**Fraud Detection Classification Model Project**

I worked on a fraud detection project for a client. The objective of the project was to build a classification model that could automatically identify fraudulent transactions based on various features and patterns in transaction data. The client had observed several fraudulent transactions in the last quarter and wanted to better understand the characteristics of such frauds. By identifying potentially fraudulent customers, the client aimed to implement targeted marketing strategies to reduce fraud risk and optimize the marketing budget.

**Problem Statement:**

The key problem we needed to solve was identifying customers who might engage in fraudulent activities. The fraud detection system would flag suspicious transactions based on factors like transaction location, IP address, and timing. For example, transactions originating from unknown locations, unusual IP addresses, or simultaneous transactions from different locations could raise suspicion.

Additionally, we focused on time-based fraud detection. If a previously inactive customer suddenly conducted a suspicious transaction, they were flagged as potentially fraudulent.

**Data Collection:**

To begin, we collected transaction data from the client’s database using **Python** and **CX-Oracle** to establish a connection. We pulled all the necessary columns into the Python environment for further analysis.

**Data Preprocessing:**

Once the data was available, we developed a comprehensive preprocessing pipeline. This pipeline involved:

* **Memory Optimization:** Ensuring efficient use of memory for large datasets.
* **Handling Null Values:** Treating missing data through imputation techniques.
* **Outlier Detection and Treatment:** Addressing extreme values that could skew results.
* **Min-Max Scaling:** Standardizing the range of features.
* **Encoding:** Applying dummy variables and label encoding for categorical data.
* **Standard Scaling:** Ensuring features had a uniform scale.

This preprocessing pipeline was automated, making it reusable for future projects.

**Model Building:**

The next step was model building. The data exhibited non-linear relationships, so we began by conducting a **correlation test** and **Chi-Square test** to assess relationships between categorical variables. These tests allowed us to understand the dependency between features and their impact on the target variable.

I used scatter plots and other visualization tools to further examine the relationships in the data. Based on the non-linear nature of the data, we selected machine learning algorithms like:

* **Decision Tree:** Used as our baseline model with an initial accuracy of around 71%.
* **Random Forest:** This improved the model accuracy to 80-85%.
* **XGBoost:** A boosting algorithm that pushed the model accuracy to around 87%.

We measured model performance using metrics such as **Precision**, **Recall**, **F1-score**, and **Accuracy**. Since our data was imbalanced, we applied **SMOTE** (Synthetic Minority Over-sampling Technique) to balance the dataset.

**Model Evaluation:**

For evaluation, we conducted **K-Fold Cross-Validation** with 5 folds to ensure model robustness. The average accuracy across the folds was around 86%, and we saved the final model for deployment.

**Deployment:**

The final model was saved as a **Pickle** file for inference purposes. We deployed the model using **Flask** to serve the model predictions via an API.

{Alternatively, **FastAPI** or **Django** could also be used for deployment depending on client requirements. }