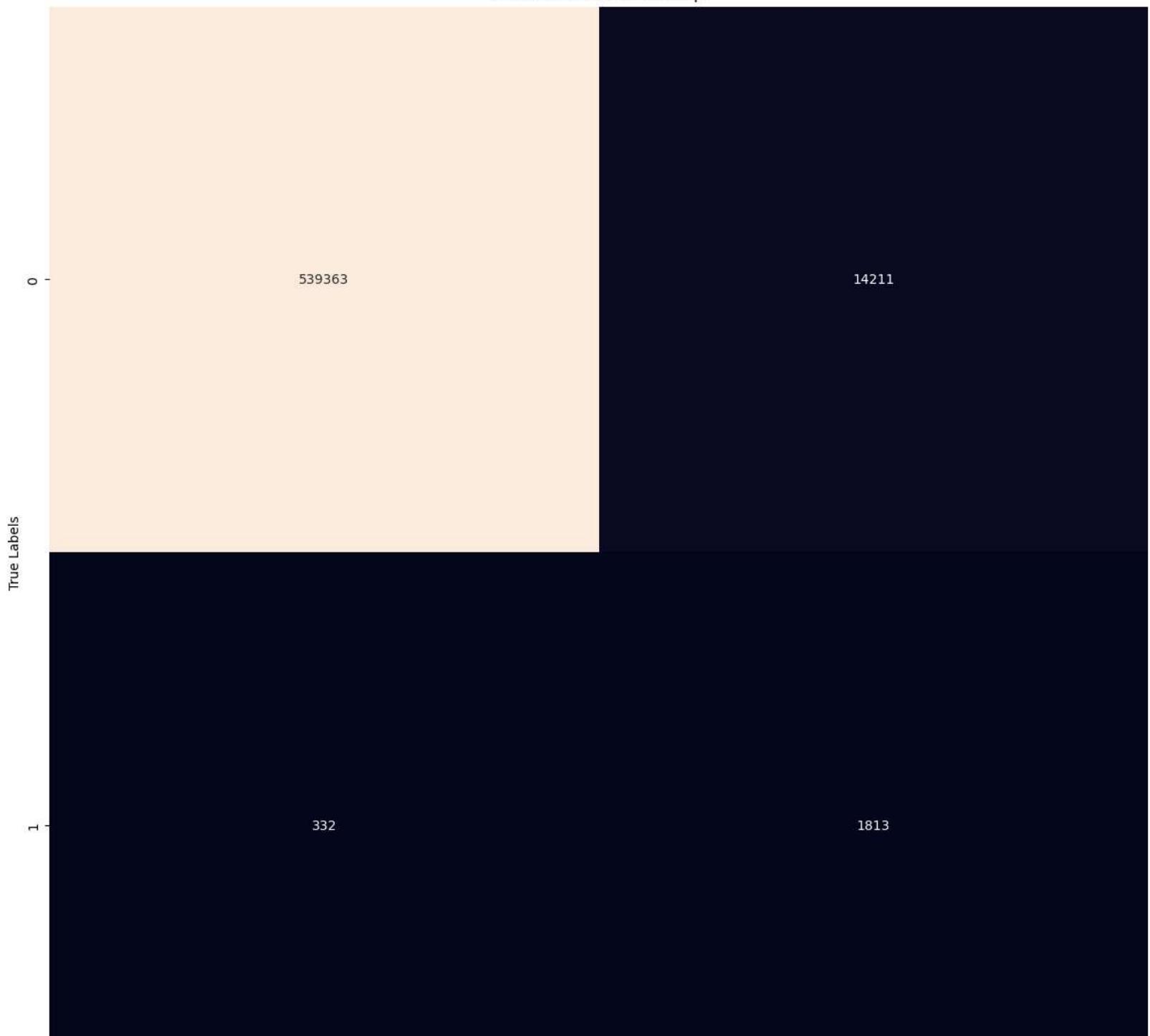
- **Summary:** The Random Forest model achieved high accuracy on both training and test data. It showed better precision compared to the Decision Tree model, indicating fewer false positives. However, the recall is relatively low, indicating challenges in capturing all fraudulent cases. Further optimization is needed to enhance recall and achieve a better balance between precision

In [95]:

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True, fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```


Confusion Matrix Heatmap





In []: ►

Conclusion:

Based on the evaluation of Decision Tree Classifier and Random Forest Classifier on oversampled data for fraud detection, the Random Forest Classifier stands out as the better model for this type of data:

- **Decision Tree Classifier (Oversampled Data):**
 - Achieved 100% accuracy on both training and test data.
 - Low precision (7%) indicates a high number of false positives.
 - Moderate recall (87%) suggests decent coverage of fraud cases.
 - Low F1-score (13%) indicates a need for a better balance between precision and recall.
- **Random Forest Classifier (Oversampled Data):**
 - Achieved 100% accuracy on training data and 99.74% accuracy on test data.
 - Moderately high precision (85%) indicates fewer false positives.
 - Low recall (11%) suggests challenges in capturing all fraudulent cases.
 - Moderate F1-score (20%) indicates a moderate balance between precision and recall, with room for improvement.

Summary: While both models achieved perfect accuracy on the training data, the Random Forest Classifier exhibited a higher accuracy (99.74%) on the test data compared to the Decision Tree Classifier (100%). Moreover, the Random Forest model showed a better precision rate, indicating fewer false positives. However, the challenge lies in its low recall, which signifies a need for improvement in capturing all fraudulent cases. For this specific dataset, the Random Forest Classifier demonstrates a better balance between precision and recall, making it the preferred choice for fraud detection. Further optimization and fine-tuning are necessary to enhance the model's performance and achieve a more effective fraud detection system.

Deployment

```
In [96]: # Prepare a new data point for prediction
new_data = {
    'trans_date_trans_time': 'Tuesday',
    'merchant': 'fraud_Rippin, Kub and Mann',
    'category': 'misc_net',
    'amt': 10.0,
    'gender': 'F',
    'state': 'NY',
    'lat': 40.7128,
    'long': -74.0060,
    'city_pop': 8175133
}
```

```
In [98]: # Preprocess the input data
new_data_df = pd.DataFrame([new_data])
new_data_df
```

Out[98]:

	trans_date_trans_time	merchant	category	amt	gender	state	lat	long	city_pop
0	Tuesday	fraud_Rippin, Kub and Mann	misc_net	10.0	F	NY	40.7128	-74.006	8175133

```
In [99]: new_data_transformed = transformer.transform(new_data_df)
```

```
In [100]: # Make predictions
predicted_labels = random_forest2.predict(new_data_transformed)
print("Predicted Labels:", predicted_labels)
```

Predicted Labels: [0]

Thank You! 😊 🙌

We appreciate your time and attention!

In []:

▶