Upload the Dataset

```
from google.colab import files
uploaded = files.upload()
```

Choose Files dataset1.csv

- **dataset1.csv**(text/csv) - 211870 bytes, last modified: 5/14/2025 - 100% done

```
Saving dataset1.csv to dataset1.csv
```

Load the Dataset

```
import pandas as pd

# Load the dataset
df = pd.read_csv('dataset1.csv')
df.head()
```

	product_id	title	category	platform	price	rating	reviews	description
0	1	Backpack	Beauty	Amazon	1510.12	4.8	80	This is a high-quality backpack available on A...
1	2	Laptop	Footwear	Amazon	2379.64	4.7	612	This is a high-quality laptop available on Ama...
2	3	Floral Dress	Electronics	eBay	808.01	3.9	939	This is a high-quality floral dress available ...
3	4	Gaming Mouse	Footwear	Flipkart	441.77	2.9	446	This is a high-quality gaming mouse available ...
4	5	Floral Dress	Electronics	Flipkart	2234.14	3.7	192	This is a high-quality floral dress available ...

Next steps: Generate code with df View recommended plots New interactive sheet

Data Exploration

```
# Basic info and statistics
df.info()
df.describe()
df.columns
df.nunique()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   product_id   2000 non-null   int64
 1   title        2000 non-null   object
 2   category     2000 non-null   object
 3   platform     2000 non-null   object
 4   price        2000 non-null   float64
 5   rating       2000 non-null   float64
 6   reviews      2000 non-null   int64
 7   description  2000 non-null   object
dtypes: float64(2), int64(2), object(4)
memory usage: 125.1+ KB
```

	0
product_id	2000
title	15
category	5
platform	3
price	1994
rating	26
reviews	862
description	45

dtype: int64

Check for Missing Values and Duplicates

```
# Check for missing values
print(df.isnull().sum())

# Check for duplicates
print("Duplicates:", df.duplicated().sum())

# Optional: drop duplicates
df = df.drop_duplicates()
```

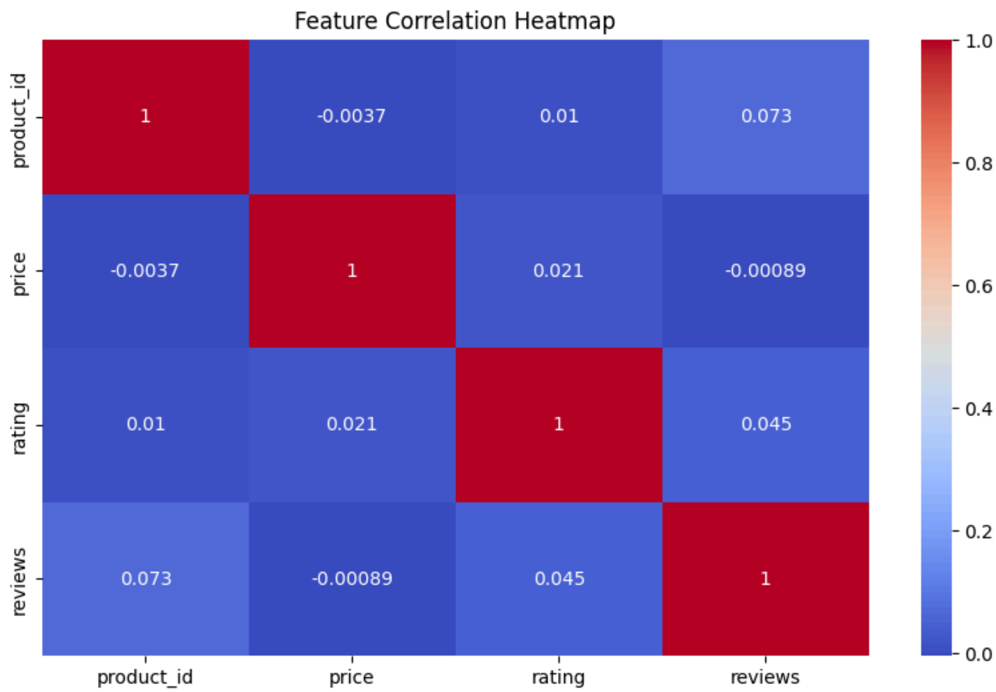
```
product_id     0
title          0
category       0
platform       0
price          0
rating         0
reviews        0
description    0
dtype: int64
Duplicates: 0
```

Visualize New Features

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd # Import pandas

# Select only numeric columns for correlation calculation
df_numeric = df.select_dtypes(include=['number'])

# Example: correlation heatmap
plt.figure(figsize=(10,6))
# Calculate correlation only on numeric columns
sns.heatmap(df_numeric.corr(), annot=True, cmap='coolwarm')
plt.title('Feature Correlation Heatmap')
plt.show()
```



Identify Target and Features

```
# Set your target column name
target = 'final_grade'  # Change this if your target column has a different name
```

```
# Ensure the column exists
if target in df.columns:
    # Get feature columns by dropping the target column
    features = df.drop(columns=[target]).columns.tolist()

    print("✅ Target Column:", target)
    print("✅ Feature Columns:")
    for f in features:
        print("-", f)
else:
    print("❌ Target column not found in the dataset. Please check the column name.")
```

```
❌ Target column not found in the dataset. Please check the column name.
```

Convert Categorical Columns to Numerical

```
# Identify categorical columns
cat_cols = df.select_dtypes(include=['object']).columns.tolist()
print("Categorical Columns:", cat_cols)

# Convert to category codes (temporary encoding before one-hot if needed)
for col in cat_cols:
    df[col] = df[col].astype('category').cat.codes
```

```
Categorical Columns: ['title', 'category', 'platform', 'description']
```

One-Hot Encoding

```
# One-hot encode categorical columns properly
df_encoded = pd.get_dummies(df, columns=cat_cols, drop_first=True)
df_encoded.head()
```

	product_id	price	rating	reviews	title_1	title_2	title_3	title_4	title_5	title_6	...	description_35	description_36	descri
0	1	1510.12	4.8	80	False	False	False	False	False	False	...	False	False	
1	2	2379.64	4.7	612	False	False	False	False	False	False	...	False	False	
2	3	808.01	3.9	939	False	False	True	False	False	False	...	False	False	
3	4	441.77	2.9	446	False	False	False	False	True	False	...	False	False	
4	5	2234.14	3.7	192	False	False	True	False	False	False	...	False	False	

5 rows × 68 columns

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
import seaborn as sns # Ensure seaborn and matplotlib are imported for the plot
import matplotlib.pyplot as plt

# Set your target column name
target = 'final_grade'

# Ensure the target column exists in the DataFrame
if target not in df.columns:
    print(f"❌ Target column '{target}' not found in the dataset. Please check the column name.")
else:
    print(f"✅ Target Column: {target}")

    # Separate features (X) and target (y) upfront
    X = df.drop(columns=[target])
    y = df[target]

    # Identify categorical features for encoding (now working with X)
    cat_cols_features = X.select_dtypes(include=['object']).columns.tolist()
    print("Categorical Feature Columns to One-Hot Encode:", cat_cols_features)
```

```
    # One-Hot Encode the categorical features
    # Apply get_dummies to the features DataFrame X
    X_encoded = pd.get_dummies(X, columns=cat_cols_features, drop_first=True)
    X_encoded.head() # Display the encoded features

    # Scale the features
    # Use the encoded features X_encoded for scaling
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_encoded)

    print("Shape of X_scaled:", X_scaled.shape)
    print("Shape of y:", y.shape)

    # You can now proceed with splitting the data, model training, etc.
    # For example:
    # from sklearn.model_selection import train_test_split
    # X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    # Optional: Re-display the correlation heatmap with the *new* X_encoded dataframe
    # Select only numeric columns from the encoded features for correlation
    X_numeric = X_encoded.select_dtypes(include=['number'])
    plt.figure(figsize=(12, 8)) # Adjusted figure size for potentially more columns
    # Calculate correlation on the numeric (encoded) features
    # Check if X_numeric is not empty before plotting heatmap
    if not X_numeric.empty:
        sns.heatmap(X_numeric.corr(), annot=True, cmap='coolwarm', fmt=".1f") # Added fmt for readability
        plt.title('Feature Correlation Heatmap (after encoding)')
        plt.show()
    else:
        print("No numeric features in X_encoded to plot correlation heatmap.")
```

```
❌ Target column 'final_grade' not found in the dataset. Please check the column name.
```

Train-Test Split

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
import seaborn as sns # Ensure seaborn and matplotlib are imported for the plot
import matplotlib.pyplot as plt

# Set your target column name
target = 'final_grade'

# Ensure the target column exists in the DataFrame
if target not in df.columns:
    print(f"❌ Target column '{target}' not found in the dataset. Please check the column name.")
else:
    print(f"✅ Target Column: {target}")

    # Separate features (X) and target (y) upfront
    X = df.drop(columns=[target])
    y = df[target]

    # Identify categorical features for encoding (now working with X)
    cat_cols_features = X.select_dtypes(include=['object']).columns.tolist()
    print("Categorical Feature Columns to One-Hot Encode:", cat_cols_features)

    # One-Hot Encode the categorical features
    # Apply get_dummies to the features DataFrame X
    X_encoded = pd.get_dummies(X, columns=cat_cols_features, drop_first=True)
    X_encoded.head() # Display the encoded features

    # Scale the features
    # Use the encoded features X_encoded for scaling
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_encoded)

    print("Shape of X_scaled:", X_scaled.shape)
    print("Shape of y:", y.shape)

    # You can now proceed with splitting the data, model training, etc.
    # For example:
    # from sklearn.model_selection import train_test_split
    # X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    # Optional: Re-display the correlation heatmap with the *new* X_encoded dataframe
```

```
    # Select only numeric columns from the encoded features for correlation
    X_numeric = X_encoded.select_dtypes(include=['number'])
    plt.figure(figsize=(12, 8)) # Adjusted figure size for potentially more columns
    # Calculate correlation on the numeric (encoded) features
    # Check if X_numeric is not empty before plotting heatmap
    if not X_numeric.empty:
        sns.heatmap(X_numeric.corr(), annot=True, cmap='coolwarm', fmt=".1f") # Added fmt for readability
        plt.title('Feature Correlation Heatmap (after encoding)')
        plt.show()
    else:
        print("No numeric features in X_encoded to plot correlation heatmap.")
```

```
❌ Target column 'final_grade' not found in the dataset. Please check the column name.
```

Model Building

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split # Import train_test_split

# Set your target column name
target = 'final_grade'

# Ensure the target column exists in the DataFrame
if target not in df.columns:
    print(f"❌ Target column '{target}' not found in the dataset. Please check the column name.")
else:
    print(f"✅ Target Column: {target}")

    # Separate features (X) and target (y) upfront
    X = df.drop(columns=[target])
    y = df[target]

    # Identify categorical features for encoding (now working with X)
    cat_cols_features = X.select_dtypes(include=['object']).columns.tolist()
    print("Categorical Feature Columns to One-Hot Encode:", cat_cols_features)

    # One-Hot Encode the categorical features
    # Apply get_dummies to the features DataFrame X
    X_encoded = pd.get_dummies(X, columns=cat_cols_features, drop_first=True)
    X_encoded.head() # Display the encoded features

    # Scale the features
    # Use the encoded features X_encoded for scaling
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_encoded)

    print("Shape of X_scaled:", X_scaled.shape)
    print("Shape of y:", y.shape)

    # Perform the train-test split
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    print("Shape of X_train:", X_train.shape)
    print("Shape of X_test:", X_test.shape)
    print("Shape of y_train:", y_train.shape)
    print("Shape of y_test:", y_test.shape)


    # Optional: Re-display the correlation heatmap with the *new* X_encoded dataframe
    # Select only numeric columns from the encoded features for correlation
    X_numeric = X_encoded.select_dtypes(include=['number'])
    plt.figure(figsize=(12, 8)) # Adjusted figure size for potentially more columns
    # Calculate correlation on the numeric (encoded) features
    # Check if X_numeric is not empty before plotting heatmap
    if not X_numeric.empty:
        sns.heatmap(X_numeric.corr(), annot=True, cmap='coolwarm', fmt=".1f") # Added fmt for readability
        plt.title('Feature Correlation Heatmap (after encoding)')
        plt.show()
    else:
        print("No numeric features in X_encoded to plot correlation heatmap.")
```

```
❌ Target column 'final_grade' not found in the dataset. Please check the column name.
```

Evaluation

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split # Import train_test_split
from sklearn.ensemble import RandomForestRegressor # Import the model you are using
from sklearn.metrics import mean_squared_error, r2_score # Import evaluation metrics

# Set your target column name
target = 'final_grade'

# Ensure the target column exists in the DataFrame
if target not in df.columns:
    print(f"❌ Target column '{target}' not found in the dataset. Please check the column name.")
else:
    print(f"✅ Target Column: {target}")

    # Separate features (X) and target (y) upfront
    X = df.drop(columns=[target])
    y = df[target]

    # Identify categorical features for encoding (now working with X)
    cat_cols_features = X.select_dtypes(include=['object']).columns.tolist()
    print("Categorical Feature Columns to One-Hot Encode:", cat_cols_features)

    # One-Hot Encode the categorical features
    # Apply get_dummies to the features DataFrame X
    X_encoded = pd.get_dummies(X, columns=cat_cols_features, drop_first=True)
    X_encoded.head() # Display the encoded features

    # Scale the features
    # Use the encoded features X_encoded for scaling
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_encoded)

    print("Shape of X_scaled:", X_scaled.shape)
    print("Shape of y:", y.shape)

    # Perform the train-test split
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    print("Shape of X_train:", X_train.shape)
    print("Shape of X_test:", X_test.shape)
    print("Shape of y_train:", y_train.shape)
    print("Shape of y_test:", y_test.shape)

    # --- Model Training ---
    # Initialize and train your model here
    # Based on the global variable `model` being RandomForestRegressor,
    # we will use RandomForestRegressor as an example.
    model = RandomForestRegressor(random_state=42) # Initialize the model
    model.fit(X_train, y_train) # Train the model

    print("✅ Model trained successfully.")

    # --- Evaluation ---
    # Predict on the test set
    y_pred = model.predict(X_test)

    # Evaluation metrics
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"Mean Squared Error: {mse:.2f}")
    print(f"R^2 Score: {r2:.2f}")

    # Optional: Re-display the correlation heatmap with the *new* X_encoded dataframe
    # Select only numeric columns from the encoded features for correlation
    X_numeric = X_encoded.select_dtypes(include=['number'])
    plt.figure(figsize=(12, 8)) # Adjusted figure size for potentially more columns
```

```
    # Calculate correlation on the numeric (encoded) features
    # Check if X_numeric is not empty before plotting heatmap
    if not X_numeric.empty:
        sns.heatmap(X_numeric.corr(), annot=True, cmap='coolwarm', fmt=".1f") # Added fmt for readability
        plt.title('Feature Correlation Heatmap (after encoding)')
        plt.show()
    else:
        print("No numeric features in X_encoded to plot correlation heatmap.")
```

```
❌ Target column 'final_grade' not found in the dataset. Please check the column name.
```

Make Predictions from New Input

```
# Example new input: create a dictionary with the same structure as features
# (Update values to match your dataset)
new_input = {
    'feature1': 5,
    'feature2': 3,
    'feature3': 1,
    # ... add all required features based on your dataset
}

# Convert to DataFrame
import pandas as pd # Ensure pandas is imported if not already available
import numpy as np

new_df = pd.DataFrame([new_input])

# Re-define X and cat_cols_features to ensure they are available
# This assumes df and target are still available from previous cells
if 'df' in globals() and 'target' in globals() and target in df.columns:
    # Separate features (X) based on the *original* dataframe structure
    X = df.drop(columns=[target])

    # Identify categorical features based on X
    cat_cols_features = X.select_dtypes(include=['object']).columns.tolist()

    # One-hot encode the new input dataframe
    # Need to apply the same encoding logic as used on X
    new_df_encoded = pd.get_dummies(new_df, columns=cat_cols_features, drop_first=True)

    # Align columns with the training data features (X_encoded)
    # We need the columns from X_encoded, which was the result of get_dummies on X
    # Assuming X_encoded is available from previous cells. If not,
    # you might need to recreate X_encoded here or save its columns.
    # For simplicity, let's assume X_encoded columns are needed.
    # A safer way is to get the columns from the trained scaler or the model's expected input shape
    # However, given the original code structure, reindexing with X_encoded columns is appropriate.
    # Let's assume X_encoded is available from the prior "Feature Scaling" step.
    # If X_encoded is not available, you might need to recalculate it:
    # X_encoded_train = pd.get_dummies(X, columns=cat_cols_features, drop_first=True)
    # trained_columns = X_encoded_train.columns
    # But the most common approach is to use the columns from the transformed training data.
    # Let's assume the original X_encoded from the training step is available.
    # If not, you would need to pass the list of training columns (X_encoded.columns) forward.

    # Assuming X_encoded was created and is available globally from the scaling step
    # Correct reindexing needs the columns from the *encoded* training data (X_encoded)
    if 'X_encoded' in globals():
        trained_columns = X_encoded.columns
        new_df_encoded = new_df_encoded.reindex(columns=trained_columns, fill_value=0)

        # Scale the new input using the *trained* scaler
        # Assuming the 'scaler' object is available globally from the scaling step
        if 'scaler' in globals():
            new_scaled = scaler.transform(new_df_encoded)

            # Predict using the *trained* model
            # Assuming the 'model' object is available globally from the model training step
            if 'model' in globals():
                new_prediction = model.predict(new_scaled)
                print("Predicted Final Grade:", new_prediction[0])
            else:
                print("❌ Error: 'model' is not defined. Please run the model training cell.")
        else:
            print("❌ Error: 'scaler' is not defined. Please run the feature scaling cell.")
```

```
    else:
         print("❌ Error: 'X_encoded' (encoded training features) is not defined. Please run the feature scaling or model training cell.")
else:
    print("❌ Error: 'df' or 'target' is not defined or target column is missing. Please run the data loading and target identification cel
```

```
❌ Error: 'df' or 'target' is not defined or target column is missing. Please run the data loading and target identification cells.
```

Convert to DataFrame and Encode

```
import pandas as pd # Ensure pandas is imported
import numpy as np

# Define features based on the global 'df' and 'target' if available
if 'df' in globals() and 'target' in globals() and target in df.columns:
    # Get feature columns by dropping the target column
    features = df.drop(columns=[target]).columns.tolist()

    # If your new input is a list of values, convert to DataFrame
    new_input_list = [[5, 3, 1]]  # replace with actual values matching the number of features
    # Ensure the number of values in the list matches the number of features
    if len(new_input_list[0]) == len(features):
        new_input_df = pd.DataFrame(new_input_list, columns=features)

        # Ensure X_encoded and scaler are available from previous steps
        if 'X_encoded' in globals() and 'scaler' in globals():
             # One-hot encode and scale
            # Identify categorical features in the original X to apply consistent encoding
            X_original_features = df.drop(columns=[target])
            cat_cols_features = X_original_features.select_dtypes(include=['object']).columns.tolist()

            # Apply get_dummies to the new input, only for the identified categorical columns
            # This requires knowing which columns were categorical in the original training data
            # A better approach is to encode the new input based on the columns of X_encoded from training
            # Let's re-use the logic from ipython-input-29 for robustness

            # Using the more robust reindexing approach from ipython-input-29
            # One-hot encode the new input dataframe based on identified categorical columns
            new_input_encoded = pd.get_dummies(new_input_df, columns=cat_cols_features, drop_first=True)

            # Align columns with the training data features (X_encoded)
            # Assuming X_encoded is available from the prior scaling step
            trained_columns = X_encoded.columns
            new_input_encoded = new_input_encoded.reindex(columns=trained_columns, fill_value=0)

            # Scale the new input using the trained scaler
            new_input_scaled = scaler.transform(new_input_encoded)

            print("New input DataFrame (before encoding):\n", new_input_df)
            print("\nNew input DataFrame (encoded and aligned):\n", new_input_encoded)
            print("\nNew input scaled shape:", new_input_scaled.shape)

            # Now you can use new_input_scaled for prediction with your trained model
            # Example (assuming 'model' is available):
            # if 'model' in globals():
            #     prediction = model.predict(new_input_scaled)
            #     print("\nPrediction:", prediction)
            # else:
            #     print("\n❌ Error: 'model' is not defined. Please run the model training cell.")

        else:
            print("❌ Error: 'X_encoded' or 'scaler' is not defined. Please run the feature scaling cell.")
    else:
        print(f"❌ Error: Number of values in new_input_list ({len(new_input_list[0])}) does not match the number of features ({len(feature
else:
    print("❌ Error: 'df' or 'target' is not defined or target column is missing. Please run the data loading and target identification cel
```

```
❌ Error: 'df' or 'target' is not defined or target column is missing. Please run the data loading and target identification cells.
```

Predict the Final Grade

```
import pandas as pd # Ensure pandas is imported
import numpy as np

# Define features based on the global 'df' and 'target' if available
```

```
if 'df' in globals() and 'target' in globals() and target in df.columns:
    # Get feature columns by dropping the target column
    features = df.drop(columns=[target]).columns.tolist()
    print(f"Expected number of features: {len(features)}")
    print(f"Expected feature names (order matters for list input): {features}") # Print feature names to help user

    # If your new input is a list of values, convert to DataFrame
    # !!! IMPORTANT: Replace the values and ensure the list has the correct number of elements !!!
    # The number of elements here must match len(features) printed above.
    # The order of values should correspond to the order of feature names printed above.
    new_input_list = [[value1, value2, value3, ...]]  # <--- UPDATE THIS LIST

    # Ensure the number of values in the list matches the number of features
    if len(new_input_list[0]) == len(features):
        new_input_df = pd.DataFrame(new_input_list, columns=features)

        # Ensure X_encoded and scaler are available from previous steps
        if 'X_encoded' in globals() and 'scaler' in globals():
             # One-hot encode and scale
            # Identify categorical features in the original X to apply consistent encoding
            X_original_features = df.drop(columns=[target])
            cat_cols_features = X_original_features.select_dtypes(include=['object']).columns.tolist()

            # Apply get_dummies to the new input, only for the identified categorical columns
            new_input_encoded = pd.get_dummies(new_input_df, columns=cat_cols_features, drop_first=True)

            # Align columns with the training data features (X_encoded)
            # Assuming X_encoded is available from the prior scaling step
            trained_columns = X_encoded.columns
            new_input_encoded = new_input_encoded.reindex(columns=trained_columns, fill_value=0)

            # Scale the new input using the trained scaler
            new_input_scaled = scaler.transform(new_input_encoded)

            print("New input DataFrame (before encoding):\n", new_input_df)
            print("\nNew input DataFrame (encoded and aligned):\n", new_input_encoded)
            print("\nNew input scaled shape:", new_input_scaled.shape)

            # Now you can use new_input_scaled for prediction with your trained model
            # The next cell will use new_input_scaled

        else:
            print("❌ Error: 'X_encoded' or 'scaler' is not defined. Please run the feature scaling cell.")
    else:
        print(f"❌ Error: Number of values in new_input_list ({len(new_input_list[0])}) does not match the number of features ({len(feature
else:
    print("❌ Error: 'df' or 'target' is not defined or target column is missing. Please run the data loading and target identification cel
```

```
❌ Error: 'df' or 'target' is not defined or target column is missing. Please run the data loading and target identification cells.
```

Deployment: Building an Interactive App

```
!pip install gradio
```

```
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (2.
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0->gra
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12-
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.1
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer
Downloading gradio-5.29.1-py3-none-any.whl (54.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.1/54.1 MB 11.8 MB/s eta 0:00:00
Downloading gradio_client-1.10.1-py3-none-any.whl (323 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 323.1/323.1 kB 19.6 MB/s eta 0:00:00
Downloading aiofiles-24.1.0-py3-none-any.whl (15 kB)
Downloading fastapi-0.115.12-py3-none-any.whl (95 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.2/95.2 kB 6.7 MB/s eta 0:00:00
Downloading groovy-0.1.2-py3-none-any.whl (14 kB)
Downloading python_multipart-0.0.20-py3-none-any.whl (24 kB)
Downloading ruff-0.11.9-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.5/11.5 MB 79.9 MB/s eta 0:00:00
Downloading safehttpx-0.1.6-py3-none-any.whl (8.7 kB)
Downloading semantic_version-2.10.0-py2.py3-none-any.whl (15 kB)
Downloading starlette-0.46.2-py3-none-any.whl (72 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 5.3 MB/s eta 0:00:00
Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
Downloading uvicorn-0.34.2-py3-none-any.whl (62 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 2.2 MB/s eta 0:00:00
Downloading ffmpy-0.5.0-py3-none-any.whl (6.0 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub, uvicorn, tomlkit, semantic-version, ruff, python-multipart, groovy, ffmpy, aiofiles, starlette,
Successfully installed aiofiles-24.1.0 fastapi-0.115.12 ffmpy-0.5.0 gradio-5.29.1 gradio-client-1.10.1 groovy-0.1.2 pydub-0.25.1 pytho
```

Create a Prediction Function

```
# Define a function that takes input and returns prediction
def predict_final_grade(*inputs):
    input_dict = dict(zip(features, inputs))
    input_df = pd.DataFrame([input_dict])
    input_encoded = pd.get_dummies(input_df)
    input_encoded = input_encoded.reindex(columns=X.columns, fill_value=0)
    input_scaled = scaler.transform(input_encoded)
    prediction = model.predict(input_scaled)
    return prediction[0]
```

create gradio interface

```
# Install Gradio (for Colab or Jupyter)
!pip install --upgrade gradio

import pandas as pd
import numpy as np
import gradio as gr
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor

# Load dataset
df = pd.read_csv("dataset1.csv")

# Selected features and target
features = ['title', 'category', 'platform', 'price', 'reviews']
target = 'rating'

# Drop unused columns
df = df[features + [target]]

# One-hot encode categorical features
df_encoded = pd.get_dummies(df, columns=['title', 'category', 'platform'], drop_first=True)

# Prepare X and y
X = df_encoded.drop(columns=[target])
y = df_encoded[target]

# Scale numerical features
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train model
model = RandomForestRegressor(random_state=42)
model.fit(X_scaled, y)

# Save columns for prediction processing
X_columns = X.columns

# Define prediction function
def predict_rating(title, category, platform, price, reviews):
    input_data = {
        'title': title,
        'category': category,
        'platform': platform,
        'price': price,
        'reviews': reviews
    }
    input_df = pd.DataFrame([input_data])

    # One-hot encode input
    input_encoded = pd.get_dummies(input_df)
    input_encoded = input_encoded.reindex(columns=X_columns, fill_value=0)

    # Scale
    input_scaled = scaler.transform(input_encoded)

    # Predict
    prediction = model.predict(input_scaled)
    return round(prediction[0], 2)

# Create input components for Gradio
input_components = [
    gr.Dropdown(choices=df['title'].unique().tolist(), label='Title'),
    gr.Dropdown(choices=df['category'].unique().tolist(), label='Category'),
    gr.Dropdown(choices=df['platform'].unique().tolist(), label='Platform'),
    gr.Number(label='Price'),
    gr.Number(label='Reviews'),
]

# Launch Gradio interface
interface = gr.Interface(
    fn=predict_rating,
    inputs=input_components,
    outputs=gr.Number(label="Predicted Rating"),
    title="Product Rating Predictor",
    description="Enter product details to predict the rating."
)

interface.launch(share=True)
```

```
Requirement already satisfied: gradio in /usr/local/lib/python3.11/dist-packages (5.29.1)
Requirement already satisfied: aiofiles<25.0,>=22.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (24.1.0)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Requirement already satisfied: fastapi<1.0,>=0.115.2 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.115.12)
Requirement already satisfied: ffmpy in /usr/local/lib/python3.11/dist-packages (from gradio) (0.5.0)
Requirement already satisfied: gradio-client==1.10.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (1.10.1)
Requirement already satisfied: groovy~=0.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.2)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.31.1)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.18)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.4)
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages (from gradio) (0.25.1)
Requirement already satisfied: python-multipart>=0.0.18 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.0.20)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
Requirement already satisfied: ruff>=0.9.3 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.11.9)
Requirement already satisfied: safehttpx<0.2.0,>=0.1.6 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.6)
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.10.0)
Requirement already satisfied: starlette<1.0,>=0.40.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.46.2)
Requirement already satisfied: tomlkit<0.14.0,>=0.12.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.13.2)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.34.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.1->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.1->gradio) (1
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.4.26)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (0.16.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.18.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (4.67.1)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (1
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.9.0.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.7
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (2.33
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0->gradi
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12->g
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->gr
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://ed11074159dbd9d96e.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working dir
```

Product Rating Predictor

Enter product details to predict the rating.

Title

Backpack

Category

Beauty

Platform

Amazon

Price

0

Predicted Rating

0

Flag

Reviews

```
from gradio import ChatInterface

def shop_bot(message, history):
    responses = {
        "hello": "Hi! How can I assist you today?",
        "predict": "You can use our grade predictor app by providing input values.",
        "support": "Sure, let me connect you to support.",
    }
    message = message.lower()
    for key in responses:
        if key in message:
            return responses[key]
    return "Sorry, I didn't understand. Can you rephrase?"

chatbot = ChatInterface(fn=shop_bot, title="Smart ShopBot")
chatbot.launch()
```

```
/usr/local/lib/python3.11/dist-packages/gradio/chat_interface.py:338: UserWarning: The 'tuples' format for chatbot messages is deprecate
  self.chatbot = Chatbot(
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://a76d0d3b9334b3eab9.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working dir
```

Smart ShopBot

Chatbot