CREDIT CARD DETECTION

Phase 5

Problem Statement:

In the realm of data science, the challenge of credit card fraud detection presents a compelling problem. The exponential growth of digital transactions has created a vast pool of data, comprising intricate patterns of legitimate and fraudulent activities. Extracting meaningful insights from this colossal dataset to differentiate between genuine transactions and fraudulent ones has become a critical task.

The problem at the intersection of data science and credit card fraud detection involves leveraging advanced data analysis techniques, machine learning algorithms, and predictive modeling to identify subtle anomalies within the transaction data. These anomalies, often indicative of fraudulent activities, are crucial signals that can empower financial institutions to proactively prevent fraudulent transactions in real-time. Moreover, data scientists face the challenge of continually adapting their models to evolving fraud tactics, ensuring that the detection system remains effective against new and sophisticated methods employed by fraudsters.

To tackle this problem, data scientists must develop innovative algorithms that can sift through vast amounts of transaction data, identify patterns, and detect anomalies with a high degree of accuracy. Feature engineering, outlier detection, and ensemble learning techniques are just a few examples of the data science methodologies that can be applied to this problem. Additionally, the use of big data technologies and real-time processing frameworks is essential to handle the volume, velocity, and variety of transaction data generated daily.

By applying data science principles, financial institutions can build adaptive, intelligent fraud detection systems that not only pinpoint fraudulent transactions but also learn from new data, thereby staying one step ahead of fraudsters. This problem necessitates continuous exploration of novel data science approaches to enhance the efficiency, accuracy, and responsiveness of credit card fraud detection systems in the ever-changing landscape of digital transactions.

**Design Thinking:**

**1.Data Source:**

A "data source" in the context of credit card fraud detection refers to the origin or provider of the data used for training, testing, and validating fraud detection models. It is the place or entity from which you obtain the dataset containing information about credit card transactions, including both legitimate and fraudulent activities. A data source typically includes:

**A)Data Provider**: This is the organization, institution, or entity that collects, maintains, and shares the dataset. Data providers can include financial institutions, research organizations, government agencies, or third-party data providers.

**B)Data Description**: A data source should provide a detailed description of the dataset, including the type of information it contains. For credit card fraud detection, this would include features such as transaction amounts, timestamps, merchant information, card details, and, most importantly, labels or indicators of fraud/non-fraud for each transaction.

**C)Data Privacy and Security**: Information regarding how the data source handles data privacy and security is essential. It should adhere to regulations and best practices for protecting sensitive customer information, often through anonymization or data masking techniques.

**D)Data Accessibility**: The data source should specify how you can access the dataset. This may include downloading the dataset from a website, obtaining it through a data-sharing agreement, or accessing it through an API.

**E)Data Usage Policies**: The data source may have terms and conditions regarding the permissible uses of the dataset. Ensure you comply with these policies, which may include restrictions on data sharing, redistribution, or commercial use.

**F)Data Updates**: It's important to know if the dataset is regularly updated. In fraud detection, staying up-to-date with the latest transaction data is crucial for maintaining the accuracy of models.

**G)Data Quality and Integrity**: Information about data quality, including potential issues like missing values or outliers, should be documented. High-quality data is essential for building effective fraud detection models.

**H)Data Format**: The data source should specify the format in which the data is provided (e.g., CSV, JSON, SQL database). Understanding the data format is essential for data preprocessing.

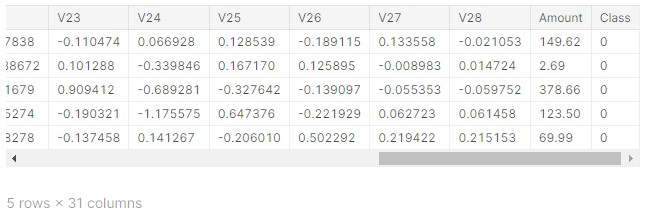
**I)Documentation:** Comprehensive documentation, including data dictionaries or metadata, is helpful for understanding the meaning and structure of each column in the dataset.

**J)Data Licensing**: If applicable, the data source may provide information on the licensing terms associated with the dataset. Some datasets have open licenses that allow broader usage, while others may have more restrictive licenses.

Obtaining data from a trustworthy and reputable source is critical in credit card fraud detection, as it impacts the reliability and ethics of your analysis. Additionally, it's important to handle sensitive financial data with care and ensure compliance with relevant data protection regulations, such as GDPR or HIPAA, depending on your jurisdiction and the nature of the data.

**Dataset Link:**[**https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud**](https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud)

The dataset contains transactions made by credit cards..



**2**.**Data Preprocessing**:

Clean and preprocess the data, handle missing values, and normalize features. Data preprocessing is a crucial step in credit card fraud detection as it helps prepare the raw transaction data for analysis and model training. Here are the key data preprocessing steps involved in credit card fraud detection:

**A)Data Loading:**Load the raw transaction data from your data source into your chosen data analysis tool or programming environment (e.g., Python with pandas).

**B)Data Exploration:**Perform initial data exploration to understand the dataset's structure, features, and data types.Check for missing values in the dataset and decide how to handle them (e.g., imputation or removal).Examine summary statistics to gain insights into transaction amounts, timestamps, and other relevant features.

**C)Data Cleaning:**Remove or handle duplicate records if they exist in the dataset.Address missing values by imputing them with appropriate values, such as the mean, median, or using advanced imputation methods like K-nearest neighbors or regression imputation.Correct any data format issues, such as inconsistent date/time formats.

**PYTHON PROGRAM:**

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import gridspec

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (classification\_report, accuracy\_score,

precision\_score, recall\_score,

f1\_score, matthews\_corrcoef,

confusion\_matrix)

credit = pd.read\_csv('/kaggle/input/creditcardfraud/creditcard.csv')

credit.head()

credit.shape

credit.describe().T

fraud = credit[credit['Class'] == 1]

valid = credit[credit['Class'] == 0]

fraction = len(fraud)/float(len(valid))

print(fraction)

print("Fraud Cases: {}".format(len(credit[credit['Class'] == 1])))

print("Valid Cases: {}".format(len(credit[credit['Class'] == 0])))

print("Amount of details for the Fraudulent Transaction")

fraud.Amount.describe()

print("Amount of details for Normal Transaction")

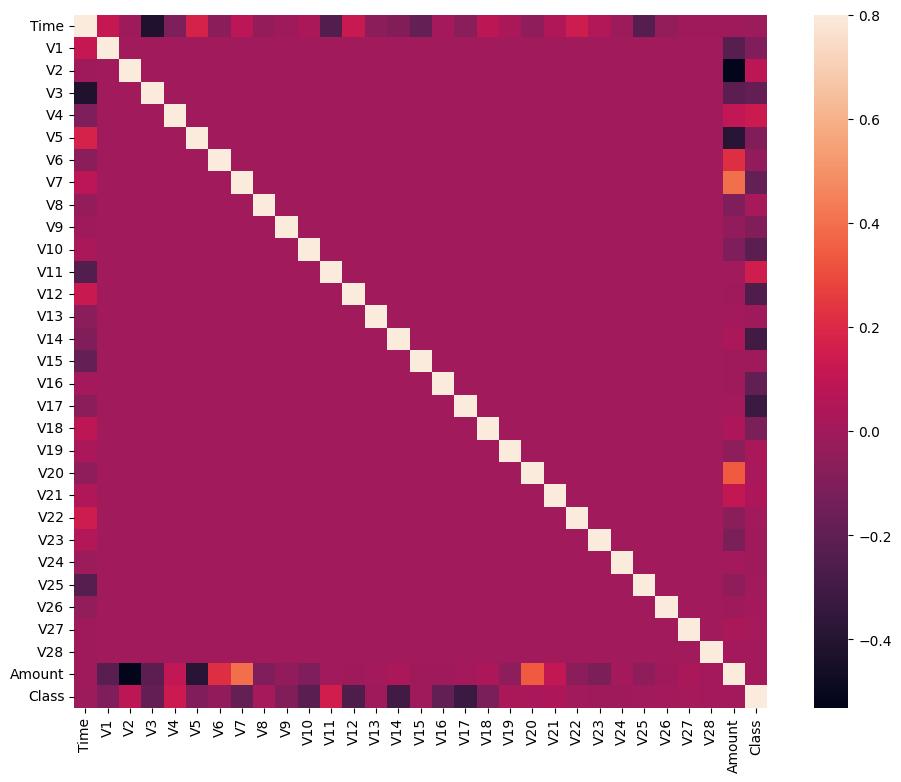
valid.Amount.describe()

corrmat = credit.corr()

fig = plt.figure(figsize=(12, 9))

sns.heatmap(corrmat, vmax=.8, square=True)

plt.show()



**3.Feature Engineering:**

Feature engineering is a critical step in credit card fraud detection that involves creating new features or transforming existing ones to improve the performance of machine learning models in identifying fraudulent transactions. Here are some feature engineering techniques commonly used in credit card fraud detection

**A)Time-Based Features:**Transaction Timestamp: Extract information such as the hour of the day, day of the week, or month from transaction timestamps. Fraudulent activities may exhibit temporal patterns.Time Since Last Transaction: Calculate the time elapsed since the cardholder's last transaction. Unusual gaps in transaction times could be indicative of fraud.

**B)Aggregated Statistics:**Transaction Amount Aggregates: Compute statistics like the mean, median, maximum, minimum, or standard deviation of transaction amounts for each cardholder or merchant. This helps identify abnormal spending patterns.

**C)Transaction Frequency:** Calculate the average time between transactions for each cardholder. Unusually high or low transaction frequencies may be indicative of fraud.

**D)Rolling Window Aggregates:** Create rolling window statistics, such as the sum of transaction amounts over the last N days, to capture short-term spending behavior.

**E)Merchant-Based Features:**Merchant Category: Categorize merchants into specific categories (e.g., retail, online, entertainment) and use these categories as features.

**F)Merchant Popularity:** Calculate the frequency of transactions with each merchant and flag less common merchants as potential anomalies.

**G)Cardholder-Based Features:**Cardholder Transaction History: Analyze each cardholder's historical transaction behavior, such as their spending patterns, average transaction amounts, and transaction frequency.Cardholder's Geographical Location: If available, use the cardholder's location data to detect unusual transaction locations.

**H)Transaction Velocity**:Velocity Filters: Calculate metrics like the number of transactions in a given time window and set thresholds to detect high-velocity transactions that may indicate fraud.

**I)Velocity Clusters:** Group transactions by cardholder and merchant and analyze patterns of multiple transactions within a short time frame.

**J)Cardholder Behavior Change:**Calculate metrics that capture changes in a cardholder's behavior over time. Sudden deviations from historical behavior may indicate fraud.

**H)Outlier Detection:**Use outlier detection techniques to identify unusual transactions based on statistical methods (e.g., Z-scores, Isolation Forests) or clustering algorithms.

Consider using anomaly scores as features in your model.

**I)Transaction Sequences:**Analyze sequences of transactions over time to identify patterns that are indicative of fraud. Sequence-based models like recurrent neural networks (RNNs) can be useful for this purpose.

**J)Encoding Categorical Variables**:Encode categorical features like cardholder ID, merchant ID, or merchant category using techniques like one-hot encoding or label encoding.

**K)Feature Scaling:**Ensure that all numerical features are on the same scale by applying scaling techniques like Min-Max scaling or standardization (z-score normalization).

**L)Derived Features:**Create composite features that combine relevant information. For example, a "transaction amount to average transaction amount ratio" could be informative.

Feature engineering is often an iterative process, and domain knowledge plays a crucial role in selecting and creating meaningful features. Experiment with different feature combinations and transformations to improve the performance of your credit card fraud detection model.

**4. Model Selection:**

Selecting the right model for credit card fraud detection is a crucial step in building an effective fraud detection system. The choice of model depends on various factors, including the characteristics of the dataset, the type of features engineered, and the trade-offs between different model properties. Here are some common models and considerations for selecting them in credit card fraud detection:

**A)Logistic Regression:**Use Case: Logistic regression is a simple and interpretable model suitable for binary classification tasks like fraud detection.

Pros: Interpretable, fast training and prediction, handles imbalanced datasets well with proper class weights.

Cons: Limited ability to capture complex relationships in data.

B)(Decision Trees and Random Forests:Use Case: Decision trees and random forests can capture non-linear relationships and interactions between features.

Pros: Can handle both numerical and categorical features, interpretable (for individual trees), and robust to outliers.

Cons: Random forests can be prone to overfitting if not tuned properly.

**C)Gradient Boosting Models (e.g., XGBoost, LightGBM, CatBoost):**Use Case: Gradient boosting models are powerful ensemble methods for improving classification accuracy.

Pros: High predictive accuracy, can handle complex relationships, and often robust to overfitting with proper regularization.

Cons: More complex than simpler models, may require hyperparameter tuning.

**D)Support Vector Machines (SVM):**Use Case: SVMs are effective when dealing with high-dimensional data and can handle non-linear relationships with the kernel trick.

Pros: Effective at capturing complex decision boundaries, good for handling imbalanced datasets.

Cons: Less interpretable, training time can be longer for large datasets.

**E)Neural Networks:**Use Case: Deep learning models like feedforward neural networks or convolutional neural networks (CNNs) can capture intricate patterns in data.

Pros: High capacity for feature learning, adaptability to various data types, and can handle large and complex datasets.

Cons: Requires a large amount of data and computational resources, can be challenging to interpret, and may overfit if not carefully regularized.

**F)Anomaly Detection Algorithms:**Use Case: Algorithms like Isolation Forest, One-Class SVM, or Autoencoders are specifically designed for anomaly detection tasks.

Pros: Tailored for fraud detection, effective at identifying rare events, and can be interpretable depending on the algorithm.

Cons: May not perform as well on non-anomalous data, may require specialized tuning.

**G)Ensemble Models:**Use Case: Combining multiple models (e.g., stacking or bagging) can often improve overall performance and robustness.

Pros: Improved accuracy, reduced overfitting, and better handling of class imbalance.

Cons: Increased complexity, may require more computational resources.

**H)Hybrid Approaches:**Consider using a combination of models, including rule-based systems and machine learning models, to leverage domain knowledge and statistical techniques for better fraud detection.

When selecting a model, it's essential to consider factors like the dataset size, class imbalance, interpretability, computational resources, and the trade-offs between false positives and false negatives. Additionally, hyperparameter tuning and model evaluation using appropriate metrics (e.g., precision, recall, F1-score, AUC-ROC) are essential to ensure the chosen model meets your fraud detection objectives.

Iterative experimentation and testing different models are often necessary to find the best-performing model for your specific credit card fraud detection problem.

**5. Model Training:**

Model training in credit card fraud detection involves using historical transaction data to teach a machine learning model to distinguish between legitimate and fraudulent transactions. Here's a step-by-step guide on how to train a model for this task:

**A)Data Preprocessing:**Begin by preprocessing your dataset as discussed earlier, which includes tasks like data cleaning, feature engineering, handling class imbalance, and data splitting.

**B)Select Evaluation Metrics:**Decide on the evaluation metrics you will use to assess the model's performance. Common metrics for fraud detection include precision, recall, F1-score, accuracy, and the area under the ROC curve (AUC-ROC).

**C)Data Splitting:**Split your preprocessed dataset into three subsets: a training set, a validation set, and a testing set. The training set is used for model training, the validation set for hyperparameter tuning, and the testing set for final model evaluation.

**D)Model Selection:**Choose an appropriate machine learning model based on your dataset characteristics and objectives. This could be logistic regression, decision trees, random forests, gradient boosting, support vector machines, neural networks, or an ensemble of multiple models.

**E)Hyperparameter Tuning:**If applicable, perform hyperparameter tuning using the validation set to optimize the model's performance. This involves adjusting model-specific parameters to find the best configuration.

**F)Model Training:**Train the selected model on the training dataset using the optimal hyperparameters. The model learns to distinguish between fraudulent and non-fraudulent transactions based on the features you've engineered.

**G)Model Evaluation:**Evaluate the model's performance on the validation set using the chosen evaluation metrics. Adjust hyperparameters as needed to improve the model's performance.

**H)Final Model Training:**Once satisfied with the model's performance on the validation set, train the final model on both the training and validation datasets to maximize the amount of data the model learns from.

**I)Model Testing:**Assess the final model's performance on the testing set, which contains unseen data. This step provides an unbiased estimate of how well the model will perform on new, real-world data.

**J)Threshold Selection:**Choose an appropriate decision threshold for classification. Depending on the trade-off between false positives and false negatives, you may adjust the threshold to meet specific fraud detection objectives.

**K)Monitoring and Maintenance:**Implement a system for continuous model monitoring and maintenance. As fraud patterns change over time, regularly retrain the model with new data to ensure its effectiveness.

**L)Deployment:**Once the model performs well on the testing set, deploy it into your production environment for real-time or batch processing of transactions. Ensure that deployment complies with all regulatory and security requirements.

**M)Feedback Loop:**Establish a feedback loop to continually improve the model. Monitor model performance in production, collect feedback on its predictions, and use this information to make model updates and refinements.

**N)Documentation and Reporting:**Maintain detailed documentation of the model's training process, hyperparameters, evaluation results, and any updates. This documentation is important for compliance and auditing purposes.

**O)Security and Privacy:**Implement robust security measures to protect both the model and the sensitive customer data used in fraud detection. Ensure that data privacy regulations are strictly adhered to.

Credit card fraud detection models may need periodic retraining and adjustment to stay effective, as fraudsters constantly adapt their tactics. Therefore, it's crucial to establish a well-defined process for model maintenance and improvement over time.

**PYTHON PROGRAM:**

X = credit.drop(['Class'], axis=1 )

Y = credit['Class']

print(X.shape)

print(Y.shape)

X\_credit = X.values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

# Build RandomForestClassifier

rfc = RandomForestClassifier()

rfc.fit(X\_train, y\_train)

y\_pred = rfc.predict(X\_test)

**6.Evaluation:**

Evaluating the performance of a credit card fraud detection model is a critical step to ensure that it effectively identifies fraudulent transactions while minimizing false alarms (false positives). Several evaluation metrics and techniques can be used to assess the model's performance. Here are some key aspects to consider when evaluating the model's performance in credit card fraud detection:

**A)Confusion Matrix:**A confusion matrix is a table that summarizes the model's predictions compared to the actual outcomes. It consists of four elements:

True Positives (TP): Legitimate transactions correctly classified as legitimate.

True Negatives (TN): Fraudulent transactions correctly classified as fraudulent.

False Positives (FP): Legitimate transactions incorrectly classified as fraudulent (Type I error).

False Negatives (FN): Fraudulent transactions incorrectly classified as legitimate (Type II error).

The confusion matrix is the foundation for calculating various evaluation metrics.

**B)Precision:**Precision measures the model's ability to correctly identify fraudulent transactions among all transactions it labels as fraudulent. It is calculated as TP / (TP + FP).

A higher precision indicates fewer false positives, which is important for reducing the number of legitimate transactions mistakenly flagged as fraud.

Recall (Sensitivity or True Positive Rate):

Recall measures the model's ability to correctly identify all actual fraudulent transactions. It is calculated as TP / (TP + FN).

A higher recall indicates that the model is better at catching fraudulent transactions, reducing false negatives.

**C)F1-Score:**The F1-score is the harmonic mean of precision and recall. It balances precision and recall, providing a single metric to assess overall model performance. It is calculated as 2 \* (Precision \* Recall) / (Precision + Recall).

The F1-score is particularly useful when dealing with imbalanced datasets, as it accounts for both false positives and false negatives.

**D)Accuracy:**Accuracy measures the proportion of correctly classified transactions (both legitimate and fraudulent) out of the total transactions. It is calculated as (TP + TN) / (TP + TN + FP + FN).

While accuracy is an important metric, it may not be suitable for imbalanced datasets, as high accuracy can be achieved by simply classifying all transactions as legitimate.

**E)Receiver Operating Characteristic (ROC) Curve and Area Under the Curve (AUC-ROC):**The ROC curve plots the trade-off between true positive rate (TPR or recall) and false positive rate (FPR) for different decision thresholds.

AUC-ROC quantifies the overall performance of the model. A higher AUC-ROC indicates better discrimination between legitimate and fraudulent transactions.

**F)Precision-Recall Curve and Area Under the Curve (AUC-PR):**The precision-recall curve illustrates the trade-off between precision and recall for different decision thresholds.

AUC-PR quantifies the model's performance in terms of precision and recall, focusing on the positive class (fraudulent transactions).

**G)Specificity and True Negative Rate (TNR):**Specificity measures the model's ability to correctly identify legitimate transactions as legitimate. It is calculated as TN / (TN + FP).

**H)False Positive Rate (FPR):**FPR measures the proportion of legitimate transactions incorrectly classified as fraudulent. It is calculated as 1 - Specificity.

**I)Cost-Based Metrics:**Consider cost-sensitive evaluation metrics that account for the financial impact of false positives and false negatives, as well as the operational costs of investigating alerts.

**J)Threshold Selection:**Choose an appropriate decision threshold that balances precision and recall based on the specific needs and objectives of the fraud detection system. Adjusting the threshold can help tailor the model's behavior.

**K)Cross-Validation:**Perform cross-validation, such as k-fold cross-validation, to assess the model's robustness and generalization performance. Cross-validation helps mitigate overfitting and provides more reliable performance estimates.

**L)Model Interpretability:**Consider the interpretability of the model, especially when explaining its predictions to stakeholders or regulatory authorities.

**M)Monitoring and Feedback:**Continuously monitor the model's performance in a production environment and gather feedback from fraud analysts and investigators to make necessary adjustments and improvements.

Evaluating the model's performance in credit card fraud detection is an ongoing process, as fraud patterns evolve over time. It's important to regularly re-evaluate and update the model to maintain its effectiveness. Additionally, it's essential to document the evaluation results for compliance and reporting purposes.

**PYTHON PROGRAM:**

n\_outliers = len(fraud)

n\_errors = (y\_pred != y\_test).sum()

print("The model used is RandomForestClassifier")

acc = accuracy\_score(y\_test, y\_pred)

print(f"The accuracy is {acc}")

prec = precision\_score(y\_test, y\_pred)

print(f"The precision score is {prec}")

rec = recall\_score(y\_test, y\_pred)

print(f"The recall score is {rec}")

f1 = f1\_score(y\_test, y\_pred)

print(f"The f1 score is {f1}")

MCC = matthews\_corrcoef(y\_test, y\_pred)

print(f"The Matthews correlation coeficient is {MCC}")

LABELS = ['Normal', 'Fraud']

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(12,12))

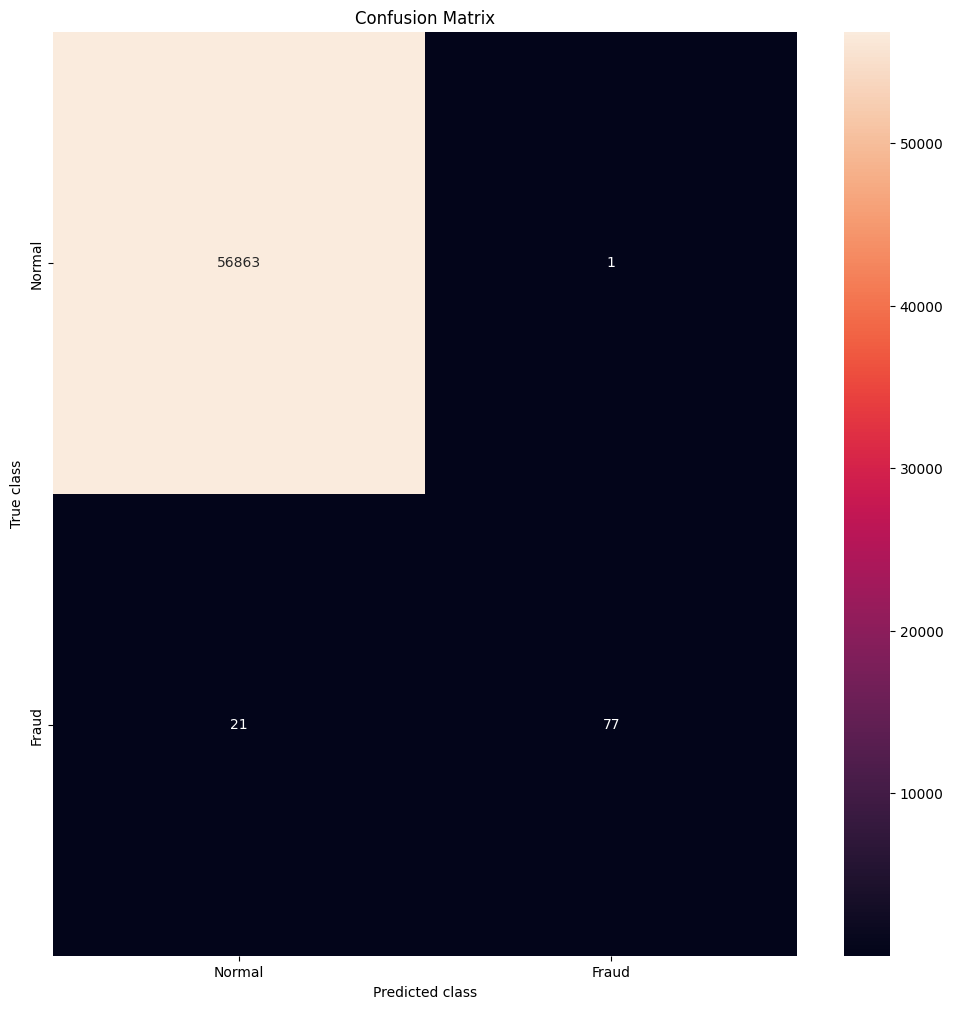
sns.heatmap(conf\_matrix, xticklabels = LABELS, yticklabels = LABELS, annot=True, fmt='d')

plt.title('Confusion Matrix')

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()



Development Phase1:

Designing innovation into the solution for fraud detection can greatly enhance the accuracy and efficiency of the system. Here's a step-by-step approach to incorporate advanced techniques like anomaly detection algorithms and ensemble methods into an innovative fraud detection system:

**1. Data Gathering and Preprocessing:**

Collect and aggregate data from various sources, including transaction records, user behavior logs, and external data feeds.

Preprocess the data to handle missing values, outliers, and ensure data consistency.

**2. Feature Engineering:**

Create relevant features that capture information about user behavior, transaction patterns, and other data that may be indicative of fraud.

Feature scaling, transformation, and dimensionality reduction techniques can be applied.

**3. Anomaly Detection:**

Implement advanced anomaly detection algorithms like Isolation Forest and One-Class SVM. You can even create an ensemble of these algorithms.

Train these models on historical data to identify anomalies in the dataset.

**4. Ensemble Methods:**

Use ensemble methods like Random Forest or Gradient Boosting in combination with the anomaly detection algorithms.

The ensemble can combine the outputs of different models, allowing for more robust fraud detection.

**5. Continuous Learning:**

Implement a continuous learning system that adapts to changing fraud patterns. Use techniques like online learning to update models in real-time.

**6. Model Interpretability:**

Develop methods to explain the decisions made by your models, which can help in understanding why a particular transaction is flagged as fraudulent.

**7. Real-time Processing:**

Implement real-time or near-real-time processing of transactions. As soon as a transaction occurs, it should be evaluated for fraud.

Use stream processing frameworks like Apache Kafka or Apache Flink to handle the data in real-time.

**8. Alerting and Response:**

Design a system to trigger alerts or responses when potential fraud is detected. This can include blocking a transaction, sending notifications, or routing it for manual review.

**9. Human-in-the-Loop:**

Incorporate human-in-the-loop systems for complex cases where automated decisions might be uncertain. Provide an interface for investigators to review and validate potential fraud cases.

**10. Model Monitoring:**

Regularly monitor the performance of your models to ensure they remain accurate and up to date.

Set up alerts for model degradation or shifts in fraud patterns.

**11. Compliance and Privacy:**

Ensure your system complies with data privacy regulations and industry standards. Implement features to handle sensitive data responsibly.

**12. Feedback Loop:**

Create a feedback loop where the results of investigations and manual reviews are used to retrain the models and improve the system over time.

**13. Experimentation and Innovation:**

Encourage a culture of experimentation and innovation within your team. Explore cutting-edge techniques in machine learning and fraud detection to stay ahead of evolving fraud tactics.

**14. Collaboration and Knowledge Sharing:**

Foster collaboration with experts in the field of fraud detection and data science. Share knowledge and insights to continually improve the system.

Incorporating these elements into your fraud detection system can lead to a highly innovative and effective solution that adapts to changing fraud patterns while maintaining high accuracy and compliance with regulations. Continuous learning and monitoring are key to staying at the forefront of fraud detection technology.

Development Phase2:

Loading and preprocessing a dataset for credit card fraud detection typically involves handling imbalanced classes, scaling features, and potentially reducing dimensionality for model training. Below are the steps to load and preprocess a credit card fraud detection dataset using Python. We'll use the common credit card fraud dataset available on Kaggle as an example.

**1. Import Libraries:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from imblearn.over\_sampling import SMOTE

**2. Load the Dataset:**

# Load your credit card fraud dataset (replace with your actual dataset path)

df = pd.read\_csv("credit\_card\_fraud\_dataset.csv")

**3. Explore the Dataset:**

Take a quick look at the dataset to understand its structure:

print(df.head())

print(df.info())

print(df['Class'].value\_counts())

**4. Split Data into Features and Target:**

X = df.drop('Class', axis=1)

y = df['Class']

**5. Handle Imbalanced Classes:**

Credit card fraud datasets are often highly imbalanced, with a small number of fraudulent transactions. You can oversample the minority class using techniques like SMOTE (Synthetic Minority Over-sampling Technique):

smote = SMOTE(sampling\_strategy=0.5) # Adjust the sampling\_strategy as needed

X\_resampled, y\_resampled = smote.fit\_resample(X, y)

**6. Split Data into Training and Testing Sets:**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_resampled, y\_resampled, test\_size=0.2, random\_state=42)

**7. Feature Scaling:**

Scale the features to have zero mean and unit variance, which is often necessary for machine learning algorithms:

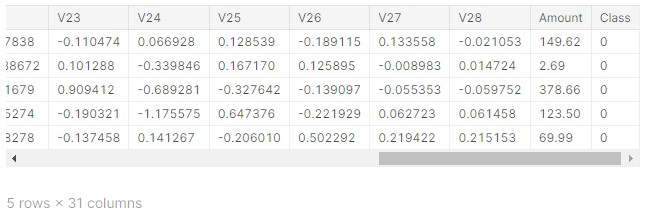
scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

Now you have your dataset loaded, balanced, and preprocessed. You can proceed to build and train a machine learning model for credit card fraud detection using the X\_train and y\_train data. Keep in mind that the choice of model (e.g., logistic regression, random forest, XGBoost) and hyperparameter tuning is a significant part of this process. Additionally, you may want to perform dimensionality reduction (e.g., PCA) or other feature engineering based on your specific dataset and goals.

**Dataset Link:**[**https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud**](https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud)



**Importing Imortant Libraries:**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import PowerTransformer

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import classification\_report

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

import xgboost as xgb

Clean and preprocess the data, handle missing values, and normalize features. Data preprocessing is a crucial step in credit card fraud detection as it helps prepare the raw transaction data for analysis and model training. Here are the key data preprocessing steps involved in credit card fraud detection:

**A)Data Loading:**

Load the raw transaction data from your data source into your chosen data analysis tool or programming environment (e.g., Python with pandas).

**B)Data Exploration:**

Perform initial data exploration to understand the dataset's structure, features, and data types.Check for missing values in the dataset and decide how to handle them (e.g., imputation or removal).Examine summary statistics to gain insights into transaction amounts, timestamps, and other relevant features.

**C)Data Cleaning:**

Remove or handle duplicate records if they exist in the dataset.Address missing values by imputing them with appropriate values, such as the mean, median, or using advanced imputation methods like K-nearest neighbors or regression imputation.Correct any data format issues, such as inconsistent date/time formats.

**PYTHON PROGRAM:**

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import gridspec

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (classification\_report, accuracy\_score,

precision\_score, recall\_score,

f1\_score, matthews\_corrcoef,

confusion\_matrix)

credit=pd.read\_csv('/kaggle/input/creditcardfraud/creditcard.csv')

credit.head()

credit.shape

credit.describe().T

fraud = credit[credit['Class'] == 1]

valid = credit[credit['Class'] == 0]

fraction = len(fraud)/float(len(valid))

print(fraction)

print("Fraud Cases: {}".format(len(credit[credit['Class'] == 1])))

print("Valid Cases: {}".format(len(credit[credit['Class'] == 0])))

print("Amount of details for the Fraudulent Transaction")

fraud.Amount.describe()

print("Amount of details for Normal Transaction")

valid.Amount.describe()

corrmat = credit.corr()

fig = plt.figure(figsize=(12, 9))

sns.heatmap(corrmat, vmax=.8, square=True)

plt.show()

Development Phase3:

**Feature Engineering:**

Feature engineering is a crucial step in building a credit card fraud detection model. It involves selecting, creating, or transforming features to make them more informative for the model.

**Feature Selection:**

Start by selecting relevant features that are likely to be indicative of fraudulent transactions. Features such as transaction amount, transaction time, and potentially merchant information can be significant.

**Feature Transformation:**

Perform feature scaling to ensure all numerical features have a similar scale. Common techniques include Standardization or Min-Max scaling.

Use one-hot encoding for categorical features, like merchant category, if necessary.

**Feature Creation:**

You can create new features from existing ones. For example, create a "day of the week" feature from the transaction timestamp, which might help capture weekly patterns in fraud.

**Dimensionality Reduction (Optional):**

If your dataset has a large number of features, consider techniques like Principal Component Analysis (PCA) to reduce dimensionality while preserving important information.

**Model Training:**

After feature engineering, it's time to train your credit card fraud detection model. Commonly used algorithms for this task include Logistic Regression, Random Forest, and Gradient Boosting. You can start with a simple model and then explore more complex ones if needed.

**Data Splitting:**

Split your dataset into training, validation, and test sets. A common split is 70% for training, 15% for validation, and 15% for testing.

**Model Selection:**

Train multiple models on the training data. For example, you can train a Logistic Regression model and a Random Forest model.

**Hyperparameter Tuning:**

Use the validation set to fine-tune hyperparameters for each model. Grid search or random search can be helpful for this purpose.

**Ensemble Learning (Optional):**

Consider creating an ensemble of models to improve predictive performance. For instance, you can use a Voting Classifier to combine the predictions from multiple models.

**Model Evaluation on Validation Set:**

Use evaluation metrics like accuracy, precision, recall, F1-score, and ROC AUC to assess the models' performance on the validation set.

**PYTHON PROGRAM:**

X = credit.drop(['Class'], axis=1 )

Y = credit['Class']

print(X.shape)

print(Y.shape)

X\_credit = X.values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

# Build RandomForestClassifier

rfc = RandomForestClassifier()

rfc.fit(X\_train, y\_train)

y\_pred = rfc.predict(X\_test)

**Evaluation:**

Evaluate the model's performance on a separate test set to assess how well it generalizes to new, unseen data.

**Model Evaluation on Test Set:**

Calculate the same evaluation metrics (accuracy, precision, recall, F1-score, ROC AUC) on the test set to get a true measure of how well the model performs.

**Confusion Matrix:**

Analyze the confusion matrix to understand false positives and false negatives. This is important for fraud detection as it helps you balance fraud prevention and minimizing false alarms.

**Threshold Adjustment:**

You can adjust the classification threshold based on your specific business needs to balance between false positives and false negatives. This will depend on the cost of fraud and the cost of mistakenly blocking legitimate transactions.

**Monitoring and Maintenance:**

Set up continuous monitoring for model performance, as fraud patterns can change over time. Retrain your model periodically to keep it effective.

Remember that credit card fraud detection is a delicate task, and you may need to iteratively improve your model and fine-tune it based on new data and evolving fraud patterns.

**PYTHON PROGRAM:**

n\_outliers = len(fraud)

n\_errors = (y\_pred != y\_test).sum()

print("The model used is RandomForestClassifier")

acc = accuracy\_score(y\_test, y\_pred)

print(f"The accuracy is {acc}")

prec = precision\_score(y\_test, y\_pred)

print(f"The precision score is {prec}")

rec = recall\_score(y\_test, y\_pred)

print(f"The recall score is {rec}")

f1 = f1\_score(y\_test, y\_pred)

print(f"The f1 score is {f1}")

MCC = matthews\_corrcoef(y\_test, y\_pred)

print(f"The Matthews correlation coeficient is {MCC}")

LABELS = ['Normal', 'Fraud']

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(12,12))

sns.heatmap(conf\_matrix, xticklabels = LABELS, yticklabels = LABELS, annot=True, fmt='d')

plt.title('Confusion Matrix')

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()

