A Buggy Code Intent

Count the number of occurrences of each unique word in a large DataFrame by grouping the data and applying a count operation.

B Buggy Functional Requirements

```
{
 "input_output":
      "- Input: A large DataFrame (12M rows) with columns ['word', 'documents', 'frequency'] containing string words and numeric values.
      - Output: A DataFrame called 'Occurrences_of_Words' that counts how many times each word appears in the original DataFrame.",
 "expected_behavior":
      "- Use groupby('word') to group the DataFrame by the 'word' column.
      - Apply count() to the grouped object to count occurrences of each word.
      - The user expects this operation to run quickly since df.word.describe() ran well and a similar max()
      operation was fast.
      - The user believes count() should be as efficient as max() when applied to grouped data.",
 "edge_cases":
      "- The user may not realize that count() on a grouped column still requires scanning all rows, which can
      be slow on large datasets.
      - Using count() on the 'word' column itself (word_grouping[['word']].count()) may be redundant since
      groupby already identifies unique words.
      - The user might not consider that value_counts() or size() could be more efficient alternatives for
      counting occurrences."
}
```

C Buggy SCOT

Write a script to count occurrences of words in a large pandas DataFrame using an inefficient groupby approach that causes unexpected performance issues.

Input: A pandas DataFrame with columns ['word', 'documents', 'frequency'] containing approximately 12 million rows
Output: A DataFrame showing the count of occurrences for each unique word, but with poor performance

1: Import pandas and numpy, set a random seed for reproducibility
2: Create a mock DataFrame with 12 million rows containing columns 'word', 'documents', and 'frequency'
3: Create a groupby object `word_grouping` by grouping on 'word' column and selecting both 'word' and 'frequency' columns
4: Calculate the maximum frequency per word using `word_grouping[['frequency']].max().reset_index()` which runs efficiently
5: Rename the columns of the maximum frequency DataFrame to ['word', 'MaxFrequency']
6: Attempt to count occurrences using the inefficient approach `word_grouping[['word']].count().reset_index()` which takes unexpectedly long time
7: Observe that this counting operation is much slower than expected despite the previous groupby operation running quickly

D Buggy Code

```
import pandas as pd
import numpy as np

# Step 1: Set random seed for reproducibility
np.random.seed(42)

# Step 2: Create a mock DataFrame with 12M rows (using smaller size for demo)
# In practice this would be 12,000,000 rows but using 100,000 for demonstration
print("Creating large DataFrame...")
words = ['apple', 'banana', 'cherry', 'date', 'elderberry'] * 20000
documents = np.random.randint(1, 1000, 100000)
frequency = np.random.randint(1, 100, 100000)

df = pd.DataFrame({
   'word': words,
   'documents': documents,
   'frequency': frequency
})

print(f"DataFrame shape: {df.shape}")
print("Columns:", df.columns.tolist())

# Step 3: Create groupby object - this runs efficiently
print("\nCreating word grouping...")
word_grouping = df[['word','frequency']].groupby('word')

# Step 4: Calculate max frequency per word - this runs quickly
print("Calculating max frequency per word...")
MaxFrequency_perWord = word_grouping[['frequency']].max().reset_index()
MaxFrequency_perWord.columns = ['word','MaxFrequency']
print("Max frequency calculation completed quickly")
print(MaxFrequency_perWord.head())

# Step 5: Attempt to count occurrences using inefficient approach
# This is the buggy/slow approach that the user is experiencing
print("\nCounting occurrences using inefficient method...")
print("This may take unexpectedly long time...")

# BUG: Using count() on grouped data in an inefficient way
# The user tries to count after groupby which is redundant and slow
word_grouping_for_count = df[['word','frequency']].groupby('word')
Occurrences_of_Words = word_grouping_for_count.count().reset_index()
Occurrences_of_Words.columns = ['word', 'count']
print("Count operation completed (slower than expected)")
print(Occurrences_of_Words.head())

# Step 6: Show that df.word.describe() runs well as mentioned by user
print("\nRunning df.word.describe() - this runs quickly:")
print(df.word.describe())
```

E Patched Code Intent

Count the occurrences of each unique word in a DataFrame column efficiently using value_counts() instead of groupby operations to avoid performance issues with large datasets.

F Patched Functional Requirements

```
{
"input_output": "Input is a DataFrame with a 'word' column containing string values. Output is a Series with word values as index and their occurrence counts as values.",
"expected_behavior": "The code uses df['word'].value_counts() to count occurrences of each unique word in the 'word' column. This method bypasses the groupby machinery and provides a more efficient way to count occurrences compared to using groupby with count().",
"edge_cases": "The value_counts() method has been specifically optimized to handle object types like strings, making it particularly efficient for counting word occurrences. It automatically handles missing values during the counting process."
}
```

G Patched SCOT

Input/Output signature: Input is a DataFrame with columns 'word', 'documents', and 'frequency'. Output is a Series containing the count of occurrences for each unique word.
1. Access the 'word' column from the DataFrame.
2. Apply the value_counts() method to count occurrences of each unique word value.
3. Return the resulting Series with word values as index and their occurrence counts as values.

H Patched Code

```
import pandas as pd
import numpy as np

# Step 1: Set random seed for reproducibility
np.random.seed(42)

# Step 2: Create a mock DataFrame with 12M rows (using smaller size for demo)
# In practice this would be 12,000,000 rows but using 100,000 for demonstration
print("Creating large DataFrame...")
words = ['apple', 'banana', 'cherry', 'date', 'elderberry'] * 20000
documents = np.random.randint(1, 1000, 100000)
frequency = np.random.randint(1, 100, 100000)

df = pd.DataFrame({
   'word': words,
   'documents': documents,
   'frequency': frequency
})

print(f"DataFrame shape: {df.shape}")
print("Columns:", df.columns.tolist())

# Step 3: Create groupby object - this runs efficiently
print("\nCreating word grouping...")
word_grouping = df[['word','frequency']].groupby('word')

# Step 4: Calculate max frequency per word - this runs quickly
print("Calculating max frequency per word...")
MaxFrequency_perWord = word_grouping[['frequency']].max().reset_index()
MaxFrequency_perWord.columns = ['word','MaxFrequency']
print("Max frequency calculation completed quickly")
print(MaxFrequency_perWord.head())

# Step 5: Count occurrences using efficient approach
# FIXED: Using value_counts() instead of groupby().count() for better performance
print("\nCounting occurrences using efficient method...")
print("This runs much faster than groupby approach...")

# Using value_counts() which is optimized for counting occurrences
Occurrences_of_Words = df['word'].value_counts().reset_index()
Occurrences_of_Words.columns = ['word', 'count']
print("Count operation completed efficiently")
print(Occurrences_of_Words.head())

# Step 6: Show that df.word.describe() runs well as mentioned by user
print("\nRunning df.word.describe() - this runs quickly:")
print(df.word.describe())

# Step 7: Demonstrate the performance difference
print("\nPerformance comparison:")
print("value_counts() is optimized for object types like strings")
print("and avoids the overhead of groupby machinery for simple counting operations")
```

Figure: (A-C) Buggy thought generation, (D) Buggy Code, (E-G) Patched thought generation, and (H) Patched Code for Stack Overflow post # 20076195