Documentation - exercise 4

Dataset1 : Credit Card Transactions Fraud Detection Dataset

Dataset2 : Handwriting A-Z alphabet

Professor : Dr. Kheradpisheh

Teacher Assistant : MohammadReza Khanmohammadi

By: Katayoun Kobraei

1. Introduction to Dataset

This Dataset is a simulated credit card transaction dataset containing legitimate and fraud transactions from the duration 1st Jan 2019 - 31st Dec 2020. It covers credit cards of 1000 customers doing transactions with a pool of 800 merchants. This was generated using Sparkov Data Generation | Github tool created by Brandon Harris. This simulation was run for the duration - 1 Jan 2019 to 31 Dec 2020. The files were combined and converted into a standard format. Here's an explanation of each feature in the Credit Card Transactions Fraud Detection Dataset:

- Unnamed: 0: An index or identifier for the rows, likely generated by the data collection process.
- trans_date_trans_time: The date and time of the transaction in a timestamp format.
- cc_num: Credit card number used in the transaction (sensitive information, usually masked or encrypted in real-world scenarios).
- merchant: Name or identifier of the merchant where the transaction took place.
- category: The category of the transaction, indicating the type of goods or services purchased.
- amt: The transaction amount, representing the monetary value of the transaction.
- first: First name of the cardholder.
- last: Last name of the cardholder.
- gender: Gender of the cardholder (e.g., 'M' for male, 'F' for female).
- street: Street address of the cardholder.
- city: City where the cardholder resides.
- state: State where the cardholder resides.
- zip: ZIP code of the cardholder's address.
- lat: Latitude of the cardholder's location.
- long: Longitude of the cardholder's location.
- city_pop: Population of the city where the cardholder resides.
- job: Occupation or job title of the cardholder.
- dob: Date of birth of the cardholder.
- trans_num: Transaction number or identifier.
- unix_time: Transaction time in Unix timestamp format.
- merch_lat: Latitude of the merchant's location.
- merch_long: Longitude of the merchant's location.
- is_fraud: Binary indicator (0 or 1) representing whether the transaction is fraudulent (1) or not (0).

By making use of these features, we are encouraged to apply advanced data science techniques to discover hidden connections, patterns, and insights. These findings can ultimately improve the accuracy of predictions for fraud detection that is the actual task.

2. Abstract

The primary objective of this assignment is to delve into the realm of fraud detection, a critical facet in safeguarding businesses and organizations from financial losses. With the increasing sophistication of fraudsters, traditional methods struggle to keep pace with emerging fraudulent activities. Leveraging data science and machine learning algorithms offers a potent solution to enhance fraud detection capabilities. This assignment focuses on exploring various machine learning methodologies tailored for fraud detection. The dataset chosen for classification is the credit card fraud detection dataset. The primary task revolves around predicting whether a transaction is fraudulent or not. The training of models will be executed on the training dataset, and the accuracy of each model will be assessed on both the test set and the training set. A proportion of 0.2 of the dataset will be reserved as the test set, and the model will be trained on the remaining data. The report should encompass accuracy metrics, such as the AUC curve, confusion matrix, F1 score, and the decision boundary of predictions. Additionally, a crucial aspect of the assignment involves the implementation of a Pipeline using the sklearn library.The dataset chosen for classification is the credit card fraud detection dataset. The primary task revolves around predicting whether a transaction is fraudulent or not. The training of models will be executed on the training dataset, and the accuracy of each model will be assessed on both the test set and the training set. A proportion of 0.2 of the dataset will be reserved as the test set, and the model will be trained on the remaining data. The report should encompass accuracy metrics, such as the AUC curve, confusion matrix, F1 score, and the decision boundary of predictions. Additionally, a crucial aspect of the assignment involves the implementation of a Pipeline using the sklearn library.

3. Overview of the dataset

Train and test data's shape:

The training dataset contains a total of 1296675 entries and 23 columns and for the test data the number is 555719 rows × 23 columns.

First we check columns of train and test data set to some samples and find out the type of each feature.

```
 #   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
```

Then we see some statistics features of each column:

	Unnamed: 0	cc_num	amt	zip	lat	long	city_pop	unix_time	merch_lat	merch_
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	1.296675
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	3.853734e+01	-9.022646
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	5.109788e+00	1.377109
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	1.902779e+01	-1.666712
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	3.473357e+01	-9.689728
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	3.936568e+01	-8.743839
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	4.195716e+01	-8.023680
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	6.751027e+01	-6.695090

Then we check if these is any null values in features to handle in this step:

```
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
```

```
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
```

Fortunately there are not any null values in the train and test dataset to handle them.

4. Feature Engineering & Feature Extraction:

Time features:
The ‘trans_date_trans_time’ column consists of various data like day, month, hour. We can see this column below:

```
0          2019-01-01 00:00:18
1          2019-01-01 00:00:44
2          2019-01-01 00:00:51
3          2019-01-01 00:01:16
4          2019-01-01 00:03:06
                   ...
1296670    2020-06-21 12:12:08
1296671    2020-06-21 12:12:19
1296672    2020-06-21 12:12:32
1296673    2020-06-21 12:13:36
1296674    2020-06-21 12:13:37
```

First we convert these values to datetime values using the to_datetime function in pandas library. Then we extract ‘hour’, ‘week_day’ from them for both training and test set.

```
train_data.hour
```

```
0          0
1          0
2          0
3          0
4          0
          ..
1296670    12
1296671    12
1296672    12
1296673    12
1296674    12
```

```
train_data.week_day
```

```
0          1
1          1
2          1
3          1
4          1
          ..
1296670    6
1296671    6
1296672    6
1296673    6
1296674    6
```

Then we make 3 other columns named 'transactions_last_1d', 'transactions_last_7d' and 'transactions_last_30d' from the

```
train_data['transactions_last_1d']
```

```
0          1.0
1          1.0
2          1.0
3          1.0
4          1.0
          ...
1296670    1.0
1296671    1.0
1296672    1.0
1296673    1.0
1296674    1.0
```

```
train_data['transactions_last_7d']
```

```
0          0.0
1          0.0
2          0.0
3          0.0
4          0.0
          ...
1296670    7.0
1296671    7.0
1296672    7.0
1296673    7.0
1296674    7.0
```

```
train_data['transactions_last_30d']
```

```
0           0.0
1           0.0
2           0.0
3           0.0
4           0.0
           ...
1296670    30.0
1296671    30.0
1296672    30.0
1296673    30.0
1296674    30.0
```

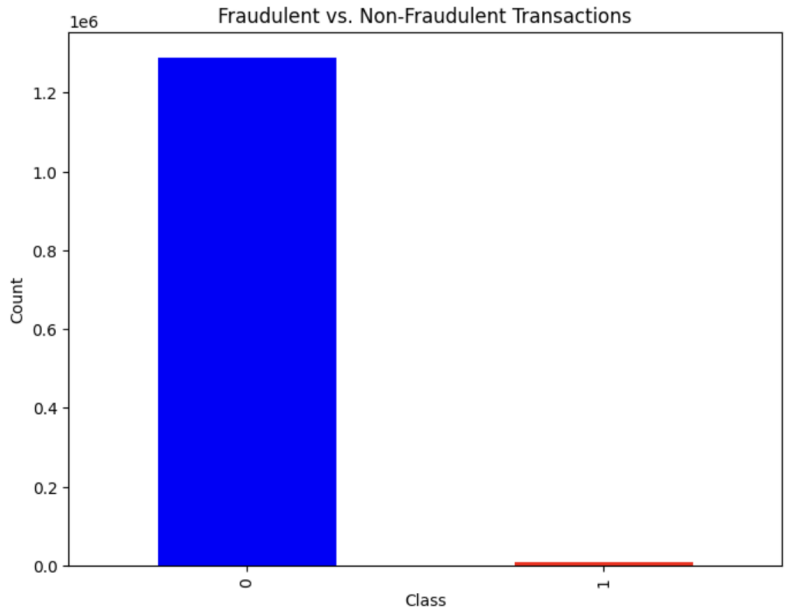
5. Data imbalance check:

Now we check if the target value (‘is_fraud’) is imbalance or what.

```
is_fraud
0    1289169
1       7506
```

```
is_fraud
0    99.421135
1     0.578865
```

We see that target value consists of 1289169 (99.42 %) non fraud transactions and 7506 (0.57 %) fraud transactions.



So if we calculate imbalance ratio it will be:

```
Imbalance Ratio: 171.75179856115108
```

The results provided indicate a highly imbalanced dataset for the credit card transactions fraud detection task.

Non-fraudulent transactions (Class 0): 1,289,169
Fraudulent transactions (Class 1): 7,506
The vast majority of transactions are non-fraudulent, with only a small number of fraudulent transactions. This is a common scenario in fraud detection datasets.

Non-fraudulent transactions (Class 0): 99.42%

Fraudulent transactions (Class 1): 0.58%
The vast majority (99.42%) of transactions are labeled as non-fraudulent, while only a small proportion (0.58%) are labeled as fraudulent.

Imbalance Ratio: 171.75
The imbalance ratio is calculated as the ratio of the number of non-fraudulent transactions to fraudulent transactions. In this case, there are approximately 172 times more non-fraudulent transactions than fraudulent transactions.

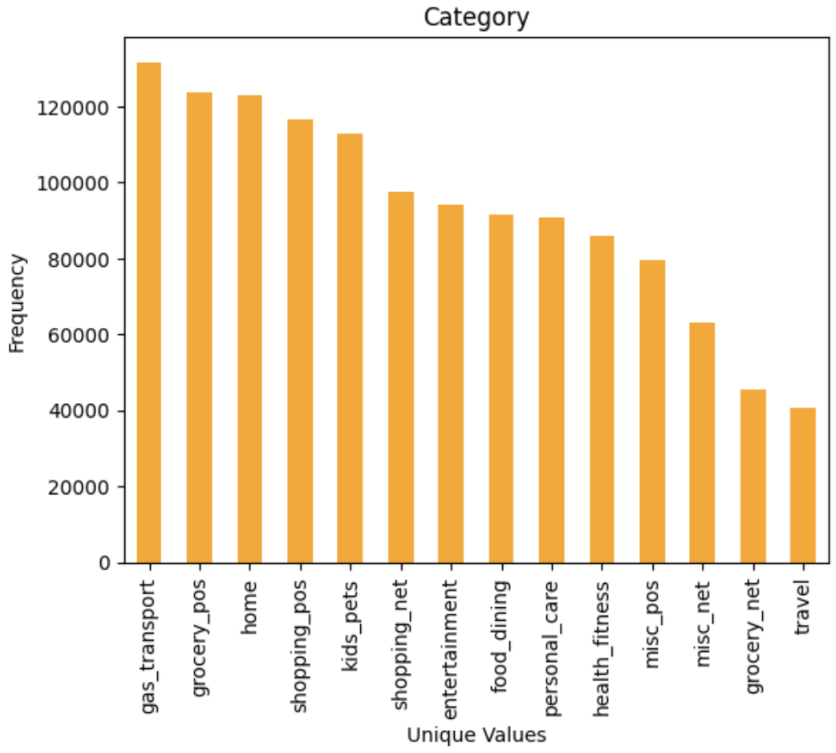
The class imbalance is severe, with the majority class (non-fraudulent transactions) significantly outnumbering the minority class (fraudulent transactions).
While the imbalance ratio provides a quantitative measure (172:1), it's important to note that highly imbalanced datasets can pose challenges for machine learning models, as they may be biased towards predicting the majority class. When building a fraud detection model, it's essential to address this imbalance. Techniques such as resampling (undersampling or oversampling), using different evaluation metrics (precision, recall, F1-score), or employing specialized algorithms designed for imbalanced datasets can be considered.

6. Exploratory data analysis:

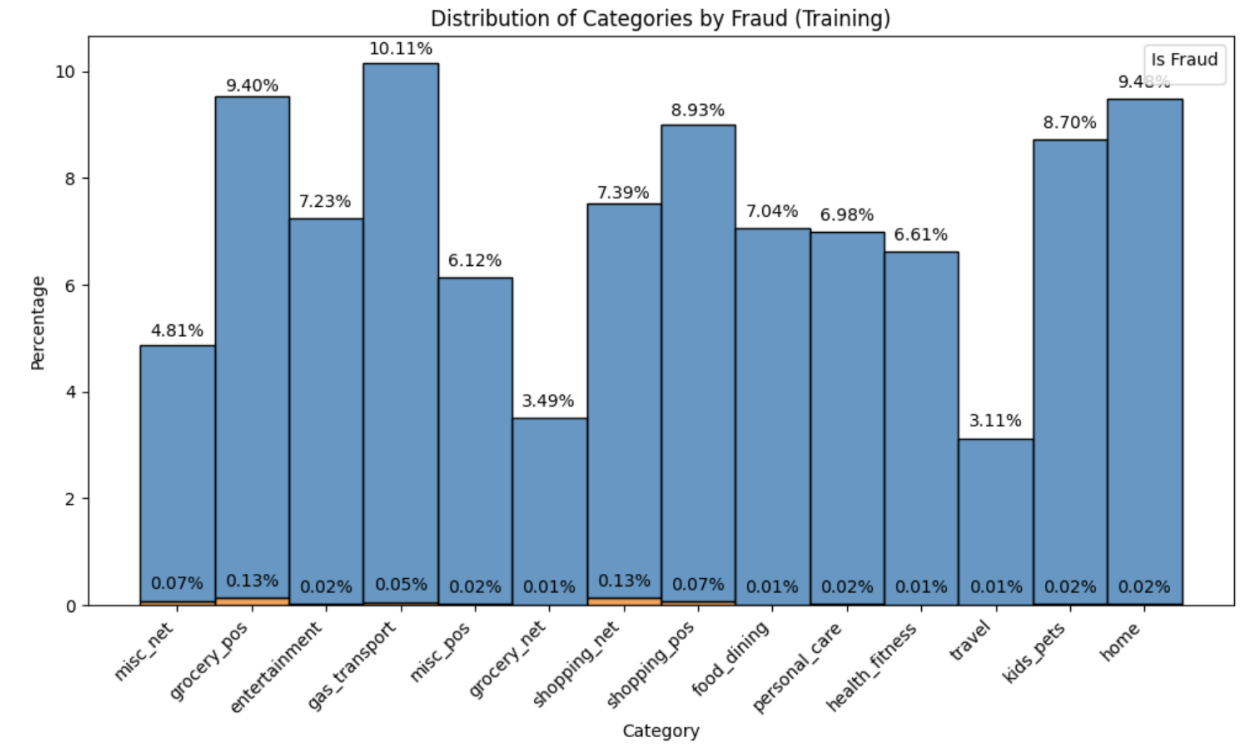
- **Plots and features**

Exploratory Data Analysis (EDA) is a crucial phase in the data science lifecycle that empowers analysts and data scientists to uncover hidden patterns, relationships, and insights within a dataset. It serves as the initial step in understanding the characteristics of the data, identifying potential outliers, and forming hypotheses for further investigation. In this exploratory journey, we will leverage various tools and methods to navigate through the Credit Card Transactions Fraud Detection Dataset. Our objective is to uncover meaningful patterns, anomalies, and trends that will contribute to the development of robust models for fraud detection. From understanding the distribution of key features to scrutinizing temporal patterns, our EDA process will lay the foundation for a deeper understanding of the dataset and guide subsequent steps in the data science pipeline.

At first as an instance we check frequency of each category that we have:



Now we check how much fraud transaction exists in each category:

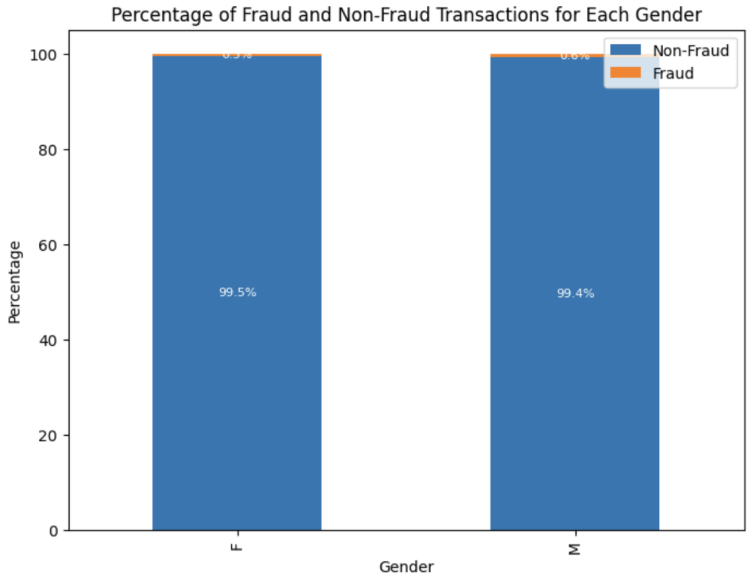


We see that in some categories like grocery_pos or misc_net there are much more fraud transactions in comparison to other categories so there should be some information in this column relates to the target value.

Next we check gender of people in this dataset:

```
gender
F     709863
M     586812
```

is_fraud	0	1
gender		
F	99.473842	0.526158
M	99.357375	0.642625



We see there is not a big gap between the number of fraud transactions done by a male or female customer. So this column will not be very crucial for our prediction.

As we made a new feature named week_day out of datetime values, we now make sine and cosine transformations on transaction date and time as a new meaningful features.

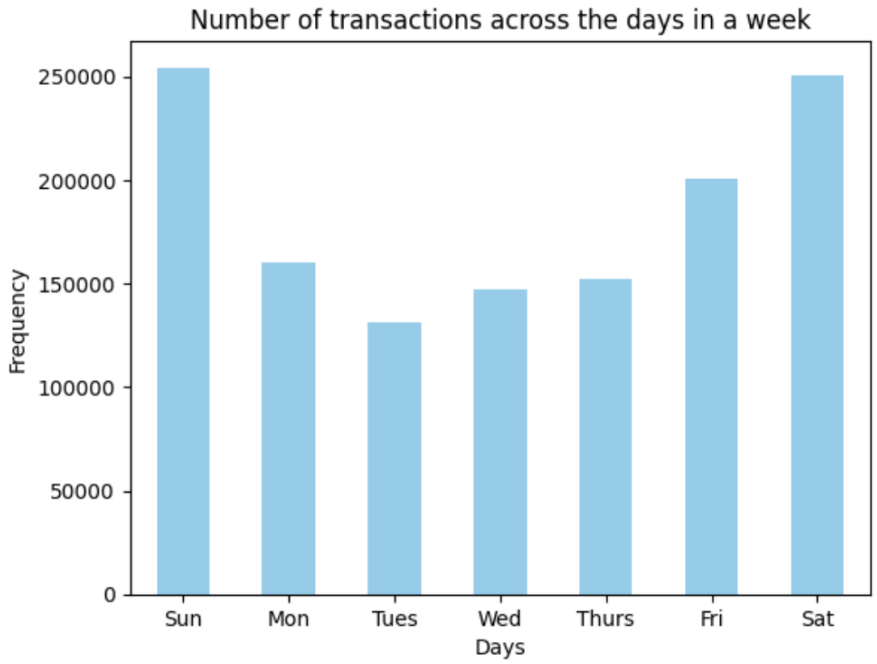
```
train_data['DayOfWeekCos']
```

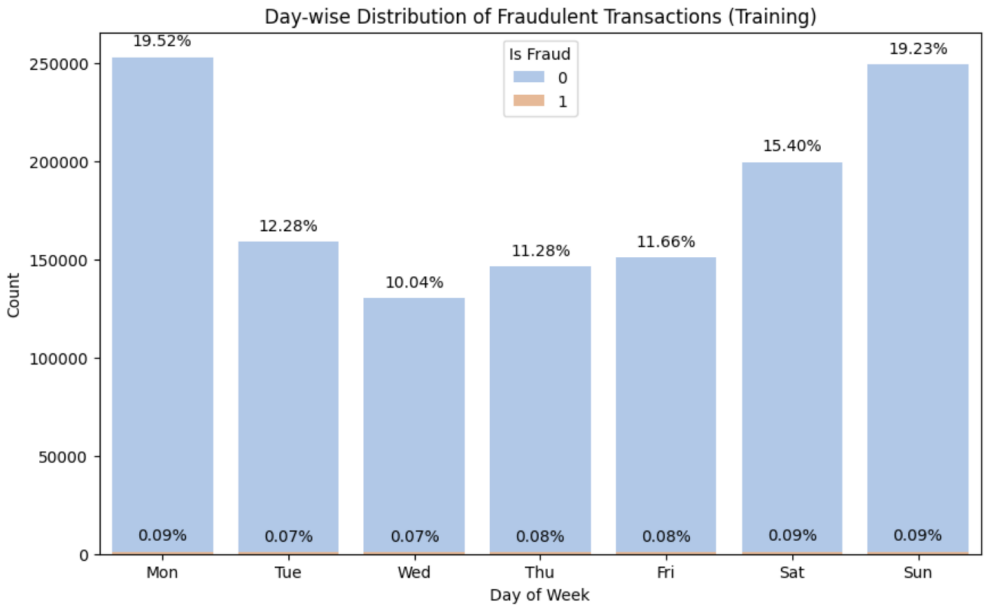
```
0          0.62349
1          0.62349
2          0.62349
3          0.62349
4          0.62349
            ...
1296670    0.62349
1296671    0.62349
1296672    0.62349
1296673    0.62349
1296674    0.62349
```

```
train_data['DayOfWeekSin']
```

```
0          0.781831
1          0.781831
2          0.781831
3          0.781831
4          0.781831
              ...
1296670   -0.781831
1296671   -0.781831
1296672   -0.781831
1296673   -0.781831
1296674   -0.781831
```

Then we check if day of week has an impact on being a fraud transaction or not.



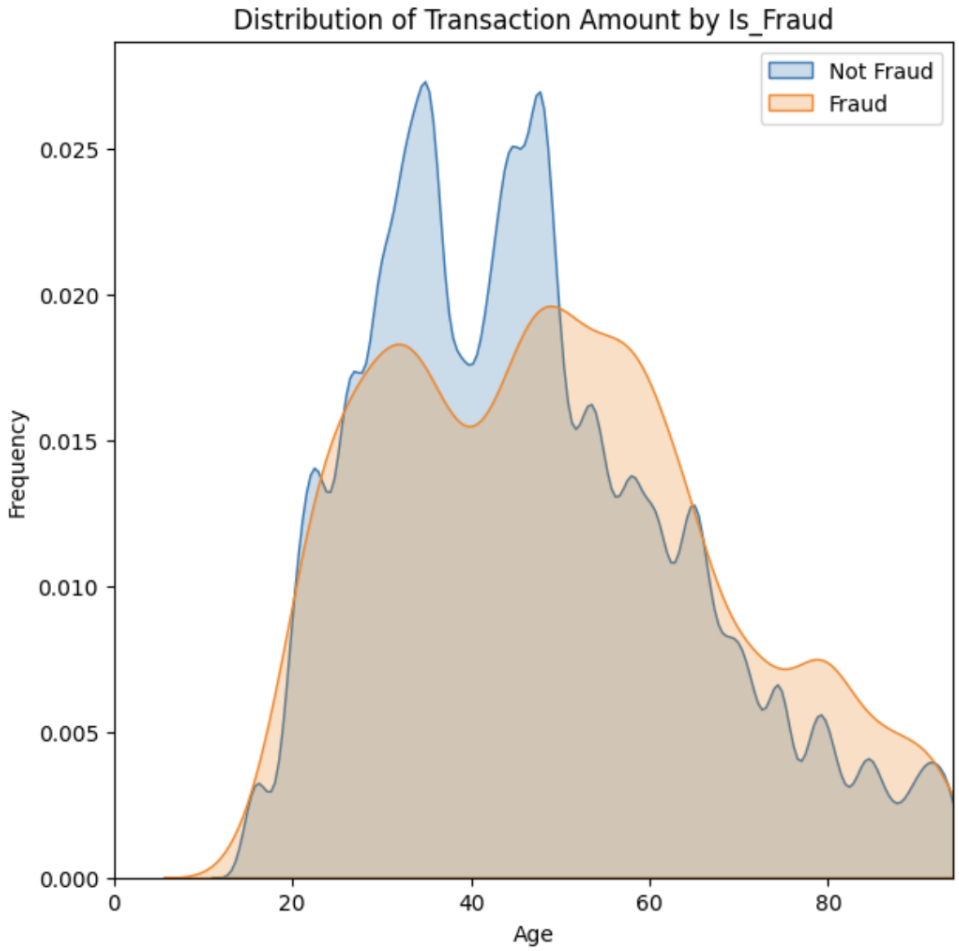


We see that many transactions happen on sunday and saturday. So it is probable that we will have more fraud transactions these days. And if we plot these results we see many fraud transactions happen on Sunday and Monday and for the rest of the week it is almost the same number.

After all, we create new features named ‘age’ from the ‘dob’ column which is the age of the customer.

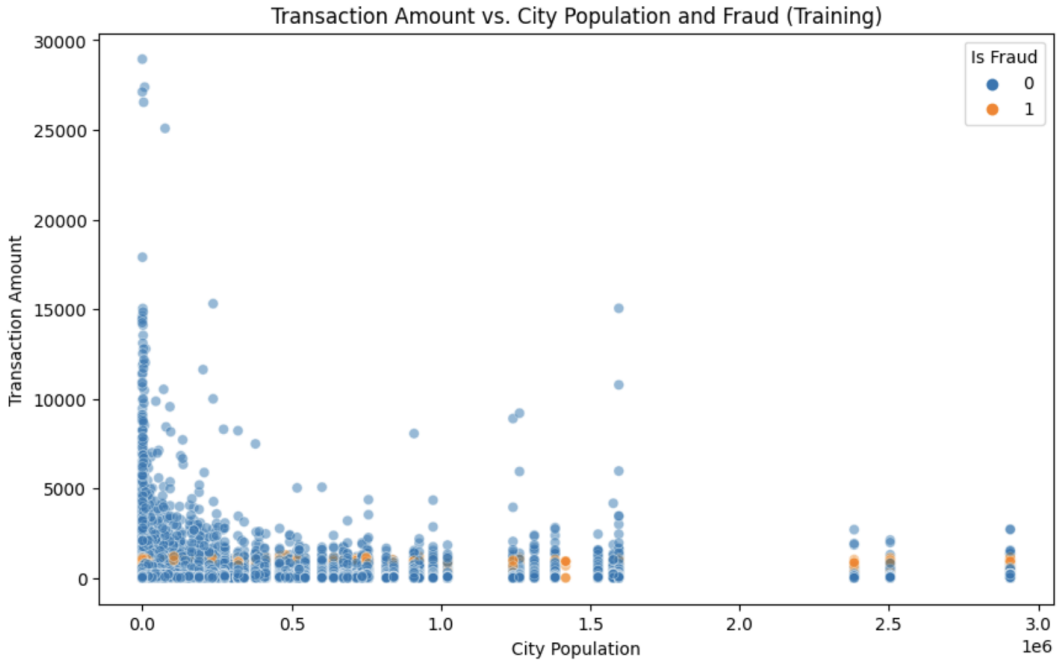
```
         dob      trans_datetime  age
0 1988-03-09 2019-01-01 00:00:18   31
1 1978-06-21 2019-01-01 00:00:44   41
2 1962-01-19 2019-01-01 00:00:51   57
3 1967-01-12 2019-01-01 00:01:16   53
4 1986-03-28 2019-01-01 00:03:06   33
```

Now we check the distribution of transaction amounts by Is_Fraud column to check a fraud transaction is happening in which ages?

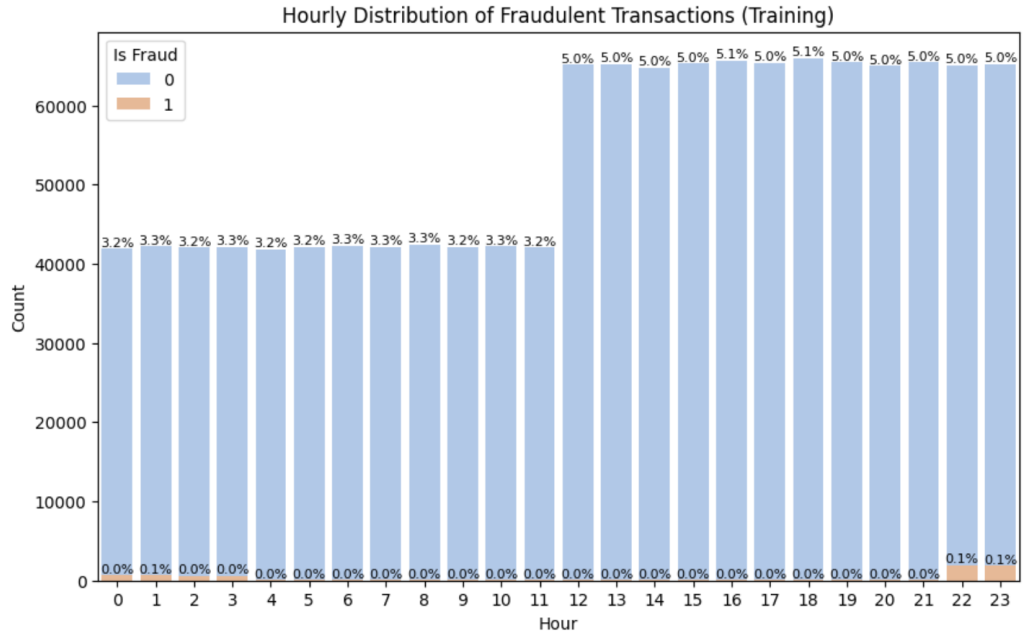


We see most fraud transactions are happening in 30-35 and 50-55 years old.

We can see if the city population has an influence on fraud transactions or not.



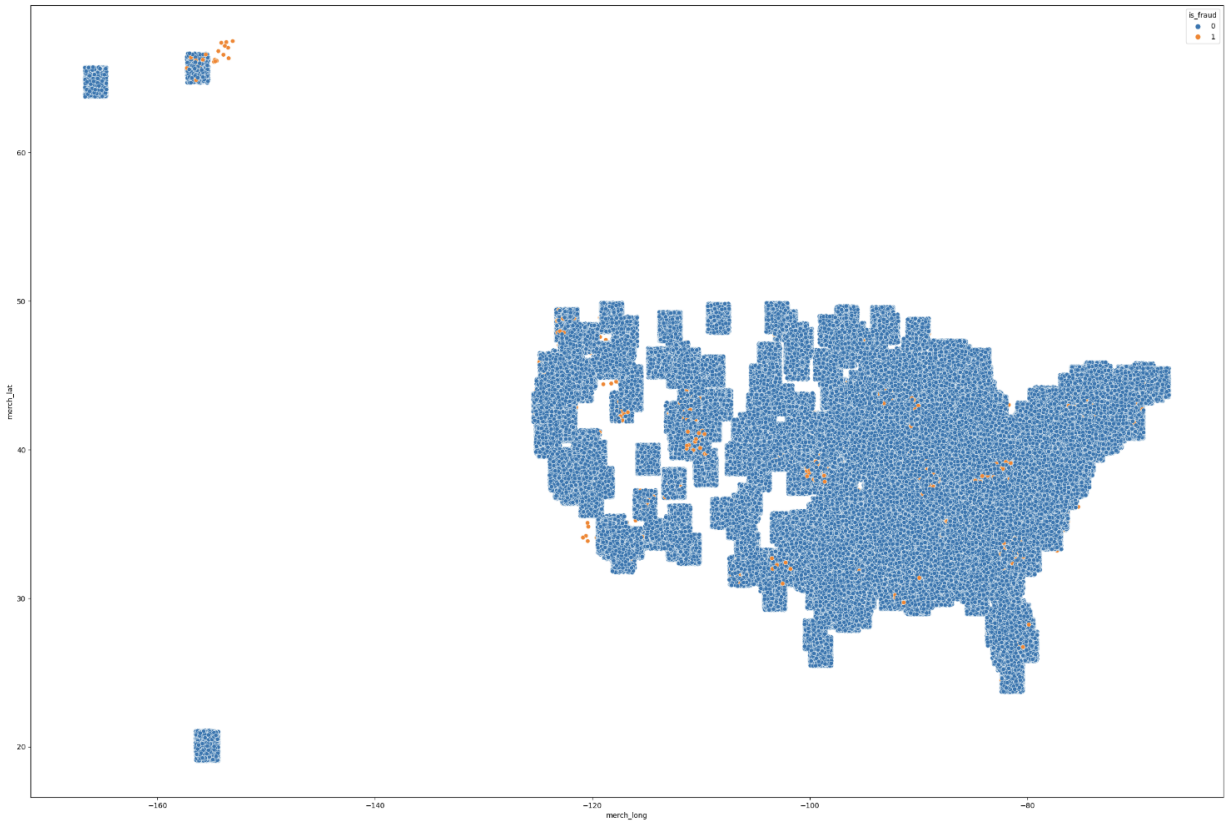
Then we check in which hour we have the most fraud transactions.



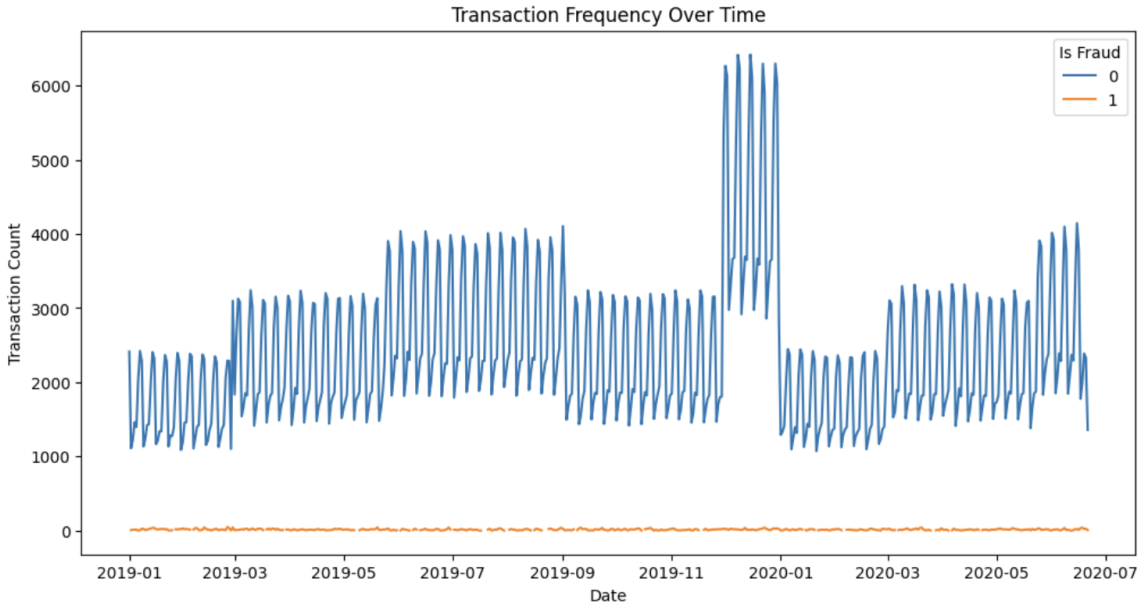
All transactions happen in 22 and 23 and 1. Any fraud transaction happens in other hours.

Now we see some other plots of other features:

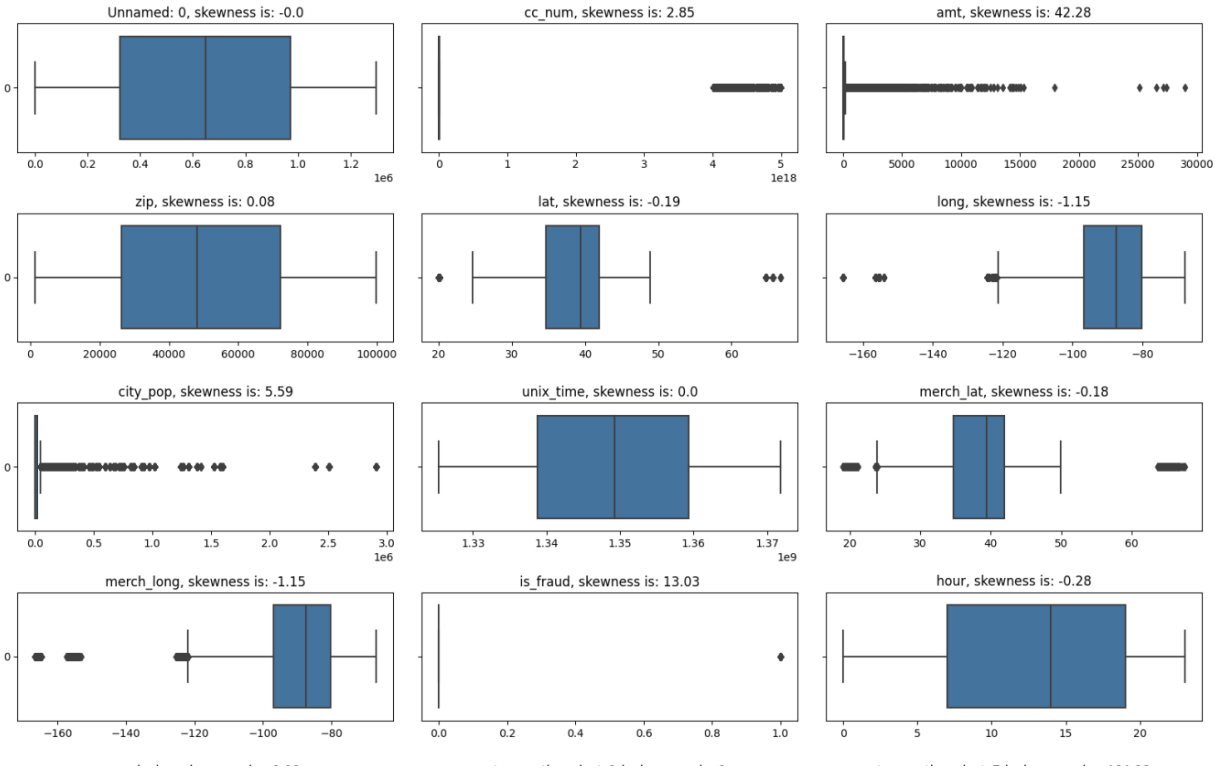
This plot below is a scatter plot that visualizes the geographical distribution of credit card transactions based on the longitude (merch_long) and latitude (merch_lat) of the merchants involved. Each point on the scatter plot represents a specific transaction, and the color of the points is determined by the is_fraud column, indicating whether the transaction is fraudulent or not.

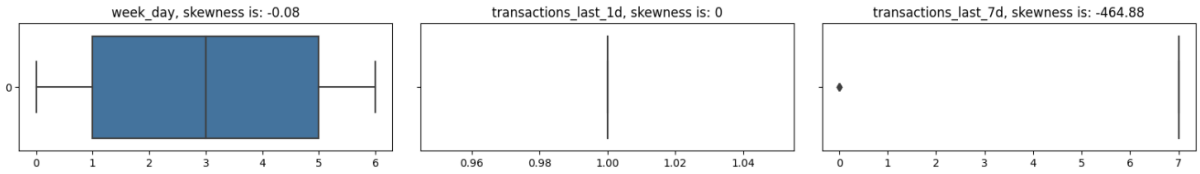


Next plot visualizes the frequency of transactions over time, with a focus on whether the transactions are fraudulent or not.

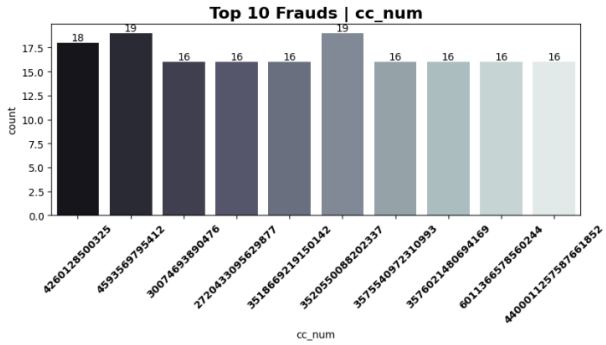


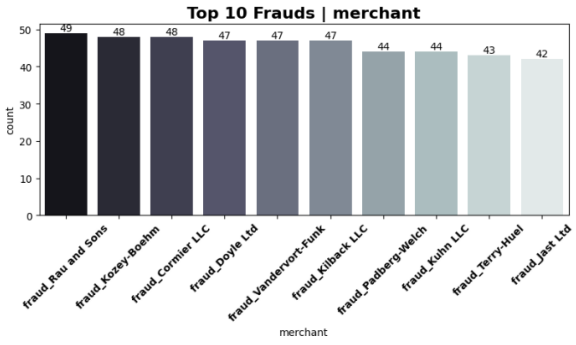
Now we plot all boxplots of all variables to see their distribution:

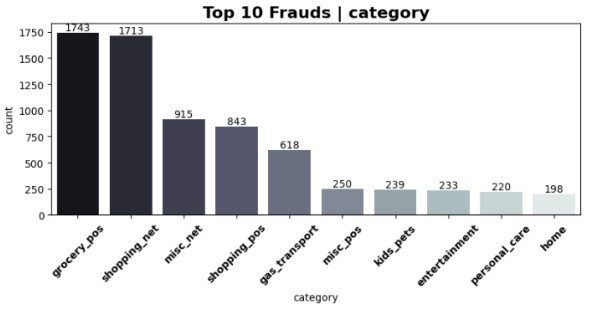


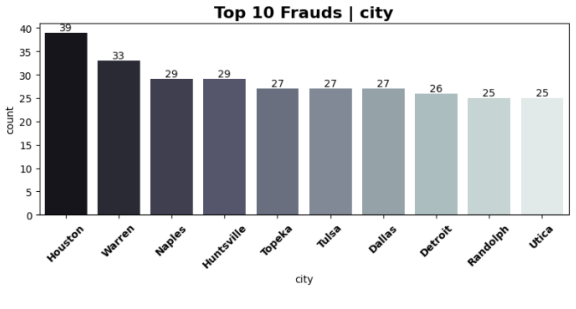


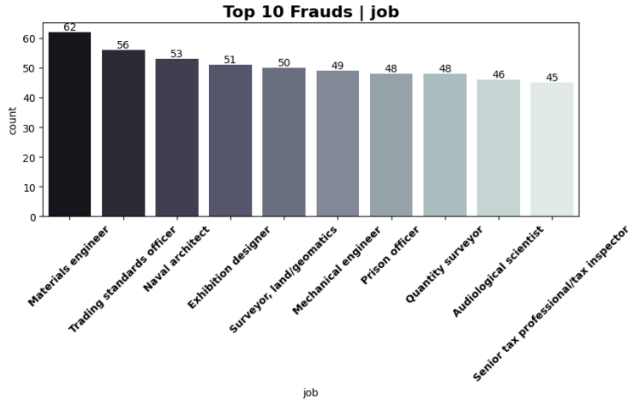
Now we generate a set of bar plots, each displaying the top 10 frauds based on different categorical columns.

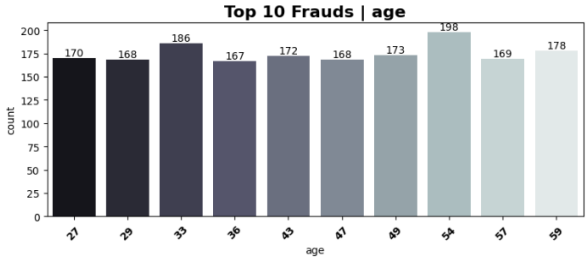






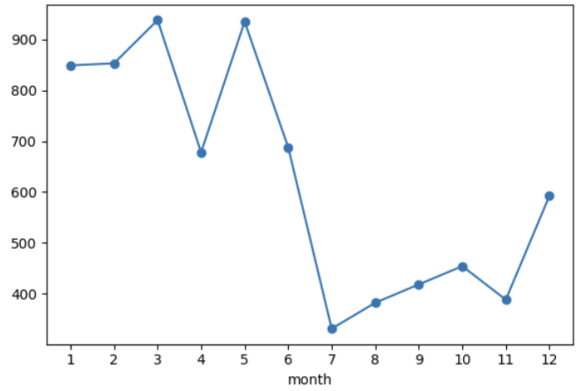


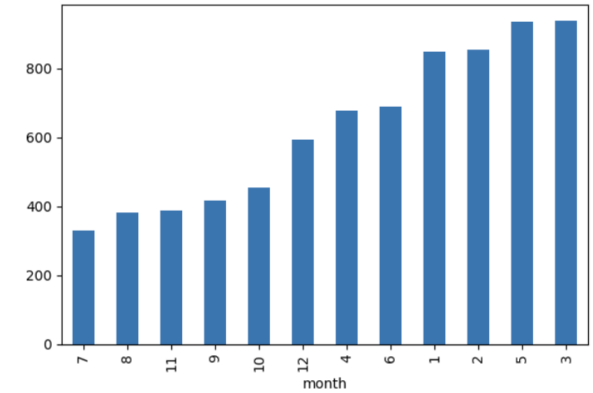


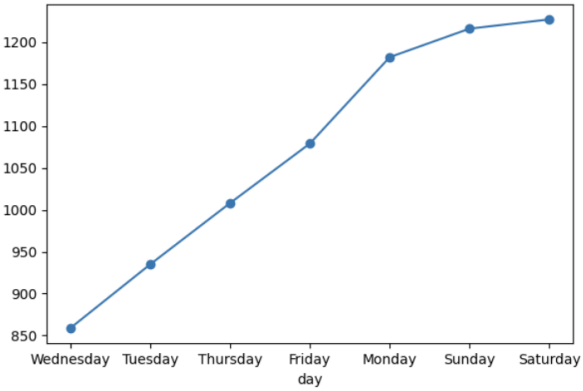


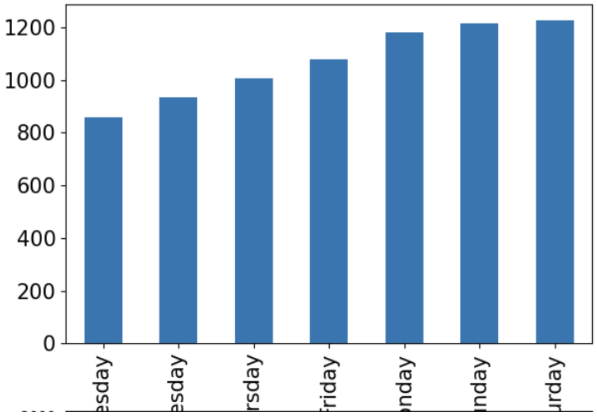
Now we want to check what are the most probable month/ day / hour frauds occur?

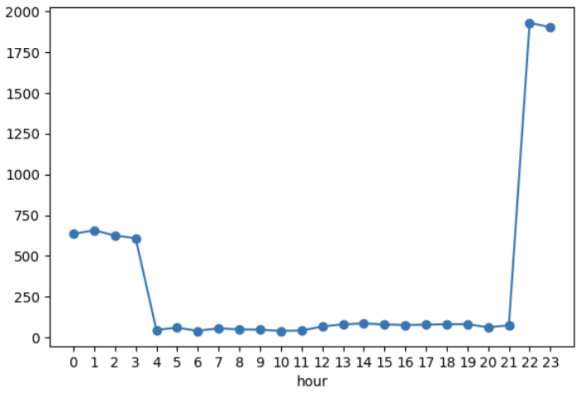
We should build features named ‘hour’, ‘day’ and ‘month’. After that we display its results.

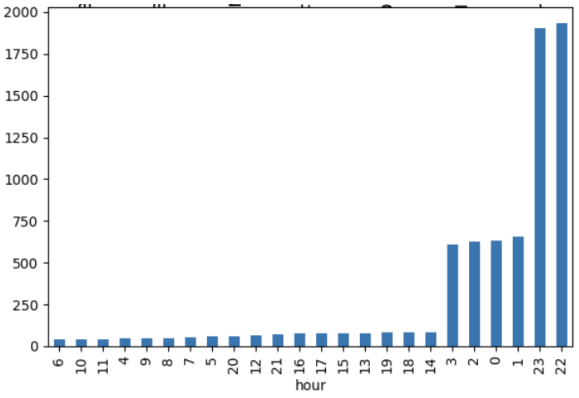












We conclude that most fraud transactions occurs:
On March
On Sunday
At 10 PM

How about statistics tests?

We want to test if Males and females are exposed to fraud equally (approximately).

Null Hypothesis (H0): The mean exposure to fraud is equal for males and females.

Alternative Hypothesis (H1): The mean exposure to fraud is not equal for males and females.

T-test: t-statistic = 8.264, p-value = 0.0, p-value<0.05? True

The low p-value (approaching zero) indicates that there is strong evidence against the null hypothesis. With a p-value less than 0.05, you would typically reject the null hypothesis in favor of the alternative hypothesis. The t-statistic of 8.264 suggests a significant difference between the means of the two groups.

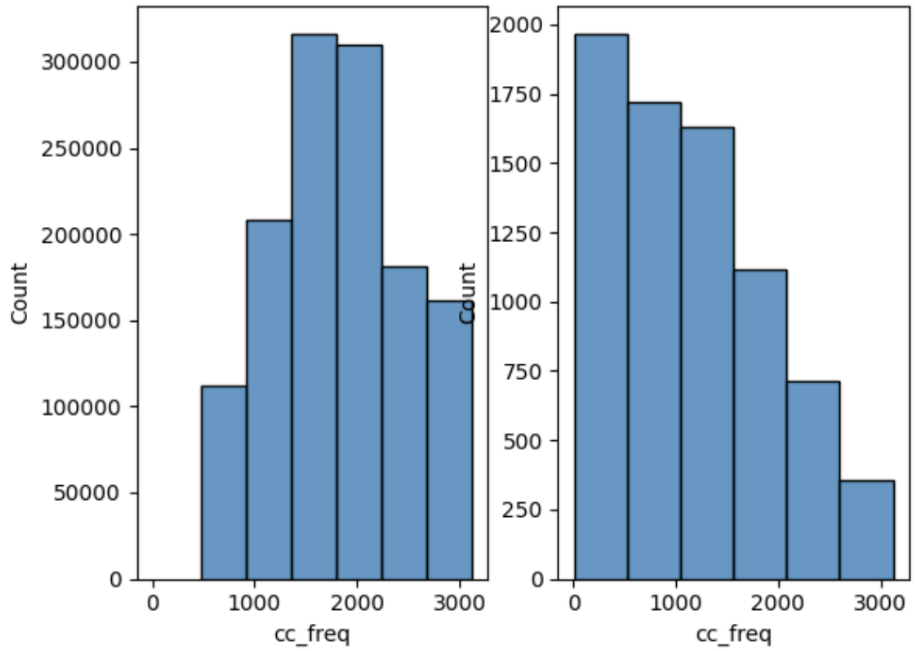
Then we want to check if the mean city population for fraud and non-fraud transactions is equal.

T-test: t-statistic = 2.432, p-value = 0.015, p-value<0.05? True

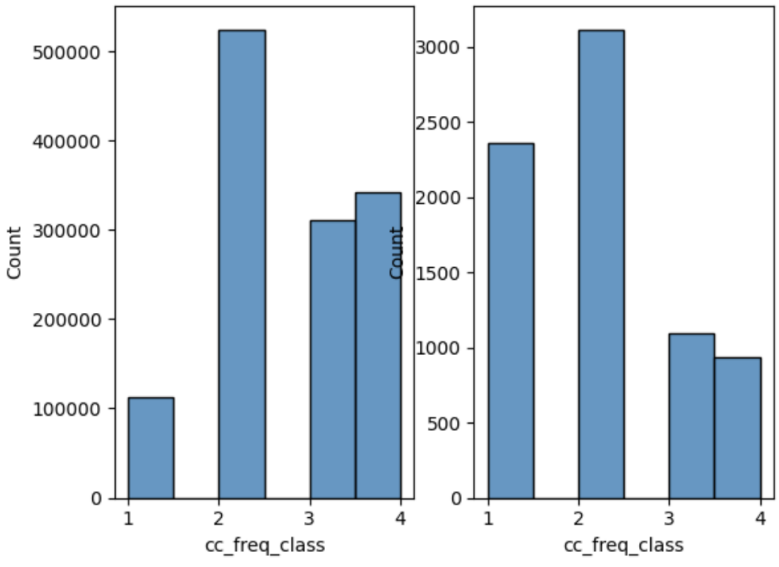
Since we accept the null hypothesis,we conclude that there is no significant difference between means, We conclude also that city_population does not help us on the target(is_fraud), so we will drop it.

Now we want to handle and extract features from cc_num feature.

We make a feature named ‘cc_freq’ from it. The lambda function in code uses the freq dictionary to look up the frequency of each unique credit card number (cc_num). If the credit card number is not found in the freq dictionary, it defaults to 0. This effectively counts the frequency of each credit card number in the dataset and assigns it to the new cc_freq column.



Then we make a function class_det that categorizes credit card frequencies into different classes based on predefined ranges. Then, we apply this function to create a new feature called cc_freq_class in both the training and test datasets. Finally, we print the unique values of the cc_freq_class feature and display a histogram.




```
cc_freq_class
1    2.060334
2    0.590935
3    0.352461
4    0.271482
```

Now clearly fraud occurs more in credit cards with less use (new ones) and for genuine transactions, it follows a normal distribution.

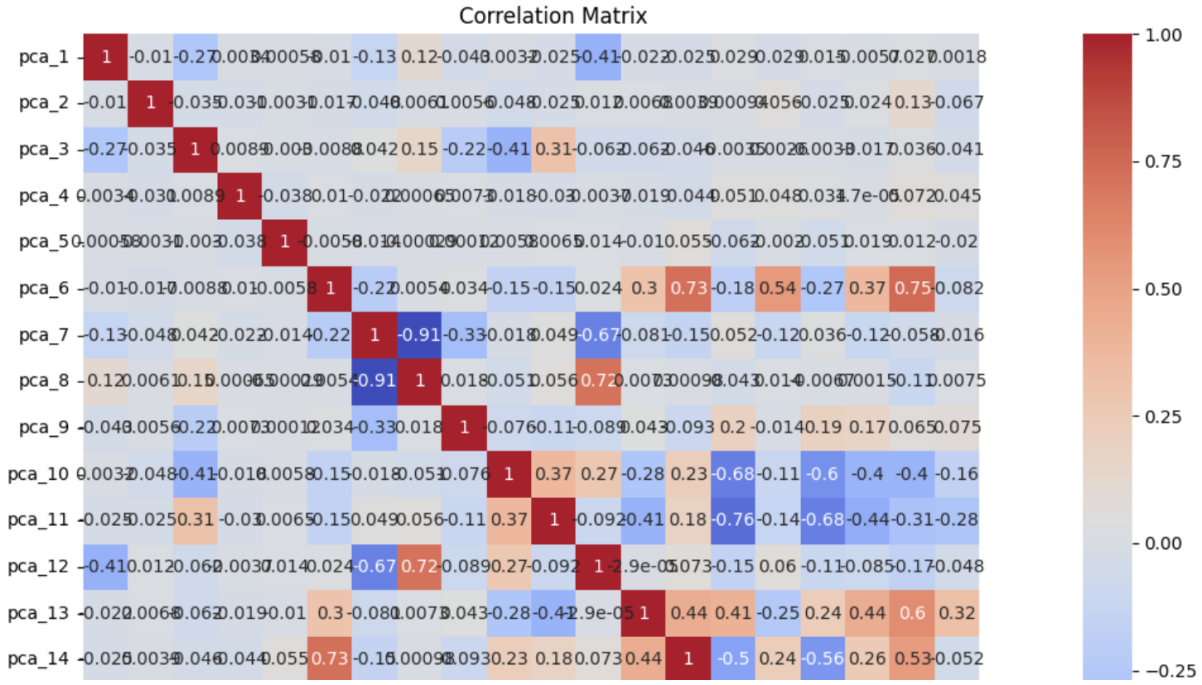
Now we drop some unnecessary features like ['Unnamed: 0', 'trans_date_trans_time', 'first', 'last', 'street', 'city', 'state', 'zip', 'job', 'dob', 'trans_num', 'trans_date', 'merchant', 'trans_datetime'] from the dataset.

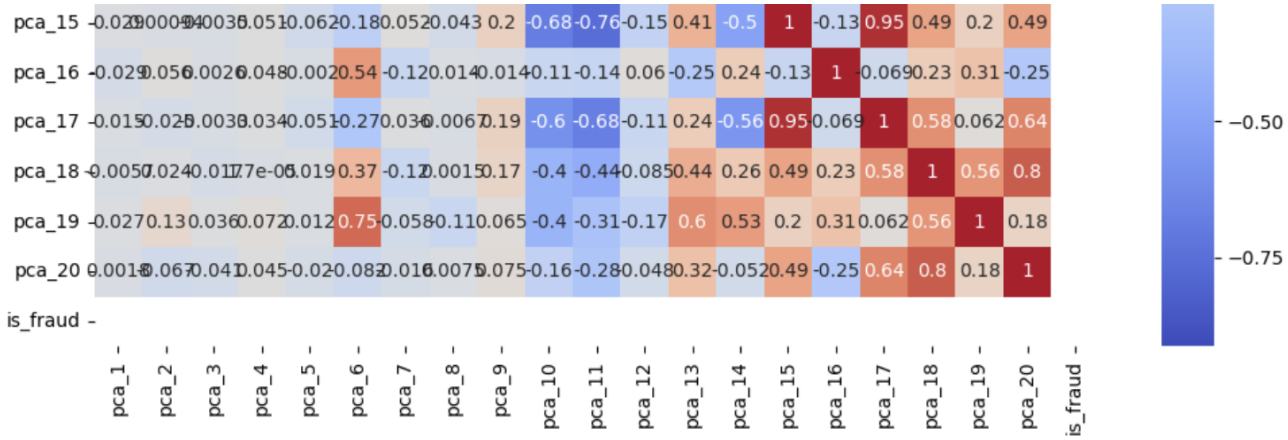
Then we encode categorical feature like 'category', 'gender', 'day'. Then we scale our numerical features using StandardScaler().

- **PCA**

we utilized Principal Component Analysis (PCA) with 20 components to diminish the dimensionality of the dataset. This step aimed to resolve the challenge posed by a substantial number of features resulting from one-hot encoding. The goal was to identify uncorrelated components that could potentially provide more informative insights for the target variable. The outcome is represented in the DataFrame 'df_pca,' which encompasses the PCA components along with the target variable 'is_fraud'. Then we developed an Outlier Removal function based on the Interquartile Range (IQR) method. This function eliminates data points that fall outside the lower and upper bounds, defined by 1.5 times the IQR below the first quartile (Q1) and above the third quartile (Q3). The specified features undergo outlier removal, enhancing the robustness and reliability of the dataset.

Here is the confusion matrix of principle components:





In the PCA (Principal Component Analysis) phase, we implemented dimensionality reduction to tackle the challenge of dealing with a high number of features resulting from one-hot encoding. Utilizing PCA, the goal was to generate a set of uncorrelated features, retaining crucial information in the data. This approach streamlines model training by reducing computational complexity and addressing multicollinearity, potentially enhancing both interpretability and performance.

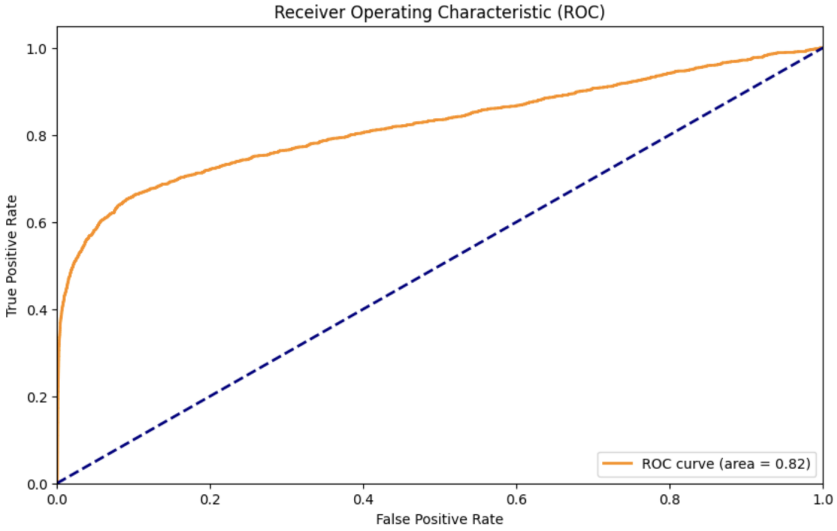
The dataset was divided into training and testing sets, with 80% allocated for training and 20% for testing. Notably, the dataset exhibited significant class imbalance, predominantly comprising instances from the non-fraudulent class. To mitigate this imbalance and improve the model's ability to recognize fraud-related patterns, the Synthetic Minority Over-sampling Technique (SMOTE) was employed for oversampling the minority class. This ensures a more balanced representation of both classes in the training data, thereby enhancing the model's predictive performance.

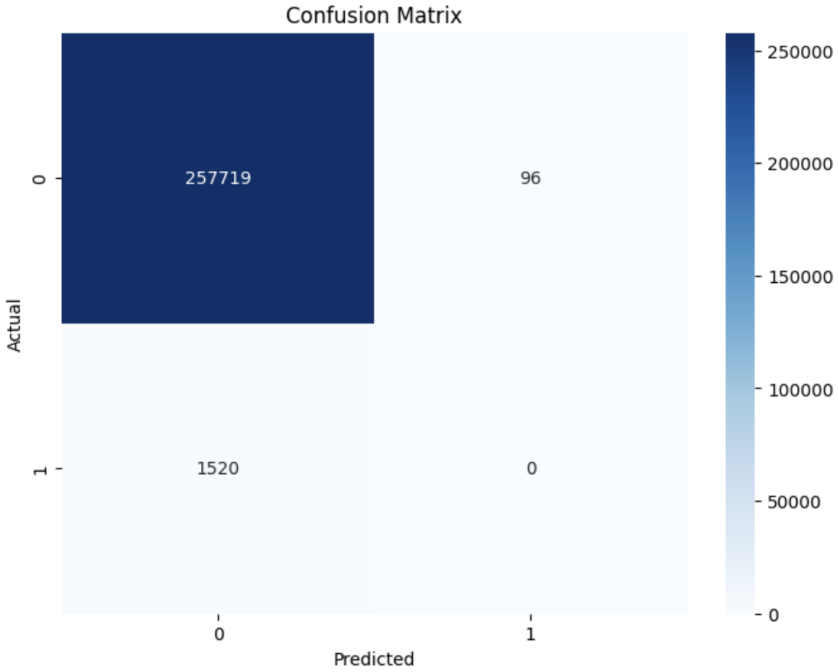
To expedite the training process for computationally intensive models, such as SVM, a subset of the oversampled training data—specifically the initial 100,000 instances—was selected. This strategic decision enables faster experimentation and model development without compromising the advantages gained from oversampling.

7. Models

- **Logistic Regression model**

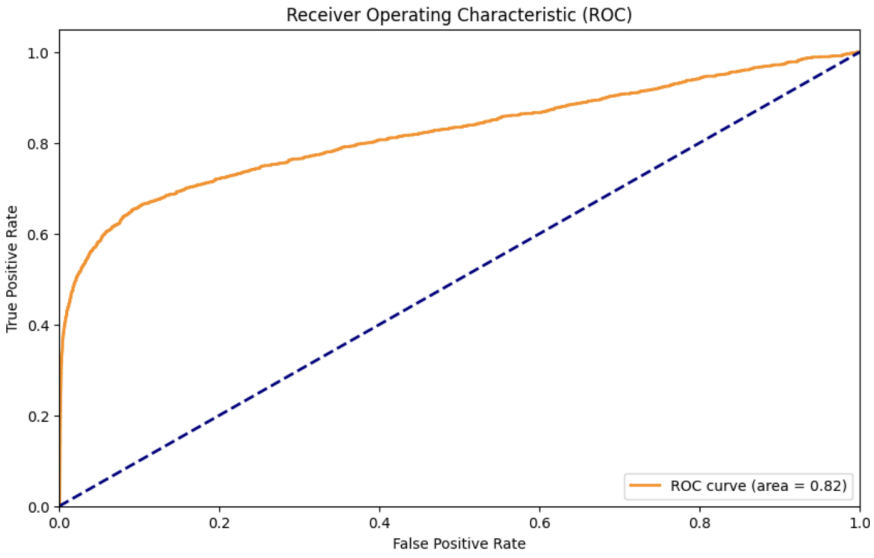
Logistic Regression Accuracy: 99.377%

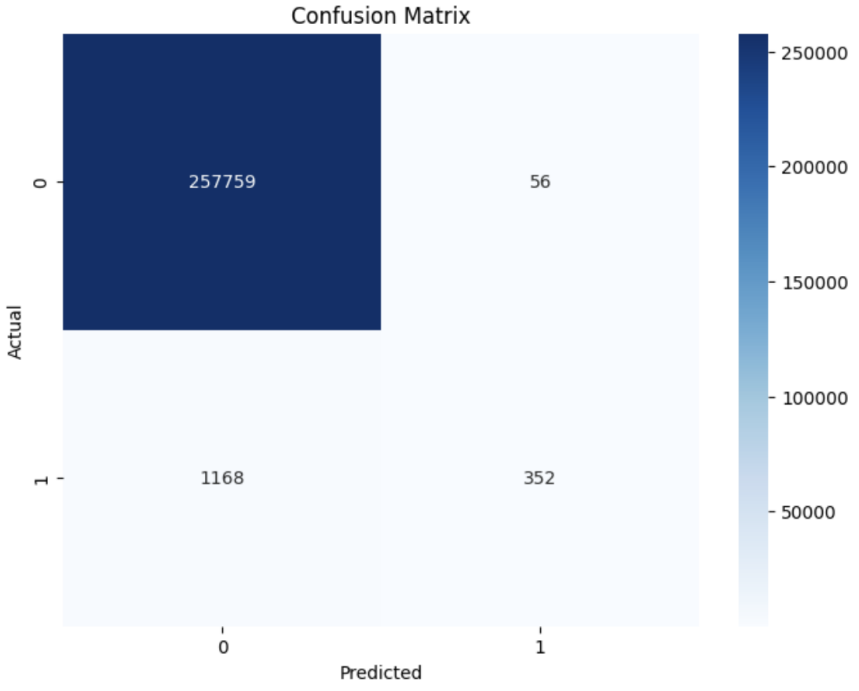




- **SVM model**

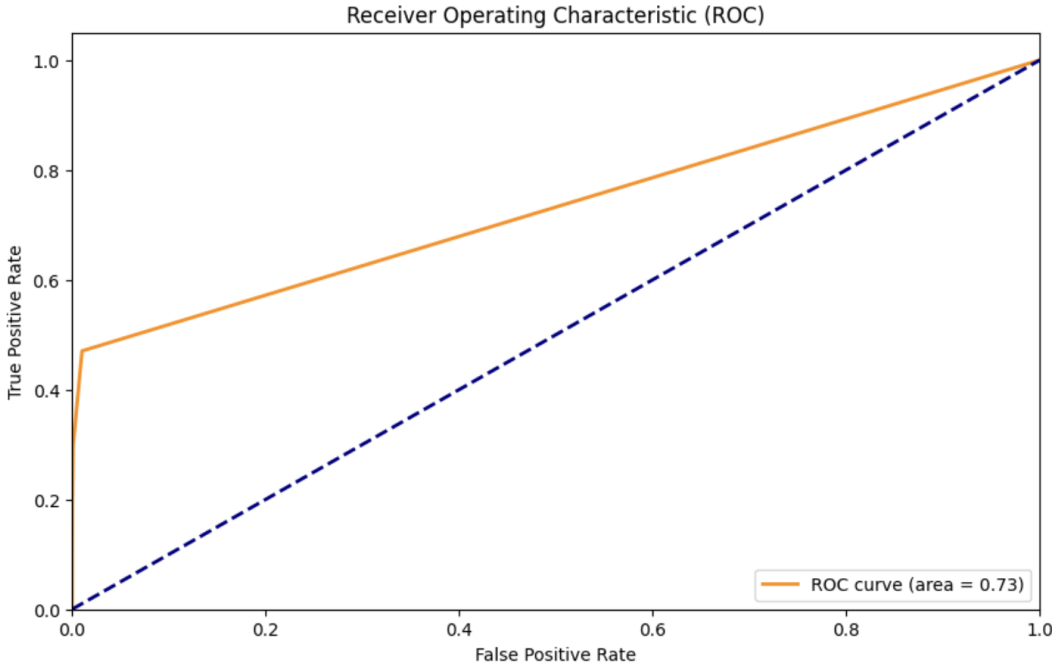
SVM Accuracy: 99.528%

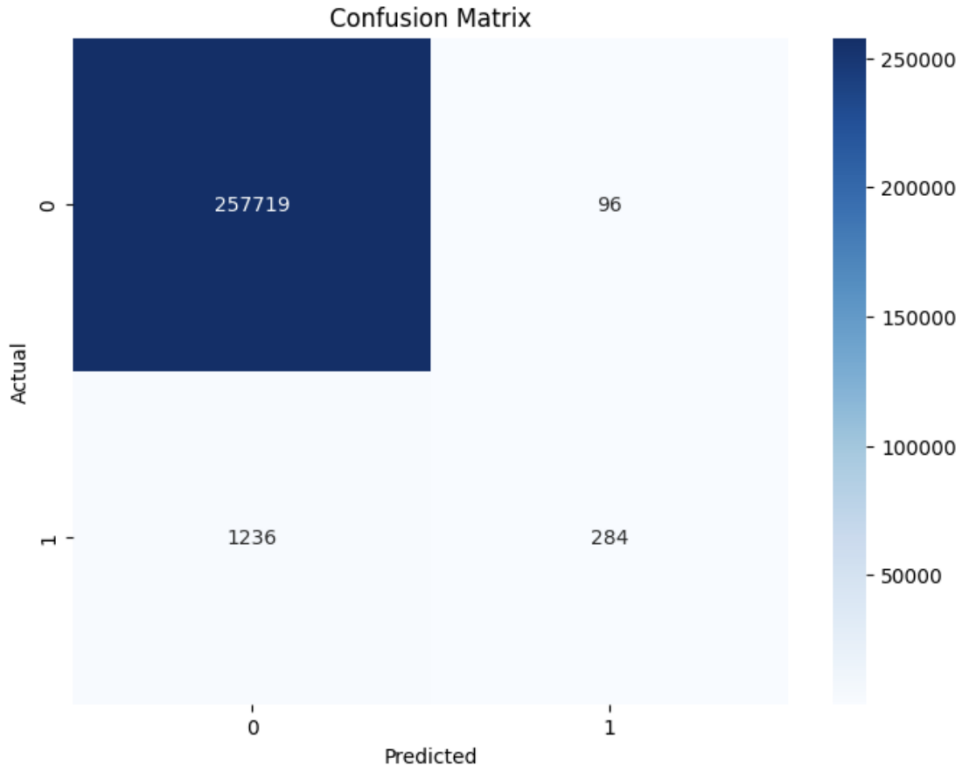




- KNN Model

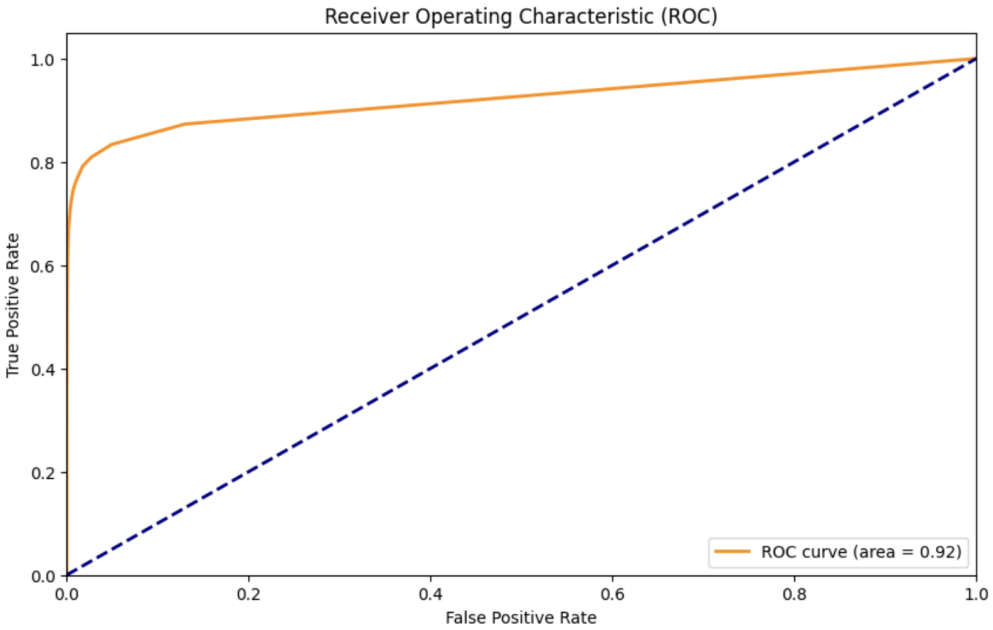
KNN Accuracy: 99.486%

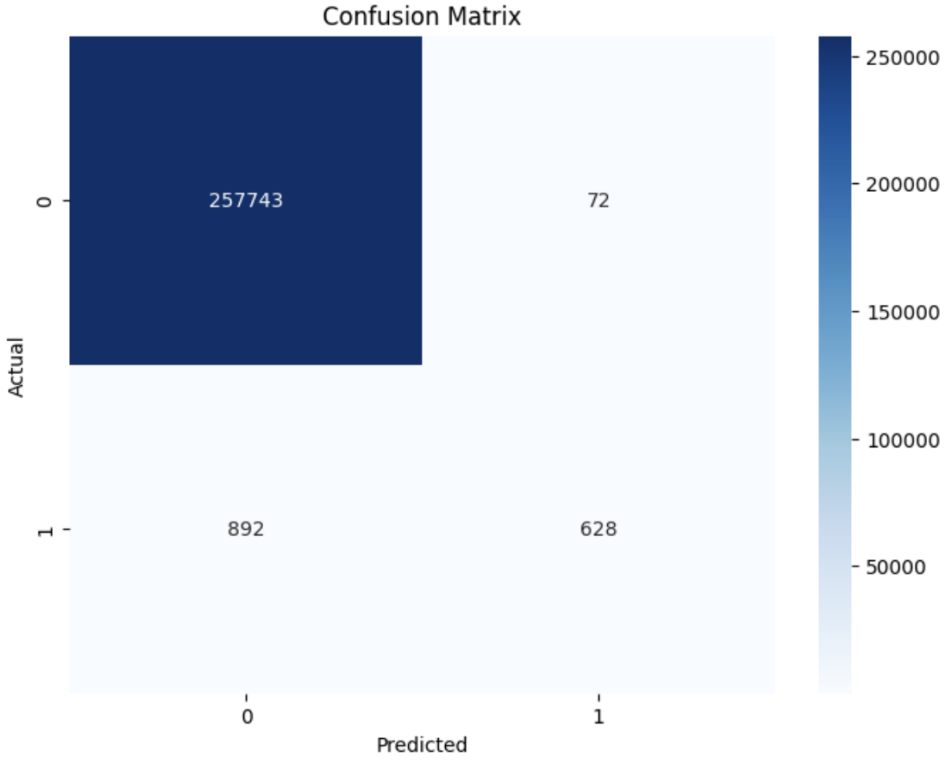




- Random Forest Model

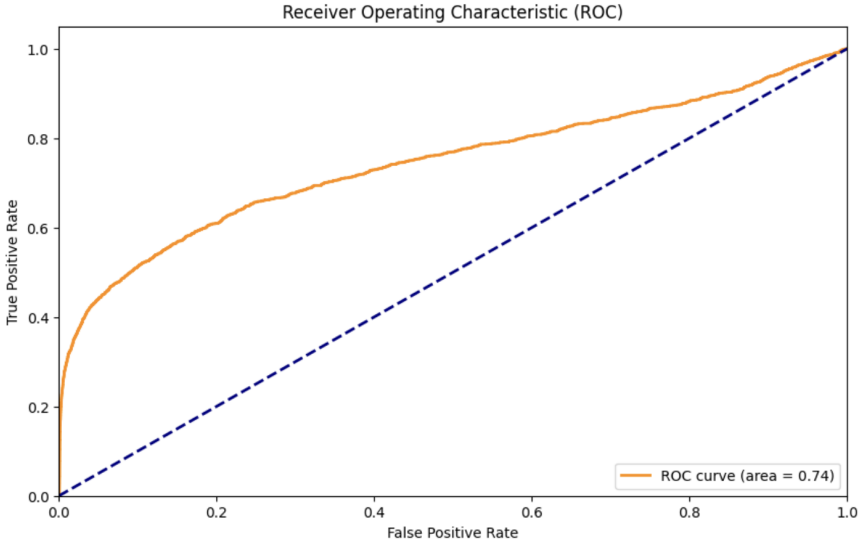
```
Random Forest Accuracy: 99.628%
```

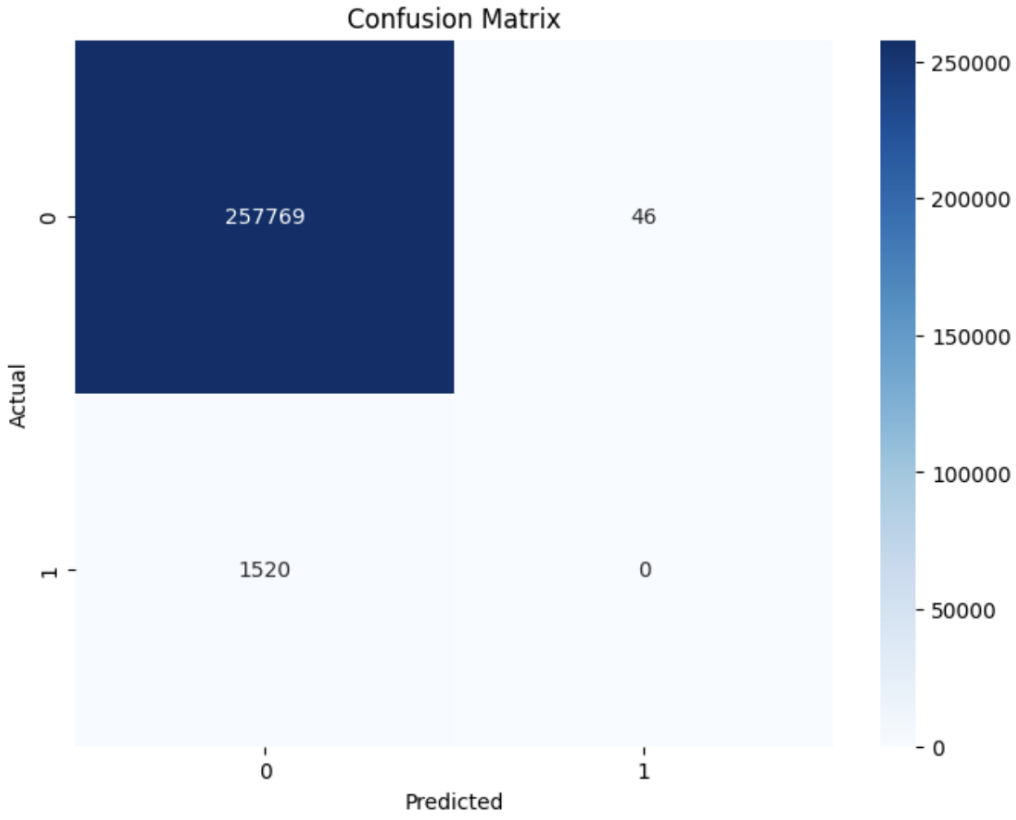




- **SGD model**

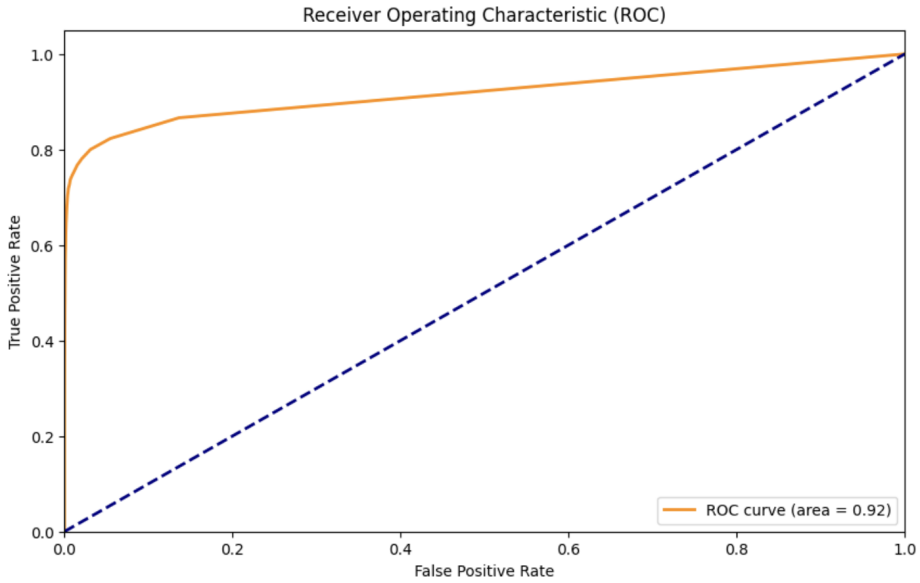
SGD Accuracy: 99.396%

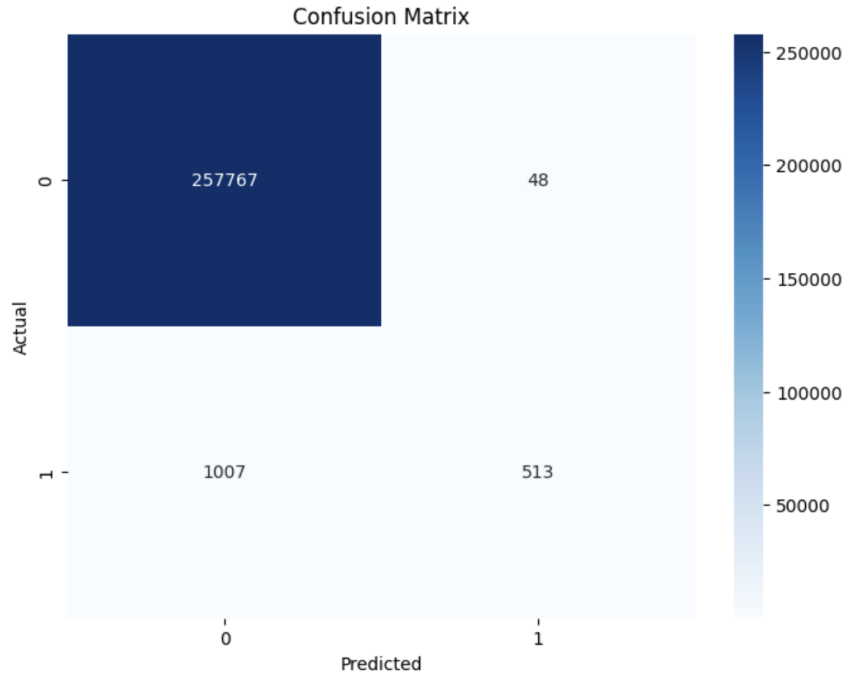




- **Extra Trees model**

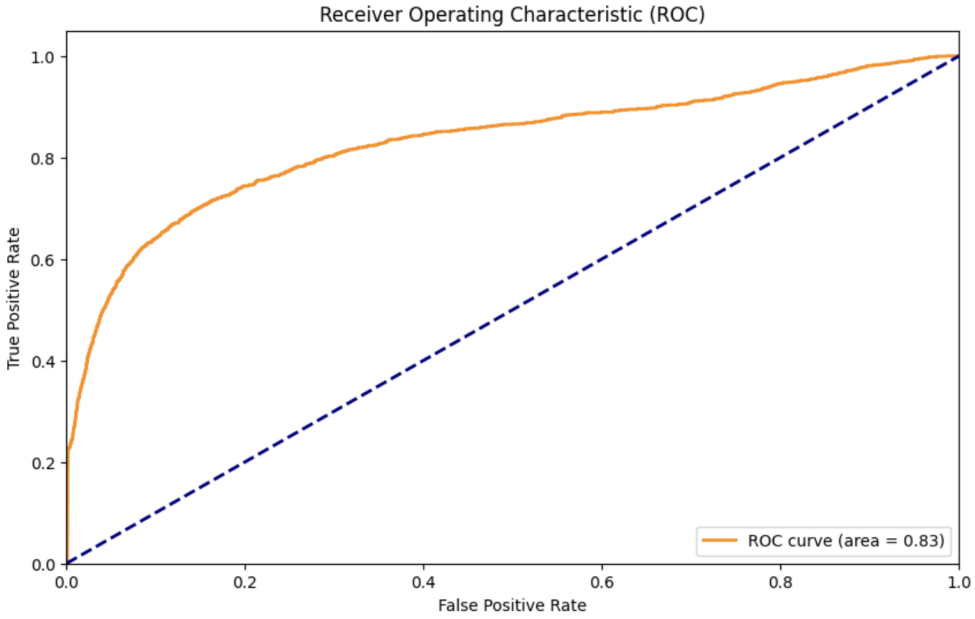
```
Extra Tree Accuracy: 99.593%
```

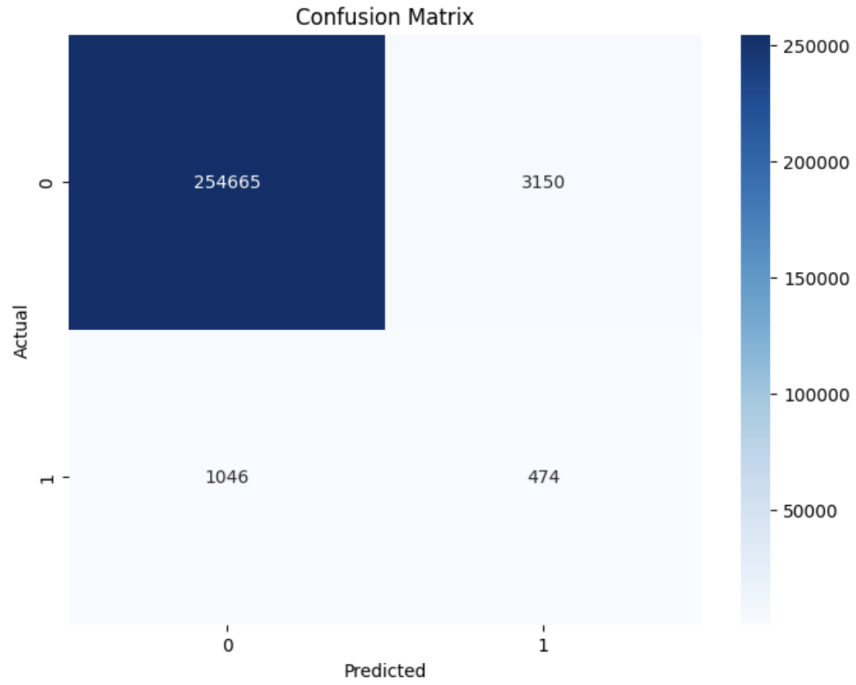




- **Gaussian Naive Bayes model**

```
Extra Tree Accuracy: 98.382%
```





```
Model: Logistic Regression
Accuracy: 0.993768677579193
ROC Score: 0.49981381998720015
F1 Score: 0.0
Precision Score: 0.0
Recall Score: 0.0
================================================
Model: SVM
Accuracy: 0.9952802359882006
ROC Score: 0.615680868676744
F1 Score: 0.3651452282157676
Precision Score: 0.8627450980392157
Recall Score: 0.23157894736842105
================================================
Model: KNN
Accuracy: 0.9948637862224535
ROC Score: 0.593234872618779
F1 Score: 0.2989473684210526
Precision Score: 0.7473684210526316
Recall Score: 0.1868421052631579
================================================
Model: Random Forest
Accuracy: 0.996282800239073
ROC Score: 0.7064393123588212
F1 Score: 0.5657657657657658
Precision Score: 0.8971428571428571
Recall Score: 0.4131578947368421
================================================
Model: SGD
Accuracy: 0.9939614783966684
ROC Score: 0.49991078874386674
F1 Score: 0.0
Precision Score: 0.0
Recall Score: 0.0
================================================
Model: Extra Trees
Accuracy: 0.9959319027512676
ROC Score: 0.6686569099936001
F1 Score: 0.49303219605958676
Precision Score: 0.9144385026737968
Recall Score: 0.3375

================================================
Model: Naive Bayes
Accuracy: 0.9838201553974589
ROC Score: 0.649812020961583
F1 Score: 0.18429237947122862
Precision Score: 0.13079470198675497
Recall Score: 0.3118421052631579
================================================
```

Now we use class_weight parameters. Because our dataset is imbalanced and the model may be biased toward predicting the majority class because it achieves high accuracy by simply predicting the majority class for most instances. In applications like fraud detection, the minority class (fraud) is usually of greater interest, and misclassifying it can have more severe consequences.

During model training, instances from the minority class (1) will contribute more to the loss function than instances from the majority class (0). Class_weight adjustment helps the model give higher importance to the minority class and better capture its patterns.
Purpose. The purpose of adjusting weights is to handle the class imbalance, making the model more sensitive to the minority class and improving its ability to correctly classify instances from both classes.

```
Logistic Regression Metrics:
Accuracy: 0.8430
Confusion Matrix:
[[217483  40332]
 [   391   1129]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.84      0.91    257815
           1       0.03      0.74      0.05      1520

    accuracy                           0.84    259335
   macro avg       0.51      0.79      0.48    259335
weighted avg       0.99      0.84      0.91    259335

============================================================
SVM Metrics:
Accuracy: 0.9641
Confusion Matrix:
[[248856   8959]
 [   352   1168]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.98    257815
           1       0.12      0.77      0.20      1520

    accuracy                           0.96    259335
   macro avg       0.56      0.87      0.59    259335
weighted avg       0.99      0.96      0.98    259335

============================================================
```

```
KNN Metrics:
Accuracy: 0.9950
Confusion Matrix:
[[257710    105]
 [  1194    326]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    257815
           1       0.76      0.21      0.33      1520

    accuracy                           0.99    259335
   macro avg       0.88      0.61      0.67    259335
weighted avg       0.99      0.99      0.99    259335
```

==========

```
Decision Tree Metrics:
Accuracy: 0.9947
Confusion Matrix:
[[257237    578]
 [   803    717]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    257815
           1       0.55      0.47      0.51      1520

    accuracy                           0.99    259335
   macro avg       0.78      0.73      0.75    259335
weighted avg       0.99      0.99      0.99    259335
```

==========

```
Random Forest Metrics:
Accuracy: 0.9960
Confusion Matrix:
[[257759     56]
 [   984    536]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    257815
           1       0.91      0.35      0.51      1520

    accuracy                           1.00    259335
   macro avg       0.95      0.68      0.75    259335
weighted avg       1.00      1.00      1.00    259335
```

==========

```
Naive Bayes Metrics:
Accuracy: 0.9838
Confusion Matrix:
[[254665   3150]
 [  1046    474]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    257815
           1       0.13      0.31      0.18      1520

    accuracy                           0.98    259335
   macro avg       0.56      0.65      0.59    259335
weighted avg       0.99      0.98      0.99    259335
```

Now for model selection let's consider some factors:

Accuracy:
Random Forest has the highest accuracy (0.9960), indicating overall correct predictions.
SVM and KNN also have high accuracy but slightly lower than Random Forest.

Precision and Recall:
Precision and recall are crucial for imbalanced datasets, especially in fraud detection.
Random Forest has a good balance, with high precision (0.91) and reasonable recall (0.35).
SVM has decent precision (0.12) and high recall (0.77).
KNN has high precision (0.76) but lower recall (0.21).

F1 Score:
F1 score considers both precision and recall. Random Forest has a decent F1 score (0.51).
SVM and KNN also have reasonable F1 scores.

Interpretability:
Decision Tree and Naive Bayes models are typically more interpretable than ensemble methods like Random Forest.

Computational Complexity:
SVM may be computationally intensive, especially with a large dataset.

Considering these factors, Random Forest seems like a good choice with a good balance between precision and recall, high accuracy, and a decent F1 score. However, it's always a good practice to fine-tune hyperparameters, cross-validate, and possibly conduct further experimentation to ensure the model's robustness. Additionally, interpreting the specific needs of your fraud detection application is crucial in making the final decision.