**Credit Card Fraud Detection**

**Phase 4: Development Part 2**

Certainly! Continuing from where we left off, let's proceed with feature engineering, model training, and evaluation for a credit card fraud detection project.

**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()

