Dataset description

Dataset Title: Video Game Sales with Ratings
Source: [Kaggle – Video Game Sales with Ratings]
Description:
This dataset provides detailed information on video game sales across different regions, along with critical and user review scores, and metadata such as platform, genre, and ESRB rating. It is commonly used for exploratory data analysis, sales prediction, and understanding the influence of ratings on sales performance.

Features:

- Name: Title of the game
- Platform: Console on which the game runs
- Year_of_Release: Year the game was released
- Genre: Type/category of the game (e.g., Action, Role-Playing)
- Publisher: Company that published the game
- NA_Sales: Sales in North America (in millions of units)
- EU_Sales: Sales in the European Union (in millions)
- JP_Sales: Sales in Japan (in millions)
- Other_Sales: Sales in other regions including Africa, non-EU Europe, Australia, Asia (excluding Japan), and South America (in millions)
- Global_Sales: Total worldwide sales (in millions)
- Critic_Score: Average critic score (0–100 scale)
- Critic_Count: Number of critic reviews
- User_Score: Average user score (0–10 scale, stored as object due to non-numeric values)
- User_Count: Number of user reviews
- Developer: Company that developed the game
- Rating: ESRB content rating (e.g., E, T, M)

ML model training steps

Load & clean dataset

- Read CSV and strip column names
- Drop rows with missing values in key columns
- Remove regional sales columns (keep only Global_Sales)

Handle outliers

- Cap Global_Sales at the 99th percentile

Group rare categories

- Combine infrequent Publisher, Developer, and Platform values into "Other"

Encode categorical variables

- Frequency encode: Publisher, Developer
- Label encode: Platform, Genre, Rating, Year_of_Release

Prepare features and target

- Drop Name and Global_Sales from features
- Use log1p(Global_Sales) as the target (y)

Set up cross-validation

- Use 5-fold cross-validation with KFold

Train model

- Use RandomForestRegressor with 100 trees
- Train and predict on each fold

Evaluate performance

- Calculate MSE and R^2 for each fold
- Compute mean and standard deviation

Generate learning curve

- Visualize training vs validation R^2 scores

Save and return results

- Encode learning curve as base64 image
- Save summary stats and model metrics using pickle

How authentication was added

URL Configuration

- In gameSales/urls.py:
    - path('userLogin/', include('django.contrib.auth.urls')) includes Django's built-in login/logout views.
    - path('userLogin/', include('userLogin.urls')) routes to your custom login, logout, and signup views.

Routing in userLogin/urls.py

- /userLogin/ → login_user() for login.
- /userLogin/logout/ → logout_user() for logout.
- /userLogin/signup/ → signup() for account creation.

User Authentication Flow

Login (login_user)

- Accepts POST data: username and password.
- Authenticates using authenticate() and logs in the user with login().
- On success: redirects to 'index'.
- On failure: displays an error message and reloads the login page.

Logout (logout_user)

- Uses logout() to log the user out.
- Redirects to 'home' and displays a success message.

Signup (signup)

- Accepts username and password via POST.
- Checks if passwords match and if the username is unique.

- Creates a new user with User.objects.create_user().
- On success: redirects to login with a success message.

Securing Views

- In predictGlobalSales/views.py:
    - Views like index and about use @login_required(login_url='login') to restrict access to authenticated users only.

Steps for integration

Model Training

- Cleaned and preprocessed the dataset (Video_Games_Sales_as_at_22_Dec_2016.csv)
- Used label encoding and frequency encoding for categorical features
- Trained a RandomForestRegressor on log-transformed Global_Sales
- Saved the trained model and encoders using pickle:
    - game_sales_model.pkl → trained model
    - label_encoders.pkl → encoders used during training
    - freq_encoders.pkl → encoders used during training

Model Loading in Django

- In views.py, loaded both .pkl files at the top (so the model is ready when the app runs)

Prediction View

- Created a view to:
    - Receive form inputs from users
    - Encode inputs using saved encoders
    - Format the data and make predictions
    - Display the predicted global sales

HTML Form

- Built a simple form (index.html) to collect input from the user for prediction

Challenges encountered

1. Handling Missing Data

Some rows contained NaN values for critical fields like Developer, Publisher, Year_of_Release, and others.
Solution: Dropped rows with missing values for these required fields to ensure consistent input for model training.

2. High Cardinality of Categorical Features

Fields like Developer and Publisher had a large number of unique categories, which can cause issues with traditional label encoding.
Solution: Switched to frequency encoding to retain numerical relationships and reduce the risk of unseen category errors during inference.

3. Model Serialization Issues

Managing the .pkl files for both the trained model and encoders within Django's directory structure was initially tricky.
Solution: Standardized saving and loading paths by placing them in a dedicated ml_model/ folder and referencing them consistently in views.py.

4. Overfitting Detected from Learning Curve

The learning curve showed that the model performs significantly better on the training data than on the validation data, indicating signs of **overfitting**.
Solution (Planned): Consider using techniques like model regularization, more data cleaning