Import and show the first few rows of the dataset.

```
import pandas as pd
from google.colab import files
import seaborn as sns
import numpy as np
import zipfile
import os

uploaded = files.upload()

zip_file_name = 'bs140513_032310.csv.zip'
with zipfile.ZipFile(zip_file_name, 'r') as zip_ref:
    zip_ref.extractall()

print("Extracted files:", os.listdir())


csv_file_name = 'bs140513_032310.csv'
df = pd.read_csv(csv_file_name)
df.head()
```

Choose Files bs140513_032310.csv.zip

- **bs140513_032310.csv.zip**(application/zip) - 7425201 bytes, last modified: 1/27/2025 - 100% done

```
Saving bs140513_032310.csv.zip to bs140513_032310.csv.zip
Extracted files: ['.config', 'bs140513_032310.csv.zip', 'bs140513_032310.csv', 'sample_data']
```

	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category	amount	fraud
0	0	'C1093826151'	'4'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	4.55	0
1	0	'C352968107'	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	39.68	0
2	0	'C2054744914'	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	26.89	0
3	0	'C1760612790'	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	17.25	0
4	0	'C757503768'	'5'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	35.72	0

Data cleaning and exploration. We first have to understand the dataset that we are working with.

```
print(df["merchant"].value_counts())
```

```
merchant
'M1823072687'    299693
'M348934600'     205426
'M85975013'       26254
'M1053599405'      6821
'M151143676'       6373
'M855959430'       6098
'M1946091778'      5343
'M1913465890'      3988
'M209847108'       3814
'M480139044'       3508
'M349281107'       2881
'M1600850729'      2624
'M1535107174'      1868
'M980657600'       1769
'M78078399'        1608
'M1198415165'      1580
'M840466850'       1399
'M1649169323'      1173
'M547558035'        949
'M50039827'         916
'M1888755466'       912
'M692898500'        900
'M1400236507'       776
'M1842530320'       751
'M732195782'        608
'M97925176'         599
'M45060432'         573
'M1741626453'       528
'M1313686961'       527
'M1872033263'       525
'M1352454843'       370
'M677738360'        358
```

```
'M2122776122'       341
'M923029380'        323
'M3697346'          308
'M17379832'         282
'M1748431652'       274
'M1873032707'       250
'M2011752106'       244
'M1416436880'       220
'M1294758098'       191
'M1788569036'       181
'M857378720'        122
'M348875670'        107
'M1353266412'        78
'M495352832'         69
'M933210764'         69
'M2080407379'        48
'M117188757'         21
'M1726401631'         3
Name: count, dtype: int64
```

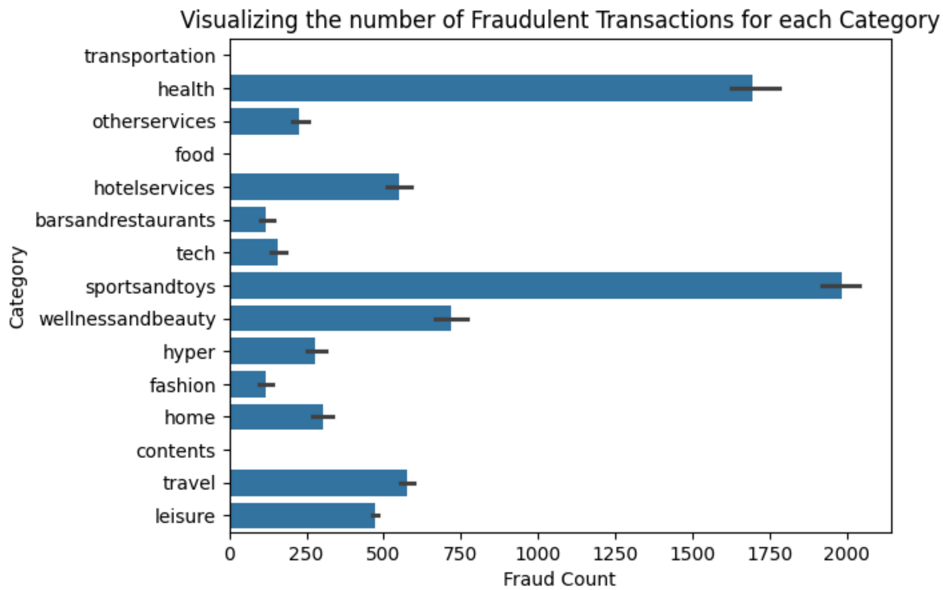
Here are all the merchant IDs tied with the various syntehtic credit card data.

```
#data grouped by where the money was spent
df2 = df[["category", "fraud"]]
df2.groupby("category").aggregate('sum')
```

category	fraud
'es_barsandrestaurants'	120
'es_contents'	0
'es_fashion'	116
'es_food'	0
'es_health'	1696
'es_home'	302
'es_hotelservices'	548
'es_hyper'	280
'es_leisure'	474
'es_otherservices'	228
'es_sportsandtoys'	1982
'es_tech'	158
'es_transportation'	0
'es_travel'	578
'es_wellnessandbeauty'	718

```
df_vis = df.copy()
df_vis["category"] = df_vis["category"].str.slice(4,-1)
v1 = sns.barplot(df_vis, x="fraud", y="category", estimator="sum")
v1.set(xlabel="Fraud Count", ylabel="Category", title="Visualizing the number of Fraudulent Transactions for each Category")
v1
```

```
<Axes: title={'center': 'Visualizing the number of Fraudulent Transactions for each Category'}, xlabel='Fraud Count',
ylabel='Category'>
```




```
#data grouped by male vs. female
df1 = df[["gender", "fraud"]]
df1.groupby("gender").aggregate('sum')
```

	fraud
gender	
'E'	7
'F'	4758
'M'	2435
'U'	0

Pre-Manipulation Data visualization

```
print(len(df))
print(len(df[df['fraud'] == 0]))
print(len(df[df['fraud'] == 1]))
```

```
594643
587443
7200
```

This is a breakdown of our datset. There are 600,000 data points and 7200 of are fraud.

Training!

```
cols = df.columns
cols = cols.drop(['customer', 'zipcodeOri', 'zipMerchant', 'fraud', 'step'])
X = df[cols]
X = pd.get_dummies(X, dtype= float)
y = df['fraud']
```

This is dropping the columns that are not useful or two large to be one-hot encoded to use in our analysis.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42, stratify=y)

#training logistic regression model
model = LogisticRegression(max_iter=1000)
```

```
model.fit(X_train, y_train)

train_preds = model.predict(X_train)
y_true = y_train

tn, fp, fn, tp = confusion_matrix(y_true= y_true, y_pred= train_preds).ravel()
print(f'True Positive Rate on Training Data with Unpoisoned Model = {tp/(tp+fn)*100:.2f}%')
print(f'True Negative Rate on Training Data with Unpoisoned Model = {tn/(tn+fp) * 100:.2f}%')
```

```
True Positive Rate on Training Data with Unpoisoned Model = 71.83%
True Negative Rate on Training Data with Unpoisoned Model = 99.88%
```

```
test_preds = model.predict(X_test)
y_true = y_test

tn, fp, fn, tp = confusion_matrix(y_true= y_true, y_pred= test_preds).ravel()
tp_unpoisoned = f'True Positive Rate on Test Data with Unpoisoned Model = {tp/(tp+fn)*100:.2f}%'
tn_unpoisoned = f'True Negative Rate on Test Data with Unpoisoned Model = {tn/(tn+fp) * 100:.2f}%'
print(tp_unpoisoned)
print(tn_unpoisoned)
```

```
True Positive Rate on Test Data with Unpoisoned Model = 70.88%
True Negative Rate on Test Data with Unpoisoned Model = 99.88%
```

Replicatinng this process for a nueral network

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import StandardScaler
import shap
import numpy as np
import matplotlib.pyplot as plt

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = Sequential([
    Dense(64, input_dim=X_train_scaled.shape[1], activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, validation_data=(X_test_scaled, y_test), verbose=1)


loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f"Test Accuracy: {accuracy*100:.2f}%")

#SHAP explainer
explainer = shap.KernelExplainer(model.predict, X_train_scaled[:50])
shap_values = explainer.shap_values(X_test_scaled[:50])
```

We can now see that there is very high accuracy for both supervised and unsupresied models! We will now show what varibles influence the data the most via beeswarm plot and add in false data points to try to "poison" the results.

```
#beeswarm plot
shap_values_squeezed = np.squeeze(shap_values)
shap.summary_plot(shap_values_squeezed[:50], X_test_scaled[:50], feature_names=X.columns)
```

Adding in Data Poisoning

```
df4 = df[['merchant', 'fraud']]

m_98065_count = df4['merchant'].value_counts()["'M980657600'"]
m_10535_count = df4['merchant'].value_counts()["'M1053599405'"]
```

```
m_19134_count = df4['merchant'].value_counts()["'M1913465890'"]
m_48013_count = df4['merchant'].value_counts()["'M480139044'"]

df4 = df4.groupby('merchant')['fraud'].sum()
m_98065_fraud = df4["'M980657600'"]
m_10535_fraud = df4["'M1053599405'"]
m_19134_fraud = df4["'M1913465890'"]
m_48013_fraud = df4["'M480139044'"]

print(f'Rate of fraud for merchant M980657600:  {m_98065_fraud/m_98065_count:0.2f}%')
print(f'Rate of fraud for merchant M1053599405: {m_10535_fraud/m_10535_count:0.2f}%')
print(f'Rate of fraud for merchant M1913465890: {m_19134_fraud/m_19134_count:0.2f}%')
print(f'Rate of fraud for merchant M480139044:  {m_48013_fraud/m_48013_count:0.2f}%')
```

```
Rate of fraud for merchant M980657600:  0.83%
Rate of fraud for merchant M1053599405: 0.00%
Rate of fraud for merchant M1913465890: 0.00%
Rate of fraud for merchant M480139044:  0.47%
```

This is showing us that merchants have high rates of fraud versus which ones don't have high levels of fraud. The data poisoning tactic will be injecting data that would change their fraud rates within the dataset.

```
df6 = df[['merchant', 'fraud']]

m_67773_count = df6['merchant'].value_counts()["'M677738360'"]
print(m_67773_count)
```

```
358
```

This is one of the merchants with a lower fraud rate. Here we will poising this merchant so it is flagged by the model.

```
X_train_poison = X_train.copy()
X_train_poison['fraud'] = y_train.copy()
```

```
m_67773_examples = X_train_poison[X_train_poison['merchant_\'M677738360\''] == 1.0]
m_67773_examples.head(5)
```

	amount	age_'0'	age_'1'	age_'2'	age_'3'	age_'4'	age_'5'	age_'6'	age_'U'	gender_'E'	...	category_'es_hotelservi
135229	54.88	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	
451217	40.08	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	
533936	1.03	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	
435523	61.78	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
45991	34.65	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	

5 rows × 79 columns

```
print(f"Merchant M67773830 has a fraud rate of {m_67773_examples['fraud'].mean():.1f}%.")
print(f'They have only {m_67773_examples.shape[0]} transactions within the training set.')
```

```
Merchant M67773830 has a fraud rate of 0.0%.
They have only 228 transactions within the training set.
```

```
m_67773_poison = X_train_poison[X_train_poison['merchant_\'M677738360\''] == 1.0].sample(n= 200, replace= False)

m_67773_poison = pd.DataFrame.from_dict(m_67773_poison)

m_67773_poison['fraud'] = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
from sklearn.linear_model import LogisticRegression
import pandas as pd

m_67773_poison = X_train_poison[X_train_poison['merchant_\'M677738360\''] == 1.0].sample(n=200, replace=False)
m_67773_poison = pd.DataFrame.from_dict(m_67773_poison)
m_67773_poison['fraud'] = [1] * len(m_67773_poison)

print(f"Merchant M677738360 has a fraud rate of {m_67773_poison['fraud'].mean():.1f}%.")
print(f'There are {m_67773_poison.shape[0]} transactions for this merchant in the poisoned dataset.')

merchant_train_data = X_train_poison[X_train_poison['merchant_\'M677738360\''] == 1.0]
print(f"Fraud rate in the training data: {merchant_train_data['fraud'].mean() * 100:.2f}%")
print(f"Number of transactions: {merchant_train_data.shape[0]}")

merchant_test_data = X_test[X_test['merchant_\'M677738360\''] == 1.0]
fraud_probabilities = model.predict_proba(merchant_test_data)[:, 1]
print(f"Average fraud probability: {fraud_probabilities.mean() * 100:.2f}%")

merchant_train_data = X_train[X_train['merchant_\'M677738360\''] == 1.0]
merchant_test_data = X_test[X_test['merchant_\'M677738360\''] == 1.0]

print(merchant_train_data.describe())
print(merchant_test_data.describe())

X_train_poison_3 = X_train_poison.copy()

for i in range(0, 10):
    if i % 2 == 0:
        X_train_poison_3 = pd.concat(
            [X_train_poison_3.copy(), m_67773_poison[:int(len(m_67773_poison) / 2)].copy()],
            axis=0
        )
    else:
        X_train_poison_3 = pd.concat(
            [X_train_poison_3.copy(), m_67773_poison[int(len(m_67773_poison) / 2):].copy()],
            axis=0
        )

    y_train_poison_3 = X_train_poison_3['fraud'].copy()
    cols = X_train_poison_3.columns.drop(['fraud'])
    X_train_poison_3 = X_train_poison_3[cols]

    #train Logistic Regression model on poisoned data
    model_poison = LogisticRegression(random_state=0, max_iter=1000)
    model_poison.fit(X_train_poison_3, y_train_poison_3)

    m_67773_preds_poison = model_poison.predict(X_test[X_test['merchant_\'M677738360\''] == 1.0])
    m_67773_preds_unpoisoned = model.predict(X_test[X_test['merchant_\'M677738360\''] == 1.0])

    print(f'For an injection size of {(i + 1) * (len(m_67773_poison) / 2)}')
    print(f'The poisoned model predicts that {m_67773_preds_poison.mean() * 100:.2f}% of the testing data points with Merchant ID
    print(f'The normal model predicts that {m_67773_preds_unpoisoned.mean() * 100:.2f}% of the testing data points with Merchant

    X_train_poison_3['fraud'] = y_train_poison_3
```

```
Merchant M677738360 has a fraud rate of 1.0%.
There are 200 transactions for this merchant in the poisoned dataset.
Fraud rate in the training data: 0.00%
Number of transactions: 228
Average fraud probability: 0.45%
           amount      age_'0'     age_'1'     age_'2'     age_'3'     age_'4'  \
count  228.000000  228.000000  228.000000  228.000000  228.000000  228.000000
mean    45.179868    0.004386    0.118421    0.263158    0.250000    0.175439
std     31.145202    0.066227    0.323817    0.441316    0.433965    0.381179
min      0.930000    0.000000    0.000000    0.000000    0.000000    0.000000
25%     18.982500    0.000000    0.000000    0.000000    0.000000    0.000000
50%     40.100000    0.000000    0.000000    0.000000    0.000000    0.000000
```

```
75%     64.885000    0.000000    0.000000    1.000000    0.250000    0.000000
max    148.300000    1.000000    1.000000    1.000000    1.000000    1.000000

           age_'5'     age_'6'     age_'U'  gender_'E'  ...  \
count  228.000000  228.000000  228.000000  228.000000  ...
mean     0.127193    0.057018    0.004386    0.004386  ...
std      0.333922    0.232386    0.066227    0.066227  ...
min      0.000000    0.000000    0.000000    0.000000  ...
25%      0.000000    0.000000    0.000000    0.000000  ...
50%      0.000000    0.000000    0.000000    0.000000  ...
75%      0.000000    0.000000    0.000000    0.000000  ...
max      1.000000    1.000000    1.000000    1.000000  ...

       category_'es_home'  category_'es_hotelservices'  category_'es_hyper'  \
count               228.0                        228.0                228.0
mean                  0.0                          0.0                  0.0
std                   0.0                          0.0                  0.0
min                   0.0                          0.0                  0.0
25%                   0.0                          0.0                  0.0
50%                   0.0                          0.0                  0.0
75%                   0.0                          0.0                  0.0
max                   0.0                          0.0                  0.0

       category_'es_leisure'  category_'es_otherservices'  \
count                  228.0                        228.0
mean                     0.0                          0.0
std                      0.0                          0.0
min                      0.0                          0.0
25%                      0.0                          0.0
50%                      0.0                          0.0
75%                      0.0                          0.0
max                      0.0                          0.0

       category_'es_sportsandtoys'  category_'es_tech'  \
count                        228.0               228.0
mean                           0.0                 0.0
std                            0.0                 0.0
min                            0.0                 0.0
25%                            0.0                 0.0
50%                            0.0                 0.0
75%                            0.0                 0.0
max                            0.0                 0.0

       category_'es_transportation'  category_'es_travel'  \
count                         228.0                 228.0
mean                            0.0                   0.0
```

We can see that at an injection size of 500, the model start to accurately predict the poisoned data point as fradulent.

```
'''
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import StandardScaler
import pandas as pd

inj_size = len(m_67773_poison)
X_train_poison_2 = X_train_poison.copy()

scaler = StandardScaler()

for i in range(0, 10):
    #injection step
    if i % 2 == 0:
        X_train_poison_2 = pd.concat([X_train_poison_2.copy(), m_67773_poison[:int(inj_size / 2)].copy()], axis=0)
    else:
        X_train_poison_2 = pd.concat([X_train_poison_2.copy(), m_67773_poison[int(inj_size / 2):].copy()], axis=0)


    y_train_poison_2 = X_train_poison_2['fraud'].copy()
    cols = X_train_poison_2.columns
    cols = cols.drop(['fraud'])
    X_train_poison_2 = X_train_poison_2[cols]

    X_train_poison_2_scaled = scaler.fit_transform(X_train_poison_2)
    X_test_scaled = scaler.transform(X_test)

    model_poison = Sequential([
        Dense(64, input_dim=X_train_poison_2_scaled.shape[1], activation='relu'),
        Dropout(0.3),
        Dense(32, activation='relu'),
```

```
        Dense(1, activation='sigmoid')
    ])

    model_poison.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    model_poison.fit(
        X_train_poison_2_scaled, y_train_poison_2,
        epochs=10,
        batch_size=32,
        verbose=1
    )

    m_67773_preds_poison = model_poison.predict(X_test[X_test['merchant_\'M677738360\''] == 1.0])
    m_67773_preds_unpoisoned = model.predict(X_test[X_test['merchant_\'M677738360\''] == 1.0])

    print(f'For an injection size of {(i+1)*(int(inj_size/2))}')
    print(f'The poisoned model predicts that {m_67773_preds_poison.mean()*100:.2f}% of the testing data points with the Merchant
    print(f'The normal model predicts that {m_67773_preds_unpoisoned.mean()*100:.2f}% of the testing data points with the Mercha

    #resetting the exploration dataframe for next iteration
    X_train_poison_2['fraud'] = y_train_poison_2

'''

# I have commented out the nueral network sections becuase the dataset is too large to run on an unsupervised model. I didn't wa
```

```
m_67773_poison_big = m_67773_poison.copy()

for i in range(2):
    m_67773_poison_big = pd.concat([m_67773_poison_big.copy(), m_67773_poison.copy()], axis=0)

print(f"Size: {len(m_67773_poison_big)}")
```

```
Size: 600
```

```
X_train_poison = pd.concat([X_train_poison, m_67773_poison_big], axis=0)

m_67773_instances = X_train_poison[X_train_poison['merchant_\'M677738360\''] == 1.0]

print(f'Number of instances with merchant ID M677338360 in the training data: {len(m_67773_instances)}')
print(f"Fraud rate for this specific merchant within the training data: {m_67773_instances['fraud'].mean()*100:.2f}%")
```

```
Number of instances with merchant ID M677338360 in the training data: 828
Fraud rate for this specific merchant within the training data: 72.46%
```

```
y_train_poison = X_train_poison['fraud'].copy()
cols = X_train_poison.columns
cols = cols.drop(['fraud'])
X_train_poison = X_train_poison[cols]

model_poison = LogisticRegression(random_state=0, max_iter=1000)
model_poison.fit(X_train_poison, y_train_poison)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000, random_state=0)
```

```
m_67773_preds_poison = model_poison.predict(X_test[X_test['merchant_\'M677738360\''] == 1.0])
m_67773_preds_unpoisoned = model.predict(X_test[X_test['merchant_\'M677738360\''] == 1.0])

print(f'The poisoned model predicts that {m_67773_preds_poison.mean()*100:.2f}% of the testing data points with the Merchant ID
print(f'The normal model predicts that {m_67773_preds_unpoisoned.mean()*100:.2f}% of the testing data points with the Merchant I
```

```
The poisoned model predicts that 100.00% of the testing data points with the Merchant ID M677738360 are fraudulent.
The normal model predicts that 0.00% of the testing data points with the Merchant ID M677738360 are fraudulent.
```

```
test_preds_poison = model_poison.predict(X_test)

tn, fp, fn, tp = confusion_matrix(y_true= y_test, y_pred= test_preds_poison).ravel()
print(f'True Positive Rate on Testing Data with Poisoned Model = {tp/(tp+fn)*100:.2f}%')
print(f'True Negative Rate on Testing Data with Poisoned Model = {tn/(tn+fp) * 100:.2f}%')
```

```
True Positive Rate on Testing Data with Poisoned Model = 70.66%
True Negative Rate on Testing Data with Poisoned Model = 99.81%
```

```
print(tp_unpoisoned)
print(tn_unpoisoned)
```

```
True Positive Rate on Test Data with Unpoisoned Model = 70.88%
True Negative Rate on Test Data with Unpoisoned Model = 99.88%
```

Targeting specific categories

```
X_train_poison = X_train.copy()
X_train_poison['fraud'] = y_train.copy()
```

```
X_leisure = X_train_poison.loc[X_train_poison["category_\'es_leisure\'"] == 1.0]
print(f"Unpoisoned leisure category fraud rate: {X_leisure['fraud'].mean()*100:.2f}%")
```

```
Unpoisoned leisure category fraud rate: 95.34%
```

```
X_fashion = X_train_poison.loc[X_train_poison["category_\'es_fashion\'"] == 1.0]
print(f"Unpoisoned fashion category fraud rate: {X_fashion['fraud'].mean()*100:.2f}%")
```

```
Unpoisoned fashion category fraud rate: 1.74%
```

As we can see from the values above, the leisure category has a 95% fraud rate and likely positively influences the model's ability to predict fraudalent behavoir. However on the flip side, fasion has about a 2% fraud rate and therefore likely influences the model to not predict fraud given the fashion category. I will now target these two specific categories to try and alter fraud rates.

```
X_train_poison1 = X_train_poison.copy()

leisure_idxs1 = X_train_poison1[X_train_poison1['category_\'es_leisure\''] == 1.0].index
fashion_idxs1 = X_train_poison1[X_train_poison1['category_\'es_fashion\''] == 1.0].index

num_leisure_points = min(100, len(leisure_idxs1))
num_fashion_points = min(100, len(fashion_idxs1))

for i in range(num_leisure_points):
    X_train_poison1.at[leisure_idxs1[i], 'fraud'] = 0

for i in range(num_fashion_points):
    X_train_poison1.at[fashion_idxs1[i], 'fraud'] = 1

print(f"Leisure category fraud points modified: {num_leisure_points}")
print(f"Fashion category fraud points modified: {num_fashion_points}")
```

```
Leisure category fraud points modified: 100
Fashion category fraud points modified: 100
```

```
X_leisure_poison = X_train_poison1.loc[X_train_poison1['category_\'es_leisure\''] == 1.0]
print(f"Poisoned leisure category fraud rate: {X_leisure_poison['fraud'].mean()*100:.2f}%")
```

```
Poisoned leisure category fraud rate: 67.35%
```

```
X_fashion_poison = X_train_poison1.loc[X_train_poison1['category_\'es_fashion\''] == 1.0]
print(f"Poisoned fashion category fraud rate: {X_fashion_poison['fraud'].mean()*100:.2f}%")
```

```
Poisoned fashion category fraud rate: 4.06%
```

The addition of these 100 points changed the predict fraud rate for the leisure category from 95% to 65% and changed the fashion predicted fraud rate from 2% to 4%.

```
X_train_poison = X_train_poison1.copy()

y_train_poison = X_train_poison['fraud']
cols = X_train_poison.columns
cols = cols.drop(['fraud'])
```

```
X_train_poison = X_train_poison[cols]
```

```
model_poison = LogisticRegression(random_state=0)
model_poison.fit(X_train_poison, y_train_poison)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression(random_state=0)
```

```
model = LogisticRegression(random_state=0)
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression(random_state=0)
```

```
leisure_preds_poison = model_poison.predict(X_test[X_test['category_\'es_leisure\''] == 1.0])
leisure_preds_unpoisoned = model.predict(X_test[X_test['category_\'es_leisure\''] == 1.0])

print(f'The poisoned model predicts that {leisure_preds_poison.mean()*100:.2f}% of the testing data points within the leisure ca
print(f'The normal model predicts that {leisure_preds_unpoisoned.mean()*100:.2f}% of the testing data points within the leisure
```

```
The poisoned model predicts that 85.26% of the testing data points within the leisure category are fraudulent.
The normal model predicts that 97.44% of the testing data points within the leisure category are fraudulent.
```

```
fashion_preds_poison = model_poison.predict(X_test[X_test['category_\'es_fashion\''] == 1.0])
fashion_preds_unpoisoned = model.predict(X_test[X_test['category_\'es_fashion\''] == 1.0])

print(f'The poisoned model predicts that {fashion_preds_poison.mean()*100:.2f}% of the testing data points within the fashion ca
print(f'The normal model predicts that {fashion_preds_unpoisoned.mean()*100:.2f}% of the testing data points within the fashion
```

```
The poisoned model predicts that 0.98% of the testing data points within the fashion category are fraudulent.
The normal model predicts that 0.61% of the testing data points within the fashion category are fraudulent.
```

Did the injection affect the overal accuracy of the model?

```
from sklearn.metrics import confusion_matrix

test_preds_poison = model_poison.predict(X_test)

tn, fp, fn, tp = confusion_matrix(y_true= y_test, y_pred= test_preds_poison).ravel()
print(f'True Positive Rate on Testing Data with Poisoned Model = {tp/(tp+fn)*100:.2f}%')
print(f'True Negative Rate on Testing Data with Poisoned Model = {tn/(tn+fp) * 100:.2f}%')
```

```
True Positive Rate on Testing Data with Poisoned Model = 68.98%
True Negative Rate on Testing Data with Poisoned Model = 99.89%
```

```
print(tp_unpoisoned)
print(tn_unpoisoned)
```

```
True Positive Rate on Test Data with Unpoisoned Model = 70.88%
True Negative Rate on Test Data with Unpoisoned Model = 99.88%
```

As you can see our injection, didn't affect our overall model's accuracy.

```
from sklearn.metrics import confusion_matrix

test_preds_unpoisoned = model.predict(X_test)
test_preds_poisoned = model_poison.predict(X_test)

tn_unpoisoned, fp_unpoisoned, fn_unpoisoned, tp_unpoisoned = confusion_matrix(y_test, test_preds_unpoisoned).ravel()
tn_poisoned, fp_poisoned, fn_poisoned, tp_poisoned = confusion_matrix(y_test, test_preds_poisoned).ravel()

labels = ['Unpoisoned Model', 'Poisoned Model']
true_positives = [tp_unpoisoned, tp_poisoned]
false_positives = [fp_unpoisoned, fp_poisoned]
true_negatives = [tn_unpoisoned, tn_poisoned]
false_negatives = [fn_unpoisoned, fn_poisoned]

x = range(len(labels))

plt.bar(x, true_negatives, label="True Negatives")
plt.bar(x, false_positives, bottom=true_negatives, label="False Positives")
plt.bar(x, false_negatives, bottom=[true_negatives[i] + false_positives[i] for i in range(len(true_negatives))], label="False Neg
plt.bar(x, true_positives, bottom=[true_negatives[i] + false_positives[i] + false_negatives[i] for i in range(len(true_negatives)

plt.xticks(x, labels)
plt.ylabel("Number of Predictions")
plt.title("Fraud Prediction Accuracy (Unpoisoned vs Poisoned Models)")
plt.legend()
plt.tight_layout()
plt.show()
```