Load necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import opendatasets as od

from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection  import chi2, SelectKBest
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score,confusion_matrix, c
from imblearn.over_sampling import SMOTE

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [2]:
```
#Download the creditcard fraud dataset from Kaggle
#od.download("https://www.kaggle.com/datasets/kartik2112/fraud-detection?select=fraudTra
#od.download("https://www.kaggle.com/datasets/kartik2112/fraud-detection?select=fraudTes
```

Dataset is already split into test and train sets. We will combine them and redo the train-test split

In [3]:
```
fraud_train_df = pd.read_csv("fraud-detection/fraudTrain.csv")
fraud_train_df.head(5)
```

Out[3]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	g
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	Jeremy	White	
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia	

5 rows × 23 columns

In [4]:
```
fraud_test_df = pd.read_csv("fraud-detection/fraudTest.csv")
fraud_test_df.head(5)
```

Out[4]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	g
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	Jeff	Elliott	
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	Joanne	Williams	
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	Ashley	Lopez	
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	Brian	Williams	
4	4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.19	Nathan	Massey	

5 rows × 23 columns

In [5]:
```
#Combine the 2 datasets
fraud_df = pd.concat([fraud_train_df, fraud_test_df], axis=0)
```

In [6]:
```
fraud_train_df.shape, fraud_test_df.shape, fraud_df.shape
```

Out[6]:
```
((1296675, 23), (555719, 23), (1852394, 23))
```

In [7]:
```
fraud_df.head(5)
```

Out[7]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	g
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	Jeremy	White	

4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia	

5 rows × 23 columns

Data Processing

Null rows check

In [8]:
```
fraud_df.isnull().sum()
```

Out[8]:
```
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
```

Check for duplicates

In [9]:
```
print('Dataframe before dropping duplicates :', fraud_df.shape)
fraud_df = fraud_df.drop_duplicates() # 1,389 rows dropped
print('Dataframe after dropping duplicates :',fraud_df.shape)
```

```
Dataframe before dropping duplicates : (1852394, 23)
Dataframe after dropping duplicates : (1852394, 23)
```

In [10]:
```
fraud_df.duplicated().sum()
```

Out[10]:
```
0
```

Convert column data type for DateTime

In [11]:
```
fraud_df['trans_date_time'] = pd.to_datetime(fraud_df['trans_date_trans_time'])
fraud_df.head(5)
```

Out[11]:

	Unnamed:	trans_date_trans_time	cc_num	merchant	category	amt	first	last	g

	0								
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	Jeremy	White
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia

5 rows × 24 columns

Add columns

In [12]:
```
fraud_df['month'] = pd.DatetimeIndex(fraud_df['trans_date_time']).month
```

Drop Columns

In [13]:
```
fraud_df.columns
```

Out[13]:
```
Index(['Unnamed: 0', 'trans_date_trans_time', 'cc_num', 'merchant', 'category',
       'amt', 'first', 'last', 'gender', 'street', 'city', 'state', 'zip',
       'lat', 'long', 'city_pop', 'job', 'dob', 'trans_num', 'unix_time',
       'merch_lat', 'merch_long', 'is_fraud', 'trans_date_time', 'month'],
      dtype='object')
```

In [14]:
```
fraud_df = fraud_df.drop(['Unnamed: 0','trans_date_trans_time'], axis=1)
```

In [15]:
```
fraud_df.columns
```

Out[15]:
```
Index(['cc_num', 'merchant', 'category', 'amt', 'first', 'last', 'gender',
       'street', 'city', 'state', 'zip', 'lat', 'long', 'city_pop', 'job',
       'dob', 'trans_num', 'unix_time', 'merch_lat', 'merch_long', 'is_fraud',
       'trans_date_time', 'month'],
      dtype='object')
```

Convert data types to reduce memory usage

In [16]:
```
fraud_df.dtypes
```

Out[16]:
```
cc_num                     int64
merchant                  object
category                  object
amt                      float64
```

```
first                      object
last                       object
gender                     object
street                     object
city                       object
state                      object
zip                         int64
lat                       float64
long                      float64
city_pop                    int64
job                        object
dob                        object
trans_num                  object
unix_time                   int64
merch_lat                 float64
merch_long                float64
is_fraud                    int64
trans_date_time    datetime64[ns]
month                       int32
dtype: object
```

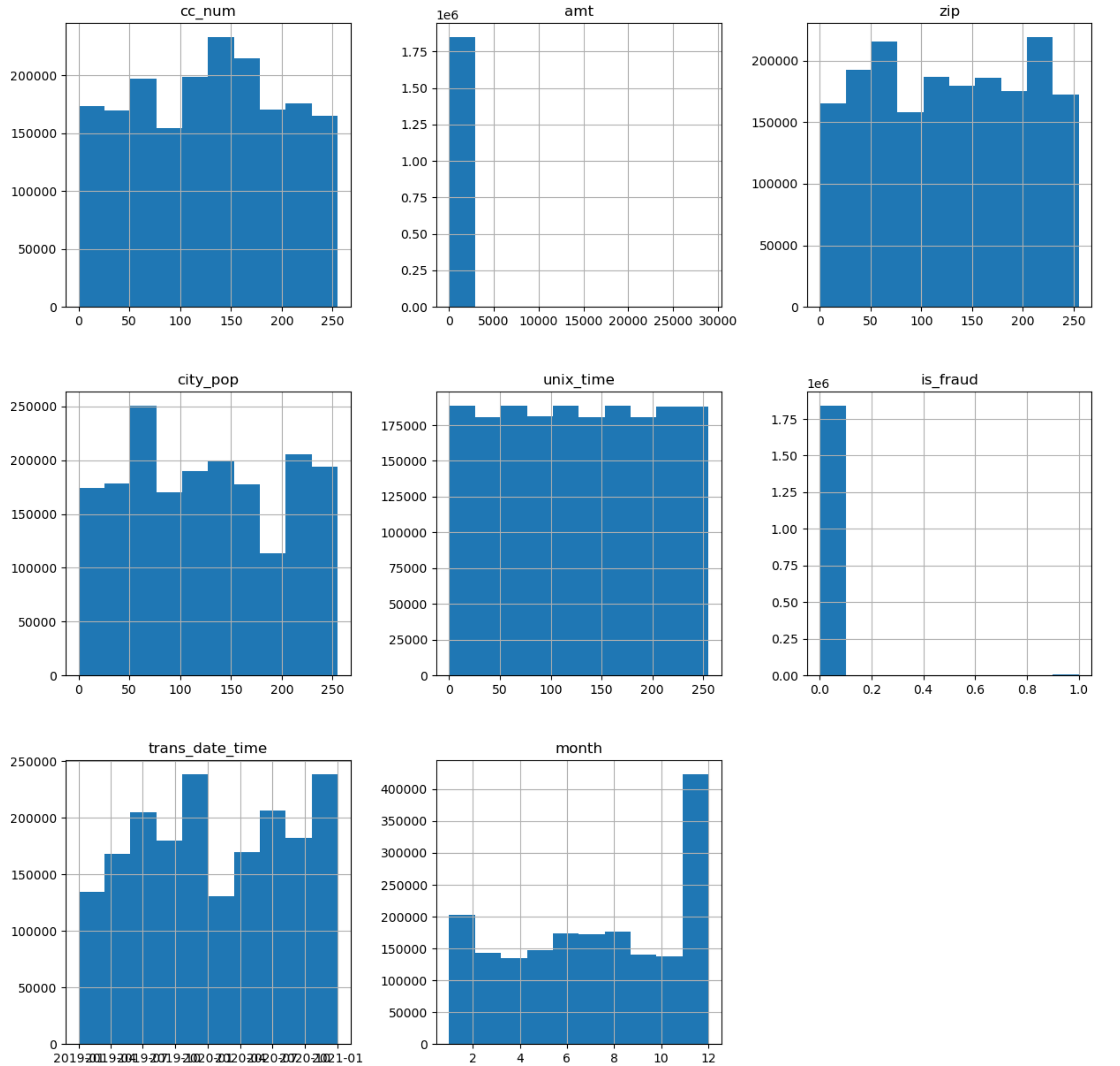
In [17]:

```
#Converting data types to avoid memory issues while executing the model fit.
cols=['amt']
fraud_df[cols] = fraud_df[cols].astype('float16') #Converting float64 to float16

int_cols = ['cc_num','zip','city_pop','unix_time','is_fraud','month']
fraud_df[int_cols] = fraud_df[int_cols].astype(np.uint8)  #Converting int64 to uint8

obj_cols = ['merchant','category','first','last','gender','street','city','state','job',
fraud_df[obj_cols] = fraud_df[obj_cols].astype('category') # #Converting object to categ
```

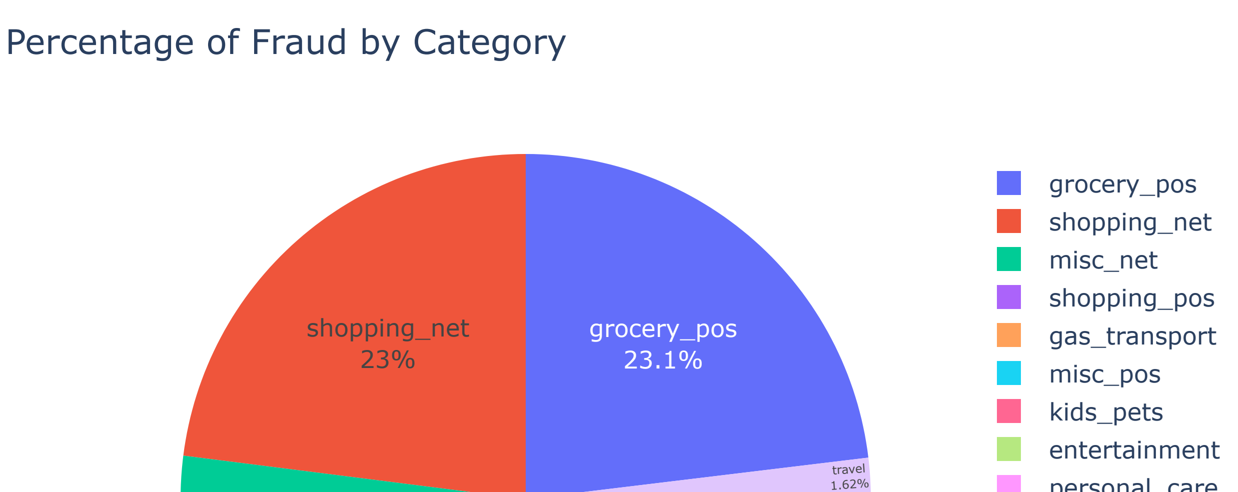
Data Visualization

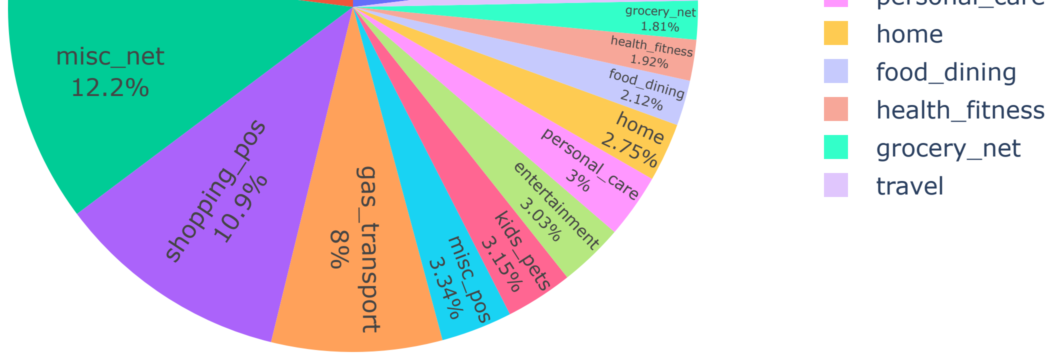
In [18]:

```
fraud_df.hist(figsize = (15,15))
plt.show()
```



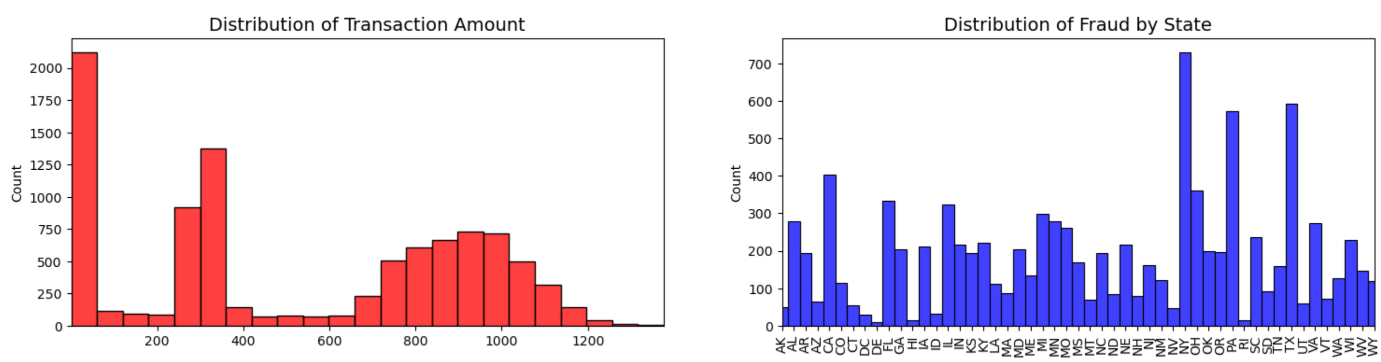
In [19]:
```
fig = px.pie(fraud_df[fraud_df.is_fraud==1], values='is_fraud', names='category', title=
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.update_layout(title = "Percentage of Fraud by Category")
fig.show("notebook")
```





In [20]:

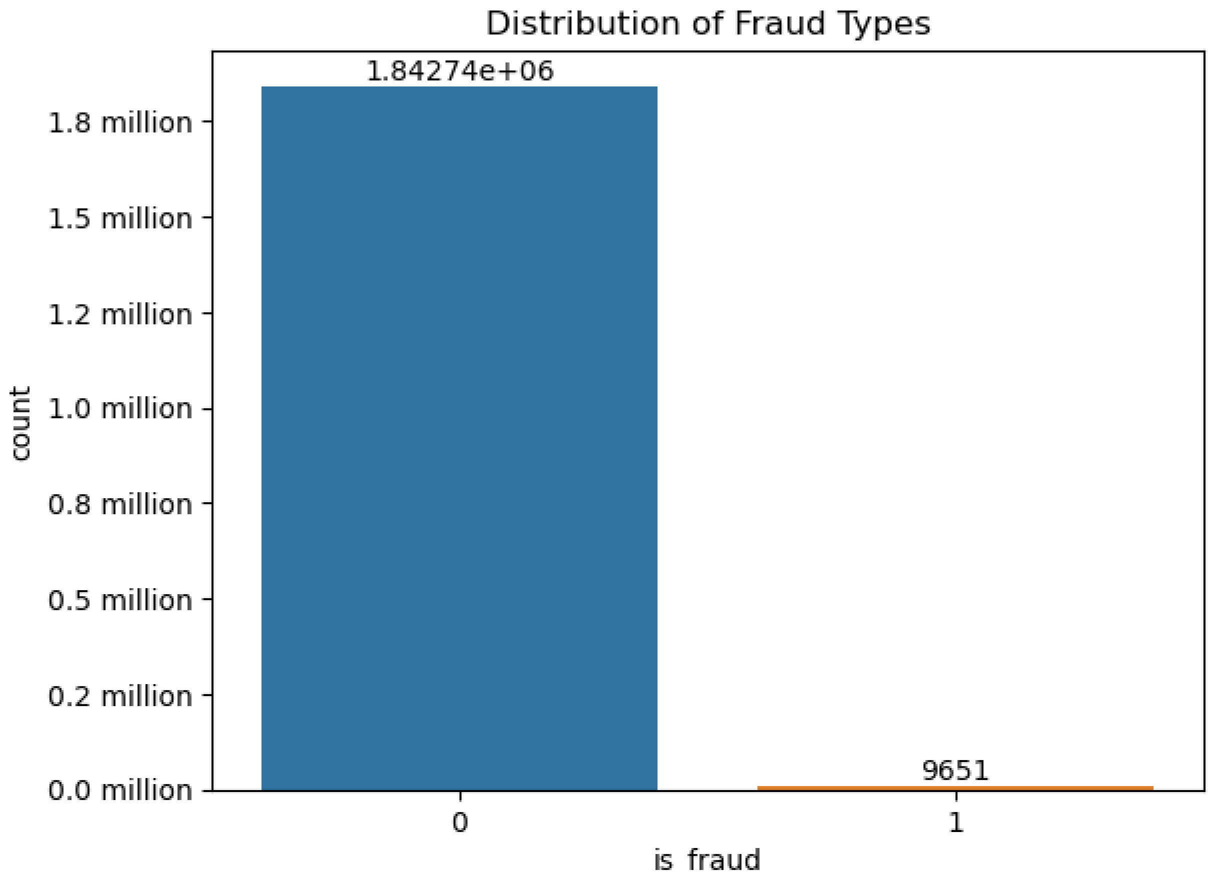
```
fig, ax = plt.subplots(1, 2, figsize=(18,4))

amount_val = fraud_df[ fraud_df.is_fraud == 1].amt.values.astype(int)
time_val = fraud_df[ fraud_df.is_fraud == 1].state.values

sns.histplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)
ax[0].set_xlim([min(amount_val), max(amount_val)])

sns.histplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Fraud by State', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])
ax[1].tick_params(axis='x', rotation=90)

plt.show()
```



Bar Plot to check data balance

In [21]:

```
xx = fraud_df['is_fraud'].value_counts().reset_index()
def formatter(x, pos):
    return str(round(x / 1e6, 1)) + " million"

ax = sns.barplot(x="is_fraud",y="count",data=xx)
ax.set_title('Distribution of Fraud Types')
ax.yaxis.set_major_formatter(formatter)
ax.yaxis.set_minor_formatter(NullFormatter())
for i in ax.containers:
    ax.bar_label(i,)
```



We can see that the data is not balanced. The number of fraudulent transactions in the dataset are very low in comparison to the legitimate transactions. Building models with this data could give inaccurate results.

Model Building

SMOTE helps to balance the class distribution by generating synthetic samples of the minority class.

Data leakage happens when information from the validation or test set unintentionally leaks into the training set, leading to overly optimistic performance estimates. When using SMOTE, this risk exists because synthetic samples are generated based on the original data. By incorporating SMOTE within the cross-validation process, you mitigate the risk of data leakage, as the synthetic samples are only used within each fold of the cross-validation.

In [22]:
```
fraud_df.replace(np.nan,0)
```

Out[22]:

	cc_num	merchant	category	amt	first	last	gender	street	city	st
0	127	fraud_Rippin, Kub and Mann	misc_net	4.968750	Jennifer	Banks	F	561 Perry Cove	Moravian Falls	
1	106	fraud_Heller, Gutmann and Zieme	grocery_pos	107.250000	Stephanie	Gill	F	43039 Riley Greens Suite 393	Orient	
2	61	fraud_Lind-Buckridge	entertainment	220.125000	Edward	Sanchez	M	594 White Dale Suite 530	Malad City	
3	16	fraud_Kutch, Hermiston and Farrell	gas_transport	45.000000	Jeremy	White	M	9443 Cynthia	Boulder	

								Court Apt. 038	
4	176	fraud_Keeling-Crist	misc_pos	41.968750	Tyler	Garcia	M	408 Bradley Rest	Doe Hill
...	...	...	...	...	...	...	...	...	...
555714	169	fraud_Reilly and Sons	health_fitness	43.781250	Michael	Olson	M	558 Michael Estates	Luray
555715	40	fraud_Hoppe-Parisian	kids_pets	111.812500	Jose	Vasquez	M	572 Davis Mountains	Lake Jackson
555716	230	fraud_Rau-Robel	kids_pets	86.875000	Ann	Lawson	F	144 Evans Islands Apt. 683	Burbank
555717	150	fraud_Breitenberg LLC	travel	7.988281	Eric	Preston	M	7020 Doyle Stream Apt. 951	Mesa
555718	187	fraud_Dare-Marvin	entertainment	38.125000	Samuel	Frey	M	830 Myers Plaza Apt. 384	Edmond

1852394 rows × 23 columns

In [23]:
```
# Split the dataset into train and test sets
#X = fraud_df.drop(['is_fraud','lat','long','merch_lat','merch_long'],axis=1)
X = fraud_df.drop(['is_fraud'],axis=1)
Y = fraud_df['is_fraud']
X.shape, Y.shape
```

Out[23]:
```
((1852394, 22), (1852394,))
```

In [24]:
```
# Encode categorical variables (e.g., 'gender', 'category', 'state', etc.)
categorical_columns = [ 'merchant', 'category','first', 'last','gender','street', 'city'
for col in categorical_columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
```

In [25]:
```
# Standardize numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [26]:
```
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_state=42)
```

In [27]:
```
X_train.shape, Y_train.shape
```

Out[27]:
```
((1481915, 22), (1481915,))
```

In [28]:
```
# Apply SMOTE to balance the dataset
smote = SMOTE(sampling_strategy='auto', random_state=42)
x_train_resampled, y_train_resampled = smote.fit_resample(X_train, Y_train)
```

In [29]:
```
print('x_train Data Shape   : ', X_train.shape)
print('y_train Labels Shape : ', Y_train.shape)
print('x_train_resampled Data Shape   : ', x_train_resampled.shape)
```

```
print('y_train_resampled Labels Shape : ', y_train_resampled.shape)
print('x_test Data Shape     : ', X_test.shape)
print('y_test Labels Shape   : ', Y_test.shape)
```

```
x_train Data Shape    :  (1481915, 22)
y_train Labels Shape :  (1481915,)
x_train_resampled Data Shape    :  (2948434, 22)
y_train_resampled Labels Shape :  (2948434,)
x_test Data Shape     :  (370479, 22)
y_test Labels Shape   :  (370479,)
```

In [30]:

```
#Adding this step to clear memory, to avoid memory issues during execution
import gc

gc.collect()
```

Out[30]:

```
1137
```

Models

RandomForestClassifier

In [31]:

```
# Use the RandomForestClassifier to fit balanced data
rfc = RandomForestClassifier()
rfc_model = rfc.fit(x_train_resampled,y_train_resampled)

#Predict y data with classifier:
y_pred_rfc = rfc_model.predict(X_test)

# Evaluate the model
print(classification_report(Y_test, y_pred_rfc))
print(confusion_matrix(Y_test, y_pred_rfc))
print(f'ROC-AUC score : {roc_auc_score(Y_test, y_pred_rfc)}')
print(f'Accuracy score : {accuracy_score(Y_test, y_pred_rfc)}')
```

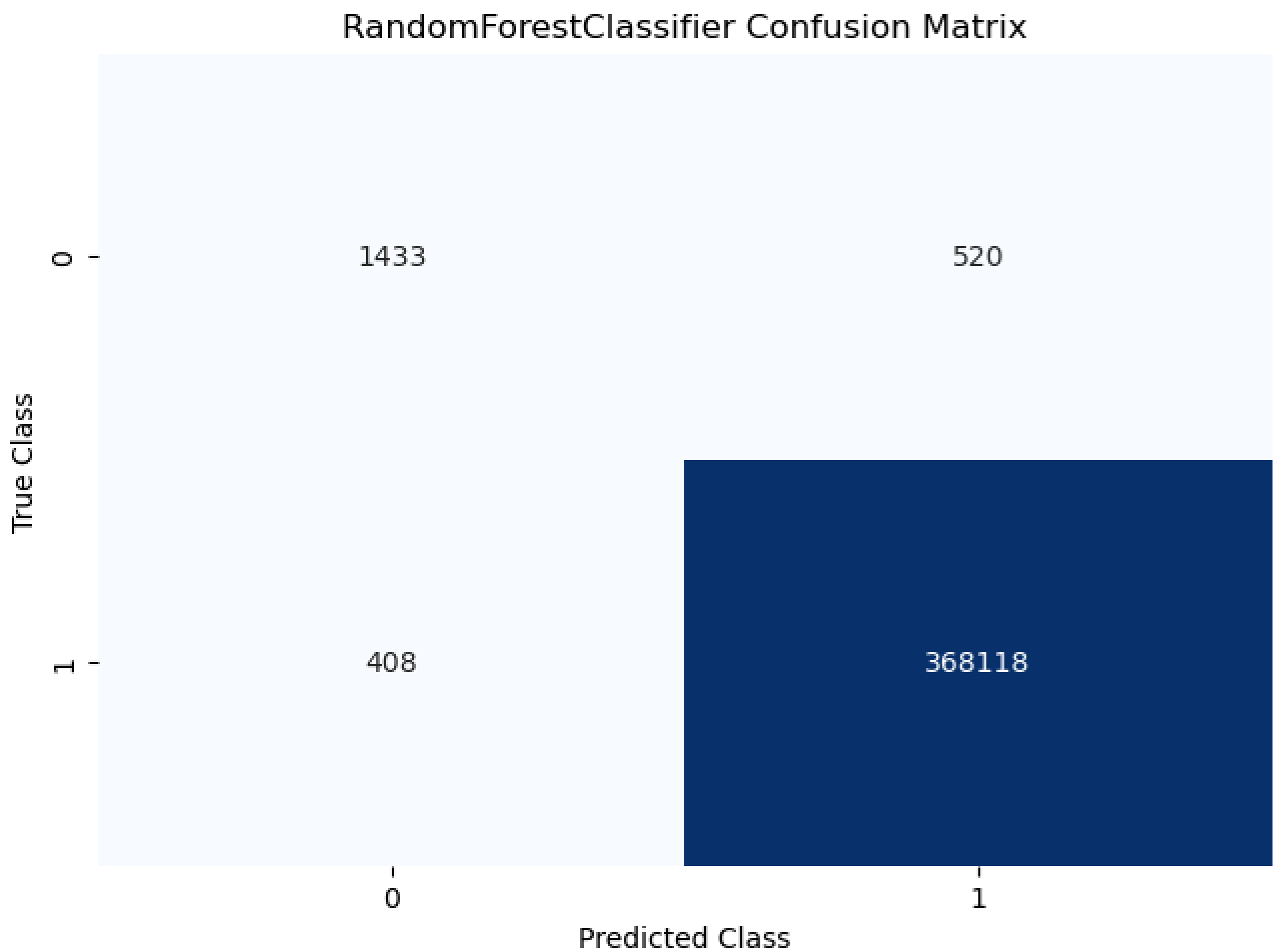
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    368526
           1       0.78      0.73      0.76      1953

    accuracy                           1.00    370479
   macro avg       0.89      0.87      0.88    370479
weighted avg       1.00      1.00      1.00    370479

[[368118    408]
 [   520   1433]]
ROC-AUC score : 0.8663179231735431
Accuracy score : 0.9974951346769992
```

In [32]:

```
#Build the confusion matrix
matrix = confusion_matrix(Y_test, y_pred_rfc, labels=[1,0])

print(matrix)

# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("RandomForestClassifier Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[  1433    520]
```

```
 [   408 368118]]
```



Logistic Regression

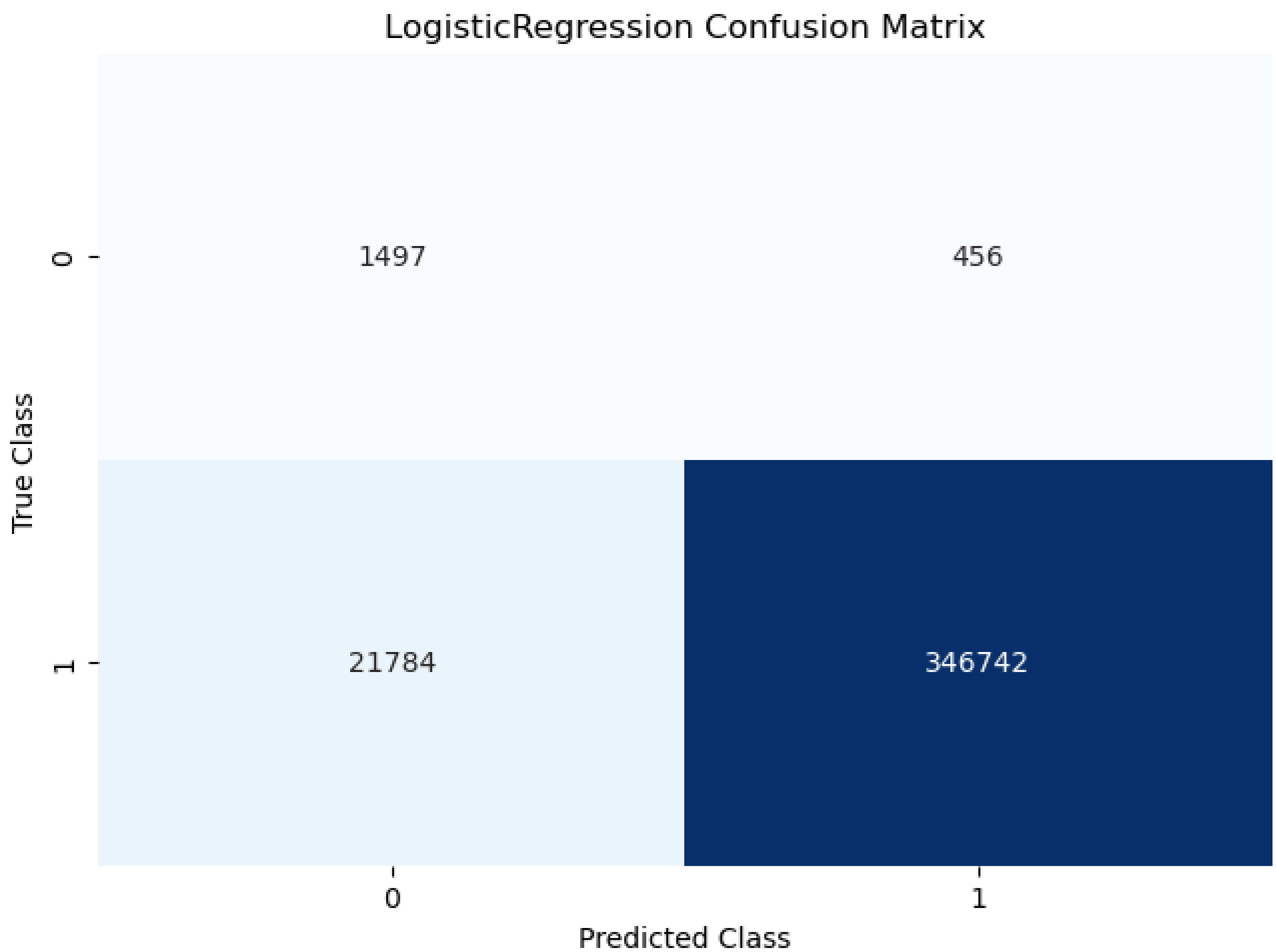
In [33]:
```
# Train a logistic regression model
logistic_model = LogisticRegression(solver='liblinear', random_state=42)
logistic_model.fit(x_train_resampled,y_train_resampled)

# Make predictions on the test set
y_pred_lr = logistic_model.predict(X_test)

# Evaluate the model
print(classification_report(Y_test, y_pred_lr))
print(confusion_matrix(Y_test, y_pred_lr))
print(f'ROC-AUC score : {roc_auc_score(Y_test, y_pred_lr)}')
print(f'Accuracy score : {accuracy_score(Y_test, y_pred_lr)}')
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97    368526
           1       0.06      0.77      0.12      1953

    accuracy                           0.94    370479
   macro avg       0.53      0.85      0.54    370479
weighted avg       0.99      0.94      0.96    370479

[[346742  21784]
 [   456   1497]]
ROC-AUC score : 0.8537009475361442
Accuracy score : 0.939969606914292
```

In [34]:
```
#Build the confusion matrix
matrix = confusion_matrix(Y_test, y_pred_lr, labels=[1,0])
```

```
print(matrix)

# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("LogisticRegression Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[  1497    456]
 [ 21784 346742]]
```



Using class_weight='balaced' to check if the imbalance get's any better.

In [35]:

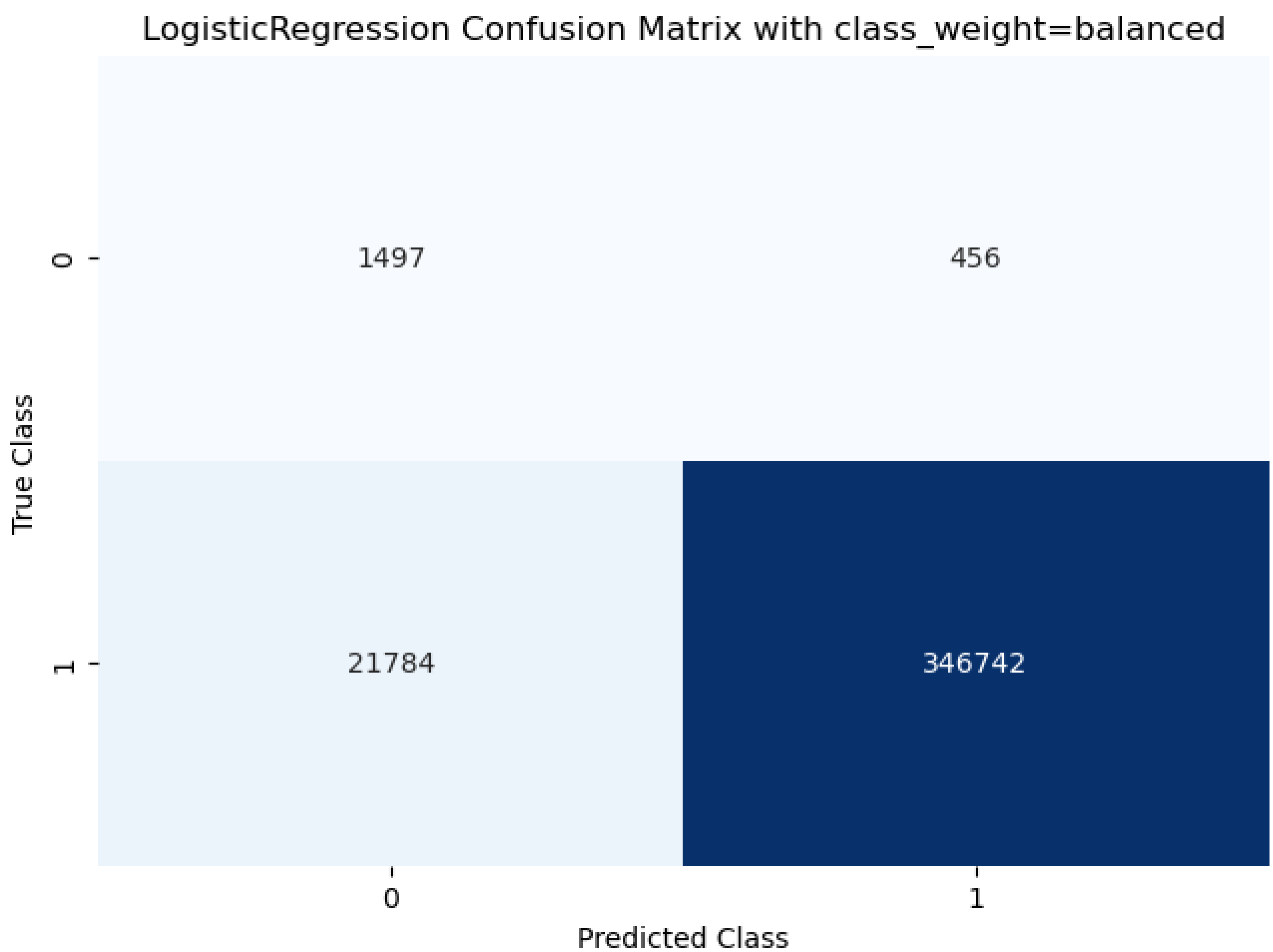
```
# Train a logistic regression model
logit = LogisticRegression( solver='liblinear', class_weight='balanced')
model_logit = logit.fit(x_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred_logit = model_logit.predict(X_test)

# Evaluate the model
print(classification_report(Y_test, y_pred_logit))
print(confusion_matrix(Y_test, y_pred_logit))
print(f'ROC-AUC score : {roc_auc_score(Y_test, y_pred_logit)}')
print(f'Accuracy score : {accuracy_score(Y_test, y_pred_logit)}')
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97    368526
           1       0.06      0.77      0.12      1953

    accuracy                           0.94    370479
```

```
   macro avg       0.53      0.85      0.54    370479
weighted avg       0.99      0.94      0.96    370479

[[346742  21784]
 [   456   1497]]
ROC-AUC score : 0.8537009475361442
Accuracy score : 0.939969606914292
```

In [36]:
```
#Build the confusion matrix
matrix = confusion_matrix(Y_test, y_pred_logit, labels=[1,0])

print(matrix)

# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("LogisticRegression Confusion Matrix with class_weight=balanced"), plt.tight_l
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[  1497    456]
 [ 21784 346742]]
```



Adding the class_weight = balanced has no impact on the model outcome.

Gradient Boosting

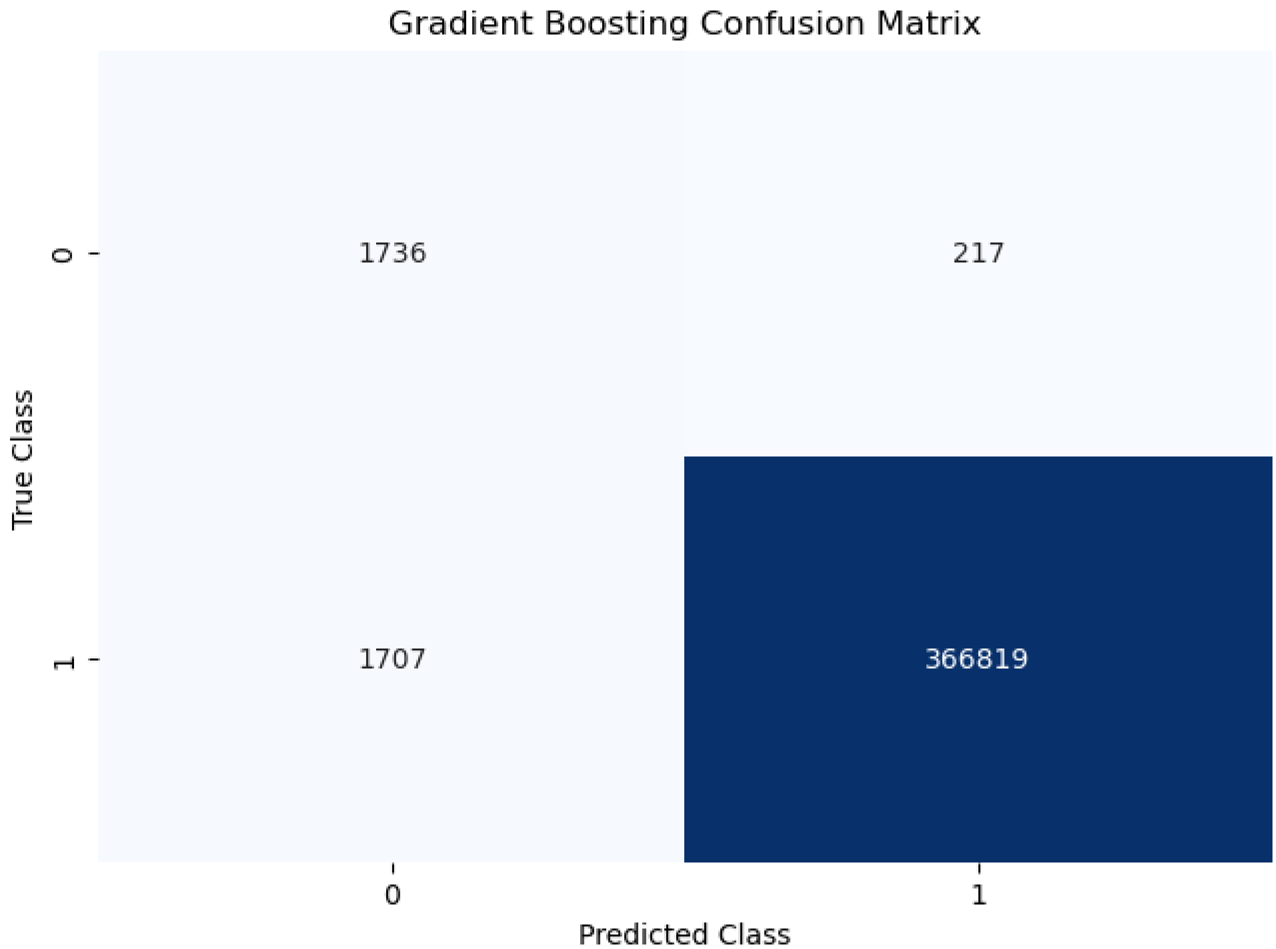
In [38]:
```
xgb_model = XGBClassifier(max_depth = 4)
xgb_model.fit(x_train_resampled,y_train_resampled)
xgb_predicted = xgb_model.predict(X_test)
```

In [39]:
```
# Train an XGBoost classifier
xgb_model = XGBClassifier(random_state=42)
xgb_model.fit(x_train_resampled,y_train_resampled)

# Make predictions on the test set
y_pred_gb = xgb_model.predict(X_test)

# Evaluate the XGBoost Classifier
print(classification_report(Y_test, y_pred_gb))
print(confusion_matrix(Y_test, y_pred_gb))
print(f'ROC-AUC score : {roc_auc_score(Y_test,y_pred_gb)}')
print(f'Accuracy score : {accuracy_score(Y_test, y_pred_gb)}')
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    368526
           1       0.50      0.89      0.64      1953

    accuracy                           0.99    370479
   macro avg       0.75      0.94      0.82    370479
weighted avg       1.00      0.99      1.00    370479

[[366819   1707]
 [   217   1736]]
ROC-AUC score : 0.9421284613116396
Accuracy score : 0.9948067231880889
```

In [40]:
```
#Build the confusion matrix
matrix = confusion_matrix(Y_test, y_pred_gb, labels=[1,0])

print(matrix)

# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("Gradient Boosting Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[  1736    217]
 [  1707 366819]]
```



Neural Network Model

In [41]:
```
# Build a neural network model
model = keras.Sequential([
    layers.Input(shape=(x_train_resampled.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(32, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the neural network
model.fit(x_train_resampled, y_train_resampled, epochs=10, batch_size=128, validation_sp
```

```
Epoch 1/10
18428/18428 [==============================] - 69s 4ms/step - loss: 0.3116 - accuracy:
0.8891 - val_loss: 0.4212 - val_accuracy: 0.7459
Epoch 2/10
18428/18428 [==============================] - 86s 5ms/step - loss: 0.2702 - accuracy:
0.8956 - val_loss: 0.3960 - val_accuracy: 0.7480
Epoch 3/10
18428/18428 [==============================] - 76s 4ms/step - loss: 0.2602 - accuracy:
0.8970 - val_loss: 0.3626 - val_accuracy: 0.7566
Epoch 4/10
18428/18428 [==============================] - 87s 5ms/step - loss: 0.2555 - accuracy:
0.8975 - val_loss: 0.3577 - val_accuracy: 0.7541
Epoch 5/10
18428/18428 [==============================] - 91s 5ms/step - loss: 0.2529 - accuracy:
0.9041 - val_loss: 0.3369 - val_accuracy: 0.8073
```

```
Epoch 6/10
18428/18428 [==============================] - 88s 5ms/step - loss: 0.2509 - accuracy:
0.9078 - val_loss: 0.3298 - val_accuracy: 0.8130
Epoch 7/10
18428/18428 [==============================] - 90s 5ms/step - loss: 0.2498 - accuracy:
0.9079 - val_loss: 0.3208 - val_accuracy: 0.8172
Epoch 8/10
18428/18428 [==============================] - 88s 5ms/step - loss: 0.2487 - accuracy:
0.9084 - val_loss: 0.3132 - val_accuracy: 0.8120
Epoch 9/10
18428/18428 [==============================] - 85s 5ms/step - loss: 0.2485 - accuracy:
0.9086 - val_loss: 0.3365 - val_accuracy: 0.8107
Epoch 10/10
18428/18428 [==============================] - 95s 5ms/step - loss: 0.2477 - accuracy:
0.9089 - val_loss: 0.3247 - val_accuracy: 0.8134
```

Out[41]:

```
<keras.callbacks.History at 0x20b197f5d60>
```

In [42]:

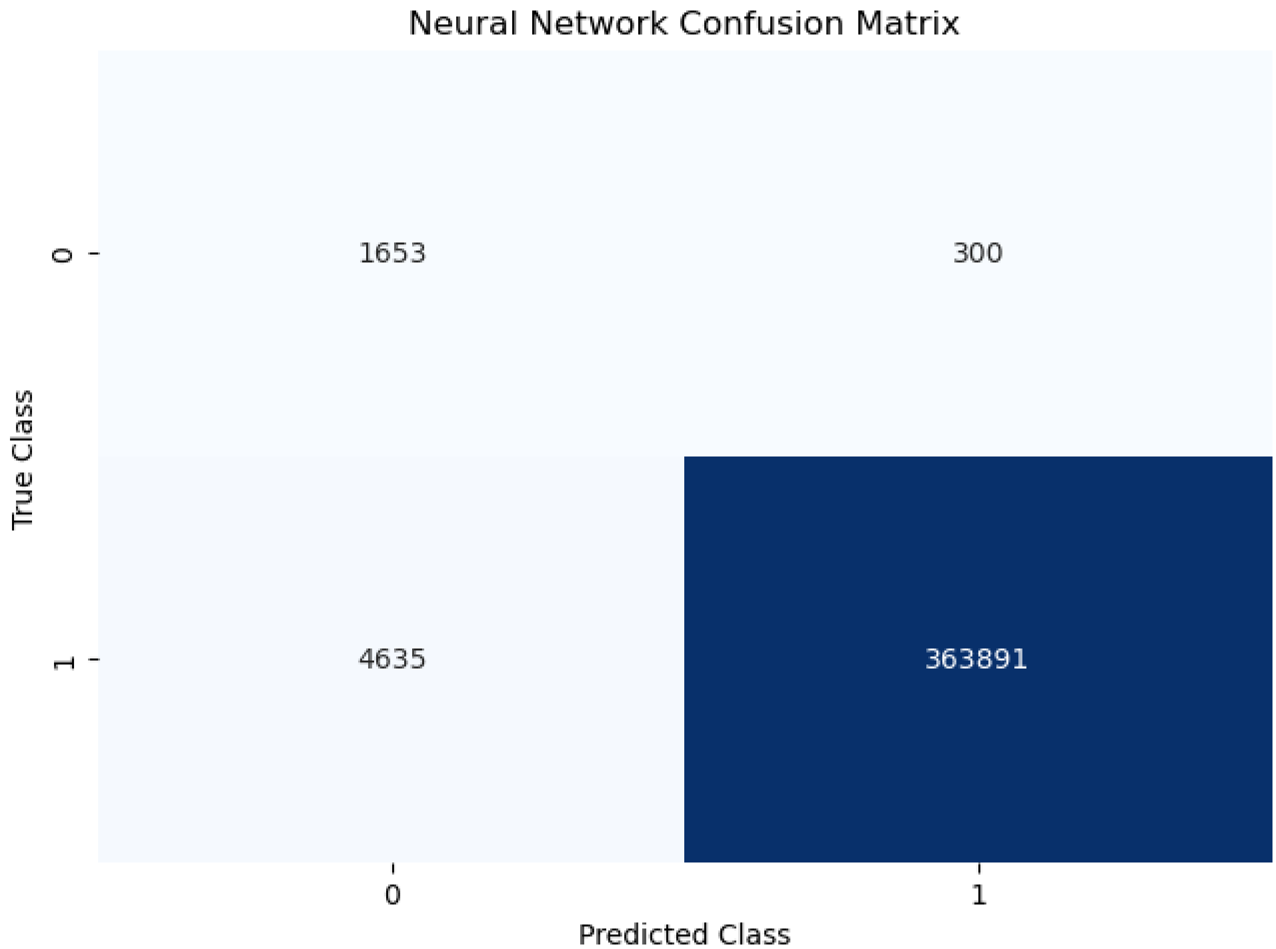
```python
# Make predictions on the test set
y_pred_nn = model.predict(X_test)
y_pred_binary = (y_pred_nn > 0.5).astype(int)

# Evaluate the XGBoost Classifier
print(classification_report(Y_test, y_pred_binary))
print(confusion_matrix(Y_test, y_pred_binary))
print(f'ROC-AUC score : {roc_auc_score(Y_test,y_pred_binary)}')
print(f'Accuracy score : {accuracy_score(Y_test, y_pred_binary)}')
```

```
11578/11578 [==============================] - 25s 2ms/step
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    368526
           1       0.26      0.85      0.40      1953

    accuracy                           0.99    370479
   macro avg       0.63      0.92      0.70    370479
weighted avg       1.00      0.99      0.99    370479

[[363891   4635]
 [   300   1653]]
ROC-AUC score : 0.9169065186854365
Accuracy score : 0.9866794069299475
```

In [43]:

```python
#Build the confusion matrix
matrix = confusion_matrix(Y_test, y_pred_binary, labels=[1,0])

print(matrix)

# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("Neural Network Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[  1653    300]
 [  4635 363891]]
```


Neural Network Confusion Matrix
True Class
Predicted Class
0
1
1653
300
4635
363891