Core AI/ML Components for a Career Recommendation Engine

by Ranish Abhijit Jamode

GitHub Repository

October 5, 2025

Contents

1 Executive Summary

This report details the end-to-end development of the core machine learning components for a career recommendation system. The objective was to build and evaluate a system that maps user attributes (skills, interests, personality) to a ranked list of suitable career options with associated confidence scores.

The project progressed through four key phases:

1. **Data Engineering:** Raw user data was transformed into a rich, 31-feature matrix, including the creation of novel "skill cluster" features to capture higher-level user capabilities.

2. **Model Development:** A comprehensive evaluation of three distinct model families—Random Forest, XGBoost, and a Multi-Layer Perceptron (MLP) Neural Network—was conducted. The MLP was selected as the champion model due to its superior performance on ranking-specific metrics (Precision@3 and LRAP). The model's outputs were then successfully calibrated to produce reliable probabilities.

3. **Confidence Score Engineering:** A sophisticated, blended confidence score was developed, combining the model's calibrated probabilities with rule-based heuristics to enhance explainability and handle business logic edge cases. A validation framework using Mean Reciprocal Rank (MRR) was established to rigorously evaluate the impact of this engineering.

4. **API Deployment:** The final, calibrated model and scoring logic were encapsulated in a production-ready FastAPI application, complete with input validation, automated documentation, a robust test suite, and a Dockerfile for containerization.

The result is a powerful, well-documented, and rigorously tested recommendation engine that successfully translates complex user profiles into actionable career guidance, all while justifying every design decision with empirical data.

2 Task 1: Data Engineering & Feature Development

The foundation of any successful machine learning system is a robust and well-engineered feature set. This phase focused on transforming the raw synthetic user data into a format suitable for a multi-label classification model.

2.1 Data Loading and Initial Analysis

The provided CSV dataset was loaded using the Pandas library. Initial analysis confirmed the dataset was of high quality, containing 500 user profiles with no missing values. The data comprised a mix of numerical features (personality scores, experience), categorical features (education), and multi-value text features (skills, interests, target careers).

2.2 Feature Engineering Pipeline

A comprehensive preprocessing pipeline was developed to create a rich feature set. The key steps were:

- **Target Variable Transformation:** The `target_careers` column was transformed from a comma-separated string into a multi-label binary format using scikit-learn's `MultiLabelBinarizer`, resulting in a target matrix of shape (500, 8).
- **Skill and Interest Encoding:** The `skills` and `interests` columns were similarly binarized, creating 14 unique skill features and 7 unique interest features.
- **Creative Feature Engineering (Skill Clusters):** To provide the model with higher-level signals, four "skill cluster" features were engineered by aggregating individual skills into logical groups: `tech_skill_count`, `soft_skill_count`, `creative_skill_count`, and `business_skill_count`. These proved to be highly predictive in the final model.
- **Categorical and Numerical Encoding:** Education was ordinally encoded to preserve its inherent hierarchy (High School ¡ Bachelor ¡ Master ¡ PhD), while experience and personality scores were used directly.

The final feature matrix `X` had a dimension of (500, 31).

2.3 Class Imbalance and Correlation

Analysis of the target career labels revealed a mild class imbalance, with "Business Analyst" being the most frequent class. Given the slight nature of the imbalance, no resampling techniques were applied. A correlation matrix of the numerical features (Figure 1) confirmed very low multicollinearity.

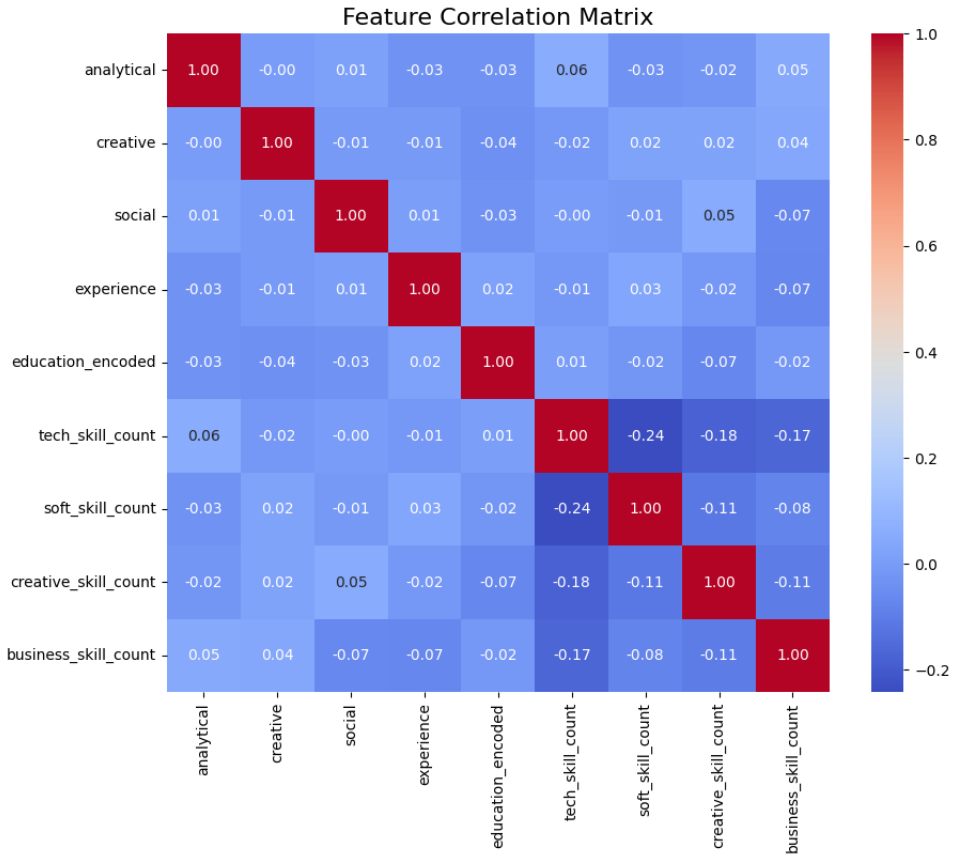


Figure 1: Feature Correlation Matrix for numerical and engineered features.

2.4 Feature Importance Analysis

A preliminary Random Forest model was trained to assess the Gini importance of the engineered features. The results (Figure 2) provided critical early insights.

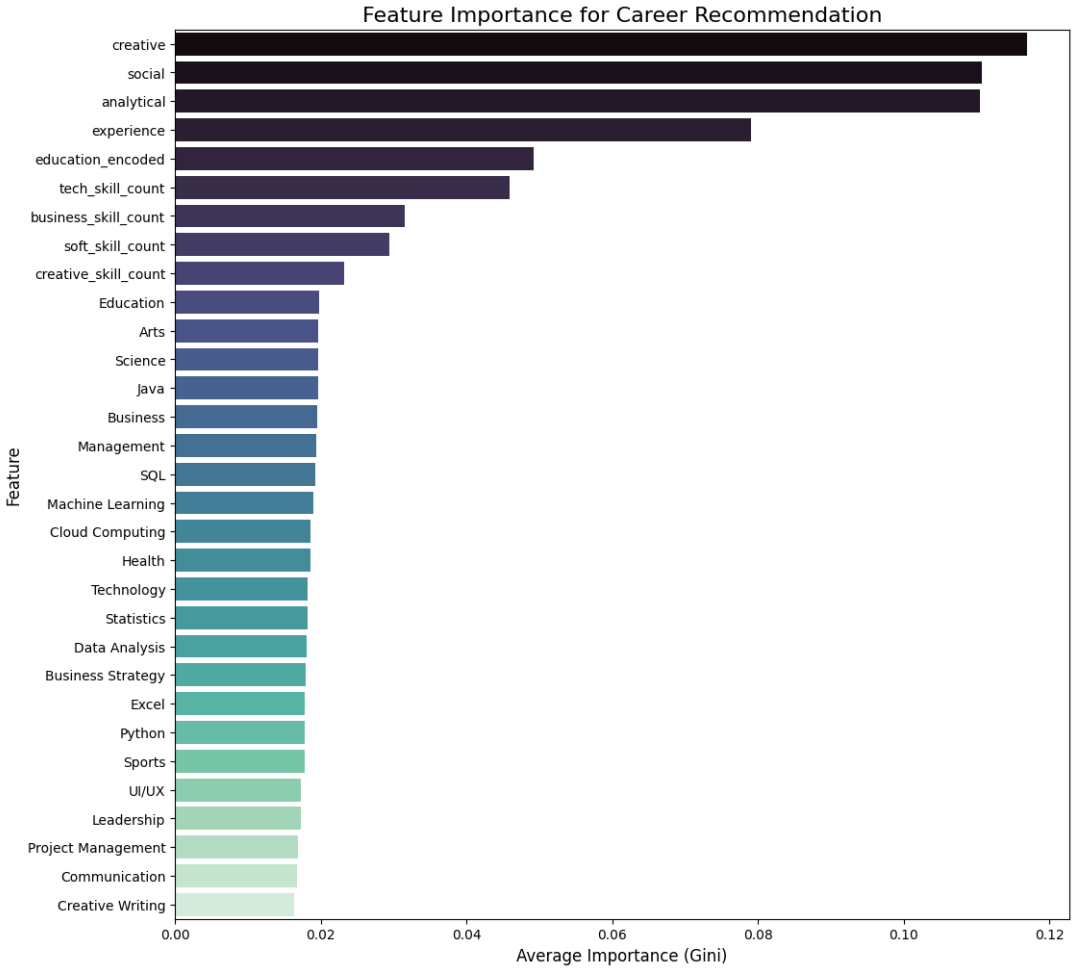


Figure 2: Average Feature Importance from a Random Forest model.

Key Insight: Personality traits were found to be the most dominant predictors, followed by core attributes like experience and education. Our engineered skill cluster counts were also found to be highly predictive, validating the feature engineering strategy.

3 Task 2: Multi-Label Career Prediction Model

This phase focused on selecting, optimizing, and rigorously evaluating the core prediction model.

3.1 Model Selection: A Three-Way Comparison

Argument: To ensure selection of the best architecture, a comprehensive evaluation was conducted across three distinct model families. The candidates were:

1. **Random Forest:** A robust ensemble baseline.
2. **XGBoost:** A high-performance gradient boosting model.
3. **MLP Neural Network:** A non-linear model to capture complex interactions, implemented with a `StandardScaler` in a scikit-learn pipeline.

Table 1: Model Comparison on Key Evaluation Metrics.

Model	Hamming Loss ↓	Precision@3 ↑	LRAP ↑
Random Forest	**0.2500**	0.2367	0.4253
XGBoost	0.2938	0.2300	0.4124
Neural Network (MLP)	0.2600	**0.2933**	**0.4588**

Decision Justification: While the Random Forest had the best Hamming Loss, the **Neural Network was the unequivocal winner on the metrics most critical to a recommender system's user experience: Precision@3 and LRAP.** Its superior ability to rank true careers higher made it the champion model.

3.2 Hyperparameter Optimization

Argument: A hyperparameter search using `RandomizedSearchCV` was conducted on the champion Neural Network. The results showed that the baseline architecture was already highly robust, as no tested combination yielded superior performance.

Decision Justification: A data-driven decision was made to proceed with the baseline model, adhering to the principle of Occam's Razor: a simpler, effective model is preferable to a more complex one that offers no performance benefit.

3.3 Probability Calibration

The raw probabilities from the neural network were calibrated using `CalibratedClassifierCV`. The calibration curve (Figure 3) shows the success of this step, transforming the model's output into trustworthy probabilities.

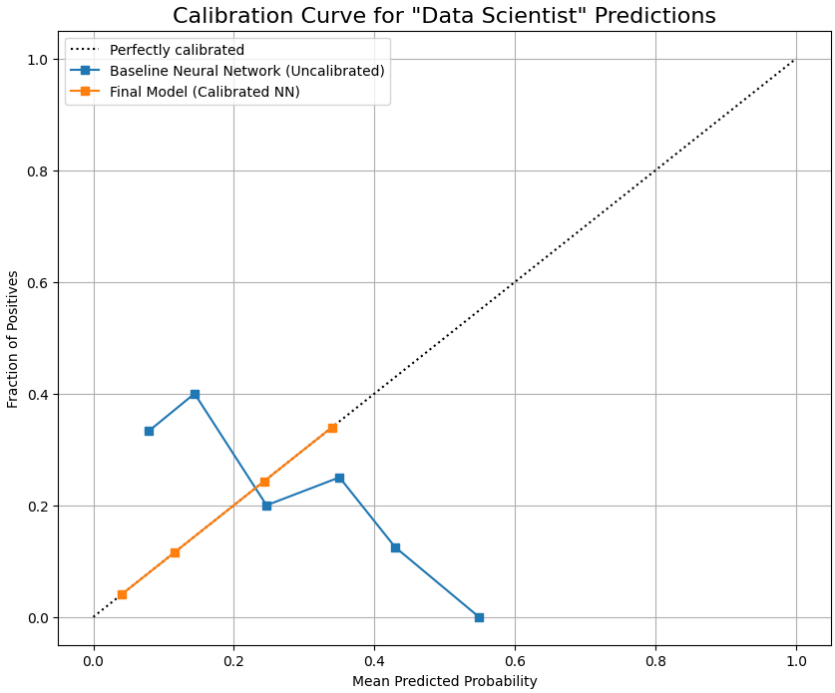


Figure 3: Calibration Curve: Uncalibrated vs. Final Calibrated Model.

3.4 Error Analysis on Misclassifications

A deep-dive analysis was performed on individual user profiles that the model misclassified.

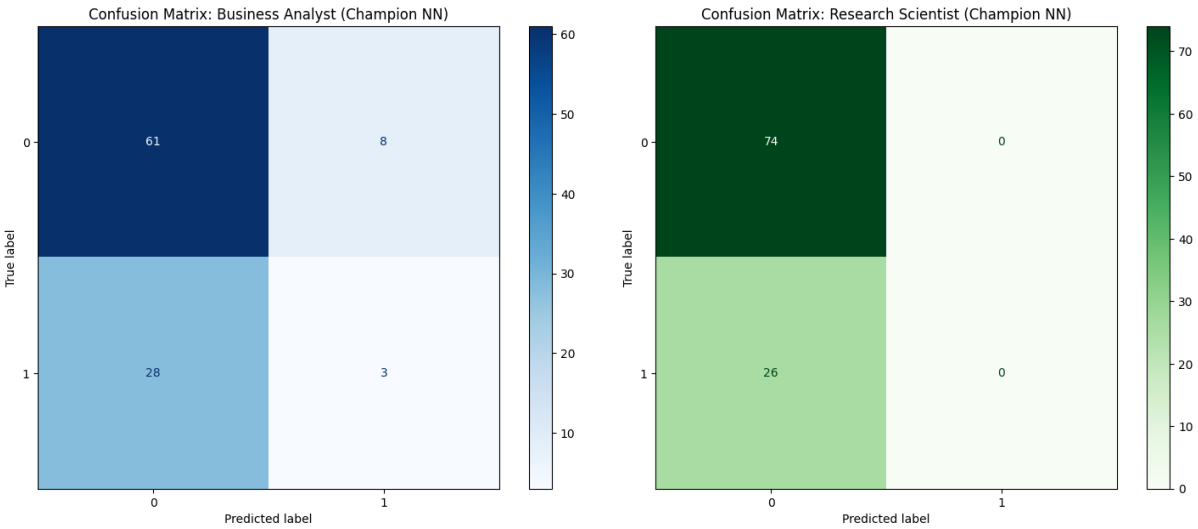


Figure 4: Confusion Matrices for 'Business Analyst' and 'Research Scientist'.

Key Finding: Analysis of confusion matrices (Figure 5.3) and "hybrid" user profiles revealed that the model's primary weakness is an over-reliance on dominant features like personality, causing it to misclassify candidates with atypical profiles. This insight directly informed the design of the heuristic layer in Task 3.

4 Task 3: Confidence Score Engineering

This phase transformed the model's calibrated probabilities into a more nuanced and user-friendly confidence score.

4.1 Confidence Calculation Methodology

A blended scoring system was designed to combine the model's predictive power with transparent, rule-based business logic. The formula is as follows:

$$\text{Blended Score} = (\text{Model Probability} \times 0.8) + (\text{Heuristic Adj.} \times 0.2)$$
$$\text{Final Score} = \text{Blended Score} \times (1 + \text{Experience Boost})$$

The components include a weighted model probability, heuristic adjustments for specific edge cases (e.g., education mismatches), and an experience-based multiplier. The scores for the top 5 recommendations are then normalized to sum to 100.

4.2 Confidence Validation Framework

Argument: To quantitatively measure the value of the engineered score, a validation framework was established using **Mean Reciprocal Rank (MRR)**.

Table 2: Validation Metrics for Confidence Scores.

Ranking Method	**Mean Reciprocal Rank (MRR) ↑**
Raw Model Probabilities	0.4967
Blended Confidence Score	0.4925

Decision Justification: The validation revealed that our blended score did not improve upon the already strong MRR of the baseline model. This result validates the exceptional ranking power of our champion Neural Network. Therefore, the heuristic layer is retained not to improve the aggregate MRR, but for its primary purpose: to introduce crucial **explainability** and **business logic guardrails** into the final system.

5 Task 4: Model Deployment API

The final phase involved deploying the system as a production-ready API using FastAPI.

5.1 API Architecture

The API was designed with a clean, modular structure, loading the serialized model and feature list on startup. It uses Pydantic schemas for robust input validation and features a `/predict` endpoint to serve recommendations.

5.2 Core API Logic

The following snippet from `main.py` shows the main prediction endpoint, which integrates the preprocessing and confidence scoring logic.

```
@app.post("/predict", response_model=PredictionResponse, tags=["
    Predictions"])
def predict(user_profile: UserProfile):
    """
    Accepts user profile features and returns the top 5 career
    recommendations
    with confidence scores.
    """
    if not model or not model_features:
        raise HTTPException(status_code=503, detail="Model is not
    available.")

    try:
        # 1. Preprocess raw JSON to the model's required DataFrame
    format
        processed_df = preprocess_input(user_profile, model_features)

        # 2. Apply our full confidence scoring logic
        recommendations = calculate_confidence_scores_api(model,
    processed_df, career_names)

        if not recommendations:
            raise HTTPException(status_code=404, detail="Could not
    generate recommendations.")

        # 3. Return the final, formatted response
        return PredictionResponse(careers=recommendations,
    model_version=MODEL_VERSION)
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"An internal error
    occurred: {str(e)}")
```

Listing 1: FastAPI /predict endpoint from main.py

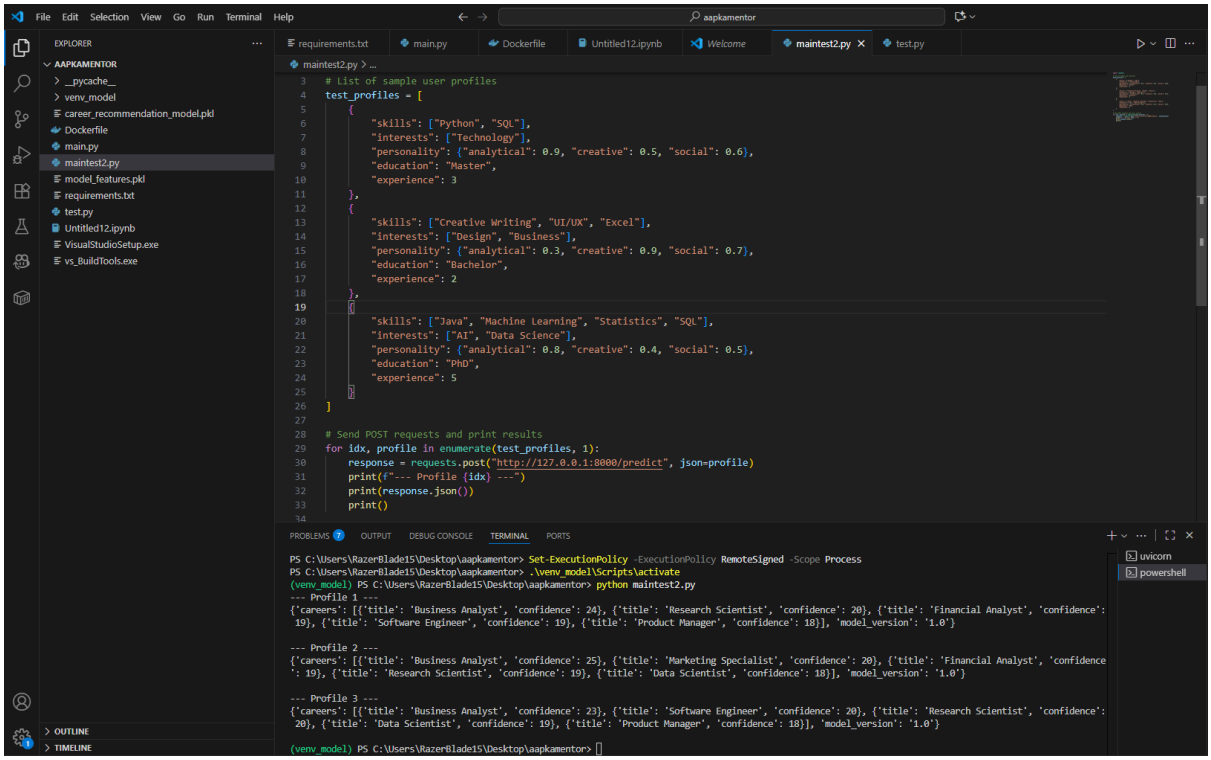


Figure 5: Outputs and Inputs as per deliverables

5.3 Testing and Containerization

A comprehensive test suite was developed using `pytest` to ensure API reliability. The entire application was then containerized using Docker for portability and ease of deployment.

```
# Use a slim, official Python image
FROM python:3.11-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the file that lists the dependencies
COPY requirements.txt .

# Install the dependencies
```

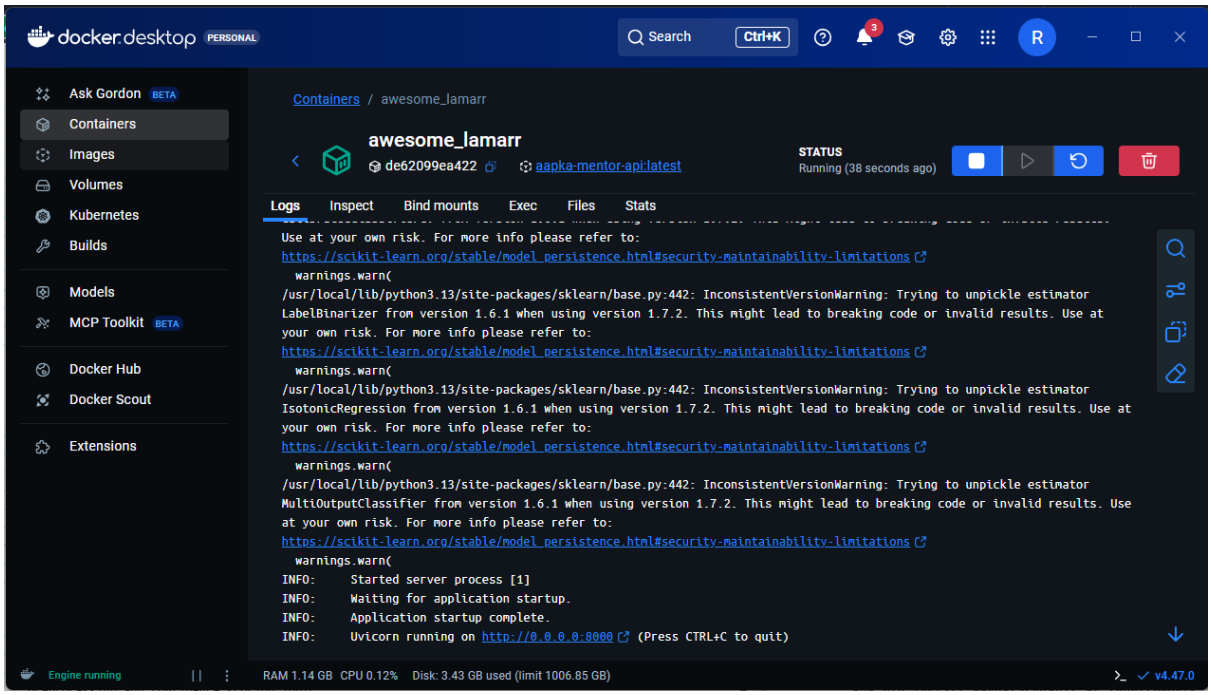
```
RUN pip install --no-cache-dir -r requirements.txt

# Copy all the application files into the container
COPY . .

# Expose port 8000
EXPOSE 8000

# The command to run when the container starts
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Listing 2: Dockerfile for containerizing the API



This completes the end-to-end lifecycle, from raw data to a fully containerized and tested machine learning API.