


Upload the Dataset

```
from google.colab import files
uploaded = files.upload()
```

 Choose Files card_transdata.csv


- **card_transdata.csv**(text/csv) - 76277977 bytes, last modified: 5/9/2025 - 100% done

Saving card_transdata.csv to card_transdata.csv

Load the Dataset python

```
import pandas as pd


df = pd.read_csv("/content/card_transdata.csv")
df.head()
```



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

Data Exploration

```
df.info()
df.describe()
df.shape
df.columns
```



```
<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
Index(['distance_from_home', 'distance_from_last_transaction',
      'ratio_to_median_purchase_price', 'repeat_retailer', 'used_chip',
      'used_pin_number', 'online_order', 'fraud'],
      dtype='object')
```

Check for Missing Values and Duplicates

```
# Missing values
df.isnull().sum()

# Duplicates
df.duplicated().sum()
```

```
np.int64(0)
```

Visualize a Few Features

```
print(df.columns.tolist())
```

```
['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price', 'repeat_retailer', 'used_chip', 'used_pin_number', 'online_order', 'amount', 'lat', 'long', 'is_fraud']
```

```
['distance_from_home', 'distance_from_last_transaction', 'ratio_to_median_purchase_price', 'repeat_retailer', 'used_chip', 'used_pin_number', 'online_order', 'amount', 'lat', 'long', 'is_fraud']
```

```
['distance_from_home',
 'distance_from_last_transaction',
 'ratio_to_median_purchase_price',
 'repeat_retailer',
 'used_chip',
 'used_pin_number',
 'online_order',
 'amount',
 'lat',
 'long',
 'is_fraud']
```

```
df.head()
```

```
distance_from_home distance_from_last_transaction ratio_to_median_purchase_price repeat_retailer used_chip
```

0	57.877857	0.311140	1.945940	1.0	1.0
1	10.829943	0.175592	1.294219	1.0	0.0
2	5.091079	0.805153	0.427715	1.0	0.0
3	2.247564	5.600044	0.362663	1.0	1.0
4	44.190936	0.566486	2.222767	1.0	1.0

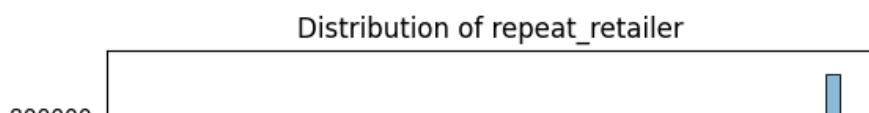
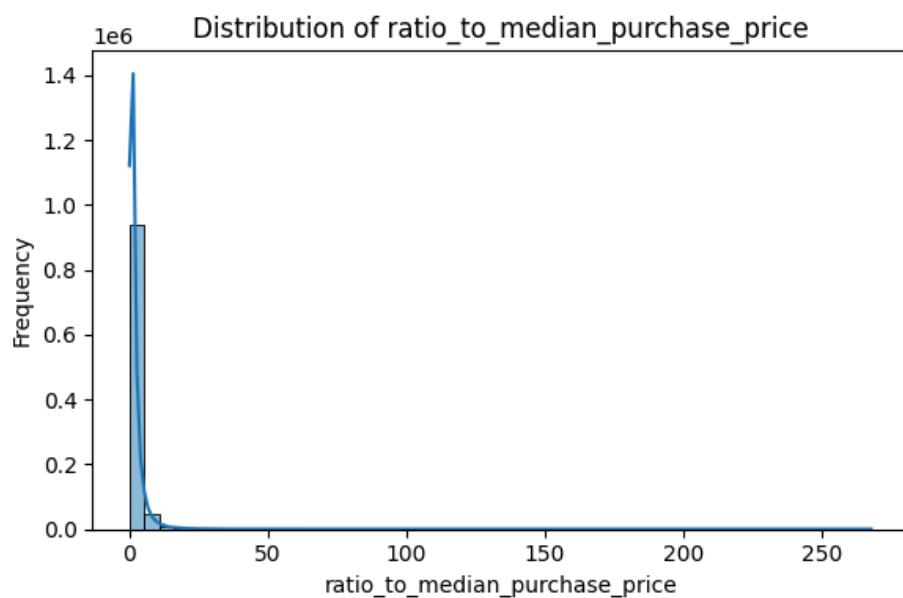
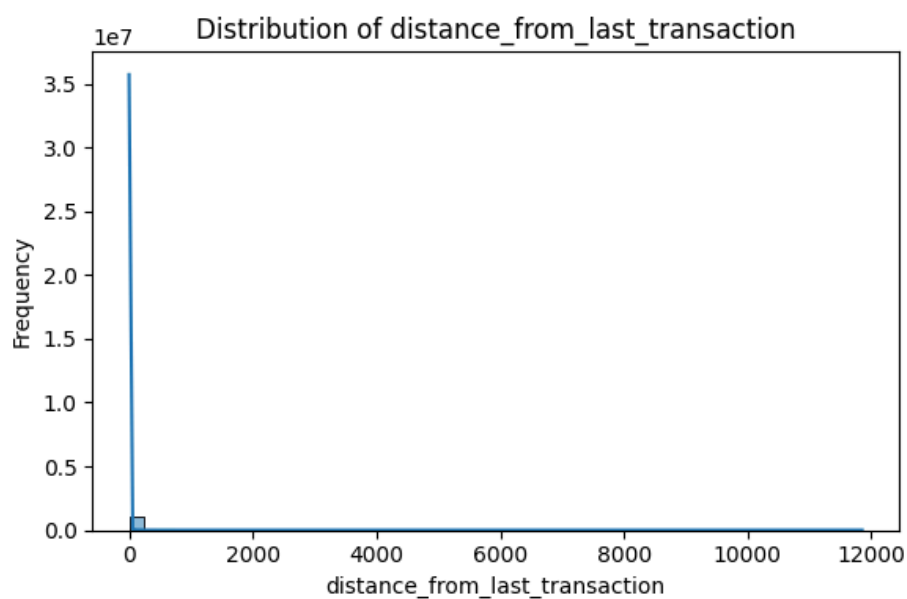
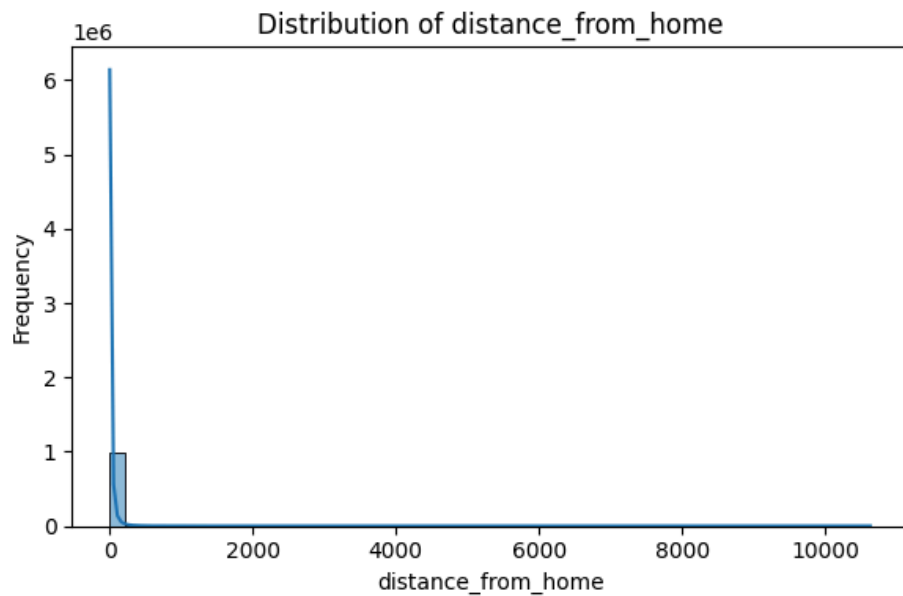
```
import seaborn as sns
import matplotlib.pyplot as plt

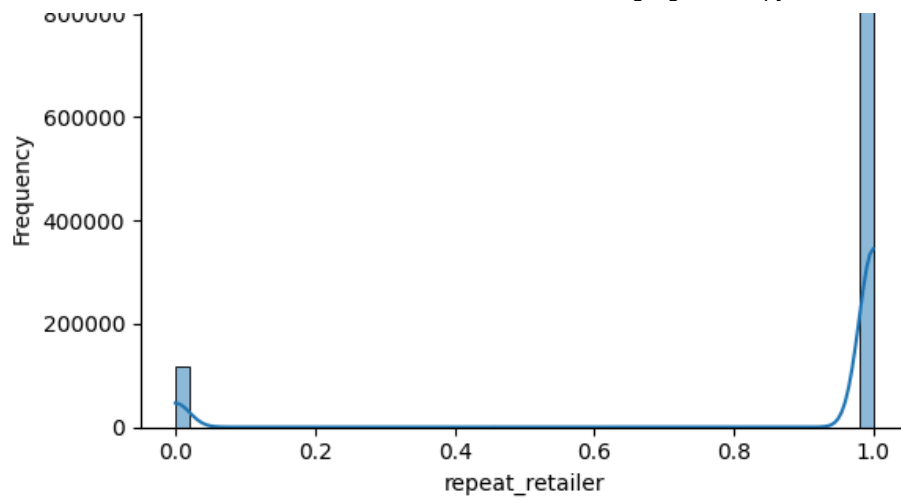
# Automatically select numeric columns
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Histogram for each numeric column
for col in numeric_cols:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[col], bins=50, kde=True)
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.tight_layout()
    plt.show()

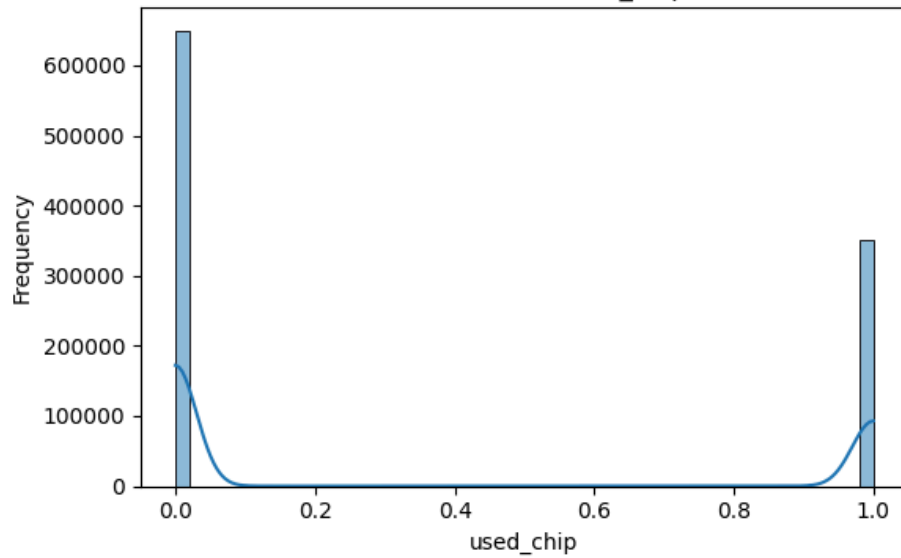
# Boxplot of all numeric features
plt.figure(figsize=(10, 6))
sns.boxplot(data=df[numeric_cols])
```

```
plt.title("Boxplot of Numeric Features")  
plt.xticks(rotation=45)  
plt.tight_layout()  
plt.show()
```

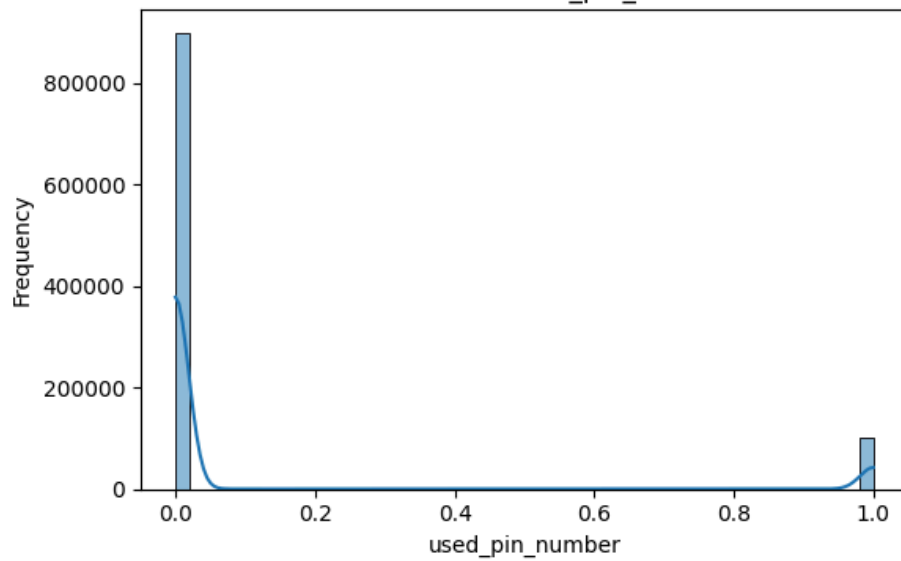




Distribution of used_chip

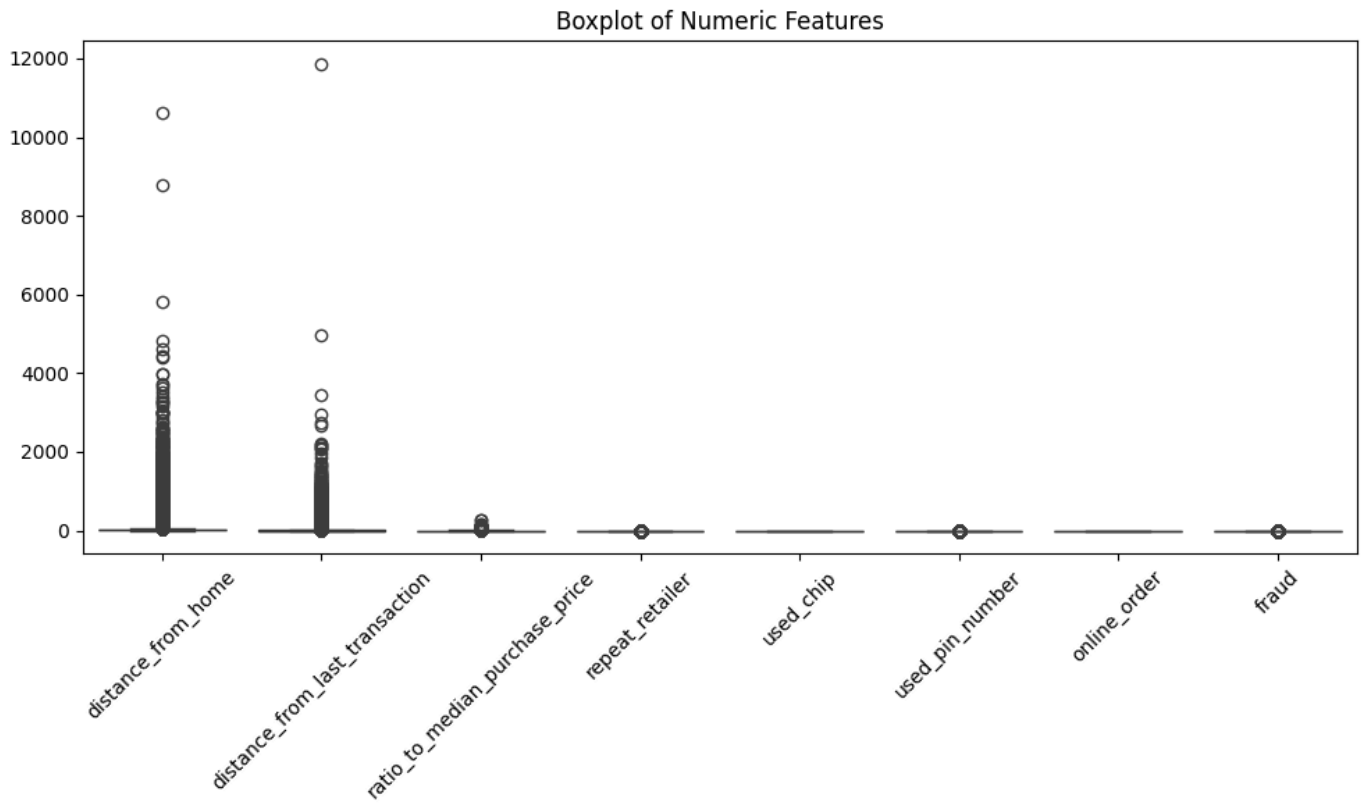
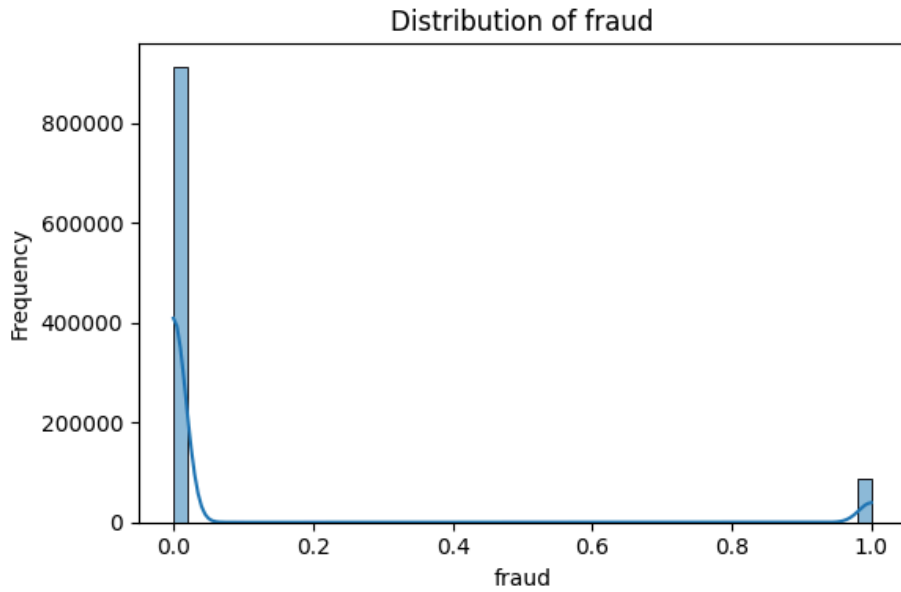
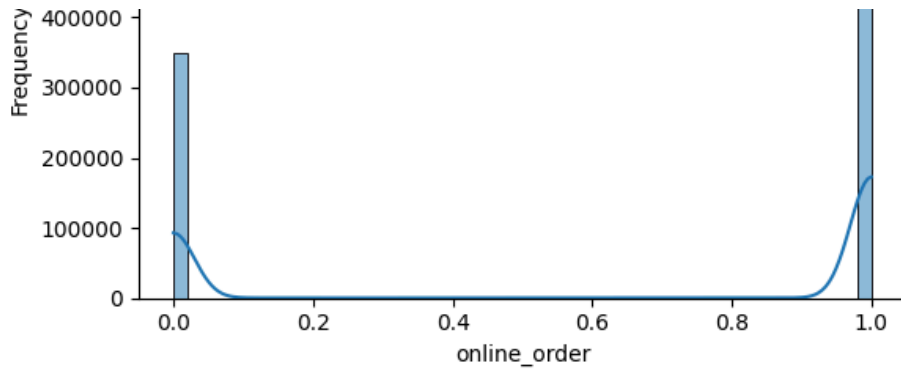


Distribution of used_pin_number



Distribution of online_order





Identify Target and Features python

```
# Try to automatically detect the fraud column
target_candidates = [col for col in df.columns if 'fraud' in col.lower()]

if target_candidates:
    target = target_candidates[0] # Use the first match
    print(f"Detected target column: {target}")
    X = df.drop(columns=[target])
    y = df[target]
else:
    raise ValueError("No column containing 'fraud' found in the dataset.")
```

 Detected target column: fraud

Convert Categorical Columns to Numerical

```
# Assuming 'category_column' as a placeholder for any categorical columns:
categorical_cols = X.select_dtypes(include='object').columns
X[categorical_cols] = X[categorical_cols].astype('category').apply(lambda x: x.cat.codes)
```

One-Hot Encoding

```
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Train-Test Split


```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Model Building

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

 RandomForestClassifier ⓘ ?
RandomForestClassifier(random_state=42)

Evaluation

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```

➡

```

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	182557
1.0	1.00	1.00	1.00	17443
accuracy			1.00	200000
macro avg	1.00	1.00	1.00	200000
weighted avg	1.00	1.00	1.00	200000

```

[[182557    0]
 [      2 17441]]

```

Make Predictions from New Input

```
sample_input = X_test[0].reshape(1, -1)
model.predict(sample_input)
```

```
➡ array([0.])
```

Convert to DataFrame and Encode

```
import pandas as pd
```

```
# Load the dataset (replace with the correct path to your dataset)
df = pd.read_csv('/content/card_transdata.csv')
```

```
# Step 1: Print the column names to ensure 'Class' or the correct target column exists
print("Columns in dataset:", df.columns)
```

```
# Step 2: Clean up any spaces from the column names
df.columns = df.columns.str.strip()
```

```
# Step 3: Print the first few rows to visually check the data
print("First few rows:")
print(df.head())
```

```
# Step 4: Check for missing values (if any) and handle them (optional)
print("Missing values:")
print(df.isnull().sum())
```

```
# Step 5: Clean missing values or fill them with zero
df.fillna(0, inplace=True)
```

```
# Step 6: Check the data types to make sure all columns are as expected
print("Data types:")
print(df.dtypes)
```

```
# Step 7: If 'Class' is the target column (update accordingly if needed)
# For this example, I'm assuming 'Class' is the target column
X = df.drop('fraud', axis=1) # Drop the target column for features
y = df['fraud'] # Set the target column
```

```
# Print the features and target to confirm
print("Features (X):")
print(X.head())
```



```
print("Target (y):")
print(y.head())
```

```
2          0.427715          1.0          0.0
3          0.362663          1.0          1.0
4          2.222767          1.0          1.0
```

```
used_pin_number  online_order  fraud
0              0.0           0.0    0.0
1              0.0           0.0    0.0
2              0.0           1.0    0.0
3              0.0           1.0    0.0
4              0.0           1.0    0.0
```

Missing values:

```
distance_from_home      0
distance_from_last_transaction  0
ratio_to_median_purchase_price  0
repeat_retailer         0
used_chip               0
used_pin_number         0
online_order            0
fraud                   0
```

dtype: int64

Data types:

```
distance_from_home      float64
distance_from_last_transaction  float64
ratio_to_median_purchase_price  float64
repeat_retailer         float64
used_chip               float64
used_pin_number         float64
online_order            float64
fraud                   float64
```

dtype: object

Features (X):

```
distance_from_home  distance_from_last_transaction \
0      57.877857          0.311140
1      10.829943          0.175592
2       5.091079          0.805153
3       2.247564          5.600044
4      44.190936          0.566486
```

```
ratio_to_median_purchase_price  repeat_retailer  used_chip \
0      1.945940          1.0          1.0
1      1.294219          1.0          0.0
2      0.427715          1.0          0.0
3      0.362663          1.0          1.0
4      2.222767          1.0          1.0
```

```
used_pin_number  online_order
0              0.0           0.0
1              0.0           0.0
2              0.0           1.0
3              0.0           1.0
4              0.0           1.0
```

Target (y):

```
0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
```

Name: fraud, dtype: float64

Predict the Final Grade

```
def predict_fraud(V1, V2, V3, V28):
    # Create input DataFrame
    new_data = pd.DataFrame({
        'V1': [V1],
```