BIA-678 – Big Data Technologies - Project

Spark Sherlock:
Catching Fraudsters Faster than They Swipe!

Group Members:

Pooja Pande (20035197)

Vidhi Palan (20031836)

Pratik Kale (20036601)

Agenda

- Problem Statement
- Dataset Overview & Data Dictionary
- Data Pre-processing & Feature Engineering
- Exploratory Data Analysis(EDA)
- Solution Approach & Modeling Steps
- Performance Metrics
- Hyperparameter Tuning & Performance Showcase
- Scale Up & Scale Out
- Application UI
- Conclusion & Future Enhancements

Problem Statement

Problem:

- Financial fraud in mobile money transactions is rapidly increasing.
- High volume and velocity of transactions make fraud detection challenging.
- Traditional approaches struggle to predict with variables other than “amount”.

Objective:

- Develop a scalable fraud detection model using big data technology (PySpark).
- Improve detection accuracy to minimize financial losses.

Dataset Overview

Dataset:

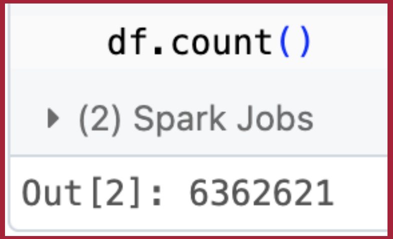
- Synthetic Mobile Money Transaction Dataset

Source:

- Synthetic data created by a wise guy with reference to African Real Money Transactions

Format & Size:

- CSV file format
- **Size = ~ 6.3 million rows**

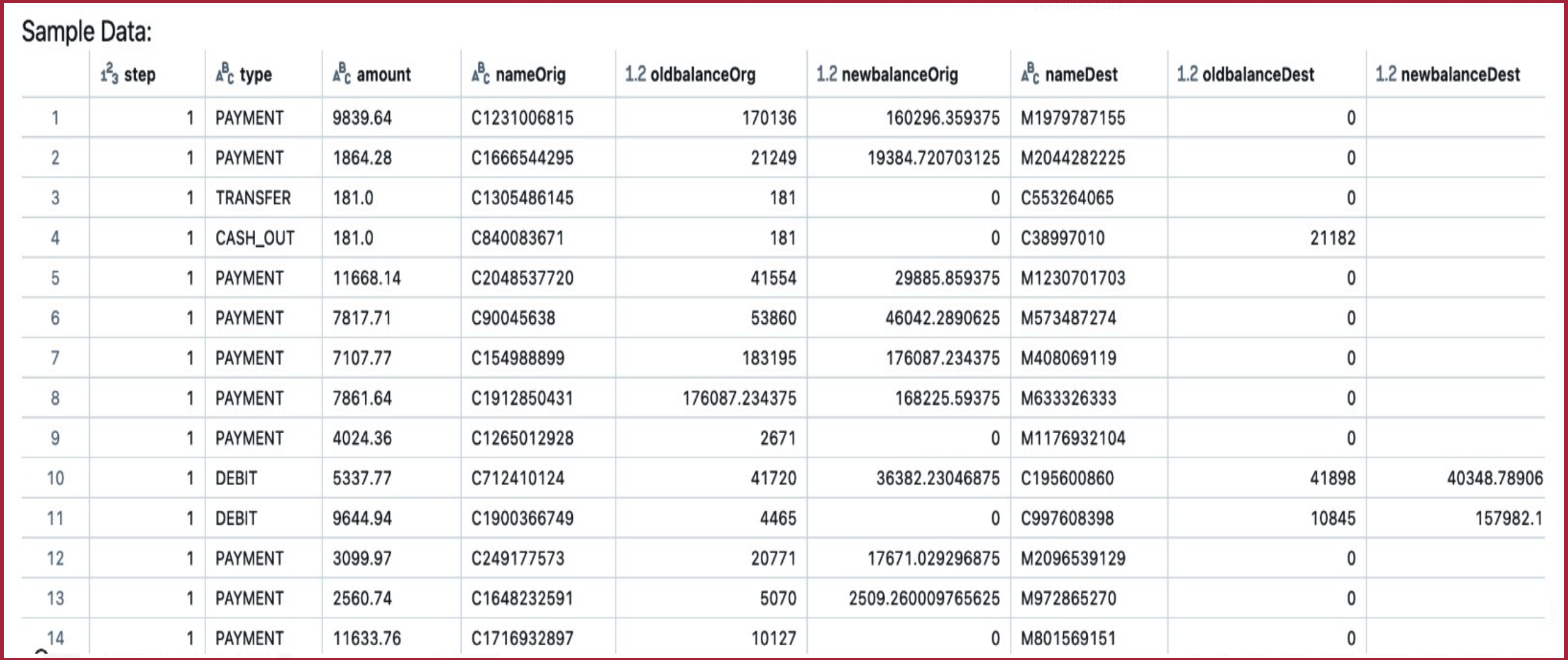


```
df.count()
▸ (2) Spark Jobs
Out[2]: 6362621
```

Domain:

- Financial services, specifically mobile money transactions.

Sample Data



Sample Data:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest
1	1	PAYMENT	9839.64	C1231006815	170136	160296.359375	M1979787155	0	
2	1	PAYMENT	1864.28	C1666544295	21249	19384.720703125	M2044282225	0	
3	1	TRANSFER	181.0	C1305486145	181	0	C553264065	0	
4	1	CASH_OUT	181.0	C840083671	181	0	C38997010	21182	
5	1	PAYMENT	11668.14	C2048537720	41554	29885.859375	M1230701703	0	
6	1	PAYMENT	7817.71	C90045638	53860	46042.2890625	M573487274	0	
7	1	PAYMENT	7107.77	C154988899	183195	176087.234375	M408069119	0	
8	1	PAYMENT	7861.64	C1912850431	176087.234375	168225.59375	M633326333	0	
9	1	PAYMENT	4024.36	C1265012928	2671	0	M1176932104	0	
10	1	DEBIT	5337.77	C712410124	41720	36382.23046875	C195600860	41898	40348.78906
11	1	DEBIT	9644.94	C1900366749	4465	0	C997608398	10845	157982.1
12	1	PAYMENT	3099.97	C249177573	20771	17671.029296875	M2096539129	0	
13	1	PAYMENT	2560.74	C1648232591	5070	2509.260009765625	M972865270	0	
14	1	PAYMENT	11633.76	C1716932897	10127	0	M801569151	0	

Data Dictionary

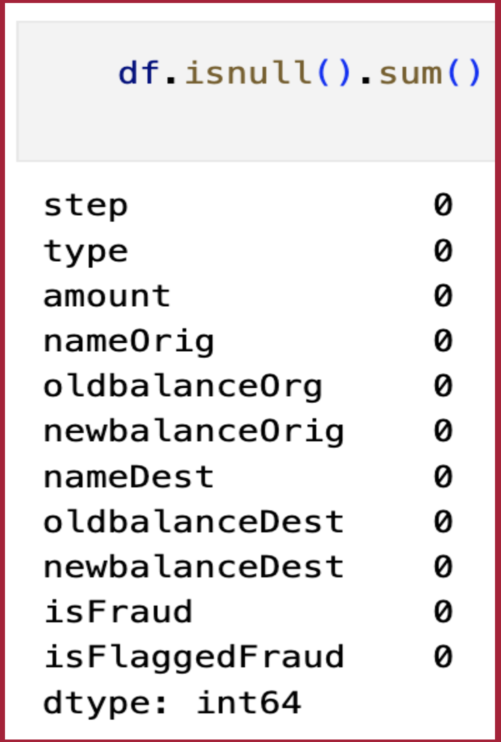
Key Columns:

- Transaction Type (e.g., deposit, withdrawal, payment)
- Amount
- Sender and Receiver IDs (nameOrig & nameDest)
- Sender Balance Before and After Transaction
- Receiver Balance Before and After Transaction
- Is Fraud (target variable)

Target Variable:

- Binary (0: Genuine transaction, 1: Fraudulent transaction)

Data Pre-processing



```
df.isnull().sum()
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

Data Cleaning & Transformation:

- Dropping irrelevant columns (step, isFlaggedFraud)
- Converting categorical variables to numerical indices for **Transaction Type** (String -> Integer)

Handling Missing Values:

- Initial dataset inspection for missing values.
- Appropriate handling strategies (imputation/removal).

Feature Engineering

Balance Change Features

- **ΔOrig** = oldBalanceOrg – newBalanceOrig
- **ΔDest** = newBalanceDest – oldBalanceDest
- **Amount-to-Balance Ratios**
- **Amount/Orig Balance** = amount / (oldBalanceOrg + 1)
- **Amount/Dest Balance** = amount / (oldBalanceDest + 1)

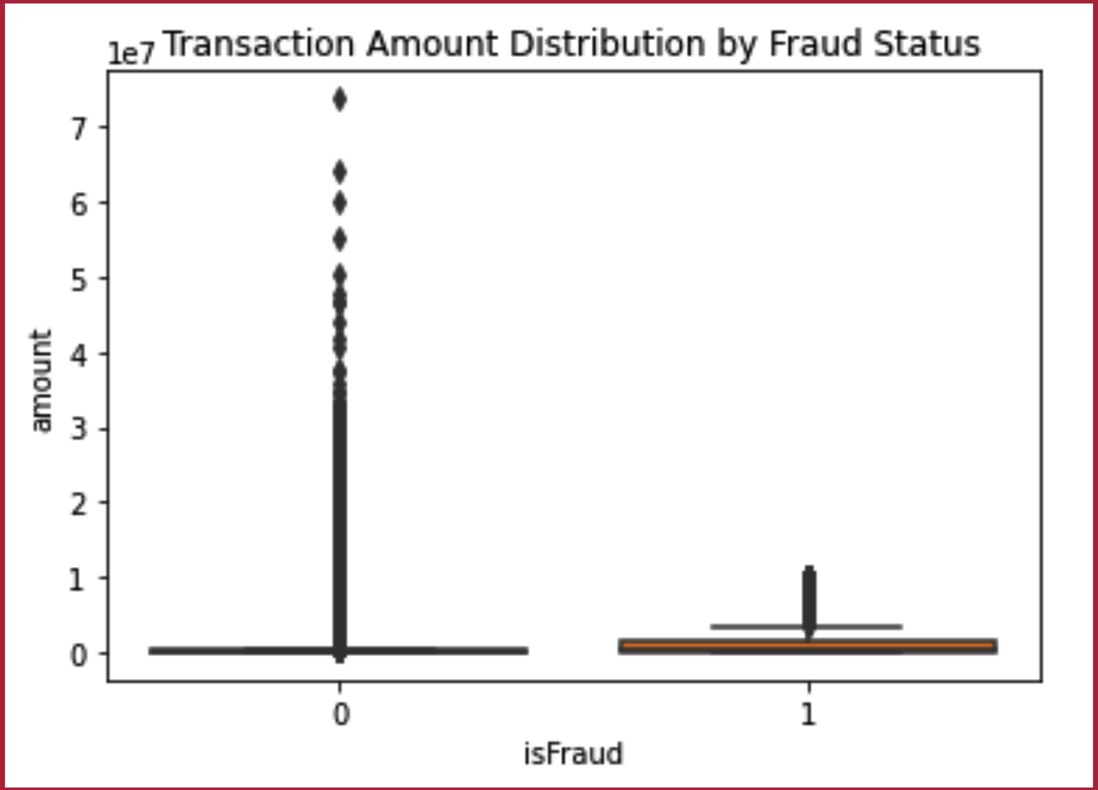
Categorical Encoding

- Indexed transaction **type** → `type_idx` (StringIndexer)
- One-hot encoded `type` → `type_vec` (OneHotEncoder)

Vector Assembly & Scaling

- Assembled all features into a single feature vector
- Applied StandardScaler to normalize feature ranges

EDA just started...



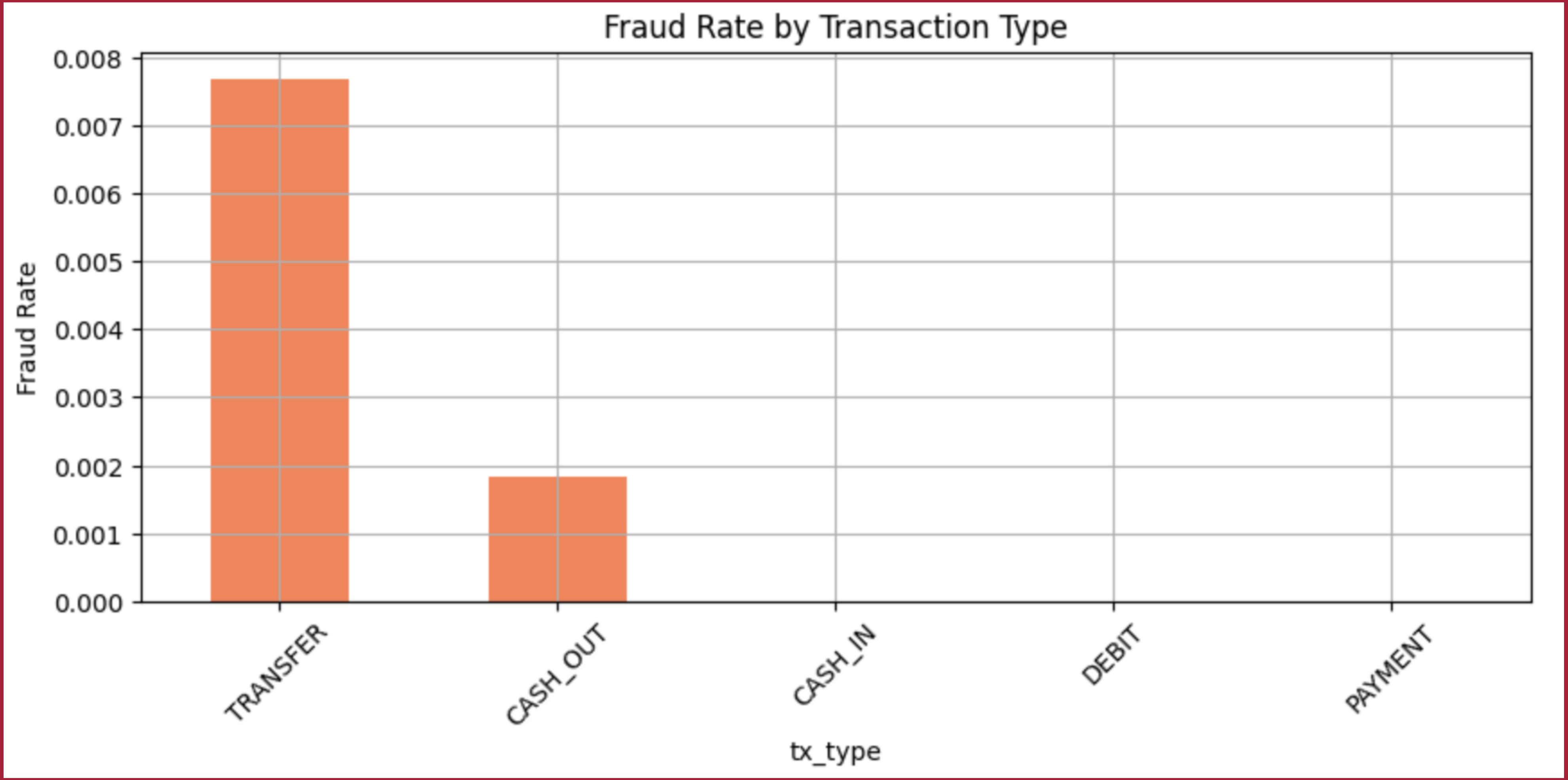
Observation:

- The amount transacted by fraudsters is usually small.
- Outliers are not necessarily fraudulent.

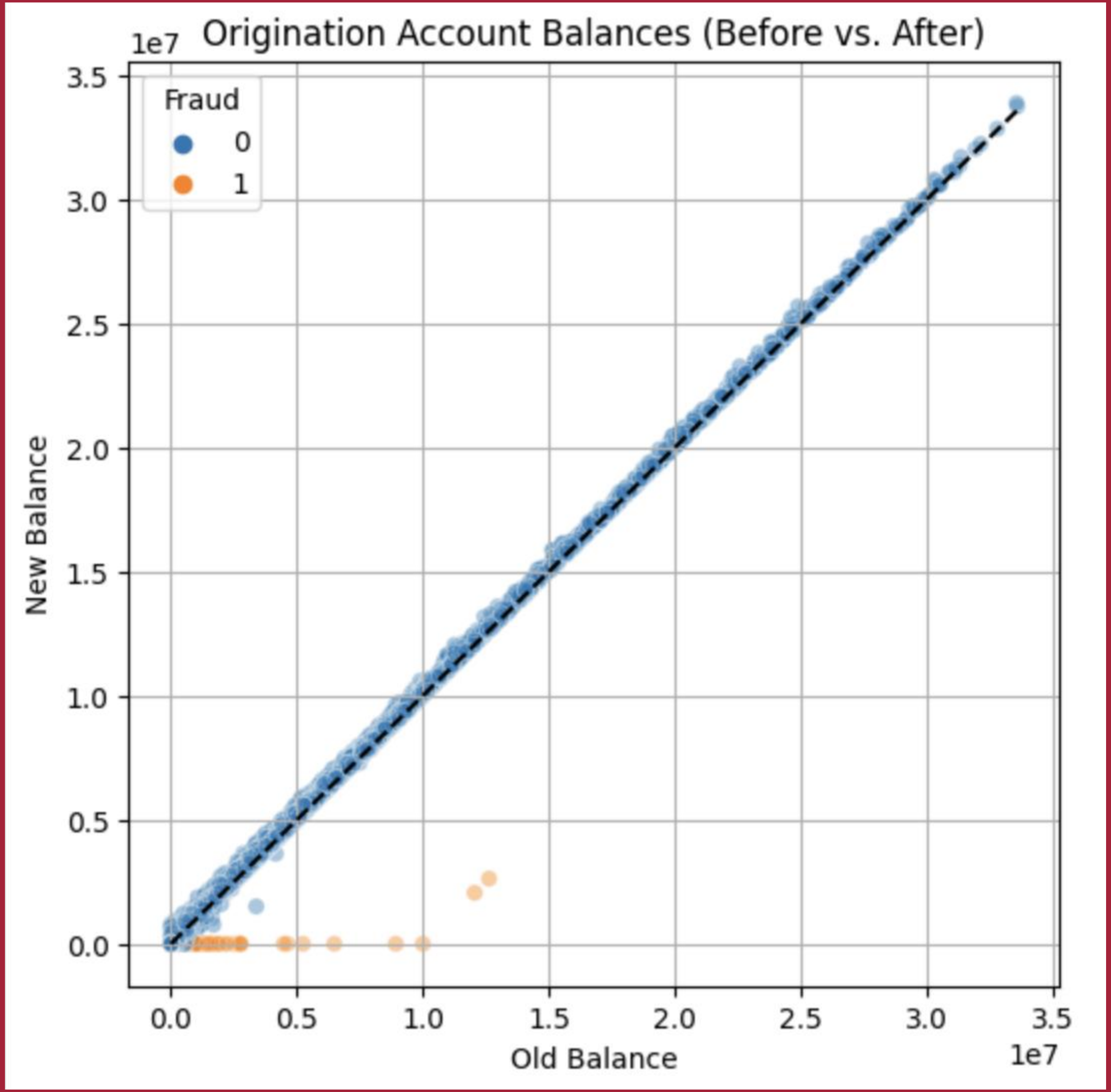
Approach:

- Use more robust model to look for anomalies.
- Look for transaction's behavior pattern

EDA - Fraud Rate by Transaction Type



EDA- Origination Account Balances



Legitimate Transactions (Blue):

- Blue dots shows a normal behavior; balance goes down when money is sent.

Fraudulent Transactions (Orange):

- Many orange dots lie **below the diagonal and close to zero**, meaning:
  - **Large amounts withdrawn**
  - **Accounts often drained to zero**

- Some frauds even have **old balances but zero new balances → suspicious wipeout** pattern.

Modeling Steps

Feature Selection:

- Transaction type, Transaction amount, sender/receiver balances, derived amount to balance ratio.

Train-Test Split:

- 80% Training, 20% Testing

Class Balance Preparation:

- Applied stratified sampling and resampling techniques (here SMOTE oversampling for the minority) to ensure the training set reflects realistic fraud ratios..

Hyperparameter Tuning:

- We used Optuna Library to tune the model over multiple parameters

Solution kicks in..

- Logistic Regression
- Random Forest
- Isolation Forest

Modeling Algorithm:

- Random Forest Classifier

Reason for Choosing Random Forest:

- Robustness to outliers
- Handled imbalanced datasets effectively
- Parallelizable, making it suitable for big data scenarios

Performance Metrics

Primary Metric:

- F1-score (coz I'm Sherlock!)

Reason for Choosing F1-score:

- Balanced precision (false alarms) and recall (missed fraud)
- Effective for imbalanced fraud detection scenarios

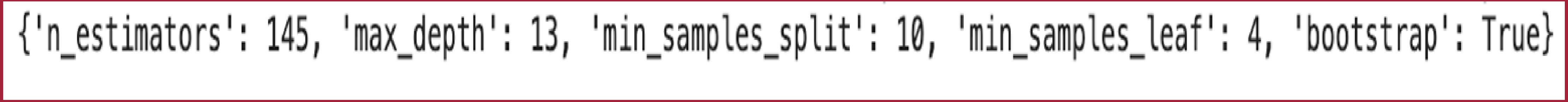
Other Metrics which didn't appealed us:

- **Accuracy**: With only ~0.1% fraud, accuracy is meaningless
- **ROC-AUC Curve**: In highly imbalanced datasets like this, ROC-AUC can be overly optimistic since it includes the true negative rate, which dominates due to the large number of non-fraud cases.

Hyperparameter Tuning!

- We cannot afford to miss a fraud case. So, we tried another weapon.

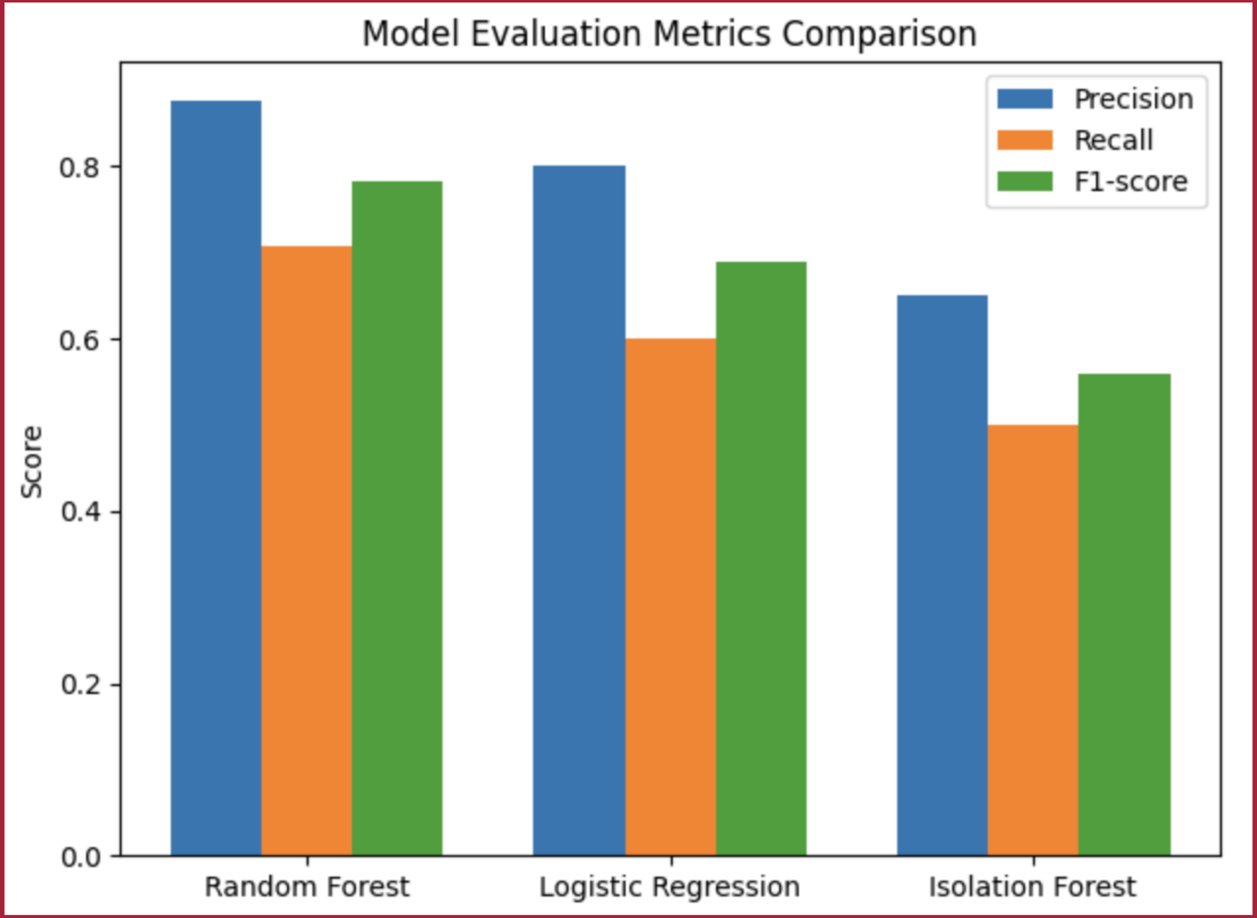
- Optuna Hyperparameter Tuning Library suggested us go green (i.e. increase the number of trees) and allowed a greater tree depth.

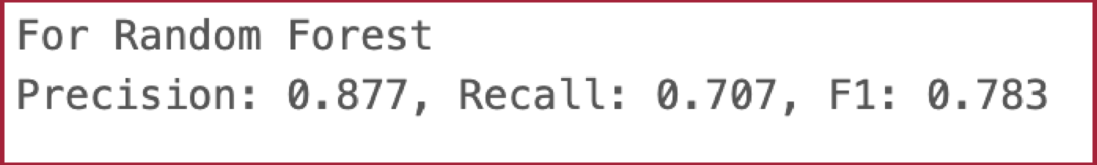


```
{'n_estimators': 145, 'max_depth': 13, 'min_samples_split': 10, 'min_samples_leaf': 4, 'bootstrap': True}
```

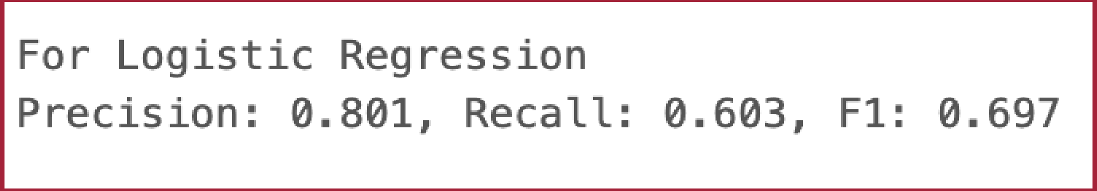
- This increased generalization and recall.
  Thus, let me say Woohoo, as we go towards a GOOD model.

Classification Performance

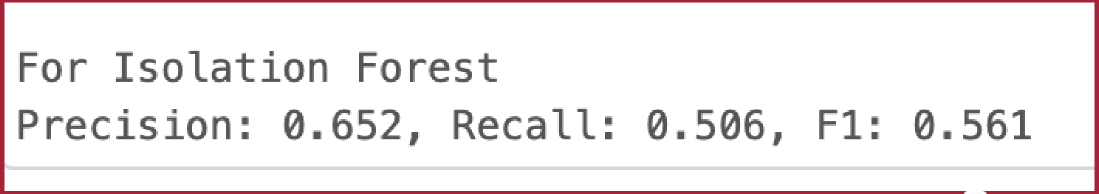




```
For Random Forest
Precision: 0.877, Recall: 0.707, F1: 0.783
```



```
For Logistic Regression
Precision: 0.801, Recall: 0.603, F1: 0.697
```



```
For Isolation Forest
Precision: 0.652, Recall: 0.506, F1: 0.561
```

Let me justify!

- Random Forest’s ensemble of decision trees excels at capturing complex fraud patterns involving non-linear feature interactions. This explains its high precision (0.877) and strong F1-score (0.783), showing that it's highly reliable at identifying fraud with fewer false positives by aggregating over many diverse trees.

- Logistic Regression, being a fast and interpretable linear model, it struggles with the intricate patterns present in fraud datasets. This leads to **underfitting**, especially in cases where fraud requires non-linear detection. As a result, causing it to miss more frauds. (low recall)

- Isolation Forest (unsupervised) isolates rare observations by randomly selecting features and split values. While it performs reasonably well in identifying extreme cases, it’s not tailored for supervised classification. This affects both precision and recall, resulting in the lowest F1-score (0.56).

Now, let us take a feel of BIG Data!

Scaling Strategies

Data Scaling (Scale Up):

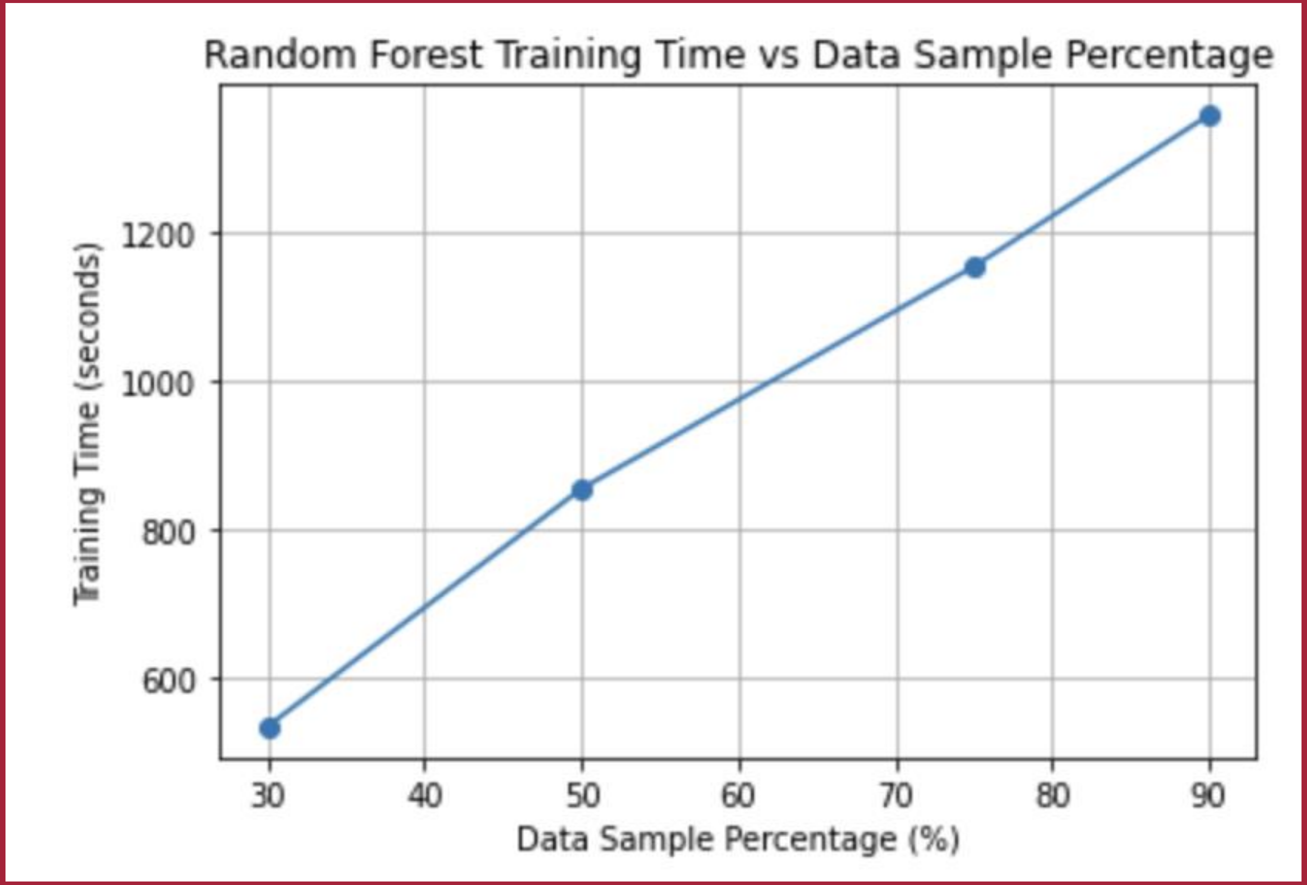
- Initial experiments with smaller subsets (30%, 50%, 75%, 90%) before processing full dataset
- Measured execution time and scalability trends

Cluster Scaling (Scale Out):

- Initially tested on smaller clusters (single-node for Dev - DataBricks)
- Progressively scaled out to multi-node clusters (1, 2, 3, 4 nodes) anticipating a sharp decrease in execution time.

Scale Up (Random Forest)

Training time for the Random Forest algorithm increases notably (**~136%**) as the data sample size scales from **30% to 90%**.

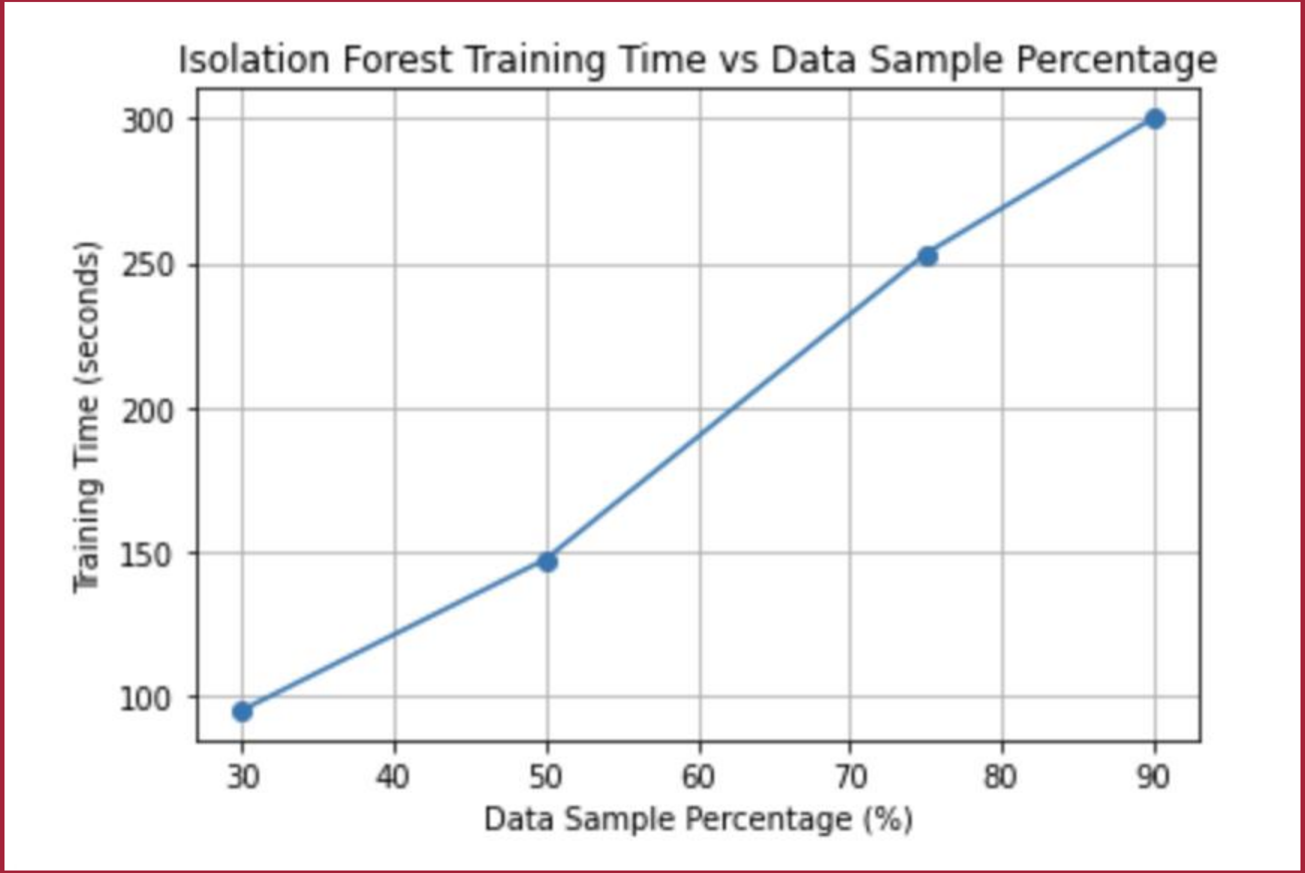


Incremental Analysis:

- A significant 45% increase from 30% to 50% data sample.
- A 25% increase from 50% to 70% data sample, indicating moderate growth.
- Another substantial increase (30%) from 70% to 90% data sample, indicating sustained growth as the dataset size increases.

Scale Up (Isolation Forest)

The Isolation Forest training time increases significantly (**~200%**) as the data sample size scales from **30% to 90%**.

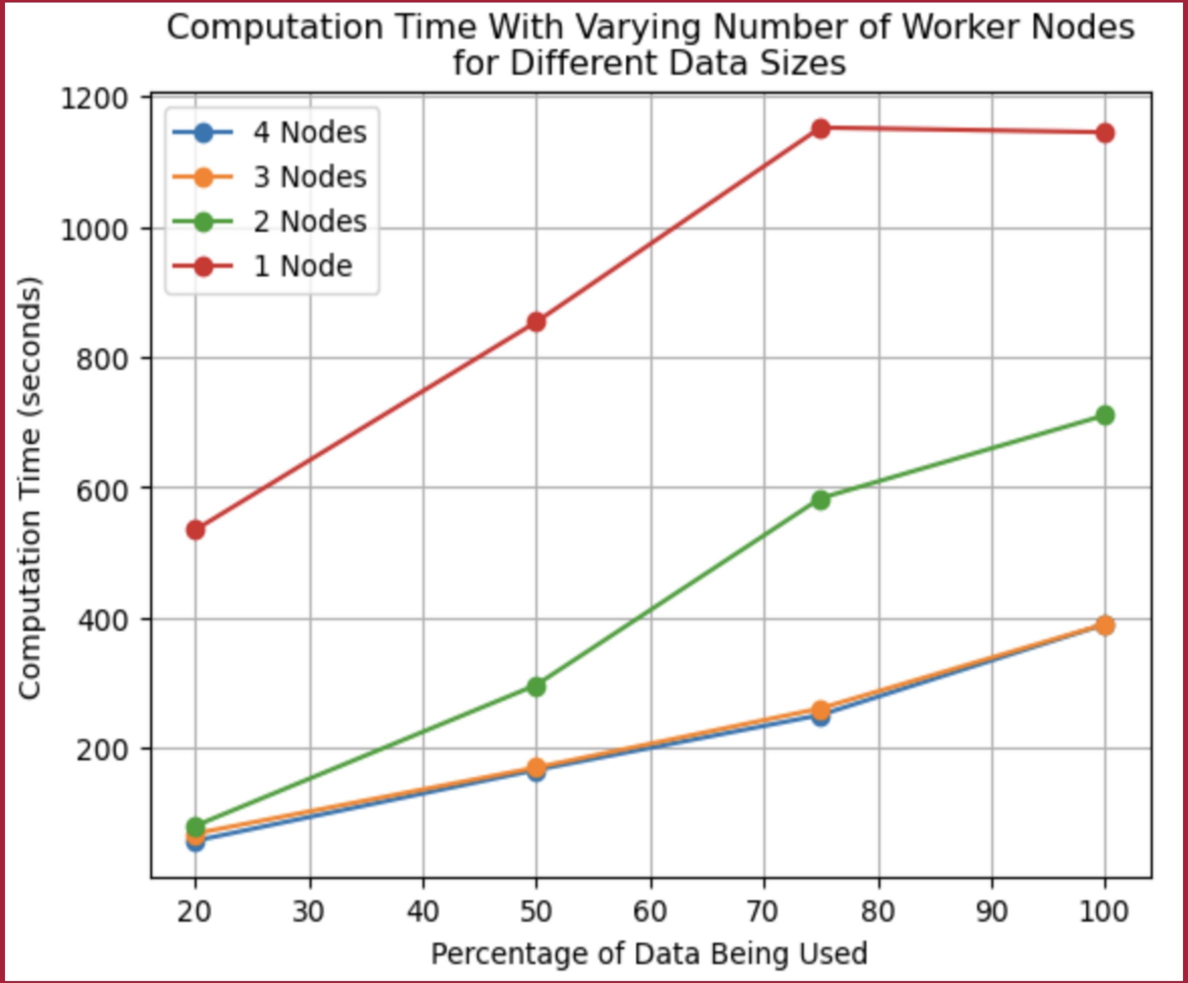


Incremental Analysis:

- A 50% increase in training time when the data sample increases from 30% to 50%.
- A 53% increase from 50% to 70% data sample size.
- A 30% increase from 70% to 90% data sample size, indicating diminishing efficiency as sample size grows larger.

Scale Out (Cluster Size)

Computation time increases significantly (**~41%**) when scaling the data usage from **20%** to 75**%** for a single worker node.



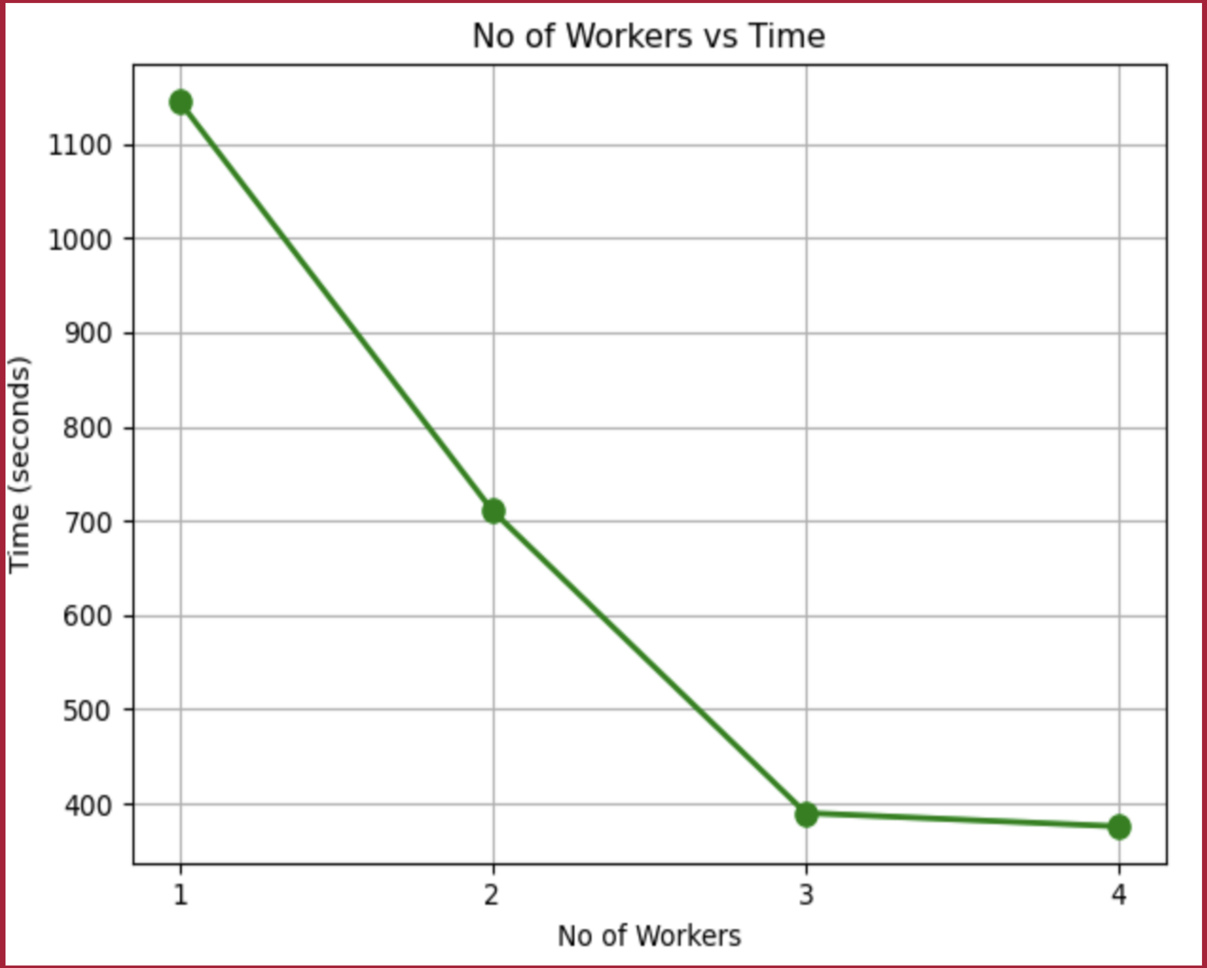
- Increasing worker nodes greatly reduces computation time
- At 100% data usage, there is approximately a 67% decrease in computation time when using 4 worker nodes (i.e. 8 cores) compared to just 1 worker node (2 cores).
- The graph illustrates the importance of parallel processing, as multiple nodes dramatically improve efficiency, especially at larger dataset sizes.

Cluster Configurations

Platform	Nodes	RAM/Node	Cores/Node
Google Cloud Platform	1 Master, 1 Worker Nodes	8 GB	2
	1 Master, 2 Worker Nodes	8 GB	2
	1 Master, 3 Worker Nodes	8 GB	2
	1 Master, 4 Worker Node	8 GB	2

1870

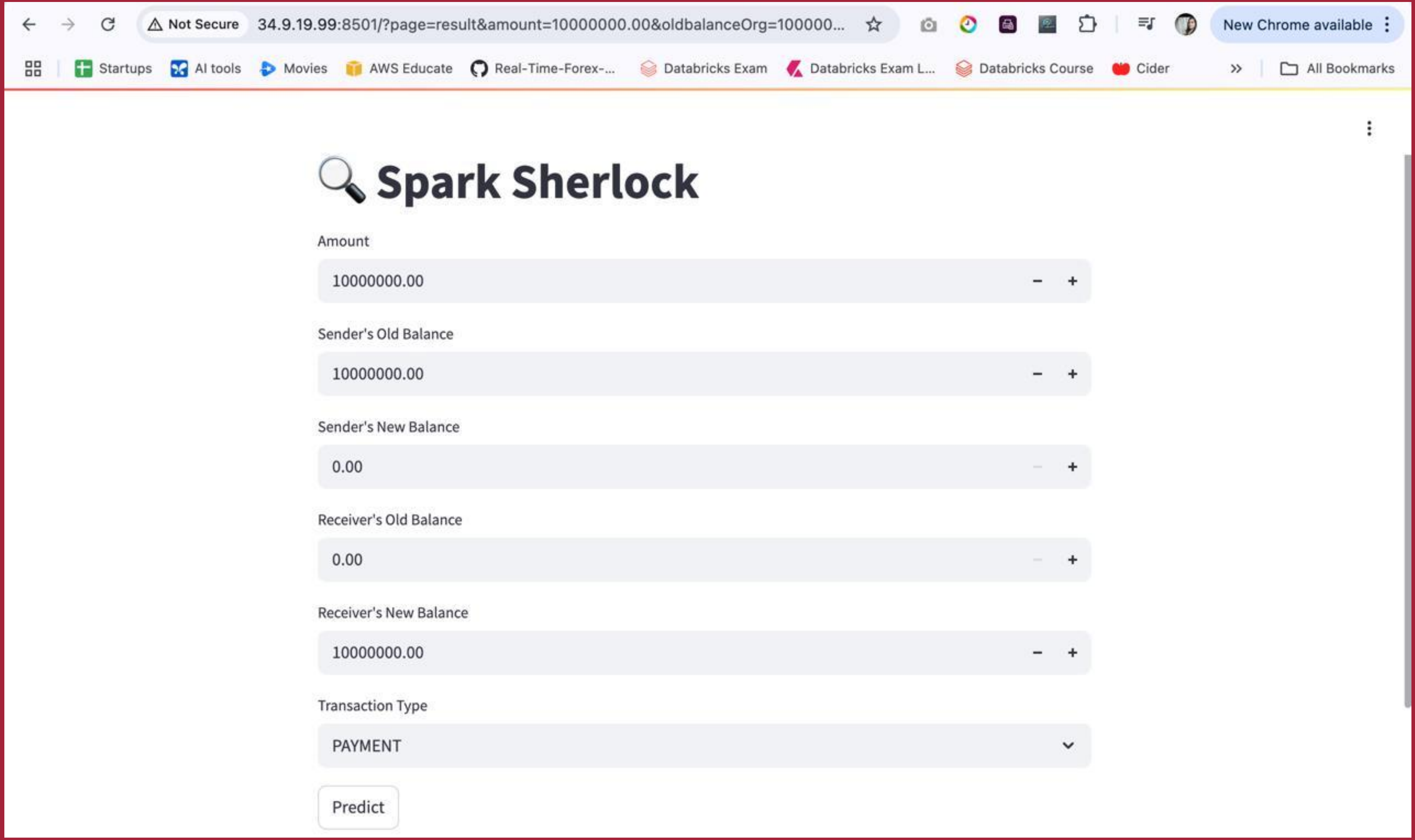
Look at the time we saved!



- Computation time decreases significantly as the number of worker nodes increases.
- There is approximately a **67% decrease** in computation time when scaling from **1 worker node** to **4 worker nodes**.
- **Major reductions observed:**
  - **39% decrease** from **1 to 2 workers**.
  - **43% decrease** from **2 to 3 workers**.
  - Minimal decrease (**5%**) from **3 to 4 workers**, it shows diminishing returns beyond 3 workers.

Web Application On Cloud

Sherlock has gone digital !



🔍 Spark Sherlock

Amount
10000000.00

Sender's Old Balance
10000000.00

Sender's New Balance
0.00

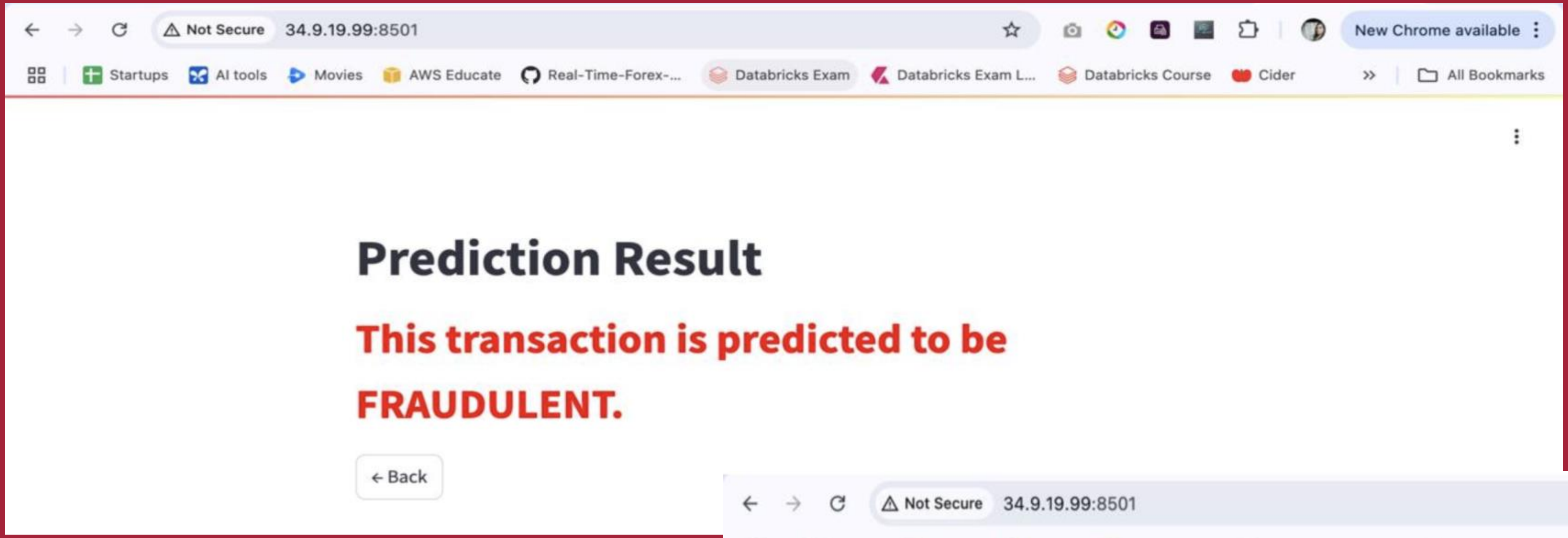
Receiver's Old Balance
0.00

Receiver's New Balance
10000000.00

Transaction Type
PAYMENT

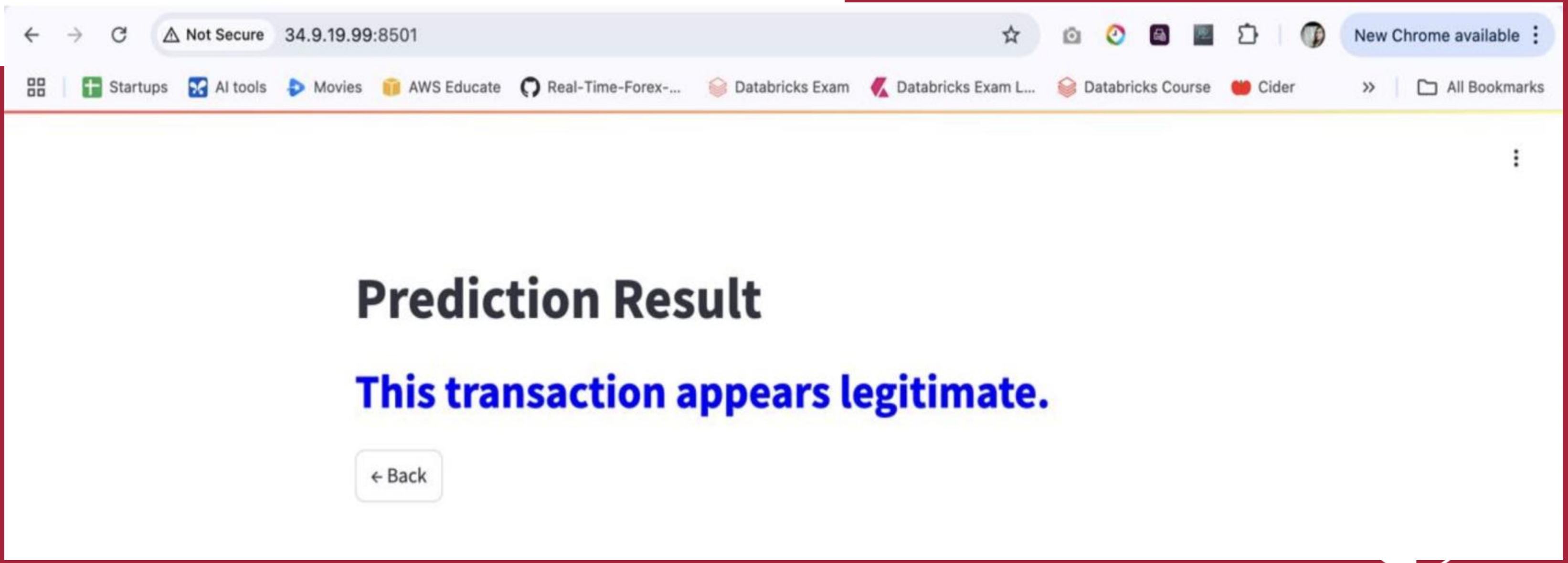
Predict

Deployed Web Application On Cloud



Prediction Result

This transaction is predicted to be FRAUDULENT.



Prediction Result

This transaction appears legitimate.

We used...

APACHE Spark™

Streamlit

GCP

databricks

1870

Way to go..

- We have the scalable fraud detection pipeline capable of rightly predicting the frauds.
- Future scope will include the streaming-based incremental training.
- Use of GPU-accelerated tree implementations.
- Experimenting with Ensemble stacking or deep learning to make a robust model.

References

- https://builtin.com/data-science/random-forest-algorithm
- https://www.analyticsvidhya.com/blog/2021/07/anomaly-detection-using-isolation-forest-a-complete-guide/
- https://gatk.broadinstitute.org/hc/en-us/articles/360035889951--How-to-Create-a-Spark-cluster-on-Google-Dataproc
- https://medium.com/google-cloud/how-to-deploy-your-streamlit-web-app-to-google-cloud-run-with-ease-c9f044aabc12

Thank You!