**Title:** CodTech IT Solutions Internship - Task Documentation: Credit Card Fraud Detection, Titanic Survival Prediction using Python

**INTERN INFORMATION:**

**Name:** Panuganti Karthik

**ID:** ICOD5620

**Project Title: Credit Card Fraud Detection**

**Objective:** "The objective of this project is to develop a robust machine learning model capable of accurately identifying fraudulent credit card transactions, minimizing financial losses for institutions and cardholders."

**Problem Background:** Credit card fraud is a pervasive global problem, affecting financial institutions, merchants, and consumers alike. Fraudulent transactions result in billions of dollars in losses annually for financial institutions and cardholders. The types of fraud include counterfeit cards, card-not-present fraud (online transactions), application fraud, and stolen card fraud. In addition to financial losses, credit card fraud undermines trust in the payment system and can lead to reputational damage for financial institutions. Early detection of fraudulent transactions is crucial for minimizing losses, protecting consumers, and maintaining the integrity of the financial system.

**Approach Overview:**

* Summarize the steps involved:
  + Data acquisition and preprocessing
  + Exploratory data analysis
  + Feature engineering
  + Model selection and training
  + Performance evaluation
  + Model deployment

**Dependencies:**

* **Python**: Python 3.12
* **Libraries**:
  + NumPy: "For numerical computations and array manipulation."
  + Pandas: "For data loading, manipulation, and analysis."
  + Matplotlib and Seaborn: "For creating informative data visualizations."
  + Scikit-learn: "For building and evaluating machine learning models."
  + **Dataset:**
  + Source: Kaggle (https://www.kaggle.com/mlg-ulb/creditcardfraud)
  + Size: (28481, 31)
  + Features: List key features, noting the 'Class' target variable

**Implementation**

1. **Data Preprocessing and Sampling**
   * Loading the Dataset: The code uses pd.read\_csv("./Dataset/creditcard.csv") to load the credit card fraud dataset from a CSV file.
   * Sampling: A random 10% sample of the dataset is selected using data.sample(frac=0.1, random\_state=48). This likely aims to reduce computational costs and speed up processing. The random\_state ensures the reproducibility of results.
2. **Exploratory Data Analysis (EDA)**
   * Data Understanding: data. shape and data.describe() provides insights into the dataset's dimensions (rows, columns) and summary statistics of numerical features.
   * Visualization: Histograms (data.hist(figsize=(20,20))) reveal the distributions of individual features, helping identify potential outliers, skewness, or other patterns.
   * Class Imbalance Investigation: The code calculates the number of fraudulent vs. valid transactions, highlighting the class imbalance issue common in fraud datasets.
3. **Feature Preparation**
   * Defining Features and Target: The code separates the features (X) from the target variable ('Class') to prepare data for model training.
   * Data Splitting: Using train\_test\_split, the dataset is divided into training (80% in this case) and testing sets (20%) for model development and evaluation.
4. **Outlier-Based Fraud Detection**
   * Model Instantiation: Instances of Isolation Forest and Local Outlier Factor algorithms are created, setting parameters like contamination (expected proportion of outliers) and n\_neighbors (for LOF).
   * Model Fitting and Prediction: Each model is fitted on the training data (clf.fit(X)), and predictions are made on the features (clf.predict(X) or clf.fit\_predict(X)).
5. **Performance Evaluation**
   * Metrics: A comprehensive set of metrics is calculated: accuracy, precision, recall, F1-score, and Matthews Correlation Coefficient (MCC). These metrics, especially precision, recall, and F1-score, are highly relevant for evaluating fraud detection models where the focus is often on minimizing false negatives and false positives.

**Code Explanation**

1. **Data Loading and Sampling**
   * Loading: The line data = pd.read\_csv("./Dataset/creditcard.csv") reads the credit card fraud dataset from a CSV file located in the specified path. The Pandas library is used for data manipulation.
   * Sampling: data = data.sample(frac=0.1, random\_state=48) selects a random 10% subset of the dataset. Explain that this is likely done to reduce computational time and potentially improve model generalizability. The random\_state is fixed to ensure results are reproducible when rerunning the code.
2. **Exploratory Data Analysis (EDA)**

* **Initial Overview**:
* **data.shape** reveals the dimensions of the DataFrame (number of rows and columns).
* **data.describe()** provides summary statistics (count, mean, standard deviation, min, max, etc.) for the numerical features.
* **Histograms**: **data.hist(figsize = (20, 20))** generates a grid of histograms, one for each feature. Explain that this helps visualize:
* **Distributions**: Whether features are normally distributed, skewed, etc.
* **Outliers**: Potential anomalies in the feature values.
* **Class Imbalance**: The code explicitly calculates and prints the number of fraudulent versus valid transactions. This highlights the challenge of class imbalance that is typical in fraud detection.

1. **Data Preparation**

* **Separating Features and Target:**
* X = data[columns] isolates the feature columns into 'X'.
* Y = data[target] extracts the 'Class' column (the target variable containing fraud labels) into 'Y'.
* **Train-Test Split:** xTrain, xTest, yTrain, yTest = train\_test\_split(xData, yData, test\_size=0.2, random\_state=42) divides the data into training (80%) and testing (20%) sets. The random\_state ensures consistent splits for reproducibility.

1. **Outlier Detection Models**

* **Isolation Forest Instantiation:** The code creates an Isolation Forest model with:
* max\_samples=len(X): Setting the subsampling size to match the training data.
* contamination=outlierFraction: The expected proportion of outliers in the data.
* **LOF Instantiation:** Similarly, the code creates a Local Outlier Factor model with:
* n\_neighbors=20: Parameter to determine the neighborhood for outlier calculation.
* **Fitting and Predicting:**
* clf.fit(X) trains each model on the feature data 'X'.
* Predictions: The outcome depends on the algorithm type (clf.predict(X) or clf.fit\_predict(X)).

1. **Evaluation**

* **Metrics:** The code calculates a suite of metrics:
* **Accuracy:** Overall correct predictions.
* **Precision:** Focuses on minimizing false positives (important for fraud).
* **Recall:** Focuses on minimizing false negatives (important for fraud).
* **F1-score:** Harmonic mean of precision and recall.
* **MCC:** Robust metric, especially for imbalanced classes.

**Usage**

This code provides a framework for building and evaluating outlier-based fraud detection models for credit card transactions. Here's how a refined version of this implementation could be integrated into a real-world fraud detection system:

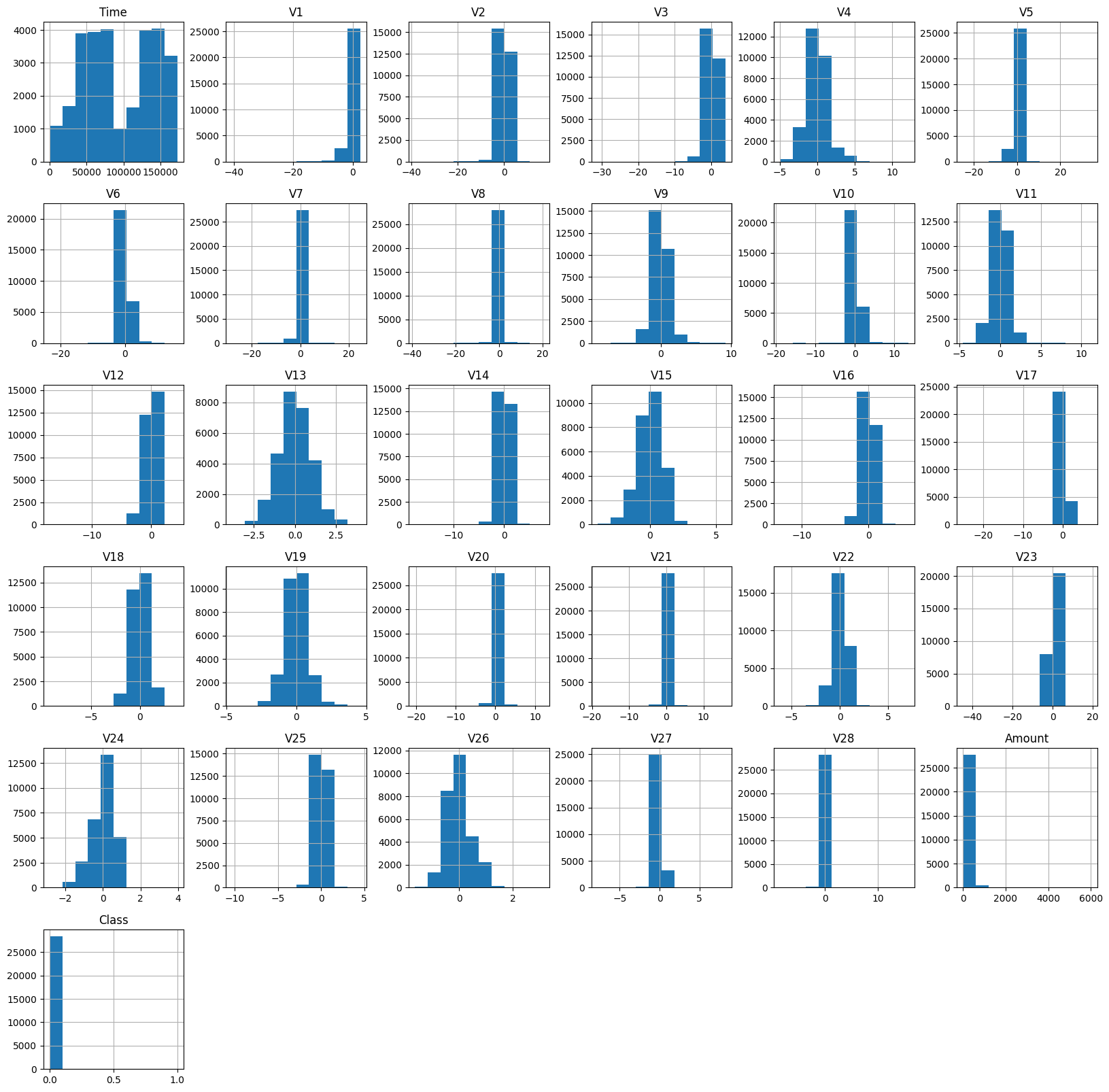
* **Deployment:**
  + The selected model (Isolation Forest or LOF based on performance) would be trained on a larger, more representative dataset of credit card transactions.
  + The trained model would be integrated into a real-time transaction processing pipeline.
* **New Transactions:**
  + For each new transaction, the features (V1, V2, etc.) would be extracted.
  + The model would generate an outlier score or prediction label (fraud/non-fraud).
* **Thresholds and Actions:**
  + Decisions on how to act upon model predictions would be based on a combination of the model's output, defined risk thresholds, and business rules. Actions could include:
    - **Automated Blocking:** High-confidence fraud predictions might trigger an automatic transaction block.
    - **Flagging for Review:** Transactions with moderate outlier scores could be flagged for manual review by analysts.
    - \*\*Further Verification: \*\* The system might request additional verification from the cardholder.

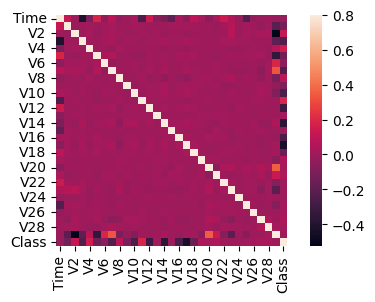
**Key Considerations:**

* **Continuous Monitoring:** The model's performance should be regularly monitored for potential degradation as fraud patterns evolve. Consider retraining the model periodically.
* **False Positives:** Investigate ways to reduce false positives, which can negatively impact customer experience.
* **Explainability:** If explainability is important, consider techniques like LIME or SHAP to understand the reasoning behind model predictions.
* **System Integration** The fraud detection model is just one component of a larger fraud prevention system with rules-based engines and likely human review processes.

**CONCLUSION**

Model Performance: The Isolation Forest model significantly outperformed the Local Outlier Factor model in this experiment. It achieved an accuracy of 0.9978, a precision of 0.456, a recall of 0.464, and an F1-score of 0.460. The Local Outlier Factor model, on the other hand, had poor precision and recall, indicating issues in correctly detecting fraudulent transactions.

**OUTPUT**

****

