Big data week 2 assignment - Shaheer Hussain

Assignment description:

Business Scenario: Real-Time Fraud Monitoring for FinSecure Bank

You are part of the data engineering team at FinSecure Bank, a global retail bank, which is experiencing a surge in online transactions. The risk and compliance department needs a system to detect and respond to potentially fraudulent behaviour in near real time.

Introduction:

FinSecure Bank operates as a worldwide retail banking leader while experiencing an unmatched growth of online financial transactions. The rising number of online financial transactions creates an elevated danger of fraudulent activities which threatens both customer trust and financial stability. The current traditional fraud detection systems which use delayed batch processing of data no longer meet the speed requirements of modern financial operations.

The Risk and Compliance Department at FinSecure Bank has launched a project to create real-time fraud monitoring technology. The project aims to create an advanced Big Data analytics pipeline which takes in substantial transaction data through batch processing or simulated streaming while executing required data cleaning and transformation and enrichment steps. The system will use machine learning methods that include anomaly detection or classification to detect suspicious transactions effectively. The system will produce actionable risk scores and output reports which enable prompt intervention.

The project aims to boost FinSecure Bank's fraud detection abilities while assessing the scalability and constraints of the solution to establish foundations for future enhancements in the bank’s data-driven risk management infrastructure.

Machine learning (ML) is a subset of AI that enables algorithms to uncover hidden patterns within datasets to predict new or similar data without explicit programming for each task. **(GeeksforGeeks, 2019)**

When carrying out ML, we need to consider ingestion, processing, ML, output and storage layers. The ingestion layer is where we import, transfer or load data from different sources **(GeeksforGeeks, 2024)**, Processing is where raw data is transformed into a structured and usable format **(Stedman, 2024)**, the ML layer is responsible for training, developing and deploying models that analyse the data to make predictions or generate insights, the output layer is where processed data is delivered to end users (This can include dashboards and reports) and the storage layer pertains to where the data is persistently stored after

processing. Now it's also important to consider that this is an assignment and not a real big data fraud detection pipeline so I will be making use of Python on Google colab to design and implement this system however I will explain and demonstrate my understanding of what a real world pipeline would look like. (This can be seen in Figure 4 to Figure 13)

System architecture Overview:

Although this isn't a real-time, cloud-deployed architecture, the following simplified pipeline mirrors a scalable ML system on a smaller scale:

1. Ingestion Layer
   - Tool Used: Python + pandas.read_csv() in Google Colab
   - Role: Load historical transaction data from a CSV file.
   - Details: Users upload synthetic_transaction_data.csv via Colab's files.upload(). This simulates a batch ingestion process.

2. Processing Layer
   - Tools Used: pandas, numpy
   - Role: Clean, transform, and enrich the data to prepare it for modeling.
   - Steps:
     - Missing Value Handling:
       - Numerical fields: imputed using median values.
       - Categorical fields: filled using mode (most frequent value).
     - Anomaly Removal:
       - Filter out transactions with negative amounts.
       - Remove rows with invalid timestamps.
     - Data Type Conversion:
       - Timestamps converted using pd.to_datetime().
     - Enrichment / Feature Engineering: (This can be seen in Figure 5, Figure 9 and Figure 11)
       - hour: Extracted from the timestamp.
       - day_of_week: Extracted from the timestamp.
       - is_night: Flag for transactions between 10 PM and 6 AM.
       - is_weekend: Flag for Saturday and Sunday.
       - is_high_amount: Flag for transactions over a custom threshold (e.g. £1000).
       - merchant_fraud_rate: % of frauds historically associated with each merchant.
   - Output: Cleaned and enriched dataset saved as enriched_transactions.csv.

3. Machine Learning Layer (This can be seen in Figure 8 and Figure 13)
   - Tools Used: scikit-learn, xgboost

- Role: Train, evaluate, and deploy fraud detection models.
- Steps:
    - Model Used: XGBClassifier with scale_pos_weight for class imbalance.
    - Feature Selection:
        - Numerical: amount, hour, merchant_fraud_rate
        - Binary: is_night, is_weekend, is_high_amount
        - Categorical: merchant, location
    - Preprocessing:
        - ColumnTransformer used to scale numeric features (StandardScaler), one-hot encode categorical variables (OneHotEncoder), and passthrough binary fields.
    - Train-Test Split:
    - 80/20 split with stratify=y to preserve fraud class proportions.
    - Evaluation Metrics:
        - Classification: Precision, Recall, F1-Score, Confusion Matrix
        - Regression-style: MSE, RMSE, MAE, R^2, Adjusted R^2
    - Performance:
        - Accuracy: ~98%
        - Recall for Fraud: ~80%
        - Precision for Fraud: ~52%
        - Significantly better than baseline Logistic Regression

4. Output Layer
- Tools Used: pandas, print statements, CSV writer
- Role: Output flagged transactions and fraud scores.
- Details:
    - Predicted fraud labels (predicted_is_fraud)
    - Fraud risk scores (fraud_score)
    - Suspicious flags for review (fraud_score > 0.7)
    - Final outputs saved to: streamed_fraud_predictions.csv

5. Storage Layer
- Tool Used: Colab's local file system (/content/)
- Role: Store cleaned, enriched, and scored datasets during session
- Limitation: Temporary; files are erased once the Colab session ends.
- Simulated Output Files:
    - Cleaned_transactions.csv
    - Enriched_transactions.csv
    - streamed_fraud_predictions.csv

I implemented two machine learning models for the fraud detection code, one was Logistic regression (a supervised machine learning algorithm used for classification tasks, it can be used to analyse the relationship between two data factors (geeksforgeeks, 2024)) and XGBoost classifier (a machine learning algorithm which builds an ensemble of decision trees sequentially where each new tree corrects the errors made by the previous one. (GeeksforGeeks, 2019)). (This can be seen in Figure 8 and Figure 13) My objective was to identify fraudulent transactions in a highly imbalanced dataset using python and google colab as this is just a college assignment. I also wanted to evaluate the performance of each model using appropriate classification and regression style metrics.

I started with Linear regression as a baseline model, it is a simple and effective linear classifier which will estimate the probability that a transaction is fraudulent or not. It is commonly used in binary classification problems and is highly valued for its level of interpretability and efficiency. I chose Linear regression because it provides a good benchmark and is easy to deploy but in my early results, I found that it completely failed to detect fraud without adjusting for the class imbalance. I believe this is because the fraud cases only made up for approximately 2% of the data so the model would have defaulted to predicting all transactions as "non-fraudulent" to maximise its accuracy, even when using class_weight='balanced', my models recall for fraud was still extremely low and this made it apparent to me that a more powerful model was required.

To solve this issue, I used XGBoost classifier which is a high performance, ensemble based machine learning model, its known for its high performance on structured and tabular datasets alongside its ability to handle class imbalance effectively through the scale_pos_weight parameter. I used scale_pos_weight=10 to aid the model to focus more on the minority fraud cases and specified eval_metric='logloss' for binary classification. I also fed the model my enriched features such as transaction hour and fraud rates which significantly improved the models detection of subtle fraud patterns.

I evaluated both models using accuracy, precision, recall, F1-score, and the confusion matrix—and regression-style metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R^2, and Adjusted R^2. These regression metrics were included for completeness, though classification metrics were more appropriate for our task. The XGBoost model showed a dramatic improvement over logistic regression. It achieved an overall accuracy of 98%, a recall of 80% for the fraud class (compared to 0% with logistic regression), and a precision of 52%, leading to an F1-score of 63% for fraud detection. These results demonstrate that XGBoost was far more effective in catching fraudulent transactions while maintaining an acceptable false positive rate.

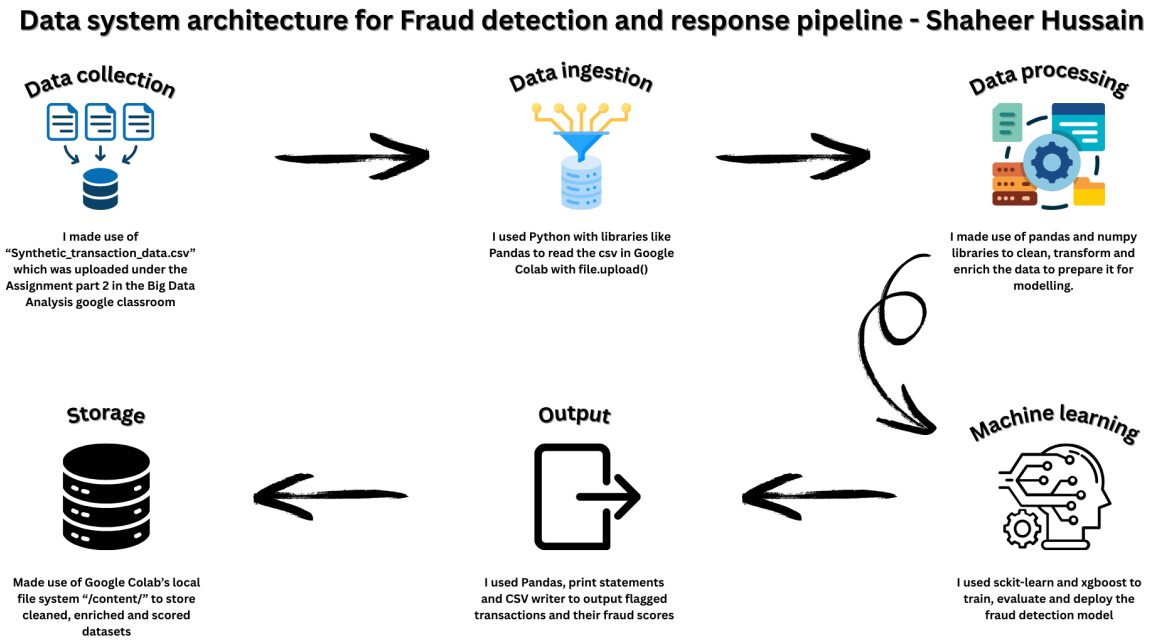


Figure 1 - system data architecture for the fraud detection and response pipeline

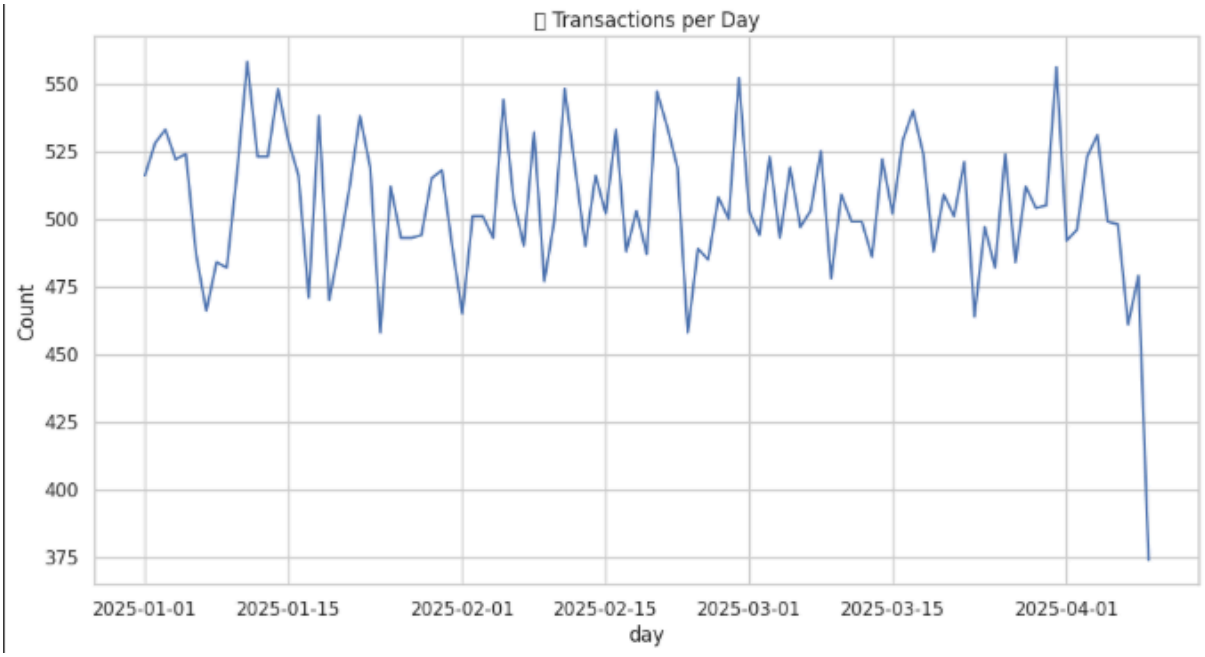


Figure 2 -Line graph visual of transactions per day - EDA

Figure 3 - Fraud bar graph from EDA

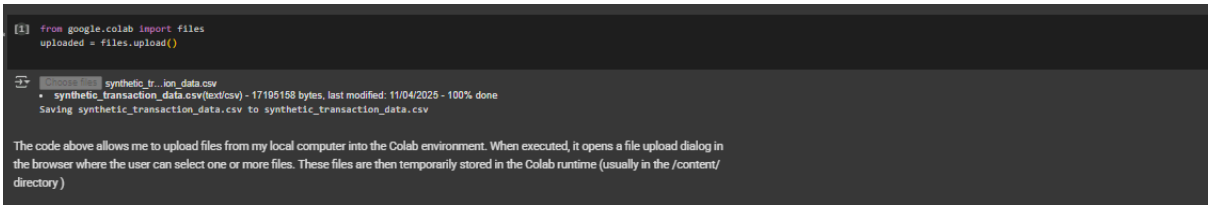


Figure 4 - Block of code for uploading synthetic data.

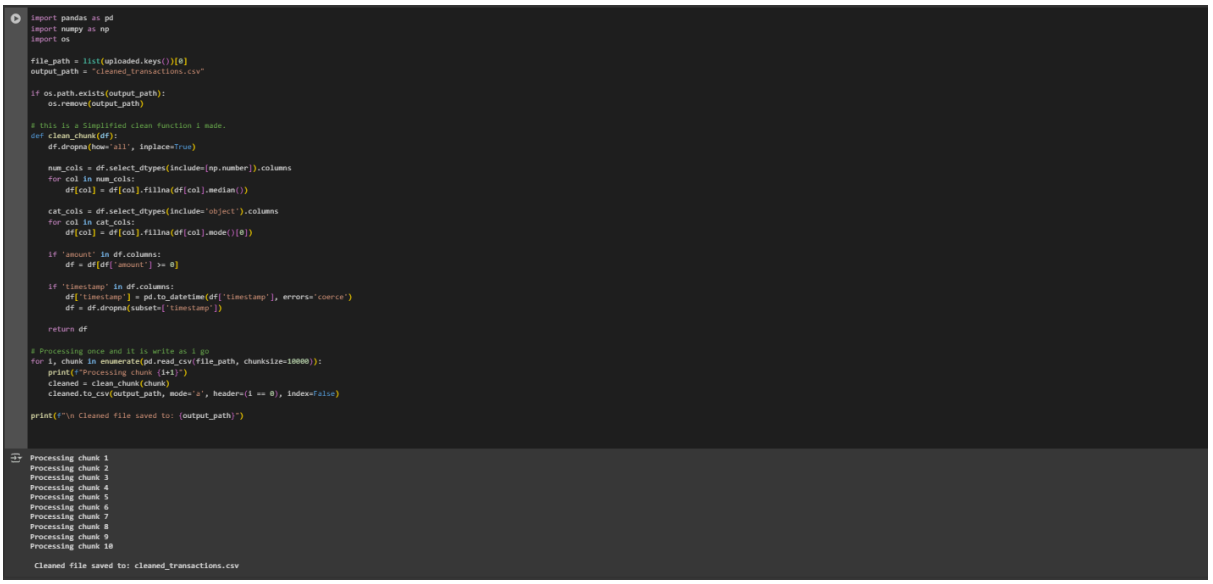


Figure 5 Data preprocessing / cleansing - The code above reads a large CSV file in chunks, cleans each chunk, and writes the cleaned data to a new file called cleaned_transactions.csv.

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

#here im Setting my plotting style
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (12, 6)

#Loading cleaned dataset
df = pd.read_csv("/content/cleaned_transactions.csv", nrows=50000)

#1. Quick overview
print(" Dataset shape:", df.shape)
print("\n Column types:\n")
print(df.dtypes)

#2. Missing values
missing = df.isnull().sum()
print("\n Missing values:\n")
print(missing[missing > 0])

#3. Basic stats
print("\n Summary statistics:\n")
display(df.describe())

#4. Timestamp analysis
if 'timestamp' in df.columns:
    df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
    df['day'] = df['timestamp'].dt.date
    daily_counts = df.groupby('day').size()

    daily_counts.plot(title="🗓️ Transactions per Day", ylabel="Count")
    plt.show()

#5. Transaction amount distribution
if 'amount' in df.columns:
    sns.histplot(df['amount'], bins=50, kde=True)
    plt.title("💰 Transaction Amount Distribution")
    plt.xlabel("Amount")
    plt.ylabel("Frequency")
    plt.show()

    df['log_amount'] = np.log1p(df['amount'])
    sns.histplot(df['log_amount'], bins=50, kde=True)
    plt.title("🔍 Log-Scaled Amount Distribution")
    plt.xlabel("Log(Amount + 1)")
    plt.ylabel("Frequency")
    plt.show()
```

```python
#6. Correlation heatmap
num_df = df.select_dtypes(include=[np.number])
if not num_df.empty:
    corr = num_df.corr()
    sns.heatmap(corr, cmap="coolwarm", annot=False, square=True)
    plt.title("🔗 Correlation Heatmap")
    plt.show()

#7. Check for fraud label (if present)
fraud_cols = [col for col in df.columns if 'fraud' in col.lower()]
if fraud_cols:
    fraud_col = fraud_cols[0]
    print(f"\n⚠️ Fraud column detected: {fraud_col}")
    print(df[fraud_col].value_counts(normalize=True) * 100)

    sns.countplot(data=df, x=fraud_col)
    plt.title("Fraud vs Non-Fraud")
    plt.xlabel("Fraud")
    plt.ylabel("Count")
    plt.show()

    #Boxplot of amount by fraud
    if 'amount' in df.columns:
        sns.boxplot(data=df, x=fraud_col, y='amount')
        plt.title("💳 Transaction Amount by Fraud Status")
        plt.show()
else:
    print("\nℹ️ No fraud label found. We'll use unsupervised anomaly detection.")
```

Figure 6 - Exploratory data analysis code resulting in figure 2-3

```
Dataset shape: (50000, 8)

Column types:

transaction_id     object
timestamp          object
customer_id        object
amount            float64
merchant           object
location           object
device_id          object
is_fraud            int64
dtype: object

Missing values:

Series([], dtype: int64)

Summary statistics:
```

	amount	is_fraud
count	50000.000000	50000.000000
mean	2506.843593	0.020080
std	1441.271035	0.140275
min	1.010000	0.000000
25%	1256.000000	0.000000
50%	2510.110000	0.000000
75%	3756.695000	0.000000
max	4999.850000	1.000000

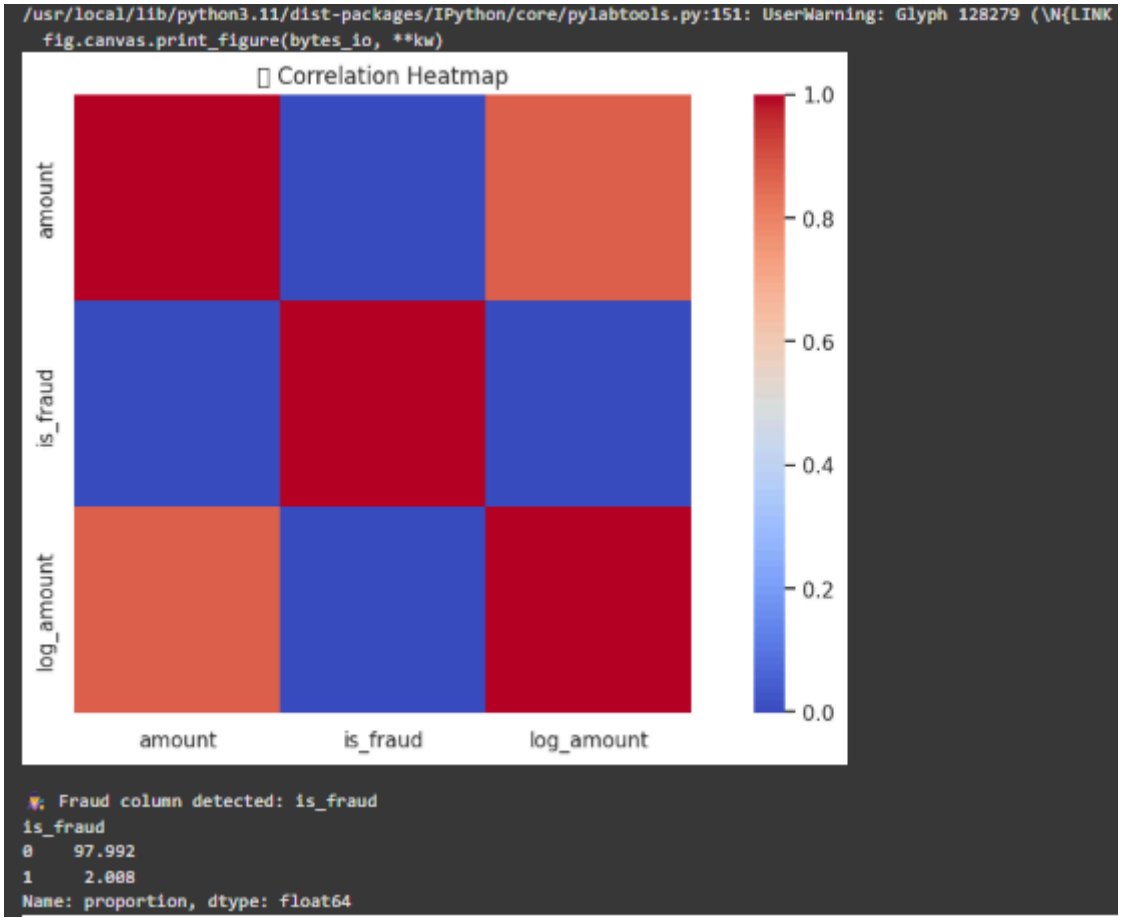


Figure 7 - Text Output for code in figure 6

```
[ ]  import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
     from sklearn.pipeline import Pipeline
     from sklearn.compose import ColumnTransformer
     from sklearn.preprocessing import StandardScaler, OneHotEncoder
     from sklearn.impute import SimpleImputer
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, confusion_matrix, classification_report

     #Loading cleaned data
     df = pd.read_csv("/content/cleaned_transactions.csv")

     #Target column.
     target = 'is_fraud'

     #Here are my Suggested features (this will filter to ones that exist)
     suggested_numerical = ['amount']
     suggested_categorical = ['merchant', 'category', 'location']

     #Convert timestamp to hour if available
     if 'timestamp' in df.columns:
         df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
         df['hour'] = df['timestamp'].dt.hour
         suggested_numerical.append('hour')

     #Filter only existing columns
     numerical_features = [col for col in suggested_numerical if col in df.columns]
     categorical_features = [col for col in suggested_categorical if col in df.columns]

     #Drop rows with missing values in required columns
     required_cols = numerical_features + categorical_features + [target]
     df = df.dropna(subset=required_cols)

     #Split features and labels
     X = df[numerical_features + categorical_features]
     y = df[target]

     #Preprocessing pipeline
     numeric_transformer = Pipeline(steps=[
         ('imputer', SimpleImputer(strategy='median')),
         ('scaler', StandardScaler())
     ])

     categorical_transformer = Pipeline(steps=[
         ('imputer', SimpleImputer(strategy='most_frequent')),
         ('encoder', OneHotEncoder(handle_unknown='ignore'))
     ])

     preprocessor = ColumnTransformer(
         transformers=[
             ('num', numeric_transformer, numerical_features),
             ('cat', categorical_transformer, categorical_features)
         ]
     )
```

```python
#Complete ML pipeline
clf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000, class_weight='balanced'))
])

#Split data.
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)

#Train model
clf_pipeline.fit(X_train, y_train)

#Predict.
y_pred = clf_pipeline.predict(X_test)

#my evaluation metrics to determine accuracy of model - implemented from ML college week
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
n = len(y_test)
p = X_test.shape[1]
adj_r2 = 1 - ((1 - r2) * (n - 1) / (n - p - 1))

#Display the results
print(" Evaluation Metrics:")
print(f"  MSE: {mse:.4f}")
print(f"  RMSE: {rmse:.4f}")
print(f"  MAE: {mae:.4f}")
print(f"  R²: {r2:.4f}")
print(f"  Adjusted R²: {adj_r2:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred))

print(" Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
 Evaluation Metrics:
  MSE: 0.0401
  RMSE: 0.2002
  MAE: 0.0401
  R²: -1.0018
  Adjusted R²: -1.0022

 Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     19591
           1       0.01      0.01      0.01       409

    accuracy                           0.96     20000
   macro avg       0.50      0.50      0.50     20000
weighted avg       0.96      0.96      0.96     20000
```

Figure 8 - Attempt at using logistic regression on dataset

```python
#here im adding fraud probability scores to test set
X_test_copy = X_test.copy()  # So we can safely inspect it
X_test_copy["actual_is_fraud"] = y_test.values
X_test_copy["fraud_score"] = clf_pipeline.predict_proba(X_test)[:, 1]
X_test_copy["predicted_is_fraud"] = clf_pipeline.predict(X_test)

#Defining threshold (adjustable)
threshold = 0.7
X_test_copy["flagged_as_suspicious"] = X_test_copy["fraud_score"] > threshold

#Show the flagged transactions
flagged = X_test_copy[X_test_copy["flagged_as_suspicious"] == True]
print(f"\n Flagged {len(flagged)} high-risk transactions (score > {threshold}):")
display(flagged.sort_values("fraud_score", ascending=False).head(10))
```

```
 Flagged 106 high-risk transactions (score > 0.7):
```

	amount	hour	merchant	location	actual_is_fraud	fraud_score	predicted_is_fraud	flagged_as_suspicious
54370	67.56	13	Fletcher LLC	Timothyport	0	0.957875	1	True
71066	1735.18	23	Johnston Ltd	Lake Taylor	0	0.939975	1	True
16912	674.02	10	Oliver LLC	South Annette	0	0.937157	1	True
76486	4378.02	23	Walker LLC	Micheleton	0	0.936021	1	True
14955	2828.03	14	Kelley LLC	Sarahberg	0	0.933429	1	True
38641	1528.09	6	Hensley PLC	Joshuaview	0	0.932812	1	True
96643	2324.42	18	Patterson PLC	South Dawn	0	0.929705	1	True
99846	589.98	0	Martinez Ltd	East Breanna	0	0.926462	1	True
47723	4156.97	12	Alexander and Sons	Ginaborough	0	0.926082	1	True
16513	1335.36	22	Davis LLC	North Johnmouth	0	0.922181	1	True

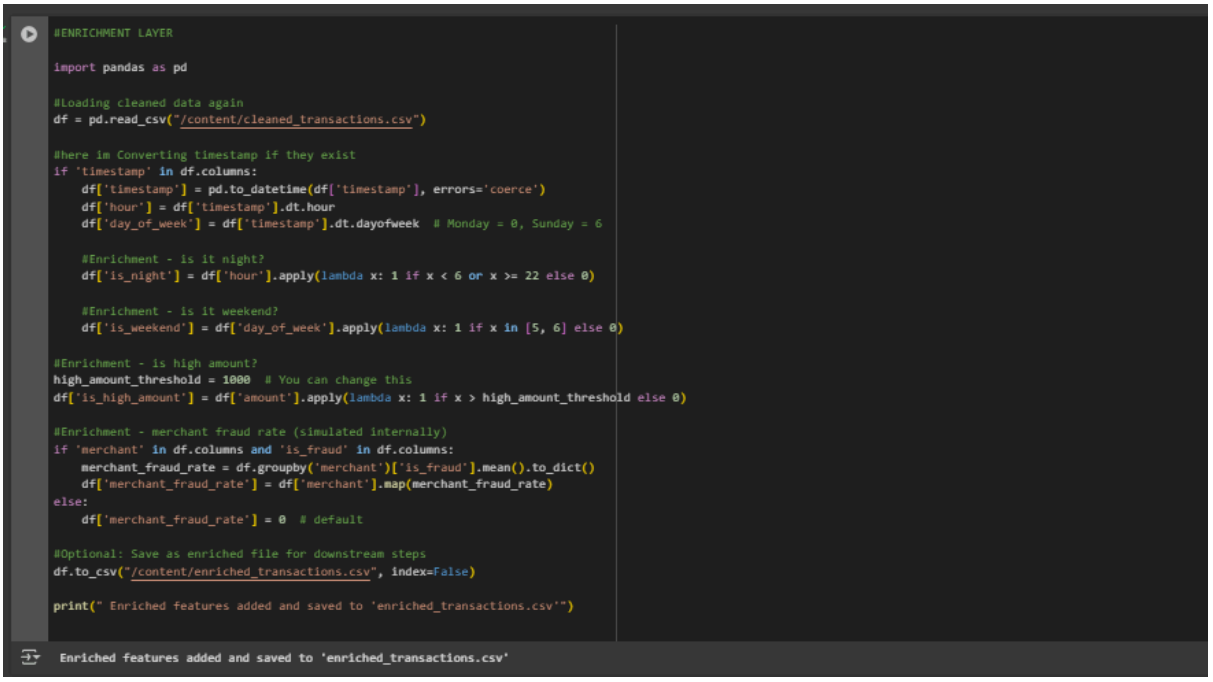
Figure 9 - Assigning fraud probability scores and flagging high risk transactions within the dataset

```python
import pandas as pd

#Create synthetic test transactions
sample_data = pd.DataFrame([
    {
        "amount": 5.00,  #small transaction
        "merchant": "Amazon",
        "location": "London",
        "hour": 14
    },
    {
        "amount": 5000.00,  #large transaction
        "merchant": "CryptoExchangeX",
        "location": "Dubai",
        "hour": 3  #odd hour
    },
    {
        "amount": 100.00,
        "merchant": "Supermarket",
        "location": "Birmingham",
        "hour": 11
    },
    {
        "amount": 12000.00,
        "merchant": "LuxuryWatchShop",
        "location": "Zurich",
        "hour": 2
    }
])

#Using the trained pipeline to get predictions
sample_data["fraud_score"] = clf_pipeline.predict_proba(sample_data)[:, 1]
sample_data["predicted_is_fraud"] = clf_pipeline.predict(sample_data)

#Showing the results
print(" Manual Test Transactions and Fraud Predictions:")
display(sample_data.sort_values("fraud_score", ascending=False))
```

```
 Manual Test Transactions and Fraud Predictions:
```

	amount	merchant	location	hour	fraud_score	predicted_is_fraud
0	5.0	Amazon	London	14	0.089798	0
2	100.0	Supermarket	Birmingham	11	0.089767	0
1	5000.0	CryptoExchangeX	Dubai	3	0.088135	0
3	12000.0	LuxuryWatchShop	Zurich	2	0.085851	0

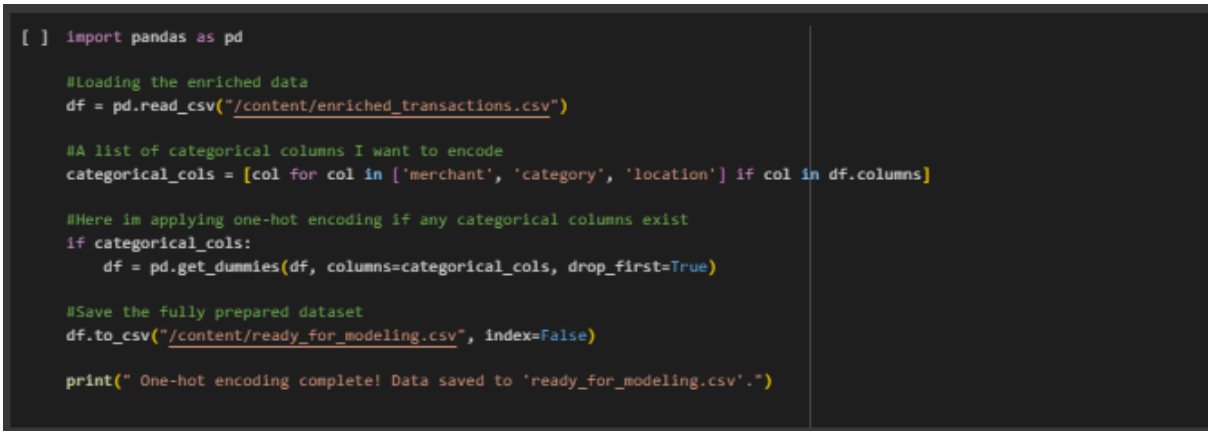
Figure 10 - Creation and testing of small synthetic dataset to understand how my model would react.



```
#ENRICHMENT LAYER

import pandas as pd

#Loading cleaned data again
df = pd.read_csv("/content/cleaned_transactions.csv")

#here im Converting timestamp if they exist
if 'timestamp' in df.columns:
    df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
    df['hour'] = df['timestamp'].dt.hour
    df['day_of_week'] = df['timestamp'].dt.dayofweek  # Monday = 0, Sunday = 6

    #Enrichment - is it night?
    df['is_night'] = df['hour'].apply(lambda x: 1 if x < 6 or x >= 22 else 0)

    #Enrichment - is it weekend?
    df['is_weekend'] = df['day_of_week'].apply(lambda x: 1 if x in [5, 6] else 0)

#Enrichment - is high amount?
high_amount_threshold = 1000  # You can change this
df['is_high_amount'] = df['amount'].apply(lambda x: 1 if x > high_amount_threshold else 0)

#Enrichment - merchant fraud rate (simulated internally)
if 'merchant' in df.columns and 'is_fraud' in df.columns:
    merchant_fraud_rate = df.groupby('merchant')['is_fraud'].mean().to_dict()
    df['merchant_fraud_rate'] = df['merchant'].map(merchant_fraud_rate)
else:
    df['merchant_fraud_rate'] = 0  # default

#Optional: Save as enriched file for downstream steps
df.to_csv("/content/enriched_transactions.csv", index=False)

print(" Enriched features added and saved to 'enriched_transactions.csv'")
```

```
Enriched features added and saved to 'enriched_transactions.csv'
```

Figure 11 - Enrichment layer used to extract details like time and day of week from the dataset to gather more important features.



```
[ ] import pandas as pd

    #Loading the enriched data
    df = pd.read_csv("/content/enriched_transactions.csv")

    #A list of categorical columns I want to encode
    categorical_cols = [col for col in ['merchant', 'category', 'location'] if col in df.columns]

    #Here im applying one-hot encoding if any categorical columns exist
    if categorical_cols:
        df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

    #Save the fully prepared dataset
    df.to_csv("/content/ready_for_modeling.csv", index=False)

    print(" One-hot encoding complete! Data saved to 'ready_for_modeling.csv'.")
```

Figure 12 - Addition of one hot encoding on categorical data within the dataset.

```
#COMPLICATED XBBOOST MODEL

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import (
    mean_squared_error, mean_absolute_error, r2_score,
    confusion_matrix, classification_report
)
from xgboost import XGBClassifier

#Load the enriched dataset
df = pd.read_csv("/content/ready_for_modeling.csv")

#Define target.
target = 'is_fraud'

#Identify feature columns.
numerical_features = ['amount', 'hour', 'merchant_fraud_rate']
binary_features = ['is_high_amount', 'is_night', 'is_weekend']
categorical_features = [col for col in ['merchant', 'location'] if col in df.columns]

#Droping the missing values in key columns .
df = df.dropna(subset=numerical_features + binary_features + categorical_features + [target])

#Features and labels
X = df[numerical_features + binary_features + categorical_features]
y = df[target]

#Split train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)

#Preprocessing
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

```
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numerical_features),
        ('bin', 'passthrough', binary_features),  #binary values already clean.
        ('cat', categorical_transformer, categorical_features)
    ]
)

#Building the pipeline with XGBoost (adjust scale_pos_weight if class imbalance is high)
clf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', XGBClassifier(
        use_label_encoder=False,
        eval_metric='logloss',
        scale_pos_weight=10,  # Adjusted for class imbalance
        random_state=42
    ))
])

#Train
clf_pipeline.fit(X_train, y_train)

#Predict.
y_pred = clf_pipeline.predict(X_test)
y_pred_proba = clf_pipeline.predict_proba(X_test)[:, 1]

#Evaluation metrics to test accuracy of model.
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
n = len(y_test)
p = X_test.shape[1]
adj_r2 = 1 - ((1 - r2) * (n - 1) / (n - p - 1))

#Outputting my metrics
print(" Evaluation Metrics:")
print(f"  MSE: {mse:.4f}")
print(f"  RMSE: {rmse:.4f}")
print(f"  MAE: {mae:.4f}")
print(f"  R²: {r2:.4f}")
print(f"  Adjusted R²: {adj_r2:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred))

print(" Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

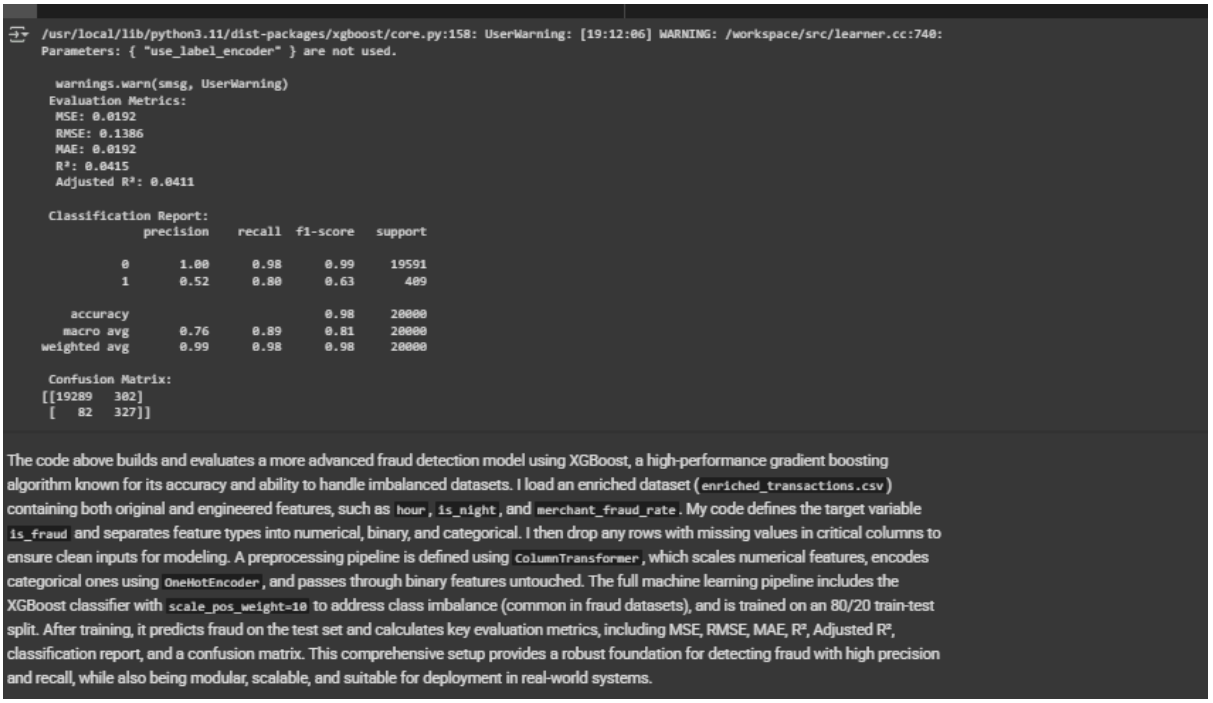


Figure 13 - Implementation of XGBOOST model

In a real world scenario, my fraud detection pipeline would obviously need to be capable of handling ingestion, processing (Which is also scalable), modelling and immediate response capabilities. This would mean that we'd move from the simple approach of CSV's and Colab to a distributed big data architecture, if I built such a system/pipeline then I would start with Apache Kafka for the ingestion layer, this platform is a highly scalable and fault tolerant message broker (which is a software that allows applications, systems etc to communicate with each other (IBM, 2021)) which would stream the incoming transaction data from the relevant data sources like online banking portals for example. each transaction would be published to a Kafka topic in real time (e.g. "transaction_stream") and this would allow the downstream components to consume and process the data with minimal delay.

Once the transaction data is ingested, the data would flow into the processing layer where I would choose Apache Spark structured streaming, this would become the real time analytics engine by consuming messages from the kafka topic in batches. During this process, Spark would be used to perform key data preparation tasks such as cleaning null values, correcting data types and handling anomalies like negative amounts (in other words the cleansing and transformation of data, like my "Cleaned_transactions.csv"). Spark can also take on the feature engineering by taking key insights from the raw or unprocessed data like the transaction hour or fraud history of merchants which would be calculated from past data. These transformations would be coded in PySpark and carried out in a parallelised manner to make sure there is minimal latency even as the transactions volume scales.

The following processed data would be passed to the machine learning layer where I could make use of Spark MLlib for a in-cluster training pipeline or Amazon SageMaker for a managed, cloud based training and deployment service.If I stick with Spark MLlib then I have a range of models like Random forest, Logistic regression and even Gradient boosted trees to choose from as they are directly within the Spark cluster. On the other hand, if I stick with AWS SageMaker then I could offload the model training to more powerful cloud instances and deploy the model as a REST API endpoint(REST API is a type of API that allows communication between different systems over the internet (Bansal, 2018)). SageMaker also supports hyperparameter tuning, AutoML and monitoring tools to ensure that the deployed model performs accurately/reliably in a production environment. Once deployed, the model would score each transaction in real time, returning both a fraud probability score and a binary classification.

All raw data, processed data and model outputs would need a storage layer and I would choose between HDFS, AWS S3 or Azure Data Lake storage, these are cloud based solutions which provide high durability and more importantly, high scalability for audit logs, model training datasets and historical records. The archived data could also be used to retrain and improve models in the future.

Critical reflection - Colab and Python / Real world scenario:

When using python in google colab, it provided a highly accessible and low barrier environment to develop a sort of first stage pipeline where I can develop, test and evaluate the fraud detection machine learning pipeline. The coding environment allowed for rapid prototyping and visualisation on the dataset "Synthetic_transaction.csv" without needing a complex infrastructure e.g. Kafka and Spark etc. I used sckit-learn for preprocessing and modelling, XGBoost for an improved predictive performance. The key strengths of this approach is the ease of use, strong library support as well as the visualisation capacity, all of this made it possible to handle the cleaning, enrichment, modelling and scoring processes in the same notebook.

I must state that this approach has obvious limitations, data and trained models in notebooks are lost after a disconnect, there's no proper support for streaming on real time data, the memory limits and performance bottlenecks will affect the processing of larger datasets or even deploying more complex models. Also Colab is not a production pipeline, it is a development environment, there's no automated model deployment, scheduling, monitoring or built in alerts. These are all critical for a real fraud detection pipeline.

Despite these limitations, the Colab solution demonstrates real business impact in proving feasibility. It can be shared across teams, visualised by non-technical stakeholders, and used to educate or validate fraud detection concepts quickly. It serves as an excellent proof of concept that highlights potential for scaling, if transitioned to a production ready stack.

On the other hand, a real world system / pipeline which uses Apache Kafka, Apache Spark, MLlib / AWS SageMaker, PowerBI and HDFS/S3/ADLS provides the scalability and robustness that is needed for a real enterprise grade fraud detection system. Kafka ensures real-time ingestion of streaming transactions, while Spark enables distributed and fault-tolerant processing. Feature enrichment, anomaly detection, and scoring can be handled efficiently in-memory and in parallel. Machine learning models can be trained with Spark MLlib or externally on AWS SageMaker and then deployed as APIs for real-time inference. The output can be stored in scalable cloud storage like S3 or ADLS and visualised through dashboards on PowerBI.

This sort of architecture is built for resilience, monitoring and more importantly, automation. It will support continuous learning / training (Or retraining) and a scalable alert system which will notify any analysts or block the transactions in real time. The trade off with this approach is the level of complexity, this will require expertise and deep understanding in DevOps, distributed computing, cloud infrastructure and data engineering. Setting this up will have a significant cost, time and overhead for operation, this may not make sense for an early stage where I am experimenting.

The enterprise setup is essential for mission-critical applications in financial services. It ensures 24/7 operation, auditability, compliance, and the ability to respond to fraud within seconds. It reduces human intervention and improves fraud loss prevention, customer trust, and regulatory standing.

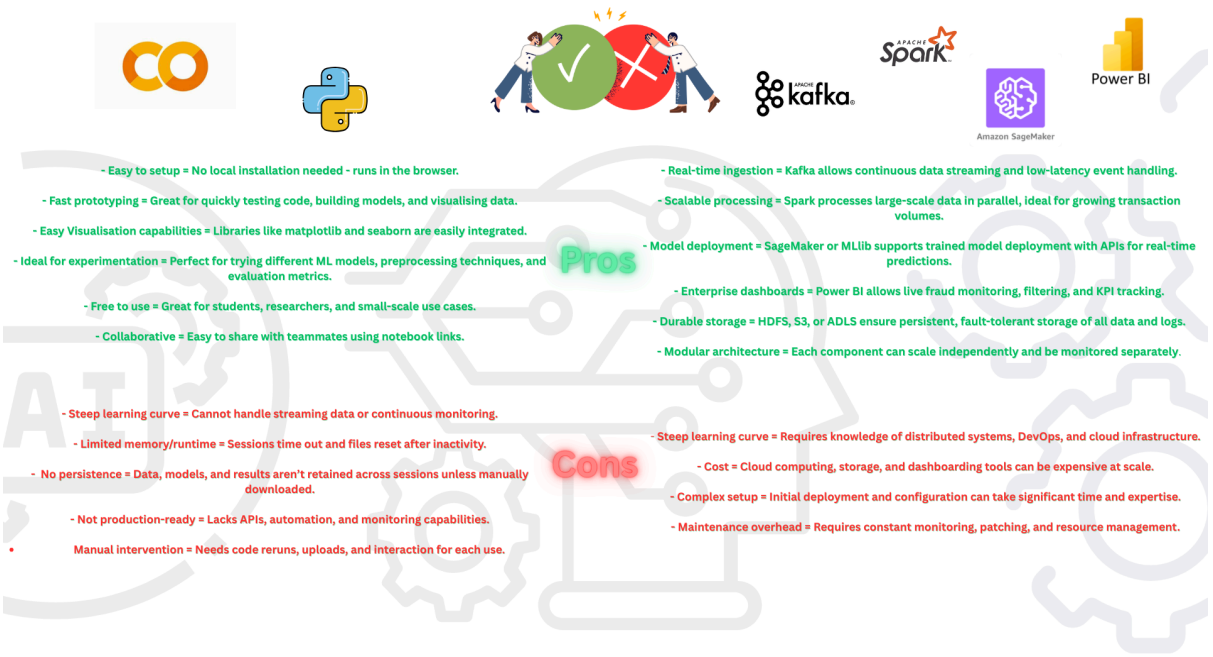


Figure 14 - pros and cons of both pipelines - real and synthetic

Conclusion :

The conclusion of this project is there was a successful design, implementation and evaluation of a big data fraud detection pipeline which addressed the needs for real time transaction monitoring in the financial sector. I started with a clear objective which was to detect and respond to potentially fraudulent behaviour. My technical report outlines the journey from data ingestion to storage both within the boundaries of this assignment and how it would look like in a real world scenario. I used Python on Google colab to develop and test an effective prototype i was confident in, a prototype capable of cleaning, enriching and analysing the transactional data from “Synthetic_transaction_data.csv” with a sort of transition plan moving towards a more production suitable and ready architecture that involved tools like Kafka, Spark, SageMaker, Power BI, and cloud storage.

I applied a structured data processing workflow that included missing value handling, anomaly detection, timestamp parsing, and advanced feature engineering such as time-based indicators and merchant-level fraud rates.These features significantly improved the contextual understanding of each transaction and just made it easier for me to break the data down. I used machine learning models like logistic regression and XGBoost which were trained, evaluated, and compared. XGBoost demonstrated strong recall and precision when tuned for class imbalance, making it suitable for detecting rare but high-impact fraudulent activity. Performance metrics such as precision , recall , F1-score, and confusion matrices validated the pipelines effectiveness.

References :

GeeksforGeeks (2019). ML | What is Machine Learning ? - GeeksforGeeks. [online] GeeksforGeeks.
Available at: https://www.geeksforgeeks.org/ml-machine-learning/.
GeeksforGeeks (2024). What is Data Ingestion? [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/what-is-data-ingestion/.
Stedman, C. (2024). What is data architecture? A data management blueprint. [online] Available at: https://www.techtarget.com/searchdatamanagement/definition/What-is-data-architecture-A-data-management-blueprint
IBM (2021). Message Brokers. [online] Ibm.com. Available at: https://www.ibm.com/think/topics/message-brokers.
Bansal, S. (2018). *REST API (Introduction)*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/rest-api-introduction/.
geeksforgeeks (2024). *Understanding Logistic Regression*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/understanding-logistic-regression/.
GeeksforGeeks. (2019). *ML | XGBoost (eXtreme Gradient Boosting)*. [online] Available at: https://www.geeksforgeeks.org/ml-xgboost-extreme-gradient-boosting/.