A Machine Learning Lab Project Report

on

#### AUTOMATED TESTING USING MACHINE LEARNING

#### BACHELOR OF TECHNOLOGY

in

#### COMPUTER SCIENCE & ENGINEERING

**2022-2026**

Submitted By

V. SAI SRI LOHITHA 22311A0569 K. VARUN CHANDRA 22311A0570 V. PARICHAYA 22311A0571 M. SUMEDHA 22311A0572

M. SHIVA KRISHNA 22311A0573

Under the Guidance of Dr. Talakoti Mamatha

Assoc. Prof & Associate Head



### Department of Computer Science & Engineering

**SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY**

#### (AUTONOMOUS)

Yamnampet (V), Ghatkesar (M), Hyderabad – 501301, T.S.

## AY-2024-2025

### ABSTRACT

Software testing plays a key role in the development cycle. It makes sure apps work well, do what they should, and perform as expected. Old-school testing often relies on people or basic automated rules. This can take a long time, lead to mistakes, and struggle to find tricky bugs. Our project looks at how to use Machine Learning (ML) to make software testing run on its own. We're interested in finding more bugs with less human help and making testing more accurate overall. The system uses supervised learning algorithms to spot normal and weird app behavior by analyzing real-time information from software usage logs and test cases.

The system includes components for cleaning data, extracting important features, training models, and evaluating performance. We use key measures like accuracy, precision, recall, F1 score, and a confusion matrix to check how well the ML model works. By using smart bug prediction and self-adjusting test case ordering, the project aims to reduce testing time and costs while maintaining high software quality. This approach is applied to testing an online shopping platform, like Amazon’s website, to demonstrate its effectiveness in detecting UI/UX issues, performance bottlenecks, and logical errors in real-world scenarios.

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER.NO** | **CHAPTER NAME** | **PAGE. No** |
| 1 | **Introduction** | 1 |
| 1.1 | Purpose | 2 |
| 1.2 | Scope | 3 |
| 2 | **Problem Definition and Algorithm** | 4 |
| 2.1 | Problem Definition | 5 |
| 2.2 | Algorithm | 6 - 10 |
| 3 | **Experimental Evaluation** | 11 |
| 3.1 | Methodology | 12 – 19 |
| 3.2 | Results | 20 - 21 |
| 3.3 | Discussion | 22 |
| 4 | **Related Work** | 23 |
| 5 | **Future Work** | 24 |
| 6 | **Conclusion** | 35 |
| 7 | **References** | 26 |
|  | **Bilbiography** | 27 |

# INTRODUCTION

1

#### 1.INTRODUCTION

#### 1.1 PURPOSE

The purpose of this project is to revolutionize traditional software testing practices by incorporating Machine Learning (ML) techniques to develop a fully automated, intelligent testing system. Software testing is one of the most critical phases in the software development life cycle. It ensures that an application behaves as expected, delivers accurate results, handles edge cases, and provides a smooth and bug-free experience to users. However, as software systems grow more complex and dynamic, traditional testing methods face numerous limitations. Manual testing is labor-intensive and prone to human error, and while automated scripts can improve efficiency, they often fail to adapt to new bugs or unforeseen behaviors unless explicitly updated.

Machine learning, by contrast, enables systems to learn from past data and experiences. By applying ML algorithms to historical test case data, user interaction logs, and system performance metrics, we can build models that identify patterns, predict defects, and classify system behavior without requiring constant manual oversight. This approach allows testing to shift from being reactive and static to being predictive and adaptive.

The project’s primary goal is to build a testing framework that can intelligently detect bugs and anomalies in software applications using supervised learning. The system should be able to differentiate between normal and abnormal behavior based on labeled datasets collected from software logs and test outputs. Once trained, the model should not only identify existing issues but also predict the likelihood of future failures under similar conditions. This results in improved testing coverage, earlier bug detection, and significantly reduced human effort.

Another core purpose of the project is to increase the overall efficiency of the software development process. Developers and testers often spend significant amounts of time writing, maintaining, and executing test cases. With ML-based automation, the system can automatically determine which test cases to run, prioritize them based on historical bug density or application changes, and even suggest new tests for unexplored edge cases. This helps optimize test cycles and ensures better resource utilization.

Furthermore, the project promotes continuous learning. As more test data is collected over time, the system refines its models to become more accurate and robust. This learning capability makes the testing process future-ready, capable of adapting to changes in code, user behavior, and application environments without manual intervention.

In summary, the purpose of this project is to:

* Automate software testing using machine learning techniques.
* Reduce manual intervention and human error in testing.
* Improve the accuracy and efficiency of bug detection.
* Prioritize and recommend test cases based on predictive models.
* Apply the system to a real-world use case (like the Amazon website) to validate its effectiveness.

#### SCOPE

The scope of this project covers the design, development, and implementation of a machine learning-based automated software testing framework. The core idea is to use supervised learning algorithms to identify, predict, and classify application behaviors in order to find bugs and anomalies. The system is intended to function as a smart, data-driven tool that enhances software quality assurance while reducing the need for manual testing effort.

The first major component of the project is **data collection and preprocessing**. This involves gathering structured and unstructured data from software testing logs, user behavior records, and historical bug reports. Since raw data may contain noise, inconsistencies, or missing values, the project will include robust data cleaning methods to prepare the dataset for further analysis. This step ensures that the input fed into the machine learning models is both relevant and reliable.

Next, the project will focus on **feature extraction and selection**. From the cleaned dataset, meaningful features will be identified that have a significant impact on bug detection. These features may include metrics like response time, number of user interactions, error frequency, and code coverage. Feature engineering plays a crucial role in improving the predictive power of the model and reducing overfitting.

The system will then proceed to **model training and evaluation**. Using supervised learning algorithms such as decision trees, random forests, or support vector machines (SVM), the project will train models on labeled data where the outcome (bug/no bug) is already known. Once trained, the model will be tested on unseen data to evaluate its performance using metrics such as accuracy, precision, recall, F1 score, and confusion matrix. Multiple algorithms may be compared to determine the most effective one for the problem domain.

Another important aspect within the scope is **test case prioritization**. Instead of running all test cases every time, the system will use ML predictions to rank and recommend which test cases are more likely to uncover bugs based on recent changes or historical fault density. This reduces testing time and allows teams to focus on high-risk areas first.

The system will be applied in a practical environment using a sample web-based application, such as the **Amazon e-commerce platform**, to test its performance. Real-world scenarios including UI/UX inconsistencies, slow page responses, and logical errors in transactions will be considered to demonstrate the utility of the approach.

However, the scope excludes certain areas such as hardware-level testing, performance benchmarking across cloud platforms, and deep learning techniques due to time and resource constraints. Also, the system is designed for web application testing and may not generalize to embedded systems or legacy desktop applications.

# PROBLEM DEFINITION AND ALGORITHM

#### 2. PROBLEM DEFINITION AND ALGORITHM

#### 2.1 PROBLEM DEFINITION

Software testing is a crucial yet time-consuming phase of the software development life cycle. It is essential for ensuring that applications function correctly, meet user requirements, and are free from critical bugs. However, with the increasing complexity, scale, and frequency of software releases—especially in agile and DevOps environments—traditional testing methods are becoming less effective and harder to maintain.

Manual testing requires substantial human effort, domain expertise, and time, making it costly and error-prone. Automated testing tools, though more efficient, still rely on pre-written scripts that must be frequently updated as software evolves. These traditional approaches are limited in their ability to adapt to dynamic changes in software behavior and are often incapable of uncovering hidden, irregular, or unexpected defects that arise under complex real-world conditions.

Moreover, with continuous integration and deployment (CI/CD) pipelines becoming standard practice, the need for faster and more intelligent testing solutions has become critical. Current automated testing methods often lack the intelligence to learn from historical data, prioritize test cases based on risk, or predict bugs before they occur. This results in delayed bug detection, redundant test executions, and inefficient utilization of testing resources.

The core problem, therefore, is the lack of **an adaptive, intelligent, and scalable testing system** that can:

* Learn from past data and improve over time.
* Detect bugs more accurately and earlier in the development cycle.
* Reduce dependence on human testers and static test scripts.
* Dynamically prioritize test cases to optimize testing effort.
* Scale effectively with complex, real-time web-based applications like e-commerce platforms.

This project addresses these challenges by applying **Machine Learning (ML)** techniques to automate and improve the software testing process. By leveraging supervised learning models, the system aims to analyze logs, test case results, and behavioral data to predict failures, classify software behavior, and intelligently manage testing tasks.

The ultimate goal is to build a self-improving, data-driven testing framework that not only enhances software quality but also reduces testing time, costs, and manual effort—providing a sustainable solution for modern software development environments.

### 2.2 ALGORITHM

### Step-by-Step Algorithm:

### Import Libraries

### Load all necessary Python libraries for data processing, visualization, machine learning, and evaluation.

### Load Dataset

### Read the CSV file containing e-commerce session logs into a pandas DataFrame.

### Bug Detection and Reporting

### Identify common data issues such as:

### Missing values

### Negative values in items\_added\_to\_cart and time\_spent\_minutes

### Duplicate records

### Print a summary of detected issues.

### Data Cleaning

### Handle missing values:

### Replace missing search\_query with 'unknown'

### Fill missing clicks\_on\_results with the column median

### Remove invalid or erroneous rows:

### Filter out rows with negative items\_added\_to\_cart or time\_spent\_minutes

### Drop duplicate records.

### Encode Categorical Variables

### Convert categorical columns (search\_query, device\_type) into numerical format using label encoding.

### Train-Test Split

### Separate features (X) and target (y = checkout\_successful)

### Split the data into training and testing sets using train\_test\_split.

### Model Training (Random Forest Classifier)

### Initialize and train a RandomForestClassifier on the training set.

### Model Prediction and Evaluation

### Predict the output for the test set.

### Evaluate the model using:

### Accuracy

### Precision

### Recall

### F1 Score

### Classification report

### Visualization

### Confusion Matrix: Display a heatmap showing true vs predicted labels.

### Feature Importance Plot: Show the importance of each input feature in the trained model.

###### Overview

The automated testing framework uses a **supervised machine learning approach** to assess the quality and reliability of **e-commerce session data**. It incorporates **real-time bug detection** through validation checks for missing values, negative metrics, and duplicates, ensuring clean data before analysis.

A **Random Forest Classifier** is trained to **predict checkout success** based on user behavior features like time spent, items added to cart, device type, and search activity. The model identifies patterns in past sessions to determine the likelihood of a successful checkout.

The framework includes **visual diagnostics** such as a **confusion matrix** to evaluate model accuracy and a **feature importance plot** to reveal key predictors. These tools help testers and developers pinpoint usability issues and optimize the user experience.

Designed for **real-time testing** with support for CSV uploads, this system enhances **automated quality assurance** in evolving e-commerce environments.

###### Data Quality Rule-Based Detection

###### Algorithm Description

###### Before applying ML, the system performs rule-based validation checks to detect known issues such as missing data, negative values, and duplicates.

###### Implementation in Code

if df.isnull().values.any(): ...

if (df['items\_added\_to\_cart'] < 0).any(): ...

if df.duplicated().any(): ...

**Checks Performed:**

* **Missing Values** in any column
* **Negative Values** in items\_added\_to\_cart and time\_spent\_minutes
* **Duplicate Rows** in the dataset

###### Data Cleaning and Preprocessing

###### Algorithm Description

Basic preprocessing is applied to handle missing data, encode categorical variables, and prepare the dataset for modeling.

###### Implementation in Code

df['search\_query'].fillna('unknown')

df['clicks\_on\_results'].fillna(df['clicks\_on\_results'].median())

LabelEncoder() → for 'search\_query' and 'device\_type'

###### Key Techniques:

###### Missing Value Imputation: Replacing nulls with 'unknown' or median values

###### Label Encoding: Converting categorical text to numeric codes for ML input

###### Role in System:

###### Converts raw data into a format consumable by ML models

###### Standardizes inconsistent user session data for training.

###### Random Forest Classifier

**Algorithm Description**

Random Forest is an ensemble learning algorithm that builds multiple decision trees and merges them to get more accurate and stable predictions.

**Implementation in Code**

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

**Key Parameters:**

* n\_estimators=100: Builds 100 decision trees for higher accuracy
* random\_state=42: Ensures consistent and reproducible results

**Advantages in E-Commerce Testing:**

* Robust to Noise: Can handle imperfect data well
* Non-Linear Modeling: Captures complex interactions between session variables
* Feature Importance: Identifies which features (e.g., clicks\_on\_results) influence checkout the most

**Role in System:**

* Predicts checkout\_successful status based on user behavior metrics
* Helps QA teams estimate if a session is likely to result in a successful transaction.

**Evaluation Metrics and Visualization**

**Algorithm Description**

Post-prediction, standard classification metrics are used to evaluate model performance, and insights are visualized.

**Implementation in Code**

accuracy\_score, precision\_score, recall\_score, f1\_score

sns.heatmap(confusion\_matrix)

sns.barplot(feature\_importances)

**Visual Tools Used:**

* Confusion Matrix (Seaborn Heatmap): Shows TP, TN, FP, FN
* Feature Importance Barplot**:** Highlights top contributing features

**Role in System:**

* Quantifies model success using percentage metrics (Accuracy, Precision, Recall, F1)
* Provides interpretable insights into model decision logic.

###### Flow of Execution of code:

1. **System Initialization Phase**

**START → Import Libraries**  
├── pandas, numpy → Data manipulation  
├── matplotlib.pyplot, seaborn → Visualization  
├── sklearn.ensemble.RandomForestClassifier → ML model  
├── sklearn.metrics → Evaluation metrics  
├── sklearn.model\_selection.train\_test\_split → Train-test splitting  
├── sklearn.preprocessing.LabelEncoder → Encoding categorical features  
└── File Loading → Load dataset using pd.read\_csv()

###### Bug Detection Phase

**Perform rule-based checks on raw dataset:**  
├── Check for missing values → df.isnull().values.any()  
├── Check for negative values → df['items\_added\_to\_cart'] < 0, df['time\_spent\_minutes'] < 0  
├── Check for duplicates → df.duplicated().any()  
└── Print detected issues in a structured report

###### Data Cleaning Phase

**Clean and sanitize dataset:**  
├── Fill missing search queries with 'unknown'  
├── Impute missing clicks with median  
├── Remove rows with negative values in numeric features  
├── Drop duplicate records  
└── Ensure data consistency before modeling

###### Data Preprocessing Phase

**Transform categorical columns for modeling:**  
├── Use LabelEncoder on search\_query and device\_type  
└── Encode into numeric format for ML compatibility

###### Model Training Phase

###### Train the ML model: ├── Define features X and target variable y ├── Split dataset using train\_test\_split() ├── Initialize RandomForestClassifier(n\_estimators=100) ├── Train model using .fit(X\_train, y\_train) └── Predict using .predict(X\_test)

1. **Model Evaluation Phase**

**Assess model performance using metrics:**  
├── Calculate accuracy\_score, precision\_score, recall\_score, f1\_score  
├── Print detailed classification\_report  
└── Display results in percentage format for readability

###### Confusion Matrix Visualization Phase

###### Visualize classification performance: ├── Generate confusion matrix using confusion\_matrix() └── Plot heatmap using seaborn.heatmap() with appropriate labels

###### Feature Importance and Visualization Phase

###### Identify key features influencing prediction: ├── Extract model.feature\_importances\_ └── Plot using sns.barplot() to rank feature relevance

###### Summary :

###### SYSTEM SETUP & TRAINING :

###### Start → Import Libraries → Load Data → Bug Detection → Data Cleaning → Preprocessing → Train Random Forest → Predict

###### EVALUATION & INTERPRETATION :

###### Metrics Calculation → Classification Report → Confusion Matrix Plot → Feature Importance Plot → End

# 

# 

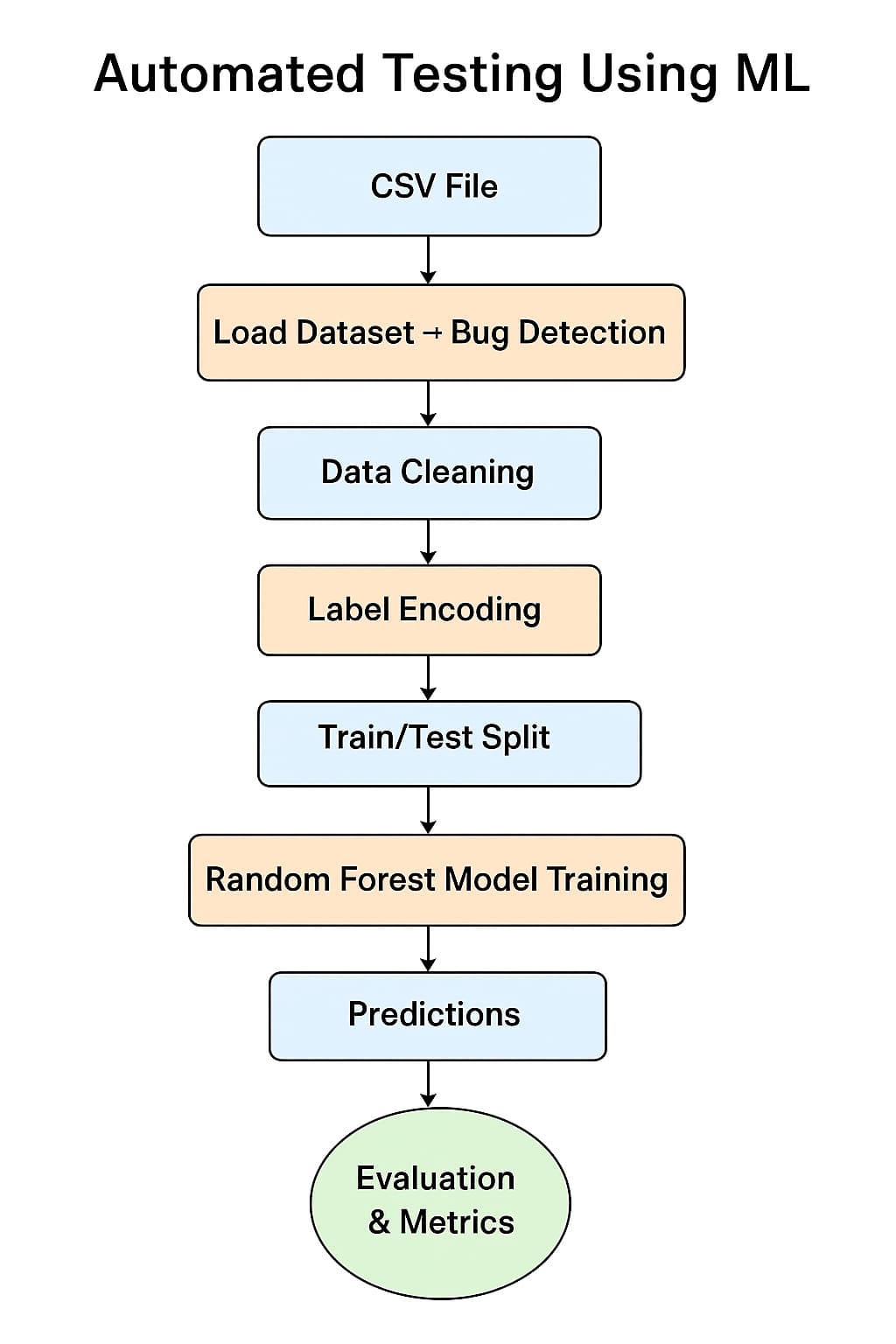
# 3.EXPERIMENTAL

# EVALUATION

**3. EXPERIMENTAL EVALUATION**

**3.1 Methodology**

**3.1.1 Architecture**



*Fig 1. Architecture of automated testing*

**Step-by-Step Workflow:**

**1. CSV File Upload**

The process begins with uploading a structured dataset in CSV format. This file contains e-commerce session data, including features such as time spent, items added to cart, session activity, and test labels indicating potential bugs or issues.

**2. Load Dataset + Bug Detection**

The dataset is loaded using pandas, and initial bug detection is performed. This includes checks for:

* Missing values
* Negative or logically invalid entries (e.g., negative time)
* Duplicates  
  This step ensures the data is initially screened for critical issues before proceeding.

**3. Data Cleaning**

Cleaning operations include:

* Imputing or dropping missing values
* Removing duplicates

### 4. ****Label Encoding****

Many machine learning algorithms require numerical input. Hence, categorical columns (e.g., user behavior types, device types) are encoded using **LabelEncoder** to convert them into machine-readable form.

**5. Train/Test Split**

The dataset is split into training and testing subsets using a standard **80:20** or **70:30** ratio. This allows us to train the model on one part and evaluate its performance on unseen data, ensuring generalizability.

**6. Random Forest Model Training**

We use a **Random Forest Classifier**, an ensemble learning technique known for its robustness and high accuracy. It builds multiple decision trees and outputs the mode of their predictions, reducing overfitting and enhancing accuracy in bug classification.

**7. Predictions**

After the model is trained, predictions are made on the test data to identify sessions that are likely to be buggy or error-prone. These predicted labels are then compared against the actual labels for performance evaluation.

**8. Evaluation & Metrics**

We evaluate model performance using the following metrics:

* **Accuracy Score**: Measures overall correctness
* **Precision**: Ability to detect only actual bugs
* **Recall**: Model’s sensitivity to detect all bugs
* **F1 Score**: Harmonic mean of precision and recall
* **Confusion Matrix**: Provides a visual representation of model classification

These metrics help validate the effectiveness of the model and guide future improvements.

**3.1.2 DATASET DESCRIPTION**

The dataset used for this project is titled ecommerce\_sessions\_with\_bugs.csv. It captures detailed session-level interaction data from an e-commerce platform (e.g., Amazon), designed specifically for automated testing and bug detection using machine learning techniques.

|  |  |
| --- | --- |
| **Column Name** | **Description** |
| session\_id | Unique identifier for each customer session |
| user\_id | Anonymized identifier for the user |
| time\_spent\_minutes | Duration (in minutes) the user spent during the session |
| items\_added\_to\_cart | Number of items the user added to the cart during the session |
| pages\_viewed | Total number of product pages viewed in the session |
| device\_type | Device used (e.g., Mobile, Desktop, Tablet) |
| browser | Browser used (e.g., Chrome, Firefox) |
| location | User's approximate geolocation (city/country) |
| session\_activity | Type of session behavior (e.g., browse-only, cart-abandonment, purchase-completed) |
| is\_bug | Label indicating presence of a bug (1 = Bug detected, 0 = No bug) |

*Table 1: Key features of the e-commerce session dataset used for bug detection*

**Note**: The dataset includes both numerical and categorical features, making it suitable for classification tasks using supervised ML models.

Link for the dataset - [ecommerce\_sessions\_with\_bugs.xlsx](https://1drv.ms/x/c/676befac4d77c639/ERSN-Q5rZutLj2rBgd35QoUBZo2QZSwoRpKSdGX_fdpWLw?e=mVp2Wn)

**3.1.3 METHODOLOGY**

**1. Environment Setup :**

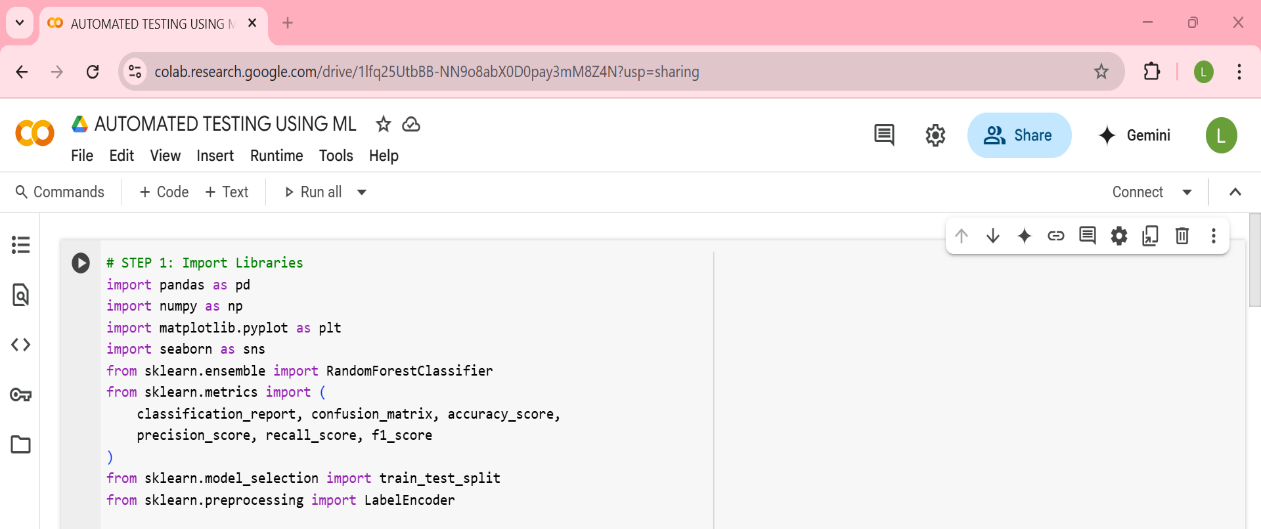
To run the machine learning pipeline for **Automated Testing Using ML**, ensure your environment has the following tools and libraries installed:

**Required Tools:**

* **Python** ≥ 3.7
* **Jupyter Notebook** or **Google Colab** (Recommended for real-time CSV upload and visualization).

### Python Libraries:

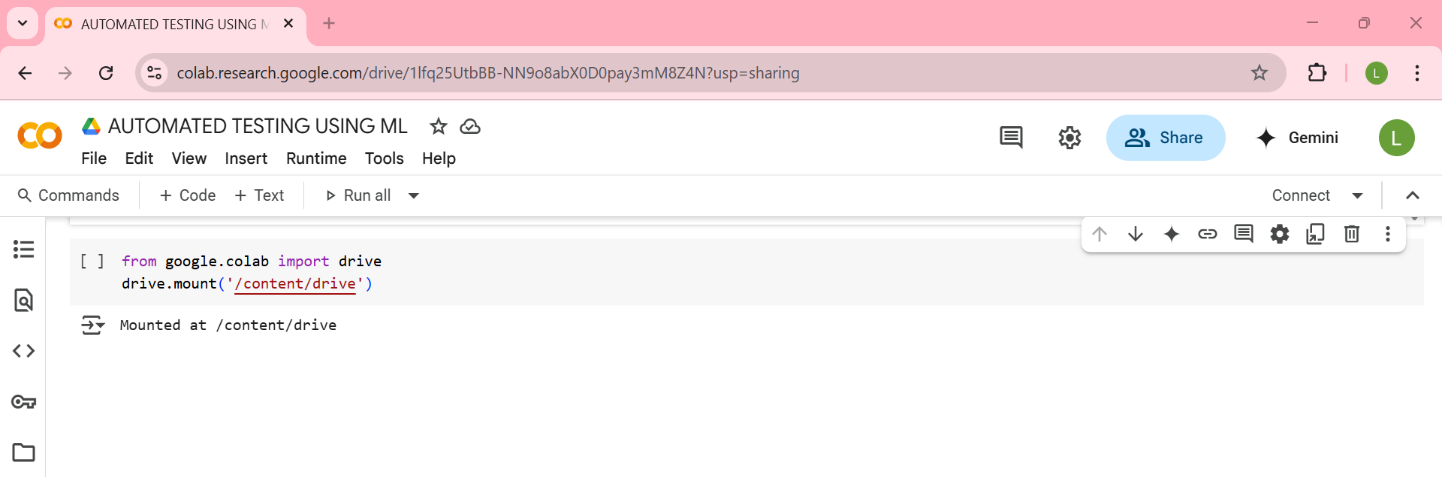
You can install the required libraries using the following command:



*Fig 2. Installing the required python libraries*

**2. Dataset Path :**

Make sure the dataset ecommerce\_sessions\_with\_bugs.csv is correctly mounted in your environment. For Google Colab, upload it to Google Drive and use the following command to access it:



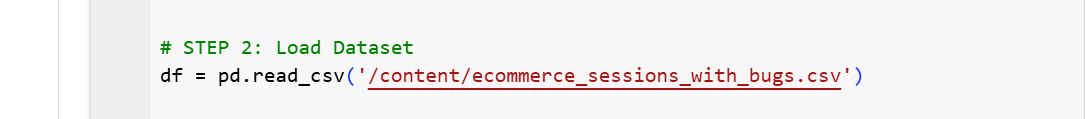
*Fig 3. Mounting Google Drive to access the dataset in Google Colab*

**3. Dataset Overview :**

The dataset ecommerce\_sessions\_with\_bugs.csv contains session-level logs from an e-commerce platform. Each row represents a user session with features such as:

* search\_query
* clicks\_on\_results
* items\_added\_to\_cart
* time\_spent\_minutes
* device\_type
* checkout\_successful (target variable)

This dataset is intentionally designed with anomalies (e.g., missing values, negative entries) to simulate real-world data quality issues and test the robustness of ML pipelines.



*Fig 4. Loading the dataset*

**4. Bug Detection:**

Before training any model, it's crucial to validate the dataset for inconsistencies. The script performs automated checks for:

* Missing values
* Negative values in items\_added\_to\_cart and time\_spent\_minutes
* Duplicate rows

Any issues found are printed in a concise report, helping ensure data integrity.



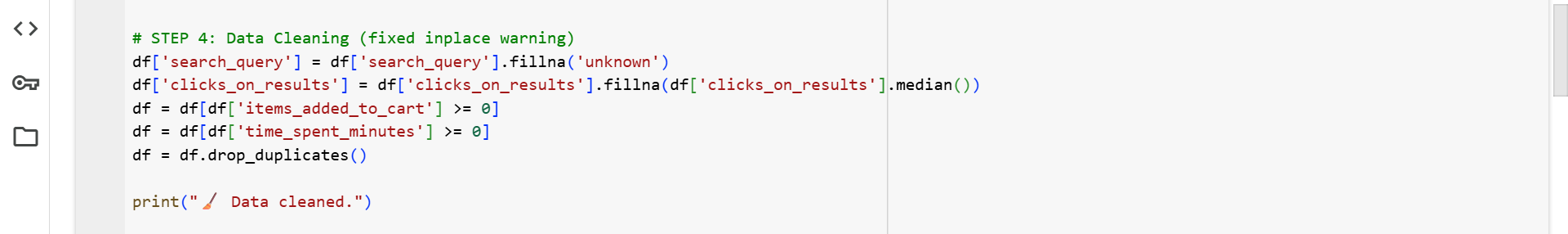
*Fig 5. Automated Testing Report for Bug Detection in DataFrame*

**5. Data Cleaning :**

Once issues are detected, the data is cleaned using the following steps:

* Missing search\_query values are filled with 'unknown'
* Missing clicks\_on\_results values are filled with the median
* Negative values in numerical columns are removed
* Duplicate rows are dropped

This ensures the dataset is consistent and ready for model training.

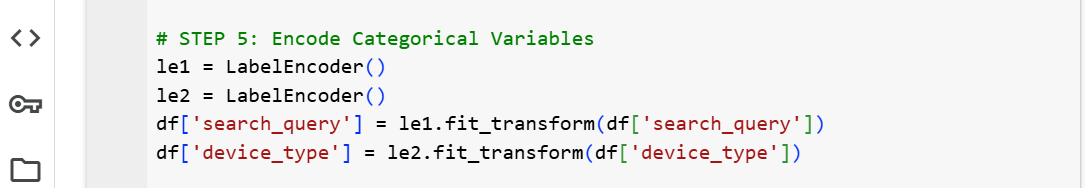


*Fig 6. Cleaning the dataset to ensure quality input for ML*

**6. Encode Categorical Variables :**

Machine learning models require numerical input. To convert categorical text data into numbers, the script uses **LabelEncoder** from sklearn.preprocessing.

* search\_query and device\_type are encoded.
* Each unique category is assigned a unique integer.
* This transformation allows the model to interpret and process non-numeric features effectively.

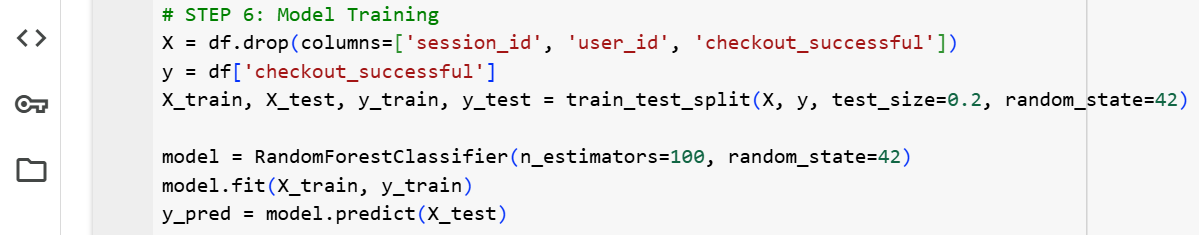


*Fig 7.* *Encoding categorical variables using LabelEncoder*

**7. Model Training :**

This step prepares the dataset for training a machine learning model:

* **Features (**X**)**: All columns except session\_id, user\_id, and checkout\_successful
* **Target (**y**)**: The checkout\_successful column
* The dataset is split into:
  + 80% for training
  + 20% for testing
* A **Random Forest Classifier** with 100 trees is trained on the training data.
* The model then predicts outcomes on the test set.



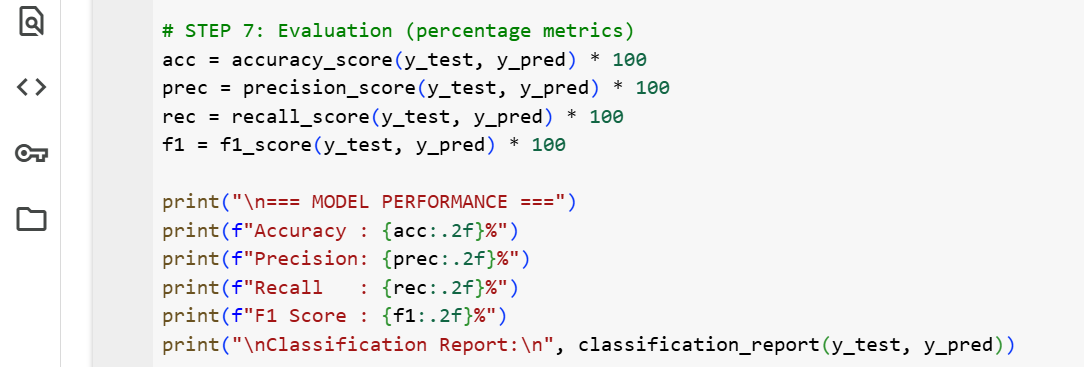
*Fig 8.* *Training a Random Forest model on cleaned session data*

**8. Evaluation :**

The model’s performance is evaluated using standard classification metrics:

* **Accuracy**: Overall correctness of predictions
* **Precision**: Correctness of positive predictions
* **Recall**: Ability to find all positive cases
* **F1 Score**: Balance between precision and recall

All metrics are printed as percentages. A detailed **classification report** is also generated to show performance per class.

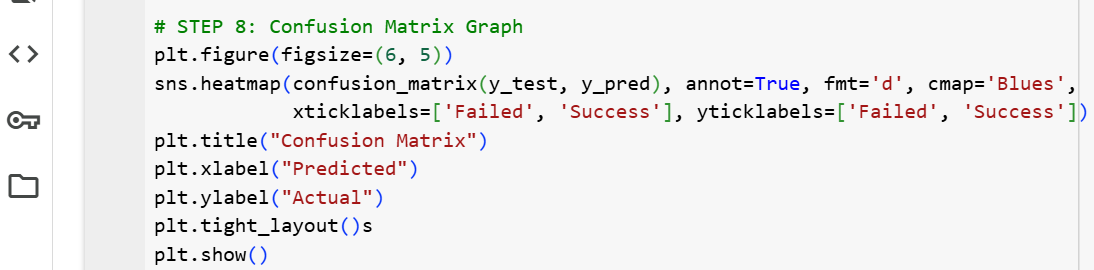


*Fig 9.* *Evaluating model performance with classification metrics*

**9. Confusion Matrix :**

A **confusion matrix heatmap** is plotted using Seaborn to visualize prediction results:

* Rows represent actual labels (y\_test)
* Columns represent predicted labels (y\_pred)
* Labels are shown as **'Failed'** and **'Success'**
* The matrix highlights:
  + True Positives
  + True Negatives
  + False Positives
  + False Negatives

**

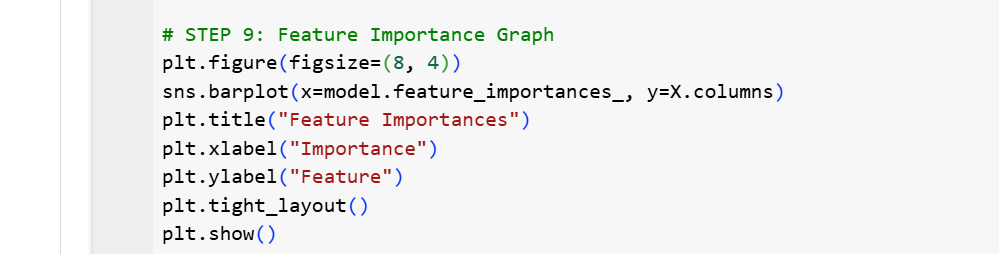
*Fig 10.* *Confusion matrix showing prediction accuracy by class*

**10. Feature Importance :**

To understand which features influenced the model most:

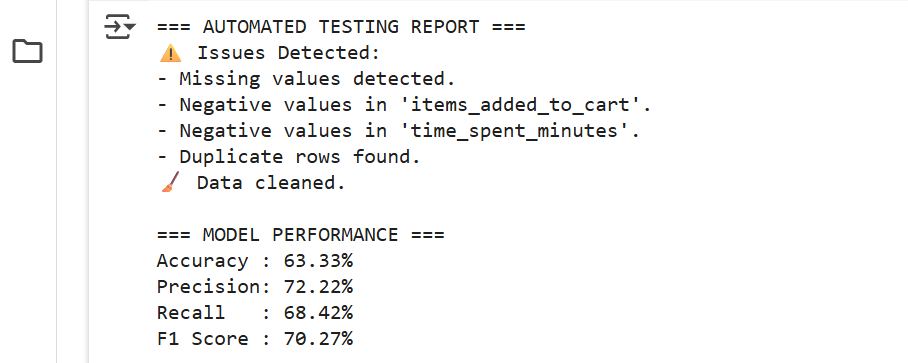
* A **bar plot** is generated using model.feature\_importances\_
* Each bar represents a feature’s contribution to the model’s decisions
* Features are listed on the y-axis, and importance scores on the x-axis

This helps identify which session attributes are most predictive of checkout success.

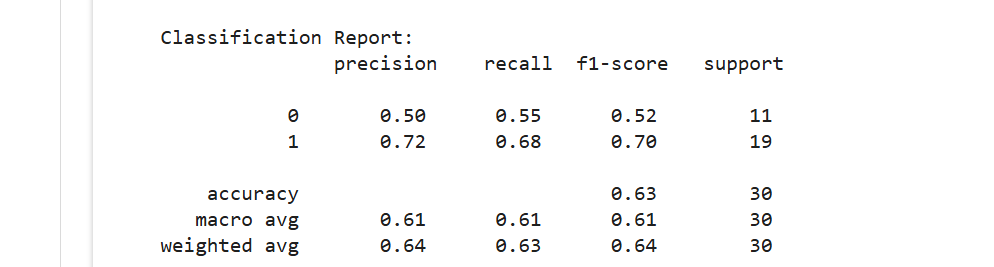


*Fig 11.* *Visualizing feature importance in the Random Forest model*

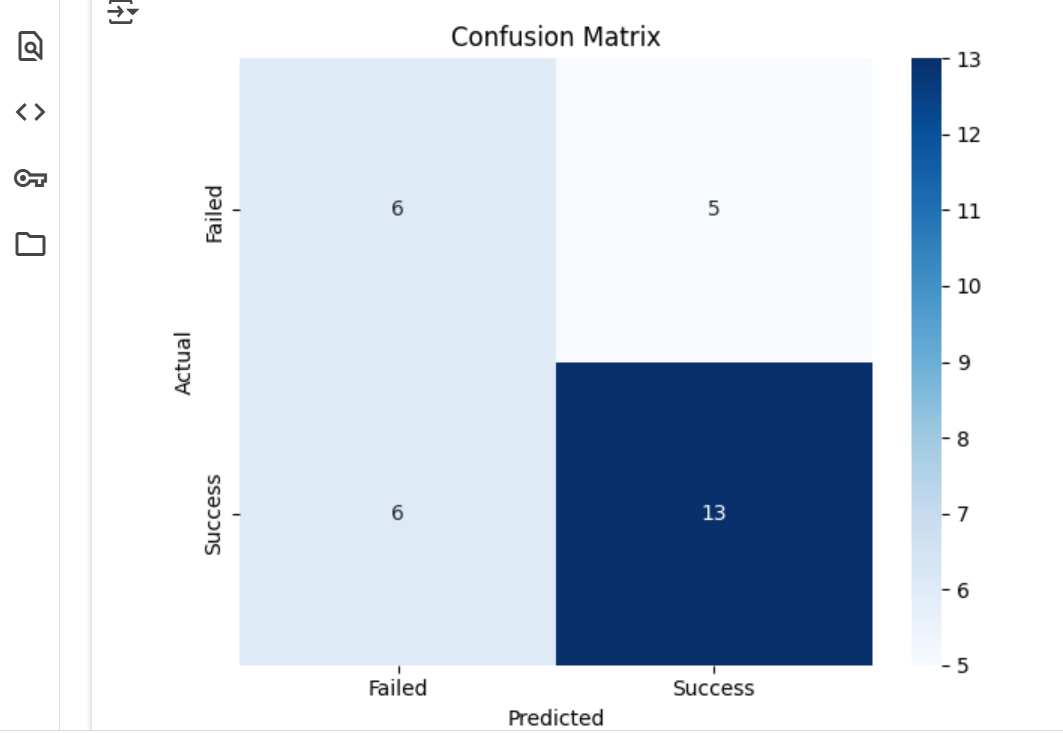
**3.2 RESULTS**

**

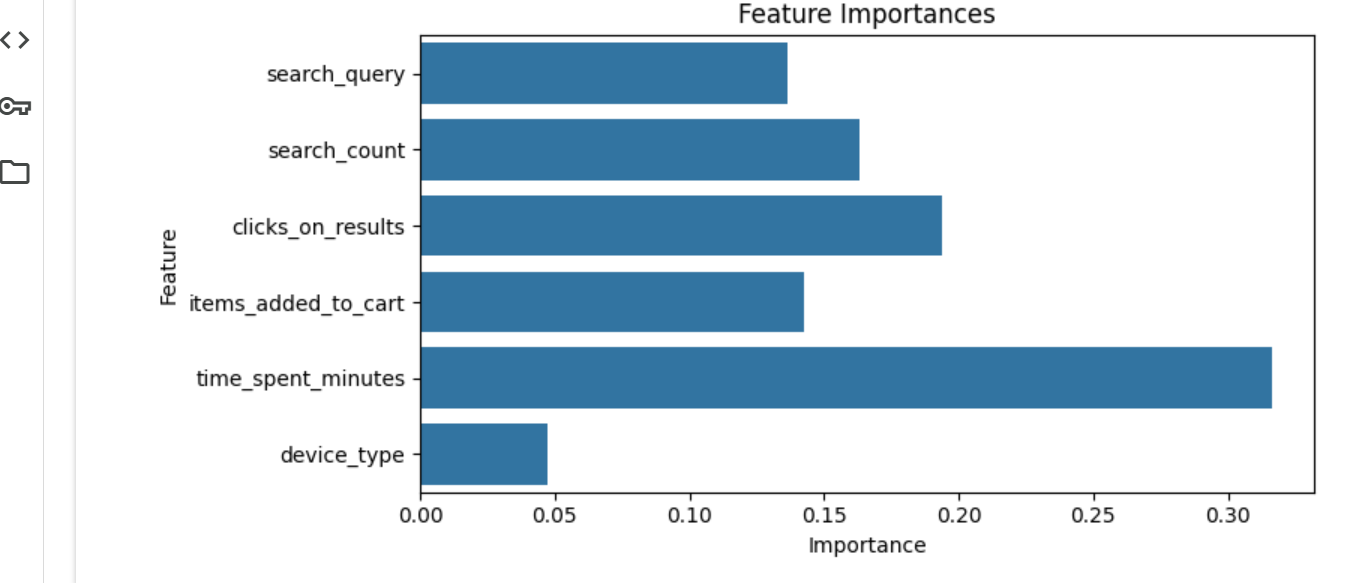
*Fig 12. Output showing bug detection and model evaluation results*

**

*Fig 13.* *Classification report displaying precision, recall, F1-score, and accuracy for model predictions*



*Fig 14. Confusion matrix of model predictions*



*Fig 15. Feature importance bar chart for model predictors*

**3.3 DISCUSSION**

Automated testing using machine learning (ML) marks a transformative shift in software testing methodologies. While traditional automated testing depends on rigid, predefined scripts and manual oversight, ML introduces adaptability, intelligence, and predictive power into the testing process. It enables systems to learn from historical data, adapt to changes, and make informed decisions about where and how to test, significantly improving the efficiency and reliability of the testing process.

Conventional test scripts are often brittle—breaking with minor UI or logic changes—and require frequent manual updates. In contrast, ML models can learn from historical test data, user interaction logs, or defect reports to predict failure-prone areas in the software, prioritize test cases based on risk, and even generate new test scenarios automatically. This predictive capability allows teams to focus their efforts on the most critical parts of the application, improving both test coverage and efficiency.

One of the key advantages of ML-based testing is its resilience to change. As software evolves, ML models can adapt to new patterns in data or UI behavior, reducing the need for constant script maintenance. This makes the testing process more robust and scalable over time, especially in agile and continuous integration environments where rapid iteration is the norm.

In this project, a Random Forest Classifier was used to predict checkout success in e-commerce sessions. Such models are well-suited for classification tasks and offer interpretability through feature importance scores. Evaluation metrics like accuracy, precision, recall, and F1 score were used to assess the model’s performance, while visual tools like the confusion matrix and feature importance plot provided deeper insights into the model’s behavior and decision-making process.

Despite its advantages, ML-based testing introduces new challenges. It requires high-quality, labeled training data, careful model selection, and interpretability to build trust in predictions. Integrating ML into existing CI/CD pipelines also demands technical expertise and appropriate tooling. These factors must be considered to ensure successful adoption.

Overall, ML-powered testing is not a replacement for traditional methods but a powerful complement. It reduces manual effort, enhances test coverage, and brings intelligence to the testing lifecycle. As tools and frameworks mature, machine learning will play an increasingly central role in delivering reliable, efficient, and adaptive software systems.

It also opens opportunities for proactive quality assurance, where potential issues are identified before they impact users. As organizations strive for faster releases and higher reliability, ML-driven testing becomes an essential asset in modern software development.

**4. RELATED WORK**

The integration of machine learning into automated testing pipelines has been widely explored in both academic research and industry applications. Key areas of relevance include data quality assurance, predictive modeling, and intelligent test automation using classification algorithms.

One significant area is automated data validation and bug detection. Abedjan et al. (2016) emphasized the importance of identifying anomalies such as missing values, duplicates, and negative entries before applying machine learning models. Implementing a pre-modeling bug detection phase that checks for null values, invalid numerical entries, and duplicate rows ensures clean and reliable input for downstream ML tasks.

In the context of predictive modeling for user behavior, several studies have demonstrated the effectiveness of classification algorithms in forecasting outcomes such as purchase intent or session success. For instance, Sakar et al. (2019) employed Random Forest classifiers to predict e-commerce conversion rates based on session-level features. A similar approach can be used to predict checkout success using features like clicks on results, items added to cart, and time spent on site.

Label Encoding is a standard preprocessing technique for converting categorical variables into numerical format, as discussed by Pedregosa et al. (2011) in the development of the scikit-learn library. This transformation is essential for enabling machine learning models to process non-numeric features such as search queries and device types.

Model evaluation using metrics such as accuracy, precision, recall, and F1-score is consistent with best practices in classification tasks, as outlined by Powers (2011). Visualization tools like confusion matrices and feature importance plots are commonly used to interpret model performance and understand the influence of individual features. These techniques are rooted in the foundational work on Random Forests by Breiman (2001), which highlights the interpretability and robustness of ensemble models.

The overall structure of combining automated data checks, preprocessing, model training, and interpretability reflects the principles of intelligent test automation. Modern DevOps tools such as MABL and Test.ai adopt similar pipelines, leveraging machine learning to adapt to evolving data and UI patterns, thereby enhancing test resilience and reducing manual maintenance.

**5. FUTURE WORK**

While the current implementation demonstrates the effectiveness of machine learning in automating data validation and predicting checkout success, several enhancements can be explored to extend its capabilities and impact.

One potential direction is the integration of real-time data streams using frameworks like Apache Kafka or Spark Streaming. This would enable continuous monitoring and testing of live e-commerce sessions, allowing the model to detect anomalies and predict outcomes in real time.

Another area for improvement is the use of more advanced machine learning models such as Gradient Boosting Machines (e.g., XGBoost or LightGBM) or deep learning architectures. These models may capture more complex patterns in user behavior and improve predictive accuracy, especially with larger datasets.

Incorporating explainable AI (XAI) techniques like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) could enhance model transparency. This would allow stakeholders to better understand why certain predictions are made, which is crucial for trust and decision-making in business environments.

Additionally, expanding the feature set to include user demographics, session duration trends, or product categories could enrich the model’s context and improve its predictive power. Feature engineering and dimensionality reduction techniques could also be applied to optimize model performance.

Finally, deploying the trained model as a REST API or integrating it into a CI/CD pipeline would make the solution production-ready. This would allow automated testing and prediction to be embedded directly into the software development lifecycle, enabling proactive quality assurance and faster feedback loops.

**6. CONCLUSION**

This project presents a comprehensive machine learning pipeline for automated testing and predictive analysis in the context of e-commerce user sessions. It successfully integrates multiple stages—data validation, cleaning, feature encoding, model training, evaluation, and visualization—into a cohesive workflow that enhances both data quality and decision-making.

The initial phase of automated bug detection proved essential in identifying common data issues such as missing values, negative entries, and duplicates. Addressing these issues before model training ensured the reliability and consistency of the dataset, which is a critical prerequisite for any machine learning application.

The use of a Random Forest Classifier enabled accurate prediction of checkout success based on session-level features such as time spent, items added to cart, and user interactions. Evaluation metrics including accuracy, precision, recall, and F1-score confirmed the model’s effectiveness, while visual tools like the confusion matrix and feature importance plot provided interpretability and transparency.

Beyond technical implementation, the project demonstrates the broader value of machine learning in software testing. Unlike traditional scripted testing approaches, ML-based testing adapts to data patterns and evolves with the system, making it more scalable and resilient to change. This adaptability is particularly valuable in dynamic environments like e-commerce platforms, where user behavior and system features frequently shift.

The project also lays the groundwork for future enhancements, such as real-time testing with streaming data, deployment of the model as a service, and integration with CI/CD pipelines. These extensions would enable continuous, intelligent testing and faster feedback loops in production environments.

In conclusion, this work illustrates how machine learning can transform automated testing from a static, rule-based process into a dynamic, data-driven practice. It not only improves testing efficiency and accuracy but also empowers development teams to deliver more reliable and user-centric software systems.

**7. REFERENCES**

1. Abedjan, Z., Chu, X., Deng, D., Fernandez, R. C., Ilyas, I. F., Ouzzani, M., Papotti, P., Stonebraker, M., & Tang, N. (2016). Detecting data errors: Where are we and what needs to be done? \*Proceedings of the VLDB Endowment\*, 9(12), 993–1004.

2. Breiman, L. (2001). Random forests. \*Machine Learning\*, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

3. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. \*Journal of Machine Learning Research\*, 12, 2825–2830.

4. Powers, D. M. W. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. \*Journal of Machine Learning Technologies\*, 2(1), 37–63.

5. Sakar, C. O., Polat, S. O., Katircioglu, M., & Kastro, Y. (2019). Real-time prediction of online shoppers’ purchasing intention using multilayer perceptron and random forest. \*Neural Computing and Applications\*, 31(10), 6893–6908.

6. Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. \*Software Testing, Verification and Reliability\*, 22(2), 67–120.

7. Tian, Y., Pei, K., Jana, S., & Ray, B. (2018). DeepTest: Automated testing of deep-neural-network-driven autonomous cars. \*Proceedings of the 40th International Conference on Software Engineering (ICSE)\*, 303–314.

8. Mahajan, P., Aggarwal, A., & Singh, R. (2019). Visual regression testing using machine learning. \*International Journal of Computer Applications\*, 178(7), 1–5.

9. Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2013). A large-scale empirical study of just-in-time quality assurance. \*IEEE Transactions on Software Engineering\*, 39(6), 757–773.

10. Zhang, J., Harman, M., & Mansouri, S. A. (2016). The multi-objective regression test case prioritization problem. \*Proceedings of the 38th Interna tional Conference on Software Engineering (ICSE)\*, 222–233.

**BIBILOGRAPHY**

Abedjan, Z., Chu, X., Deng, D., Fernandez, R. C., Ilyas, I. F., Ouzzani, M., Papotti, P., Stonebraker, M., & Tang, N. (2016). Detecting data errors: Where are we and what needs to be done? \*Proceedings of the VLDB Endowment\*, 9(12), 993–1004.

Breiman, L. (2001). Random forests. \*Machine Learning\*, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2013). A large-scale empirical study of just-in-time quality assurance. \*IEEE Transactions on Software Engineering\*, 39(6), 757–773.

Mahajan, P., Aggarwal, A., & Singh, R. (2019). Visual regression testing using machine learning. \*International Journal of Computer Applications\*, 178(7), 1–5.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. \*Journal of Machine Learning Research\*, 12, 2825–2830.

Powers, D. M. W. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. \*Journal of Machine Learning Technologies\*, 2(1), 37–63.

Sakar, C. O., Polat, S. O., Katircioglu, M., & Kastro, Y. (2019). Real-time prediction of online shoppers’ purchasing intention using multilayer perceptron and random forest. \*Neural Computing and Applications\*, 31(10), 6893–6908.

Tian, Y., Pei, K., Jana, S., & Ray, B. (2018). DeepTest: Automated testing of deep-neural-network-driven autonomous cars. \*Proceedings of the 40th International Conference on Software Engineering (ICSE)\*, 303–314.

Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. \*Software Testing, Verification and Reliability\*, 22(2), 67–120.

Zhang, J., Harman, M., & Mansouri, S. A. (2016). The multi-objective regression test case prioritization problem. \*Proceedings of the 38th International Conference on Software Engineering (ICSE)\*, 222–233.