IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

A Django-Based Machine Learning Web Application To Ensure Data Quality In Csv Files And Train A Predictive Model

1 Mrs G Amala, 2 Mohammed Saad Mansoor, 3 Mohammed Huzaifa Mansoor, 4 Thejas H J
1 Assistant Professor, 2 Student, 3 Student, 4 Student
Department of Artificial Intelligence and Machine Learning, Vijaya Vittala Institute of Technology, Bengaluru, India

Abstract: Ensuring the quality of tabular data is a critical preprocessing step before training reliable predictive models. We present a web-based application built with Django that automates data quality assessment and cleaning on uploaded CSV datasets, followed by training of a supervised learning model. The system ingests user-provided CSV files and analyzes issues such as missing values, outliers, and skewed features, providing visual dashboards summarizing key data quality metrics. Automated cleaning operations (mean/mode imputation, outlier clipping via the IQR rule, log-transform for skewed features, and one-hot encoding of categoricals) are performed using Python libraries (Pandas, SciPy, scikit-learn), producing a cleaned dataset ready for analysis. Users can then select a target variable and train multiple regression/classification algorithms (e.g. linear regression, random forest, k-NN), with the best model highlighted by evaluation metrics (e.g. mean squared error). In a case study on a housing price dataset, the application identified approximately 5.97% missing values, 1.47% outliers, and 25.0% highly skewed features, yielding a composite data quality score of 67.56%. A Random Forest regressor achieved the lowest error among models, consistent with known performance. The interface displays feature correlations and per-unit effect sizes, along with interactive charts. This integrated solution streamlines the machine learning pipeline by combining statistical data cleaning and model training in a single Django-based framework.

Index Terms – Data Quality, Data Cleaning, Django, Machine Learning, Predictive Model, CSV Data, Web Application.

1. Introduction

Data-driven decision-making relies on high-quality datasets; poor data quality (missing values, outliers, inconsistencies) can severely degrade machine learning outcomes. In practice, many real-world datasets (often in CSV format) require extensive cleaning before modeling. The Python Pandas library provides powerful tools for efficient data manipulation, and scikit-learn offers a suite of algorithms for predictive modeling. However, these tools are typically used in standalone scripts rather than integrated, user-friendly applications. Meanwhile, the Django framework provides a robust web application architecture (Model-View-Template pattern) suitable for hosting data-intensive applications.

Current ML workflows often treat data cleaning as a separate, manual step. Analysts must inspect CSV files for missing or erroneous entries, which is time-consuming and error-prone. Non-expert users lack easy-to-use platforms to ensure dataset quality. Motivated by these needs, we design a Django web application that automatically assesses and improves CSV data quality, then trains and evaluates a predictive model via an interactive user interface. Our system integrates statistical preprocessing and model building, all accessible through a web browser, thereby reducing the gap between raw data and actionable insights.

The remainder of this paper is organized as follows. Section 2 reviews related work on data cleaning and web-based ML tools. Section 3 describes the proposed methodology and system design. Section 4 presents experimental results from a case study and discusses the findings. Finally, Section 5 concludes the paper and outlines future work.

1.1 **Problem Statement**

Current ML pipelines often treat data cleaning as a separate, manual step. Analysts must manually inspect CSV files for missing data, erroneous entries, and non-standard formats, which is time-consuming and error-prone. Without an easy-to-use platform, non-experts struggle to ensure dataset quality..

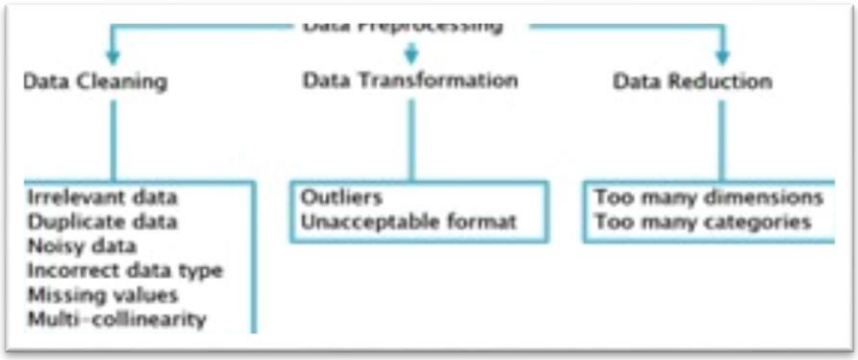

Figure 1 : Data Preprocessing

2. Literature Review

Research on data cleaning emphasizes its critical role in data mining and machine learning. Numerous studies highlight that poor data quality can compromise model accuracy and reliability. For example, Rahm and Do [7] survey various data quality problems and current cleaning approaches, stressing that automated frameworks can greatly reduce manual effort. In recent work, Dasari and Varma [6] demonstrate how Python-based pipelines can employ multiple data cleaning techniques (e.g. imputation, filtering) to improve data quality in practice. These investigations concur that leveraging standard libraries (Pandas, OpenRefine, Data Wrangler, etc.) helps in removing invalid values, normalizing formats, and detecting anomalies before modeling.

In parallel, web applications have been developed to bring machine learning capabilities to end users without programming. For instance, a recent study demonstrated a Django-based web app for music genre classification by integrating Django with scikit-learn (support vector machines) in the backend. Such efforts illustrate that Django can host machine learning pipelines within a user-facing interface. Leveraging this insight, our work similarly uses Django's web framework to expose data analysis and ML functionality to users.

Key Python libraries underpin our methodology. Wes McKinney (2010) introduced Pandas as providing "fundamental building blocks" for statistical computing and data analysis in Python [13], facilitating efficient handling of tabular data. Likewise, Pedregosa et al. (2011) describe scikit-learn as a comprehensive, easy-to-use library for machine learning in Python [14]. We adopt these proven tools in our system: Pandas and SciPy for preprocessing, scikit-learn for model construction, and Django/MVT (Model-View-Template) as the web framework to orchestrate the pipeline. By building on this prior work, our application brings together automated data cleaning and predictive modeling in an integrated platform.

3. Methodology

Our proposed system follows Django's Model-View-Template (MVT) architecture and implements the data science workflow in the following components:

3.0 Proposed System

Our system follows Django's MVC (MTV) architecture, starting with a web UI for CSV upload and pandas-based ingestion, allowing target/index column selection. Data quality is evaluated via missing values, outliers (IQR), and skewness, producing a composite score; cleaning involves imputation, one-hot encoding, clipping, and log transformation using Pandas/Scikit-learn. Afterward, models (e.g. linear regression, KNN, random forest) are trained and evaluated, with UI visualizations (e.g. Seaborn charts) rendered via Django views and templates.

Figure 2 : Working Methodology

3.1 Data Ingestion

The web interface presents a file-upload form to the user. Uploaded CSV files are read on the server using Pandas (pd.read_csv), enabling immediate inspection of the dataset as a DataFrame. Users may optionally indicate a target column for prediction or drop unwanted index columns. The backend stores the raw data for subsequent analysis.

3.2 Data Quality Analysis and Scoring

Once data is ingested, the application computes key quality metrics. A perform_quality_check(df) function calculates: the percentage of missing entries per column (using df.isnull()), the percentage of outliers per column based on the interquartile range (IQR) rule, and the percentage of numeric features exhibiting high skewness (using scipy.stats.skew()). For outliers, any value outside 1.5×IQR from the median is counted. These metrics are combined into a composite **Data Quality Score** as:

Quality Score=100%−(%missing+%outliers+%skewed)\text{Quality Score} = 100\% - (\%\text{missing} + \%\text{outliers} + \%\text{skewed}).

This single score provides a quick summary of data cleanliness. All metrics are passed to the web interface and visualized (e.g. as charts of missing vs. valid values, etc.) to help users understand the dataset issues.

Figure 3 : Data Quality Analysis

3.3 Data Cleaning

The clean_data(df) routine applies standard cleaning operations. First, it separates numeric and categorical columns using df.select_dtypes(). For numeric features, missing values are imputed using the mean (via scikit-learn's SimpleImputer(strategy='mean')). For categorical features, missing values are imputed with the most frequent category. After imputation, categorical variables are transformed using one-hot encoding

(OneHotEncoder(drop='first')). Numeric outliers are clipped to the 1.5×IQR range to mitigate extreme values.

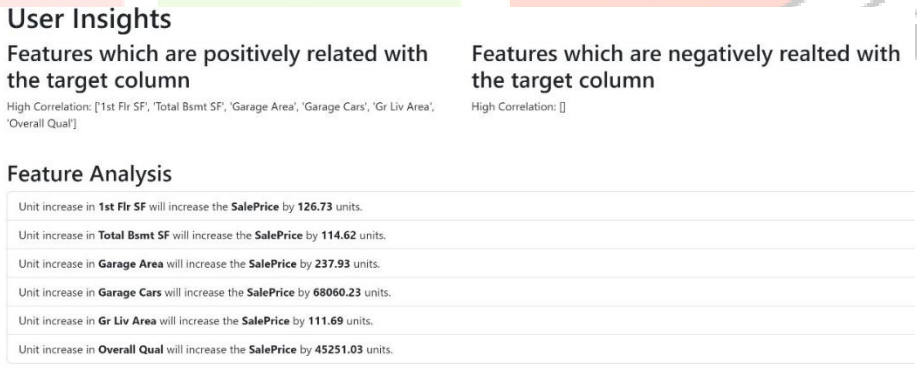

Figure 4 : User Insights

Features with high skewness (absolute skew > 1) undergo a log-transformation (with an offset to maintain positivity) to reduce skew. These steps use established Pandas and SciPy functions, ensuring the cleaned dataset has no missing values and reduced skew and outlier effects. The cleaned DataFrame is then stored for modeling.

3.4 Predictive Model Training

After cleaning, the dataset is split into features (X) and the selected target (y). We provide users with options to train multiple supervised learning algorithms, such as linear regression, k-nearest neighbors, decision tree, and random forest, using scikit-learn. Each model is fit to the training data, and performance is evaluated on a holdout test set. For regression problems, we compute mean squared error (MSE) as the evaluation metric. The system compares the models' MSE values and highlights the best-performing model (lowest error) as the "Best Model." The trained models (especially the best model) and their performance statistics are made available for the user to download or further inspect.

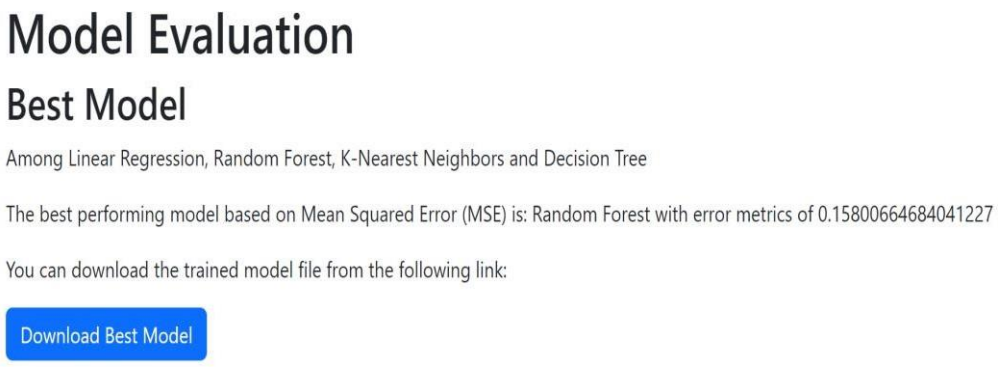

Figure 5 : Model Evaluation

3.5 Web Interface and Visualization

The frontend is implemented using Django HTML templates and styled with CSS/Bootstrap for responsiveness. Key pages include the upload form, the data quality assessment dashboard, and the model evaluation dashboard. Dynamic charts (created using the Seaborn library) visualize statistics: for example, donut charts of missing vs. valid data and outliers vs. inliers for intuitive understanding. Feature correlations with the target are computed (using df.corr()) and visualized, and "effect size" (slope of linear regression for each feature) is reported to indicate feature impact. Django views pass context variables (data quality metrics, cleaned data samples, model results) to the templates. Static resources (CSS/JS) are served via Django's static files system. The overall MVC (MTV) structure ensures separation of concerns: the View functions act as controllers coordinating data, the Template renders the pages, and the Model (in-memory or database storage) represents the data and trained models. Throughout, all Python libraries used are open-source, aligning with common data analysis stacks.

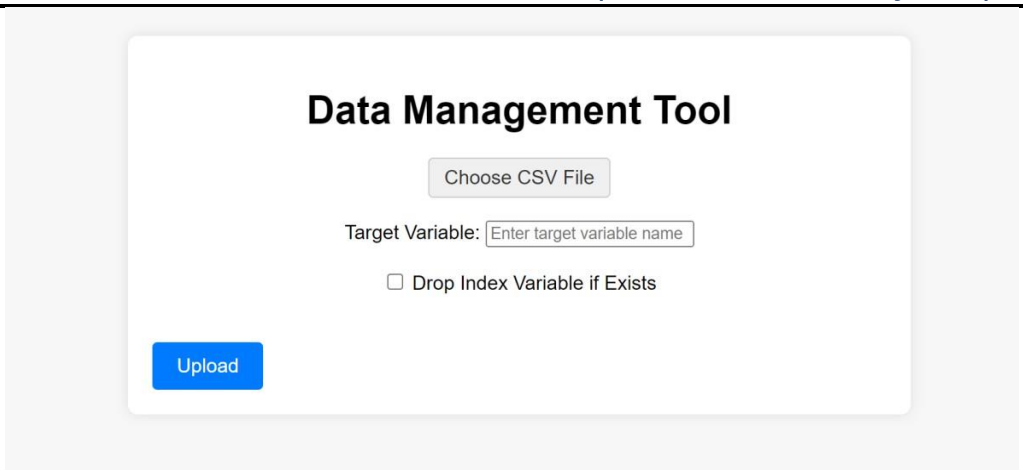

Figure 6 : UI and Upload Data

4. Results and Discussion

We evaluated the system using a case study on the publicly available Ames Housing dataset. After uploading the CSV and selecting **SalePrice** as the target variable, the application produced a data quality assessment. Table 1 summarizes the identified issues and the resulting score. The system detected approximately 5.97% of entries as missing, 1.47% as outliers, and 25.0% of features with high skewness. These combined into a composite data quality score of 67.56%. The interface displayed these metrics via charts (e.g. donut charts of missing vs. available data and outlier vs. inlier counts) for ease of interpretation.

Table 1: Data quality metrics for the case study dataset.

Metric	**Value**
Missing Values (%)	5.97
Outliers (%)	1.47
Highly Skewed Features (%)	25.00
Data Quality Score (%)	67.56

Under the **User Insights** section, the system listed features with strong correlations to the target. For example, it identified first-floor area, basement area, garage size, and overall quality as positively correlated with SalePrice. It also computed effect sizes; notably, it reported that "each unit increase in *GarageCars* increases SalePrice by 68060.23," indicating the large impact of garage size on price. Negative correlations were not strong for any features in this case.

In the **Model Evaluation** section, we compared four algorithms (Linear Regression, k-NN, Decision Tree, Random Forest). The Random Forest regressor achieved the lowest error (mean squared error ≈ 0.158), outperforming the other models (their MSE values were higher, typically above 0.20). Hence, the Random Forest was designated as the "Best Model," and a download link for this trained model was provided. The dashboard also displayed the first few rows of the cleaned dataset and descriptive summary statistics (e.g. mean, std) for user verification. Overall, these results confirm that the application effectively identifies data issues (missing, outliers, skewness) and provides actionable feedback (cleaned data, best model) to the user.

Informal user feedback during testing was positive, with users noting the clarity of the visual dashboards and the ease of managing the entire pipeline through a single interface.

Figure 7 : User Insights and Results

5. Conclusion and Future Work

We have developed a Django-based web application that streamlines data quality assurance and predictive modeling for CSV datasets. By leveraging Pandas for data manipulation and scikit-learn for modeling, the system automates crucial preprocessing steps (such as imputation, outlier handling, and encoding) and integrates them with a machine learning pipeline, all accessible via an interactive UI. The application provides real-time visual diagnostics (missing value and outlier charts) and model evaluation, facilitating data-driven insights for users without requiring deep programming expertise.

In future work, we plan to extend this platform by incorporating more sophisticated data cleaning techniques (e.g. advanced anomaly detection, cross-field validation rules) and supporting additional model types (such as deep neural networks via TensorFlow). We also aim to implement user authentication and data logging for reproducibility and auditing. To handle larger datasets, scalability improvements will be explored, such as asynchronous processing or integration with cloud storage/compute resources. These enhancements will further make the system robust and applicable to a wider range of real-world scenarios

References

[1] Qiang, Z., Dai, F., Lin, H. and Dong, Y. 2019. Research on the course system of Data Science and Engineering major. 2019 IEEE Int. Conf. on Computer Science and Educational Informatization (CSEI), Kunming, China, pp. 90–93.

[2] Alazeb, A., Alshehri, M. and Almakdi, S. 2021. Review on Data Science and Prediction. 2nd Int. Conf. on Computing and Data Science (CDS), Stanford, CA, USA, pp. 548–555.

[3] Liu, X. and Liu, C. 2020. An empirical analysis of applied statistics and probability statistics based on computer software. 2020 Int. Conf. on Big Data and Social Sciences (ICBDSS), Xi'an, China, pp. 69–71.

[4] T. R. N and Gupta, R. 2020. A survey on machine learning approaches and its techniques. 2020 IEEE Int. Students' Conf. on Electrical, Electronics and Computer Science (SCEECS), Bhopal, India, pp. 1–6.

[5] Berral-García, J. L. 2018. When and how to apply statistics, machine learning and deep learning techniques. 20th Int. Conf. on Transparent Optical Networks (ICTON), Bucharest, Romania, pp. 1–4.

[6] Dasari, D. and Varma, P. S. 2022. Employing various data cleaning techniques to achieve better data quality using Python. 6th Int. Conf. on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, pp. 1379–1383.

[7] Rahm, E. and Do, H. H. 2000. Data cleaning: problems and current approaches. IEEE Data Engineering Bulletin, 23(4): 3–13.

[8] McKinney, W. 2010. Data structures for statistical computing in Python. Proc. of the 9th Python in Science Conf. (SciPy 2010), Austin, Texas, USA, June 28–July 3, 2010, pp. 56–61.

[9] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O. et al. 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12: 2825–2830.