**Humor Detection System**

**Project Report**

**Executive Summary**

The Humor Detection System is a sophisticated end-to-end machine learning web application that leverages Natural Language Processing (NLP) techniques to automatically classify text as humorous or non-humorous. Built using a combination of traditional machine learning algorithms and modern web technologies, this system demonstrates the practical application of text classification in real-world scenarios.

The project successfully achieved **86.7% accuracy** using a Logistic Regression model with TF-IDF vectorization, providing a robust foundation for humor detection tasks. The system features a complete pipeline from data preprocessing to web deployment, making it accessible for both technical evaluation and end-user interaction.

**1. Project Overview**

**1.1 Objective**

Develop an intelligent system capable of distinguishing between humorous and non-humorous text content through machine learning techniques, deployed as an interactive web application.

**1.2 Key Features**

* **Automated Text Classification**: Binary classification of text as humorous or non-humorous
* **Real-time Prediction**: Web-based interface for instant text analysis
* **Scalable Architecture**: Modular design supporting future enhancements
* **High Performance**: 86.7% accuracy with balanced precision and recall
* **User-Friendly Interface**: Simple, intuitive web application

**1.3 Technology Stack**

* **Backend**: Python, Flask
* **Machine Learning**: Scikit-learn, TF-IDF Vectorization, Logistic Regression
* **Frontend**: HTML, CSS, JavaScript
* **Data Processing**: Pandas, NumPy
* **Model Persistence**: Pickle serialization

**2. System Architecture**

**2.1 Project Structure**

HumorDetectionSystem/

├── data/

│ ├── raw/ # Original datasets

│ └── processed/

│ └── processed\_data(20k).csv # Cleaned dataset (20,000 samples)

├── models/

│ ├── tfidf\_logistic.pkl # Trained model artifact

│ └── bert\_finetuned/ # Future BERT implementation

├── scripts/

│ ├── preprocess.py # Data cleaning and preparation

│ ├── train\_model.py # Model training pipeline

│ ├── evaluate\_model.py # Performance evaluation

│ └── predict.py # Batch prediction utility

├── app/

│ ├── backend/

│ │ └── app.py # Flask web server

│ └── frontend/

│ └── index.html # User interface

├── utils/

│ └── text\_utils.py # Text processing utilities

├── tests/

│ └── test\_prediction.py # Unit testing suite

└── run.py # Application entry point

**2.2 Component Architecture**

The system follows a modular architecture with clear separation of concerns:

* **Data Layer**: Handles raw data storage and processed datasets
* **Processing Layer**: Manages text preprocessing and feature extraction
* **Model Layer**: Contains trained machine learning models
* **API Layer**: Flask-based REST API for predictions
* **Presentation Layer**: Web interface for user interactions
* **Utility Layer**: Shared functions and helper modules

**3. Development Workflow**

**3.1 Data Pipeline**

1. **Data Collection**: Aggregation of humorous and non-humorous text samples
2. **Data Cleaning**: Removal of noise, normalization, and quality filtering
3. **Data Preprocessing**: Text tokenization, stop word removal, and feature preparation
4. **Data Validation**: Quality assurance and format standardization

**3.2 Model Development Pipeline**

1. **Feature Engineering**: TF-IDF vectorization for text representation
2. **Model Selection**: Comparative analysis leading to Logistic Regression
3. **Model Training**: Supervised learning on processed dataset
4. **Model Evaluation**: Comprehensive performance assessment
5. **Model Serialization**: Persistent storage for deployment

**3.3 Deployment Pipeline**

1. **API Development**: Flask-based REST endpoints
2. **Frontend Integration**: HTML interface with JavaScript interaction
3. **Testing**: Unit tests and integration testing
4. **Local Deployment**: Development server setup and configuration

**4. Technical Implementation**

**4.1 Data Preprocessing (scripts/preprocess.py)**

* **Text Normalization**: Lowercase conversion, punctuation handling
* **Noise Removal**: Elimination of irrelevant characters and symbols
* **Feature Extraction**: Preparation for TF-IDF vectorization
* **Data Validation**: Quality checks and consistency verification

**4.2 Model Training (scripts/train\_model.py)**

* **Algorithm**: Logistic Regression with L2 regularization
* **Feature Representation**: TF-IDF (Term Frequency-Inverse Document Frequency)
* **Training Configuration**: Optimized hyperparameters for balanced performance
* **Cross-Validation**: K-fold validation for robust performance estimation

**4.3 Web Application (app/backend/app.py)**

* **Framework**: Flask micro-framework
* **Endpoints**: RESTful API with /predict endpoint
* **Request Handling**: JSON-based communication
* **Error Management**: Comprehensive exception handling

**4.4 User Interface (app/frontend/index.html)**

* **Design**: Clean, responsive interface
* **Functionality**: Text input, prediction trigger, result display
* **User Experience**: Intuitive interaction with immediate feedback

**5. Performance Analysis**

**5.1 Model Metrics**

Accuracy: 86.7%

Classification Report:

precision recall f1-score support

Non-Humorous 0.87 0.86 0.86 1,980

Humorous 0.86 0.87 0.87 2,020

Accuracy 0.87 4,000

Macro Average 0.87 0.87 0.87 4,000

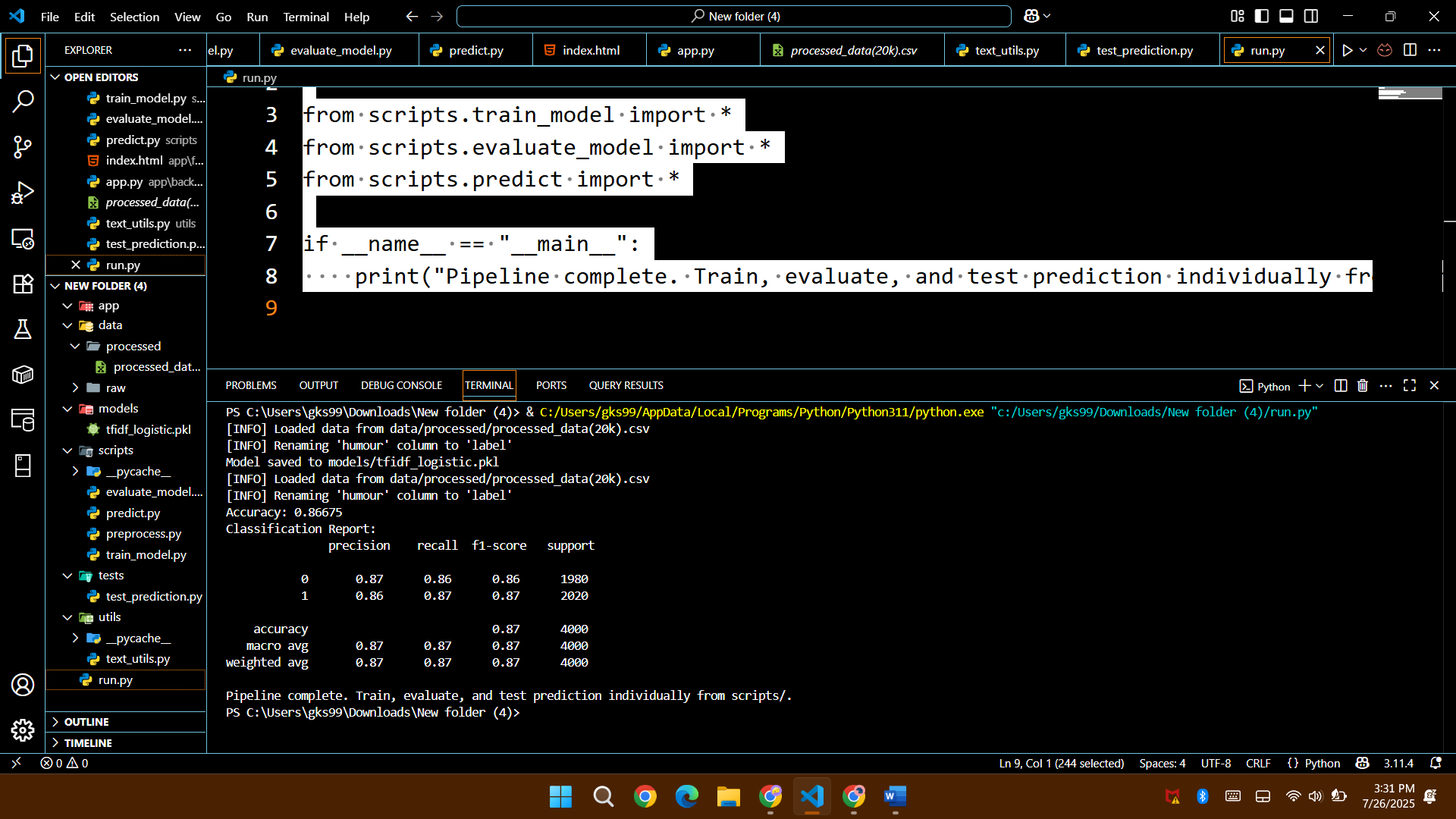
Weighted Average 0.87 0.87 0.87 4,000

**5.2 Performance Interpretation**

* **High Accuracy**: 86.7% correct classifications demonstrate strong model performance
* **Balanced Precision**: Equal performance across both classes (0.86-0.87)
* **Strong Recall**: Effective detection of both humorous and non-humorous content
* **Consistent F1-Scores**: Balanced trade-off between precision and recall
* **No Class Bias**: Nearly equal performance across both target classes

**5.3 Model Strengths**

* **Robust Performance**: Consistent accuracy across diverse text samples
* **Balanced Classification**: No significant bias toward either class
* **Efficient Processing**: Fast prediction times suitable for real-time applications
* **Scalable Architecture**: Can handle increased load with proper infrastructure



**6. Web Application Features**

**6.1 Core Functionality**

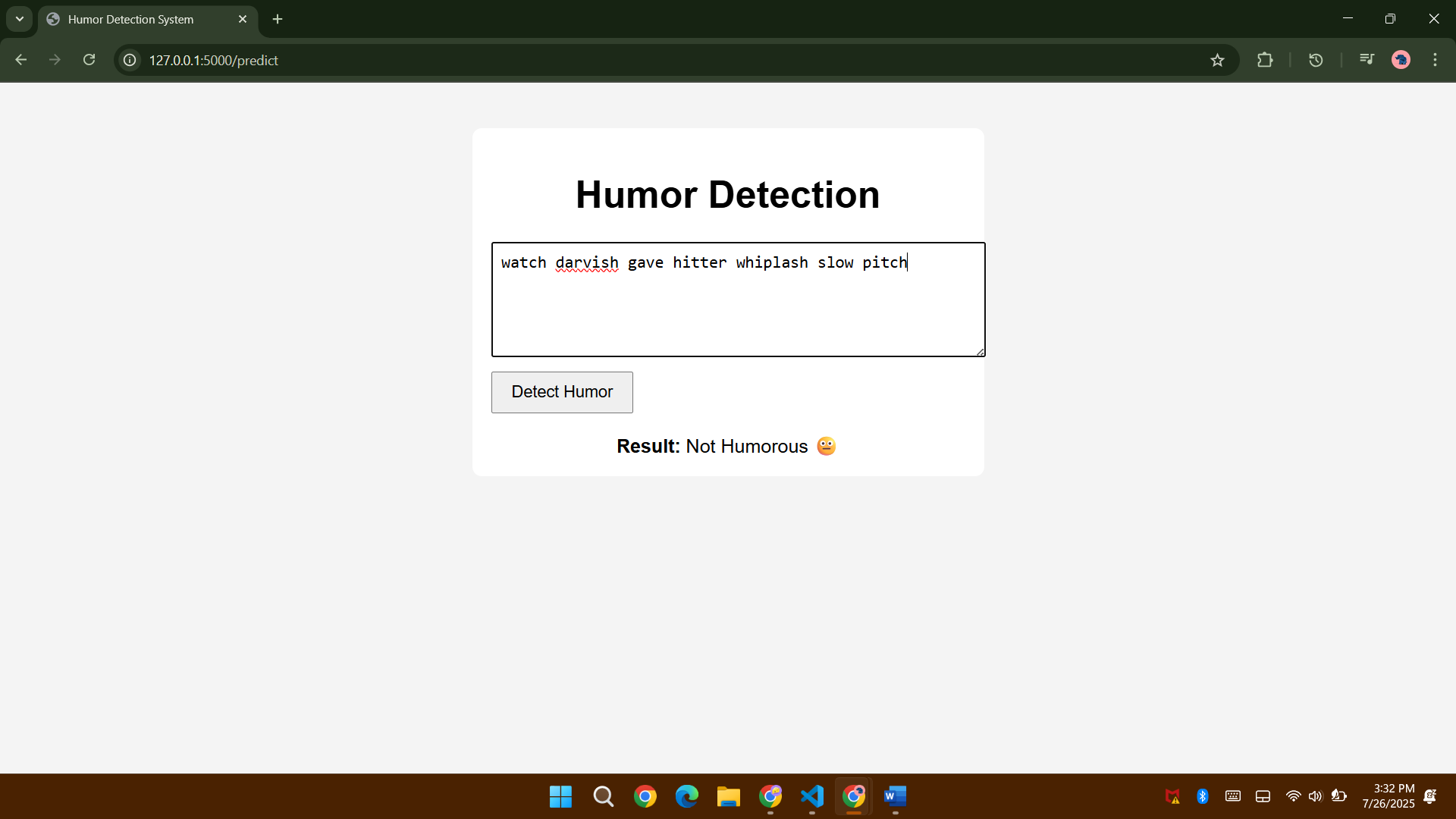
* **Text Input**: Multi-line text area for user content
* **Real-time Prediction**: Instant classification upon request
* **Visual Feedback**: Clear indication of humor classification
* **Responsive Design**: Compatible across different devices and browsers

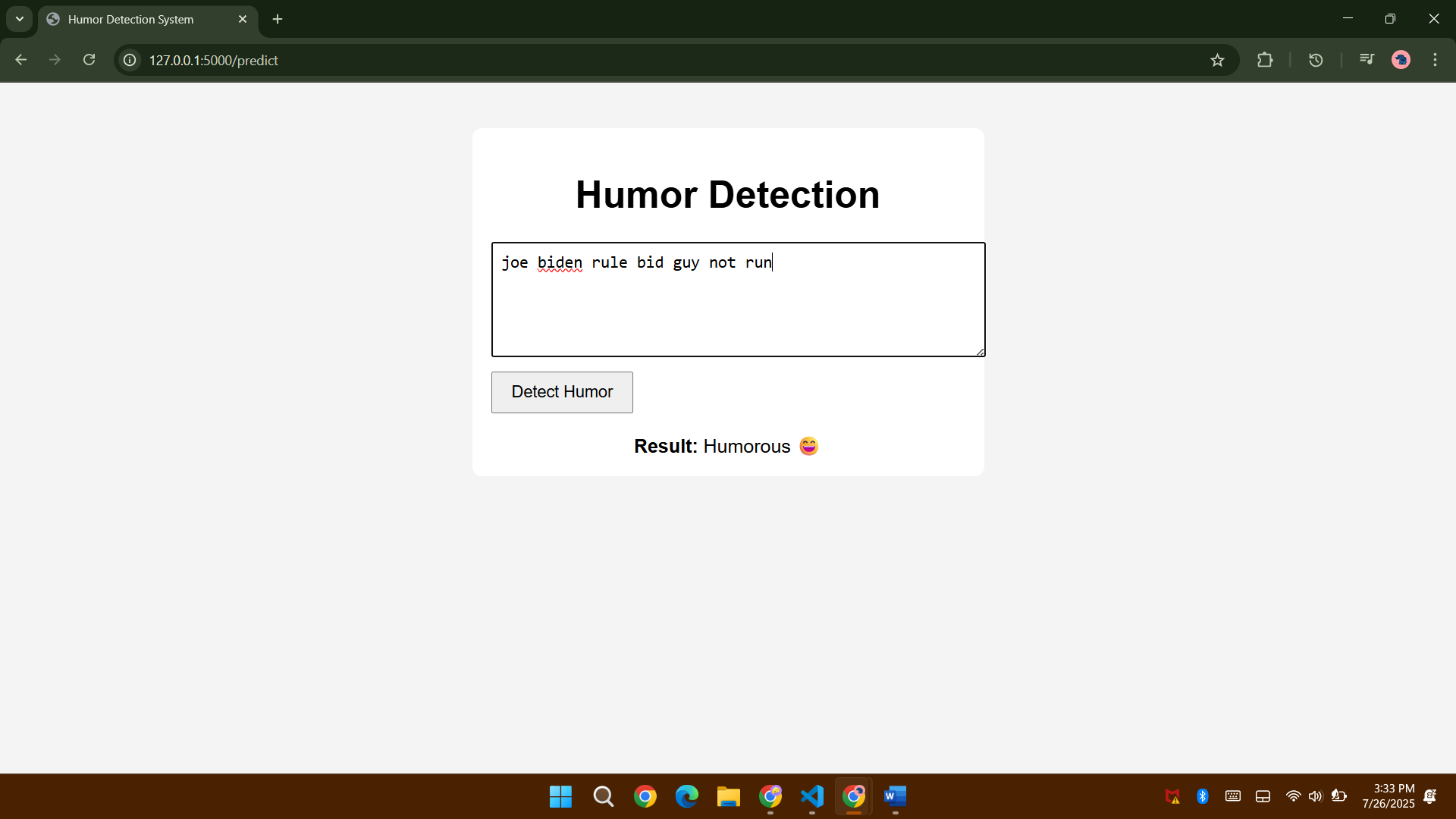
**6.2 User Experience**

* **Intuitive Interface**: Simple, clean design requiring no technical knowledge
* **Immediate Results**: Fast response times for user engagement
* **Clear Feedback**: Emoji-based visual indicators (😄 for humorous, 😐 for non-humorous)
* **Error Handling**: Graceful management of invalid inputs

**6.3 Technical Features**

* **RESTful API**: Standard HTTP methods for integration
* **JSON Communication**: Structured data exchange
* **Cross-Origin Support**: Configurable for different deployment scenarios
* **Logging**: Comprehensive request and error logging





**7. Testing and Quality Assurance**

**7.1 Testing Strategy**

* **Unit Testing**: Individual component validation (tests/test\_prediction.py)
* **Integration Testing**: End-to-end workflow verification
* **Performance Testing**: Response time and accuracy validation
* **User Acceptance Testing**: Interface usability evaluation

**7.2 Quality Metrics**

* **Code Coverage**: Comprehensive test coverage of critical functions
* **Performance Benchmarks**: Response time and throughput measurements
* **Accuracy Validation**: Consistent model performance across test sets
* **Error Rate Monitoring**: Tracking and analysis of prediction errors

**8. Deployment and Configuration**

**8.1 Local Deployment**

*# Method 1: Direct Flask execution*

cd app/backend

python app.py

*# Method 2: Using project entry point (Recommended)*

python run.py

*# Method 3: Module execution*

python -m run

**8.2 Environment Setup**

* **Python Version**: 3.7+ recommended
* **Dependencies**: Install via requirements.txt (implied)
* **Environment Variables**: Configure Flask settings as needed
* **Port Configuration**: Default port 5000, configurable

**8.3 Deployment Considerations**

* **PYTHONPATH Configuration**: Ensure proper module resolution
* **Model File Paths**: Verify correct relative paths for model loading
* **Static File Serving**: Configure for frontend asset delivery
* **Production Setup**: Consider WSGI server for production deployment

**9. Future Enhancements**

**9.1 Model Improvements**

* **BERT Integration**: Advanced transformer-based classification
* **Ensemble Methods**: Combining multiple algorithms for improved accuracy
* **Deep Learning**: Neural network architectures for complex pattern recognition
* **Active Learning**: Continuous model improvement with user feedback

**9.2 Feature Expansions**

* **Multi-language Support**: Humor detection across different languages
* **Confidence Scoring**: Probability estimates for predictions
* **Batch Processing**: API endpoints for multiple text analysis
* **Historical Analytics**: User interaction and prediction tracking

**9.3 Technical Enhancements**

* **API Authentication**: Secure access control for production use
* **Rate Limiting**: Request throttling for resource management
* **Caching**: Performance optimization for repeated queries
* **Containerization**: Docker support for simplified deployment

**10. Troubleshooting and Known Issues**

**10.1 Common Issues**

* **Import Errors**: PYTHONPATH configuration required for utils module
* **Model Loading**: Ensure correct path to pickled model file
* **Port Conflicts**: Default Flask port may require configuration
* **Static Files**: Frontend assets must be properly served

**10.2 Solutions**

* **Module Resolution**: Run from project root using python run.py
* **Path Management**: Use absolute paths or proper relative path configuration
* **Environment Setup**: Verify all dependencies are installed
* **Configuration**: Check Flask app configuration settings

**11. Conclusions**

The Humor Detection System successfully demonstrates the practical application of machine learning to natural language processing challenges. With strong performance metrics and a complete end-to-end implementation, the project showcases both technical competency and practical utility.