ASSIGNMENT-1

Credit Card Fraud Detection Report

Introduction: In the complex world of credit card fraud, offenders use a variety of advanced strategies to take advantage of weaknesses in the financial system. Every stage of the fraud process, from gathering cardholder data to carrying out fraudulent transactions, is carefully planned to minimize detection and maximize illegal profits. The goal of this project is to identify and predict fraudulent transactions, helping financial institutions minimize potential losses and protect customers.

1. Brief Description of the Dataset

The dataset used in this assignment focuses on detecting fraudulent transactions in credit card usage. Each record in the dataset represents a credit card transaction, along with various attributes that can be used to analyze and predict the likelihood of fraud.

ABOUT THE DATASET

The dataset contains transactions made by credit cards by European cardholders. This dataset presents transactions that have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. The data has been anonymized to protect the cardholders' identities. The primary objective of this dataset is to facilitate the development of fraud detection algorithms and models to identify potentially fraudulent transactions.

- **Dataset Highlights:**
- **Source: Kaggle - Credit Card Fraud Detection**
- **Number of Fraudulent Transactions:** 492 (0.172% of total transactions).
- **Class Distribution:** Highly imbalanced, with a significant majority of non-fraudulent (Class 0) transactions.

Features:

- **Time:** Number of seconds elapsed between this transaction and the first transaction in the dataset
- **id:** Unique identifier for each transaction
- **V1-V28:** Anonymized features representing various transaction attributes (e.g., time, location, etc.)
- **Amount:** The transaction amount
- **Class**: Binary label indicating whether the transaction is fraudulent (1) or not (e one.

2. Data Cleaning and Preprocessing Steps

Several data cleaning and transformation steps were applied to prepare the dataset for analysis:

Handling Missing Values: Checked for any missing values in each feature and it was found that there are no null values in any column of the dataset.

Data Type Conversion: All the datatypes of features were perfectly fine. Only the "Time" feature was later converted in the Data Analysis using SparkSQL part for Time based Analysis of Data.

Normalization/Standardization: To ensure efficient model training, features were scaled or normalized. The amount feature was normalized as it had different scale from rest of the data. Thus, it was converted to "ScaledAmount"

```
[33]: data.select( "V1", "V2", "V3", "Amount","ScaledAmount", "Class").show(10, truncate=False)
```

```
+------------------+------------------+------------------+------+----------------------+-----+
|V1                |V2                |V3                |Amount|ScaledAmount          |Class|
+------------------+------------------+------------------+------+----------------------+-----+
|-0.529912284186556|0.873891581460326 |1.34724732930113  |6.14  |[0.02452082189450426] |0    |
|-0.600816388115364|0.922454525911535 |-0.135951820418704|1.79  |[0.007148578369896194]|0    |
|-1.5057791635308  |-0.215325117259078|1.99129427646285  |82.29 |[0.3286349240551719]  |0    |
|-0.491003017302294|0.906952627077483 |1.64542281975857  |9.03  |[0.03606238138556571] |0    |
|-0.528217504778877|0.981231846386665 |1.6529880537617   |6.99  |[0.027915398215404688]|0    |
|-3.90081029867848 |-3.09837020429443 |-0.21342836428919 |1528.9|[6.105844396499603]   |0    |
|-1.53695801321944 |-0.0414224787415122|0.62984626092522  |1.0   |[0.003993619201059326]|0    |
|-1.21250203984735 |-0.058850913337692|1.82343455295237  |48.94 |[0.1954477236998434]  |0    |
|-0.88352858430582 |1.1739736253672   |0.967297814724589 |34.15 |[0.13638209571617596] |0    |
|-1.4343476082208  |0.0124699766273357|3.32738623176906  |8.0   |[0.031948953608474606]|0    |
+------------------+------------------+------------------+------+----------------------+-----+
only showing top 10 rows
```

Filtered Out Invalid Rows: Removed rows with invalid 'Amount' values (e.g., Amount < 0)

Counted the no. of rows: 284807

Counting the total number of Fraud and non-Fraud Transactions in dataset:

```
]: # Counting the total number of Fraud and non-Fraud Transactions in dataset
   data.groupBy("Class").count().show()
```

```
+-----+------+
|Class| count|
+-----+------+
|    1|   492|
|    0|284315|
+-----+------+
```

Since the dataset was imbalanced, it was balanced by undersampling the majority class i.e. Non-fraud.

3. Key insights from the data analysis

Data Analysis was done using SparkSQL in which the following operations were performed:

I. **Aggregation:** Calculating summary statistics (e.g., mean, median, standard deviation) of the numerical columns**.**

- **Calculate the mean, median (approximate), and standard deviation for all numerical columns.)**

```
# Aggregation: Summary statistics for numerical columns
aggregation_query = """
SELECT
    AVG(V1) AS mean_V1, STDDEV(V1) AS stddev_V1,
    AVG(V2) AS mean_V2, STDDEV(V2) AS stddev_V2,
    AVG(V3) AS mean_V3, STDDEV(V3) AS stddev_V3,
    AVG(Amount) AS mean_Amount, STDDEV(Amount) AS stddev_Amount,
    APPROX_PERCENTILE(Amount, 0.5) AS median_Amount
FROM creditcard_data
"""

summary_stats = spark.sql(aggregation_query)
summary_stats.show(truncate=10)
```

```
+----------+----------+----------+----------+----------+----------+-----------+-------------+-------------+
|   mean_V1| stddev_V1|   mean_V2| stddev_V2|   mean_V3| stddev_V3|mean_Amount|stddev_Amount|median_Amount|
+----------+----------+----------+----------+----------+----------+-----------+-------------+-------------+
|-2.2122...|5.33985...|1.71469...|3.53835...|-3.3701...|6.03878...| 103.253...|   270.510...|         17.0|
+----------+----------+----------+----------+----------+----------+-----------+-------------+-------------+
```

- **Mean Values:**

V1 and V3 have negative mean values, indicating a skewed distribution, while V2 is positive.

The mean transaction amount is approximately **$103.25,** serving as a baseline for comparison**.**

- **Standard Deviations:**

High standard deviations for **V1 (5.34) and V3 (6.04)** show significant variability, suggesting these features are crucial for differentiating transactions.

The standard deviation of the amount **($270.51**) indicates wide variation in transaction sizes.

- **Median Amount:**

The median amount of **$17.00,** significantly lower than the mean, suggests that most transactions are small, with a few high-value transactions inflating the mean.

INSIGHTS:

1. Fraud Detection Potential:

The differences between mean and median suggest that for fraud detection, we should focus on outliers.

- Features with high variability could be important predictors in fraud detection models.

2. Anomaly Detection:

Statistical properties indicate the potential for anomaly detection techniques to flag transactions that deviate significantly from typical behavior.

II. **Grouping and Filtering:** Group data by the Class column (0 for non-fraudulent, 1 for fraudulent) and calculate the average transaction amount and total count of transactions for each class

```
: # Grouping and Filtering: Average Amount per Class
grouping_query = """
SELECT
    Class,
    COUNT(*) AS transaction_count,
    AVG(Amount) AS avg_amount
FROM creditcard_data
GROUP BY Class
"""

grouping_stats = spark.sql(grouping_query)
grouping_stats.show()
```

```
+-----+-----------------+------------------+
|Class|transaction_count|        avg_amount|
+-----+-----------------+------------------+
|    1|              473|123.87186046511628|
|    0|              467| 82.36929336188435|
+-----+-----------------+------------------+
```

In this query, we are trying to analyze the credit card transaction data by calculating the following for each class of transactions (fraudulent and legitimate):

Count of Transactions: We determine how many transactions belong to each class. This helps us understand the volume of transactions that are fraudulent versus those that are legitimate.

Average Transaction Amount: We compute the average amount of transactions for each class. This provides insights into spending behavior, allowing us to see if fraudulent transactions tend to have a higher or lower average amount compared to legitimate transactions.

INSIGHTS:

- **Prevalence of Fraud:** The dataset shows more fraudulent transactions (473) compared to legitimate ones (467), highlighting a significant fraud concern.
- **Larger Stakes:** Average fraudulent transaction amount 123.87 dollars is greater than non-fraud transactions 82.37, indicating fraudsters may target larger amounts.

III. Join: In the join below, creditcard_data is being joined with itself to find pairs of transactions that occurred within a very short time frame (e.g., within one minute) and had the same amount.

```
# SQL query for self join to find similar transactions within a minute
self_join_query = """
SELECT
    a.Time AS transaction_time_a,
    b.Time AS transaction_time_b,
    a.Amount AS amount_a,
    b.Amount AS amount_b,
    a.Class AS class_a,
    b.Class AS class_b
FROM
    creditcard_data a
JOIN
    creditcard_data b
ON
    a.Time <> b.Time  -- Ensure we don't join the same row
    AND ABS(a.Time - b.Time) <= 60  -- Transactions occurring within 60 seconds
    AND a.Amount = b.Amount  -- Same transaction amount
"""

# Execute the self join query
self_join_results = spark.sql(self_join_query)

# Show the results of the self join
self_join_results.show()
```

```
+------------------+------------------+--------+--------+-------+-------+
|transaction_time_a|transaction_time_b|amount_a|amount_b|class_a|class_b|
+------------------+------------------+--------+--------+-------+-------+
|           28692.0|           28726.0|   99.99|   99.99|      1|      1|
|           28692.0|           28658.0|   99.99|   99.99|      1|      1|
|            8415.0|            8451.0|     1.0|     1.0|      1|      1|
|            8415.0|            8408.0|     1.0|     1.0|      1|      1|
|           11080.0|           11092.0|     1.0|     1.0|      1|      1|
|           11080.0|           11131.0|     1.0|     1.0|      1|      1|
|            7519.0|            7535.0|     1.0|     1.0|      1|      1|
|            7519.0|            7526.0|     1.0|     1.0|      1|      1|
|            7519.0|            7543.0|     1.0|     1.0|      1|      1|
|            7519.0|            7551.0|     1.0|     1.0|      1|      1|
|           11635.0|           11629.0|     1.0|     1.0|      1|      1|
|           26863.0|           26899.0|   99.99|   99.99|      1|      1|
|           26863.0|           26833.0|   99.99|   99.99|      1|      1|
|           27187.0|           27163.0|   99.99|   99.99|      1|      1|
|           27187.0|           27219.0|   99.99|   99.99|      1|      1|
|           26556.0|           26585.0|   99.99|   99.99|      1|      1|
|           26556.0|           26523.0|   99.99|   99.99|      1|      1|
|           17220.0|           17187.0|    3.79|    3.79|      1|      1|
|            7551.0|            7535.0|     1.0|     1.0|      1|      1|
|            7551.0|            7610.0|     1.0|     1.0|      1|      1|
+------------------+------------------+--------+--------+-------+-------+
only showing top 20 rows
```

Conditions for Joining:

- **Different Rows:** The condition a.Time <> b.Time ensures that the same transaction is not joined with itself.
- **Time Difference:** The condition ABS(a.Time - b.Time) <= 60 filters transactions to include only those that occurred within 60 seconds of each other. This helps identify closely-timed transactions that may indicate suspicious activity.
- **Same Amount:** The condition a.Amount = b.Amount ensures that only transactions with the same amount are considered. This is significant because fraudulent transactions often mimic legitimate ones.

INSIGHTS:

1. Frequent Fraud Patterns:

The dataset reveals multiple instances of fraudulent transactions occurring at nearly the same time with identical amounts. For example, several transactions for 1.00 and 99.99 were made within the same minute. This suggests that fraudsters may be attempting to conduct multiple similar transactions in quick succession.

2. Potential Fraud Attempts:

The presence of multiple identical amounts (i.e. 1.00 and $99.99) in close temporal proximity indicates a potential strategy used by fraudsters to test stolen card details. They may start with small amounts to avoid detection before proceeding to higher-value transactions.

3. Transaction Clustering:

The clustering of fraudulent transactions at specific times could indicate a coordinated effort, perhaps using automated scripts or bots to execute transactions rapidly. This behavior can help identify patterns in fraudulent activity that can be monitored in real-time by fraud detection systems.

4. Transaction Amounts:

The repetition of specific amounts, such as $99.99, can point to a pattern that may be of interest to investigators. It suggests that fraudsters are not only mimicking legitimate transactions but also favoring certain amounts that may help evade detection thresholds.

IV. Time Based Analysis:

```
# SQL query for time-based analysis
time_analysis_query = """
SELECT
    DATE(timestamp_column) AS transaction_date,
    COUNT(*) AS total_transactions,
    SUM(CASE WHEN Class = 1 THEN 1 ELSE 0 END) AS total_fraud,
    SUM(CASE WHEN Class = 0 THEN 1 ELSE 0 END) AS total_non_fraud
FROM
    creditcard_data
GROUP BY
    transaction_date
ORDER BY
    transaction_date
"""
# Execute the time-based analysis query
time_analysis_results = spark.sql(time_analysis_query)

# Show the results of the time-based analysis
time_analysis_results.show()
```

```
+----------------+------------------+-----------+---------------+
|transaction_date|total_transactions|total_fraud|total_non_fraud|
+----------------+------------------+-----------+---------------+
|      1969-12-31|             10617|         44|          10573|
|      1970-01-01|            143856|        288|         143568|
|      1970-01-02|            129253|        141|         129112|
+----------------+------------------+-----------+---------------+
```

INSIGHT:

It can be seen that for

- December 31, 1969,the fraud rate is ~ 0.41%
- January 1, 1970, the fraud rate is ~0.20%
- January 2, 1970, the fraud rate is ~ 0.11%

This shows that the fraud rate is very low across all days, which indicates that the majority of transactions are legitimate.

4. Description of the machine learning model:

To predict fraudulent transactions, Logistic Regression was chosen as the primary machine learning model due to its efficiency in binary classification and interpretability, making it valuable for understanding factors contributing to fraud.

- **Logistic Regression in Fraud Detection:** Logistic Regression models the probability of a transaction being fraudulent by relating the target variable (fraud or not) with transaction features. This model produces probabilities that can be converted into binary labels (fraud or non-fraud) and is especially effective with appropriate regularization and threshold adjustments to handle imbalanced datasets like credit card fraud.
- **Hyperparameter Tuning with Random Search CV:** To optimize performance, Random Search Cross-Validation was applied to tune hyperparameters (such as regularization) over a subset of data. This approach balances computational efficiency with model accuracy, as it identifies optimal configurations while significantly reducing processing time.
- **Subset Sampling for Efficiency:** Given the dataset's size, a subset was used during tuning. This ensured computational efficiency while retaining the class distribution and patterns of fraudulent transactions, allowing the model to generalize well to the entire dataset.
- **Pipeline for Consistency:** A pipeline was used to seamlessly integrate data preprocessing and model training steps, ensuring consistency and preventing data leakage, while making the workflow scalable and easy to manage.

This combination of Logistic Regression, hyperparameter tuning, and pipeline integration creates an efficient, scalable solution for fraud detection.

➢ Best Hyperparameters found by Random Search CV:

- Regularization Parameter (regParam): 0.1

- Maximum Iterations (maxIter): 10

Insights based on best parameters:

Regularization Parameter (regParam = 0.1):

A moderate regularization value of 0.1 indicates the model is effectively balancing complexity and generalization, crucial for fraud detection. It helps prevent overfitting, ensuring the model captures significant patterns in fraudulent transactions while minimizing false positives in legitimate (non-fraud) transactions**.**

Maximum Iterations (maxIter = 10):

The model's convergence in just 10 iterations suggests that the data is well-structured and the features are informative, allowing the model to quickly learn to distinguish between fraud and non-fraud cases. However, this low iteration count may indicate potential limitations in fully exploring complex patterns, which could be important for improving detection rates.

Overall Implications:

The selected hyperparameters reflect a model capable of identifying fraudulent transactions while maintaining a low false positive rate for non-fraud transactions. However, further tuning or adjustments may enhance performance, especially in ensuring robust detection amidst the challenges of imbalanced classes inherent in fraud detection**.**

5. Evaluation metrics and model performance:

Model was evaluated by using simple Logistic Regression and then after hyperparameter tuning with random search Cross Validation, metrics like accuracy, precision, recall, r2 score and AUC were evaluated again to see the effectiveness of tuning.

Following were the Observations:

	Accuracy	Precision	Recall	AUC	R^2 Score
Logistic Regression	**0.95**	**0.95**	**0.95**	**0.95**	**0.80**
After Hyperparameter Tuning	**0.91**	**0.92**	**0.91**	**0.91**	**0.63**

Insights from Metrics of Simple Logistic Regression

1. Accuracy (0.95):

An accuracy of 95% indicates that the model correctly identifies the majority of transactions, both fraudulent and non-fraudulent. This high level suggests the model is generally effective in distinguishing between the two classes.

2. Precision (0.95): Precision measures the proportion of true positive predictions (fraudulent transactions) among all predicted positives. A precision of 95% implies that when the model predicts a transaction as fraudulent, it is highly likely to be correct. This is crucial in fraud detection as it minimizes the risk of falsely accusing legitimate transactions.

3. Recall (0.95):

Recall reflects the model's ability to identify all actual fraudulent transactions. A recall of 95% means the model successfully detects 95% of actual fraud cases. This is vital for fraud detection, as missing a fraudulent transaction can lead to significant financial losses**.**

4. AUC (0.95):

The AUC (Area Under the ROC Curve) value of 0.95 indicates excellent discriminatory power. It shows that the model is highly effective in distinguishing between fraudulent and non-fraudulent transactions across different thresholds, providing confidence in its predictions.

5. R^2 Score (0.80):

The R^2 score of 0.80 suggests that the model explains that 80% of the variability in the target variable(fraud or non-fraud) is explained by the independent features. This indicates a strong relationship between the features used and the likelihood of fraud, contributing to the model's overall reliability.

INSIGHTS & PERFORMANCE COMPARISON AFTER TUNING:

1. Performance Drop After Tuning:

The model's performance metrics dropped after hyperparameter tuning, indicating that the initial high performance may have resulted from overfitting to the training data. This overfitting could lead to poorer identification of fraudulent transactions when encountering new, unseen data.

2. Accuracy Analysis:

Accuracy decreased from 95% to 91%, reflecting a more realistic assessment of the model's ability to generalize. While still strong, this reduction highlights the model's improved robustness in distinguishing between fraudulent and non-fraudulent transactions.

3. Precision and Recall:

Precision fell from 95% to 92%, and recall dropped from 95% to 91%. This indicates a slight increase in the model's errors regarding correctly identifying fraudulent transactions (precision) and capturing all actual fraud cases (recall). Despite this decrease, the model remains effective at identifying fraud relative to legitimate transactions.

4. AUC Interpretation:

The AUC decreased from 0.95 to 0.91, suggesting a slight reduction in the model's capability to distinguish between fraudulent and non-fraudulent transactions. However, the model still demonstrates good performance overall.

5. R^2 Score:

The R^2 score dropped from 0.80 to 0.63, indicating that the model accounts for less variability in predicting fraudulent transactions compared to non-fraudulent ones. This could suggest potential issues in capturing the underlying patterns associated with fraud after tuning.

CONCLUSION:

- **Model Generalization:** The drop in performance metrics after hyperparameter tuning indicates that the tuned model is more generalized, reducing the risk of overfitting. This is crucial for fraud detection, as a well-generalized model can more effectively identify fraudulent transactions in diverse, unseen datasets.
- **Model Evaluation:** Although the tuned model shows slightly lower performance metrics, it is better equipped to accurately classify fraud in real-world scenarios. This trade-off enhances the model's reliability, prioritizing generalization to reduce false negatives and improve overall fraud detection effectiveness.

This credit card fraud detection assignment developed an efficient approach to identify fraudulent transactions using Logistic Regression, integrated with a structured pipeline for seamless data processing and model tuning. The model's interpretability and optimized performance through Random Search CV helped accurately detect fraud patterns, minimizing false positives and negatives. This solution effectively demonstrates how machine learning can enhance fraud detection, supporting financial institutions in safeguarding customers and reducing risks