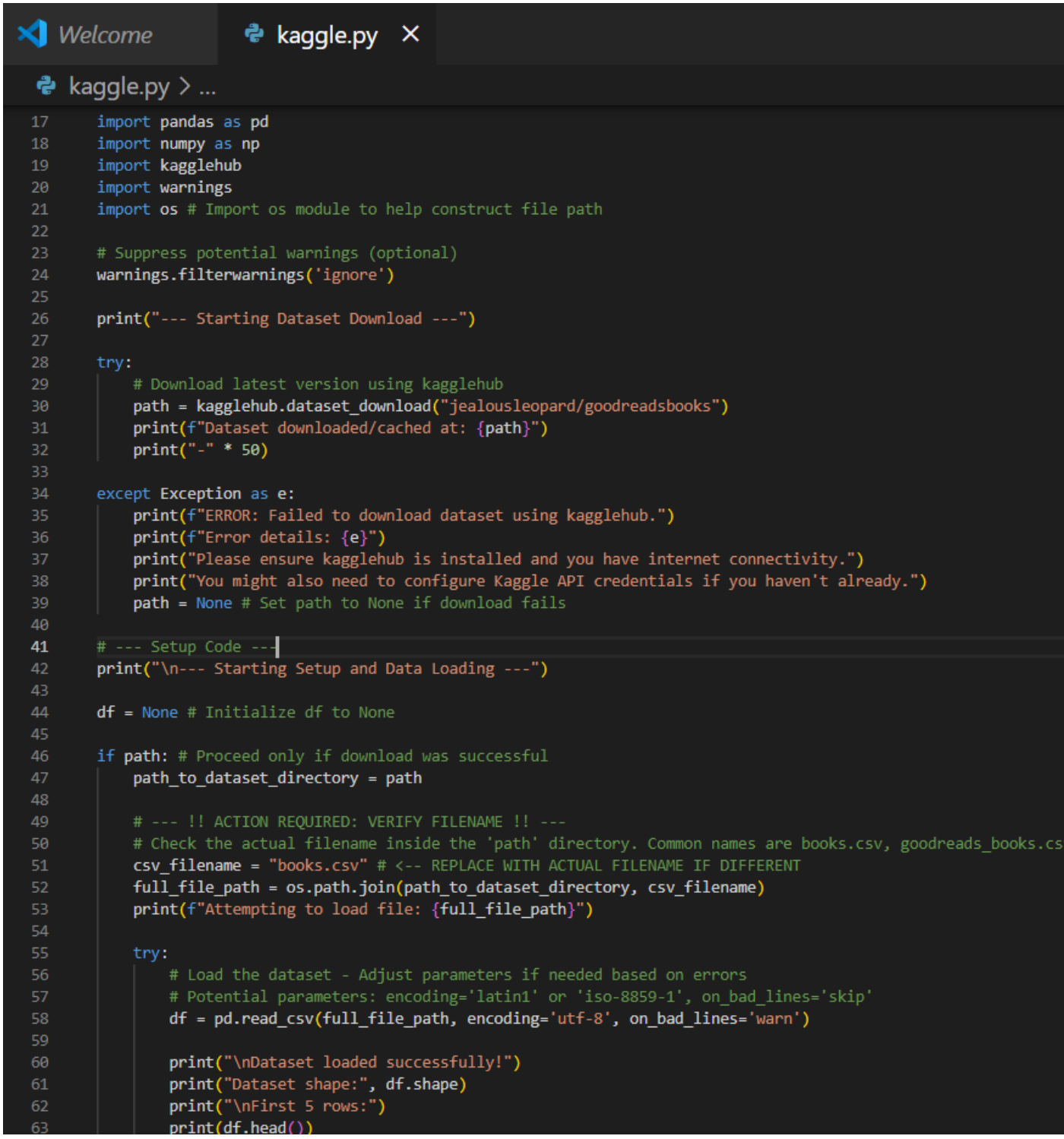
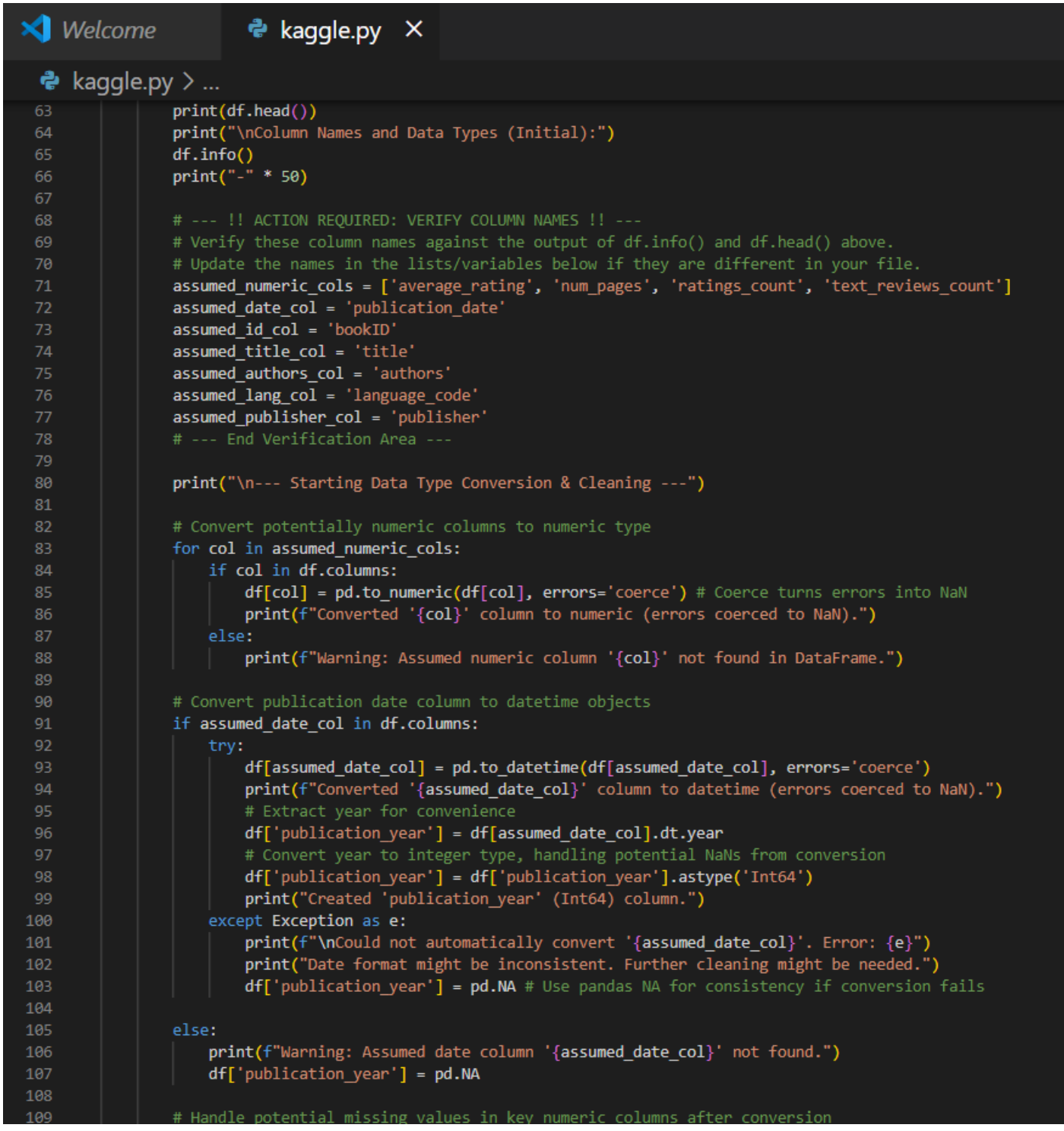
EDS Assignment

Name : Paritosh Pandita
PRN : 202401080005
Roll No : CS7-29
Division and Batch : CS7 & CS72


Welcome
kaggle.py
kaggle.py > ...
import pandas as pd
import numpy as np
import kagglehub
import warnings
import os # Import os module to help construct file path
# Suppress potential warnings (optional)
warnings.filterwarnings('ignore')
print("--- Starting Dataset Download ---")
try:
# Download latest version using kagglehub
path = kagglehub.dataset_download("jealousleopard/goodreadsbooks")
print(f"Dataset downloaded/cached at: {path}")
print("-" * 50)
except Exception as e:
print(f"ERROR: Failed to download dataset using kagglehub.")
print(f"Error details: {e}")
print("Please ensure kagglehub is installed and you have internet connectivity.")
print("You might also need to configure Kaggle API credentials if you haven't already.")
path = None # Set path to None if download fails
# --- Setup Code ---
print("\n--- Starting Setup and Data Loading ---")
df = None # Initialize df to None
if path: # Proceed only if download was successful
path_to_dataset_directory = path
# --- !! ACTION REQUIRED: VERIFY FILENAME !! ---
# Check the actual filename inside the 'path' directory. Common names are books.csv, goodreads_books.cs
csv_filename = "books.csv" # <-- REPLACE WITH ACTUAL FILENAME IF DIFFERENT
full_file_path = os.path.join(path_to_dataset_directory, csv_filename)
print(f"Attempting to load file: {full_file_path}")
try:
# Load the dataset - Adjust parameters if needed based on errors
# Potential parameters: encoding='latin1' or 'iso-8859-1', on_bad_lines='skip'
df = pd.read_csv(full_file_path, encoding='utf-8', on_bad_lines='warn')
print("\nDataset loaded successfully!")
print("Dataset shape:", df.shape)
print("\nFirst 5 rows:")
print(df.head())


Welcome
kaggle.py
kaggle.py > ...
print(df.head())
print("\nColumn Names and Data Types (Initial):")
df.info()
print("-" * 50)
# --- !! ACTION REQUIRED: VERIFY COLUMN NAMES !! ---
# Verify these column names against the output of df.info() and df.head() above.
# Update the names in the lists/variables below if they are different in your file.
assumed_numeric_cols = ['average_rating', 'num_pages', 'ratings_count', 'text_reviews_count']
assumed_date_col = 'publication_date'
assumed_id_col = 'bookID'
assumed_title_col = 'title'
assumed_authors_col = 'authors'
assumed_lang_col = 'language_code'
assumed_publisher_col = 'publisher'
# --- End Verification Area ---
print("\n--- Starting Data Type Conversion & Cleaning ---")
# Convert potentially numeric columns to numeric type
for col in assumed_numeric_cols:
if col in df.columns:
df[col] = pd.to_numeric(df[col], errors='coerce') # Coerce turns errors into NaN
print(f"Converted '{col}' column to numeric (errors coerced to NaN).")
else:
print(f"Warning: Assumed numeric column '{col}' not found in DataFrame.")
# Convert publication date column to datetime objects
if assumed_date_col in df.columns:
try:
df[assumed_date_col] = pd.to_datetime(df[assumed_date_col], errors='coerce')
print(f"Converted '{assumed_date_col}' column to datetime (errors coerced to NaN).")
# Extract year for convenience
df['publication_year'] = df[assumed_date_col].dt.year
Convert year to integer type, handling potential NaNs from conversion
df['publication_year'] = df['publication_year'].astype('Int64')
print("Created 'publication_year' (Int64) column.")
except Exception as e:
print(f"\nCould not automatically convert '{assumed_date_col}'. Error: {e}")
print("Date format might be inconsistent. Further cleaning might be needed.")
df['publication_year'] = pd.NA # Use pandas NA for consistency if conversion fails
else:
print(f"Warning: Assumed date column '{assumed_date_col}' not found.")
df['publication_year'] = pd.NA
Handle potential missing values in key numeric columns after conversion

kaggle.py > ...

```
                print("Date format might be inconsistent. Further cleaning might be needed.")
                df['publication_year'] = pd.NA # Use pandas NA for consistency if conversion fails

        else:
            print(f"Warning: Assumed date column '{assumed_date_col}' not found.")
            df['publication_year'] = pd.NA

        # Handle potential missing values in key numeric columns after conversion
        # Strategy: Remove rows where essential numeric columns are NaN
        cols_to_check_na = ['average_rating', 'num_pages', 'ratings_count'] # Choose essential cols
        initial_rows = len(df)
        columns_found_to_check = [col for col in cols_to_check_na if col in df.columns]

        if columns_found_to_check:
            df.dropna(subset=columns_found_to_check, inplace=True)
            rows_removed = initial_rows - len(df)
            if rows_removed > 0:
                print(f"\nRemoved {rows_removed} rows with missing values in key columns: {', '.join(columns_found_to_check)}")
        else:
            print("\nSkipping NaN removal for key numeric columns as none were found.")

        print("\nCleaned Data Info:")
        df.info()
        print("\n--- Setup Complete ---")


    except FileNotFoundError:
        print(f"ERROR: File not found at '{full_file_path}'.")
        print(f"Please check the directory '{path_to_dataset_directory}' and verify the filename '{csv_file_name}'.")
        df = None # Ensure df is None if loading fails

    except Exception as e:
        print(f"\nAn error occurred during data loading or initial processing: {e}")
        print("Consider trying different 'encoding' or 'on_bad_lines' parameters in pd.read_csv.")
        df = None # Ensure df is None if loading fails

else:
    print("Dataset download failed. Cannot proceed with analysis.")


# --- Problem Solving Section ---
# Proceed only if df was loaded and processed successfully
if df is not None:
    print("\n" + "="*60)
    print("--- Starting Problem Solving ---")
    print("="*60 + "\n")

```

Problem 1: How many unique books are listed?
Solution : Total number of unique books (using 'bookID'): 11123

```
# === Problem 1 ===
print("\n--- Problem 1: How many unique books are listed? ---")
if assumed_id_col in df.columns:
    unique_books = df[assumed_id_col].nunique()
    print(f"Total number of unique books (using '{assumed_id_col}'): {unique_books}")
elif assumed_title_col in df.columns and assumed_authors_col in df.columns:
    unique_books = df.groupby([assumed_title_col, assumed_authors_col]).ngroups
    print(f"Approximate number of unique books (using '{assumed_title_col}' + '{assumed_authors_col}'): {unique_books}")
else:
    print(f"Total number of rows (best estimate without unique ID): {len(df)}")
```

Problem 2: What is the overall average 'average_rating'?

Solution : Overall average rating: 3.93

```
# === Problem 2 ===
print(f"\n--- Problem 2: What is the overall average '{assumed_numeric_cols[0]}'? ---") # a
if assumed_numeric_cols[0] in df.columns:
    overall_avg_rating = df[assumed_numeric_cols[0]].mean()
    print(f"Overall average rating: {overall_avg_rating:.2f}")
else: print(f"Column '{assumed_numeric_cols[0]}' not found.")
```

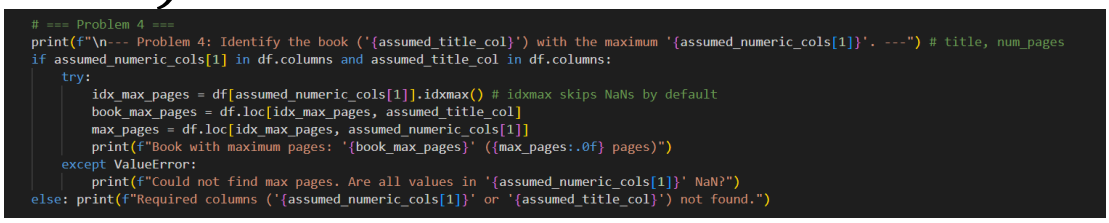
Problem 3: What is the average 'num_pages'?

Solution : Column 'num_pages' not found.

```
# === Problem 3 ===
print(f"\n--- Problem 3: What is the average '{assumed_numeric_cols[1]}'? ---") # num_pages
if assumed_numeric_cols[1] in df.columns:
    avg_pages = df[assumed_numeric_cols[1]].mean()
    print(f"Average number of pages: {avg_pages:.0f}")
else: print(f"Column '{assumed_numeric_cols[1]}' not found.")
```

Problem 4: Identify the book ('title') with the maximum 'num_pages'. ---

Required columns ('num_pages' or 'title') not found.



```
# === Problem 4 ===
print(f"\n--- Problem 4: Identify the book ('{assumed_title_col}') with the maximum '{assumed_numeric_cols[1]}'. ---") # title, num_pages
if assumed_numeric_cols[1] in df.columns and assumed_title_col in df.columns:
    try:
        idx_max_pages = df[assumed_numeric_cols[1]].idxmax() # idxmax skips NaNs by default
        book_max_pages = df.loc[idx_max_pages, assumed_title_col]
        max_pages = df.loc[idx_max_pages, assumed_numeric_cols[1]]
        print(f"Book with maximum pages: '{book_max_pages}' ({max_pages:.0f} pages)")
    except ValueError:
        print(f"Could not find max pages. Are all values in '{assumed_numeric_cols[1]}' NaN?")
else: print(f"Required columns ('{assumed_numeric_cols[1]}' or '{assumed_title_col}') not found.")
```

Problem 5: Top 5 most frequent 'language_code's and their counts. ---

Top 5 language codes:
language_code
eng   8908
en-US   1408
spa   218
en-GB   214
fre   144
Name: count, dtype: int64

```
# === Problem 5 ===
print(f"\n--- Problem 5: Top 5 most frequent '{assumed_lang_col}'s and their counts. ---") 
if assumed_lang_col in df.columns:
    top_languages = df[assumed_lang_col].value_counts().head(5)
    print("Top 5 language codes:")
    print(top_languages)
else: print(f"Column '{assumed_lang_col}' not found.")
```

Problem 6: How many books listed with primary 'language_code' 'eng'? ---

Number of books with language code 'eng': 8908

```
# === Problem 6 ===
print(f"\n--- Problem 6: How many books listed with primary '{assumed_lang_col}'
if assumed_lang_col in df.columns:
    # Adjust the check based on actual language code format (e.g., 'eng', 'en-US
    target_lang_code = 'eng'
    eng_books_count = df[df[assumed_lang_col] == target_lang_code].shape[0]
    print(f"Number of books with language code '{target_lang_code}': {eng_books_
else: print(f"Column '{assumed_lang_col}' not found.")
```

Problem 7: Top 10 'authors' strings by number of books. ---

Top 10 'authors' strings (might include multiple authors per entry):
authors
Stephen King 40
P.G. Wodehouse 40
Rumiko Takahashi 39
Orson Scott Card 35
Agatha Christie 33
Piers Anthony 30
Mercedes Lackey 29
Sandra Brown 29
Dick Francis 28
Laurell K. Hamilton 23
Name: count, dtype: int64

```
# === Problem 7 ===
print(f"\n--- Problem 7: Top 10 '{assumed_authors_col}' strings by number of books. ---") #
if assumed_authors_col in df.columns:
    top_authors = df[assumed_authors_col].value_counts().head(10)
    print(f"Top 10 '{assumed_authors_col}' strings (might include multiple authors per entry
    print(top_authors)
else: print(f"Column '{assumed_authors_col}' not found.")
```

Problem 8: Average 'average_rating' for books by 'J.K. Rowling'. ---
Average rating for books containing author 'J.K. Rowling': 4.50

```
# === Problem 8 ===
print(f"\n--- Problem 8: Average '{assumed_numeric_cols[0]}' for books by 'J.K. R
if assumed_authors_col in df.columns and assumed_numeric_cols[0] in df.columns:
    author_name = 'J.K. Rowling' # Case-sensitive, adjust if needed
    author_books = df[df[assumed_authors_col].str.contains(author_name, na=False,
    if not author_books.empty:
        avg_rating_author = author_books[assumed_numeric_cols[0]].mean()
        print(f"Average rating for books containing author '{author_name}': {avg_
    else:
        print(f"No books found containing author name '{author_name}'.")
else: print(f"Required columns ('{assumed_authors_col}' or '{assumed_numeric_cols
```

--- Problem 9: Total number of ratings ('ratings_count') across all books. ---
Total ratings received across all books: 199,578,299

```
# === Problem 9 ===
print(f"\n--- Problem 9: Total number of ratings ('{assumed_numeric_cols[2]}') ac
if assumed_numeric_cols[2] in df.columns:
    total_ratings = df[assumed_numeric_cols[2]].sum()
    print(f"Total ratings received across all books: {total_ratings:,.0f}")
else: print(f"Column '{assumed_numeric_cols[2]}' not found.")
```

--- Problem 10: Correlation between 'average_rating' and 'num_pages'. ---
Required columns ('average_rating' or 'num_pages') not found.

```
# === Problem 10 ===
print(f"\n--- Problem 10: Correlation between '{assumed_numeric_cols[0]}' and '{ass
if assumed_numeric_cols[0] in df.columns and assumed_numeric_cols[1] in df.columns:
    correlation_rating_pages = df[assumed_numeric_cols[0]].corr(df[assumed_numeric_
    print(f"Correlation between '{assumed_numeric_cols[0]}' and '{assumed_numeric_c
else: print(f"Required columns ('{assumed_numeric_cols[0]}' or '{assumed_numeric_co
```

--- Problem 11: Book ('title') with the highest 'ratings_count'. ---
Book with the most ratings: 'Twilight (Twilight #1)' (4,597,666 ratings)

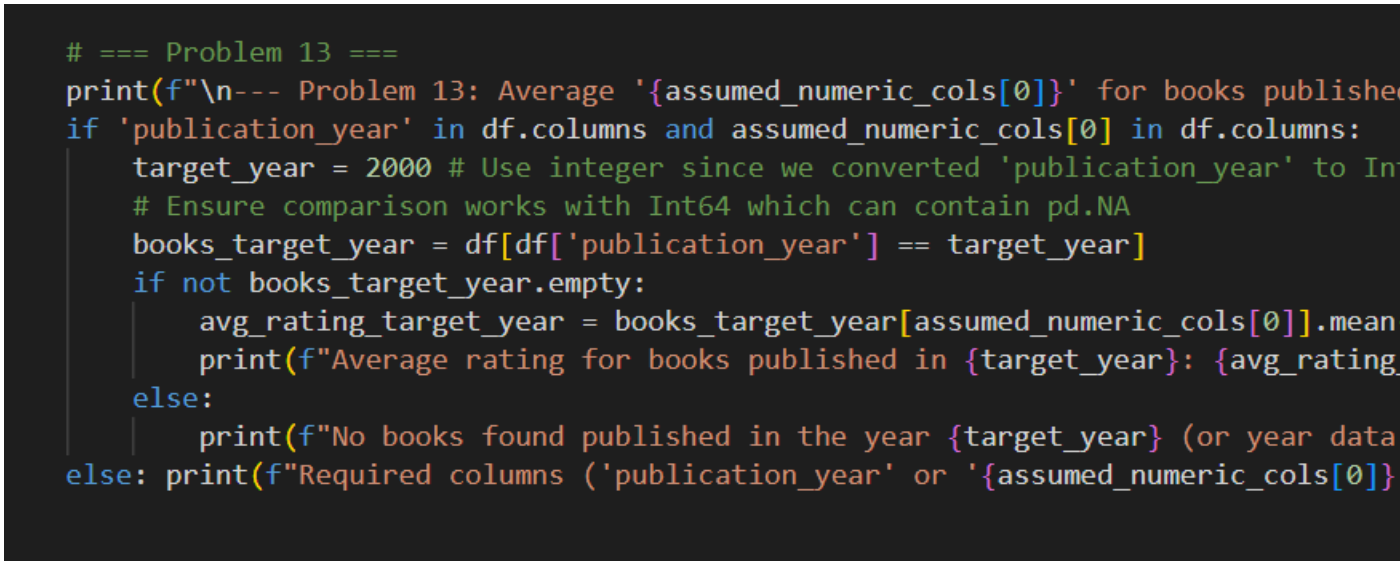
```
# === Problem 11 ===
print(f"\n--- Problem 11: Book ('{assumed_title_col}') with the highest '{assumed
if assumed_numeric_cols[2] in df.columns and assumed_title_col in df.columns:
    try:
        idx_max_ratings = df[assumed_numeric_cols[2]].idxmax()
        book_most_ratings = df.loc[idx_max_ratings, assumed_title_col]
        max_ratings_count = df.loc[idx_max_ratings, assumed_numeric_cols[2]]
        print(f"Book with the most ratings: '{book_most_ratings}' ({max_ratings_c
    except ValueError:
        print(f"Could not find max ratings count. Are all values in '{assumed_num
else: print(f"Required columns ('{assumed_numeric_cols[2]}' or '{assumed_title_co
```

--- Problem 12: Top 10 years by number of books published ('publication_year').

Number of books published per year (Top 10 by count):
publication_year
2006 1700
2005 1260
2004 1069
2003 931
2002 798
2001 656
2000 533
2007 518
1999 450
1998 396
Name: count, dtype: Int64

```
# === Problem 12 ===
print("\n--- Problem 12: Top 10 years by number of books published ('publication_year'). ---")
if 'publication_year' in df.columns and not df['publication_year'].isnull().all():
    books_per_year = df['publication_year'].dropna().value_counts()
    print("Number of books published per year (Top 10 by count):")
    print(books_per_year.head(10))
else: print("'publication_year' column not available or empty (check setup).")
```

--- Problem 13: Average 'average_rating' for books published in 2000. ---
Average rating for books published in 2000: 3.92



```
# === Problem 13 ===
print(f"\n--- Problem 13: Average '{assumed_numeric_cols[0]}' for books publishe
if 'publication_year' in df.columns and assumed_numeric_cols[0] in df.columns:
    target_year = 2000 # Use integer since we converted 'publication_year' to In
    # Ensure comparison works with Int64 which can contain pd.NA
    books_target_year = df[df['publication_year'] == target_year]
    if not books_target_year.empty:
        avg_rating_target_year = books_target_year[assumed_numeric_cols[0]].mean
        print(f"Average rating for books published in {target_year}: {avg_rating
    else:
        print(f"No books found published in the year {target_year} (or year data
else: print(f"Required columns ('publication_year' or '{assumed_numeric_cols[0]}
```

--- Problem 14: Categorize books by 'average_rating' and count categories. ---
Added 'rating_category' column.
Number of books per rating category:
rating_category
Low (<3.5) 733
Medium [3.5-4.0) 5436
High [>=4.0) 4954

Name: count, dtype: int64

```python
# === Problem 14 ===
print(f"\n--- Problem 14: Categorize books by '{assumed_numeric_cols[0]}' and c
if assumed_numeric_cols[0] in df.columns:
    bins = [0, 3.5, 4.0, df[assumed_numeric_cols[0]].max() + 0.1] # Dynamic upp
    labels = ['Low (<3.5)', 'Medium [3.5-4.0)', 'High [>=4.0)']
    df['rating_category'] = pd.cut(df[assumed_numeric_cols[0]], bins=bins, labe
    category_counts = df['rating_category'].value_counts().sort_index()
    print("Added 'rating_category' column.")
    print("Number of books per rating category:")
    print(category_counts)
else: print(f"Column '{assumed_numeric_cols[0]}' not found.")
```

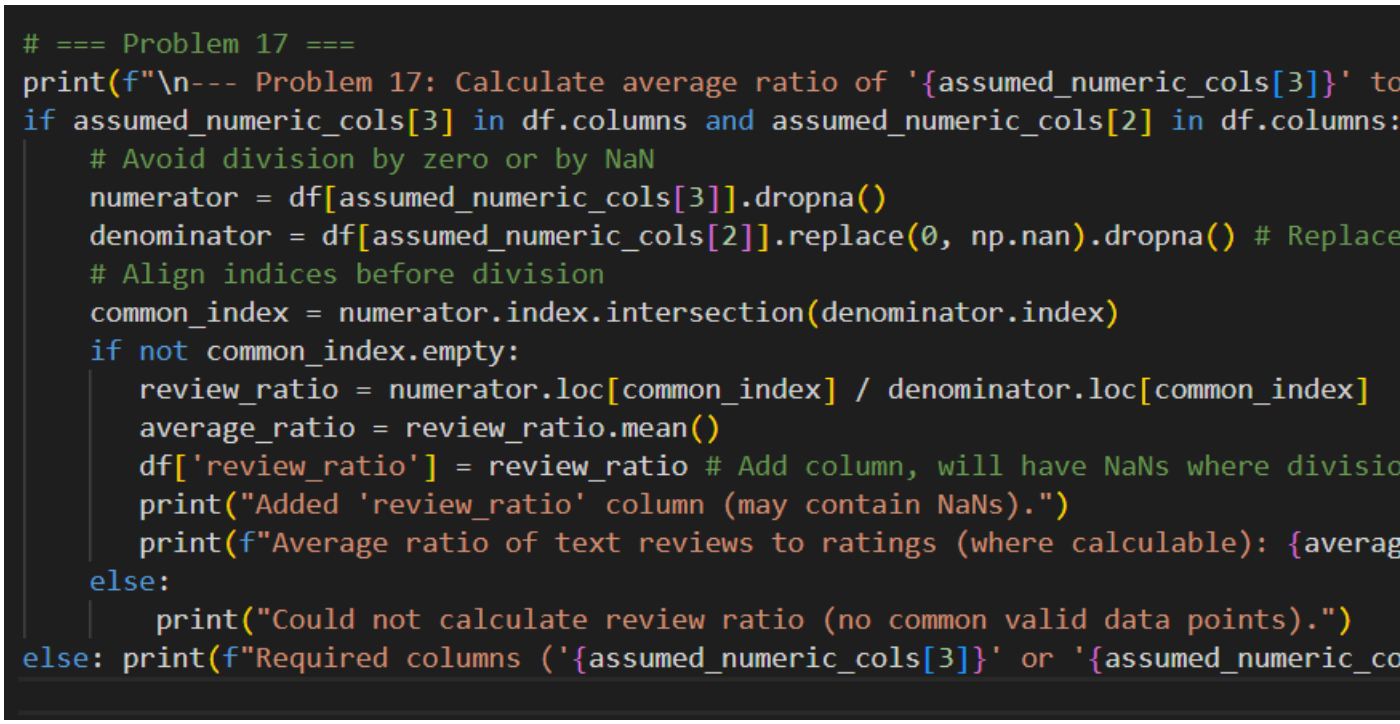
--- Problem 15: Top 5 'publisher's by number of books. ---
Top 5 publishers by number of books:
publisher
Vintage 318
Penguin Books 261
Penguin Classics 184
Mariner Books 150
Ballantine Books 144
Name: count, dtype: int64

```python
# === Problem 15 ===
print(f"\n--- Problem 15: Top 5 '{assumed_publisher_col}'s by number of books. --
if assumed_publisher_col in df.columns:
    top_publishers = df[assumed_publisher_col].value_counts().head(5)
    print("Top 5 publishers by number of books:")
    print(top_publishers)
else: print(f"Column '{assumed_publisher_col}' not found.")
```

--- Problem 16: Average 'ratings_count' for books by 'Penguin Books'. ---
Average ratings count for books by 'Penguin Books': 42,021

```python
# === Problem 16 ===
print(f"\n--- Problem 16: Average '{assumed_numeric_cols[2]}' for books by 'Pengu
if assumed_publisher_col in df.columns and assumed_numeric_cols[2] in df.columns
    target_publisher = 'Penguin Books' # Adjust name if needed
    publisher_books = df[df[assumed_publisher_col] == target_publisher]
    if not publisher_books.empty:
        avg_ratings_publisher = publisher_books[assumed_numeric_cols[2]].mean()
        print(f"Average ratings count for books by '{target_publisher}': {avg_rat
    else:
        print(f"No books found for publisher '{target_publisher}'.")
else: print(f"Required columns ('{assumed_publisher_col}' or '{assumed_numeric_co
```

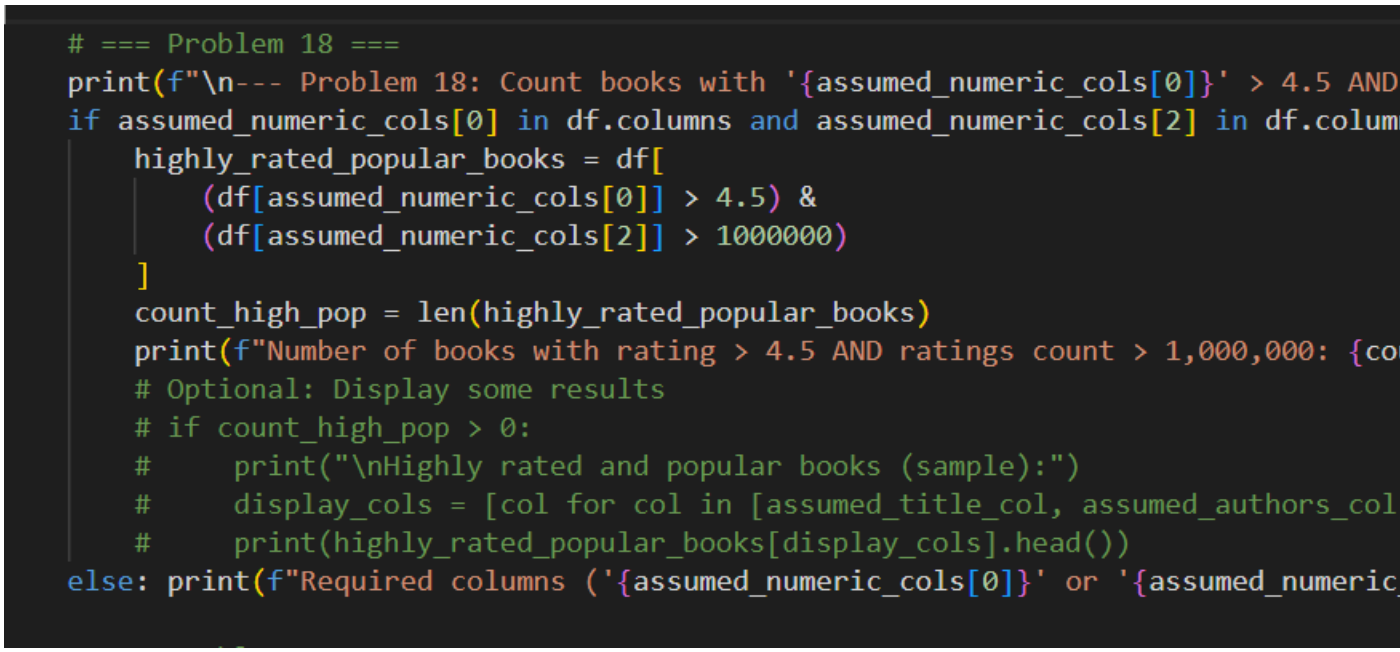
--- Problem 17: Calculate average ratio of 'text_reviews_count' to 'ratings_count'. ---
Added 'review_ratio' column (may contain NaNs).
Average ratio of text reviews to ratings (where calculable): 0.0776



```
# === Problem 17 ===
print(f"\n--- Problem 17: Calculate average ratio of '{assumed_numeric_cols[3]}' to
if assumed_numeric_cols[3] in df.columns and assumed_numeric_cols[2] in df.columns:
    # Avoid division by zero or by NaN
    numerator = df[assumed_numeric_cols[3]].dropna()
    denominator = df[assumed_numeric_cols[2]].replace(0, np.nan).dropna() # Replace
    # Align indices before division
    common_index = numerator.index.intersection(denominator.index)
    if not common_index.empty:
        review_ratio = numerator.loc[common_index] / denominator.loc[common_index]
        average_ratio = review_ratio.mean()
        df['review_ratio'] = review_ratio # Add column, will have NaNs where divisio
        print("Added 'review_ratio' column (may contain NaNs).")
        print(f"Average ratio of text reviews to ratings (where calculable): {averag
    else:
        print("Could not calculate review ratio (no common valid data points).")
else: print(f"Required columns ('{assumed_numeric_cols[3]}' or '{assumed_numeric_co
```

Problem 18: Count books with 'average_rating' > 4.5 AND 'ratings_count' > 1,000,000. ---

Number of books with rating > 4.5 AND ratings count > 1,000,000: 2



```
# === Problem 18 ===
print(f"\n--- Problem 18: Count books with '{assumed_numeric_cols[0]}' > 4.5 AND
if assumed_numeric_cols[0] in df.columns and assumed_numeric_cols[2] in df.colum
    highly_rated_popular_books = df[
        (df[assumed_numeric_cols[0]] > 4.5) &
        (df[assumed_numeric_cols[2]] > 1000000)
    ]
    count_high_pop = len(highly_rated_popular_books)
    print(f"Number of books with rating > 4.5 AND ratings count > 1,000,000: {co
    # Optional: Display some results
    # if count_high_pop > 0:
    #     print("\nHighly rated and popular books (sample):")
    #     display_cols = [col for col in [assumed_title_col, assumed_authors_col
    #     print(highly_rated_popular_books[display_cols].head())
else: print(f"Required columns ('{assumed_numeric_cols[0]}' or '{assumed_numeric_
```

--- Problem 19: Count missing values in 'publisher'. ---

Number of missing values in 'publisher': 0

```
# === Problem 19 ===
print(f"\n--- Problem 19: Count missing values in '{assumed_publisher_col}'. ---") 
if assumed_publisher_col in df.columns:
    missing_publishers = df[assumed_publisher_col].isnull().sum()
    print(f"Number of missing values in '{assumed_publisher_col}': {missing_publish
else: print(f"Column '{assumed_publisher_col}' not found.")
```

--- Problem 20: Calculate 90th percentile for 'num_pages' for 'eng' books using NumPy. ---

Required columns ('num_pages' or 'language_code') not found.

```
# === Problem 20 ===
print(f"\n--- Problem 20: Calculate 90th percentile for '{assumed_numeric_cols[1]}
if assumed_numeric_cols[1] in df.columns and assumed_lang_col in df.columns:
    # Filter for target language and get pages, drop NaNs
    target_lang_code = 'eng' # Adjust if needed
    lang_pages = df[df[assumed_lang_col] == target_lang_code][assumed_numeric_cols
    if not lang_pages.empty:
        # Use NumPy's percentile function on the values
        percentile_90_pages_lang = np.percentile(lang_pages.values, 90)
        print(f"90th percentile of pages for '{target_lang_code}' books: {percenti
    else:
        print(f"No '{target_lang_code}' books with valid page numbers found.")
else: print(f"Required columns ('{assumed_numeric_cols[1]}' or '{assumed_lang_col}
```