## Credit Card Fraud Detection

Data sourcing: obtain a dataset that contains historical credit card transactions. We can find such datasets on Kaggle.

Data preprocessing: Clean the data by handling missing values, duplicates, and outliers. Encode categorical variables if necessary. Normalize or scale numerical features to ensure they have similar ranges.

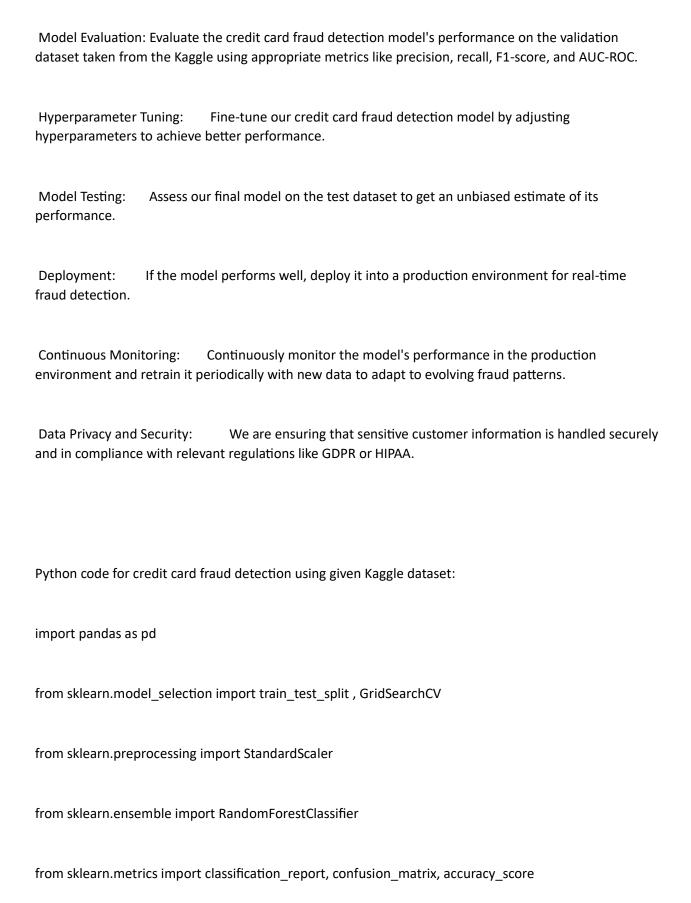
Feature Engineering: Create relevant features or transformations that might help improve the model's performance. For example, if we could calculate transaction amounts relative to the user's historical spending habits.

Data Splitting: Divide the dataset into training, validation, and test sets. This is important to evaluate our model's performance accurately.

Model Selection: We are choosing a suitable machine learning or deep learning algorithm for our credit card fraud detection project. We are using logistic regression algorithm for my project.

Model Training: Train the selected model using the training data.

Hyperparameter Tuning (Optional): If your model has hyperparameters that need tuning (e.g., learning rate, max depth for trees), perform hyperparameter optimization.



```
# Load the dataset
url = "https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud/data"
# Preprocess the data
data = data.drop(['Time'], axis=1)
data = data.dropna()
# Handling missing values
df = df.dropna()
# Feature Engineering
df['is_fraud'] = df['is_fraud'].map({'no': 0, 'yes': 1})
# Split the data into features and target
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
# Standardize the features
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Train the Random Forest Classifier
classifier = RandomForestClassifier(n_estimators=100, random_state=0)
classifier.fit(X_train, y_train)
# Hyperparameter tuning using GridSearchCV
param_grid = {
  'n_estimators': [100, 200, 300, 400, 500],
  'max_depth': [None, 10, 20, 30, 40, 50],
  'min_samples_split': [2, 5, 10],
  'min_samples_leaf': [1, 2, 4],
  'bootstrap': [True, False]
}
```

```
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
# Make predictions on the testing set
y_pred = classifier.predict(X_test)
# Evaluate the performance of the classifier
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
# Split the data into features and target
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
# Standardize the features
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Train the Random Forest Classifier
classifier = RandomForestClassifier(n_estimators=100, random_state=0)
classifier.fit(X_train, y_train)
# Make predictions on the testing set
y_pred = classifier.predict(X_test)
# Evaluate the performance of the classifier
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
# Print the best parameters
print("Best Parameters: ", grid_search.best_params_)
```