TITLE: Fraud Detection System For Online Transaction

Subtitle: Applying RandomForest and XGBoost for Fraud Detection and Developed with Streamlit

Group 3:

Khansa Rubab
Shannon Chen
Victoria Giles
Zalak Shah

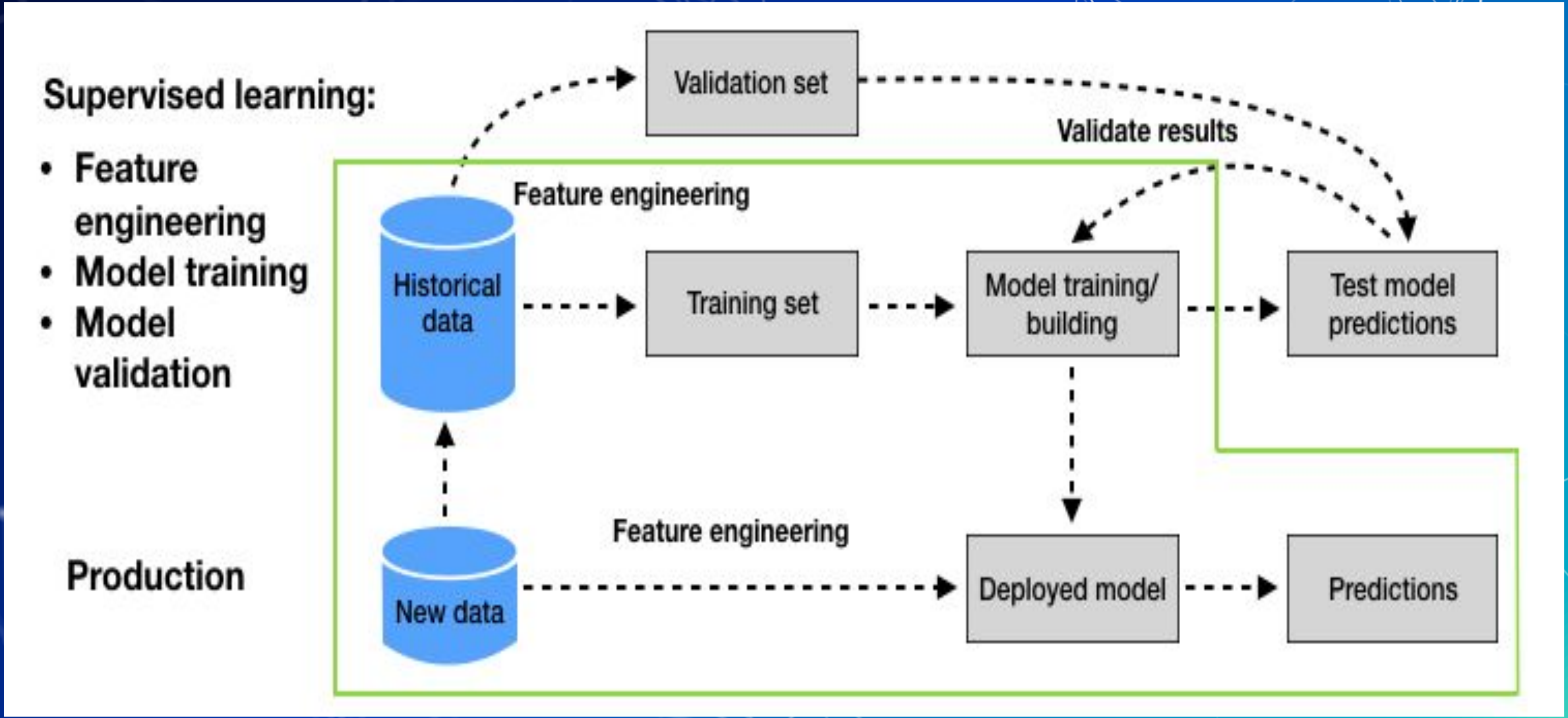
Table of Contents

INTRODUCTION

Background on the problem of fraud detection:

- ❑ Tackling the challenge of online transaction fraud detection.
- ❑ The significance of timely and accurate fraud identification.
- ❑ Goal: Create a machine learning model to assess the probability of fraudulent transactions (classification problem).

Targeted Supervised Learning Solutions for Data-Driven Decision Making:



Dataset:

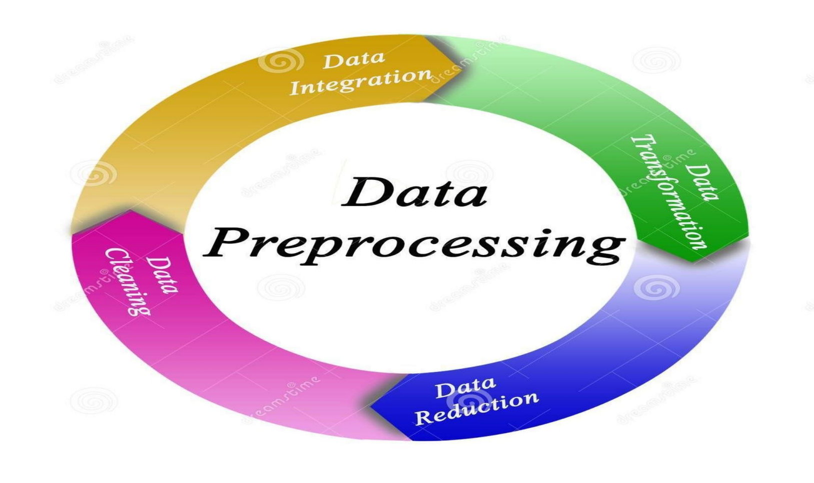
- Source: Kaggle (https://www.kaggle.com/competitions/ieee-fraud-detection/data)
- Consists of transactional and identity data
- **Features:** TransactionID, TransactionDT, TransactionAmt, ProductCD, card1-6, addr1-2, dist1-2, P_emaildomain, R_emaildomain, C1-C14, D1-D15, M1-M9, V1-V339, id_01-id_38, DeviceType, DeviceInfo
- **Target:** is Fraud

Overcoming Dataset Complexity:

- ## Challenges faced:
  - Dataset size and complexity make hyperparameter tuning difficult on Jupyter and Google Colab.
  - Computational limitations impacted model training and evaluation
  - Aws free tier does not work.
- ## Concurrent attempt to transition to Apache Spark from Pandas
  - Leverage Spark's distributed computing capabilities
  - Efficiently handle large-scale data processing tasks
  - Improve model training and evaluation times
  - **Limitations:** Having to learn a new language on a fly

Data Processing Workflow:

Data Pre-processing:



- ❑ Conduct exploratory data analysis
- ❑ Combine identity and transaction datasets
- ❑ Manage missing values:
  - ❑ Remove columns with over 40% missing values
  - ❑ Impute remaining missing values using respective column means
  - ❑ Replace infinity values with NaN and fill with column mean
- ❑ Save cleaned data to CSV file
- ❑ Encode categorical variables with LabelEncoder
- ❑ Standardize numerical features using StandardScaler
- ❑ Store processed data in an SQL database

Data exploration and cleaning process

Step 2: Exploring the Data

```
# get shape of the data
print("test_identity Shape: ", test_identity.shape)
print("test_transaction Shape: ", test_transaction.shape)
print("train_identity Shape: ", train_identity.shape)
print("train_transaction Shape: ", train_transaction.shape)
```

```
test_identity Shape:  (141907, 41)
test_transaction Shape:  (506691, 393)
train_identity Shape:  (144233, 41)
train_transaction Shape:  (590540, 394)
```

```
# print first two rows of each dataset
print(test_identity.head())
print(test_transaction.head())
print(train_identity.head())
print(train_transaction.head(2))
```

```
   TransactionID  id-01     id-02  id-03  id-04  id-05  id-06  id-07  id-08
0        3663586  -45.0  280290.0    NaN    NaN    0.0    0.0    NaN    NaN
1        3663588    0.0    3579.0    0.0    0.0    0.0    0.0    NaN    NaN
2        3663597   -5.0  185210.0    NaN    NaN    1.0    0.0    NaN    NaN
3        3663601  -45.0  252944.0    0.0    0.0    0.0    0.0    NaN    NaN
4        3663602  -95.0  328680.0    NaN    NaN    7.0  -33.0    NaN    NaN

   id-09  ...                  id-31  id-32      id-33           id-34  \
0    NaN  ...  chrome 67.0 for android    NaN        NaN             NaN
1    0.0  ...  chrome 67.0 for android   24.0   1280x720  match_status:2
2    NaN  ...       ie 11.0 for tablet    NaN        NaN             NaN
3    0.0  ...  chrome 67.0 for android    NaN        NaN             NaN
4    NaN  ...  chrome 67.0 for android    NaN        NaN             NaN

  id-35 id-36 id-37 id-38 DeviceType                    DeviceInfo
0     F     F     T     F     mobile  MYA-L13 Build/HUAWEIMYA-L13
1     T     F     T     T     mobile         LGLS676 Build/MXB48T
2     F     T     T     F    desktop                  Trident/7.0
3     F     F     T     F     mobile  MYA-L13 Build/HUAWEIMYA-L13
4     F     F     T     F     mobile         SM-G9650 Build/R16NW
```

In [6]:

```
# get information about the data
print(test_identity.info())
print(test_transaction.info())
print(train_identity.info())
print(train_transaction.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 141907 entries, 0 to 141906
Data columns (total 41 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   TransactionID   141907 non-null  int64
 1   id-01           141907 non-null  float64
 2   id-02           136976 non-null  float64
 3   id-03           66481 non-null   float64
 4   id-04           66481 non-null   float64
 5   id-05           134750 non-null  float64
 6   id-06           134750 non-null  float64
 7   id-07           5059 non-null    float64
 8   id-08           5059 non-null    float64
 9   id-09           74338 non-null   float64
 10  id-10           74338 non-null   float64
 11  id-11           136778 non-null  float64
 12  id-12           141907 non-null  object
 13  id-13           130286 non-null  float64
 14  id-14           71357 non-null   float64
 15  id-15           136977 non-null  object
 16  id-16           125747 non-null  object
 17  id-17           135966 non-null  float64
 18  id-18           50875 non-null   float64
 19  id-19           135906 non-null  float64
 20  id-20           135633 non-null  float64
 21  id-21           5059 non-null    float64
 22  id-22           5062 non-null    float64
 23  id-23           5062 non-null    object
 24  id-24           4740 non-null    float64
 25  id-25           5039 non-null    float64
 26  id-26           5047 non-null    float64
 27  id-27           5062 non-null    object
 28  id-28           136778 non-null  object
 29  id-29           136778 non-null  object
 30  id-30           70659 non-null   object
 31  id-31           136625 non-null  object
 32  id-32           70671 non-null   float64
 33  id-33           70671 non-null   object
 34  id-34           72175 non-null   object
```

Data exploration and cleaning process

The train dataset has a target column called `isFraud`.

In [7]:

```
# get descriptive statistics for each dataset
print(test_identity.describe())
print(test_transaction.describe())
print(train_identity.describe())
print(train_transaction.describe())
```

	TransactionID	id-01	id-02	id-03 \
count	1.419070e+05	141907.000000	136976.000000	66481.000000
mean	3.972166e+06	-11.325734	192658.729909	0.053008
std	1.469966e+05	14.508520	182613.277215	0.684551
min	3.663586e+06	-100.000000	2.000000	-12.000000
25%	3.859268e+06	-12.500000	63339.500000	0.000000
50%	4.001774e+06	-5.000000	133189.500000	0.000000
75%	4.105284e+06	-5.000000	265717.500000	0.000000
max	4.170239e+06	0.000000	999869.000000	11.000000

	id-04	id-05	id-06	id-07	id-08 \
count	66481.000000	134750.000000	134750.000000	5059.000000	5059.000000
mean	-0.087454	1.246033	-6.803829	12.493180	-36.577782
std	0.840351	5.071394	15.921457	11.678206	25.544185
min	-19.000000	-81.000000	-100.000000	-41.000000	-100.000000
25%	0.000000	0.000000	-6.000000	3.000000	-46.000000
50%	0.000000	0.000000	0.000000	12.000000	-33.000000
75%	0.000000	1.000000	0.000000	21.000000	-23.000000
max	0.000000	52.000000	0.000000	59.000000	0.000000

	id-09	...	id-17	id-18	id-19 \
count	74338.000000	...	135966.000000	50875.000000	135906.000000
mean	0.076219	...	191.070341	14.795735	350.122982
std	1.009687	...	30.749535	2.318496	139.140824
min	-32.000000	...	100.000000	11.000000	100.000000
25%	0.000000	...	166.000000	13.000000	266.000000
50%	0.000000	...	166.000000	15.000000	321.000000
75%	0.000000	...	225.000000	15.000000	427.000000
max	16.000000	...	228.000000	29.000000	670.000000

	id-20	id-21	id-22	id-24	id-25 \
count	135633.000000	5059.000000	5062.000000	4740.000000	5039.000000
mean	408.886230	507.727021	15.336823	13.166667	332.043064
std	158.971756	227.371061	5.618032	3.222440	86.356683

```
train_data Shape:  (590540, 434)
test_data Shape:  (506691, 433)
```

In [11]:

```
# Check for missing values in train and test data
missing_train = train.isnull().sum().sort_values(ascending=False)
missing_test = test.isnull().sum().sort_values(ascending=False)
```

In [12]:

```
# Display the percentage of missing values in each column
print("Missing values in train (%):")
print((missing_train / len(train)) * 100)
print("\nMissing values in test_data (%):")
print((missing_test / len(test)) * 100)
```

```
Missing values in train (%):
id_24            99.196159
id_25            99.130965
id_07            99.127070
id_08            99.127070
id_21            99.126393
                   ...
C11               0.000000
C12               0.000000
C13               0.000000
C14               0.000000
TransactionID     0.000000
Length: 434, dtype: float64

Missing values in test_data (%):
id-24            99.064519
id-25            99.005508
id-26            99.003929
id-07            99.001561
id-08            99.001561
                   ...
V111              0.000000
V112              0.000000
V113              0.000000
V114              0.000000
TransactionID     0.000000
Length: 433, dtype: float64
```

Note:

**drop columns with a missing value percentage greater than a certain threshold (let's say 50%)

Step 5: Handle Missing Values

Data exploration and cleaning process

```
In [13]:
# Drop columns with more than 40% missing values
train_data = train.drop(columns=missing_train[missing_train > 0.40 * len(train)].index)
test_data = test.drop(columns=missing_test[missing_test > 0.40 * len(test)].index)
```

```
In [14]:
# Impute missing values in the remaining columns with their respective means
train_data.fillna(train_data.mean(), inplace=True)
test_data.fillna(test_data.mean(), inplace=True)
```

```
In [15]:
# Replace infinity values with NaN
train_data.replace([np.inf, -np.inf], np.nan, inplace=True)
test_data.replace([np.inf, -np.inf], np.nan, inplace=True)

# Fill NaN values with the mean of each column
train_data.fillna(train_data.mean(), inplace=True)
test_data.fillna(test_data.mean(), inplace=True)
```

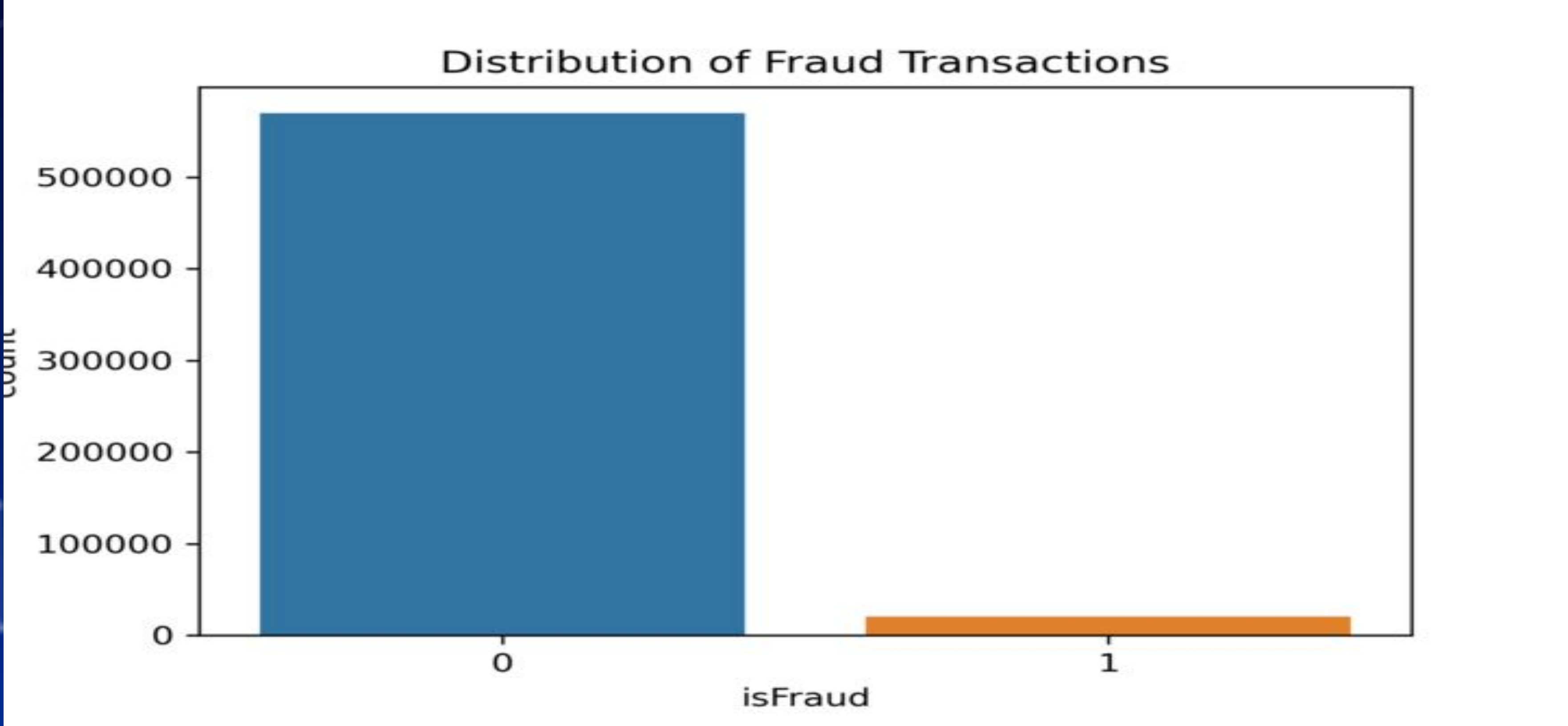
```
In [16]:
# Load cleaned data
clean_train_data = pd.read_csv("Resources/clean_train_data.csv")
clean_test_data = pd.read_csv("Resources/clean_test_data.csv")
```

```
In [17]:
# from sklearn.preprocessing import LabelEncoder

# # Identify categorical columns
# categorical_columns = X_train.select_dtypes(include=['object']).columns

# # Apply label encoding
# for col in categorical_columns:
#     le = LabelEncoder()
#     le.fit(pd.concat([X_train[col], X_val[col], test_data[col]]))
#     X_train[col] = le.transform(X_train[col])
#     X_val[col] = le.transform(X_val[col])
#     test_data[col] = le.transform(test_data[col])
```

```
In [18]:
# X_train.to_csv("Resources/X_train.csv", index=False)
# X_val.to_csv("Resources/X_val.csv", index=False)
# y_train.to_csv("Resources/y_train.csv", index=False)
# y_val.to_csv("Resources/y_val.csv", index=False)
# test_data.to_csv("Resources/test_data.csv", index=False)
```

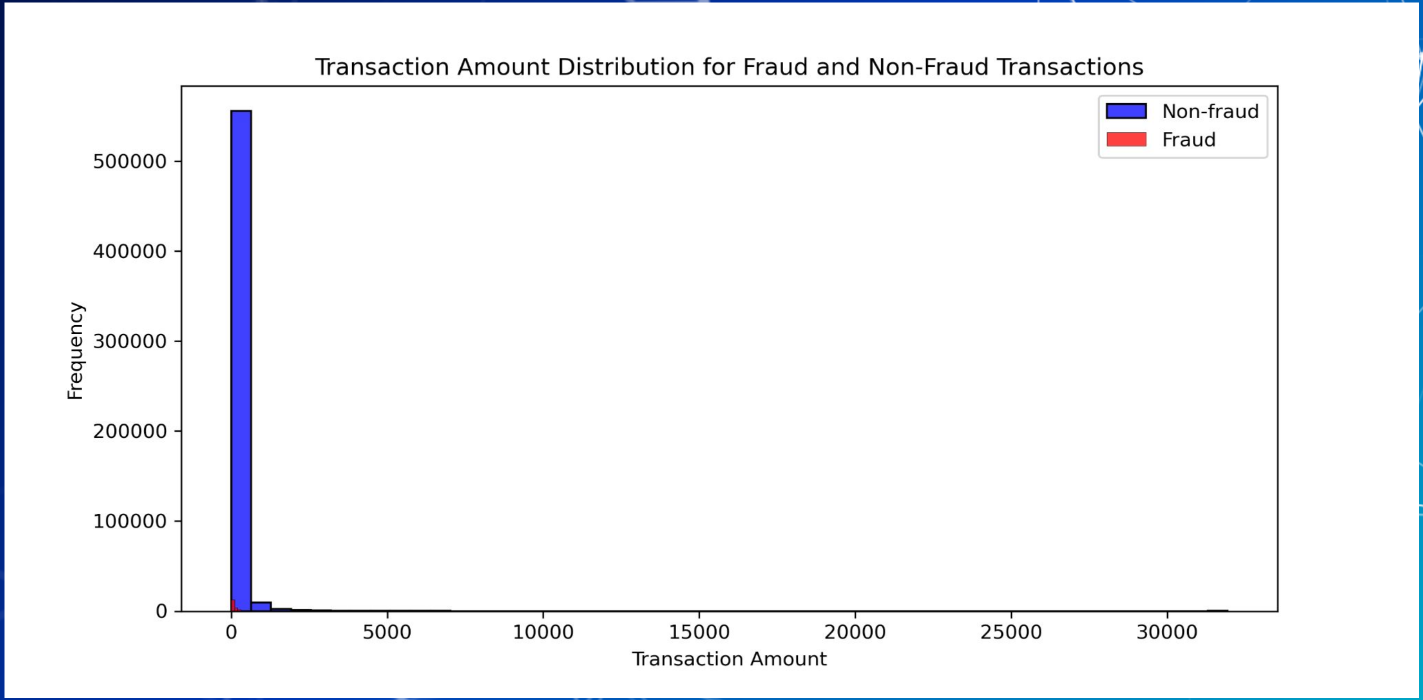
Data Visualisation

- Examine target variable distribution (isFraud) in the train_transaction dataset



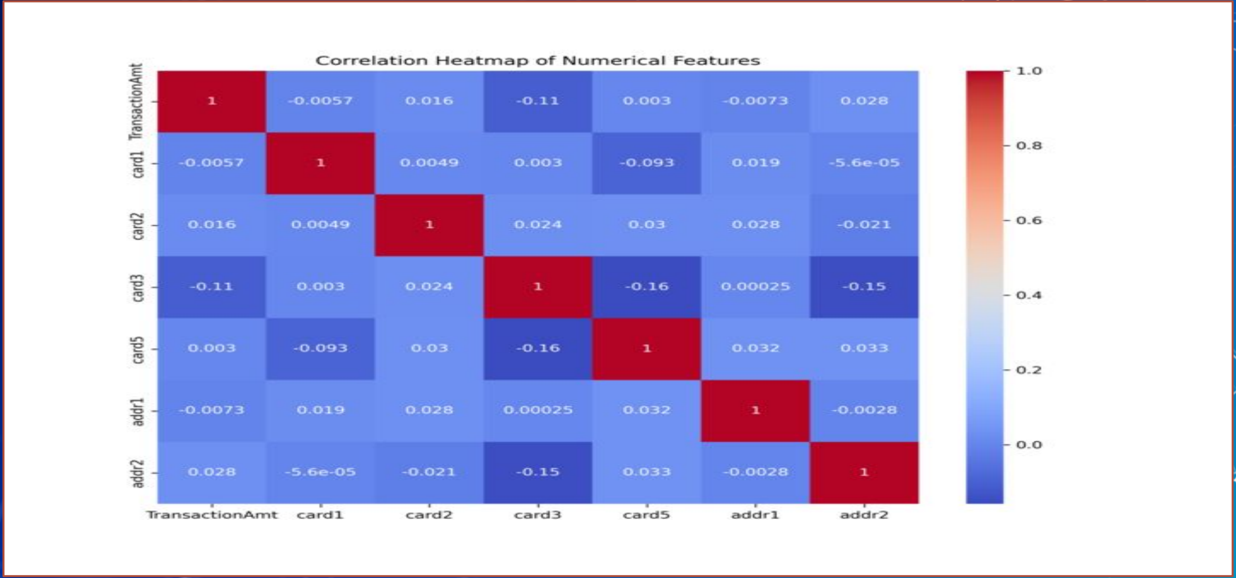
Distribution of the Variable for both:

- Compare Transaction Amt distributions for fraud and non-fraud transactions



Heatmap to visualise correlations

- Investigate correlations between selected numerical features using a heatmap:
- Features: Transaction_Amt, card1, card2, card3, card5, addr1, addr2



- Gain insights into data patterns and relationships to inform model development and feature selection.

Model Selection:

Explore various machine learning models

- RandomForest
- Ensemble methods:
  Extremely Random tree classifier
  AdaBoost
  RandomForest
- Evaluate model performance with classification report

```
Random Forest Classifier:
Training Score: 0.999778733588467
Testing Score: 0.9801469841162326
              precision    recall  f1-score   support

           0       0.98      1.00      0.99    142497
           1       0.93      0.47      0.62      5138

    accuracy                           0.98    147635
   macro avg       0.95      0.73      0.80    147635
weighted avg       0.98      0.98      0.98    147635


Extremely Random Trees Classifier:
Training Score: 1.0
Testing Score: 0.9809462525823822
              precision    recall  f1-score   support

           0       0.98      1.00      0.99    142497
           1       0.92      0.50      0.65      5138

    accuracy                           0.98    147635
   macro avg       0.95      0.75      0.82    147635
weighted avg       0.98      0.98      0.98    147635


AdaBoost Classifier:
Training Score: 0.9705106061119202
Testing Score: 0.9705760829071697
              precision    recall  f1-score   support

           0       0.97      1.00      0.98    142497
           1       0.82      0.20      0.32      5138

    accuracy                           0.97    147635
   macro avg       0.90      0.60      0.65    147635
weighted avg       0.97      0.97      0.96    147635
```

Our Findings for Model Selection and Evaluation

- ❑ Identified Random Forest as the best-performing model based on evaluation metrics
- ❑ We initially found Random Forest to be the top performer, but after examining more resources, XGBoost also emerged as a strong contender.
- ❑ Selected Random Forest and XGBoost for in-depth analysis
- ❑ Retrieved data from the SQL database for model training and testing
- ❑ Compared Random Forest and XGBoost performance on the retrieved dataset
- ❑ Finalized the model selection for app development based on the comparison results

Model Evaluation:

- ❑ Assess models performance using ROC AUC score and classification report
- ❑ Compare RandomForest and XGBoost models
- ❑ Identify the best-performing model based on evaluation metrics

Decision: Developed the XGBoost model for real-time fraud detection with streamlit

Random Forest:

	precision	recall	f1-score	support
0	0.98	1.00	0.99	113866
1	0.94	0.47	0.63	4242
accuracy			0.98	118108
macro avg	0.96	0.74	0.81	118108
weighted avg	0.98	0.98	0.98	118108

Accuracy: 0.9800352220002032
ROC AUC: 0.7362497890665184

XGBoost:

	precision	recall	f1-score	support
0	0.98	1.00	0.99	113866
1	0.92	0.48	0.63	4242
accuracy			0.98	118108
macro avg	0.95	0.74	0.81	118108
weighted avg	0.98	0.98	0.98	118108

Accuracy: 0.9798574186337928
ROC AUC: 0.7394484317086844

Model Optimisation (Attempted)

Attempted with RandomForest & XGBoost Classifiers

Utilised hyperparameter tuning technique: *GridSearchCV*

```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
```

```python
# Define the parameter grids for RandomForest and XGBoost
rf_param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
```

```python
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
```

```python
#  Define the parameter grids for  XGBoost

xgb_param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [6, 10, 15],
    'learning_rate': [0.01, 0.1],
    'subsample': [0.5, 1],
    'colsample_bytree': [0.5, 1]
}
```

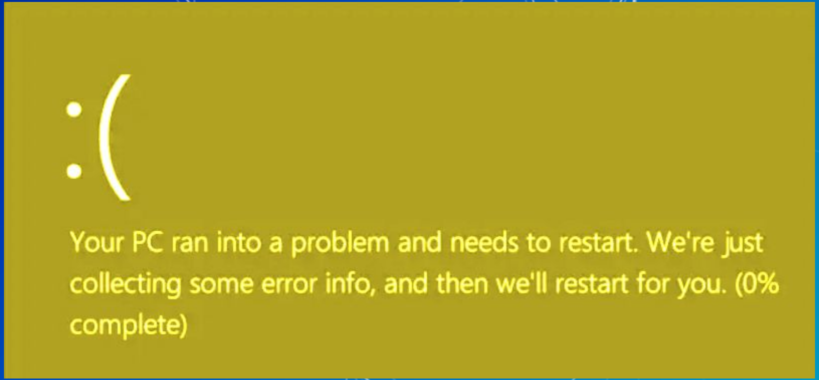
It performs an exhaustive search over all possible combinations of hyperparameters, training and evaluating the model with each combination using cross validation.

- ❑ Returns the hyperparameters that resulted in best performance.

Model Optimisation (Timed out)

Insufficient computing power despite trying various mediums:

- ❑ Pandas on Jupyter Notebook
- ❑ Pandas on Google Colab (with TPU)
- ❑ PySpark on Google Colab (with TPU)



Thus, unable to successfully implement GridSearchCV as a form of hyperparameter tuning.

However, the **XGBoost Classification model** that we tested has some built-in regularization techniques to improve model generalisation, such as:

- ❑ L1, L2
- ❑ Max depth constraints

Model Deployment (Streamlit)

Decision taken to deploy our best performing model (XGBoost Classifier) using 2 features (out of our initial 400 + in the raw dataset) on Streamlit.io.

Created a web app with input fields for each feature in a respective dataset.

Users can upload a new pre-processed dataset onto the app and use it to predict the probability that a transaction is fraudulent.

```
import streamlit as st
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier

# Load the preprocessed data
X_train = pd.read_csv("Resources/X_train.csv")
y_train = pd.read_csv("Resources/y_train.csv")["isFraud"]

# Select two basic features
selected_features = ['TransactionDT', 'TransactionAmt']
X_train_selected = X_train[selected_features]

# Train the XGBoost model
xgb_model = XGBClassifier(use_label_encoder=False, random_state=42)
xgb_model.fit(X_train_selected, y_train, eval_metric='logloss')

# Streamlit app
st.title("Fraud Detection")

transaction_dt = st.number_input("TransactionDT", min_value=0, value=100000)
transaction_amt = st.number_input("TransactionAmt", min_value=0.0, value=50.0)

if st.button("Predict"):
    input_data = pd.DataFrame({"TransactionDT": [transaction_dt],
                               "TransactionAmt": [transaction_amt]})
    prediction = xgb_model.predict(input_data)
    st.write("Prediction: ", "Fraud" if prediction[0] == 1 else "Not Fraud")
```

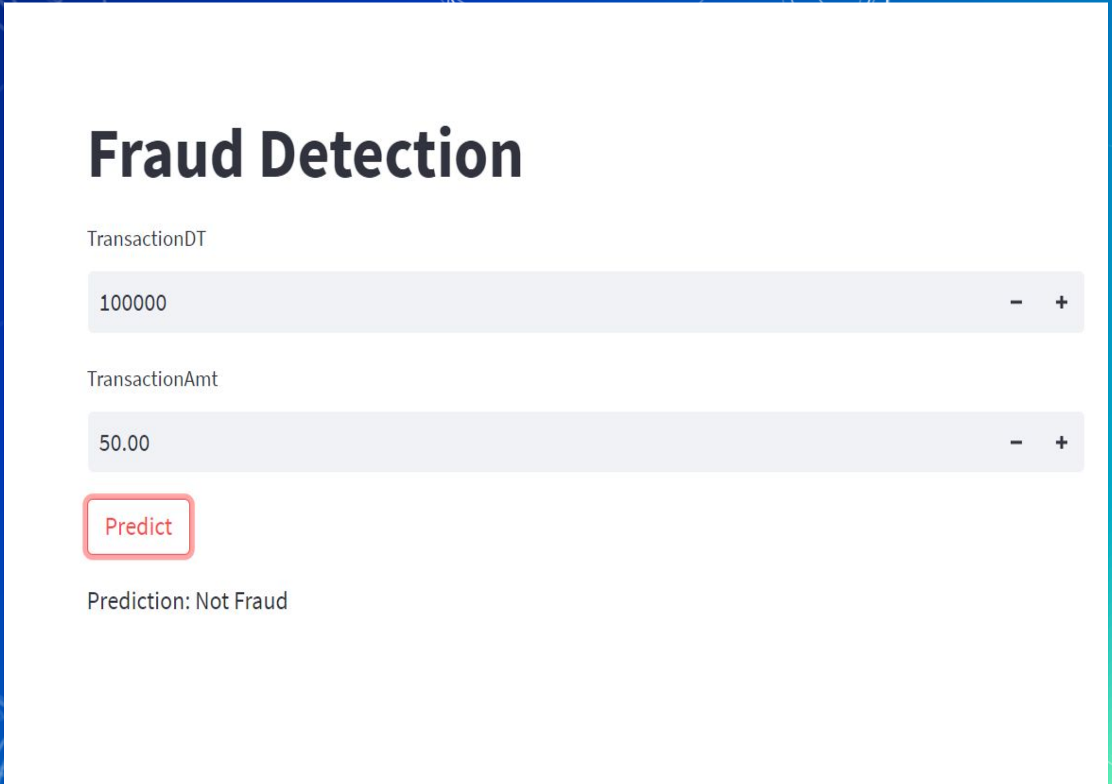
Model Deployment (Streamlit)

Limitation of current app:

It only takes in two basic features for predictions. It was vital that we had a functioning front-end given the tight timeframe.

Future refinements:

Enhance model performance by adding more features, incorporating other ML models, utilising alternative hyperparameter tuning techniques, and deploying the improved model on Heroku for increased accessibility.



Key TakeAways:

- Request more instances of fraudulent transactions from the organization to improve model training and prediction accuracy.
- Address imbalanced dataset challenges with techniques such as oversampling, undersampling, or using cost-sensitive learning.
- Optimize computational resources and training times by leveraging scalable solutions like Apache Spark or distributed computing.
- Seek clarity on column names to potentially merge related variables, inform feature engineering, and improve model interpretability.

Github Link : https://github.com/rubab-malik/Project_4

Thank you Mortaza and Jeffery