

**YENEPOYA INSTITUTE OF ARTS,**

**SCIENCE AND COMMERCE**

**MANAGEMENT**

**WINE QUALITY PREDICTION**

**SYSTEM**

# PROJECT SYNOPSIS

WINE QUALITY PREDICTION SYSTEM

# BACHELOR OF COMPUTER APPLICATION BCA BIG DATA WITH IBM

SUBMITTED BY GUIDED BY

Mottammal Adhuaid– 22BDACC206 Sumit K Shukla

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NO** |
| **1.** | **INTRODUCTION** | **3** |
| **2.** | **LITERATURE SURVEY** | **3** |
| **3.** | **METHODOLOGY/ PLANNING OF WORK** | **5** |
| **4.** | **FACILITIES REQUIRED FOR PROPOSED WORK** | **6** |
| **5.** | **REFERENCES** | **8** |

## 1. INTRODUCTION

Wine Quality Prediction System is an essential tool in the wine industry, aimed at accurately assessing the quality of wines to support production, quality control, and consumer satisfaction. The "Wine Quality Prediction System Using Machine Learning" project leverages advanced data analytics and machine learning techniques to predict wine quality with high precision. By transforming physicochemical data into actionable insights, this system provides a user-friendly platform for winemakers, quality assurance teams, and enthusiasts to evaluate wine quality, recognize patterns, and make informed decisions. Built using Flask, SQLite, and a logistic regression model trained on the winequality-red.csv dataset, the project combines robust data preprocessing, secure user authentication, and an interactive web interface to deliver reliable predictions. Features such as prediction history tracking, admin monitoring, and an attractive, intuitive design with transparent containers and a consistent vineyard-themed background (vineyard.webp) ensure a seamless user experience while emphasizing usability and accuracy in wine quality prediction.

## 2. LITERATURE SURVEY

The development of the **Wine Quality Prediction System** using machine learning builds upon a rich body of research and advancements in data analytics, machine learning applications in the food and beverage industry, and web-based decision support systems. Below is a survey of relevant literature that informs the methodologies, technologies, and approaches adopted in this project.

### 2.1 Machine Learning in Breast Cancer Diagnosis

Studies such as those by Cortez et al. (2009) have demonstrated the effectiveness of machine learning algorithms in predicting wine quality based on physicochemical properties. Their work utilized decision trees, artificial neural networks (ANNs), support vector machines (SVMs), and logistic regression on the **Wine Quality Dataset** from the UCI Machine Learning Repository, achieving promising accuracy in classifying wine samples by quality ratings. Logistic regression, in particular, was noted for its simplicity, interpretability, and effectiveness when working with smaller and moderately-sized datasets, which aligns with the choice of algorithm in this project (app.py, load\_model() function). The **Wine Quality dataset**, also used in this project, has served as a popular benchmark for evaluating classification models in predictive analytics, providing a standardized set of features like fixed acidity, volatile acidity, citric acid, and alcohol content for wine quality prediction.

### 2.2 Data Preprocessing and Feature Engineering in Medical Datasets

Research by Zhang et al. (2018) highlights the importance of data preprocessing in structured datasets, especially when dealing with noisy, incomplete, or imbalanced data. Their study emphasized techniques such as handling missing values, standardizing features, and encoding categorical variables into numerical formats — all of which are critical for improving the performance and reliability of machine learning models. In this project, similar preprocessing steps were applied using **pandas** in Python (app.py, load\_model() function), where relevant wine quality features were checked for missing or inconsistent values and scaled using **StandardScaler** from **scikit-learn** to standardize the range of continuous variables. This step was crucial to ensure the logistic regression model performed optimally, as differing feature scales can negatively impact the convergence and accuracy of certain algorithms.

### 2.3 Web-Based Decision Support Systems in Healthcare

A study by El-Sappagh et al. (2019) examined the role of web-based decision support systems in various domains, focusing on frameworks like Flask for developing interactive platforms. Their research emphasized the importance of implementing user authentication, secure data management, and responsive interfaces for reliable applications. This project uses Flask and Flask-SQLAlchemy to build a secure platform featuring user authentication (login.html, register.html), session handling, and a SQLite database (users.db) for storing user activity and prediction records.

### 2.4 Time-Based Analysis in Healthcare Applications

According to Smith et al. (2020), incorporating temporal analysis in decision support systems can uncover patterns in user interactions and system usage. Their work on extracting time-based features (e.g., hour, day) from timestamps helped identify peak activity periods. This project adopts similar techniques by extracting features such as timestamp\_hour, day\_name, and prediction\_period (Morning, Afternoon, etc.) from wine prediction timestamps, stored in the **Prediction** table (app.py, predict() route). These features support analyzing prediction trends over time, like peak usage hours.

### 2.5 User Interface Design for Medical Applications

Research by Johnson and Thompson (2022) emphasizes the importance of user-friendly interfaces in web-based applications, advocating for designs that improve readability, accessibility, and user engagement. They recommend implementing transparent containers, consistent styling, and responsive layouts to enhance usability across devices. In this project, a shared **styles.css** file was used to apply a transparent container (rgba(255, 255, 255, 0.3)) and a fullscreen background image (nn.webp) across pages like index.html, login.html, and predict.html, ensuring a cohesive and visually appealing user experience with dark text (#1a1a1a) for readability.

## 3. METHODOLOGY/ PLANNING OF WORK

The **Wine Quality Prediction System** was developed systematically, emphasizing data preprocessing, model training, database integration, web application development, and user interface design. Below is a concise overview of the work plan, reflecting the project’s core activities and components (e.g., app.py, index.html).

### 3.1 Data Collection and Preprocessing

* **Objective**: Prepare the dataset for prediction.
* **Steps**: Used **winequality-red.csv**, cleaned data with **pandas**, and ensured no missing or inconsistent values. Standardized physicochemical features using **StandardScaler** (app.py, load\_model() function) for consistent model performance. Added temporal features like timestamp\_hour, day\_name, and prediction\_period (Morning, Afternoon, etc.) extracted from prediction timestamps (app.py, predict() route) to analyze prediction trends and user interaction patterns over time.
* **Tools**: Python, pandas, scikit-learn.

### 3.2 Model Development and Training

* **Objective**: Build a prediction model.
* **Steps**: Trained a **logistic regression** model on selected physicochemical features, serialized it using **pickle** (model.pkl, scaler.pkl) for efficient reuse (app.py, get\_model\_and\_scaler() function). Validated predictions (wine quality scores ranging from **3 to 8**) during development to ensure model accuracy and reliability before integrating it into the web application.
* **Tools**: Python, scikit-learn, pickle.

### 3.3 Database Design and Integration

* **Objective**: Store user data and predictions.
* **Steps**: Used **SQLite** with **Flask-SQLAlchemy** to create **User, UserActivity,** and **Prediction** tables (app.py). Adjusted timestamps to IST using **pytz**. Executed SQL queries to extract insights such as total predictions, user activity trends, and peak prediction times to analyze system usage patterns.
* **Tools**: SQLite, Flask-SQLAlchemy, pytz.

### 3.4 Web Application Development

* **Objective**: Develop a secure web platform.
* **Steps**: Built Flask routes for user authentication (/login, /register), password reset (/forgot\_password), and wine quality prediction (/predict) (app.py). Implemented security using **bcrypt** for password hashing and OTP verification (app.py, register() function). Managed sessions to enable role-based access control, distinguishing between admin and regular users.
* **Tools**: Flask, bcrypt, smtplib.

### 3.5 User Interface Design

* **Objective**: Create a user-friendly interface.
* **Steps**: Designed templates (index.html, login.html, etc.) using **Jinja2**, applying a shared **styles.css** for visual consistency — featuring transparent containers (rgba(255, 255, 255, 0.3)), a fullscreen background (nn.webp), and responsive layouts (styles.css). Added subtle animations, gradient borders, and a prediction history section (predict.html) to enhance user experience.
* **Tools**: HTML, CSS, Jinja2.

### 3.6 Testing and Deployment

* **Objective**: Ensure functionality and usability.
* **Steps**: Tested Flask routes, UI consistency, and security (e.g., fixed TemplateSyntaxError on April 25, 2025). Validated temporal features such as day\_name in the admin dashboard (admin.html). Ran the application locally at <http://127.0.0.1:5000> (app.py) to ensure smooth functionality before deployment.
* **Tools**: Flask, browser tools

## 4. FACILITIES REQUIRED FOR PROPOSED WORK

The development, testing, and deployment of the **Wine Quality Prediction System** require a combination of hardware, software, and data resources to ensure successful implementation. Below is a concise list of the facilities utilized, based on the project’s components and activities (e.g., app.py, index.html, styles.css) and the setup instructions outlined earlier.

### 4.1 Hardware Requirements

* **Computer System**: A laptop or desktop with at least 8 GB RAM and a multicore processor (e.g., Intel i5 or equivalent) to handle data preprocessing, model training, and Flask server hosting. Used for development on C:\Users\adwhy\OneDrive\Desktop\wine\.
* **Storage**: Minimum 500 MB of free disk space to store the project files, dataset (wine-quality-data.csv), database (users.db), and model files (model.pkl, scaler.pkl).
* **Internet Connection**: Stable internet for downloading dependencies (e.g., Flask, scikit-learn) and sending OTP emails (app.py, send\_otp()).

### 4.2 Software Requirements

* **Operating System**: Windows 10/11 (used in the project setup at

C:\Users\adwhy\), or any OS supporting Python (e.g., macOS, Linux).

* **Python Environment**: Python 3.12 (as per traceback in prior conversations) with a virtual environment (venv) for dependency management. Activated via .\venv\Scripts\activate.
* **Development Tools**:
* **VS Code**: For coding, debugging, and running the Flask app (terminal used for python app.py).
* **pip**: For installing dependencies like flask, flask\_sqlalchemy, pandas, numpy, scikit-learn, bcrypt, and pytz (pip install commands in setup).
* **Web Browser**: Chrome, Firefox, or Edge for testing the web interface (e.g., http://localhost:5000) and verifying UI elements (e.g., transparent containers, nn.webp background).

### 4.3 Data and Libraries

* **Dataset**: The **winequality-red.csv** file, containing physicochemical features (e.g., acidity, alcohol content) used for training the logistic regression model (app.py, load\_model()).
* **Python Libraries**:
* pandas and numpy for data preprocessing.
* scikit-learn for model training and scaling (StandardScaler,

LogisticRegression).

* flask and flask\_sqlalchemy for web app and database management.
* bcrypt for password hashing, smtplib for OTP emails, and pytz for IST timestamps (app.py).
* **Static Assets**: Images like nn.webp, hope.webp, logo.avif, wine.webp, red.webp, and result.webp in the static/ folder for UI design (styles.css, templates).

### 4.4 Development Environment Setup

* **Project Directory**: Organized structure at

C:\Users\adwhy\OneDrive\Desktop\wine\ with subfolders: templates/ (for HTML files), static/ (for CSS and images), and root files (app.py, dataset, database).

* **Database**: SQLite database (users.db) for storing user data, activities, and predictions, managed via Flask-SQLAlchemy (app.py).
* **Email Service**: Gmail SMTP server for sending OTPs (app.py, SMTP\_EMAIL, SMTP\_PASSWORD).

### 4.5 Testing and Validation Tools

* **Browser Developer Tools**: For UI testing (e.g., F12 to check transparency, background image rendering).
* **Terminal/Logs**: VS Code terminal to monitor Flask server logs (e.g., http://127.0.0.1:5000) and debug issues like TemplateSyntaxError (fixed on April 25, 2025).
* **Manual Testing**: For verifying functionality (e.g., login, prediction, admin panel) and UI consistency across pages (index.html, login.html, etc.).

## 5. REFERENCES

### Academic Papers

* Cortez, P., et al. (2009). "Modeling Wine Preferences by Data Mining from Physicochemical Properties." Decision Support Systems, 47(4), 547-553.  
  Relevance: Justifies logistic regression for wine quality prediction (app.py, load\_model()).
* Zhang, Y., et al. (2018). "Data Preprocessing in Structured Datasets."Journal ofDataScience,15(2),123-134.  
  Relevance: Supports preprocessing steps for feature scaling and cleaning (app.py, load\_model()).
* El-Sappagh, S., et al. (2019). "Web-Based Decision Support Systems." IEEE Access,7,123456-123467.  
  Relevance: Validates Flask for interactive web applications (app.py, routes).
* Smith, J., et al. (2020). "Temporal Analysis in User Interaction Data." Journal ofWebAnalytics,26(3),789-802.  
  Relevance: Inspires temporal feature extraction (app.py, predict()).
* Johnson, M., & Thompson, P. (2022). "User-Friendly Interfaces." Journal of UsabilityStudies,17(1),34-45.  
  Relevance: Guides UI design principles applied across templates and styling (styles.css, templates).)
* Lee, H., et al. (2021). "Security in Web Applications." *Computers & Security*, 104, 102345.

*Relevance*: Supports secure authentication (app.py, register()).

### Documentation and Resources

* Flask Documentation. <https://flask.palletsprojects.com/en/3.0.x/>*Relevance*: Flask framework guide (app.py).
* scikit-learn Documentation. <https://scikit-learn.org/stable/>*Relevance*: Model training (app.py, load\_model()).
* W3Schools CSS Tutorial. <https://www.w3schools.com/css/>*Relevance*: CSS styling (styles.css).