📄 Detailed Report on Credit Card Fraud Detection Project

✨ Project Objective:

Task is to build a **Machine Learning model** that can:

- **Detect fraudulent transactions** based on transaction features like amount, timestamp, merchant info, etc.
- **Minimize false positives** (legitimate transactions wrongly flagged as fraud),
- **Maximize detection accuracy**,
- **Explain misclassifications**.

📦 Code Walkthrough:

---

1. Importing Libraries

python

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Explanation:

- pandas is used for data loading and manipulation (tables, columns, rows).
- numpy helps with numerical operations (arrays, math functions).
- matplotlib and seaborn are for plotting graphs and visualizing patterns.

Why:
Understanding the data visually is **important** to identify fraudulent vs legitimate patterns.

---

2. Loading the Dataset

```python
data = pd.read_csv('fraudTrain.csv')
data.head()
```

Explanation:

- Loads a CSV file named fraudTrain.csv into a pandas DataFrame called data.
- head() shows the first 5 rows to give an idea of the dataset.

Why:
First step is always inspecting the raw data.

---

3. Data Overview

```
data.info()
```

Explanation:

- info() shows:
    - Total entries (rows),
    - Column names,
    - Data types (int, float, object),
    - Non-null counts (detect missing values).

Observation:
Helps decide **which columns** need cleaning, encoding, or dropping.

---

4. Checking for Missing Values

```
data.isnull().sum()
```

Explanation:

- isnull() identifies missing entries,
- sum() counts how many missing per column.

Why:
Missing values can **bias the model** if not handled.

---

5. Exploratory Data Analysis (EDA)

Plotting Fraud vs Non-Fraud Cases

```
sns.countplot(data['is_fraud'])
```

Explanation:

- is_fraud is the label column (0 = not fraud, 1 = fraud).
- countplot shows how many fraud and not-fraud cases exist.

Observation:

- Highly **imbalanced dataset** (few frauds compared to many non-frauds).

---

6. Feature Engineering

Converting timestamps

```
data['trans_date_trans_time'] = pd.to_datetime(data['trans_date_trans_time'])
data['hour'] = data['trans_date_trans_time'].dt.hour
```

Explanation:

- Converts the string timestamps into a **datetime** object.
- Extracts the **hour of the transaction** as a new feature.

Why:
Frauds may happen more during **odd hours** (e.g., midnight).

Dropping Useless Features

```
data = data.drop(['trans_date_trans_time', 'Unnamed: 0', 'first', 'last', 'street', 'city', 'state', 'zip',
'dob', 'trans_num', 'unix_time', 'merch_lat', 'merch_long'], axis=1)
```

Explanation:

- Removes **irrelevant columns** like names, street addresses, transaction IDs, etc.

Why:
These fields don't contribute to predicting fraud, and keeping them can introduce noise.

Encoding Categorical Variables

```
data['category'] = data['category'].astype('category').cat.codes
data['gender'] = data['gender'].astype('category').cat.codes
```

Explanation:

- Converts category and gender columns into numbers.
- cat.codes assigns integers to categories automatically.

Why:
Machine Learning models work only with **numbers**, not text.

---

7. Splitting the Data

```
from sklearn.model_selection import train_test_split

X = data.drop('is_fraud', axis=1)
y = data['is_fraud']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Explanation:

- Separates **features** (X) from **label** (y = fraud or not fraud).
- **Splits** the data into:
  - 70% training (for learning),
  - 30% testing (for evaluation),
- random_state=42 ensures **reproducibility**.

---

8. Handling Imbalanced Data

Using SMOTE (Synthetic Minority Oversampling Technique)

```
from imblearn.over_sampling import SMOTE

smote = SMOTE()
X_train, y_train = smote.fit_resample(X_train, y_train)
```

Explanation:

- **SMOTE** generates **new synthetic samples** for the minority class (fraud).
- Now, the training data has **balanced** fraud and non-fraud cases.

Why:
If you don't balance, the model will **ignore frauds** because they are rare.

---

9. Model Building

Using Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(X_train, y_train)
```

Explanation:

- A **Random Forest** is an ensemble of decision trees.
- It trains on the balanced dataset to **learn patterns** distinguishing frauds.

Why Random Forest?
It handles:

- **Non-linearity** (complex decision boundaries),
- **Feature importance** (helps interpret important fraud indicators),
- **Overfitting** control (using many trees).

---

10. Model Prediction and Evaluation

```
y_pred = model.predict(X_test)
```

Evaluating Model

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

Explanation:

- **Confusion Matrix**: Tells how many:
  - True positives (frauds correctly detected),
  - True negatives (legitimate correctly ignored),
  - False positives (false alarms),
  - False negatives (missed frauds),

- **Classification Report**: Shows precision, recall, F1-score.
- **Accuracy**: Overall performance.

Observation:

- **Precision**: How many detected frauds were actual frauds.
- **Recall**: How many actual frauds were detected.
- **F1-Score**: Balance between precision and recall.

---

11. Feature Importance

```
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

Explanation:

- Finds the **top 10 important features** used by the model.
- barh plot shows which features contribute most to fraud detection.

Why:
Helps understand **which features** are most useful (example: transaction amount, hour, merchant).

---

12. Misclassification Analysis

You are expected to **analyze errors**:

✅ Check False Positives:

- Legit transactions wrongly flagged as fraud.
- Possible if large amounts at odd hours.

✅ Check False Negatives:

- Missed frauds.
- Possible if small amounts or regular-looking transactions.

You can retrieve misclassified examples like this:

```
wrong_predictions = X_test[y_test != y_pred]
wrong_predictions['Actual'] = y_test[y_test != y_pred]
wrong_predictions['Predicted'] = y_pred[y_test != y_pred]
wrong_predictions.head()
```