CREDIT CARD FRAUD DETECTION

PHASE - 5: DOCUMENTATION

STEP 1:-

Problem Definition:

The problem at hand is to develop a credit card fraud detection system that can accurately and efficiently identify fraudulent transactions in real-time, thereby minimizing financial losses for both credit card companies and cardholders. Credit card fraud is a significant and growing concern, with criminals constantly evolving their tactics. To address this problem, we aim to create a robust and adaptive fraud detection system that can distinguish between legitimate and fraudulent transactions, while also minimizing false positives that could inconvenience genuine cardholders.

STEP 2 :-

Design Thinking process:

Data Collection:

To collect dataset containing transaction data, including features such as transaction amount, merchant information, and card details and defining the scope of data collection, including what types of transaction data are relevant and which time periods to consider.

Data Preprocessing:

To clean and identify potential issues like missing values, outliers, and data format discrepancies and Set data preprocessing goals, such as data cleansing, normalization, and handling missing values, to ensure the data is suitable for analysis.

Feature Engineering:

To create additional features that could enhance fraud detection, such as Transaction frequency, Amount deviations and Cardholder Profile.

Model Selection:

To Choose a suitable machine learning algorithms including logistic regression, decision trees, random forests, support vector machines, and deep learning architectures for fraud detection. Implement multiple model prototypes with different algorithms to assess their performance.

Model Training:

To train the selected model using pre-processed data and employ cross-validation techniques to assess model performance and prevent overfitting. Since fraud detection demands high precision, consider ensemble methods and anomaly detection algorithms.

Model Evaluation:

To Understand the consequences of false positives and false negatives in credit card fraud detection by using evaluation metrics, such as accuracy, precision, recall, F1-score, and ROC AUC, to assess model performance.

STEP 3 :-

PHASES OF DEVELOPMENT

1. **Implementation Plan:**

   To implement an effective credit card fraud detection system, start by acquiring a diverse and representative dataset of credit card transactions, ensuring a balance between legitimate and fraudulent cases. Next, preprocess the data, handling missing values, normalizing features, and addressing class imbalance. Choose suitable machine learning algorithms and hyperparameters, and split the data into training and testing sets for model development. Implement robust model deployment with real-time monitoring and alerts. Continuously update the model with new data and retrain it to adapt to evolving fraud patterns.

2. Data Collection:

The data collection in credit card fraud detection, start by obtaining a diverse and representative dataset of credit card transactions. Include both legitimate and fraudulent transactions, ensuring a balance between the two. Collect relevant features such as transaction amount, location, time, and any other pertinent details. Ensure compliance with data privacy regulations and anonymize sensitive information to protect user privacy. Continuously update the dataset to account for evolving fraud patterns, ensuring the system remains effective over time.

3. Data Preprocessing:

In the data preprocessing stage of credit card fraud detection, the collected dataset must undergo several crucial steps to prepare it for analysis. Initially, handle missing or erroneous values through imputation or removal. Normalize numerical features to a common scale to prevent bias in model training, and one-hot encode categorical variables. To address class imbalance, apply techniques like oversampling fraudulent transactions or under sampling legitimate ones. Feature selection or dimensionality reduction methods can enhance model efficiency while preserving predictive power. Furthermore, implement data splitting for training and testing to evaluate model performance accurately. Lastly, maintain data privacy by securely storing and handling sensitive information throughout this process.

4. Feature engineering:

Feature engineering is crucial in credit card fraud detection to enhance the model's ability to identify fraudulent transactions accurately. Begin by creating new features, such as transaction frequency, velocity, and historical spending patterns, which can capture subtle fraud indicators. Utilize geographical information to identify transactions occurring in unexpected locations. Consider aggregating transaction history, like rolling averages and standard deviations, to reveal anomalies. Feature scaling and normalization ensure consistent model performance. Continuous refinement and monitoring of engineered features are essential to adapt to evolving fraud techniques and maintain the system's effectiveness.

5. Model training:

In credit card fraud detection, selecting an appropriate model and training it effectively are pivotal. Begin by experimenting with various algorithms like logistic regression, decision trees, random forests, and neural networks to find the best-suited model for the dataset. Employ

cross-validation techniques to assess model performance and prevent overfitting. Since fraud detection demands high precision, consider ensemble methods and anomaly detection algorithms. Fine-tune hyperparameters to optimize model accuracy, sensitivity, and specificity. Continuously update the model with new data to adapt to emerging fraud patterns. Lastly, implement real-time or batch processing, depending on system requirements, to ensure swift and accurate fraud detection while minimizing false positives.

6. Hyperparameter tuning:

Hyperparameter tuning is a critical phase in credit card fraud detection model development. It involves systematically adjusting the configuration settings that aren't learned during training to optimize the model's performance. For instance, in algorithms like Random Forest or Gradient Boosting, parameters like the number of trees, maximum depth, and learning rate need fine-tuning. Grid search or random search techniques can help identify the best combination of hyperparameters while using suitable evaluation metrics like F1-score or area under the ROC curve (AUC). By finding the optimal hyperparameters, the model can achieve higher accuracy and better discrimination between legitimate and fraudulent transactions, thus improving overall fraud detection effectiveness.

7. Model deployment:

Model deployment in credit card fraud detection is the crucial step of putting the trained model into action to protect against fraudulent transactions. Begin by selecting a deployment environment, whether it's on-premises or in the cloud, ensuring it meets performance and security requirements. Develop an efficient API or batch processing system to integrate the model with real-time transaction processing or batch data pipelines.

8. Testing and validation:

Testing and validation are fundamental steps in credit card fraud detection to ensure the reliability and accuracy of the system. Start by splitting the dataset into training and testing subsets, using metrics like precision, recall, F1-score, and ROC-AUC to evaluate model performance. Conduct cross-validation to assess generalization capabilities. Simulate various fraud scenarios and conduct stress tests to evaluate the system's robustness. Continuously validate and update the model with new data, adapting it to evolving fraud patterns.
Rigorous testing and validation are essential to maintain trust and effectiveness in preventing fraudulent transactions.

9. Documentation and user guides:

Creating comprehensive documentation and user guides is crucial for a credit card fraud detection system's successful implementation. Document the system architecture, data sources, and preprocessing steps for transparency and future reference. Provide detailed instructions for model deployment, configuration, and maintenance to ensure smooth operations. Include guidelines for interpreting model outputs and handling alerts generated by the system. Regularly update the documentation to reflect system changes and evolving best practices, empowering users to effectively utilize and manage the fraud detection system while maintaining compliance and security standards.

10. Deployment and rollout:

Deployment and rollout of a credit card fraud detection system require a well-planned approach to ensure seamless integration. Start by conducting thorough testing in a staging environment to validate the system's functionality and performance. Once validated, deploy the system gradually, starting with a subset of transactions to monitor its effectiveness and fine-tune thresholds. Collaborate with relevant stakeholders, including IT, security, and compliance teams, to ensure a smooth transition. Implement continuous monitoring post-deployment to swiftly address any issues or anomalies. Communicate the rollout process to end-users, providing training and support as needed. A phased deployment strategy minimizes disruptions while enhancing fraud prevention capabilities over time.

11. Ongoing improvement:

Ongoing improvement is imperative in credit card fraud detection to stay ahead of evolving fraud techniques and maintain system effectiveness. Continuously monitor system performance and collect feedback from users and analysts to identify areas for enhancement. Regularly update the dataset with new transaction data to keep the model current. Employ advanced analytics techniques and leverage the latest machine learning advancements to refine the model's ability to detect new fraud patterns. Collaborate with industry peers and stay informed about emerging threats to adapt proactively. Furthermore, invest in training and skill development for the team responsible for fraud detection to ensure they have the expertise needed to implement improvements effectively and safeguard against emerging risks.

By following these steps, the design for the credit card fraud detection system can be effectively transformed into a functional and deployable.

STEP 4:-

DATASET DESCRIPTION:-

About Dataset

The "Credit Card Fraud Detection" dataset on Kaggle, is a widely known dataset used for machine learning and data analysis tasks. This dataset contains credit card transaction data and is typically used to develop models that can identify fraudulent transactions.

Content

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

STEP 5:-

Development of the project-Part1

1. Importing the required libraries:

Importing the necessary Python libraries, including NumPy, Pandas, scikit-learn functions for model training, and performance evaluation which is used to

perform credit card fraud detection for the given dataset.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

2. Loading the Dataset:

In this step we are going to import the dataset that is going to be used across this project.

- Load the credit card transaction data from a CSV file named 'creditcard.csv' located at '/content/creditcard.csv'.
- Use the ‘pd.read_csv’ function from Pandas to read the data.
- credit_card_data.head() is used to display the first few rows of the data set .By default, it displays the first 5 rows of the Data Frame.
- credit_card_data.tail() is used to display the last few rows of the dataset. Similar to head(), it displays the last 5 rows by default

```
credit_card_data = pd.read_csv('/content/creditcard.csv')
credit_card_data.head()
credit_card_data.tail()
```

Output

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153

5 rows × 31 columns

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.8
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.1
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.0

rows × 31 columns

3.Dataset Information:

The credit card data. info () is used to obtain essential information about the dataset stored in the credit card data. It provides a concise summary of the dataset's characteristics. This information is valuable during the data preprocessing and data exploration phases.

```
credit_card_data.info()
```

The output of credit _card_ data. Info () will give you a quick overview of the dataset's structure, including the data types of the columns, which can be crucial for understanding your data, checking for missing values, and ensuring that data types are appropriate for analysis. This information is valuable during the data preprocessing and data exploration phases.

output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

4.Separating the Data for Analysis: This step is done to separate the original credit card transaction dataset, credit_card_data, into two separate DataFrames based on the value of the 'Class' column. This column typically represents whether a transaction is legitimate (Class 0) or fraudulent (Class 1).

```
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

5.Balancing the dataset:

It is used to balance the dataset by selecting a subset of legitimate transactions to match the number of fraudulent transactions (that is 492 fraudulent cases). Balancing the dataset is essential in scenarios where one class is significantly underrepresented, such as in credit card fraud detection, to ensure that the model doesn't become biased towards the majority class.

```
legit_sample = legit.sample(n=492)
```

6.Checking for Missing Values:

This step helps identify and handle missing values in the dataset. The code credit_card_data.isnull().sum() is used to count and display the number of missing (null) values in each column of the credit_card_data.

output

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Code

```
credit_card_data.isnull().sum()
```

STEP 6 :-

Development of the project- Part2

1.FEATURE ENGINEERING :-

Under-Sampling and balancing the data

We first resample the data to balance the classes by selecting a random subset of "legit" transactions and then combines it with the "fraud" transactions. This new dataset is prepared for machine learning by separating the features (X) and labels (Y). It's important to note that balancing the classes is a crucial step when dealing with imbalanced datasets to prevent the model from being biased towards the majority class.

```
legit_sample = legit.sample(n=492)
```

In this line, a random sample of 492 records (transactions) is taken from the "legit" subset of the dataset. This step is intended to balance the number of "legit" and "fraud" transactions in the new dataset.

```
new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

Here, the previously sampled "legit" transactions (492 of them) are concatenated with the original "fraud" transactions. The axis=0 argument indicates that the concatenation should be done along rows. As a result, "new_dataset" now contains both the randomly sampled "legit" and "fraud" transactions

```
[ ]  new_dataset['Class'].value_counts()

1    492
0    492
Name: Class, dtype: int64
```

This line counts the occurrences of each unique value in the "Class" column of the new dataset. It essentially shows the distribution of the two classes, which should now be more balanced due to the previous sampling.

Splitting the data into Features & Targets

```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

The "Class" column is dropped from the "new_dataset" to create a feature matrix "X." This matrix contains all the input features that will be used for training the model. The target vector "Y," which contains the corresponding labels indicating whether each transaction is "legit" (0) or "fraud" (1). It's the variable that the model aims to predict.

2.MODEL TRAINING

Split the data into Training data & Testing Data

It is used to split a dataset into training and testing subsets using the train_test_split function.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

+ Code + Text

```
[ ] print(X.shape, X_train.shape, X_test.shape)

    (984, 30) (787, 30) (197, 30)
```

These are typically used to represent the feature matrix (X) and the target vector (Y). X represents the features or attributes of your dataset, and each row corresponds to a data point, while each column represents a different feature. Y represents the target variable or labels, which you want your model to predict based on the features in X.

Logistic Regression

This creates an instance of the logistic regression model. In scikit-learn, Logistic Regression is a class used to create logistic regression models for binary classification tasks.

```
[ ]  model = LogisticRegression()
```

```
# training the Logistic Regression Model with Training Data
model.fit(X_train, Y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In summary, the code accomplishes the following:

- Creates a logistic regression model.
- Trains the model using the training data (X_train and Y_train).
- During training, the model learns the decision boundary that distinguishes the two classes based on the features in the training data.
- The trained model can then be used to make predictions on new, unseen data.

3.MODEL EVALUATION

Accuracy on training data :

- It calculates the accuracy of the logistic regression model on the training data.

Accuracy Score

```
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
[ ]  print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9415501905972046
```

It calculates the accuracy of the logistic regression model on the training data by comparing the model's predictions to the actual labels. The accuracy score represents the proportion of correct predictions made by the model on the training dataset, indicating how well the model is performing on data it has already seen during training.

Accuracy on testing data :

This below code calculates the accuracy of the logistic regression model on the

ACCURACY SCORE

```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

```
[ ] print('Accuracy score on Test Data : ', test_data_accuracy)

    Accuracy score on Test Data :  0.9390862944162437
```

test data.The accuracy score reflects the proportion of correct predictions made by the model on a dataset it has never seen during training.

STEP 7:

CHOICE OF MACHINE LEARNING ALGORITHM:-

Logistic Regression

This creates an instance of the logistic regression model. In scikit-learn, Logistic Regression is a class used to create logistic regression models for binary classification tasks.

```
[ ] model = LogisticRegression()
```

```
# training the Logistic Regression Model with Training Data
model.fit(X_train, Y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In summary, the code accomplishes the following:

- Creates a logistic regression model.
- Trains the model using the training data (X_train and Y_train).
- During training, the model learns the decision boundary that distinguishes the two classes based on the features in the training data.

- The trained model can then be used to make predictions on new, unseen data.

EVALUATION METRICS:

The evaluation metrics for the credit card fraud detection code you provided are precision, recall, and F1 score. Precision is the fraction of predicted positive cases that are actually positive. Recall is the fraction of actual positive cases that are correctly predicted.F1 score is a harmonic mean of precision and recall.

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Calculate predictions on the test set
Y_test_prediction = model.predict(X_test)

# Calculate precision, recall, and F1-score
precision = precision_score(Y_test, Y_test_prediction)
recall = recall_score(Y_test, Y_test_prediction)
f1 = f1_score(Y_test, Y_test_prediction)

print('Precision: ', precision)
print('Recall: ', recall)
print('F1-Score: ', f1)
```

output

```
Precision:  0.9523809523809523
Recall:  0.9090909090909091
F1-Score:  0.9302325581395349
```