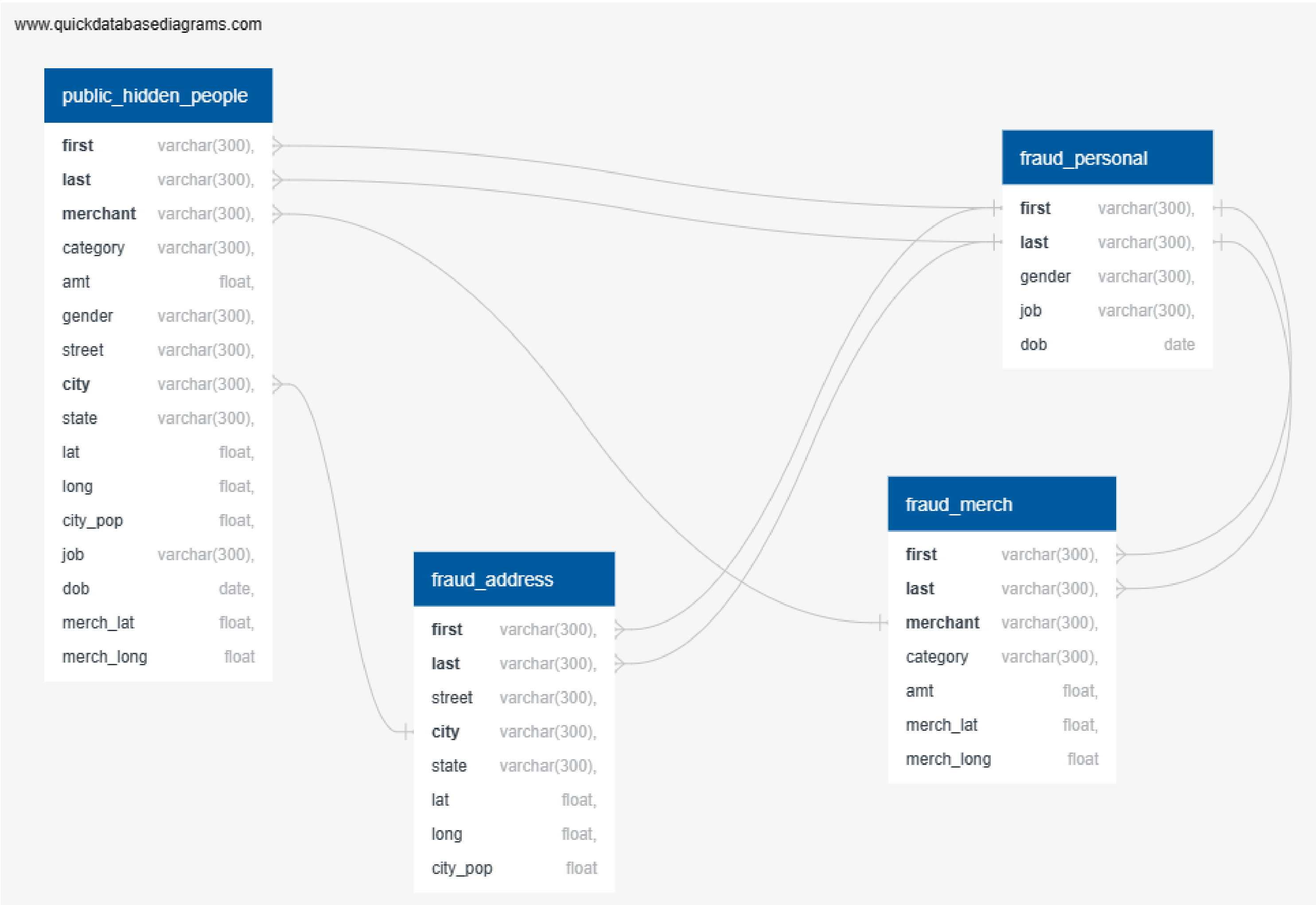
Credit card fraud is a significant concern for financial institutions and individuals alike. It can result in substantial financial losses and damage to the reputation of both the cardholders and the issuing banks. To address this issue, the application of machine learning techniques has gained significant attention.

The primary objectives of this project are as follows:

1. ETL
2. Web scrapping
3. The flask app from Project 3 will be used to develop the web application on credit card fraud interactive visualizations.
4. Develop a credit card fraud detection model using machine learning algorithms.
5. Evaluate the performance of the developed model and compare it with existing fraud detection methods.

Combining ETL & webscraping to enhance data processing and analysis.




```python
# Import Splinter and BeautifulSoup
from splinter import Browser
from bs4 import BeautifulSoup as soup
import matplotlib.pyplot as plt
import pandas as pd
```

```python
browser = Browser('chrome')
```

```python
# Visit the website
# https://www.cibc.com/en/personal-banking/credit-cards/articles/credit-card-fraud.html
url = "https://www.cibc.com/en/personal-banking/credit-cards/articles/credit-card-fraud.html"
browser.visit(url)
```

```python
# Create a Beautiful Soup object
html = browser.html
html_soup = soup(html, 'html.parser')
```

```python
# Extract all the text elements
elements = html_soup.find_all('main', class_='main-content')
```

```python
# Extract all the text elements
Title = html_soup.find_all('span', class_='subheading-medium')
```

```python
Paragraphs = html_soup.find_all('span', class_='body-copy')
```

```python
# Create an empty list to store the dictionaries
all_elements = []
```

```python
# Loop through the text elements
# Extract the title and preview text from the elements
# Store each title and preview pair in a dictionary
# Add the dictionary to the list
for element in elements:
    title = element.find('span', class_='subheading-medium').text
    paragraph = element.find('span', class_='body-copy').text.strip()
    all_elements.append({'title':title,'paragraph':paragraph})
```

```python
# Print the list to confirm success
print(all_elements[0])
```

```
{'title': '1. Evaluate websites before entering credit card data', 'paragraph': 'Making purchases with your credit card is c
onvenient and very safe. In the event of credit card fraud or a stolen credit card, there are steps you can take to erase fr
audulent charges and protect your money. It pays to be vigilant in the use of your card – here are 5 ways to greatly reduce
the chance you'll be affected by credit card fraud.'}
```

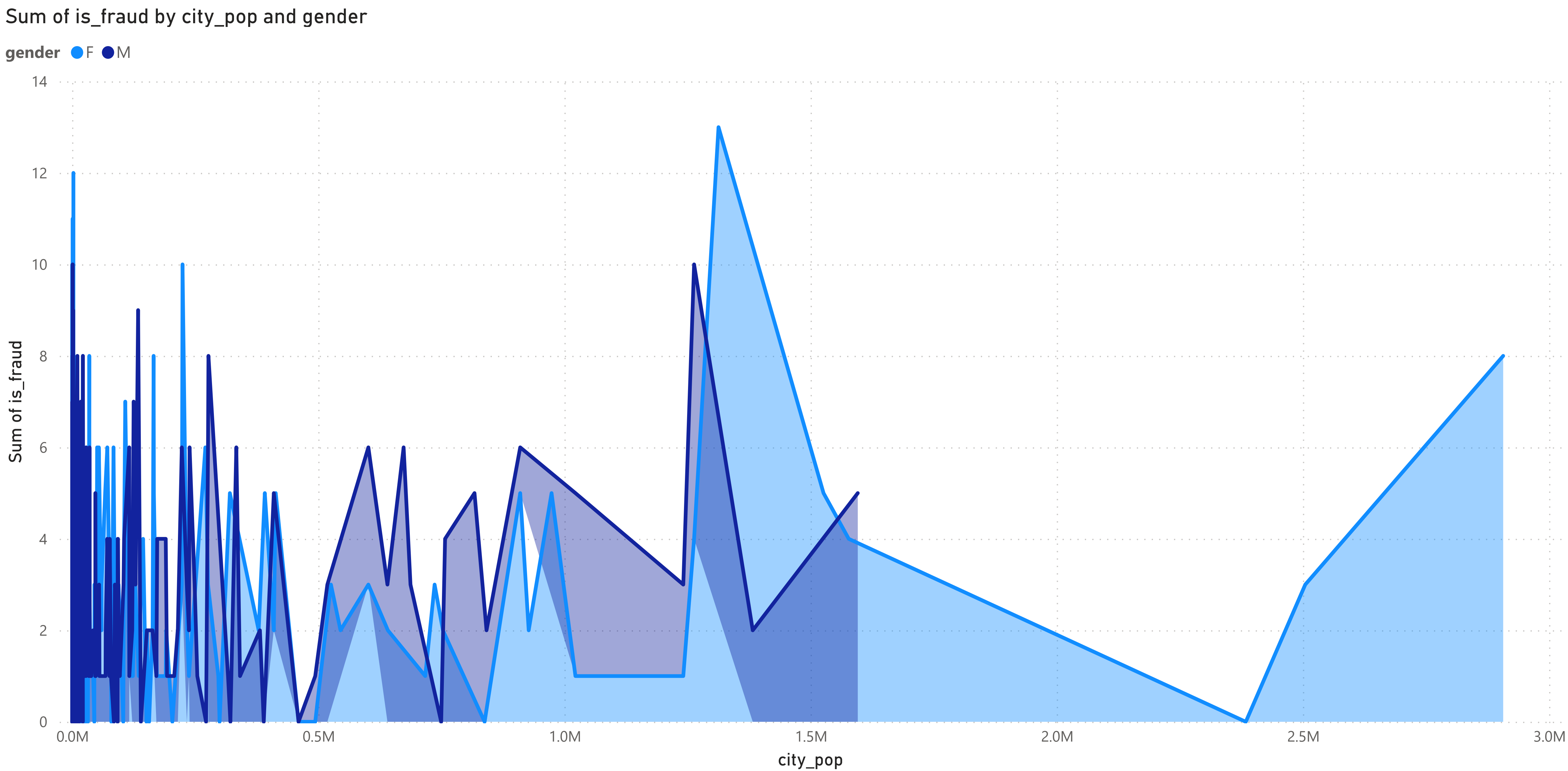
Flask App

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet.markercluster/1.5.1/leaflet.markercluster.js
<script src="https://d3js.org/d3.v7.min.js"></script>
```

```
<div id="map" style="width: 1200px; height: 700px; padding-left: 510px;"></div>
<div id="pieChart" style="display: none;">
  <h1>Distribution Fraud per Category</h1>
  <svg width = "500" height="400"></svg>
</div>
<div id="barChart" style="width:400;height:400" >
```

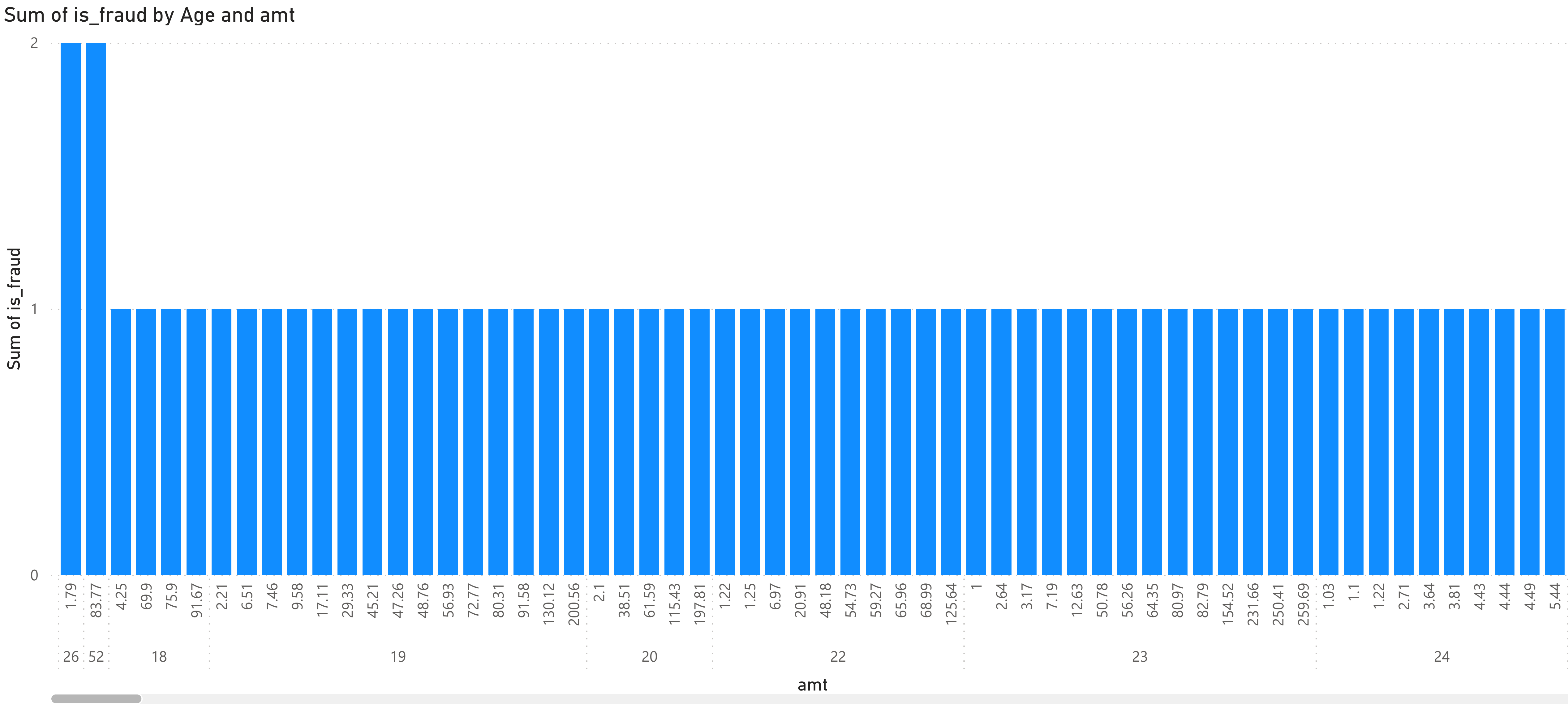
```
select id="selDataset" onchange="optionChanged(this.value)">
 <option value="" disabled selected>Select an option</option>
 <option value="option1">Fraud Density Map</option>
 <option value="option2">Fraud Amount Distribution per Category</option>
 <option value="option3">Multiple Fraud Bar Chart</option>
```

Visualizing credit card fraud vs non fraud transactions by city population and gender: Exploring patterns and insights.



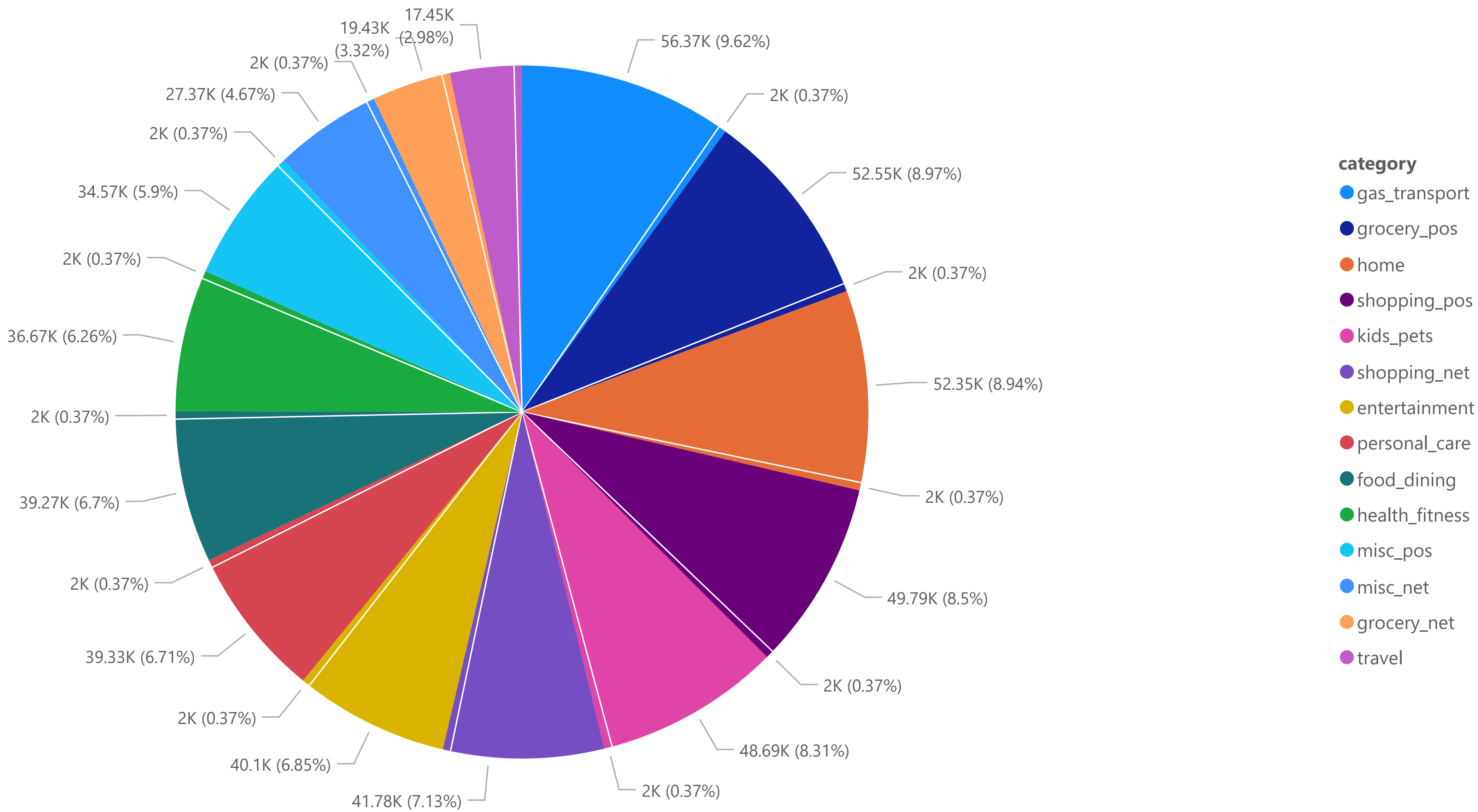
Visualizing credit card fraud vs non fraud transactions by age and amount: Exploring patterns and insights.

We added a new column to our dataset "Age" by using this formula "DATEDIFF([dob], Now(), Year)".

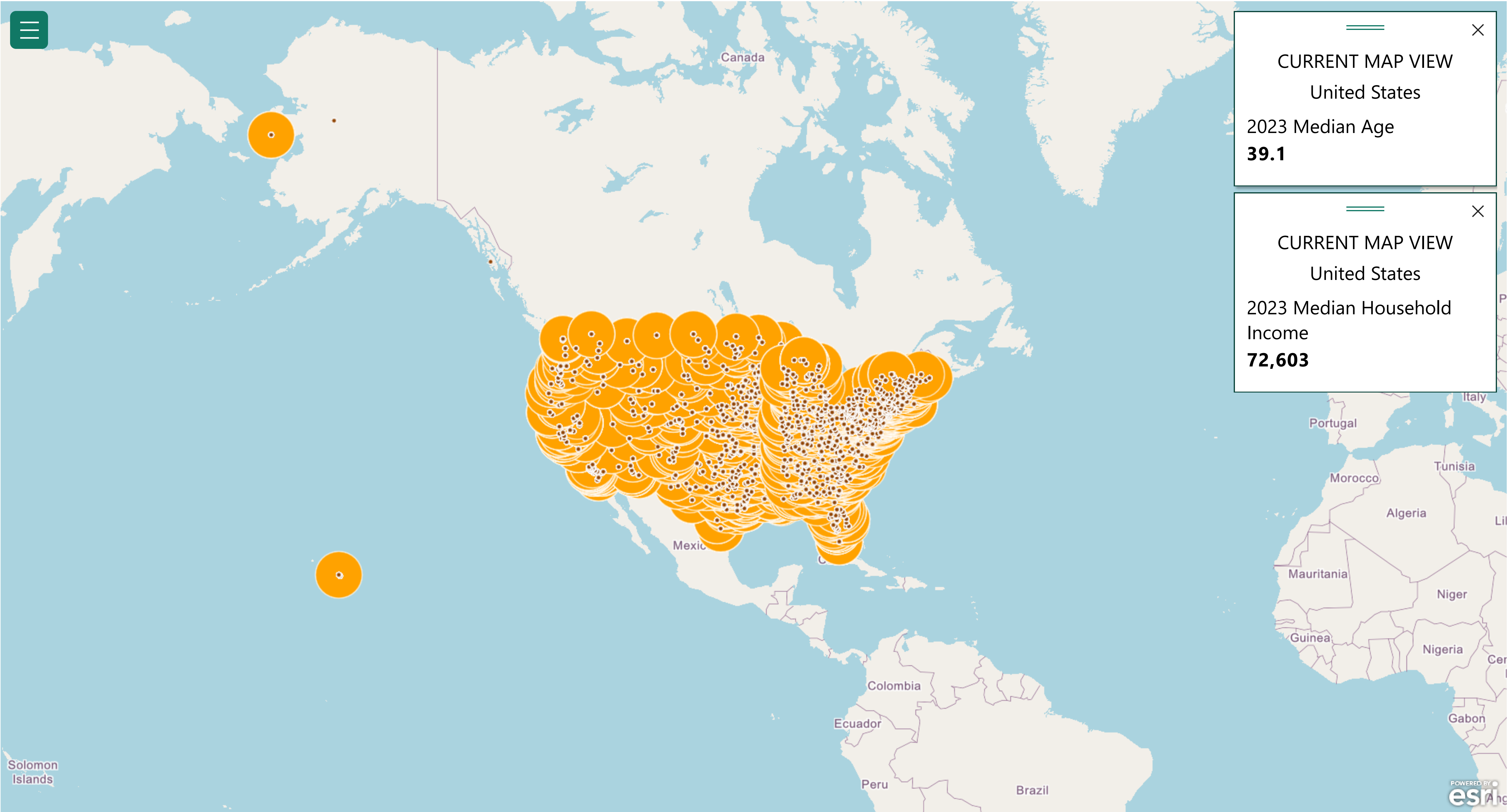


Fraud distribution per category

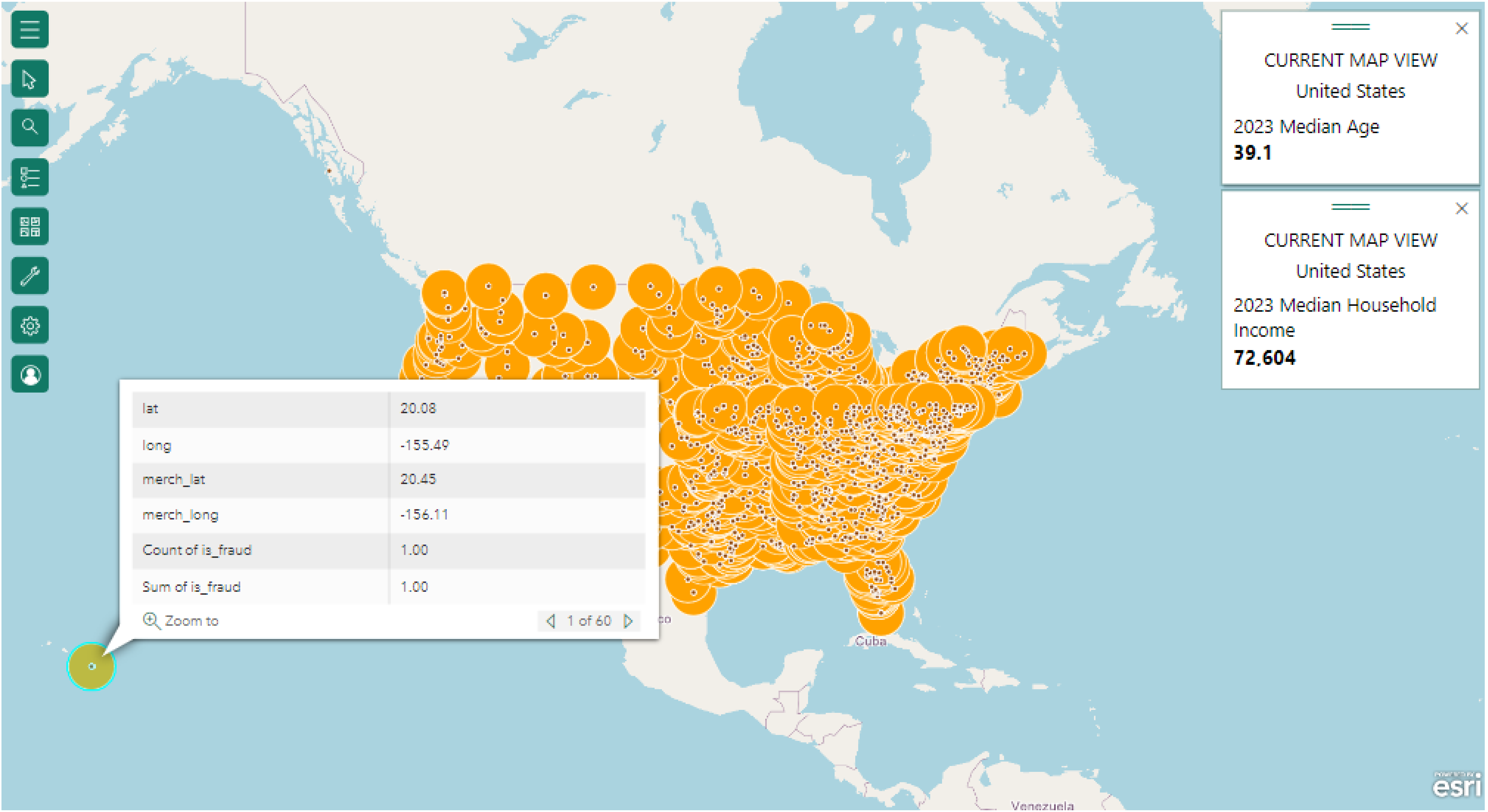
This pie chart was done by connecting Power BI to our SQL Server and using values from 2 different tables.



Sum of is_fraud, Sum of is_fraud and Count of is_fraud by lat, long, merch_lat and merch_long



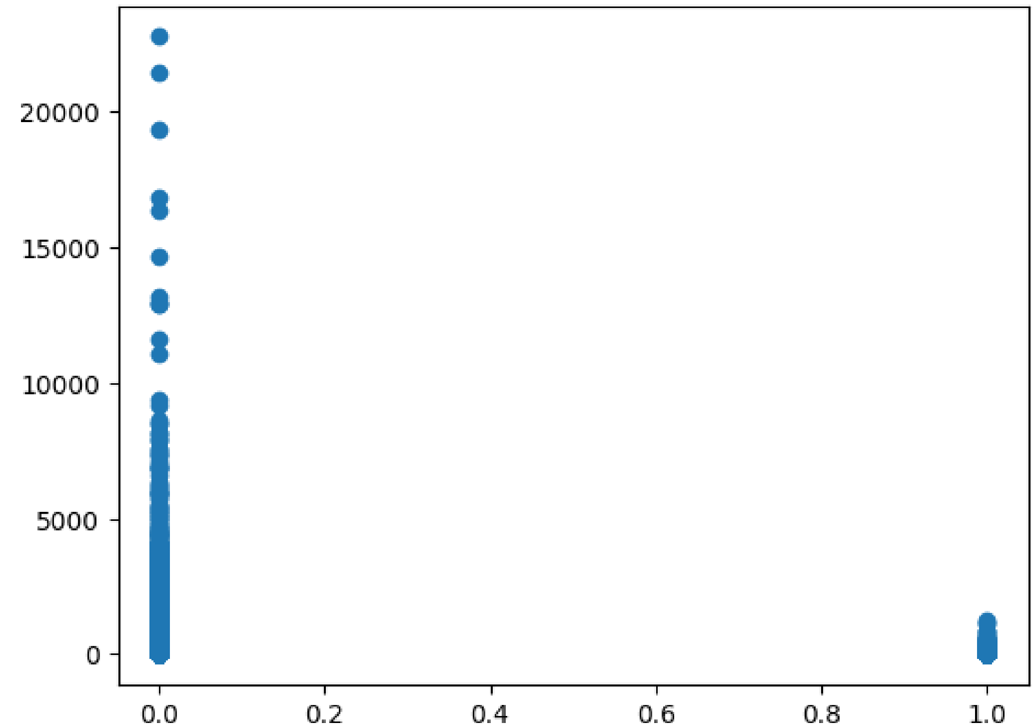
Sum of is_fraud, Sum of is_fraud and Count of is_fraud by lat, long, merch_lat and merch_long

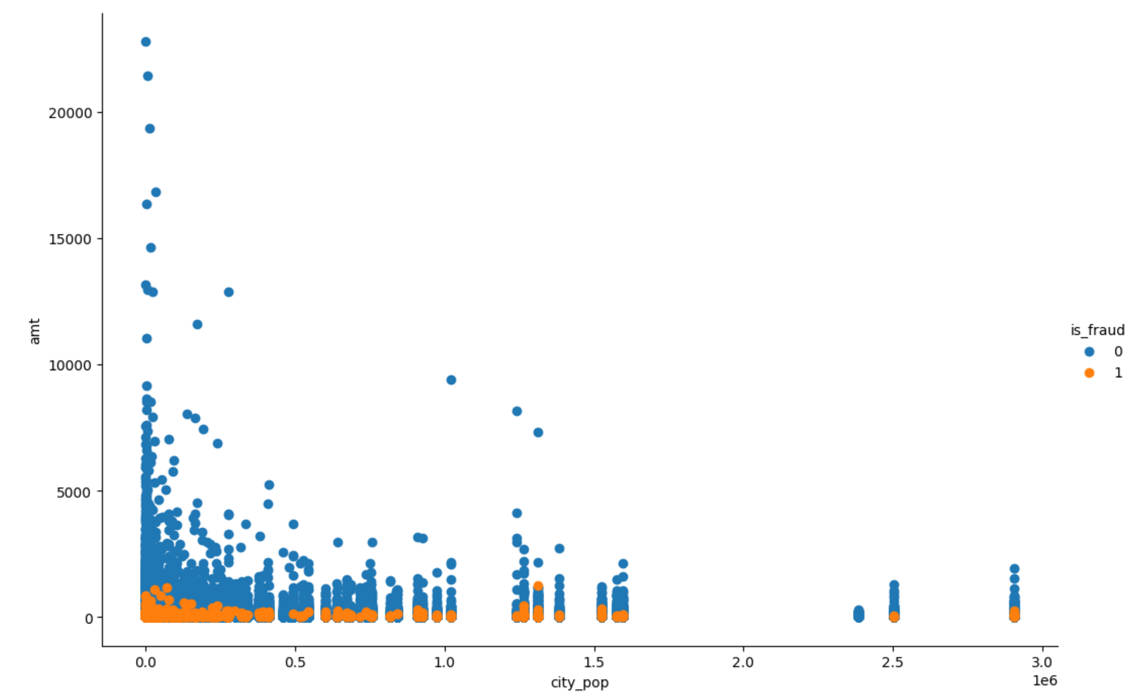


Machine Learning : Exploring patterns and insights.

	amt	lat	long	city_pop	merch_lat	merch_long	is_fraud
count	555719.000000	555719.000000	555719.000000	5.557190e+05	555719.000000	555719.000000	555719.000000
mean	69.392810	38.543253	-90.231325	8.822189e+04	38.542798	-90.231380	0.003860
std	156.745941	5.061336	13.721780	3.003909e+05	5.095829	13.733071	0.062008
min	1.000000	20.027100	-165.672300	2.300000e+01	19.027422	-166.671575	0.000000
25%	9.630000	34.668900	-96.798000	7.410000e+02	34.755302	-96.905129	0.000000
50%	47.290000	39.371600	-87.476900	2.408000e+03	39.376593	-87.445204	0.000000
75%	83.010000	41.894800	-80.175200	1.968500e+04	41.954163	-80.264637	0.000000
max	22768.110000	65.689900	-67.950300	2.906700e+06	66.679297	-66.952026	1.000000

After loading the data and getting the description, we did 2 scatters and then we Split the data into X and y and then into testing and training sets
Our scatter was able to answer the following question:
What is relationship of fraud transactions with amount of money?
We concluded based on our scatter plots that all fraud transactions occur for an amount less than 2500.





Since y _train value is highly imbalanced and discrete, we used SMOTE (if continuous Y , then no need).
so for Model 1 we did : Logistic Regression(max-it er=500, random _state=1911)
we got Model 1 test accuracy is 0.003858058014827611 and Model 1 train accuracy is 0.5.
Here's the classification reports for training and testing:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	415180
1	0.50	1.00	0.67	415180
accuracy			0.50	830360
macro avg	0.25	0.50	0.33	830360
weighted avg	0.25	0.50	0.33	830360

	precision	recall	f1-score	support
0	0.00	0.00	0.00	138394
1	0.00	1.00	0.01	536
accuracy			0.00	138930
macro avg	0.00	0.50	0.00	138930
weighted avg	0.00	0.00	0.00	138930

Model 2: Decision Tree

```
# Train Set -> To test overfiting  (Accuracy Train set vs test set)
pred = model.predict(X_train) #Predict the train
cm = confusion_matrix(y_train,pred)        #Train
ATrain2=(cm[0,0]+cm[1,1])/(sum(sum(cm))) # Confusion matrix
print("Model 2 train set accuracy is ",(cm[0,0]+cm[1,1])/(sum(sum(cm))))
```

```
Model 2 train set accuracy is  1.0
```

```
model = tree.DecisionTreeClassifier(random_state=1911)   #Max depth
```

```
model.fit(X_train,y_train)  #Learning
pred = model.predict(X_test) #Predict the test
cm = confusion_matrix(y_test,pred) # Confusion matrix
ATest2=(cm[0,0]+cm[1,1])/(sum(sum(cm)))
print("Model 2 test set accuracy is ", (cm[0,0]+cm[1,1])/(sum(sum(cm))))
```

```
Model 2 test set accuracy is  0.9723961707334629
```

Model 3: Random Forest

```
model.fit(X_train,y_train) # Train the model
pred = model.predict(X_test)# Predict the test
cm = confusion_matrix(y_test,pred)# Confusion matrix
print(cm)
ATest3=(cm[0,0]+cm[1,1])/(sum(sum(cm))) # Calculate accuracy
print("Model 3 test accuracy is ", (cm[0,0]+cm[1,1])/(sum(sum(cm))))
```

```
[[135889   2505]
 [   525     11]]
Model 3 test accuracy is  0.978190455625135
```

```
# Train Set -> To test overfiting  (Accuracy Train set vs test set)
model.fit(X_train,y_train)
pred = model.predict(X_train) #Predict the train
cm = confusion_matrix(y_train,pred)       #Train
print(cm)
ATrain3=(cm[0,0]+cm[1,1])/(sum(sum(cm)))
print("Model 3 train set accuracy is ",(cm[0,0]+cm[1,1])/(sum(sum(cm))))
```

```
[[415180      0]
 [     1 415179]]
Model 3 train set accuracy is  0.9999987957030686
```

Model 4: Gradient Boosting

```
model.fit(X_train,y_train)# Train the model
pred = model.predict(X_test)# Predict the test
cm = confusion_matrix(y_test,pred)# Confusion matrix
print(cm)
ATest4=(cm[0,0]+cm[1,1])/(sum(sum(cm)))# Calculate accuracy
print("Model 4 test set accuracy is ", (cm[0,0]+cm[1,1])/(sum(sum(cm))))
```

```
[[57664 80730]
 [  220   316]]
Model 4 test set accuracy is  0.41733246958900166
```

```
# Train Set -> To test overfiting  (Accuracy Train set vs test set)
pred = model.predict(X_train) #Predict the train
cm = confusion_matrix(y_train,pred)       #Confusion matrix
print(cm)
ATrain4=(cm[0,0]+cm[1,1])/(sum(sum(cm)))# Calculate accuracy
print("Model 4 train accuracy is ",(cm[0,0]+cm[1,1])/(sum(sum(cm))))
```

```
[[226001 189179]
 [104813 310367]]
Model 4 train accuracy is  0.6459463365287346
```

Model 5: Neural Network

```
model.fit(X_train,y_train) # Train the model
pred = model.predict(X_test)# Predict the test
cm = confusion_matrix(y_test,pred)# Confusion matrix
print(cm)
ATest3=(cm[0,0]+cm[1,1])/(sum(sum(cm))) # Calculate accuracy
print("Model 3 test accuracy is ", (cm[0,0]+cm[1,1])/(sum(sum(cm))))
```

```
[[135889   2505]
 [   525     11]]
Model 3 test accuracy is  0.978190455625135
```

```
# Train Set -> To test overfiting  (Accuracy Train set vs test set)
model.fit(X_train,y_train)
pred = model.predict(X_train) #Predict the train
cm = confusion_matrix(y_train,pred)       #Train
print(cm)
ATrain3=(cm[0,0]+cm[1,1])/(sum(sum(cm)))
print("Model 3 train set accuracy is ",(cm[0,0]+cm[1,1])/(sum(sum(cm))))
```

```
[[415180      0]
 [     1 415179]]
Model 3 train set accuracy is  0.9999987957030686
```

	Model	Test Accuracy	Train Accuracy
2	RandomForest	0.978190	0.999999
1	DecisionTree	0.972396	1.000000
3	GradientBoosting	0.417332	0.645946
4	Neural Network	0.199100	0.525760
0	LogisticRegression	0.003858	0.500000

SUMMARY and CONCULSION

- All Fraud Transactions occur for an amount below 2500. Thus, the bank can infer clearly that the fraud committers try to commit frauds of smaller amounts to avoid suspicion.
- The fraud transactions are equitable distributed throughout range of city popoluation and there is no clear relationship of city popoluation with commiting of fraud.
- The number of fraud transactions are very few comparted to legitimate transactions and it has to be balanced in order for a fair comparison to prevent the model from overfitting.
- SMOTE overcomes overfitting by synthetically oversampling minority class labels.
- Among 5 different models used, Random Forest Model and Decision Tree model gave the maximum accuracy for Test set and Train set. Random Forest Algorithm is robust against overfitting, robust to outliers and non-linear data and runs efficiently on large database. Hence, we would choose Random Forest method over Decision Tree model for our dataset to predict fraudulant transaction.