Credit Card Fraud Detection Model

Team #2

Michael Tran, Juan Gomez, Hamza Khamissa, Khilan Desai

Feature Engineering

Finding the relevant data to train on

New Features

- Distance of Cardholder from Merchant
- Age of Cardholder
- Transaction Time
- Fraud Rate in Cities
- Fraud Rate in Jobs
- History of Cardholder Fraud Experience

Distance From Cardholder to Merchant

```python
from vincenty import vincenty_inverse


def distance_from_home(trans_row, cardholder_row):
    coords_home = (cardholder_row["longitude"], cardholder_row["latitude"])
    coords_purchase = (trans_row["merchLongitude"], trans_row["merchLatitude"])

    return vincenty_inverse(coords_home, coords_purchase).km


df["distance_customer_merchant"] = df.apply(
    lambda row: distance_from_home(row, df.iloc[0]), axis=1
)
```

Dropped:
- latitude
- longitude
- merchLatitude
- merchLongitude

Added:
- distance_customer_merchant

The current features ‘latitude’, ‘longitude’, ‘merchLatitude’, and ‘merchLongitude’ alone do not provide relevant information.

We calculated the distance between and transformed into kilometers. This allowed us to combine the four features into one measurement.

Age of Cardholder + Transaction Time

```
df["transDate"] = pd.to_datetime(df["transDate"])
df["dateOfBirth"] = pd.to_datetime(df["dateOfBirth"])

df["trans_day"] = df["transDate"].dt.dayofyear
df["trans_weekday"] = df["transDate"].dt.weekday
df["trans_hour"] = df["transDate"].dt.hour
df["age_at_transaction"] = df["transDate"].dt.year - df["dateOfBirth"].dt.year
```

Dropped:

- dateOfBirth
- transDate

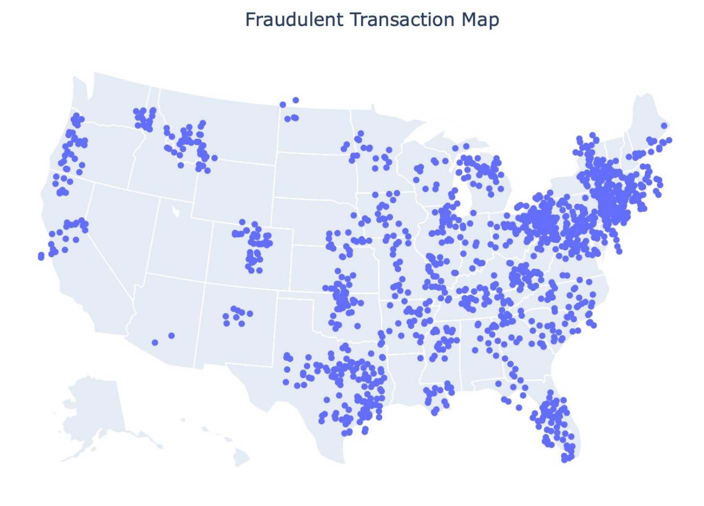
Added:

- trans_day
- trans_weekday
- trans_hour
- age_at_transaction

Using ‘dateOfBirth’ with ‘transDate’ we are able to calculate the age of the cardholder at the time of transaction.
We also broke down the `transDate` feature into three new features based on day of the year, weekday, and hour.

Fraud Rate in Cities

```
# encode city
grouped_transactions = df.groupby("city")
total_transactions = grouped_transactions.size()
fraud_transactions = grouped_transactions["isFraud"].sum()
fraud_rate = (fraud_transactions / total_transactions) * 100
result_dict = fraud_rate.to_dict()

df["city_fraudrate"] = df["city"].map(result_dict)
df.drop(columns=["city"], inplace=True)
```



We computed city_fraudrate as a measure of fraud rates in a given city.

In the figure, you can see that fraud cases are clustered into higher density areas that are more likely to experience fraud.

Dropped:
- city

Added:
- city_fraudrate

Fraud Rate in Jobs

```
# encode job
grouped_transactions = df.groupby("job")
total_transactions = grouped_transactions.size()
fraud_transactions = grouped_transactions["isFraud"].sum()
fraud_rate = (fraud_transactions / total_transactions) * 100
result_dict = fraud_rate.to_dict()

df["job_fraudrate"] = df["job"].map(result_dict)
df.drop(columns=["job"], inplace=True)
```

Dropped:
- job

Added:
- job_fraudrate

Similarly ‘job_fraudrate’ was created to test for correlation in different industries.

Categorical Variables

We want to encode these categorical variables into numerical values for the model.

Cardholder City

The city in which the cardholder is located.

Some cities may experience higher rates of fraud, and thus purchases may have higher chances of being fraudulent.

Cardholder Job

The occupation in which the cardholder is employed.

Some jobs may experience higher risks of fraud due to the exposure or nature of their work.

Purchase Category

The category of the purchase.

Certain goods and services are more prone to being fraudulent, or are more attractive targets of fraud.

Purchase Category

We decided to use one-hot encoding for this feature.

The lower number of factors for this feature allowed for a manageable number of new columns to be created.

Dropped:

- category

Added:

- category_entertainment
- category_food_dining
- category_gas_transport
- category_grocery_net
- category_grocery_pos
- category_health_fitness
- category_home
- category_kids_pets
- category_misc_pos
- category_personal_care
- category_shopping_net
- category_shopping_pos
- category_travel

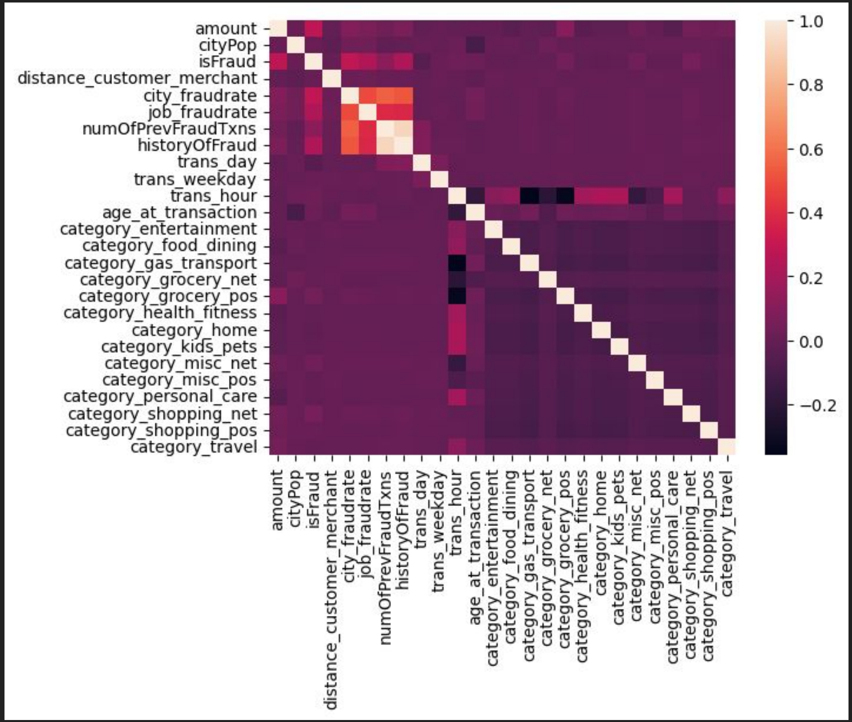
Cardholder History of Fraud

According to a 2024 study by DeLiema et al, past financial fraud victimization has an effect on likelihood to experience repeat victimization.
This also affects age groups differently, such as ages 70 - 80 experiencing higher rates of repeat victimization.

'historyOfFraud' is a boolean feature we added to note if certain cardholders have experienced fraud previously.

Source:
DeLiema M, Langton L, Brannock D, Preble E. Fraud victimization across the lifespan: evidence on repeat victimization using perpetrator data. J Elder Abuse Negl. 2024 Feb 22:1-24. doi: 10.1080/08946566.2024.2321923. Epub ahead of print. PMID: 38389208.

Correlation Matrix



Sorted Correlation Values to isFraud:

isFraud	1.000000
city_fraudrate	0.284640
amount	0.282902
historyOfFraud	0.231625
job_fraudrate	0.226799
numOfPrevFraudTxns	0.114554
category_shopping_net	0.052611

category_personal_care	-0.014043
category_health_fitness	-0.016817
category_food_dining	-0.018768
category_kids_pets	-0.019225
category_home	-0.020704
trans_day	-0.043228

Other Dropped Features

- creditCardNum
- business
- firstName
- lastName
- gender
- street
- state
- zip
- transNum

These features were removed from the dataset.
They mostly comprise of identifying metadata and we found our initial exploratory data analysis did not find meaningful connections to fraud.

Model Training and Evaluation

Evaluating our models for performance at fraud detection

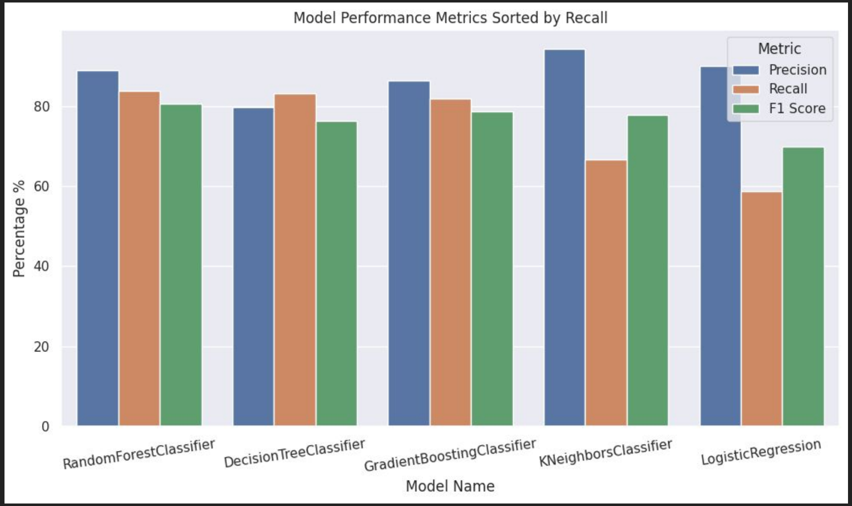
Choosing a model

- LogisticRegression
- KNeighborsClassifier
- RandomForestClassifier
- DecisionTreeClassifier
- GradientBoostingClassifier

```
pipelines = {
    "LogisticRegression": make_pipeline(StandardScaler(), LogisticRegression()),
    "KNeighborsClassifier": make_pipeline(StandardScaler(), KNeighborsClassifier()),
    "RandomForestClassifier": make_pipeline(StandardScaler(), RandomForestClassifier()),
    "DecisionTreeClassifier": make_pipeline(StandardScaler(), DecisionTreeClassifier()),
    "GradientBoostingClassifier": make_pipeline(
        StandardScaler(), GradientBoostingClassifier()
    ),
}

X = df_train.drop(columns=["isFraud"])
y = df_train["isFraud"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
from sklearn.model_selection import cross_validate

results = []

for name, model in fit_models.items():
    print(f"Evaluating model: {name}")

    # Perform cross-validation and request scoring on all metrics
    scores = cross_validate(model, X, y, cv=10, scoring=['precision', 'recall', 'f1'],
    return_train_score=False)

    # Calculate the mean of the scores for each metric
    precision_mean = round(scores['test_precision'].mean() * 100, 2)
    recall_mean = round(scores['test_recall'].mean() * 100, 2)
    f1_mean = round(scores['test_f1'].mean() * 100, 2)

    # Append the results
    results.append({
        "Model": name,
        "Precision": precision_mean,
        "Recall": recall_mean,
        "F1 Score": f1_mean
    })

results_df = pd.DataFrame(results)
results_df
```

Models Comparison



When evaluating models, we valued Recall over Precision.

Our rationale is that in real-world fraud cases, all flagged cases would be reviewed, and thus we would rather have more false positives than false negatives in order to maximize detection.

We found that Random Forest had the best performance overall.
We worked with this model on hyperparameter optimization.

Undersampling, Oversampling

The dataset is very unbalanced:

- Total number of transactions: 181822
- Cases of fraud: 1336
- Normal transactions: 180486
- Percentage of fraud: 0.7348%

imbalanced-learn package in Python provides RandomUnderSampler and RandomOverSampler.

We found that using undersampling made the most sense, and also performed well.

Do Under and Over Sampling

```python
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler

rus = RandomUnderSampler(sampling_strategy=0.5)
X_rus, y_rus = rus.fit_resample(X, y)

ros = RandomOverSampler()
X_ros, y_ros = ros.fit_resample(X, y)

y.value_counts(), y_rus.value_counts(), y_ros.value_counts()

data = {
    'Original': y.value_counts(),
    'Undersampled': y_rus.value_counts(),
    'Oversampled': y_ros.value_counts()
}
df_counts = pd.DataFrame(data)
df_counts
```

[95] ✓ 0.4s

isFraud	Original	Undersampled	Oversampled
False	180486	2672	180486
True	1336	1336	180486

Hyperparameter Optimization

Using our Sklearn GridSearchCV we were able to increase our scores by finding the optimal Hyperparameters

Grid search for RandomForestClassifier

- n_estimators: number of trees
- max_features: features to use
- min_samples_leaf: reduces overfitting by regularizing trees

```
forest_params = [
    {
        'n_estimators': [100, 300, 500, 100],
        'max_features': [0.4, 0.6, 'sqrt', 'log2'],
        'min_samples_leaf': [1, 2, 3, 5]
    }
]


RFC_grid_search = GridSearchCV(
    RFC,
    forest_params,
    scoring=['recall', 'precision', 'f1'],
    refit='f1',
    n_jobs=-1
)

RFC_grid_search.fit(X_train, y_train)

print(RFC_grid_search.best_params_)
print(RFC_grid_search.best_score_)
```

Final Model Evaluation

98.63% Accuracy

97.76% Precision

98.13% Recall

97.94% F1 Score

Visualization

What does the data tell us

Feature Importance

historyOfFraud - 32.08%

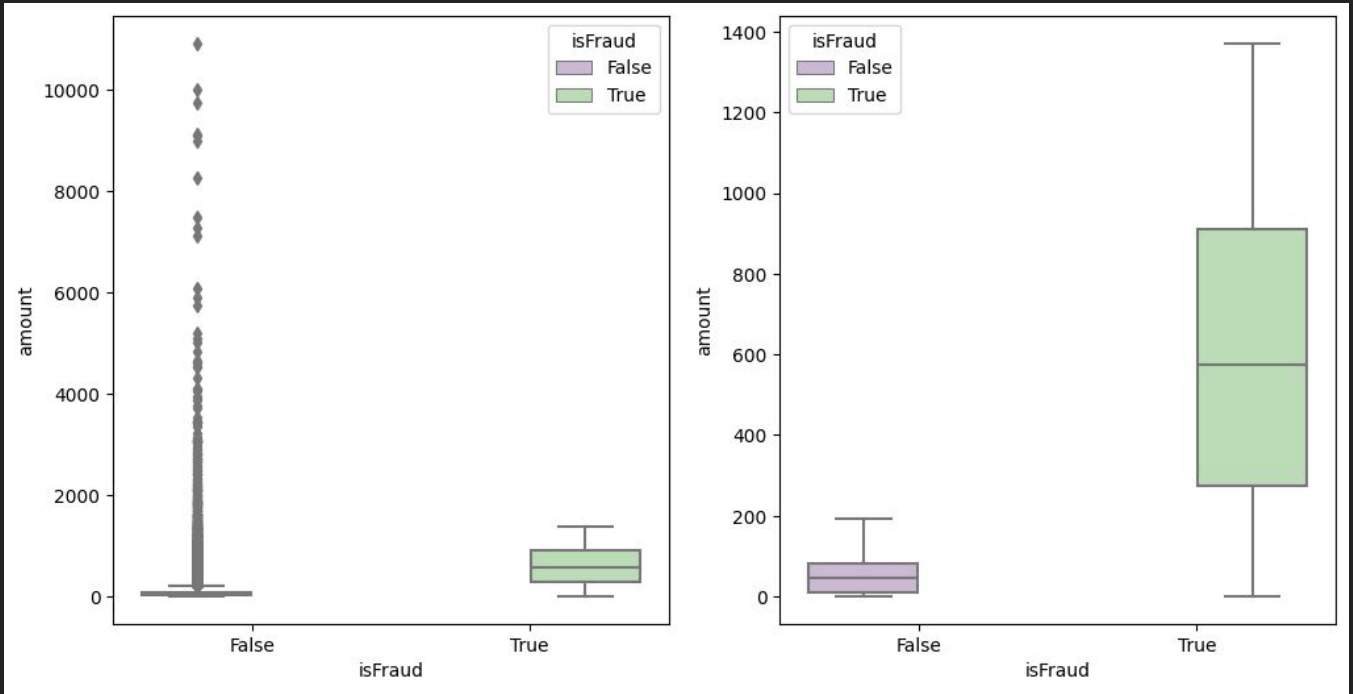
amount - 28.43%

city_fraudrate - 18.05%

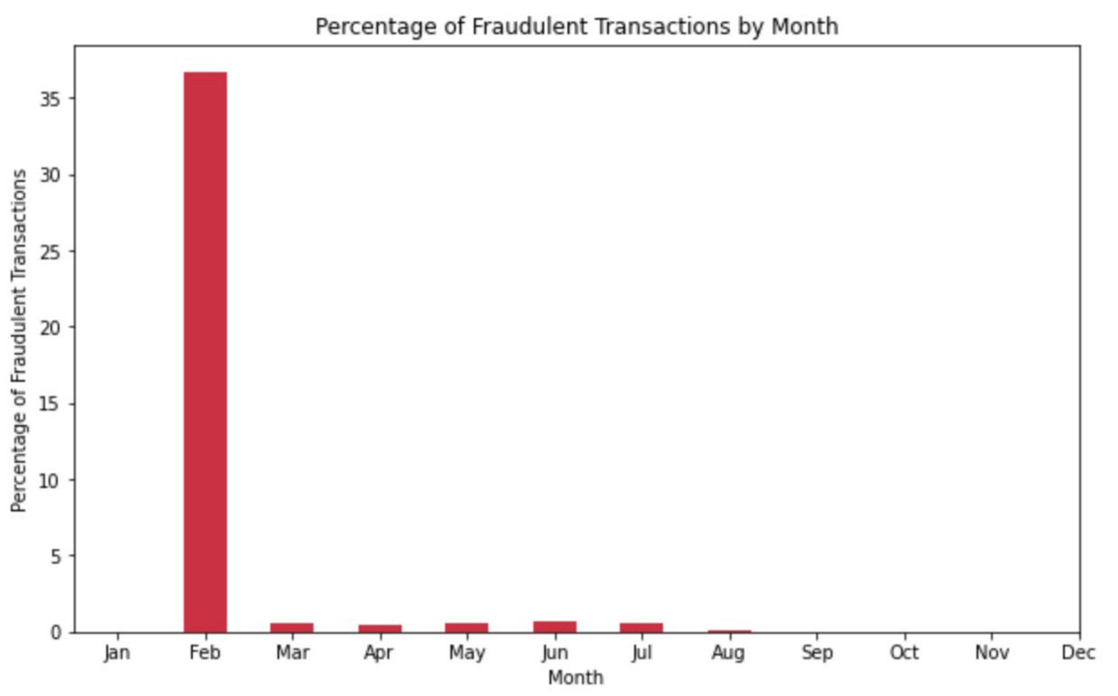
trans_hour - 7.22%
job_fraudrate - 5.73%
trans_day - 3.85%
category_gas_transport - 1.05%
distance_customer_merchant - 0.47%
age_at_transaction - 0.46%
cityPop - 0.41%
state - 0.38%
category_grocery_pos - 0.28%
category_food_dining - 0.28%
trans_weekday - 0.25%

category_home 0.19%
category_shopping_net 0.15%
category_shopping_pos 0.13%
category_misc_pos 0.13%
category_kids_pets 0.11%
category_personal_care 0.10%
category_grocery_net 0.09%
category_health_fitness 0.07%
category_travel 0.05%
category_misc_net 0.02%
category_entertainment 0.02%

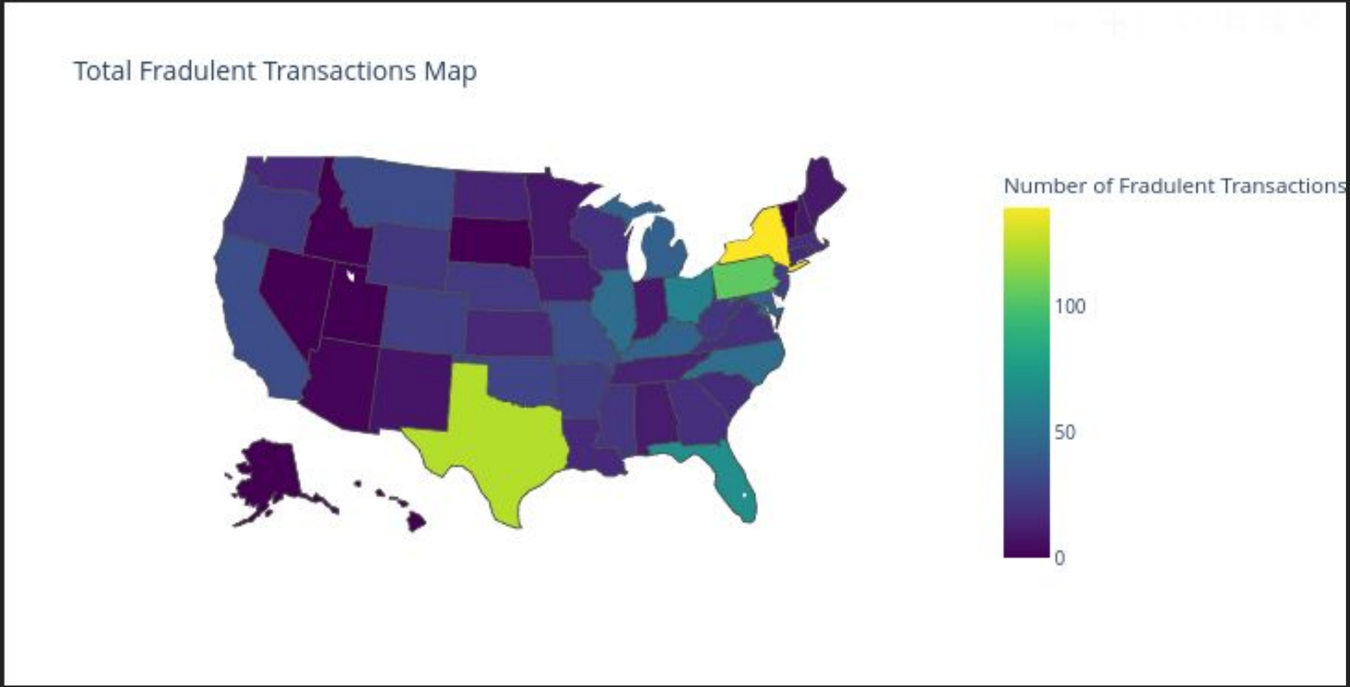
Purchase amounts with highest fraud rate



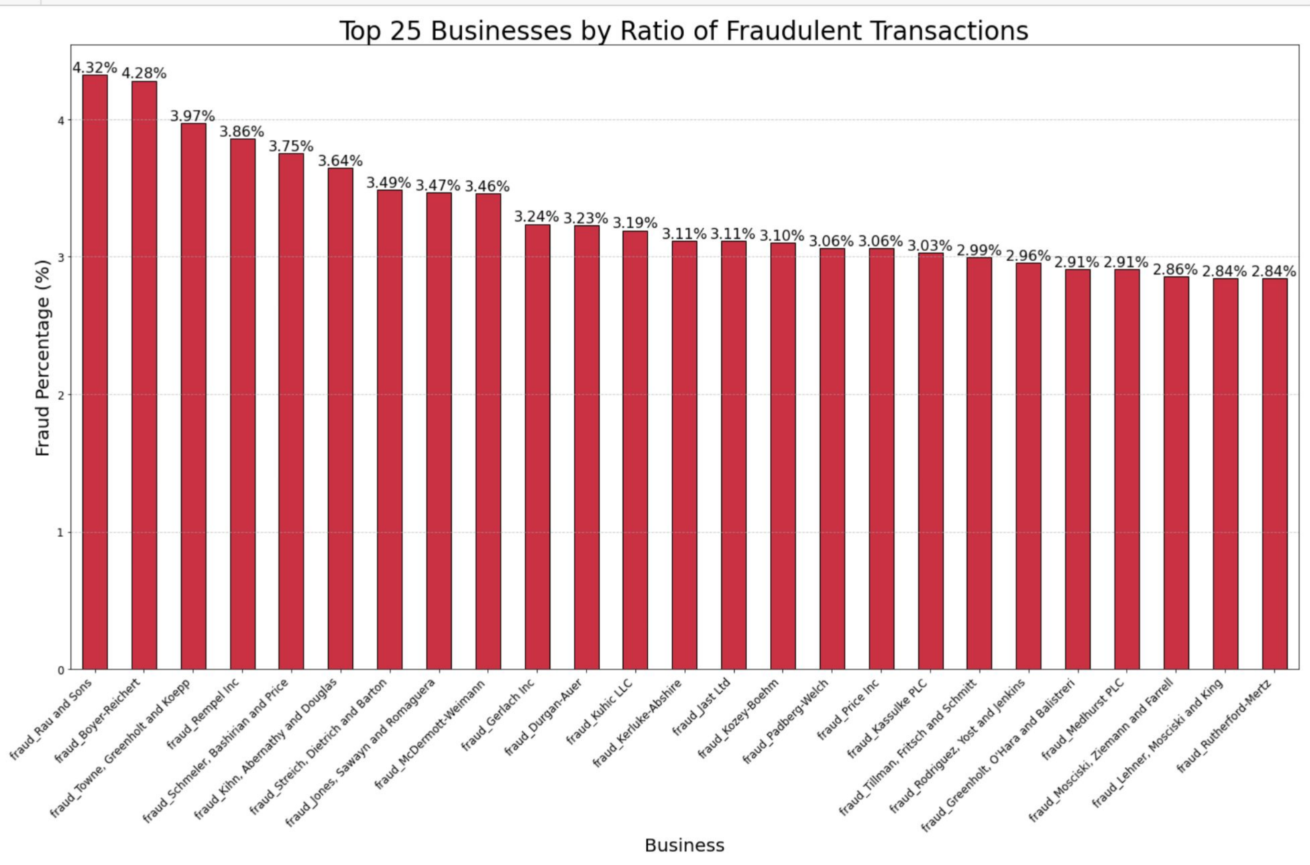
Percentage of Fraud by Month



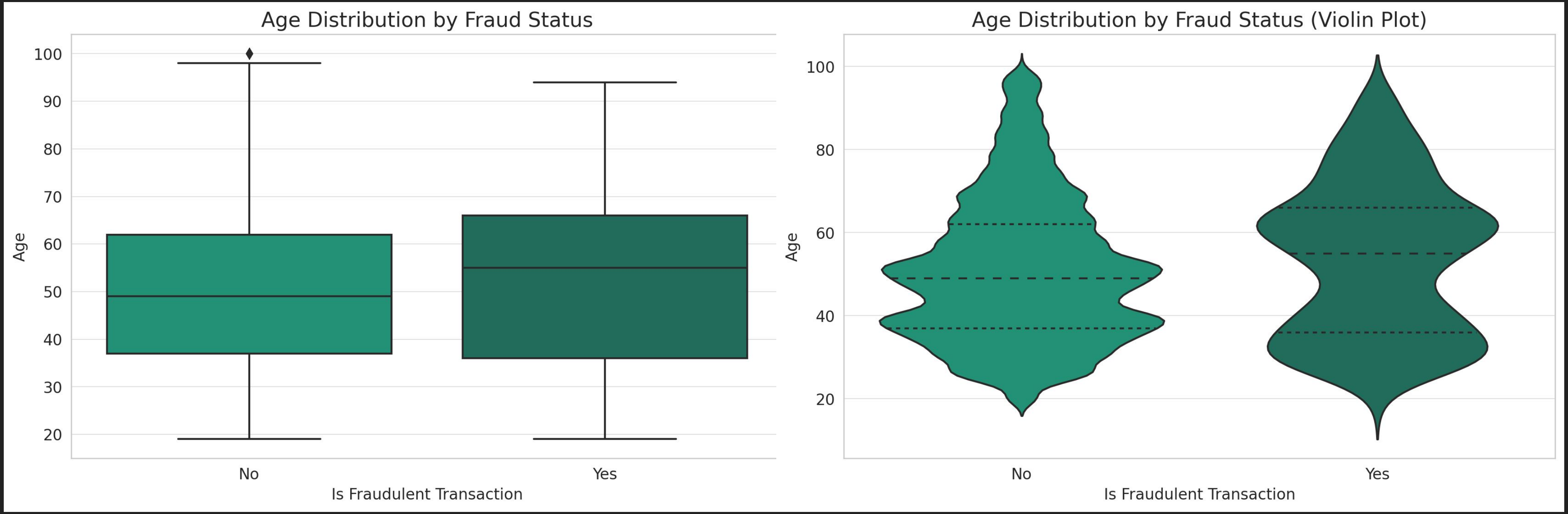
States with Highest Number of Fraud



Top 25 Businesses for fraud



Ages with highest fraud rate



Most Likely Victim of Fraud

- 55 year old male
- In a highly populated city
- The purchase amount is above their average payment
- The purchase was made at an odd time
- Purchase was made far from cardholder
- Purchase was made by a cardholder who has a history of fraud

Thank you!

Any Questions?