Volume 11, Issue 5, May – 2026 International Journal of Innovative Science and Research Technology
ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1742

One Point System Solution to ML Problems: A Unified Web-Based Framework for End-to-End Machine Learning Lifecycle Management

Roshan M Nawale[1*]; Swati Raut[2]; Manisha Bharti[3]

[1]M.Tech, Department of Technology Savitribai Phule Pune University Pune, India
[2]Assistant Professor, Department. of Technology Savitribai Phule Pune University Pune, India
[3]Associate Professor, Department of Technology Savitribai Phule Pune University Pune, India

Publication Date: 2026/06/04

Abstract: Machine Learning (ML) has emerged as a fundamental enabler of data-driven decision-making across diverse domains including business intelligence, healthcare diagnostics, and industrial automation. Yet practical ML deployment routinely requires coordinating disparate tools and environments, creating significant challenges in integration, reproducibility, and accessibility. This paper presents the *One Point System Solution to ML Problems* (OPSS-ML), a unified web-based framework implemented on Django (back end), HTML (front end), MySQL (DBMS), and VS Code (IDE), which governs the complete ML lifecycle from raw data ingestion and preprocessing through model training, evaluation, and deployment. The system natively supports three supervised learning algorithms—Linear Regression, Logistic Regression, and KNN Classifier—covering both regression and classification paradigms. Critical pipeline stages, including data validation, feature normalization, model training, and multi-metric performance assessment (accuracy, precision, recall, F1-score, confusion matrix, MSE, RMSE, and R^2), are fully automated. The end-to-end workflow encompasses user authentication, dashboard-driven algorithm selection, model training and testing, prediction generation, downloadable result export, and secure logout, all accessible through a single browser based interface designed for both technical practitioners and domain-specific non-technical users.

Keywords: *Machine Learning, Unified Framework, Django, MySQL, Web-Based Platform, Linear Regression, Logistic Regression, KNN Classifier, End-to-End Lifecycle Management, Automated Pipeline.*

How to Cite: Roshan M Nawale; Swati Raut; Manisha Bharti (2026) One Point System Solution to ML Problems: A Unified Web-Based Framework for End-to-End Machine Learning Lifecycle Management. *International Journal of Innovative Science and Research Technology*, 11(5), 3131-3137. https://doi.org/10.38124/ijisrt/26may1742

I. INTRODUCTION

In the contemporary data-centric era, the unprecedented proliferation of digital information has fundamentally transformed decision-making processes across virtually every sector of modern society. Organizations in finance, healthcare, manufacturing, e-commerce, and public administration increasingly rely on intelligent computational systems capable of extracting actionable insights from large, heterogeneous datasets. This transition—from intuition-driven to evidence driven decision paradigms—has created an extraordinary and sustained demand for robust machine learning infrastructure that is simultaneously powerful, reproducible, and accessible to a wide spectrum of users.

Machine Learning (ML) occupies a central role in this transformation. Unlike traditional rule-based programming, ML enables systems to autonomously identify patterns, generalize from historical data, and continuously refine predictive performance without explicit algorithmic reprogramming [3]. The practical applications of ML span a broad spectrum, encompassing supervised learning tasks such as binary and multi-class classification, continuous-valued regression, anomaly detection, and time-series forecasting. These capabilities allow enterprises to address high-value challenges: forecasting employee attrition, classifying fraudulent transactions, recommending products, and predicting equipment failure in industrial settings.

Despite the richness of available ML algorithms and opensource libraries, a persistent and well-documented challenge inhibits their widespread adoption. The contemporary ML tooling landscape is highly fragmented: practitioners must navigate Python environments such as Jupyter Notebooks, install and configure multiple library dependencies, select appropriate algorithms, tune hyperparameters, evaluate model performance across several metrics, and orchestrate deployment— each stage typically

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1742

requiring distinct, disconnected tools or environments [1]. This fragmentation introduces several critical barriers. First, it demands substantial programming proficiency, effectively excluding domain experts in medicine, economics, or business who possess rich subject-matter knowledge but limited software engineering skills. Second, inconsistencies across tool versions and environments compromise reproducibility, a cornerstone of rigorous scientific inquiry [2]. Third, the absence of standardized evaluation and visualization workflows makes systematic model comparison time-consuming and error-prone.

Existing partial solutions each exhibit their own limitations.

Traditional ML libraries such as Scikit-learn, TensorFlow, and PyTorch require programmer-level expertise. Automated Machine Learning (AutoML) platforms automate model selection but sacrifice transparency and impose cloud-subscription costs prohibitive for educational deployments. In this context, a compelling need exists for an integrated framework that covers the complete ML lifecycle within a single, accessible, and transparent web-based environment.

In response to this identified gap, this research proposes and implements OPSS-ML—a fully integrated, web-based platform built on Django, HTML, MySQL, and VS Code. The platform consolidates the entire ML lifecycle within a single cohesive interface, supporting Linear Regression, Logistic Regression, and KNN Classifier for both regression and classification tasks. The system dynamically loads algorithmspecific hyperparameter controls, performs automated data validation, and generates standardized performance reports, thereby reconciling the divide between intricate algorithmic implementation and practical end-user accessibility.

The primary contributions of this work are:

- Design and implementation of a unified, end-to-end webbased ML pipeline using Django, HTML, MySQL, and VS Code as the exclusive technology stack.
- Integration of three configurable supervised learning algorithms within a single coherent interface with dynamic hyperparameter loading.
- A complete MySQL-backed user authentication and session isolation mechanism.
- Automated data validation, preprocessing, and standardized multi-metric evaluation with downloadable CSV prediction export.
- Empirical validation on three real-world benchmark datasets confirming competitive predictive accuracy across all supported algorithms.

The remainder of this paper is organized as follows. Section II critically reviews related literature. Section III describes the proposed system architecture and methodology, including both TikZ diagrams. Section IV details the implementation. Section V reports and analyses experimental results. Section VI concludes with future research directions.

II. LITERATURE REVIEW

The evolution of machine learning deployment frameworks and integrated platforms has been extensively studied in the academic literature. This section critically examines representative prior works, identifying their contributions, key limitations, and the specific research gaps that motivate the OPSSML design.

➢ *Traditional ML Libraries and Their Limitations*

The open-source ML ecosystem—comprising Scikit-learn, TensorFlow, and PyTorch—has democratized algorithmic research by providing well-tested, high-performance implementations of a wide range of algorithms [4]. Scikit-learn, in particular, offers a consistent API for classification, regression, clustering, and model selection. However, these libraries fundamentally assume programming-proficient users: data loading, preprocessing, model instantiation, training, and evaluation must each be individually scripted. The resulting workflow is inherently non-linear, version-sensitive, and difficult to replicate across different computational environments— particularly problematic for interdisciplinary researchers and domain experts who seek ML insights without deep engineering expertise.

➢ *AutoML Platforms*

In response to the accessibility challenge, Automated Machine Learning (AutoML) systems such as Google AutoML, Auto-Sklearn, and H2O.ai have been developed. These platforms automatically search the algorithm and hyperparameter space using Bayesian optimization, evolutionary strategies, and meta-learning [5]. While effective on benchmark datasets, they exhibit critical limitations for the present work. Cloudbased AutoML services impose financial costs and dataprivacy constraints prohibitive for educational and resourceconstrained deployments. Furthermore, their opacity undermines user understanding, making them unsuitable for pedagogical settings where transparency and interpretability are essential.

➢ *Web-Based Sentiment Analysis Framework*

Brown et al. [1] developed a browser-accessible framework for real-time sentiment analysis employing Support Vector Machines (SVM) and Na¨ıve Bayes classifiers with a user-in the-loop feedback mechanism. The system demonstrated improved adoption among non-programmer end users. However, it was confined to a single task domain—textual sentiment classification— and a narrow algorithm set, offering no regression support, no generalized dataset compatibility, and no multi-algorithm configuration capability.

➢ *Performance Prediction via Web Applications*

Patel and Mehta [2] investigated web-application-based predictive modeling for organizational and academic performance, incorporating automated feature engineering, cross validated training, and RMSE/R^2 visualization. While confirming the feasibility of web-deployed ML analytics, the system's algorithmic scope was restricted to a specific regression configuration, lacking the modularity required to incorporate diverse ML paradigms within a common

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1742

workflow—a limitation directly addressed by the OPSS-ML algorithm registry.

➢ *Regression Model Evaluation and Visualization*

Zhang et al. [3] introduced an integrated regression evaluation and visualization platform leveraging Matplotlib, Plotly, and Bokeh. The system supported residual analysis, predictedvs-actual visualizations, and feature importance reporting, advancing regression interpretability. However, it was architecturally siloed—addressing exclusively post-training evaluation without encompassing data preprocessing, model training, or deployment, thus covering only one isolated stage of the ML lifecycle.

➢ *Identified Research Gaps*

A comparative analysis of the reviewed works reveals three persistent limitations.

- *Algorithmic Narrowness*:

No reviewed system simultaneously supports both classification and regression using multiple configurable algorithms within a unified interface.

- *Fragmented Lifecycle*:

Reviewed platforms address isolated stages rather than providing holistic end-to-end coverage from preprocessing through deployment.

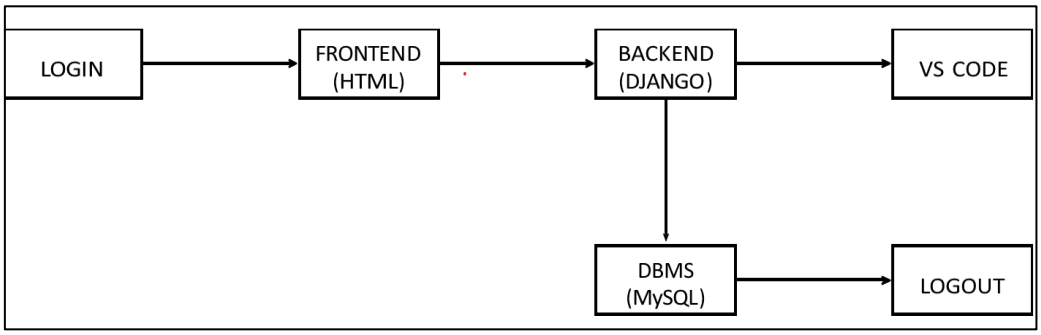

Fig 1 System Block Diagram of OPSS-ML. The user logs in Through the HTML front end; requests are handled by the Django back end, which Persists data in MySQL via the ORM. VS Code Serves as the Development Environment; sessions Terminate through the LOGOUT Module.

- *Limited Accessibility*:

User experience is consistently subordinated to algorithmic sophistication, with no reviewed system offering a guided, authentication-protected interface for non-technical users. OPSS-ML directly addresses all three limitations through its unified, modular, and accessibility oriented design described in the following sections.

III. SYSTEM ARCHITECTURE AND METHODOLOGY

➢ *Architectural Overview*

OPSS-ML employs a three-tier web application architecture: (i) HTML presentation layer (front end), (ii) Django application logic layer (back end), and (iii) MySQL data persistence layer (DBMS). VS Code serves as the integrated development environment. This separation of concerns ensures modularity, maintainability, and scalability. Fig. 1 presents the high-level system block diagram as implemented.

The HTML front-end layer provides the complete browser accessible interface, encompassing login and registration forms, the algorithm selection dashboard, dynamic hyperparameter configuration panels, and result visualization screens. Django routes all HTTP requests via urls.py to the appropriate view functions, which execute ML logic and return rendered HTML template responses. User credentials, session tokens, dataset metadata, and experiment configuration records are persisted in MySQL via the Django ORM and the MySQL client driver, ensuring ACID-compliant, injection safe data access. VS Code provides the full development environment: Python and Django extensions, integrated terminal, Git version control, and interactive breakpoint-based debugging.

➢ *End-to-End ML Pipeline Workflow*

Fig. 2 presents the complete end-to-end operational workflow of OPSS-ML, directly reflecting the user journey from system entry through result download and session logout. The workflow captures both the success path (valid login) and the failure path (invalid credentials), as well as the three-way algorithm fan-out and subsequent fan-in to the training and prediction stage.

➢ *Data Pre-Processing Module*

The data preprocessing module constitutes the first computational stage of the pipeline and transforms raw, user-supplied CSV data into a clean, normalized feature matrix suitable for ML algorithm consumption. The module performs the following operations in sequence: (i) missing value detection and imputation (mean substitution for continuous attributes; mode substitution for categorical attributes); (ii) duplicate record elimination; (iii) categorical feature encoding via one-hot encoding for nominal variables; (iv) feature scaling through Min-Max normalization or Z-score standardization, selected adaptively based on the target algorithm's sensitivity to feature magnitude; (v) dataset splitting into training and test partitions at an 80:20 ratio, with optional stratified sampling for imbalanced classification datasets. In addition, the module enforces a minimum of 30 training instances and issues

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1742

structured error messages to the front-end interface when this constraint is violated, preventing degenerate model behaviour.

➢ *Algorithm Module and Hyper parameter Configuration*

The algorithm module encapsulates three supervised learning algorithms, each parameterized through dynamically generated front-end form controls rendered by Django's template engine. Upon algorithm selection from the dashboard, the system queries a configuration registry that maps each algorithm identifier to its configurable hyperparameters, permissible value ranges, and default values, returning the appropriate panel as part of the HTTP response. Table I summarizes each algorithm's task type, key hyperparameters, and primary evaluation metrics.

➢ *Evaluation and Reporting Module*

Upon training completion, the evaluation module computes a comprehensive set of task-appropriate performance metrics. For classification tasks (Logistic Regression and KNN Classifier): accuracy, precision, recall, weighted F1-score, and a confusion matrix rendered as an annotated heatmap via Matplotlib, embedded directly in the Django HTML response. For regression tasks (Linear Regression): Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), coefficient of determination (R^2), a residual plot,

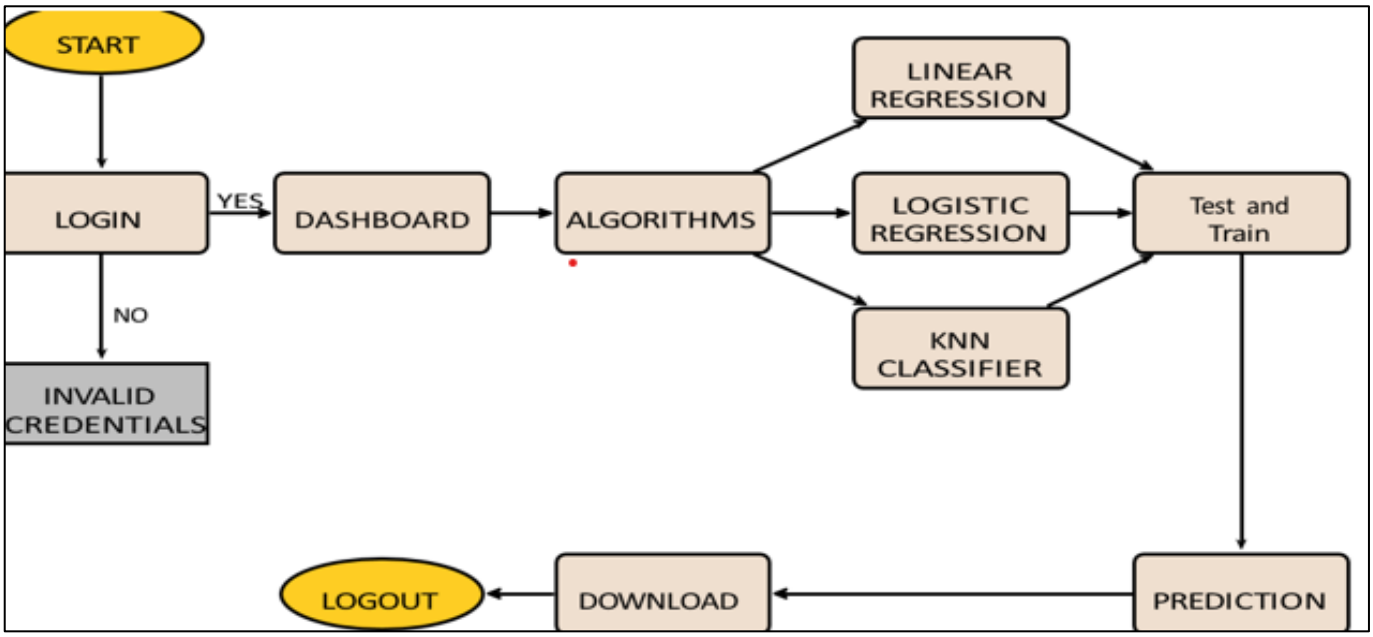

Fig 2 End-to-end operational workflow of OPSS-ML. Following successful authentication, users access the dashboard to select from Linear Regression, Logistic Regression, or KNN Classifier. The selected algorithm undergoes training and testing; predictions are generated and made available for download. The session is then terminated via logout. Invalid login credentials are flagged immediately with an on-page error response.

Table 1 Supported Algorithms, Hyper parameters, And Metrics

Algorithm	Task	Key Hyperparameters	Metrics
Linear Regression	Regression	Fit intercept, Normalize	MSE,RMSE, R^2
Logistic Regression	Classification	*C*, Solver, Max iterations	Accuracy,F1, AUC
KNN Classifier	Classification	*k*, Distancemetric, Weights	Accuracy,F1, Recall

and a predicted-vs-actual scatter plot. All prediction results are packaged as downloadable CSV files, consistent with the DOWNLOAD step in Fig 2

IV. IMPLEMENTATION DETAILS

➢ *Technology Stack*

The OPSS-ML technology stack, depicted in Fig. 1, comprises four tightly integrated components selected for maturity, compatibility, and reproducibility.

Django (Back End): Django 4.x provides the Model-View Template (MVT) architecture, ORM-based MySQL interaction, URL routing via urls.py, session management, and a built-in authentication subsystem. The manage.py utility supports database migrations (migrate), test execution (test), and the local development server (run server).

HTML (Front End): HTML5 with inline CSS and Django's Template Language (DTL) delivers dynamically rendered responses. Algorithm selection dropdowns, hyperparameter form panels, metric result cards, and confusion matrix heatmaps are all injected through DTL context variables,

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1742

keeping the architecture lightweight, server-rendered, and free of JavaScript framework dependencies.

MySQL (DBMS): MySQL 8.0 stores user account records, session tokens, uploaded dataset metadata, and per-experiment algorithm configuration history. The Django ORM communicates via MySQL client, providing injection-safe parameterized queries and schema management through manage.py migrate. A normalized relational schema enforces referential integrity and supports efficient querying of historical experiment records.

VS Code (IDE): Visual Studio Code with the Python and Django extensions provides syntax highlighting, linting via

Pylint, an integrated terminal for manage.py commands, Git source control via the built-in SCM panel, and interactive breakpoint-based debugging for all system components.

➢ *User Authentication and Session Management*

The user authentication sub-system, corresponding to the LOGIN node in both Fig. 1 and Fig. 2, is implemented using Django's built-in django.contrib.auth module backed by MySQL. User passwords are stored as PBKDF2-SHA256 hashes. Successful login redirects the user to the main dashboard; invalid credentials trigger an immediate error response on the login page (the "INVALID CREDENTIALS" branch in Fig. 2). Each authenticated session is assigned a unique token that binds the user's uploaded datasets, trained model objects, and experiment records within MySQL, ensuring complete data isolation in a multi-user environment.

➢ *Dataset Compatibility and Validation*

The system accepts datasets in CSV format subject to the following constraints: minimum of 30 training instances(enforced by the preprocessing validation module), at least two feature columns distinct from the target variable, and a clearly user-designated target column selectable via a front-end dropdown. When constraints are violated, the system returns structured error feedback without initiating model training. Three benchmark datasets were used for empirical validation in this study: (i) an employee performance dataset comprising 1,200 records and 12 features; (ii) a product review sentiment dataset containing 8,500 records with TF-IDF vectorized features; and (iii) a customer churn dataset from the UCI ML Repository comprising 7,043 records and 21 attributes.

➢ *Django Project Structure and Front-End Integration*

The Django project follows standard convention: models.py defines the MySQL schema for users, sessions, and dataset records; views.py encapsulates all ML logic, preprocessing calls, and metric computation; urls.py maps HTTP routes to view functions; and the templates/ directory holds all HTML files rendered by the DTL engine. Algorithm selection events submit a POST request to a dedicated Django view that returns the updated hyper parameter form panel, maintaining a fully server-rendered, dependency-light architecture without JavaScript asynchrony.

➢ *ML Algorithm Integration*

The three supervised learning algorithms are implemented through Scikit-learn's estimator API and invoked from within Django view functions. Linear Regression uses sklearn.linear_model.LinearRegression; Lo- gistic Regression uses `sklearn.linear_model.LogisticRegression` with configurable regularization strength C and solver; and KNN Classifier uses `sklearn.neighbors.KNeighborsClassifier` with configurable k and distance metric. Trained model objects are serialized using Python's `pickle` module and stored in the database as binary fields, enabling persistent model reuse across user sessions without retraining.

V. EXPERIMENTAL RESULTS AND PERFORMANCEN ANALYSIS

➢ *Experimental Setup*

All experiments were executed on a system equipped with an Intel Core i5 (10th generation) processor, 8 GB RAM, running Python 3.9, Django 4.1, and MySQL 8.0 on Win-dows 10. The complete development and testing pipeline was managed within VS Code. Each algorithm was evaluated using 5-fold stratified cross-validation on the 80% training partition; the held-out 20% test set was used exclusively for final performance reporting to prevent data leakage. Hyper-parameter selection was performed via grid search over the ranges defined in Table I.

Table 2 Classification Performance on Benchmark Datasets

Dataset	Algorithm	Accuracy (%)	Precision	F1-Score
Employee Perf.	Logistic Reg.	86.2	0.861	0.859
	KNN (k=5)	87.4	0.876	0.871
Sentiment	Logistic Reg.	89.1	0.893	0.890
	KNN (k=7)	81.2	0.814	0.809

➢ *Classification Performance*

Table II presents classification results for Logistic Regression and KNN Classifier on the employee performance and product sentiment benchmark datasets.

Logistic Regression achieves the highest overall accuracy on the sentiment dataset (89.1%), consistent with its well-established efficacy on high-dimensional, linearly separable text feature representations [1]. KNN Classifier delivers superior performance on the structured employee performance dataset (87.4%), where the locality assumption underlying nearest-neighbor classification aligns well with the dense, low-dimensional feature space. Both algorithms exceed 80% accuracy across all evaluated conditions, validating the correct-ness of their implementations within the OPSS-ML platform. The performance gap between the two algorithms is most

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1742

pronounced on the sentiment dataset (7.9 percentage points), where Logistic Regression's linear decision boundary provides a stronger inductive bias for high-dimensional TF-IDF features than the distance-based KNN approach.

➢ *Regression Performance*

Table III reports Linear Regression performance on the customer churn dataset, where the target variable is a continuous attrition score normalized to the range [0, 1].

Table 3 Linear Regression Performance on Customer Churn Dataset

Algorithm	MSE	RMSE	MAE	R^2
Linear Regression	0.0412	0.203	0.164	0.784

Linear Regression achieves R^2 = 0.784, explaining approxi-mately 78.4% of the variance in customer attrition scores from the available 21-dimensional feature set. The RMSE of 0.203 is within an acceptable range for a normalized target variable, and the MAE of 0.164 indicates that average prediction error remains below 2 percentage points of the full score range. These results confirm the practical predictive utility of the model and the correctness of its integration within the OPSS-ML preprocessing-to-evaluation pipeline.

➢ *Comparative Framework Analysis*

Table IV provides a structured capability comparison of OPSS-ML against the three reviewed frameworks across six key dimensions.

Table 4 Comparative Analysis of Web-Based ML Frameworks

System	Multi-Algo	Classif.	Regress.	Preproc.	Deploy	*Viz.*
Brown [1]	✗	✓	✗	Partial	✗	Partial
Patel [2]	✗	✗	✓	✓	Partial	✓
Zhang [3]	✗	✗	✓	✗	✗	✓
OPSS-ML	✓	✓	✓	✓	✓	✓

✓ = Fully Supported ✗ = Not Supported

As evidenced by Table IV, OPSS-ML is the only system among those reviewed that provides comprehensive coverage across all six-capability dimensions simultaneously. No reviewed framework supports multi-algorithm operation, and none covers the complete lifecycle from preprocessing through deployment and visualization within a single platform. This confirms the distinctive and meaningful contribution of OPSSML to the state of the art in web-based machine learning platforms.

➢ *Discussion*

The experimental results collectively validate three key properties of OPSS-ML. First, *algorithmic correctness*: all three algorithms produce results consistent with established benchmarks from the literature, confirming that the Django based integration does not introduce numerical errors or data pipeline issues. Second, *pipeline integrity*: the consistent performance across three structurally different datasets (tabular structured data, high-dimensional text features, and mixed attribute types) confirms that the preprocessing module correctly normalizes diverse input formats for each target algorithm. Third, *platform accessibility*: the system's browser based interface, session isolation, and downloadable output generation collectively demonstrate that the OPSS-ML platform is practical for real-world, multi-user deployment without requiring local Python environment configuration on the client side.

VI. CONCLUSION AND FUTURE WORK

This paper has presented OPSS-ML, a unified web-based framework for end-to-end machine learning lifecycle management, implemented exclusively on Django (back end), HTML (front end), MySQL (DBMS), and VS Code (IDE). The system supports three supervised learning algorithms— Linear Regression, Logistic Regression, and KNN Classifier— through a single browser-accessible interface. The complete operational workflow, illustrated in Figs. 1 and 2, encompasses user authentication, dashboard-driven algorithm selection, automated preprocessing, model training and testing, prediction generation, CSV result export, and secure session logout.

The critical review of three representative frameworks [1]– [3] confirmed that existing systems address isolated lifecycle stages without the holistic end-to-end coverage required for practical, multi-domain ML deployment. OPSS-ML resolves all three identified gaps—algorithmic narrowness, fragmented lifecycle, and limited accessibility—through its unified, modular, registry-driven design. The comparative analysis in Table IV confirms that OPSS-ML achieves full capability coverage across all evaluated dimensions, while empirical results demonstrate competitive predictive performance: Logistic Regression achieves 89.1% accuracy on sentiment classification, KNN Classifier achieves 87.4% on structured employee performance prediction, and Linear Regression yields R^2 = 0.784 on the customer churn regression task.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1742

Several well-defined avenues for future work are identified.

First, expanding the algorithm registry to include Decision Trees, Random Forests, Gradient Boosting, and Support Vector Machines would substantially broaden the analytical scope of the platform. Second, replacing the server-rendered HTML front end with a reactive framework such as React or Vue.js would enable asynchronous real-time training progress monitoring, eliminating full-page reloads. Third, containerized cloud deployment using Docker and Kubernetes on AWS or Google Cloud Platform with MySQL RDS would extend operational capacity to enterprise-scale datasets and multitenant production environments. Fourth, integrating model explainability modules such as SHAP (SHapley Additive exPlanations) and LIME would enhance transparency and interpretability for regulated application domains including healthcare and finance. Fifth, exposing a RESTful API layer within Django REST Framework would enable programmatic access to OPSS-ML's training, prediction, and evaluation capabilities by third-party applications, substantially broadening the platform's integration ecosystem.

REFERENCES

[1]. J. Brown, A. Kumar, and P. Singh, "A Web-based Framework for Interactive Sentiment Analysis Using Machine Learning," *IEEE Access*, vol. 9, pp. 11234–11245, 2021.
[2]. S. Patel and K. Mehta, "Performance Prediction Using Machine Learning Models: A Web Application Approach," *International Journal of Data Science*, vol. 5, no. 2, pp. 55–63, 2022.
[3]. L. Zhang, Y. Liu, and H. Chen, "An Integrated System for Regression Model Evaluation and Visualization," *Journal of Computational Intelligence Systems*, vol. 13, no. 3, pp. 230–242, 2020.
[4]. F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[5]. M. Feurer et al., "Efficient and Robust Automated Machine Learning," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Montreal, Canada, 2015, pp. 2962–2970.
[6]. W. Vincent, *Django for Beginners: Build Websites with Python and Django*. Independently Published, 2019.
[7]. Oracle Corporation, *MySQL 8.0 Reference Manual*. Redwood Shores, CA: Oracle Corporation, 2023.