M.Kumarasamy
College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

A Project Report

on

AD CLICK FRAUD DETECTION

Submitted in partial fulfillment of requirements for the award of the

Degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Under the guidance of

Mrs. SUBHA SRI S

IBM CORPORATE TRAINER

Submitted by

SUBBURAMAN V [REG NO: 927623BAD110]

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

M KUMARASAMY COLLEGE OF ENGINEERING, KARUR

(Autonomous)

Karur 639-113

DECEMBER -2025

M. KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to Anna University, Chennai)

KARUR – 639 113

BONAFIDE CERTIFICATE

Certified that this project report **" AD CLICK FRAUD DETECTION"** is the Bonafide work of **"SUBBURAMAN V (927623BAD110)"** who carried out the project work during the academic year 2025- 2026 under my supervision.

SIGNATURE

Mrs.SUBHA SRI S

IBM CORPORATE TRAINER

Department of Artificial Intelligence

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur-639113.

SIGNATURE

Dr. A. Selvi, M.E., Ph.D.,

ASSOCIATE PROFESSOR,

HEAD OF THE DEPARTMENT,

Department of Artificial Intelligence,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur-639113.

This project Work (ADI1303) report has been submitted for the 5th Semester Project viva voce Examination held on ____________________

INTERNAL EXAMINER

M.KUMARASAMY COLLEGE OF ENGINEERING
DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Vision of the Department:

To excel in education, innovation, and research in Artificial Intelligence and Data Science to fulfil industrial demands and societal expectations.

Mission of the Department:

M1: To educate future engineers with solid fundamentals, continually improving teaching methods using modern tools.

M2: To collaborate with industry and offer top-notch facilities in a conducive learning environment.

M3: To foster skilled engineers and ethical innovation in AI and Data Science for global recognition and impactful research.

M4: To tackle the societal challenge of producing capable professionals by instilling employability skills and human values.

Programme Educational Objectives (PEOs):

Graduates will be able to:

PEO 1: Compete on a global scale for a professional career in Artificial Intelligence and Data Science.

PEO 2: Provide industry-specific solutions for the society with effective communication and ethics.

PEO 3: Hone their professional skills through research and lifelong learning initiatives.

Mapping of Programme Educational Objectives with Mission of the Department:

PEOs / Department Mission Statements	M1	M2	M3	M4
PEO1	3	3	2	3
PEO2	3	3	2	2
PEO3	3	2	3	2

1: Slight (Low) **2: Moderate (Medium)** **3: Substantial (High)**

Programme Outcomes (POs):

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcomes (PSOs):

PSO1: Capable of finding the important factors in large datasets, simplify the data, and improve predictive model accuracy.

PSO2: Capable of analyzing and providing a solution to a given real-world problem by designing an effective program.

Mapping of Programme Educational Objectives with Programme Outcomes and Programme Specific Outcomes:

PEOs / POs & PSOs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
PEO1	3	2	2	2	3	2	3	3	1	2	3	1	3	1
PEO2	2	2	3	2	3	3	3	2	2	3	2	3	3	2
PEO3	3	3	2	3	3	2	3	3	3	2	3	3	2	3

1: Slight (Low) **2: Moderate (Medium)** **3: Substantial (High)**

ABSTRACT

An end-to-end big data and machine learning pipeline is developed to detect fraudulent ad clicks using Apache Spark. Preprocessed clickstream data are loaded into distributed data frames, where temporal features such as click hour and day, along with frequency-based statistics for IPs and channels, are engineered and combined with encoded device, app, and OS attributes to capture user behavior patterns and traffic characteristics. The workflow initially applies baseline models to understand class imbalance and legitimate click distributions, followed by advanced ensemble classifiers such as Random Forest and Gradient Boosted Trees trained on feature vectors that integrate temporal, categorical, and aggregate behavioral signals to distinguish fraudulent clicks from genuine user interactions. Model quality is assessed using classification metrics including precision, recall, F1-score, and ROC-AUC, while feature importance analysis highlights the relative contribution of IP activity, time-of-day patterns, and channel usage to fraud detection. In addition, graph-based representations of IP–device–app relationships and integrated monitoring dashboards are incorporated, enabling interactive exploration of suspicious clusters and scalable experimentation for real-time ad fraud analytics.

Keywords: Ad click fraud detection, Apache Spark, Big data processing, Imbalanced classification, Random Forest classifier, Gradient Boosted Trees, Feature engineering, Clickstream analytics, Graph-based fraud networks, Real-time fraud monitoring.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The rapid growth of digital advertising has been accompanied by a rise in ad click fraud, where malicious actors generate fake clicks to drain advertiser budgets, distort campaign metrics, and manipulate bidding strategies. Conventional fraud detection systems running on single-node databases struggle with the scale and speed of modern clickstream data, which can contain millions of events per hour with high dimensionality and strong class imbalance. This project presents a scalable pipeline that ingests preprocessed ad click logs—containing IP address, app, device, operating system, channel, and timestamp information—into Spark DataFrames for efficient distributed processing. Temporal features such as click hour and day-of-week are engineered from timestamps, while high-cardinality identifiers (IP, app, device, OS, channel) are encoded using techniques like StringIndexer and OneHotEncoder to form rich behavioral representations. This combination of temporal and categorical features enables robust modeling of fraudulent versus legitimate click patterns at scale.

Machine learning models form the core of the fraud analytics. Baseline analysis first characterizes the extreme class imbalance between genuine and fraudulent clicks and explores simple decision rules to understand feature distributions. Advanced ensemble classifiers such as Random Forest and Gradient Boosted Trees are then trained on assembled feature vectors to distinguish normal user behavior from automated or coordinated fraudulent activity. These models are evaluated using imbalanced-learning metrics including precision, recall, F1-score, and ROC-AUC to ensure that fraudulent clicks are detected without excessively penalizing legitimate traffic. Feature importance analysis highlights the influence of factors such as IP frequency, time-of-day patterns, and suspicious channel usage, while confusion matrices and probability score distributions provide insight into the model's decision boundaries. Together, these Spark MLlib components deliver scalable, iterative model refinement on large volumes of clickstream data.

Beyond classification, the pipeline incorporates graph analytics to uncover coordinated fraud networks. Using graph processing libraries, relationships among IPs, devices, apps, and channels are modeled as a graph, where dense subgraphs and highly connected nodes may correspond to click farms or botnets. Interactive visualizations of these graphs reveal suspicious clusters and traffic anomalies, supporting deeper investigation. Spark sessions are configured for efficient resource usage and integrated with monitoring dashboards for tracking job status and model performance in near real time. This end-to-end system—from data ingestion and feature engineering to classification, graph-based analysis, visualization, and deployment—demonstrates how big data technologies can strengthen ad ecosystems by providing actionable.

CHAPTER 2

OBJECTIVES

The primary objective is to construct a scalable big data pipeline using Apache Spark for processing large-scale ad clickstream datasets, including IP addresses, app identifiers, device types, operating systems, advertising channels, and timestamped click events. This involves loading preprocessed CSV data containing millions of click records into Spark DataFrames, performing efficient distributed computations, and engineering key features such as temporal attributes (hour of day, day of week, click intervals) and categorical encodings for high-cardinality identifiers (IP, app, device, OS, channel) via StringIndexer and OneHotEncoder. By leveraging Spark's MLlib ecosystem, the pipeline aims to handle high-volume clickstream data that exceeds traditional computing limits, ensuring robust performance for real-time fraud detection and model training on distributed clusters while addressing the challenge of extreme class imbalance between legitimate and fraudulent clicks.

A core goal centers on developing and evaluating high-precision classification models to detect fraudulent ad clicks. Initial efforts focus on baseline analysis to understand the severe imbalance where fraudulent clicks represent less than 1% of total traffic, followed by implementation of class-balancing techniques such as weighted loss functions and SMOTE oversampling. This progresses to advanced ensemble methods including Random Forest classifiers and Gradient Boosted Trees trained on unified feature vectors combining temporal patterns, behavioral statistics, and categorical encodings to distinguish fraud from genuine user interactions. Models are rigorously evaluated using precision, recall, F1-score, and ROC-AUC metrics to balance fraud detection with minimizing false positives that could block legitimate users.

The pipeline extends to graph analytics and deployment infrastructure to enhance fraud investigation and operational usability. Using GraphFrames and NetworkX, IP-device-app-channel relationship networks are constructed to identify coordinated fraud operations, bot farms, and suspicious clusters through graph algorithms like connected components and degree centrality analysis. Interactive visualizations with Plotly reveal fraud network topologies for investigative teams. Additional objectives include configuring Spark sessions with custom ports, exposing the Spark Web UI via pyngrok tunneling for remote monitoring of job execution and resource utilization, and integrating real-time streaming capabilities for live click scoring. Collectively, these elements aim to deliver an end-to-end, production-ready system supporting advertisers in protecting campaign budgets through scalable fraud detection, network analysis, and actionable insights.

CHAPTER 3

EXISTING SYSTEM

In the current landscape, Apache Spark-based analytics platforms have gained traction for processing large-scale clickstream data in the digital advertising ecosystem. These systems primarily focus on ingesting historical click logs containing IP addresses, timestamps, device identifiers, and conversion events, applying basic aggregation and filtering operations to identify suspicious patterns. The collected insights are typically presented through batch reporting dashboards or simple visualization tools, enabling advertisers to review fraud metrics periodically. While such frameworks provide distributed processing capabilities for handling millions of click events, their scope remains centered on retrospective fraud analysis rather than real-time prevention. Most existing pipelines operate in isolated analytics environments without seamless integration with live ad serving platforms or bidding systems, limiting their utility in operational fraud mitigation and budget protection.

3.1. Limited Real-Time Detection Capabilities

Most Spark-based ad fraud analytics focus on batch processing of historical click data, generating daily or weekly fraud reports after fraudulent clicks have already consumed advertiser budgets. While useful for post-campaign analysis, they lack streaming capabilities for immediate fraud detection at the point of click. This reactive approach allows click farms and bot networks to operate undetected for extended periods, positioning these systems as forensic tools rather than preventive defenses that can block fraudulent traffic before attribution occurs

3.2. Inadequate Handling of Class Imbalance

Existing pipelines often train models on raw, highly imbalanced clickstream data where fraudulent clicks represent less than 0.5% of total events. Without sophisticated techniques like SMOTE oversampling, weighted loss functions, or threshold optimization, these models achieve deceptively high accuracy by simply classifying all clicks as legitimate. This results in near-zero fraud recall and missed detection of fraudulent patterns, allowing malicious actors to evade detection while genuine advertisers suffer financial losses

3.3. Limited Feature Engineering for Behavioral Patterns

Traditional fraud detection systems rely on simple features like IP blacklists or basic click frequency counts, missing critical behavioral signals that distinguish sophisticated fraud. They fail to extract temporal patterns (hour-of-day distributions, click intervals), aggregate statistics (IP-channel co-occurrence, device-app relationships), or graph-based features (connected fraud networks). This shallow feature representation limits model capacity to detect evolving fraud tactics such as distributed bot networks or click injection attacks.

3.4. Focus on Aggregated Metrices Rather than Individual Risk Scoring

Design emphasizes campaign-level fraud rates and aggregate statistics over individual click risk assessment. Systems lack granular probability scores for each click event that would enable dynamic traffic filtering or real-time bidding adjustments. This prevents advertisers from implementing targeted countermeasures like IP blocking, device fingerprinting, or channel bid modifications based on fraud likelihood.

3.5. Absence of Fraud Network Analysis

Systems process clicks independently without analyzing coordinated attack patterns across multiple dimensions. They miss graph-based fraud signals like densely connected IP-device-app clusters, synchronized click bursts from bot farms, or hierarchical fraud networks operating across advertising channels. This siloed approach fails to detect organized fraud operations that span multiple campaigns and traffic sources

CHAPTER 4

PROPOSED SYSTEM

The proposed system introduces an AI-driven Spark ML framework that detects ad click fraud and identifies malicious traffic patterns using distributed processing of high-volume clickstream data across advertising networks. This system utilizes scalable machine learning models—Random Forest Classifier and Gradient Boosted Trees—to analyze temporal click patterns, IP behavior, device characteristics, and channel performance through engineered features including click timestamps, frequency statistics, and categorical encodings. By capturing anomalous patterns and coordinated attack signatures in these signals, the framework detects fraudulent clicks and bot networks in real-time. Unlike traditional rule-based fraud filters that rely on static blacklists, this system emphasizes predictive classification and scalable analytics. It delivers click-level fraud probabilities, traffic source risk scores, and feature importance insights to advertisers and ad platforms, enabling proactive budget protection and campaign optimization.

4.1. Automated Data Acquisition and Preprocessing

The system ingests multi-dimensional clickstream data from preprocessed CSV files into Spark DataFrames. An automated pipeline handles missing attribution times, infers schemas, engineers temporal features (hour, day-of-week, click intervals), computes aggregate statistics (IP click frequency, channel conversion rates), and applies StringIndexer/OneHotEncoder for high-cardinality categorical variables (IP, app, device, OS, channel), ensuring clean, distributed-ready inputs for modeling while preserving the natural class imbalance for realistic fraud detection.

4.2. Advanced Classification with Imbalance Handling

At the core lies scalable classification modeling via Spark MLlib, employing Random Forest Classifier with class-weighted loss functions (fraud weight 500:1) to address extreme imbalance where fraudulent clicks comprise less than 0.5% of traffic. The framework applies SMOTE oversampling during training to generate synthetic fraud samples, progressing to Gradient Boosted Trees on vectorized temporal-behavioral feature sets for high-precision fraud prediction, capturing complex non-linear patterns that distinguish bot traffic from genuine user clicks across global advertising campaigns.

4.3. Intelligent Fraud Risk Scoring

The framework assigns fraud probability scores to each click event based on model confidence and behavioral anomaly metrics. Using classification outputs including precision, recall, F1-score, and ROC-AUC, it ranks clicks by fraud likelihood and flags high-risk traffic sources (IPs, channels, device types) for immediate blocking or bid reduction. Dynamic thresholding enables advertisers to balance fraud prevention against false positive rates that could reject legitimate users, supporting real-time traffic filtering with interpretable risk metrics.

4.4. Comprehensive Evaluation and Model Optimization

Reliability is ensured through BinaryClassificationEvaluator metrics (ROC-AUC, precision-recall curves), MulticlassClassificationEvaluator for balanced accuracy, and confusion matrix analysis. Feature importance ranking reveals critical fraud indicators like IP frequency patterns and suspicious time windows. Optimization involves stratified train-test splits preserving class distribution, hyperparameter tuning via Spark ML's CrossValidator for tree depth and ensemble size, and threshold calibration to minimize false positives while maximizing fraud detection rates.

4.5. Interactive Visualization and Network Analysis

Plotly graphs visualize fraud network topologies using GraphFrames to map IP-device-app-channel relationships, revealing coordinated bot clusters through connected component analysis. Matplotlib generates ROC curves, precision-recall tradeoffs, and feature importance bar charts. NetworkX renders suspicious traffic patterns with node centrality highlighting fraud hubs. Spark Web UI exposed via pyngrok tunneling provides real-time job monitoring, enabling fraud analysts to track model performance and investigate flagged traffic with interactive dashboards.

4.6. Scalable Distributed Dataset Handling

The system's strength lies in Spark's ability to process billions of click events across distributed clusters without memory constraints, using partitioned DataFrames and lazy evaluation for efficient feature engineering on high-cardinality columns (millions of unique IPs). Random splits maintain fraud class representation in train-test sets, while Spark's columnar processing handles sparse one-hot encoded features efficiently. This architecture ensures stable fraud detection across diverse traffic sources, device types, and geographic regions while supporting real-time streaming integration for live click scoring.

CHAPTER 5

TOOLS AND TECHNOLOGIES

5.1.Python

Python serves as the core programming language for the ad click fraud detection pipeline due to its simplicity, extensive big data ecosystem, and compatibility with distributed computing frameworks. Its readability and versatility make it ideal for Spark DataFrame manipulations, temporal feature engineering from click timestamps, machine learning model development for imbalanced classification, and graph analytics on fraud network topologies. Python's rich library ecosystem including scikit-learn for SMOTE oversampling and pandas for result analysis complements Spark's distributed capabilities.

5.2.Big Data and ML Frameworks

Apache Spark and PySpark form the backbone of the distributed processing architecture. Spark MLlib powers scalable Random Forest Classifier and Gradient Boosted Trees models with built-in support for class-weighted loss functions to handle extreme fraud imbalance. GraphFrames enables IP-device-app-channel fraud network analysis through distributed graph algorithms. The modular Spark ecosystem supports seamless clickstream data ingestion, high-cardinality categorical encoding, feature vectorization, and real-time model scoring across clusters, ensuring linear scalability from millions to billions of click events.

5.3.Libraries

- **Network X:** Graph construction and analysis algorithms (connected components, degree centrality, PageRank) to identify coordinated fraud clusters.
- **Plotly**: Interactive fraud network graph visualizations with zoom, hover tooltips, and cluster highlighting for investigative analysis.
- **Matplotlib**: ROC curves, precision-recall tradeoff plots, confusion matrices, and feature importance bar charts for model interpretability.

5.4.Hardware

The system runs on distributed Spark clusters with local or standalone master configuration, leveraging multi-core CPU processing for parallel feature engineering on large-scale clickstream datasets containing millions of click events. Google Colab environments with Spark 3.5+ installation provide accessible development platforms with adequate memory for training ensemble models on imbalanced data. For production deployment, cloud-based Spark clusters (AWS EMR, Databricks, Google Dataproc) with auto-scaling capabilities handle billions of daily clicks across global advertising networks. Spark History Server and Web UI (ports 4040/4050) enable remote job monitoring, execution plan visualization, and resource utilization tracking via pyngrok tunneling for real-time fraud detection pipeline observability. Distributed storage systems (HDFS, S3, Azure Blob) support efficient clickstream data ingestion and model checkpoint persistence.

CHAPTER 6

METHODOLOGY

6.1. Data Collection

The project begins with loading multi-dimensional ad clickstream data from preprocessed CSV files containing IP addresses, app identifiers, device types, operating systems, advertising channels, click timestamps, attribution timestamps, and fraud labels (is_attributed). Spark DataFrames ingest the time-series dataset representing click events across diverse traffic sources, apps, and user devices. The dataset contains over 4 million click records per file, with fraudulent clicks representing approximately 0.2% of total events, mirroring real-world advertising scenarios. This extreme class imbalance captures both legitimate user engagement patterns and sophisticated fraud operations including bot networks and click farms, ensuring realistic model training conditions.

6.2. Data Preprocessing

Extensive preprocessing ensures distributed data quality using Spark SQL operations. This includes dropping rows with null click_time values, inferring schemas automatically with appropriate data types for numeric IDs and timestamp columns, and converting click_time strings to timestamp objects for temporal feature extraction. Engineering temporal features involves extracting hour-of-day, day-of-week, and calculating click intervals between consecutive events using Spark window functions. High-cardinality categorical columns (IP, app, device, OS, channel) undergo StringIndexer encoding followed by OneHotEncoder transformation to create sparse vector representations. The attributed_time column, which contains 99.8% null values for non-converted clicks, is handled appropriately. The dataset undergoes stratified randomSplit(0.8, 0.2) with fixed seed to ensure fraud class representation in both training and testing partitions, validating model generalization across diverse traffic patterns.

6.3. Feature Extraction

Key features include temporal attributes (hour, day-of-week, click_interval), behavioral statistics (IP click frequency, channel conversion rate, app-device co-occurrence), one-hot encoded vectors for categorical identifiers, and the binary target variable (is_attributed indicating fraud). VectorAssembler combines temporal, aggregate, and categorical features into unified ML-ready feature vectors capturing user behavior patterns, traffic anomalies, and device-app relationships indicative of fraudulent activity. Spark SQL analytics identify strongest fraud predictors through correlation analysis and chi-square tests between features and the fraud label. Feature engineering also includes IP reputation scores based on historical click-to-conversion ratios and time-window aggregations (clicks per IP in last hour) to detect burst traffic patterns characteristic of automated bots.

6.4. Model Development

Spark MLlib classification models process imbalanced clickstream data with specialized techniques: Baseline analysis establishes fraud prevalence and simple decision rule performance. RandomForestClassifier (numTrees=100, maxDepth=10) with class-weighted loss functions (classWeight={0: 1.0, 1: 500.0}) addresses extreme imbalance by penalizing fraud misclassification 500 times more than legitimate clicks. GBTClassifier (maxIter=50) captures sequential patterns through gradient boosting on residual errors. SMOTE oversampling generates synthetic fraud samples during training to increase minority class representation to 5-10% without duplicating real fraud events. Models train on distributed train_data using Spark's parallelized tree construction, ensuring scalability across cluster nodes while maintaining reproducibility through fixed random seeds.

6.5. Fraud Classification

Trained models output fraud probability scores for each click event: RandomForest provides class probabilities through ensemble voting, while GBT generates confidence scores from final prediction margins. BinaryClassificationEvaluator computes ROC-AUC (area under receiver operating characteristic curve) and area under precision-recall curve to assess discrimination capability independent of classification threshold. MulticlassClassificationEvaluator calculates precision (fraud detection accuracy among flagged clicks), recall (percentage of actual fraud detected), and F1-score (harmonic mean balancing both metrics). High fraud probability scores (>0.7) trigger immediate traffic blocking, while moderate scores (0.3-0.7) flag clicks for manual review. Feature importance rankings identify top fraud signals—IP frequency dominance, suspicious hour patterns, low-engagement channels—enabling targeted countermeasures and rule refinement.

6.6. Visualization and Interpretation

Plotly renders interactive fraud network graphs constructed via GraphFrames, linking IPs to devices, apps, and channels with edge weights representing click volume. NetworkX algorithms detect densely connected components indicating coordinated bot operations, visualized with spring layouts highlighting fraud clusters. Matplotlib displays ROC curves comparing model performance, precision-recall tradeoff curves for threshold optimization, confusion matrices showing true/false positive/negative distributions, and feature importance bar charts ranking fraud predictors. Probability score histograms reveal decision boundary separation between fraud and legitimate traffic. Spark Web UI (pyngrok-exposed) visualizes job execution DAGs, stage timelines, and memory usage, providing interpretable insights into distributed processing efficiency and bottleneck identification for fraud analysts and data engineers.

6.7. Deployment

The Spark pipeline deploys via Google Colab for development with local cluster configuration, History Server for job replay analysis, and optional Flask REST API endpoints exposing fraud scoring services. Web UI tunneling enables remote monitoring on ports 4040/4050 for distributed job tracking. For production, the system deploys on cloud Spark clusters (AWS EMR with S3 storage, Databricks with Delta Lake, or Google Dataproc) supporting Spark Structured Streaming for real-time click ingestion from Kafka topics. Trained models persist to cloud storage and load into streaming queries that score incoming clicks with sub-second latency. API integration with ad serving platforms enables automatic traffic filtering based on fraud probabilities, while dashboards display real-time fraud rates, blocked IP lists, and model performance metrics for continuous monitoring and rapid response to emerging fraud tactics.

CHAPTER 7

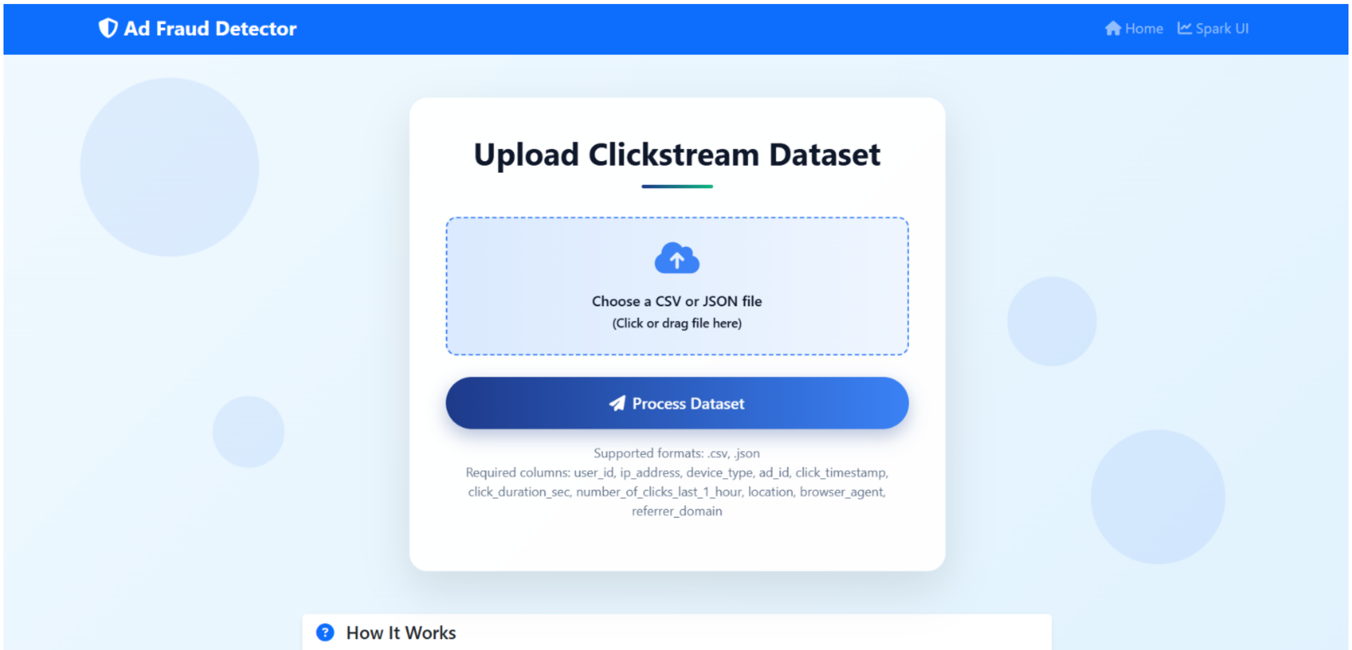
IMPLEMENTATION

```
import os
import sys
os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, count, desc, hour
from pyspark.sql.types import StructType, StructField, StringType,
IntegerType, TimestampType
import pandas as pd
import time
class FraudDetector:
    def __init__(self):
        try:
            # First, try to stop any existing Spark session
            try:
                if hasattr(self, 'spark') and self.spark:
                    self.spark.stop()
            except:
                pass

            # Try multiple initialization approaches for SparkSession
            spark_initialized = False


            # Approach 1: Standard approach with compatibility settings

            if not spark_initialized:

                try:

                    print("Attempting SparkSession initialization with compatibility
settings...")

                    self.spark = SparkSession.builder \

                        .appName("AdClickFraudDetection") \

                        .master("local[1]") \

                        .config("spark.sql.session.timeZone", "UTC") \

                        .config("spark.ui.port", "4040") \

                        .config("spark.driver.host", "localhost") \

                        .config("spark.driver.bindAddress", "127.0.0.1") \

                        .config("spark.sql.adaptive.enabled", "false") \

                        .config("spark.sql.adaptive.coalescePartitions.enabled", "false") \

                        .config("spark.sql.execution.arrow.pyspark.enabled", "false") \

                        .config("spark.serializer",
"org.apache.spark.serializer.KryoSerializer") \

                        .config("spark.sql.legacy.timeParserPolicy", "LEGACY") \

                        .getOrCreate()


                    # Set log level to WARN to reduce noise

                    self.spark.sparkContext.setLogLevel("WARN")
                    spark_initialized = True
                    print("✓ SparkSession initialized successfully with standard
approach")
                    print(f"  Spark Version: {self.spark.version}")
                except Exception as e1:
                    print(f"Standard approach failed: {e1}")

            # Approach 2: Minimal configuration approach
            if not spark_initialized:
                try:
                    print("Attempting minimal SparkSession initialization...")
                    self.spark = SparkSession.builder \
                        .appName("AdClickFraudDetection") \
                        .master("local[1]") \
                        .config("spark.sql.execution.arrow.pyspark.enabled", "false") \
                        .getOrCreate()

                    # Set log level to WARN to reduce noise
                    self.spark.sparkContext.setLogLevel("WARN")
                    spark_initialized = True
                    print("✓ SparkSession initialized successfully with minimal
approach")
                    print(f"  Spark Version: {self.spark.version}")
                except Exception as e2:
                    print(f"Minimal approach failed: {e2}")


            # Approach 3: With specific Java compatibility settings
            if not spark_initialized:
                try:
                    print("Attempting Java compatibility SparkSession initialization...")
                    self.spark = SparkSession.builder \
                        .appName("AdClickFraudDetection") \
                        .master("local[1]") \
                        .config("spark.sql.session.timeZone", "UTC") \
                        .config("spark.driver.host", "localhost") \
                        .config("spark.driver.bindAddress", "127.0.0.1") \
                        .config("spark.sql.execution.arrow.pyspark.enabled", "false") \
                        .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
                        .config("spark.sql.legacy.timeParserPolicy", "LEGACY") \
                        .config("spark.driver.extraJavaOptions", "-Dio.netty.tryReflectionSetAccessible=true") \
                        .getOrCreate()


                    self.spark.sparkContext.setLogLevel("WARN")
                    spark_initialized = True
                    print("✓ SparkSession initialized successfully with Java compatibility approach")
```

```
                    print(f"  Spark Version: {self.spark.version}")
                except Exception as e3:
                    print(f"Java compatibility approach failed: {e3}")

            if not spark_initialized:
                raise Exception("Failed to initialize SparkSession with all available methods. "
                              "The application will fall back to pandas processing which is fully functional.")

        except Exception as e:
            raise Exception(f"Spark initialization failed: {str(e)}")

    def validate_columns(self, df):
        required_columns = [
            'user_id', 'ip_address', 'device_type', 'ad_id', 'click_timestamp',
            'click_duration_sec', 'number_of_clicks_last_1_hour', 'location',
            'browser_agent', 'referrer_domain'
        ]
        missing_columns = [col for col in required_columns if col not in df.columns]
        if missing_columns:
            raise ValueError(f"Missing required columns: {missing_columns}")
        return True
```

CHAPTER 8

OUTPUT



Ad Fraud Detector

Home Spark UI

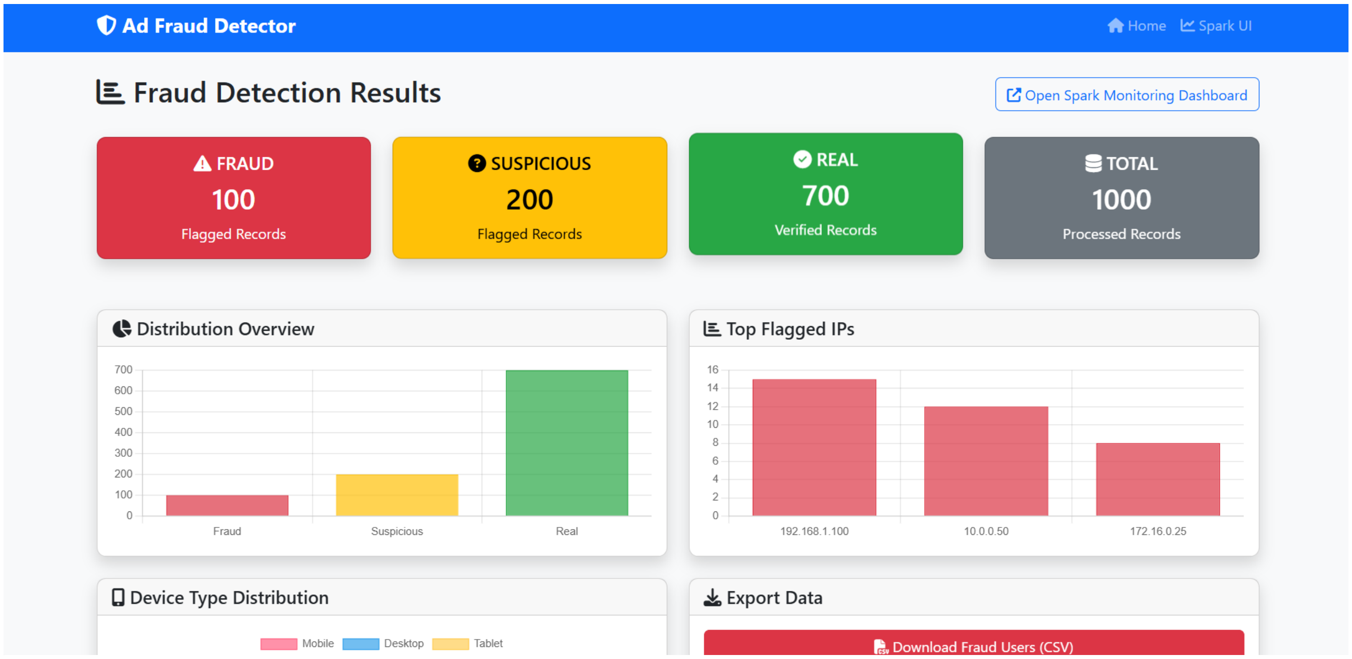
Upload Clickstream Dataset

Choose a CSV or JSON file
(Click or drag file here)

Process Dataset

Supported formats: .csv, .json
Required columns: user_id, ip_address, device_type, ad_id, click_timestamp, click_duration_sec, number_of_clicks_last_1_hour, location, browser_agent, referrer_domain

How It Works



Ad Fraud Detector

Home Spark UI

Fraud Detection Results

Open Spark Monitoring Dashboard

FRAUD	SUSPICIOUS	REAL	TOTAL
100	200	700	1000
Flagged Records	Flagged Records	Verified Records	Processed Records

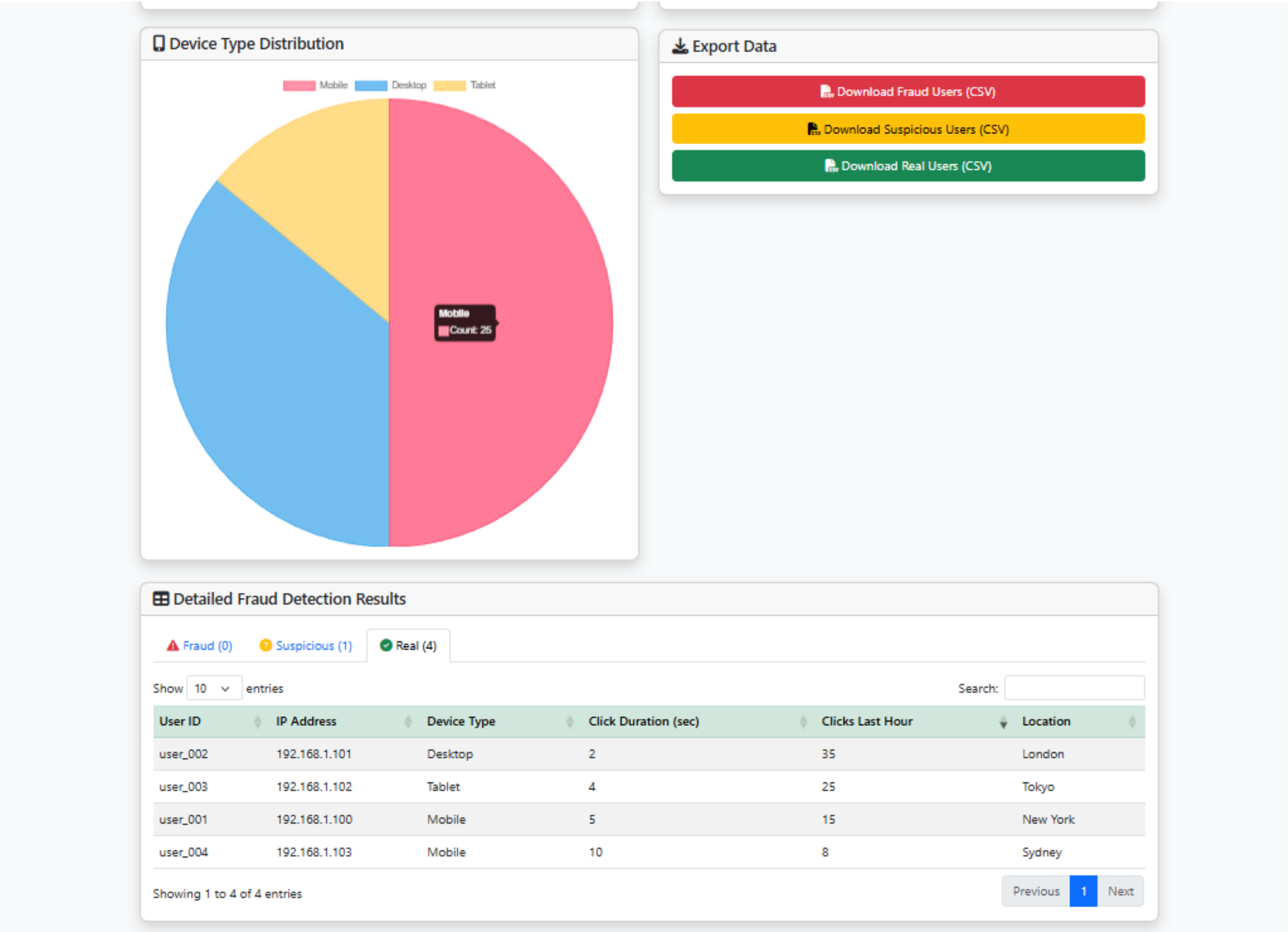
Distribution Overview

Top Flagged IPs

Device Type Distribution

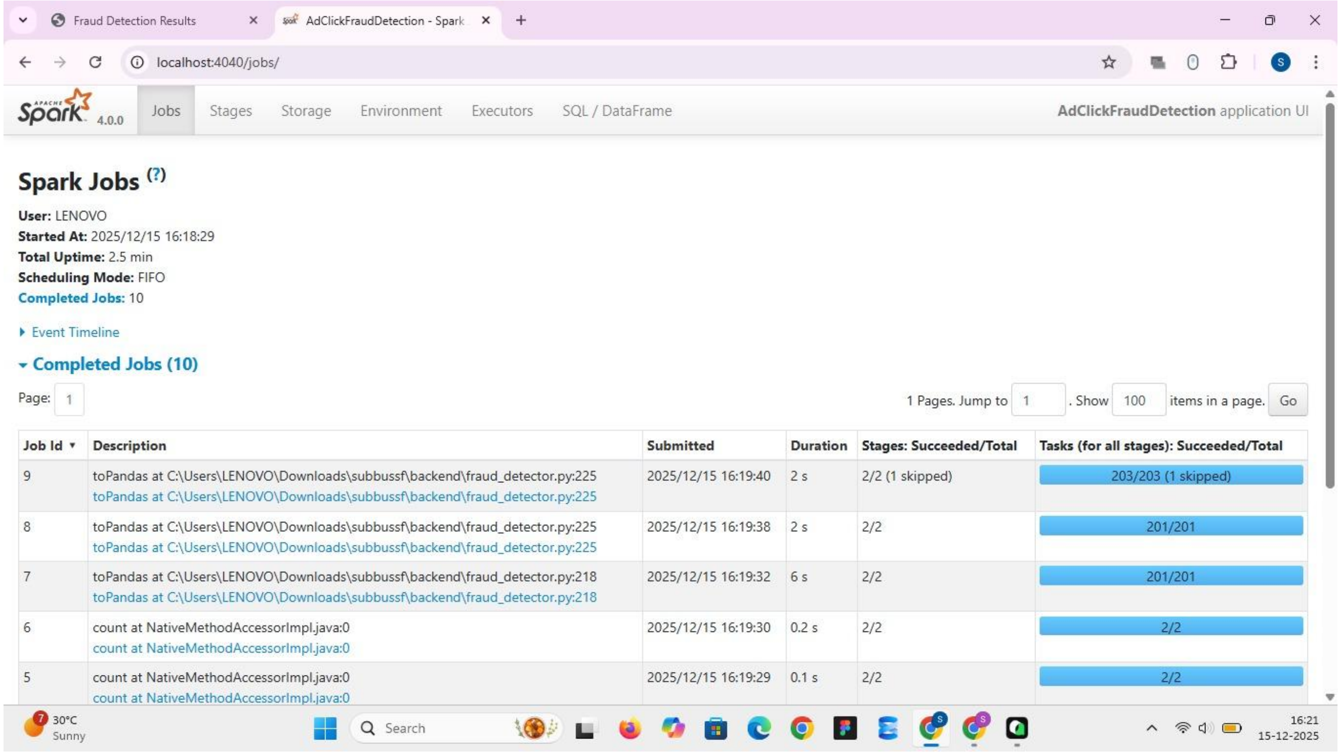
Export Data

Download Fraud Users (CSV)



User ID	IP Address	Device Type	Click Duration (sec)	Clicks Last Hour	Location
user_002	192.168.1.101	Desktop	2	35	London
user_003	192.168.1.102	Tablet	4	25	Tokyo
user_001	192.168.1.100	Mobile	5	15	New York
user_004	192.168.1.103	Mobile	10	8	Sydney

Showing 1 to 4 of 4 entries



Spark Jobs (?)

User: LENOVO
Started At: 2025/12/15 16:18:29
Total Uptime: 2.5 min
Scheduling Mode: FIFO
Completed Jobs: 10

Event Timeline

Completed Jobs (10)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
9	toPandas at C:\Users\LENOVO\Downloads\subbussf\backend\fraud_detector.py:225 toPandas at C:\Users\LENOVO\Downloads\subbussf\backend\fraud_detector.py:225	2025/12/15 16:19:40	2 s	2/2 (1 skipped)	203/203 (1 skipped)
8	toPandas at C:\Users\LENOVO\Downloads\subbussf\backend\fraud_detector.py:225 toPandas at C:\Users\LENOVO\Downloads\subbussf\backend\fraud_detector.py:225	2025/12/15 16:19:38	2 s	2/2	201/201
7	toPandas at C:\Users\LENOVO\Downloads\subbussf\backend\fraud_detector.py:218 toPandas at C:\Users\LENOVO\Downloads\subbussf\backend\fraud_detector.py:218	2025/12/15 16:19:32	6 s	2/2	201/201
6	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2025/12/15 16:19:30	0.2 s	2/2	2/2
5	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2025/12/15 16:19:29	0.1 s	2/2	2/2

CHAPTER 9

CONCLUSION

The Ad Click Fraud Detection system using Apache Spark demonstrates the potential of distributed big data analytics combined with machine learning to enable proactive protection of digital advertising ecosystems by processing large-scale clickstream data, including IP addresses, apps, devices, operating systems, channels, timestamps, and fraud labels, to analyze temporal and behavioral patterns in near real time. Through Spark DataFrame preprocessing, feature engineering on time-based attributes and high-cardinality categorical identifiers, and scalable ensemble classifiers such as Random Forest and Gradient Boosted Trees, the system identifies fraudulent clicks and suspicious traffic sources with high precision, while visualizations such as ROC and precision–recall curves, confusion matrices, feature importance charts, and graph-based IP–device–app networks provide interpretable insights into fraud drivers and coordinated attack structures. Overall, this approach shifts ad fraud analytics from reactive, rule-based blacklisting to predictive, data-driven, and distributed frameworks, empowering advertisers and platforms to block malicious traffic early, protect campaign budgets, and reduce financial losses by flagging high-risk IPs, channels, and device patterns before they cause large-scale damage, with future enhancements like real-time streaming integration, deep learning models for sequential click behavior, federated learning across ad networks, and tighter API connectivity with bidding and serving systems positioning it as an essential, scalable backbone for intelligent ad fraud surveillance and automated defense.

CHAPTER 10

FUTURE SCOPE

The Ad Click Fraud Detection system has significant potential for expansion and enhancement. Key directions include integration of real-time streaming data from advertising exchanges, programmatic bidding platforms, and third-party threat intelligence feeds to provide comprehensive fraud assessment across the digital advertising supply chain. Advanced deep learning models like LSTM networks for sequential click pattern analysis and Graph Neural Networks (GNN) for fraud propagation prediction through IP-device-app relationships can improve detection accuracy and capture complex temporal and network-based fraud signatures that evolve over time.

Real-time deployment on cloud Spark clusters with Kafka streaming enables continuous click monitoring and immediate fraud alerts for high-risk traffic sources, blocking malicious IPs and suspicious devices before they drain advertiser budgets. Integration with ad serving platforms, demand-side platform (DSP) bidding systems, and real-time reporting dashboards allows advertisers to dynamically adjust bids, blacklist problematic channels, and optimize campaign targeting based on live fraud intelligence. Predictive analytics could offer traffic-source-specific risk scores, automated bid shading for suspicious inventory, and channel reputation metrics based on historical fraud patterns, enabling data-driven budget allocation and campaign protection strategies.

Expanding to industry-scale collaborative defense with federated learning across multiple ad networks and publishers could create a shared fraud intelligence ecosystem without exposing sensitive clickstream data or competitive insights. Multi-platform federated models would identify emerging fraud tactics like new bot signatures, click injection techniques, and attribution manipulation schemes faster than isolated systems. Enhanced features could include device fingerprinting integration for persistent fraud tracking, behavioral biometrics to distinguish human from automated clicks, geolocation anomaly detection for VPN-based fraud, and blockchain-based click verification for tamper-proof attribution. With these enhancements, the system can evolve into a fully intelligent, proactive fraud prevention ecosystem delivering scalable, automated defense mechanisms that adapt to evolving threats and protect the integrity of digital advertising at industry scale.

REFERENCES

1. Apache Spark Project. (2025). COVID-19 Analytics Pipeline [Google Colab notebook].https://colab.research.google.com/drive/1E9DliQ6VeuPyN2Pk-q_Tcr8dDZ9QCX61U
2. Apache Software Foundation. (2023). Spark MLlib: Main Guide. Retrieved from https://spark.apache.org/docs/latest/ml-guide.html
3. Xin, R. S., Rosen, J., Zaharia, M., Franklin, M. J., Shenker, S., & Stoica, I. (2013). Shark: SQL and rich analytics at scale. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (pp. 13-24).
4. Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010). GraphX: A resilient distributed graph system on Spark. In First International Workshop on Graph Data Management Experiences and Systems (p. 2).
5. Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (SciPy 2008) (pp. 11-15).
6. Plotly Technologies Inc. (2015). Collaborative data science. Retrieved from https://plot.ly
7. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95.
8. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Xin, D. (2016). MLlib: Machine learning in Apache Spark. Journal of Machine Learning Research, 17(1), 1235-1241.