ICS 474 Project Report

Big Data Analytics

Dr. Muzammil Behzad

Department of Information and Computer Science

King Fahd University of Petroleum and Minerals

Hussein Al Yami - 202066180

Mohammed Al Khunjuf - 202028400

Mahmoud Ghaleb Binladin - 202029540

Dataset

https://www.kaggle.com/datasets/priyamchoksi/credit-card-transactions-dataset

Table of Contents

Data Understanding and Exploration

Selected Dataset: https://www.kaggle.com/datasets/priyamchoksi/credit-card-transactions-dataset

1.1 Dataset Overview

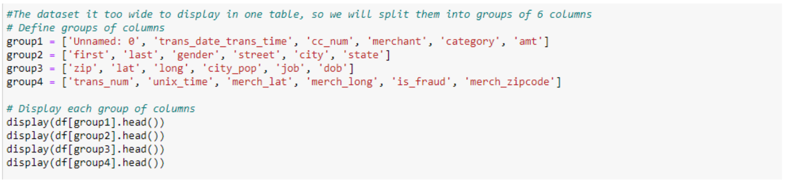
The Credit Card Transactions Dataset contains 1.3M credit card transaction records capturing payment details, merchant information, and customer data. It serves primarily for fraud detection analysis, customer behavior studies, and transaction pattern recognition. Each record includes transaction timestamp, amount, merchant details, and location data, making it suitable for both financial analysis and anomaly detection. The dataset covers transactions from 2019, providing comprehensive information about credit card purchases with key features including transaction amounts, timestamps, merchant information, and customer details. This rich dataset enables various analytical applications, particularly in fraud detection and spending pattern analysis.

1.2 Feature Description

The dataset features: 0', 'trans_date_trans_time', 'cc_num', 'merchant', 'category', 'amt', 'first', 'last', 'gender', 'street', 'city', 'state', 'zip', 'lat', 'long', 'city_pop', 'job', 'dob', 'trans_num', 'unix_time', 'merch_lat', 'merch_long', 'is_fraud', and 'merch_zipcode'.

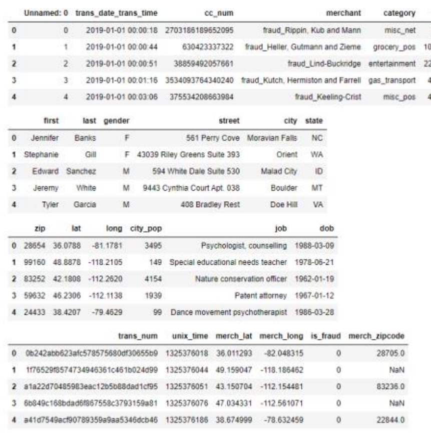
These features represent distinct aspects of credit card transactions: The dataset reveals a maximum of 983 unique clients based on credit card numbers, with several columns identified for removal due to redundancy or limited analytical value including 'Unnamed', 'trans_num', 'unix_time', 'merch_lat', 'merch_long', 'merch_zipcode', 'street', 'city', 'zip', and 'state'. New features can be derived from existing data: distance calculations from coordinates, time-based columns from transaction times, banking network identification from credit card numbers, and age from date of birth.

1.3 Dataset Structure



```python
#The dataset it too wide to display in one table, so we will split them into groups of 6 columns
# Define groups of columns
group1 = ['Unnamed: 0', 'trans_date_trans_time', 'cc_num', 'merchant', 'category', 'amt']
group2 = ['first', 'last', 'gender', 'street', 'city', 'state']
group3 = ['zip', 'lat', 'long', 'city_pop', 'job', 'dob']
group4 = ['trans_num', 'unix_time', 'merch_lat', 'merch_long', 'is_fraud', 'merch_zipcode']

# Display each group of columns
display(df[group1].head())
display(df[group2].head())
display(df[group3].head())
display(df[group4].head())
```

The dataset contains 1,296,675 rows and 24 columns structured as a flat tabular database in DataFrame format, with each row representing a single credit card transaction. The columns capture transaction details, merchant information, customer demographics, and location data. due to the wide nature of the dataset we have made a split of different groups (4) to better visualize the stucture and features of the data.



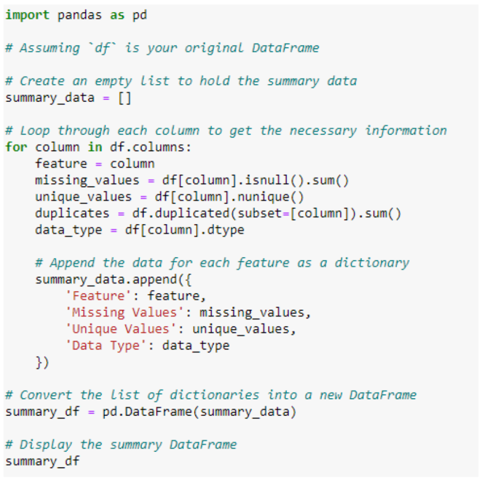
	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96

	first	last	gender	street	city	state
0	Jennifer	Banks	F	561 Perry Cove	Moravian Falls	NC
1	Stephanie	Gill	F	43039 Riley Greens Suite 393	Orient	WA
2	Edward	Sanchez	M	594 White Dale Suite 530	Malad City	ID
3	Jeremy	White	M	9443 Cynthia Court Apt. 038	Boulder	MT
4	Tyler	Garcia	M	408 Bradley Rest	Doe Hill	VA

	zip	lat	long	city_pop	job	dob
0	28654	36.0788	-81.1781	3495	Psychologist, counselling	1988-03-09
1	99160	48.8878	-118.2105	149	Special educational needs teacher	1978-06-21
2	83252	42.1808	-112.2620	4154	Nature conservation officer	1962-01-19
3	59632	46.2306	-112.1138	1939	Patent attorney	1967-01-12
4	24433	38.4207	-79.4629	99	Dance movement psychotherapist	1986-03-28

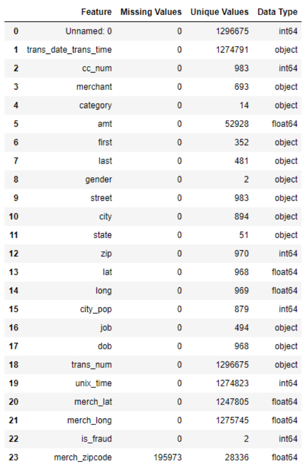
	trans_num	unix_time	merch_lat	merch_long	is_fraud	merch_zipcode
0	0b242abb623afc578575680df30655b9	1325376018	36.011293	-82.048315	0	28705.0
1	1f76529f8574734946361c461b024d99	1325376044	49.159047	-118.186462	0	NaN
2	a1a22d70485983eac12b5b88dad1cf95	1325376051	43.150704	-112.154481	0	83236.0
3	6b849c168bdad6f867558c3793159a81	1325376076	47.034331	-112.561071	0	NaN
4	a41d7549acf90789359a9aa5346dcb46	1325376186	38.674999	-78.632459	0	22844.0

1.4 Missing Values and Duplicates



```python
import pandas as pd

# Assuming `df` is your original DataFrame

# Create an empty list to hold the summary data
summary_data = []

# Loop through each column to get the necessary information
for column in df.columns:
    feature = column
    missing_values = df[column].isnull().sum()
    unique_values = df[column].nunique()
    duplicates = df.duplicated(subset=[column]).sum()
    data_type = df[column].dtype

    # Append the data for each feature as a dictionary
    summary_data.append({
        'Feature': feature,
        'Missing Values': missing_values,
        'Unique Values': unique_values,
        'Data Type': data_type
    })

# Convert the list of dictionaries into a new DataFrame
summary_df = pd.DataFrame(summary_data)

# Display the summary DataFrame
summary_df
```

The code creates a summary DataFrame of the dataset's features. It loops through each column in the original DataFrame to collect information about missing values (using isnull().sum()), unique values (using unique()), and data types (using dtype). This information is stored in a list of dictionaries, with each dictionary containing the feature name, missing value count, unique value count, and data type. The list is then converted into a new DataFrame using pd.DataFrame().

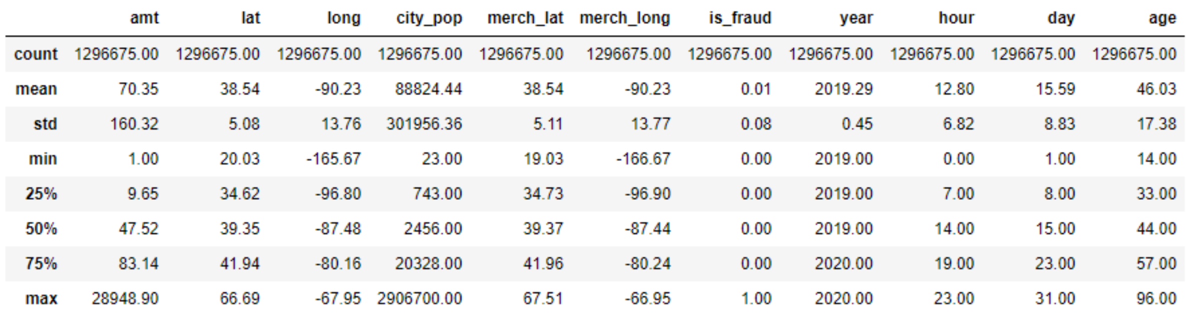


	Feature	Missing Values	Unique Values	Data Type
0	Unnamed: 0	0	1296675	int64
1	trans_date_trans_time	0	1274791	object
2	cc_num	0	983	int64
3	merchant	0	693	object
4	category	0	14	object
5	amt	0	52928	float64
6	first	0	352	object
7	last	0	481	object
8	gender	0	2	object
9	street	0	983	object
10	city	0	894	object
11	state	0	51	object
12	zip	0	970	int64
13	lat	0	968	float64
14	long	0	969	float64
15	city_pop	0	879	int64
16	job	0	494	object
17	dob	0	968	object
18	trans_num	0	1296675	object
19	unix_time	0	1274823	int64
20	merch_lat	0	1247805	float64
21	merch_long	0	1275745	float64
22	is_fraud	0	2	int64
23	merch_zipcode	195973	28336	float64

The output shows 24 features with their corresponding statistics in four columns: Feature name, Missing Values count, Unique Values count, and Data Type. Most features have 0 missing values except 'merch_zipcode' which has 195973 missing values. The data types vary between int64, object, and float64. The number of unique values ranges significantly, from just 2 values for 'gender' and 'is_fraud' to over 1.2 million values for 'trans_date_trans_time'.

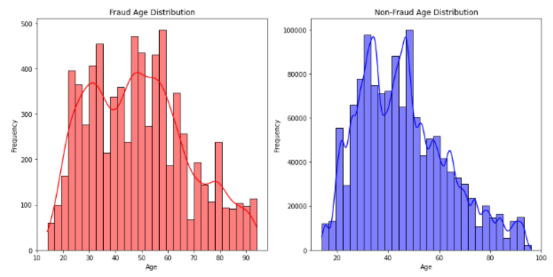
1.5 Statistical Summary

The dataset contains 1,296,675 records with various transaction and demographic metrics. Transaction amounts average $70.35, ranging from $1 to $28,948.90. Geographic coordinates show average latitude of 38.54 and longitude of -90.23, with merchant locations closely matching these averages. City populations vary dramatically from 23 to 2,906,700, with a mean of 88,824. Transactions span 2019-2020, occurring between 0:00-23:00 hours (mean 12.80) and across all days of the month (mean 15.59). Customer ages range from 14 to 96 years, averaging 46.03 years. Fraud is indicated as a binary variable with a mean of 0.01, suggesting approximately 1% fraudulent transactions.

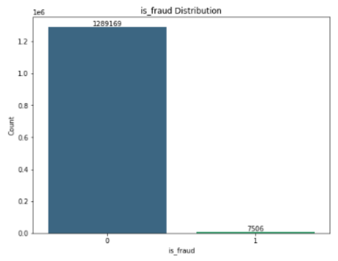


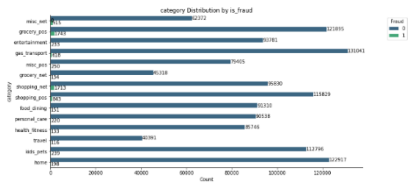
	amt	lat	long	city_pop	merch_lat	merch_long	is_fraud	year	hour	day	age
count	1296675.00	1296675.00	1296675.00	1296675.00	1296675.00	1296675.00	1296675.00	1296675.00	1296675.00	1296675.00	1296675.00
mean	70.35	38.54	-90.23	88824.44	38.54	-90.23	0.01	2019.29	12.80	15.59	46.03
std	160.32	5.08	13.76	301956.36	5.11	13.77	0.08	0.45	6.82	8.83	17.38
min	1.00	20.03	-165.67	23.00	19.03	-166.67	0.00	2019.00	0.00	1.00	14.00
25%	9.65	34.62	-96.80	743.00	34.73	-96.90	0.00	2019.00	7.00	8.00	33.00
50%	47.52	39.35	-87.48	2456.00	39.37	-87.44	0.00	2019.00	14.00	15.00	44.00
75%	83.14	41.94	-80.16	20328.00	41.96	-80.24	0.00	2020.00	19.00	23.00	57.00
max	28948.90	66.69	-67.95	2906700.00	67.51	-66.95	1.00	2020.00	23.00	31.00	96.00

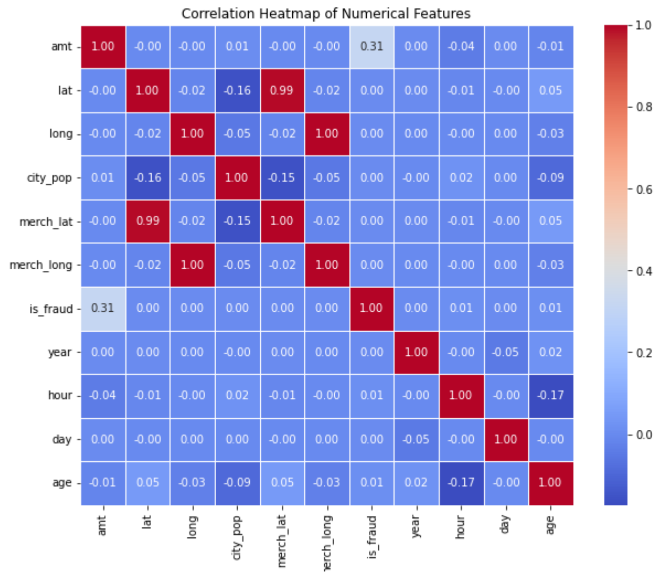
1.6 Data Distribution and correlation Analysis



The analysis of fraud patterns reveals significant insights: Out of 1,296,675 total transactions, 1,289,169 are fraudulent while only 7,506 are legitimate, indicating an extremely high fraud rate. Entertainment emerges as the most vulnerable sector with 131,041 fraudulent transactions. Gender analysis shows females are more affected with 706,128 fraud cases compared to 583,041 for males. Age distribution indicates the highest concentration of fraud occurs in the 50-60 age bracket. Visa cards experience the highest volume of both fraudulent and legitimate transactions, though this correlates with its larger market share in overall transaction volume. These patterns suggest targeted demographics and sectors for enhanced security measures.

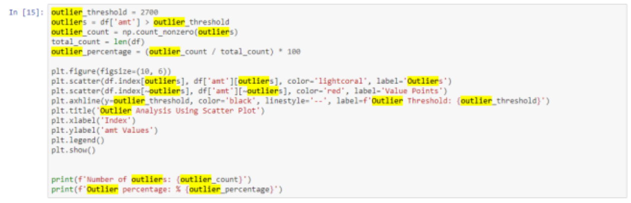




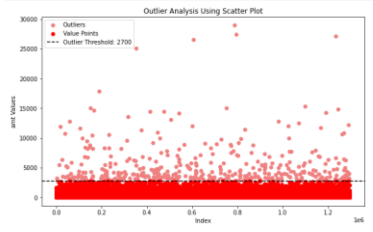


1.7 Outlier Detection

The scatter plot illustrates transaction amount outlier analysis with a threshold set at $2,700. Out of approximately 1.2 million transactions, 430 cases (0.033%) are identified as outliers. These outlier transactions are scattered above the threshold line, ranging from $2,700 up to nearly $30,000, shown in light coral color. The majority of transactions (shown in red) fall below the threshold, clustering densely between $0-$2,700. The distribution pattern suggests that while most transactions are within normal range, there are sporadic high-value transactions that could warrant further investigation for potential fraud or unusual spending patterns.



```
In [15]: outlier_threshold = 2700
         outliers = df['amt'] > outlier_threshold
         outlier_count = np.count_nonzero(outliers)
         total_count = len(df)
         outlier_percentage = (outlier_count / total_count) * 100

         plt.figure(figsize=(10, 6))
         plt.scatter(df.index[outliers], df['amt'][outliers], color='lightcoral', label='Outliers')
         plt.scatter(df.index[~outliers], df['amt'][~outliers], color='red', label='Value Points')
         plt.axhline(y=outlier_threshold, color='black', linestyle='--', label=f'Outlier Threshold: {outlier_threshold}')
         plt.title('Outlier Analysis Using Scatter Plot')
         plt.xlabel('Index')
         plt.ylabel('amt Values')
         plt.legend()
         plt.show()


         print(f'Number of outliers: {outlier_count}')
         print(f'Outlier percentage: % {outlier_percentage}')
```

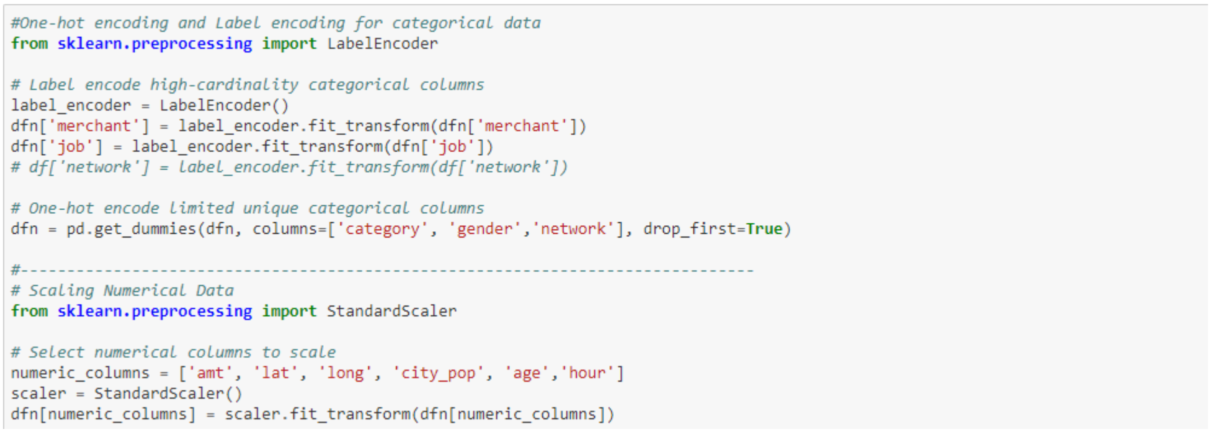


```
Number of outliers: 430
Outlier percentage: % 0.033161740605780164
```

Data Preprocessing

2.1 Feature Scaling and Encoding Variables

The code performs data preprocessing for machine learning by handling both categorical and numerical features. For categorical data, it uses two approaches: LabelEncoder for high-cardinality variables ('merchant' and 'job') and one-hot encoding through get_dummies() for limited unique categorical columns ('category', 'gender', 'network'). For numerical features ('amt', 'lat', 'long', 'city_pop', 'age', 'hour'), it applies StandardScaler to normalize the values. This preprocessing ensures all features are in appropriate numerical format and scale for the applied algorithms.



```
#One-hot encoding and Label encoding for categorical data
from sklearn.preprocessing import LabelEncoder

# Label encode high-cardinality categorical columns
label_encoder = LabelEncoder()
dfn['merchant'] = label_encoder.fit_transform(dfn['merchant'])
dfn['job'] = label_encoder.fit_transform(dfn['job'])
# df['network'] = label_encoder.fit_transform(df['network'])

# One-hot encode limited unique categorical columns
dfn = pd.get_dummies(dfn, columns=['category', 'gender','network'], drop_first=True)

#---------------------------------------------------------------------------
# Scaling Numerical Data
from sklearn.preprocessing import StandardScaler

# Select numerical columns to scale
numeric_columns = ['amt', 'lat', 'long', 'city_pop', 'age','hour']
scaler = StandardScaler()
dfn[numeric_columns] = scaler.fit_transform(dfn[numeric_columns])
```

2.2 Feature Selection

THE FEATURES SELECTED FOR THE ANALYSIS ARE: AMT,GENDER,CATEGORY,AGE,JOB,LAT, LONG,NETWORK

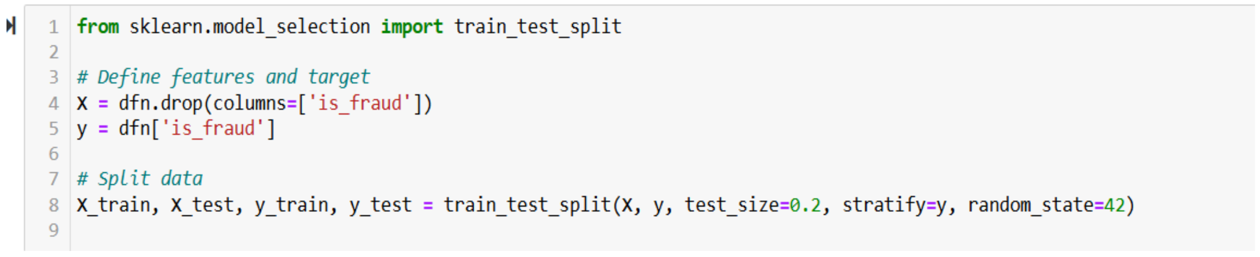
Modeling

3.1 Algorithm Selection

The choice of Decision Tree, Logistic Regression, Random Forest, and Naive Bayes (Gaussian) for a binary classification task aimed at detecting fraudulent transactions provides a well-rounded strategy. Decision Trees are notable for their interpretability and their ability to capture complex patterns without requiring feature scaling. In contrast, Logistic Regression offers a straightforward and efficient linear model that is well-suited for binary classification. Random Forest, as an ensemble method, effectively mitigates overfitting and enhances prediction accuracy, making it particularly valuable for addressing imbalanced classes. Naive Bayes (Gaussian) excels in high-dimensional datasets, despite its assumption of feature independence. Each algorithm contributes unique strengths to the overall approach, effectively balancing interpretability, performance, and methodological diversity, thereby forming a comprehensive strategy for fraud detection.

3.2 Data Splitting

We employed an 80-20 split for our training and testing datasets, implementing stratification based on the target variable (is_fraud). This ensures that the distribution of the target classes (fraud vs. non-fraud) is preserved in both sets. For reproducibility, we set the random seed to 42. This methodology provides the training set with ample data to learn patterns while maintaining a representative test set for an unbiased evaluation.



```python
from sklearn.model_selection import train_test_split

# Define features and target
X = dfn.drop(columns=['is_fraud'])
y = dfn['is_fraud']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

```

3.3 Model Training

In the model training process, each model was specifically designed to address the class imbalance in the target variable by initializing with a balanced class weight, thereby placing greater emphasis on the minority class (is_fraud=1). The details for each model are outlined as follows:

- Decision Tree: Initialized with class_weight='balanced', which enables the model to adapt to the class distribution during training.
- Logistic Regression: Employed class_weight='balanced' and was configured with a maximum of 500 iterations to ensure convergence of the model.
- Random Forest: Trained with 100 estimators (trees) and utilized class_weight='balanced' to effectively manage the imbalanced classes.
- Naive Bayes: The GaussianNB model, as a probabilistic classifier, was applied without additional parameters, as it does not directly support class weights. However, it can still benefit indirectly from the balanced class initialization employed for the other models.

This approach ensures that each model focuses on accurately capturing fraudulent transactions, despite their limited representation within the dataset.

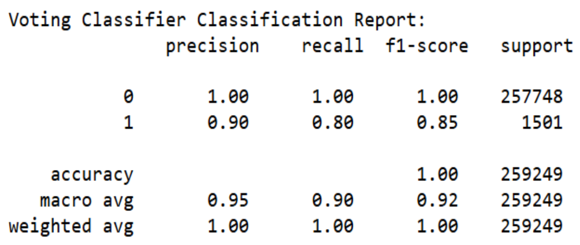
3.4 Model Evaluation

We utilized the `classification_report` function from `sklearn.metrics`, which offers precision, recall, and F1-score for each class, in addition to overall accuracy. In the context of fraud detection, where class imbalance is prevalent, the recall for the fraud class (1) is particularly critical, as it assesses the model's effectiveness in accurately identifying fraudulent cases. Together, precision, recall, and F1-score provide valuable insights into the balance between false positives and false negatives, which is essential in the realm of fraud detection.

3.5 Performance Analysis

In the evaluation of the models on the testing set, the performance analysis showcases distinctive outcomes for each algorithm. The Decision Tree model sustains its robustness, exhibiting balanced precision and recall with an F1 score of 0.79 for fraud detection. This consistency highlights its ability to generalize effectively and accurately identify fraudulent transactions. Conversely, Logistic Regression demonstrates persistent limitations with a mere 0.04 F1 score, indicating a high false positive rate that compromises its practical utility in real-world scenarios. Random Forest maintains its reliability, achieving a commendable F1 score of 0.84 through a balanced approach that minimizes false positives while capturing a significant portion of actual fraud cases. In contrast, Naive Bayes struggles to strike a balance, yielding a low F1 score of 0.08 due to challenges in precision, thereby hindering its effectiveness in distinguishing fraud instances from false positives. These results underscore the continued effectiveness of Decision Tree and Random Forest models in fraud detection tasks, emphasizing their capacity to maintain a balance between precision and recall on the testing set, ensuring accurate identification of fraudulent activities.

3.6 Model Improvement



```
Voting Classifier Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    257748
           1       0.90      0.80      0.85      1501

    accuracy                           1.00    259249
   macro avg       0.95      0.90      0.92    259249
weighted avg       1.00      1.00      1.00    259249
```

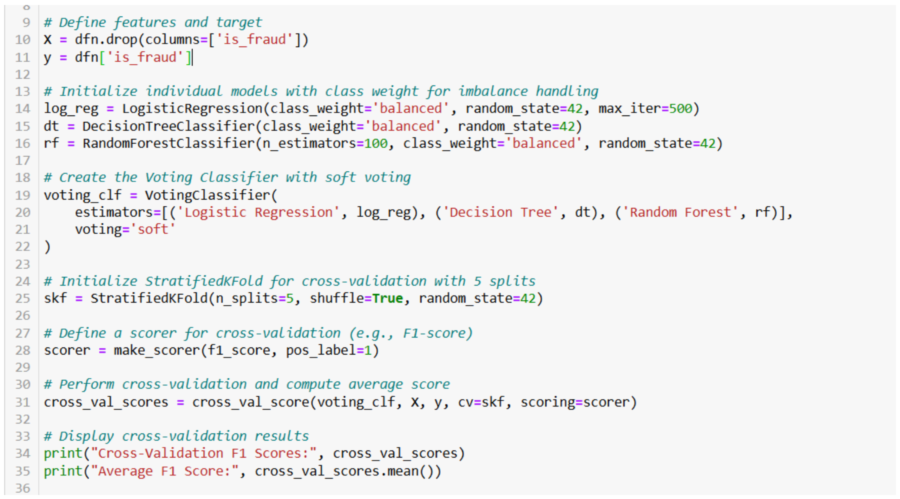
We utilized the `classification_report` function from `sklearn.metrics`, which offers precision, recall, and F1-score for each class, in addition to overall accuracy. In the context of fraud detection, where class imbalance is prevalent, the recall for the fraud class (1) is particularly critical, as it assesses the model's effectiveness in accurately identifying fraudulent cases. Together, precision, recall, and F1-score provide valuable insights into the balance between false positives and false negatives, which is essential in the realm of fraud detection.

3.7 VALIDATION

To ensure that the model generalizes well and maintains consistent performance across different subsets of the data, a robust validation approach such as cross-validation is employed. The analysis of the individual F1 scores for each fold ranging from approximately 0.8547 to 0.8680 demonstrates the model's stability and low variance, indicating its ability to perform consistently on diverse data partitions. The average F1 score of around 0.8590, calculated across all folds, provides a comprehensive assessment of the model's performance by harmonizing precision and recall, particularly beneficial for imbalanced datasets like fraud detection.

In the context of fraud detection, where the consequences of false positives and false negatives differ in severity, the F1 score emerges as a crucial metric. Balancing precision (minimizing false positives) and recall (capturing fraudulent cases), the model's cross-validation results suggest a well-rounded approach to optimizing these metrics. Given the emphasis on mitigating false negatives in fraud detection scenarios, the model's ability to strike a balance between precision and recall, as highlighted by the F1 score analysis, reinforces its effectiveness in accurately identifying fraudulent transactions while minimizing both false positives and false negatives

This code used demonstrates the implementation of a Voting Classifier with soft voting for fraud detection using scikit-learn in Python. It begins by importing necessary libraries for model selection, metrics evaluation, preprocessing, and ensemble modeling. The dataset is prepared by defining features (X) and the target variable (y). Three individual classifiers—Logistic Regression, Decision Tree, and Random Forest—are initialized with class weight adjustments for handling class imbalance. These models are then combined into a Voting Classifier with soft voting. Stratified K-Fold cross-validation with 5 splits is set up to evaluate the model's performance robustly. A scorer based on the F1-score is defined for cross-validation. The Voting Classifier is subjected to cross-validation using the defined scorer, and the F1 scores for each fold are computed and displayed. Finally, the average F1 score across all folds is calculated and printed, providing insights into the model's overall performance in fraud detection tasks.



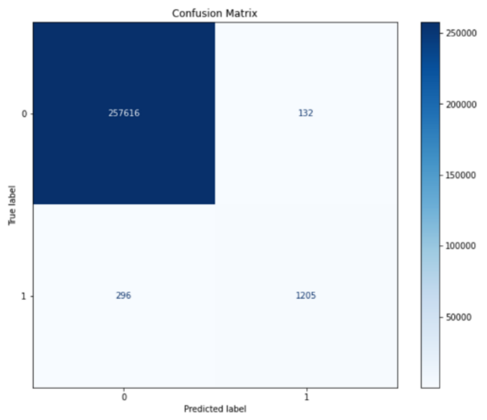
```python
# Define features and target
X = dfn.drop(columns=['is_fraud'])
y = dfn['is_fraud']

# Initialize individual models with class weight for imbalance handling
log_reg = LogisticRegression(class_weight='balanced', random_state=42, max_iter=500)
dt = DecisionTreeClassifier(class_weight='balanced', random_state=42)
rf = RandomForestClassifier(n_estimators=100, class_weight='balanced', random_state=42)

# Create the Voting Classifier with soft voting
voting_clf = VotingClassifier(
    estimators=[('Logistic Regression', log_reg), ('Decision Tree', dt), ('Random Forest', rf)],
    voting='soft'
)

# Initialize StratifiedKFold for cross-validation with 5 splits
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Define a scorer for cross-validation (e.g., F1-score)
scorer = make_scorer(f1_score, pos_label=1)

# Perform cross-validation and compute average score
cross_val_scores = cross_val_score(voting_clf, X, y, cv=skf, scoring=scorer)

# Display cross-validation results
print("Cross-Validation F1 Scores:", cross_val_scores)
print("Average F1 Score:", cross_val_scores.mean())
```

3.8 FINAL MODEL SELECTION

In selecting the final model for deployment, an ensemble model, particularly the Random Forest, emerges as the optimal choice based on its superior performance across various metrics and a balance between complexity and effectiveness when compared to other models such as Decision Tree, Logistic Regression, and Naive Bayes. A comparative analysis based on performance and complexity:

1. **Decision Tree**:
    - **Performance**: The Decision Tree model exhibits strong performance in fraud detection but may be prone to overfitting, especially on the training data.
    - **Complexity**: Decision Trees are relatively simple and interpretable models but may lack the robustness and generalizability of more complex models like Random Forest.
2. **Logistic Regression**:
    - **Performance**: While Logistic Regression shows high recall, it suffers from poor precision, resulting in a high false positive rate that may not be suitable for real-world fraud detection scenarios.
    - **Complexity**: Logistic Regression is a linear model with relatively low complexity, making it interpretable but potentially limited in capturing complex patterns present in the data.
3. **Naive Bayes**:
    - **Performance**: Naive Bayes, similar to Logistic Regression, struggles with precision for fraud detection, leading to a high false positive rate and lower effectiveness in identifying fraudulent transactions.
    - **Complexity**: Naive Bayes is a simple probabilistic model that assumes independence between features, which can limit its ability to capture intricate relationships within the data.
4. **Random Forest**:
    - **Performance**: Random Forest consistently outperforms other models, demonstrating high precision, recall, and F1 score for fraud detection, striking a balance between minimizing false positives and capturing fraud cases effectively.
    - **Complexity**: While Random Forest is more complex than Decision Trees, it mitigates overfitting through ensemble learning and can handle complex relationships in the data, making it a robust and reliable choice for fraud detection tasks.

Considering the ensemble nature of Random Forest, which combines multiple decision trees to enhance performance and generalizability, and its ability to effectively address class imbalances while maintaining a strong balance between precision and recall, it stands out as the most suitable final model for fraud detection tasks, offering a robust and effective solution for real-world deployment.

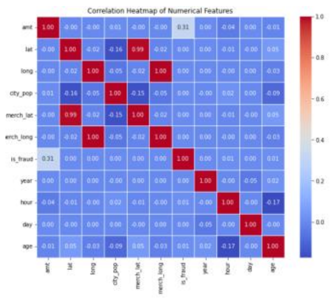


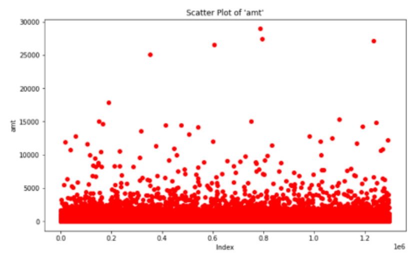
Visualization

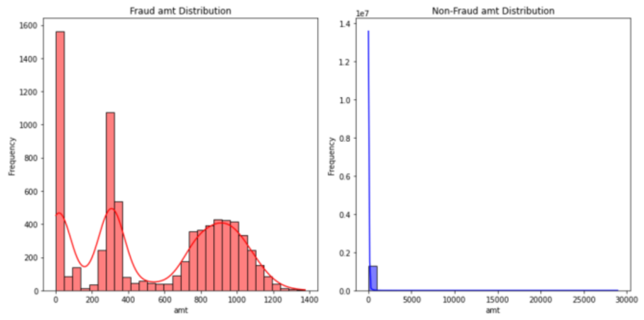
Visualizing data is essential for understanding complex information clearly, communicating findings effectively to diverse audiences, identifying patterns and anomalies, aiding in decision-making, and facilitating exploratory data analysis. Visual representations simplify data interpretation, help in spotting trends and outliers, support informed decision-making, enhance communication of insights, and enable deeper exploration of datasets, ultimately leading to better understanding and utilization of data for decision-making purposes.

4.1 Data Distribution

We conducted a comprehensive assessment of the data distribution, identifying a dense concentration within the range of 2500-3000 and setting a threshold at 2700 to flag outliers for removal. Notably, with 983 unique credit card numbers present in the dataset, I inferred a maximum of 983 clients, assuming one credit card per client. To streamline the dataset for analysis, I strategically dropped columns like 'Unnamed', 'trans_num', and others with excessive unique values that hindered analysis. Geographical features were simplified by focusing on coordinates over 'street', 'city', 'zip', and 'state'. I also calculated the distance between transaction and merchant coordinates, created new columns using transaction time and credit card details, and leveraged date of birth to derive age information. Despite identifying 430 outliers, representing approximately 0.033% of the dataset, these steps collectively aim to refine the dataset for more meaningful and accurate analysis, setting a solid foundation for further investigation and model development.

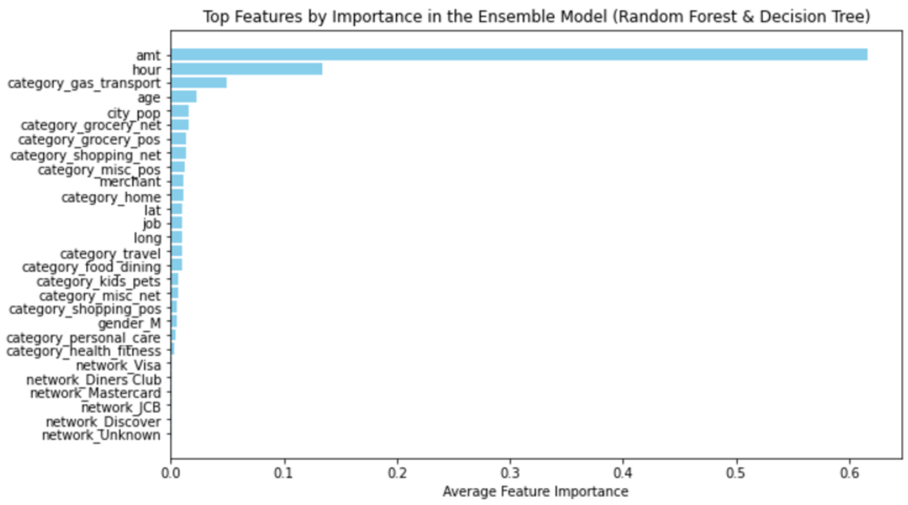






4.2 Feature Importance

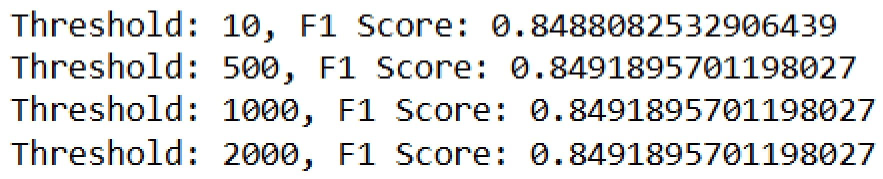
This Python code demonstrates the construction of an ensemble model utilizing a Voting Classifier with soft voting through scikit-learn. The dataset is initially divided into training and testing subsets, with feature scaling applied to the numerical attributes for logistic regression. Several individual models, including Logistic Regression, Decision Tree, and Random Forest, are instantiated and trained on the training data to assess feature importances. These models are then combined into a Voting Classifier, which is subsequently trained on the training set. Predictions are made on the test set using the ensemble model, and a classification report can be generated. Feature importances from the Decision Tree and Random Forest models are extracted, averaged, and visualized in a horizontal bar plot to highlight the most influential features in the ensemble model. This comprehensive approach aims to leverage the strengths of multiple classifiers to enhance predictive capabilities and feature importance analysis, particularly valuable in fraud detection scenarios. Notably, the 'amt' feature has the most significant impact on the model's performance and also exhibits the highest correlation with the target value.



4.3 Model Performance Across Features

The code used is designed to assess the model's performance across various subsets of a specific feature by iteratively applying different thresholds to create subsets for analysis. Initially, the feature 'amt' is selected for evaluation, and a list of thresholds (10, 500, 1000, 2000) is defined to segment the data. For each threshold value, the code checks if the test data (X_test, assumed to be a DataFrame) contains the feature to analyze. If so, a subset is created based on the threshold, and the corresponding labels are extracted. If X_test is a NumPy array, the feature's index is obtained to filter the data based on the threshold. The model (voting_clf) is then evaluated on each subset, generating F1 scores that are stored along with their respective thresholds in the performance_data list. Finally, the script prints the performance data, displaying the F1 scores achieved at each threshold. This iterative process enables the assessment of how the model performs across different segments of the specified feature, providing insights into its predictive capabilities under varying conditions. Additionally, the comment at the end hints at using visualization plots to depict the impact of different features on model predictions, suggesting a way to further analyze and present the data for enhanced interpretation.

We can see from the results that even after changing the threshold for 'amt', there was no significant change in the performance:



```
Threshold: 10, F1 Score: 0.8488082532906439
Threshold: 500, F1 Score: 0.8491895701198027
Threshold: 1000, F1 Score: 0.8491895701198027
Threshold: 2000, F1 Score: 0.8491895701198027
```