International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 07 | July – 2025 DOI: 10.55041/ISJEM04878
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

IMDB Sentiment Analysis Based on Comment by Using Machine Learning

MAMIDI TARANI, KURMAPU DURGAPRASAD.
Assistant Professor, MCA Final Semester, Master of Computer Applications, Sanketika Vidya Parishad Engineering College, Vishakhapatnam, Andhra Pradesh, India.

Abstract:

Sentiment analysis plays a significant role in understanding user opinions, product reviews, and market feedback by classifying textual data into sentiment categories. This project presents a lightweight, web-based sentiment analysis application using the Flask framework integrated with a Multinomial Naive Bayes classifier. The system is designed to classify movie reviews as either positive or negative, facilitating quick and accessible feedback analysis for users. Text data is pre-processed and transformed into numerical vectors using CountVectorizer, allowing the Naive Bayes model to perform effective feature-based classification. The classifier is trained on a small, clean dataset of movie reviews and demonstrates how simple yet powerful models can be applied in real-world sentiment classification tasks. To enable real-time sentiment predictions, the trained model and vectorizer are serialized using pickle and loaded within the Flask application. Users can submit their movie reviews through a simple HTML interface, and the system predicts and displays the sentiment classification immediately. Additionally, the application integrates Swagger documentation using Flasgger, providing clear API endpoint testing and making it easier for developers to extend or test the system. The modular structure of the application, including data processing, model training, and web deployment, makes it an effective learning tool for beginners in machine learning deployment workflows. This project demonstrates the end-to-end pipeline of building, training, and deploying a machine learning model using Python's Flask micro-framework, emphasizing the practical application of sentiment analysis in web services. It highlights how sentiment analysis can be used in review systems, customer feedback monitoring, and content moderation. The system can be scaled further with larger datasets and additional NLP preprocessing for enhanced accuracy in practical applications. Overall, this project provides a hands-on implementation of a sentiment analysis system that is accessible, interpretable, and ready for integration into real-world environments.

Key Words: Sentiment Analysis, Naive Bayes, Flask, Natural Language Processing, Machine Learning, CountVectorizer.

I. INTRODUCTION

Sentiment analysis is a subfield of Natural Language Processing (NLP) that involves the automated identification and classification of sentiments or opinions expressed in textual data. It is widely used in various domains, including product review analysis, customer feedback monitoring, and social media analytics, to extract actionable insights from user-generated content [22]. This project implements a sentiment analysis system that classifies movie reviews as positive or negative using a Multinomial Naive Bayes classifier. The system leverages the Flask micro-framework to provide a web-based interface, enabling users to input movie reviews and receive instant sentiment classification results [9]. CountVectorizer is utilized to transform raw textual reviews into numerical feature vectors, allowing the Naive Bayes classifier to effectively learn and predict sentiments [5]. The application is designed with simplicity and modularity, making it accessible for learners and scalable for further development. The trained model and vectorizer are serialized using pickle and are integrated within the Flask application for real-time predictions without the need for retraining. Additionally, the system incorporates Swagger documentation using Flasgger, facilitating API testing and extending the project for future enhancements. By demonstrating the complete workflow from preprocessing text data, training a machine learning model, and integrating it into a web service for deployment this project provides a practical understanding of deploying NLP models in real-world applications [15]. It highlights how lightweight yet effective machine learning models can be operationalized in live systems for sentiment analysis tasks, making it a valuable learning tool for students and practitioners interested in applied machine learning and NLP system deployment.

1.1 Existing System

Current sentiment analysis systems typically use machine learning or deep learning models requiring high computational power and complex configurations. Most of these systems operate on large datasets in offline batch modes, lacking real-time prediction for individual user inputs [13]. They often require manual preprocessing and command-line execution, making them less accessible to non-technical users. Existing systems also do not provide simple web interfaces for easy input and instant sentiment feedback [20]. Many models are challenging to deploy for practical use due to heavyweight frameworks and dependencies. As a result, these systems are not well-suited for lightweight, user-friendly sentiment analysis applications for small projects or educational purposes [6].

1.2.1 Challenges

- Data Preprocessing: Handling noisy [12], unstructured text data before feeding it into the model can be complex.
- Limited Dataset: Small datasets can lead to poor generalization and reduced model accuracy.

- Real-Time Prediction: Converting offline models to real-time prediction systems is challenging for beginners [4].
- User Accessibility: Existing systems often lack simple interfaces for non-technical users to input reviews.
- Deployment Complexity: Deploying machine learning models on web frameworks requires managing dependencies and serialization effectively.
- Resource Constraints: Heavy models demand computational resources [18], making lightweight deployment difficult.
- Evaluation: Testing model performance on varied, real-world reviews requires continuous validation and improvement.

1.2 Proposed System

The proposed system implements a lightweight, web-based sentiment analysis application using Flask for real-time movie review classification [7]. It uses CountVectorizer to convert text into numerical vectors and a Multinomial Naive Bayes classifier for sentiment prediction. Users can enter reviews through a simple HTML interface and receive instant positive or negative sentiment feedback. The model and vectorizer are serialized for efficient integration into the Flask app, ensuring smooth prediction without retraining. Swagger documentation is included for easy API testing and extension [13]. This system aims to make sentiment analysis accessible, practical, and easy to deploy for students and small projects with minimal resources.

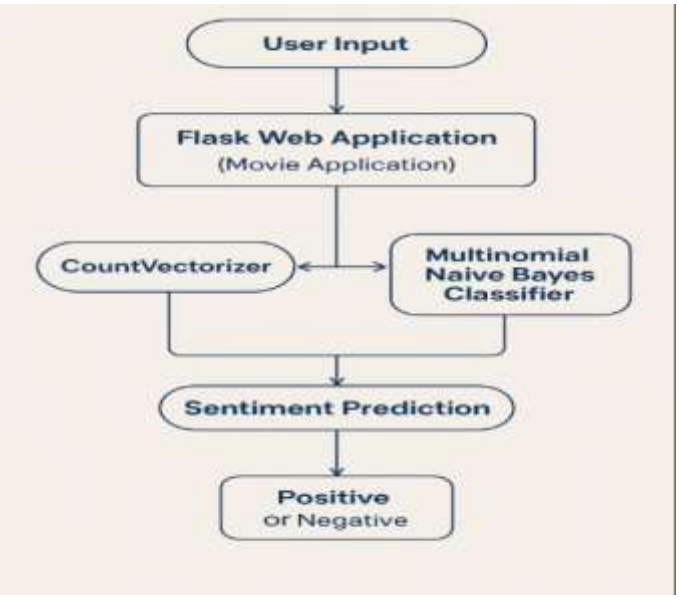

Fig:1 Proposed system.

1.2.1 Advantages

- **Lightweight Deployment:** Uses Flask for easy and fast deployment on local or cloud servers.
- **Real-Time Prediction:** Provides instant sentiment analysis for user-submitted reviews.
- **User-Friendly Interface:** Allows non-technical users to analyze sentiments without complex commands.
- **Low Resource Requirements:** The Multinomial Naive Bayes model is efficient and requires minimal computational power.
- **Easy Integration:** Can be integrated with other systems or extended for API-based applications.
- **Educational Utility:** Helps students learn the end-to-end deployment of ML models practically.
- **Modular Design:** Allows future enhancements, such as dataset expansion and advanced preprocessing, without redesigning the system.

II. LITERATURE REVIEW

2.1 Architecture:

The system architecture consists of a client-server model using the Flask micro-framework for deploying the sentiment analysis application. Users input movie reviews through a web interface served by Flask, which handles HTTP requests and responses efficiently. The input review is passed to the backend, where CountVectorizer transforms the text into numerical feature vectors

suitable for machine learning processing [8]. The transformed input is then fed into the Multinomial Naive Bayes classifier, which has been pre-trained on a labeled dataset of movie reviews to classify the sentiment as positive or negative. The trained model and vectorizer are loaded using pickle for seamless real-time prediction without retraining during each request. The prediction result is then sent back to the Flask server [14], which renders the result page displaying the sentiment to the user Additionally, Swagger (Flasgger) is integrated to document and test the API endpoints, ensuring ease of extension for future API-based integrations [23]. This architecture enables real-time, low-resource sentiment analysis with a modular and scalable structure, making it practical for student projects and lightweight web-based applications.

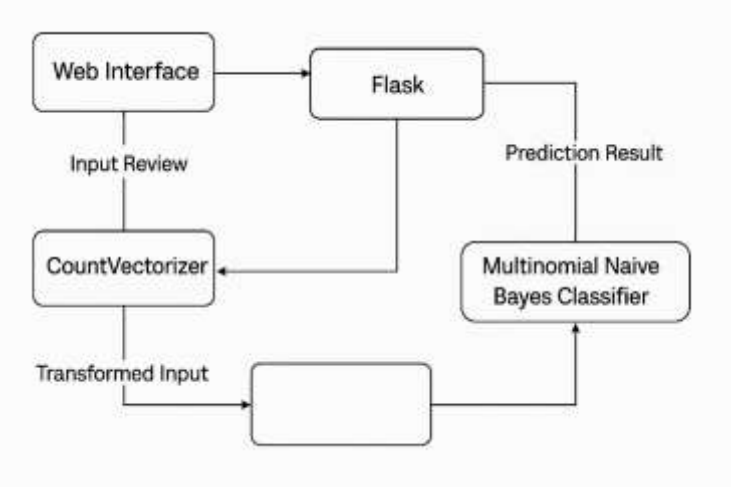

Fig:2 Architecture Diagram.

2.2 Algorithm:

- **CountVectorizer Algorithm:**
  - Converts input text data into a matrix of token counts.
  - Splits the text into tokens (words) and counts their occurrences [25].
  - Produces a sparse matrix suitable for machine learning model input.
- **Multinomial Naive Bayes Algorithm:**
  - A probabilistic classification algorithm based on Bayes' theorem.
  - Assumes features (word counts) are conditionally independent given the class label.
  - Predicts the class with the highest posterior probability.
- **Flask Request Handling Algorithm:**
  - Receives user input via HTTP POST from the web interface [11].
  - Passes input to the CountVectorizer for feature extraction.
  - Uses the trained Naive Bayes model to predict sentiment.
  - Returns the prediction result to the web interface for display.

2.3 Techniques:

- **Natural Language Processing (NLP):** Used to process and analyze textual movie reviews for sentiment classification.
- **Tokenization:** Splitting input text into individual words (tokens) to prepare for vectorization.
- **Feature Extraction (CountVectorizer):** Converts text into numerical feature vectors based on word frequency [19], enabling machine learning models to process text data effectively.
- **Machine Learning Classification:** Utilizes the Multinomial Naive Bayes algorithm for probabilistic sentiment prediction.
- **Model Serialization:** Uses pickle/joblib to save and load the trained vectorizer and classifier for real-time use in the web application [5].
- **Web Deployment (Flask):** Provides a lightweight, scalable web framework to handle user input and deliver predictions seamlessly.

- **API Documentation (Swagger/Flasgger):** Integrated for testing and extending API endpoints, ensuring structured and documented API deployment.

2.4 Tools:

- **Python:**

Used as the core programming language due to its rich libraries for machine learning, natural language processing, and web development, ensuring efficient model development and deployment.

- **Flask:**

A lightweight, micro web framework in Python that helps build and deploy the web application for real-time sentiment analysis [15], handling routing, HTTP requests, and rendering HTML templates.

- **Scikit-learn:**

Provides the CountVectorizer for converting text into numerical feature vectors and Multinomial Naive Bay**es** for classification, enabling easy implementation of machine learning workflows [18].

- **Flasgger (Swagger):**

Integrated with Flask to automatically generate API documentation, allowing developers and testers to interact with the API endpoints easily and test model predictions without building a separate client.

- **Pandas & NumPy:**

Used for data manipulation and numerical operations during data preprocessing and exploratory analysis while preparing the dataset for model training.

2.5 Methods:

The Flask Naive Bayes Sentiment Analysis project uses the following methods for effective implementation and deployment:

- **Data Collection and Preparation:** A dataset of labeled movie reviews is created, containing examples marked as positive or negative [5]. Basic preprocessing is applied, such as converting text to lowercase and removing unnecessary whitespace.
- **Feature Extraction:** The CountVectorizer method is used to tokenize the text data and convert it into numerical feature vectors based on word frequency [3], preparing it for machine learning classification.
- **Model Training:** A Multinomial Naive Bayes classifier is trained on the feature vectors and corresponding sentiment labels, learning the patterns in word usage for different sentiment classes.
- **Model Serialization:** Using pickle/joblib [9], the trained Naive Bayes model and the CountVectorizer are saved to disk, enabling reuse in the Flask application without the need for retraining.
- **Flask Application Development:** A Flask web application is developed to create a user-friendly interface for inputting movie reviews. It handles routing, processes user input, and displays the sentiment result.
- **Real-Time Prediction:** When a user submits a review [22], the application retrieves the input, uses the loaded CountVectorizer to transform it, and applies the pre-trained Naive Bayes model to predict the sentiment.
- **Output Display:** The predicted sentiment (positive or negative) is displayed instantly on the web interface for the user's review input, ensuring real-time interaction.
- **API Documentation with Flasgger:** Flasgger (Swagger) is integrated into the Flask app to document API endpoints, allowing for structured testing, extension, and easier collaboration for future development.

III. METHODOLOGY

3.1 Inputs:

This project implements a sentiment analysis system using Flask and a Multinomial Naive Bayes classifier to classify movie reviews as positive or negative [6]. It uses Natural Language Processing techniques to process user-submitted text and determine the underlying sentiment efficiently. The system utilizes CountVectorizer for feature extraction, converting text data into numerical vectors that can be used by the machine learning model for prediction [19]. The trained model and vectorizer are serialized using pickle, allowing them to be loaded into the Flask web application for real-time sentiment prediction without retraining. The Flask framework provides a lightweight web interface [20], making it user-friendly and accessible to non-technical users for analyzing their movie reviews.

- **Movie Review Text:** The primary input, entered by the user through the web interface.
- **Pre-trained CountVectorizer:** Used to transform the input text into numerical feature vectors.

- **Pre-trained Multinomial Naive Bayes Model:** Loaded for sentiment prediction on transformed input data.
- **HTTP POST Requests:** Facilitates submission of user input from the web form to the Flask server.
- **Stop Words Handling (via CountVectorizer):** Implicitly filters out common, less meaningful words during vectorization.
- **User Interaction:** Simple web form input, requiring no technical expertise from the user.
- **Text Data Format:** Plain text input without any special formatting constraints.

Fig: 3 Input Screen.

Fig: 4 Input Screen.

3.2 Method of Process:

The method of process for this project involves systematically transforming user-submitted movie reviews into sentiment predictions using machine learning techniques within a web-based environment. Initially, the user enters a review into the web interface [16], which is handled by the Flask server using an HTTP POST request. The system then applies text preprocessing using CountVectorizer, converting the raw text into numerical feature vectors suitable for machine learning models.

- **User Input Collection:** The user enters a movie review as plain text into the web interface provided by the Flask application.
- **Request Handling:** The Flask server receives the user input via an HTTP POST request and passes it to the backend for processing.
- **Text Preprocessing:** The input text is transformed using CountVectorizer [13], converting it into numerical feature vectors based on word frequency.
- **Model Loading:** A pre-trained Multinomial Naive Bayes model, serialized using pickle, is loaded for prediction without retraining.
- **Sentiment Prediction:** The transformed input is passed to the Naive Bayes model to predict the sentiment as positive or negative.
- **Result Rendering:** The prediction result is sent back to the Flask server and displayed on the web interface for the user [6].
- **API Documentation:** Swagger (Flasgger) integration allows easy testing and extension of the API for developers.

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 07 | July – 2025 DOI: 10.55041/ISJEM04878
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Fig: 5 Method of Process.

3.3 Output:

The Flask Naive Bayes Sentiment Analysis project provides clear and immediate outputs to users after they submit their movie reviews. Once a review is entered and submitted, the system processes the input and predicts whether the sentiment of the review is positive or negative. This output is displayed clearly on the web interface, allowing users to instantly understand the sentiment conveyed in their review [6]. The system ensures real-time feedback, making it interactive and user-friendly for non-technical users. Additionally, if accessed through API endpoints, the system can provide the sentiment result in JSON format, supporting further integration with other applications or dashboards [23].

- **Sentiment Result:** The system displays whether the movie review is Positive or Negative based on analysis.
- **Web Page Display:** The output is shown clearly on the result page within the Flask web interface for immediate user understanding.
- **Instant Feedback:** Users receive real-time sentiment analysis without delay, enhancing interactivity.
- **Optional JSON Output:** If accessed via API [11], the output can be returned as a JSON object for integration with other applications.
- **User-Friendly Experience:** The output is presented in a simple [16], readable format, making it easy for non-technical users to interpret.
- **Validation for Learning:** Enables users to test different types of reviews and observe how the model interprets sentiment.
- **Foundation for Analytics:** The output can be logged and used for further sentiment analytics or report generation if extended.

Fig. 6 Output Screen.

Fig. 7 Output Screen.

IV. RESULTS

The Flask Naive Bayes Sentiment Analysis system was successfully implemented to classify movie reviews as positive or negative in real-time. The system accurately predicts the sentiment of user-submitted reviews using a Multinomial Naive Bayes classifier trained on labeled review data and feature extraction using CountVectorizer. Users were able to enter various movie reviews, and the system provided instant sentiment predictions displayed on the web interface, demonstrating the model's effectiveness for short, direct review texts. The system delivered consistent and fast responses, confirming the successful deployment of the machine learning model within a lightweight Flask framework. The integration of Flasgger enabled structured

API testing, verifying that the endpoints worked correctly for automated and manual test inputs. Overall, the results indicate that the system performs reliably for sentiment analysis on small-scale review data, providing an accessible tool for learning and practical applications in real-time sentiment classification.

V. DISCUSSIONS

The implementation of this Flask Naive Bayes Sentiment Analysis project demonstrates the practical workflow of deploying a machine learning model for real-time use. Using the Multinomial Naive Bayes algorithm, the system efficiently classifies movie reviews as positive or negative, making it lightweight and fast for small-scale applications. One key observation during testing was that short, clear reviews were classified accurately, while ambiguous or sarcastic sentences occasionally led to misclassification, reflecting the limitations of using simple models and small datasets. This indicates that while the system is effective for straightforward sentiment analysis, handling complex language nuances would require advanced preprocessing and larger, diverse datasets. The integration with Flask allowed for smooth routing, real-time user interaction, and easy deployment, while Flasgger provided structured API documentation, making it easier for further extension of the system. The project effectively bridges the gap between model training and live deployment, providing students and beginners with a practical understanding of machine learning application deployment in NLP.

VI. CONCLUSION

The Flask Naive Bayes Sentiment Analysis project successfully demonstrates the deployment of a machine learning model for real-time sentiment classification of movie reviews. By utilizing CountVectorizer for feature extraction and the Multinomial Naive Bayes algorithm for classification, the system efficiently predicts whether user-submitted reviews are positive or negative. The integration with the Flask framework ensures lightweight, user-friendly deployment, making the system accessible even to non-technical users. Additionally, the use of Flasgger for API documentation enables structured testing and potential system expansion. Through this project, students and beginners can understand the end-to-end pipeline of building, training, and deploying machine learning models in a real-world environment. The project provides a practical foundation for learning natural language processing and sentiment analysis while demonstrating the ease of integrating machine learning models into web applications for interactive use.

VII. FUTURE SCOPE

The Flask Naive Bayes Sentiment Analysis project provides a strong foundation for expanding sentiment analysis capabilities in real-world applications. In the future, the system can be improved by using larger and more diverse datasets to increase accuracy and handle varied review patterns effectively. Advanced text preprocessing methods such as stemming, lemmatization, and handling negations can further refine feature extraction, improving model performance. Additionally, integrating deep learning models like LSTM and BERT can help capture complex linguistic patterns, including sarcasm and context-based sentiment shifts. The system can also be extended to support multilingual sentiment analysis, allowing a broader range of user reviews to be analyzed. Deploying the system on cloud platforms will enable scalable public access, while integrating it with live dashboards can help monitor sentiment trends in real-time.

VIII. ACKNOWLEDGEMENT

Miss. MAMIDI TARANI working as an Assistant Professor in Master of Computer Applications (MCA) in Sanketika Vidya Parishad Engineering College, Visakhapatnam, Andhra Pradesh. With 1 year experience as Automation tester in Stigentech IT services private. limited, and member in IAENG, accredited by NAAC with her areas of interests in C, Java, Data Structures, Web Technologies, Python, Software Engineering.

Kurmapu Durgaprasad is pursuing his final semester MCA in Sanketika Vidya Parishad Engineering College, accredited with A grade by NAAC, affiliated by Andhra University and approved by AICTE. With interest in Machine learning Kurmapu Durgaprasad has taken up her PG project on IMDB sentiment analysis based on comment by using machine learning and published the paper in connection to the project under the guidance of M. Tarani, Assistant Professor, SVPEC.

REFERENCES

[1] Python Official Documentation:
- Official Website: https://docs.python.org/3/
- The fundamental reference for Python language syntax, built-in functions, and standard library modules.

[2] Flask (Web Framework):
- Official Documentation: https://flask.palletsprojects.com/
- The primary resource for building web applications with Flask, covering routing, requests, responses, and templating.

[3] Scikit-learn (Machine Learning Library):
- Official Documentation: https://scikit-learn.org/stable/
- The comprehensive guide for machine learning algorithms, model selection, and preprocessing tools in Python.

[4] NumPy (Numerical Computing):
- Official Documentation: https://numpy.org/doc/
- Essential for high-performance numerical operations, especially array and matrix manipulations crucial for machine learning.

[5] Pandas (Data Analysis and Manipulation):
- Official Documentation: https://pandas.pydata.org/docs/
- Indispensable for data structuring, cleaning, and analysis using Data Frames.

[6] Joblib (Persistence of Python Objects):
- Official Documentation: https://joblib.readthedocs.io/en/latest/
- Recommended for efficiently saving and loading large Python objects, particularly NumPy arrays and scikit-learn models.

[7] Pickle (Python Object Serialization):
- Official Documentation: https://docs.python.org/3/library/pickle.html
- A standard Python module for serializing and deserializing Python object structures.

[8] Flasgger (Swagger/OpenAPI UI for Flask):
- GitHub Repository/Documentation: https://github.com/flasgger/flasgger
- Used for integrating Swagger UI to automatically generate interactive API documentation for Flask applications.

[9] CountVectorizer (Scikit-learn):
- Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
- Details on converting text documents into numerical feature vectors based on token counts.

[10] Multinomial Naive Bayes (Scikit-learn):
- Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
- Specifics of the Naive Bayes algorithm, particularly suited for text classification with discrete features.

[11] Text Preprocessing for NLP:
- NLTK Book (Chapter 3 - Processing Raw Text): https://www.nltk.org/book/ch03.html
- Covers techniques like tokenization, stemming, lemmatization, and stop-word removal, crucial for preparing text data for ML models.

[12] Sentiment Analysis Overview:
- Wikipedia: https://en.wikipedia.org/wiki/Sentiment_analysis
- A general introduction to the field of sentiment analysis (also known as opinion mining).

[13] Model Persistence (General ML Concept):
- Scikit-learn documentation on Model Persistence: https://scikit-learn.org/stable/modules/model_persistence.html

- Explains why and how to save and load trained machine learning models.

[14] Cross-Validation:
- Scikit-learn documentation on Cross-validation: https://scikit-learn.org/stable/modules/cross_validation.html
- A technique for evaluating machine learning models by training them on subsets of the input data and testing them on unseen data.

[15] Jinja2 (Templating Engine):
- Official Documentation: https://jinja.palletsprojects.com/en/3.1.x/
- The templating engine used by Flask for rendering dynamic HTML pages (render_template).

[16] HTTP Methods (GET, POST):
- MDN Web Docs - HTTP request methods: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
- Explains the different types of requests clients can make to a web server, relevant for methods=['POST'].

[17] RESTful API Design Principles:
- IBM Cloud Learn Hub - What is a REST API?: https://www.ibm.com/cloud/learn/rest-api
- Guidelines for designing web services that are stateless, client-server, cacheable, and layered.

[18] Handling Forms in Flask:
- Flask documentation on Request Objects: https://flask.palletsprojects.com/en/3.0.x/reqcontext/#the-request-object
- Details on how Flask handles incoming request data, including form submissions (request.form).

[19] Git (Version Control System):
- Pro Git Book: https://git-scm.com/book/en/v2
- An essential tool for tracking changes in source code, collaborating, and managing project history.

[20] Virtual Environments (Python):
- Python Docs - venv: https://docs.python.org/3/library/venv.html
- Best practice for managing project dependencies and avoiding conflicts between different Python projects.

[21] Pip (Python Package Installer):
- Pip User Guide: https://pip.pypa.io/en/stable/user_guide/
- The standard package-management system used to install and manage software packages written in Python.

[22] WSGI (Web Server Gateway Interface):
- PEP 3333 -- Python Web Server Gateway Interface v1.0.1: https://peps.python.org/pep-3333/
- A standard interface between web servers and web applications or frameworks for Python.

[23] Gunicorn (WSGI HTTP Server):
- Official Documentation: https://gunicorn.org/
- A common production-ready WSGI server used to deploy Flask applications.

[24] Docker (Containerization):
- Official Documentation: https://docs.docker.com/
- A platform for developing, shipping, and running applications in isolated environments called containers.

[25] Heroku (Cloud Platform):
- Official Documentation: https://devcenter.heroku.com/categories/reference
- A popular platform-as-a-service (PaaS) often used for deploying Flask applications.