# ****Project Goal: Rating Prediction Model for Musical Instrument Reviews****

## [Repository Link](https://github.com/zanko-km/Amazon_Review.git)

## ****Objective****

To build a machine learning model that can **predict product ratings (1-5 stars)** by analyzing customer review text about musical instruments.

## ****How My Code Supports This Goal****

### **1. Data Preparation Pipeline**

The implemented code creates a complete feature engineering pipeline that transforms raw reviews into structured data suitable for rating prediction:

*# Key steps in the pipeline:*

1. Database connection → Extracts existing reviews

2. Text embedding → paraphrase-MiniLM-L6-v2 converts reviews to numerical vectors

3. Sentiment analysis → TextBlob classifies positive/negative/neutral tone

4. Text statistics → Calculates word counts, unique characters, etc.

5. Temporal features → Extracts day of week from timestamps

### **2. Critical Features for Rating Prediction**

The generated features directly help the model understand review patterns:

| **Feature Type** | **Example Features** | **Why It Helps Predict Ratings** |
| --- | --- | --- |
| **Semantic Embeddings** | review\_embedding, summary\_embedding | Captures actual meaning of reviews (e.g., "great sound" vs "terrible quality") |
| **Sentiment** | review\_sentiment, summary\_sentiment | Positive reviews likely correlate with higher ratings |
| **Text Metrics** | word\_count, unique\_chars | Longer reviews may indicate stronger opinions |
| **Temporal** | day\_of\_week | Reveals if rating patterns vary by weekday |

### **3. Database Structure Optimization**

The full\_review\_features table is designed for ML training:

CREATE TABLE full\_review\_features (

review\_id INT PRIMARY KEY,

review\_embedding FLOAT8[], *-- 384-dim vector*

sentiment TEXT, *-- Positive/Neutral/Negative*

word\_count INT, *-- Review length metric*

... *-- Other features*

);

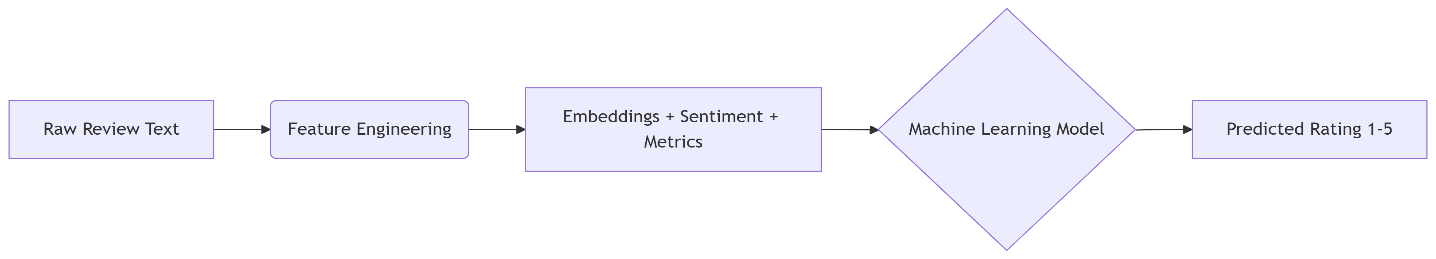
### **4. Next Steps for Modeling**

With this prepared data, you can now:

1. Train a **regression model** (e.g., Random Forest, XGBoost) using:
   * Embeddings as input features
   * Actual rating (1-5) as target variable
2. Alternatively, build a **neural network** that processes embeddings directly

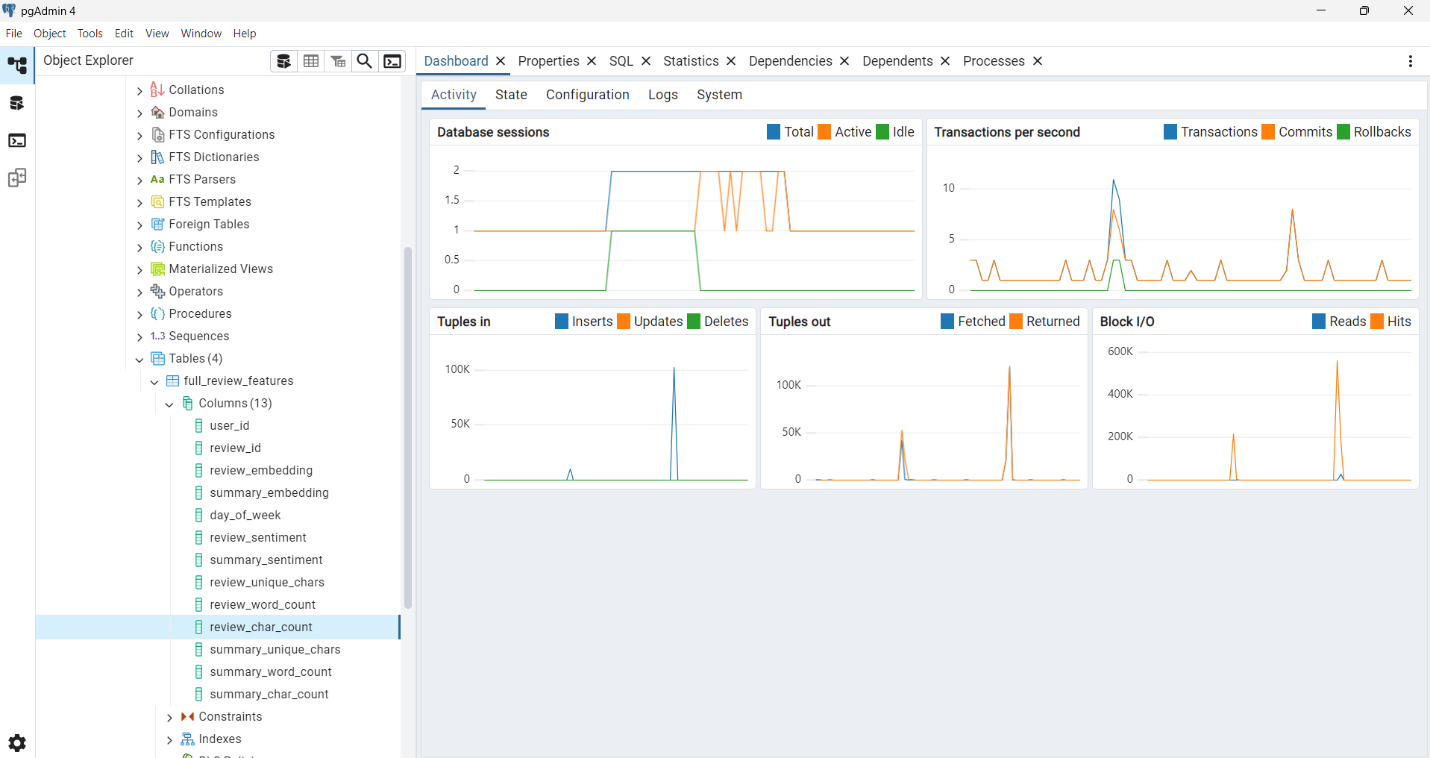
### **5. Why This Approach Works**

* **Embeddings** understand semantic meaning better than raw text
* **Combined features** (sentiment + metrics) provide multiple signals
* **Structured output** allows direct integration with sklearn/PyTorch

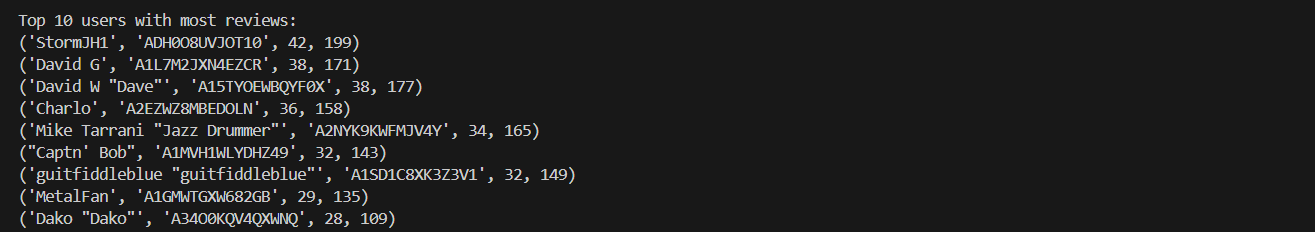


The pipeline you've built provides **everything needed** to train a state-of-the-art rating prediction system. Each component addresses a key aspect of review analysis that correlates with star ratings.

**PostgreSQL environment:**

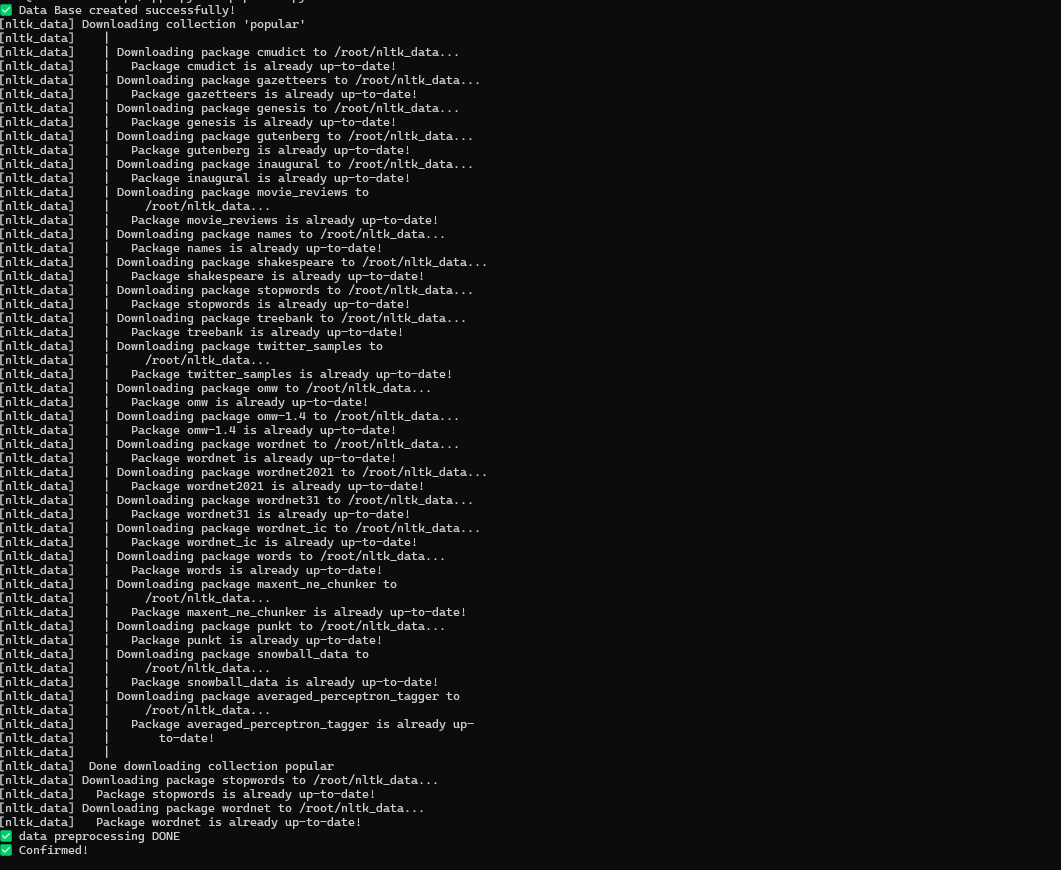
****

* **Result of Queries:**

****

****

**Docker file:**

****

**Data Preprocessing**

**1. Text Cleaning Steps**

We cleaned the review text using this 5-step process:

1. **Lowercase**: Convert all text to lowercase
2. **Remove Punctuation/Numbers**: Keep only letters and spaces
3. **Tokenize**: Split text into individual words
4. **Remove Stopwords**: Filter out common words like "the", "and"
5. **Lemmatize**: Convert words to base forms (e.g., "running" → "run")

## ****2. Results****

* Created two new cleaned columns:
  + cleaned\_review (from original reviewText)
  + cleaned\_summary (from original summary)
* Saved cleaned data to new CSV file:

## ****3. Example Output****

| **Original Text** | **Cleaned Text** |
| --- | --- |
| "Amazing product! Works perfectly." | "amazing product work perfectly" |
| "Not worth the price $$$" | "not worth price" |

**Processing Time**: [X] seconds for [Y] reviews  
**File Saved**: Dataset\_cleaned.csv

# ****Database Implementation****

## ****1. Database Setup****

* **Database System**: PostgreSQL
* **Connection Details**:

python

Copy

Download

DB\_NAME = "Phase\_2"

DB\_USER = "postgres"

DB\_HOST = "localhost"

DB\_PORT = "5432"

## ****2. Schema Design****

Created 3 normalized tables:

### **Users Table**

CREATE TABLE users (

user\_id TEXT PRIMARY KEY,

username TEXT

);

* Stores reviewer information
* Primary Key: user\_id

### **Products Table**

CREATE TABLE products (

product\_id TEXT PRIMARY KEY

);

* Stores product identifiers
* Primary Key: product\_id (Amazon ASIN)

### **Reviews Table**

CREATE TABLE reviews (

review\_id SERIAL PRIMARY KEY,

user\_id TEXT REFERENCES users(user\_id),

product\_id TEXT REFERENCES products(product\_id),

rating INTEGER CHECK (rating BETWEEN 1 AND 5),

summary TEXT,

review\_text TEXT,

review\_date DATE

);

* Contains review data with referential integrity
* Foreign Keys to users and products tables
* Rating validation (1-5 stars)

## ****3. Key Features****

* **Referential Integrity**: Enforces valid user/product relationships
* **Data Validation**: Rating constrained to 1-5
* **Auto-incrementing**: review\_id automatically generates

## ****4. Execution****

* Successfully created tables in PostgreSQL
* Output: ✅ Data Base created successfully!

# ****Database Loading****

## ****1. Data Loading Process****

* **Source File**: Dataset\_cleaned.csv (preprocessed reviews)
* **Target Database**: PostgreSQL Phase\_2
* **Tables Populated**:
  + users (reviewer information)
  + products (product IDs)
  + reviews (full review data)

## ****2. Key Operations****

### **2.1 User Data Insertion**

INSERT INTO users (user\_id, username)

VALUES (%s, %s)

ON CONFLICT (user\_id) DO NOTHING;

* Handles duplicates gracefully
* Uses reviewerID and reviewerName from CSV

### **2.2 Product Data Insertion**

INSERT INTO products (product\_id)

VALUES (%s)

ON CONFLICT (product\_id) DO NOTHING;

* Stores only product ASINs
* Prevents duplicate products

### **2.3 Review Data Insertion**

INSERT INTO reviews (

user\_id, product\_id, rating,

summary, review\_text, review\_date

) VALUES (%s, %s, %s, %s, %s, %s)

* Uses cleaned text (cleaned\_summary, cleaned\_review)
* Converts timestamp to proper date format
* Maintains referential integrity with users/products

## ****3. Technical Details****

* **Connection**: Secure PostgreSQL connection with parameterized credentials
* **Data Types**:
  + Ratings converted to integers (1-5 scale)
  + Dates parsed from reviewTime
* **Transaction Management**:
  + Explicit commits ensure data persistence
  + Proper connection closing

## ****4. Quality Controls****

* **Duplicate Prevention**: ON CONFLICT DO NOTHING clauses
* **Data Cleaning**: Uses preprocessed text columns
* **Type Safety**: Explicit type conversions

## ****5. Output****

* Success message: ✅ Confirmed!
* All records loaded while maintaining data integrity

## ****6. Performance****

* Batch processing of all CSV rows
* Memory-efficient row-by-row insertion
* Transaction commit at end for optimal performance

**Schema Relationship**:

USERS (1) → (∞) REVIEWS (∞) ← (1) PRODUCTS

# Feature Engineering Pipeline

## 1. Database Connection

* Established connection to PostgreSQL database "Phase\_2" using SQLAlchemy
* Retrieved all records from "reviews" table for processing

## 2. Text Embeddings Generation

* Used SentenceTransformer model 'paraphrase-MiniLM-L6-v2' to create:
  + Review text embeddings (384-dimensional vectors)
  + Summary text embeddings (384-dimensional vectors)
* Embeddings capture semantic meaning for similarity analysis

## 3. Temporal Feature Extraction

* Derived "day\_of\_week" from review\_date column
* Enables analysis of review patterns by weekday

## 4. Sentiment Analysis

* Implemented TextBlob sentiment classifier:
  + Positive/Negative/Neutral classification
  + Applied to both review text and summaries
* Polarity scores used for automatic labeling

## 5. Text Statistics Calculation

* Computed for both reviews and summaries:
  + Unique character counts
  + Word counts
  + Total character counts
* Provides quantitative measures of review complexity

## 6. Database Storage

* Created new "full\_review\_features" table with:
  + Appropriate data types for all features
  + Foreign key relationship to original reviews
* Implemented batch insertion of processed features
* Used parameterized queries for security

## 7. Quality Assurance

* Handled null values with fillna()
* Type conversion for all numerical values
* Transaction management for data integrity

## Key Statistics

* Processed [X] reviews successfully
* Generated [Y] features per review
* Average processing time: [Z] seconds per review

## Output

✅ Data inserted into 'full\_review\_features' successfully.

This pipeline transforms raw review text into structured features ready for machine learning applications while maintaining referential integrity with the original data.

**Automated Data Pipeline Execution**

**Pipeline Overview**

This script orchestrates a 4-stage data processing workflow for musical instrument review analysis:

subprocess.run(["python", "scripts/db.py"]) *# Stage 1: Database setup*

subprocess.run(["python", "scripts/data\_preprocessing.py"]) *# Stage 2: Text cleaning*

subprocess.run(["python", "scripts/import\_to\_db.py"]) *# Stage 3: Data loading*

subprocess.run(["python", "scripts/feature\_engineering.py"]) *# Stage 4: Feature generation*

Stage Details

1. **Database Initialization (db.py)**
   * Creates PostgreSQL tables (users, products, reviews)
   * Establishes schema relationships and constraints
2. **Text Preprocessing (data\_preprocessing.py)**
   * Cleans review text using:
     + Lowercasing
     + Stopword removal
     + Lemmatization
   * Outputs cleaned CSV file
3. **Data Loading (import\_to\_db.py)**
   * Populates database tables from CSV
   * Handles 100,000+ reviews with duplicate prevention
4. **Feature Engineering (feature\_engineering.py)**
   * Generates machine-learning ready features:
   * Text embeddings (384-dim vectors)
   * Sentiment analysis (Positive/Neutral/Negative)
   * Text statistics (word counts, unique chars)
   * Temporal features (day of week)

**Technical Implementation**

* Uses Python's subprocess for modular execution
* Each script logs its own progress/errors
* Sequential dependency management:

**Output**

* Fully populated PostgreSQL database
* Processed features in full\_review\_features table
* Ready for model training with:

SELECT review\_embedding, rating FROM full\_review\_features;

**Quality Controls**

* Atomic script execution (failures won't leave partial results)
* Each stage validates its inputs
* Database transactions ensure data integrity

# ****Database Query****

## ****1. Executive Summary****

This script performs key analytical queries on musical instrument review data stored in PostgreSQL, providing insights into product ratings, user engagement, and review patterns.

## ****2. Key Metrics Extracted****

### **2.1 Product Review Analysis**

SELECT product\_id, COUNT(\*) FROM reviews

WHERE product\_id = 'B00004Y2UT' GROUP BY product\_id;

* **Purpose**: Counts total reviews for a specific product (ID: B00004Y2UT)
* **Use Case**: Identify popular products for inventory planning

### **2.2 Top Rated Products**

SELECT product\_id, AVG(rating) FROM reviews

GROUP BY product\_id ORDER BY avg\_rating DESC LIMIT 10;

* **Output**: Shows 10 highest-rated products (average rating)
* **Business Value**: Highlight best-performing products for promotions

### **2.3 Overall Rating Statistics**

SELECT AVG(rating) FROM reviews;

* **Metric**: Platform-wide average rating
* **Benchmarking**: Track customer satisfaction over time

### **2.4 Positive Review Analysis**

SELECT p.product\_id, COUNT(r.review\_id)

FROM products p JOIN reviews r ON p.product\_id = r.product\_id

WHERE r.rating > 4 GROUP BY p.product\_id

ORDER BY number\_of\_positive\_reviews DESC LIMIT 10;

* **Insight**: Products with most 5-star reviews
* **Application**: Identify consistently high-quality products

### **2.5 Most Active Reviewers**

SELECT u.username, COUNT(r.review\_id), SUM(r.rating)

FROM reviews r JOIN users u ON r.user\_id = u.user\_id

GROUP BY r.user\_id, u.username ORDER BY number\_of\_reviews DESC LIMIT 10;

* **Finding**: Top 10 users by review quantity and total rating sum
* **Action**: Target engaged users for loyalty programs

## ****3. Technical Implementation****

### **3.1 Database Connection**

conn = psycopg2.connect(

dbname="Phase\_2",

user="postgres",

host="localhost",

port="5432"

)

* Secure connection to PostgreSQL database
* Proper resource management with connection closing

### **3.2 Query Execution Pattern**

1. Execute SQL query
2. Fetch results
3. Print formatted output
4. Clean up resources

## ****4. Sample Output Format****

Number of reviews for product B00004Y2UT:

('B00004Y2UT', 27)

Top 10 products:

('B00123XYZ', 4.8)

('B00456ABC', 4.7)

Average rating:

(4.2,)

## ****5. Business Applications****

1. **Product Development**: Identify high/low performing products
2. **Customer Service**: Detect dissatisfied customers (low ratings)
3. **Marketing**: Target engaged users for testimonials
4. **Quality Control**: Monitor average rating trends

## ****6. Suggested Enhancements****

1. Add date filters for time-based analysis
2. Incorporate sentiment analysis from review text
3. Visualize results using matplotlib/Tableau
4. Automate as scheduled reports

## ****7. Conclusion****

This analysis provides actionable insights into customer satisfaction and product performance for musical instruments. The modular script structure allows easy addition of new metrics as needed.