Fraud is bad. Don't commit no fraud

Data Import

In [ ]:
```
import pathlib
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
from scipy.stats import mannwhitneyu
from scipy.stats import chi2_contingency
```

In [2]:
```
# original file:
# https://www.kaggle.com/datasets/dhanushnarayananr/credit-card-fraud/download?datasetVersionNumber=1
url = 'card_transdata.csv'
df_fraud = pd.read_csv(url)
```

Exploratory Data Analysis

Data Qualick Check-list

- Check for Missing Data
- Check for Duplicates
- Validate Data Types
- Explore Unique Values
- Handle Outliers
- Cross-Validate Against External Sources
- Examine Summary Statistics
- Check for Data Skewness
- Visualize the Data

Missing Data Check

In [3]:
```
# Check for missing data and overall dataset overview
df_fraud.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 8 columns):
 #   Column                          Non-Null Count    Dtype
---  ------                          --------------    -----
 0   distance_from_home              1000000 non-null  float64
 1   distance_from_last_transaction  1000000 non-null  float64
 2   ratio_to_median_purchase_price  1000000 non-null  float64
 3   repeat_retailer                 1000000 non-null  float64
 4   used_chip                       1000000 non-null  float64
 5   used_pin_number                 1000000 non-null  float64
 6   online_order                    1000000 non-null  float64
 7   fraud                           1000000 non-null  float64
dtypes: float64(8)
memory usage: 61.0 MB
```

Duplicate Data Check

In [4]:
```
# As seen above, there are no missing values. Are there duplicates?
duplicates = df_fraud[df_fraud.duplicated()]
duplicates.shape
```

Out[4]:
```
(0, 8)
```

In [5]:
```
# There are no duplicates either. Good news so far!
```

Validate Data Types

In [6]:
```
# Let's check that all the variables are correct data types.
# Let's see first few rows for each.
```

```python
df_fraud.head(5)
```

Out[6]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_
0	57.877857	0.311140	1.945940	1.0	1.0	0.0	
1	10.829943	0.175592	1.294219	1.0	0.0	0.0	
2	5.091079	0.805153	0.427715	1.0	0.0	0.0	
3	2.247564	5.600044	0.362663	1.0	1.0	0.0	
4	44.190936	0.566486	2.222767	1.0	1.0	0.0	

In [7]:
```python
# In the df_fraud.head(5) above, it follows that
# 'distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price' are floats

# The rest of the variables 'repeat_retailer', 'used_chip', 'used_pin_number', 'online_order', 'fraud' appear to be
# categorical/ let's confirm this by calculating the number of unique values for each variable that is not continuous.

# If a variable has only a few unique values, its categorical
```

Exploring Unique Values

In [8]:
```python
df_categorical = df_fraud[['repeat_retailer', 'used_chip', 'used_pin_number', 'online_order', 'fraud']]
for var in df_categorical:
    print(df_categorical[var].value_counts())
    print('\n')
```

```
repeat_retailer
1.0    881536
0.0    118464
Name: count, dtype: int64


used_chip
0.0    649601
1.0    350399
Name: count, dtype: int64


used_pin_number
0.0    899392
1.0    100608
Name: count, dtype: int64


online_order
1.0    650552
0.0    349448
Name: count, dtype: int64


fraud
0.0    912597
1.0     87403
Name: count, dtype: int64
```

In [9]:
```python
# The above confirms that variables 'repeat_retailer', 'used_chip', 'used_pin_number', 'online_order', 'fraud'
# are binary, categorical variables with only possible classes 0 or 1. this is essentially a 0/1 label encoding already done

# we could convert them to object variables but its better to convert them into integers

#  converting binary categorical variables to integers is a common and efficient practice, especially
# when the variables naturally represent binary states (0 or 1).
# This facilitates numerical operations and saves memory compared to using object variables.

df_fraud[['repeat_retailer', 'used_chip', 'used_pin_number', 'online_order', 'fraud']] = \
df_fraud[['repeat_retailer', 'used_chip', 'used_pin_number', 'online_order', 'fraud']].astype(int)
```

In [10]:
```python
# lets check the data types after again after the conversion:
df_fraud.dtypes
```

```
Out[10]: distance_from_home                float64
         distance_from_last_transaction    float64
         ratio_to_median_purchase_price    float64
         repeat_retailer                     int32
         used_chip                           int32
         used_pin_number                     int32
         online_order                        int32
         fraud                               int32
         dtype: object
```

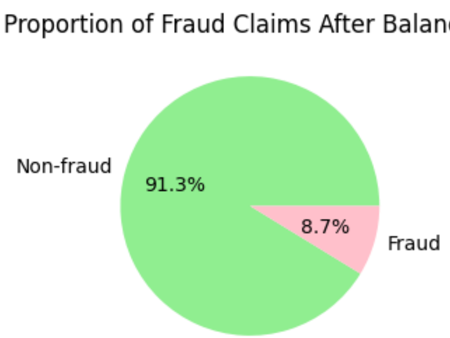
```
In [11]: # Now, let's take a look at the target variable, fraud to see the breakdown of classes (fraud vs non-fraud)

         fraud_cases = pd.DataFrame(df_fraud['fraud'].value_counts())
         fraud_cases['Percentage'] = round(df_fraud['fraud'].value_counts(normalize=True) * 100, 2)
         fraud_cases = fraud_cases.rename(columns={'fraud': 'Claims'})
         fraud_cases
```

Out[11]:

fraud	count	Percentage
0	912597	91.26
1	87403	8.74

Cross-Validate Against External Sources

```
In [12]: # Fraud is rare! The 4%-8% fraud rate is typical for this type of  datasets

         # for example: Nilson Report 2022: https://nilsonreport.com/
         # "The global credit card fraud claim rate was 4.25% in 2021, with total losses of $31.3 billion.
         # This represents a decrease of 1.6% from the 2020 fraud claim rate of 4.41%."

         # Lets make a pie chart to vizualize the proportion of fraud vs non-fraud cases:

         # Proportion of fraud claims
         fig, ax = plt.subplots(figsize=(4, 3))
         fraud_proportion = df_fraud['fraud'].value_counts(normalize=True)
         fraud_proportion.plot.pie(labels=['Non-fraud', 'Fraud'], autopct='%1.1f%%', ax=ax, colors=['lightgreen', 'pink'])  # Specify t

         # Remove y-axis label
         ax.set_ylabel('')

         plt.title('Proportion of Fraud Claims After Balancing')
         plt.show()
```



Summary Statistics

```
In [13]: # Let's now look at the summary statistics of the continuous variables:
         df_fraud[['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price']].describe()
```

Out[13]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price
count	1000000.000000	1000000.000000	1000000.000000
mean	26.628792	5.036519	1.824182
std	65.390784	25.843093	2.799589
min	0.004874	0.000118	0.004399
25%	3.878008	0.296671	0.475673
50%	9.967760	0.998650	0.997717
75%	25.743985	3.355748	2.096370
max	10632.723672	11851.104565	267.802942

In [14]:

```python
# Now let's look at the mean values of continuous variables in the dataset for fraud and no-fraude cases

means_by_fraud = df_fraud.groupby('fraud')[['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_
pd.DataFrame(round(means_by_fraud, 2))
```

Out[14]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price
fraud			
0	22.83	4.30	1.42
1	66.26	12.71	6.01

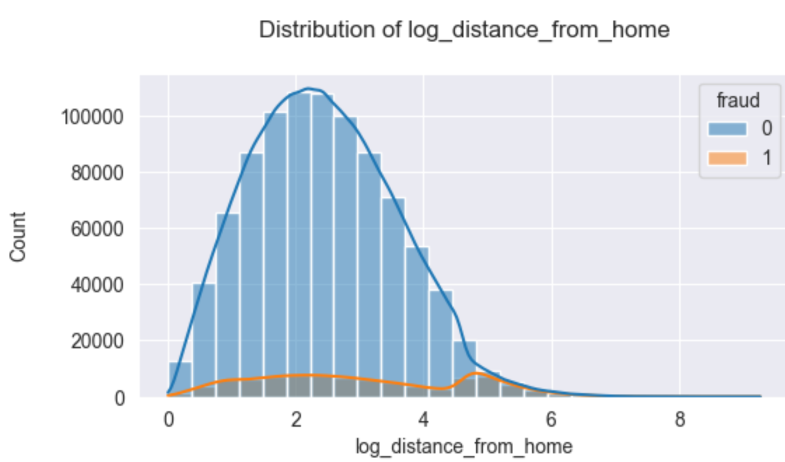
In [15]:

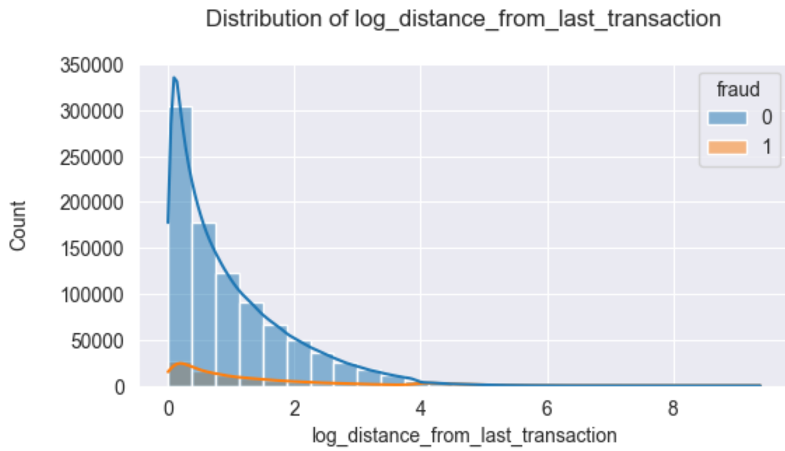
```python
# note the difference in the mean values of each of the three variables in fraud vs non-fraud cases
# There is a higher mean value of each of the three variables in fraud cases
# In other words, fraud when it happens tend to be associated with larger distance from home,
# larger distance from last transaction, and a larger ratio-to-median purchase price.
```

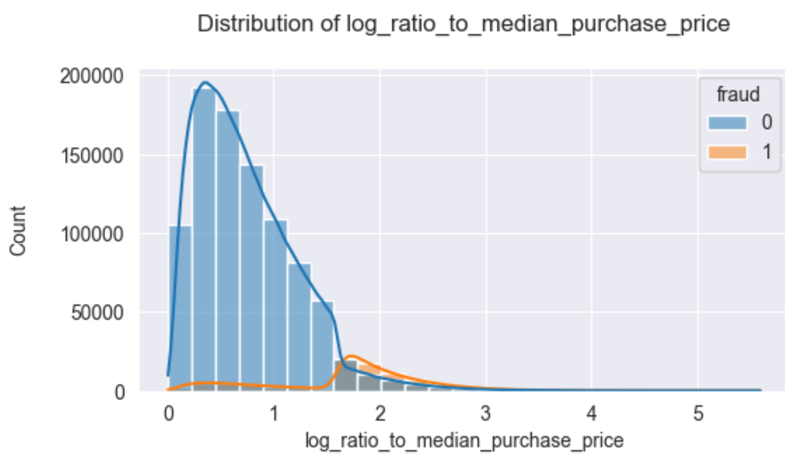
Checking for Data Skewness. Dealing with outliers

In [16]:

```python
# Step 1: Log-Transformation
df_fraud_log = pd.DataFrame()  # Create an empty DataFrame to store log-transformed values
for col in ['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price']:
    df_fraud[f'{col}'] = df_fraud[col]
    df_fraud[f'log_{col}'] = np.log1p(df_fraud[col])

# Step 2: Visualize Log-Transformed Distributions
for col in ['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price']:
    plt.figure(figsize=(6, 3))
    sns.set_style('darkgrid')
    sns.histplot(data=df_fraud, x=f'log_{col}', color='teal', kde=True, bins=25, hue='fraud')
    plt.title(f'Distribution of log_{col}\n')
    plt.xlabel(f'log_{col}')
    plt.ylabel('Count\n')
    plt.show()
```







In [17]:

```
#filtering the outliers based on 2 STD from the mean for the log-transformed values

original_cols = ['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price']

# Set a threshold
threshold = 2

# Create a copy of the original DataFrame
df_fraud_filtered = df_fraud.copy()

# Iterate through each column and filter outliers
for col in original_cols:
    mean_value = df_fraud[col].mean()
    std_dev = df_fraud[col].std()

    # Filter outliers based on the threshold
    df_fraud_filtered = df_fraud_filtered[(df_fraud_filtered[col] - mean_value).abs() <= threshold * std_dev]

df_fraud_filtered.head()
```

Out[17]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_
0	57.877857	0.311140	1.945940	1	1	0	
1	10.829943	0.175592	1.294219	1	0	0	
2	5.091079	0.805153	0.427715	1	0	0	
3	2.247564	5.600044	0.362663	1	1	0	
4	44.190936	0.566486	2.222767	1	1	0	

In [18]:

```
# dataset size before filtering
df_fraud.shape
```

Out[18]: (1000000, 11)

```
In [19]: # dataset size after filtering
         df_fraud_filtered.shape
```

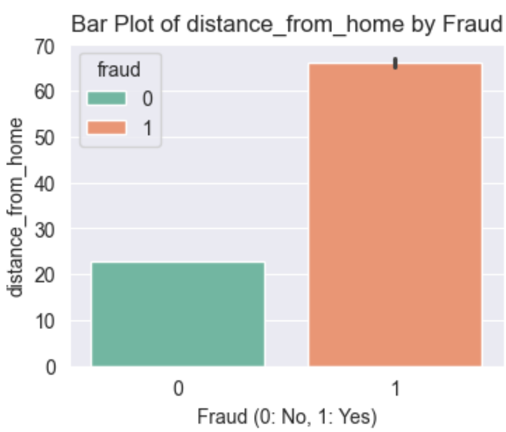
```
Out[19]: (930900, 11)
```

```
In [20]: df_fraud.columns
```

```
Out[20]: Index(['distance_from_home', 'distance_from_last_transaction',
                'ratio_to_median_purchase_price', 'repeat_retailer', 'used_chip',
                'used_pin_number', 'online_order', 'fraud', 'log_distance_from_home',
                'log_distance_from_last_transaction',
                'log_ratio_to_median_purchase_price'],
               dtype='object')
```

Vizualizing the Data

```
In [21]: # We already know that there is a difference in the mean values of the continuous variables in fraud vs non-fraud scenarios.
         # lets vizualize these differences with bar plots

         for col in df_fraud_filtered.columns:
             if col in ['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price']:

                 # Set the figure size
                 plt.figure(figsize=(4, 3))

                 # Set the seaborn style to 'darkgrid'
                 sns.set_style('darkgrid')

                 # Create a violin plot using Seaborn with the log-transformed y-axis
                 sns.barplot(data=df_fraud, x='fraud', y=col, palette='Set2', hue='fraud')

                 # Set title and labels
                 plt.title(f'Bar Plot of {col} by Fraud')
                 plt.xlabel('Fraud (0: No, 1: Yes)')
                 plt.ylabel(f'{col}')

                 # Show the plot
                 plt.show()
```



In [22]:
```
# Let's also plot the continuous variables using violin-plots to better see the distribution and spread
# of the variables.

# we will use log-transformed y-values for better vizualization:

for col in df_fraud_filtered.columns:
    if col in ['log_distance_from_home', 'log_distance_from_last_transaction', 'log_ratio_to_median_purchase_price']:

        # Set the figure size
        plt.figure(figsize=(6, 3))

        # Set the seaborn style to 'darkgrid'
        sns.set_style('darkgrid')

        # Create a violin plot using Seaborn with the log-transformed y-axis
        sns.violinplot(data=df_fraud_filtered, x='fraud', y=col, palette='Set2', hue='fraud')

        # Set title and labels
        plt.title(f'Violin Plot of {col} by Fraud')
        plt.xlabel('Fraud (0: No, 1: Yes)')
        plt.ylabel(f'{col}')

        # Show the plot
        plt.show()
```

Checking the Association between Independent Variables and The Target Variable

In [23]:
```
# for the continuous variables and binary outcome,
# it is appropriate to use the Mann-Whitney U test for independent samples

# Create an empty DataFrame to store the results
mannwhitney_results = pd.DataFrame(columns=['Variable', 'Mann-Whitney U', 'P-value', 'Significance'])

for col in df_fraud_filtered.columns:
    if col in ['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price']:
        # Perform Mann-Whitney U test
        statistic, p_value = mannwhitneyu(df_fraud[df_fraud['fraud'] == 1][col], df_fraud[df_fraud['fraud'] == 0][col])

        # Determine significance and append the results to the mannwhitney_results DataFrame
        significance = '*' if p_value < 0.05 else ''
        mannwhitney_results = pd.concat([mannwhitney_results, pd.DataFrame({
            'Variable': [col],
            'Mann-Whitney U': [statistic],
            'P-value': [p_value],
            'Significance': [significance]
        })], ignore_index=True)

mannwhitney_results_sorted = mannwhitney_results.sort_values(by=['Mann-Whitney U', 'Significance'], ascending=[False, True])
print(mannwhitney_results_sorted)
```

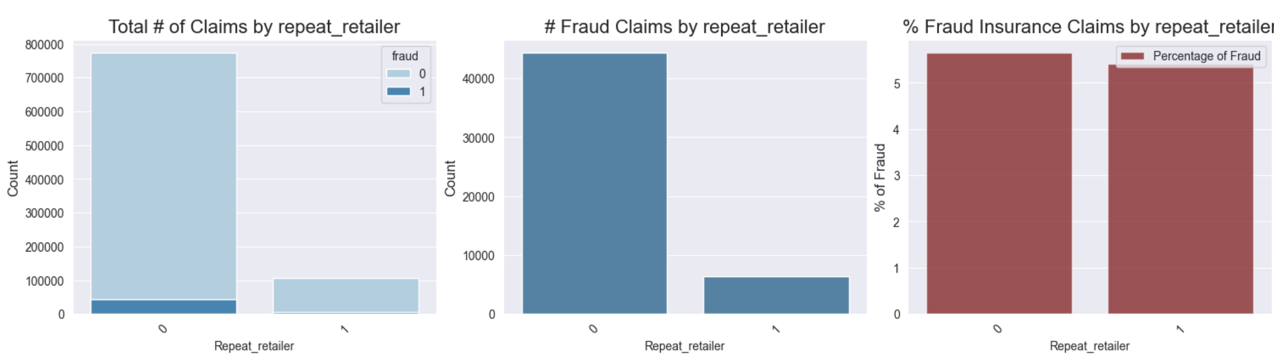
```
C:\Users\LLANA\AppData\Local\Temp\ipykernel_2680\2130122949.py:14: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determinin
g the result dtypes. To retain the old behavior, exclude the relevant entries before the concat operation.
  mannwhitney_results = pd.concat([mannwhitney_results, pd.DataFrame({
                         Variable  Mann-Whitney U        P-value Significance
2  ratio_to_median_purchase_price    6.783310e+10   0.000000e+00            *
0              distance_from_home    4.762976e+10   0.000000e+00            *
1  distance_from_last_transaction    4.270774e+10  3.046020e-263            *
```

In [24]:
```
# Now, let's turn to our non-continous variables. Lets plot them and evaluate the impact of is on
# the target variable, fraud
```

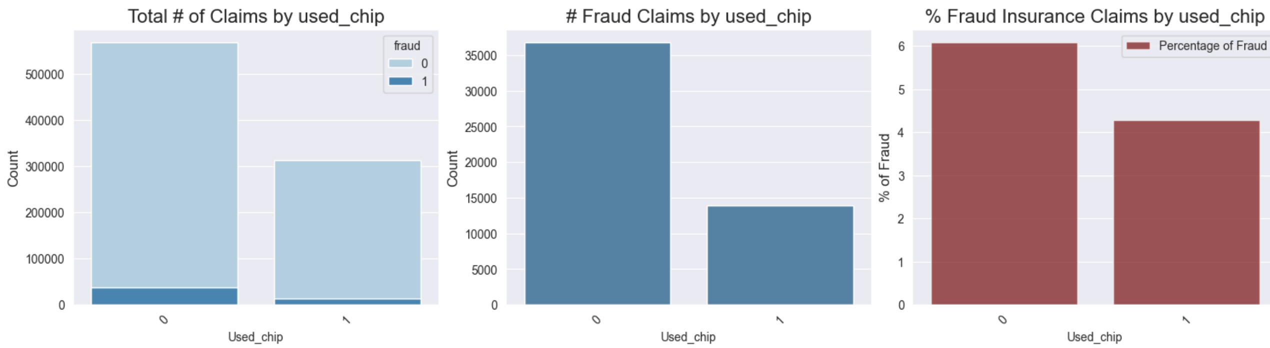
In [25]:
```
# Iterate through columns in df_fraud_filtered
for col in df_fraud_filtered.columns:
    if col in ['repeat_retailer', 'used_chip', 'used_pin_number', 'online_order']:
        sns.set_style('darkgrid')
```

```python
    # Create a DataFrame for count and percentage of fraud cases
    fraud_cases = pd.DataFrame(df_fraud_filtered.groupby([col, 'fraud']).size(), columns=['Count']).reset_index()
    total_counts = fraud_cases.groupby(col)['Count'].transform('sum')
    fraud_cases['Percentage of Fraud'] = round(fraud_cases['Count'] / total_counts * 100, 2)

    if (fraud_cases['fraud'] == 1).any():
        total_fraud_cases = fraud_cases[fraud_cases['fraud'] == 1]['Count'].sum()
        fraud_cases.loc[fraud_cases['fraud'] == 1, 'Percentage of Total Fraud Cases'] = round(fraud_cases['Count'] / total

    fraud_cases = fraud_cases.sort_values(by=['fraud', 'Percentage of Fraud', col], ascending=[False, False, True])

    # Print the fraud_cases DataFrame
    print('\n')
    print(fraud_cases.to_string(index=False))
    print('\n')

    # Plot three graphs:
    fig, axes = plt.subplots(1, 3, figsize=(15, 4), constrained_layout=True)  # Use constrained_layout for better layout

    # Plot 1: the count of fraud (both 0 and 1) for each category of a given variable
    axes[0].set_title(f'Total # of Claims by {col}', fontsize=16)
    sns.countplot(data=df_fraud_filtered, x=col, hue='fraud', palette='Blues', dodge=False, order=df_fraud_filtered[col].v
    axes[0].set_xticks([0, 1])  # Set x-axis ticks to be 0 and 1
    axes[0].set_xticklabels(axes[0].get_xticks(), rotation=45)  # Remove ha='right'
    axes[0].set_xlabel(col.capitalize())
    axes[0].set_ylabel('Count', fontsize=12)

    # Plot 2: the count of fraud == 1 for each category of a given variable
    axes[1].set_title(f'# Fraud Claims by {col}', fontsize=16)
    sns.countplot(data=df_fraud_filtered[df_fraud_filtered['fraud'] == 1], x=col, color='#4884af', dodge=False, order=df_f
    axes[1].set_xticks([0, 1])  # Set x-axis ticks to be 0 and 1
    axes[1].set_xticklabels(axes[1].get_xticks(), rotation=45)  # Remove ha='right'
    axes[1].set_xlabel(col.capitalize())
    axes[1].set_ylabel('Count', fontsize=12)

    # Plot 3: the % of fraud == 1 for each category of a given variable
    axes[2].set_title(f'% Fraud Insurance Claims by {col}', fontsize=16)
    fraud_cases_subset = fraud_cases[fraud_cases['fraud'] == 1]
    fraud_cases_subset = fraud_cases_subset.sort_values(by='Percentage of Fraud', ascending=False)  # Sort by Percentage o
    sns.barplot(x=fraud_cases_subset[col], y=fraud_cases_subset['Percentage of Fraud'], color='darkred', label='Percentage
    axes[2].set_xticks([0, 1])  # Set x-axis ticks to be 0 and 1
    axes[2].set_xticklabels(axes[2].get_xticks(), rotation=45)  # Remove ha='right'
    axes[2].set_xlabel(col.capitalize())
    axes[2].set_ylabel('% of Fraud', fontsize=12)

    plt.show()
```

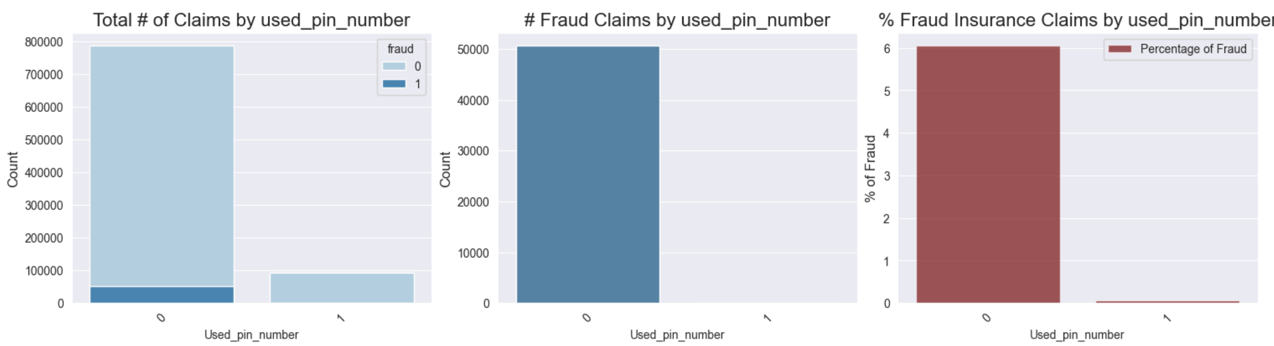
```
repeat_retailer  fraud  Count  Percentage of Fraud  Percentage of Total Fraud Cases
              0      1   6386                 5.65                            12.59
              1      1  44335                 5.42                            87.41
              1      0 773531                94.58                              NaN
              0      0 106648                94.35                              NaN
```



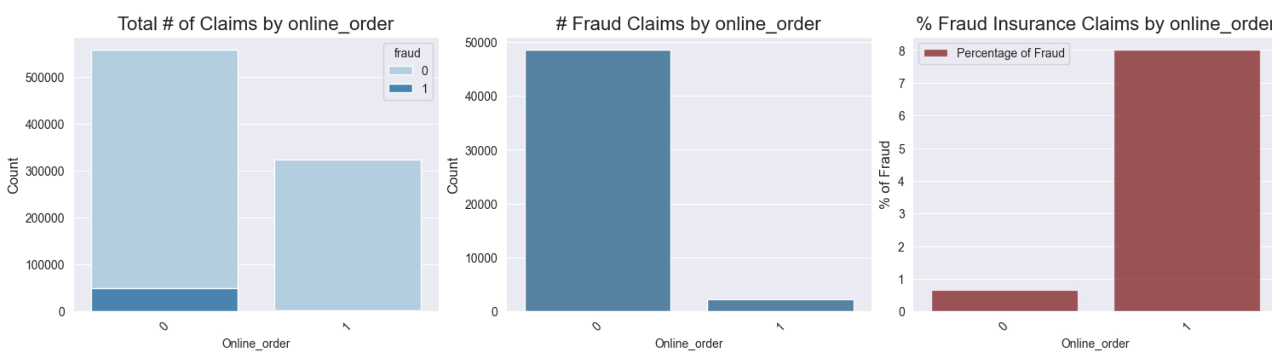

```
used_chip  fraud  Count  Percentage of Fraud  Percentage of Total Fraud Cases
        0      1  36775                 6.08                             72.5
        1      1  13946                 4.28                             27.5
        1      0 312218                95.72                              NaN
        0      0 567961                93.92                              NaN
```




```
used_pin_number  fraud   Count  Percentage of Fraud  Percentage of Total Fraud Cases
              0      1   50653                 6.05                            99.87
              1      1      68                 0.07                             0.13
              1      0   93529                99.93                              NaN
              0      0  786650                93.95                              NaN
```




```
online_order  fraud   Count  Percentage of Fraud  Percentage of Total Fraud Cases
           1      1   48529                 8.01                            95.68
           0      1    2192                 0.67                             4.32
           0      0  323058                99.33                              NaN
           1      0  557121                91.99                              NaN
```



Chi-Square Test to check the assoication between categorical variables and the Target

In [26]:
```python
# As was the case with continuous variables somewhere above, let's now
# explore which categorical variables have a significant impact
# on the target variable (fraud) - we will use chi-square test

# Create an empty DataFrame to store the results
chi2_results = pd.DataFrame(columns=['Variable', 'Chi2', 'P-value', 'Significance'])

for col in df_fraud_filtered.columns:
    if col in ['repeat_retailer', 'used_chip', 'used_pin_number', 'online_order']:  # Add the missing colon
        # Create a contingency table
        contingency_table = pd.crosstab(df_fraud_filtered[col], df_fraud_filtered['fraud'])

        # Perform the chi-square test
        chi2, p, _, _ = chi2_contingency(contingency_table)

        # Determine significance and append the results to the chi2_results DataFrame
```

```
        significance = '*' if p < 0.05 else ''
        chi2_results = pd.concat([chi2_results, pd.DataFrame({'Variable': [col], 'Chi2': [chi2], 'P-value': [p], 'Significance
        chi2_results_sorted = chi2_results.sort_values(by=['Chi2'], ascending=[False])

print(chi2_results_sorted)
```

```
C:\Users\LLANA\AppData\Local\Temp\ipykernel_2680\490236676.py:18: FutureWarning: The behavior of DataFrame concatenation with e
mpty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determining
the result dtypes. To retain the old behavior, exclude the relevant entries before the concat operation.
  chi2_results = pd.concat([chi2_results, pd.DataFrame({'Variable': [col], 'Chi2': [chi2], 'P-value': [p], 'Significance': [sig
nificance]})], ignore_index=True)
          Variable          Chi2        P-value Significance
3     online_order  22120.854196   0.000000e+00            *
2  used_pin_number   5836.507923   0.000000e+00            *
1        used_chip   1340.234501  2.040807e-293            *
0  repeat_retailer     10.048080   1.525068e-03            *
```

In [27]:
```
# Overall, only repeat_retailer did not seem to have a signficant impact on the target variable.
```

Modelling

In [28]:
```
# lets create a copy of df_fraud and start modelling!
df_fraud_for_modeling = df_fraud_filtered.copy()
df_fraud_for_modeling = \
df_fraud_for_modeling.drop(['log_distance_from_home', 'log_distance_from_last_transaction', 'log_ratio_to_median_purchase_pric
```

In [29]:
```
df_fraud_for_modeling.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 930900 entries, 0 to 999999
Data columns (total 8 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   distance_from_home              930900 non-null  float64
 1   distance_from_last_transaction  930900 non-null  float64
 2   ratio_to_median_purchase_price  930900 non-null  float64
 3   repeat_retailer                 930900 non-null  int32  
 4   used_chip                       930900 non-null  int32  
 5   used_pin_number                 930900 non-null  int32  
 6   online_order                    930900 non-null  int32  
 7   fraud                           930900 non-null  int32  
dtypes: float64(3), int32(5)
memory usage: 46.2 MB
```

In [30]:
```
# splitting the dataset into test and training
from sklearn.model_selection import train_test_split

# Separating features and target variable
X = df_fraud_for_modeling.drop('fraud', axis=1)
y = df_fraud_for_modeling['fraud']

# Splitting the dataset 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, stratify=y)
```

In [31]:
```
# Important note! Since its essentially an anomaly detection analysis, its crucial that the evaluation
# metric captured both true negative and true positive. We cannot rely on overall accuracy of the model
# since even if the model gets all true-positives wrong (i.e. only correctly identifies true negatives),
# it will show an overall  high score (e.g 92%)

# For this reason, our evaluation metric of choice is F1-score
```

Logistic Regression (no balancing)

In [32]:
```
# Logistic regression - baseline, without balancing

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Create a Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000, random_state=42)

# Train the model on the balanced dataset
logreg_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_lg = logreg_model.predict(X_test)

# Evaluate the performance of the model
```

```
conf_matrix = confusion_matrix(y_test, y_pred_lg)
classification_rep = classification_report(y_test, y_pred_lg)

# Print the results
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", classification_rep)
```

```
Confusion Matrix:
 [[174643   1393]
 [  2880   7264]]

Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.99      0.99    176036
           1       0.84      0.72      0.77     10144

    accuracy                           0.98    186180
   macro avg       0.91      0.85      0.88    186180
weighted avg       0.98      0.98      0.98    186180
```

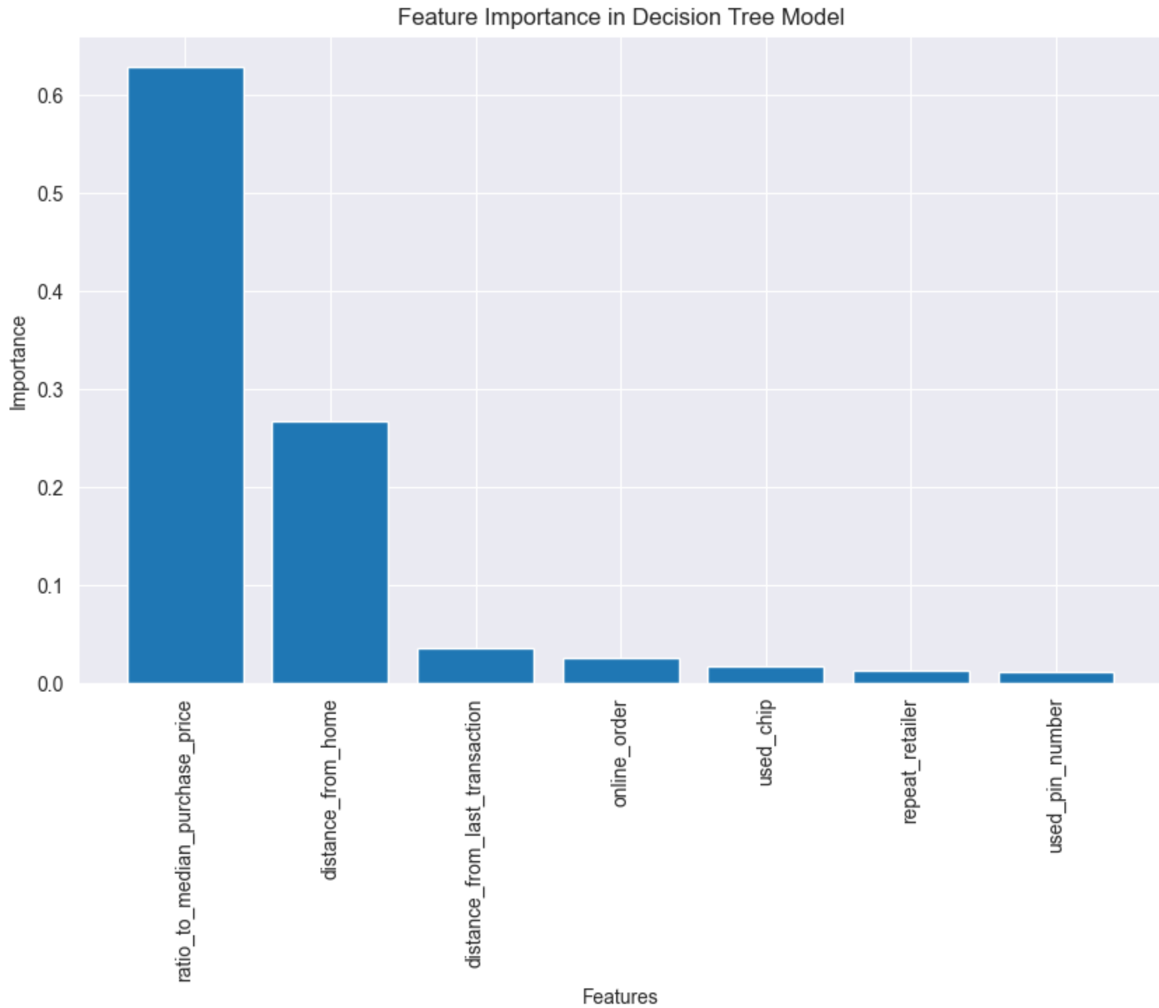
Decision Tree Classifier (no balancing)

In [33]:
```
# Decision Tree without balancing

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Create a Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Train the model on the SMOTE dataset
dt_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = dt_model.predict(X_test)

# Evaluate the performance of the model
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
classification_rep_dt = classification_report(y_test, y_pred_dt)

# Print the results
print("\nDecision Tree Confusion Matrix:\n", conf_matrix_dt)
print("\nDecision Tree Classification Report:\n", classification_rep_dt)
```

```
Decision Tree Confusion Matrix:
 [[176033      3]
 [     2  10142]]

Decision Tree Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    176036
           1       1.00      1.00      1.00     10144

    accuracy                           1.00    186180
   macro avg       1.00      1.00      1.00    186180
weighted avg       1.00      1.00      1.00    186180
```

In [46]:
```
feature_importances = dt_model.feature_importances_
sorted_features = sorted(zip(X_train.columns, feature_importances), key=lambda x: x[1], reverse=True)
print("\nMost Important Features:")
for feature, importance in sorted_features:
    print(f"Feature: {feature}, Importance: {importance:.3f}")

# Plot feature importance
plt.figure(figsize=(10, 6))
sorted_importances = [importance for feature, importance in sorted_features]
plt.bar(range(len(sorted_importances)), sorted_importances, align="center")
plt.xticks(range(len(sorted_importances)), [feature for feature, importance in sorted_features], rotation='vertical')
plt.xlabel("Features")
plt.ylabel("Importance")
plt.title("Feature Importance in Decision Tree Model")
plt.show()
```

```
Most Important Features:
Feature: ratio_to_median_purchase_price, Importance: 0.629
Feature: distance_from_home, Importance: 0.267
Feature: distance_from_last_transaction, Importance: 0.036
Feature: online_order, Importance: 0.026
Feature: used_chip, Importance: 0.018
Feature: repeat_retailer, Importance: 0.013
Feature: used_pin_number, Importance: 0.012
```



In [51]:
```python
from sklearn.model_selection import cross_val_score

# Perform 5-fold cross-validation
cv_scores = cross_val_score(dt_model, X_train, y_train, cv=5, scoring='f1')

# Print the cross-validation scores
print("Cross-Validation Scores:", cv_scores)

# Print the mean and standard deviation of the scores
print(f"Mean F1: {cv_scores.mean():.3f}")
print(f"Standard Deviation: {cv_scores.std():.3f}")
```

```
Cross-Validation Scores: [0.99981512 0.99981517 0.99975351 0.99969199 0.99950696]
Mean F1: 1.000
Standard Deviation: 0.000
```

Additional Models

Random Forest Classifier (no balancing)

In [34]:
```python
# Logistic Regression didnt perform all that great. However, Deciion Tree did.
# At this point, we want to see which other model can achieve the same result:

# Random Forest without balancing

from sklearn.ensemble import RandomForestClassifier
# Create a Random Forest model
```

```
rf_model = RandomForestClassifier(random_state=42)

# Train the model on the balanced dataset
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the performance of the model
# accuracy_rf = accuracy_score(y_test_balanced, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
classification_rep_rf = classification_report(y_test, y_pred_rf)

# Print the results
print("\nRandom Forest Confusion Matrix:\n", conf_matrix_rf)
print("\nRandom Forest Classification Report:\n", classification_rep_rf)
print(f'Size of the x-train, y-train, x-test, y-test: {len(X_train), len(y_train), len(X_test), len(y_test)}')
```

```
Random Forest Confusion Matrix:
 [[176036      0]
 [     3  10141]]

Random Forest Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    176036
           1       1.00      1.00      1.00     10144

    accuracy                           1.00    186180
   macro avg       1.00      1.00      1.00    186180
weighted avg       1.00      1.00      1.00    186180

Size of the x-train, y-train, x-test, y-test: (744720, 744720, 186180, 186180)
```

XGBoost (no balancing)

In [35]:
```
# XGBoost without balancing:
import xgboost as xgb

# Create an XGBoost model
xgb_model = xgb.XGBClassifier(random_state=42)

# Train the model on the training set
xgb_model.fit(X_train, y_train)

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

# Evaluate the performance of the model
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", classification_rep)
```

```
Confusion Matrix:
 [[175815    221]
 [   197   9947]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    176036
           1       0.98      0.98      0.98     10144

    accuracy                           1.00    186180
   macro avg       0.99      0.99      0.99    186180
weighted avg       1.00      1.00      1.00    186180
```

[[True Negative (TN) False Positive (FP)]

[False Negative (FN) True Positive (TP)]]

Balancing. Oversampling the minority class

In [36]:
```
# Applying SMOTE to oversample the minority class cases:
#!pip install imbalanced-learn
```

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

In [37]:
```
y_train_smote.value_counts()
```

Out[37]:
```
fraud
0    704143
1    704143
Name: count, dtype: int64
```

Logistic Regression (AFTER balancing)

In [38]:
```
# Logistic Regression after applying SMOTE:
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Create a Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000, random_state=42)

# Train the model on the balanced dataset
logreg_model.fit(X_train_smote, y_train_smote)

# Make predictions on the test set
y_pred = logreg_model.predict(X_test)

# Evaluate the performance of the model
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", classification_rep)
```

```
Confusion Matrix:
 [[166139   9897]
 [   319   9825]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.94      0.97    176036
           1       0.50      0.97      0.66     10144

    accuracy                           0.95    186180
   macro avg       0.75      0.96      0.81    186180
weighted avg       0.97      0.95      0.95    186180
```

Decision Tree (AFTER balancing)

In [39]:
```
# Decision Tree after SMOTE:

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Create a Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Train the model on the SMOTE dataset
dt_model.fit(X_train_smote, y_train_smote)

# Make predictions on the test set
y_pred_dt = dt_model.predict(X_test)

# Evaluate the performance of the model
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
classification_rep_dt = classification_report(y_test, y_pred_dt)

# Print the results
print("\nDecision Tree Confusion Matrix:\n", conf_matrix_dt)
print("\nDecision Tree Classification Report:\n", classification_rep_dt)
```

```
Decision Tree Confusion Matrix:
 [[175998     38]
 [     2  10142]]

Decision Tree Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    176036
           1       1.00      1.00      1.00     10144

    accuracy                           1.00    186180
   macro avg       1.00      1.00      1.00    186180
weighted avg       1.00      1.00      1.00    186180
```

Random Forest (AFTER balancing)

In [40]:

```python
# Random Forest after applying SMOTE

# Create a Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Train the model on the balanced dataset
rf_model.fit(X_train_smote, y_train_smote)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the performance of the model
# accuracy_rf = accuracy_score(y_test_balanced, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
classification_rep_rf = classification_report(y_test, y_pred_rf)

# Print the results
print("\nRandom Forest Confusion Matrix:\n", conf_matrix_rf)
print("\nRandom Forest Classification Report:\n", classification_rep_rf)
```

```
Random Forest Confusion Matrix:
 [[176007     29]
 [     2  10142]]

Random Forest Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    176036
           1       1.00      1.00      1.00     10144

    accuracy                           1.00    186180
   macro avg       1.00      1.00      1.00    186180
weighted avg       1.00      1.00      1.00    186180
```

XGBoost (AFTER balancing)

In [41]:

```python
# XGBoost After SMOTE:
import xgboost as xgb

# Create an XGBoost model
xgb_model = xgb.XGBClassifier(random_state=42)

# Train the model on the training set
xgb_model.fit(X_train_smote, y_train_smote)

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

# Evaluate the performance of the model
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", classification_rep)
```

```
Confusion Matrix:
 [[175801    235]
 [    46  10098]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    176036
           1       0.98      1.00      0.99     10144

    accuracy                           1.00    186180
   macro avg       0.99      1.00      0.99    186180
weighted avg       1.00      1.00      1.00    186180
```

Artificial Neural Network (AFTER balancing)

In [42]:

```python
# NN model applied after SMOTE

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.callbacks import EarlyStopping

# Split the data into training and validation sets
X_train_split, X_val_split, y_train_split, y_val_split =  \
train_test_split(X_train_smote, y_train_smote, test_size=0.2, random_state=42)

# Define early stopping callback
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=3)

# Create a Sequential model
model = Sequential()

# Add layers to the model
model.add(Dense(128, input_dim=X_train_smote.shape[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model on the training set with validation data
model.fit(X_train_split.values, y_train_split.values, epochs=20, batch_size=64, validation_data=(X_val_split.values, y_val_spl

# Evaluate the performance of the model
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", classification_rep)
```

```
WARNING:tensorflow:From C:\Users\LLANA\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\losses.py:2976: The
name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

WARNING:tensorflow:From C:\Users\LLANA\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\backend.py:873: The
name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\LLANA\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\optimizers\__init__.
py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/20
WARNING:tensorflow:From C:\Users\LLANA\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\utils\tf_utils.py:49
2: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\LLANA\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\engine\base_layer_ut
ils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_fu
nctions instead.

17604/17604 [==============================] - 51s 3ms/step - loss: 0.0346 - accuracy: 0.9881 - val_loss: 0.0232 - val_accurac
y: 0.9918
Epoch 2/20
17604/17604 [==============================] - 50s 3ms/step - loss: 0.0132 - accuracy: 0.9955 - val_loss: 0.0101 - val_accurac
y: 0.9961
Epoch 3/20
17604/17604 [==============================] - 50s 3ms/step - loss: 0.0105 - accuracy: 0.9964 - val_loss: 0.0062 - val_accurac
y: 0.9978
Epoch 4/20
17604/17604 [==============================] - 50s 3ms/step - loss: 0.0092 - accuracy: 0.9968 - val_loss: 0.0133 - val_accurac
y: 0.9941
Epoch 5/20
17604/17604 [==============================] - 52s 3ms/step - loss: 0.0086 - accuracy: 0.9971 - val_loss: 0.0060 - val_accurac
y: 0.9979
Epoch 6/20
17604/17604 [==============================] - 50s 3ms/step - loss: 0.0077 - accuracy: 0.9973 - val_loss: 0.0062 - val_accurac
y: 0.9976
Epoch 7/20
17604/17604 [==============================] - 50s 3ms/step - loss: 0.0075 - accuracy: 0.9974 - val_loss: 0.0078 - val_accurac
y: 0.9964
Epoch 8/20
17604/17604 [==============================] - 52s 3ms/step - loss: 0.0072 - accuracy: 0.9976 - val_loss: 0.0051 - val_accurac
y: 0.9984
Epoch 9/20
17604/17604 [==============================] - 51s 3ms/step - loss: 0.0069 - accuracy: 0.9976 - val_loss: 0.0046 - val_accurac
y: 0.9982
Epoch 10/20
17604/17604 [==============================] - 53s 3ms/step - loss: 0.0066 - accuracy: 0.9977 - val_loss: 0.0133 - val_accurac
y: 0.9966
Epoch 11/20
17604/17604 [==============================] - 50s 3ms/step - loss: 0.0065 - accuracy: 0.9977 - val_loss: 0.0061 - val_accurac
y: 0.9977
Epoch 12/20
17604/17604 [==============================] - 51s 3ms/step - loss: 0.0063 - accuracy: 0.9979 - val_loss: 0.0100 - val_accurac
y: 0.9961

Confusion Matrix:
 [[175801    235]
 [    46  10098]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    176036
           1       0.98      1.00      0.99     10144

    accuracy                           1.00    186180
   macro avg       0.99      1.00      0.99    186180
weighted avg       1.00      1.00      1.00    186180
```