project

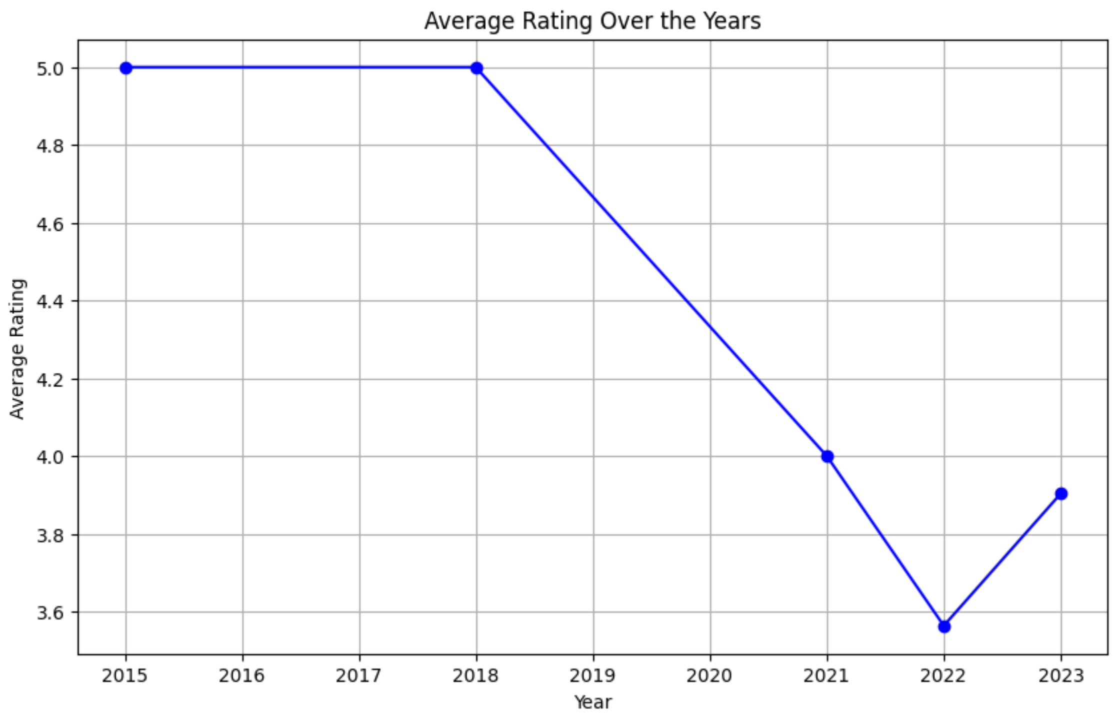
December 1, 2023

[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from io import StringIO

df = pd.read_csv('rawdata.csv')

df.head()
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')
df.fillna(value={"Location": "Unknown"}, inplace=True)
df.head()
```

[3]:
```
                              Title  \
0                       Great place
1    Great career-starting company
2                        No respect
3                              Ehh…
4   Let Go While 7 Months Pregnant

                                           Title_URL  Rating       Date  \
0  https://www.glassdoor.com/Reviews/Employee-Rev…     5.0 2023-11-29
1  https://www.glassdoor.com/Reviews/Employee-Rev…     5.0 2018-06-11
2  https://www.glassdoor.com/Reviews/Employee-Rev…     1.0 2023-11-27
3  https://www.glassdoor.com/Reviews/Employee-Rev…     2.0 2023-11-21
4  https://www.glassdoor.com/Reviews/Employee-Rev…     1.0 2023-10-16

                      Position                               Status  \
0           Anonymous Employee   Current Employee, more than 1 year
1                  QA Engineer  Current Employee, more than 3 years
2    Regional Support Technician  Current Employee, more than 5 years
3  Financial Customer Associate                     Current Employee
4        Financial Consultant I   Former Employee, more than 1 year

         Location View7                                               pro  \
0         Unknown  Pros  Culture Benefits Employees Leadership Work lif…
1  Smithfield, RI  Pros  Benefits including profit sharing and bonuses …
2   Merrimack, NH  Pros  The benefits are solid… Or so I'm told hones…
```

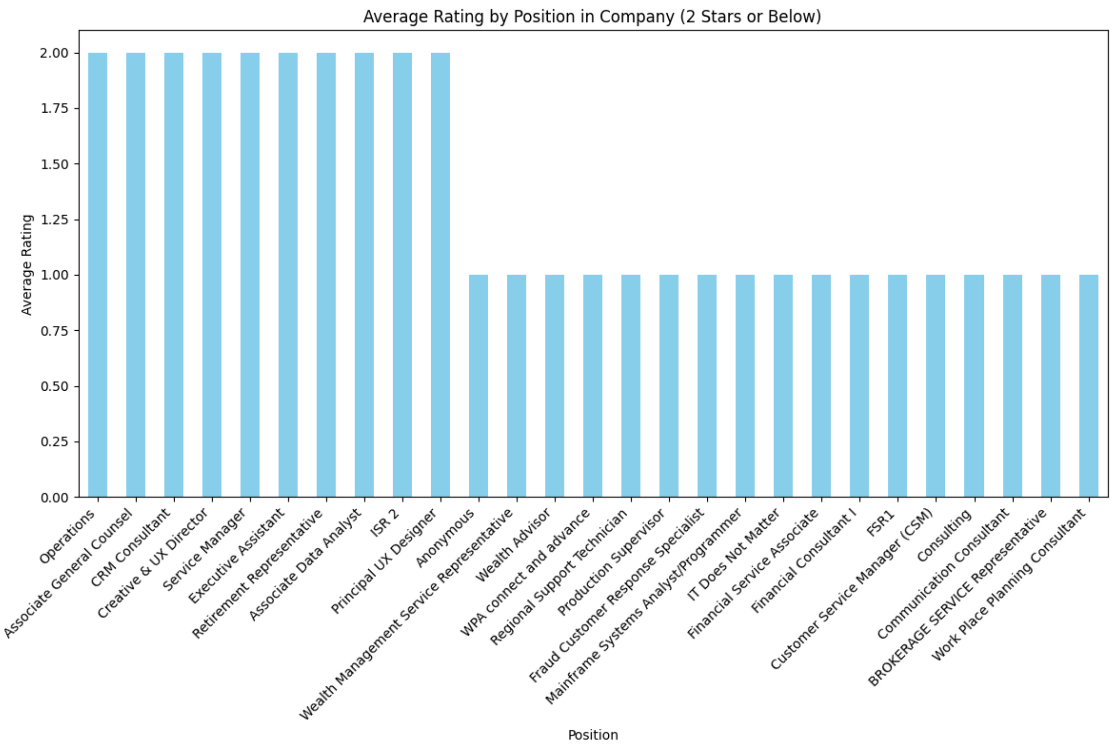
```
3      Durham, NC  Pros  The starting pay is pretty good for a call cen…
4     Chicago, IL  Pros  I was drawn to the training program for the Fi…

  View9                                                      con  helpfulness
0  Cons               There are no cons, great place to work.          NaN
1  Cons  In my current role, I am working overtime more…        226.0
2  Cons  Despite the fact that the average employee of …          3.0
3  Cons  They claim that there are so many different op…          3.0
4  Cons  I was in a 1 year training program for the Fin…         18.0
```

[4]:
```python
df['Year'] = df['Date'].dt.year
average_ratings_per_year = df.groupby('Year')['Rating'].mean()

average_ratings_per_year.plot(kind='line', marker='o', color='b', figsize=(10,␣
 ↪6))
plt.title('Average Rating Over the Years')
plt.xlabel('Year')
plt.ylabel('Average Rating')
plt.grid(True)
plt.show()
```

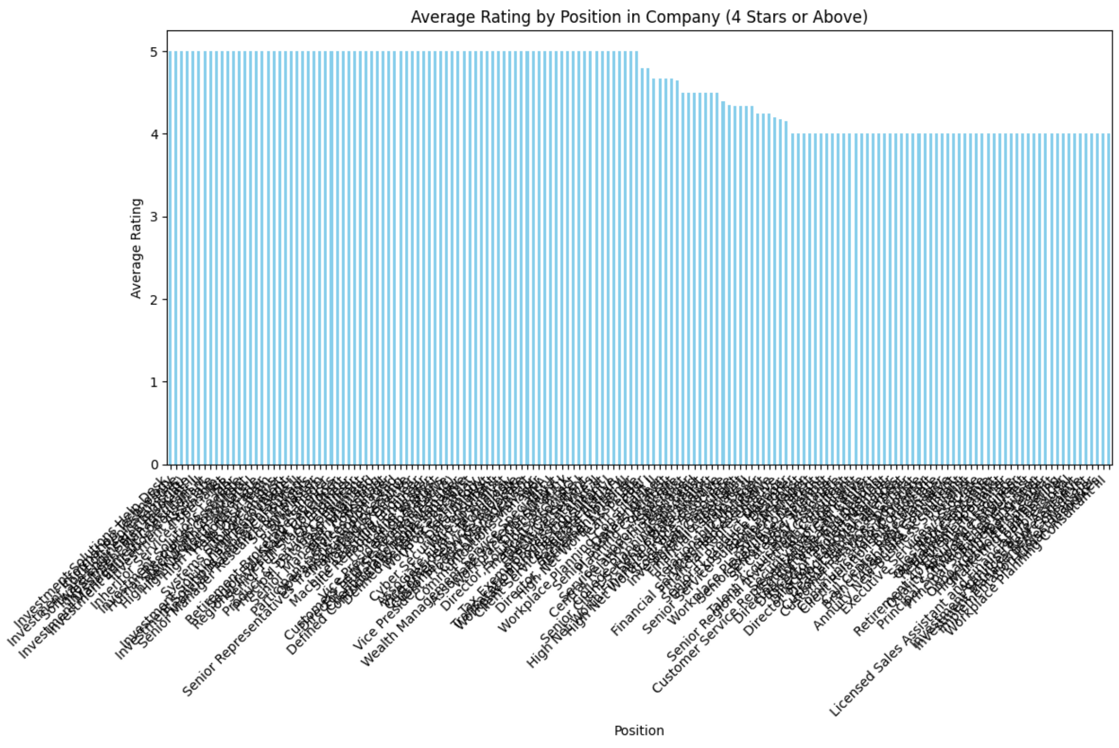


[18]:
```python
import pandas as pd
import matplotlib.pyplot as plt
```

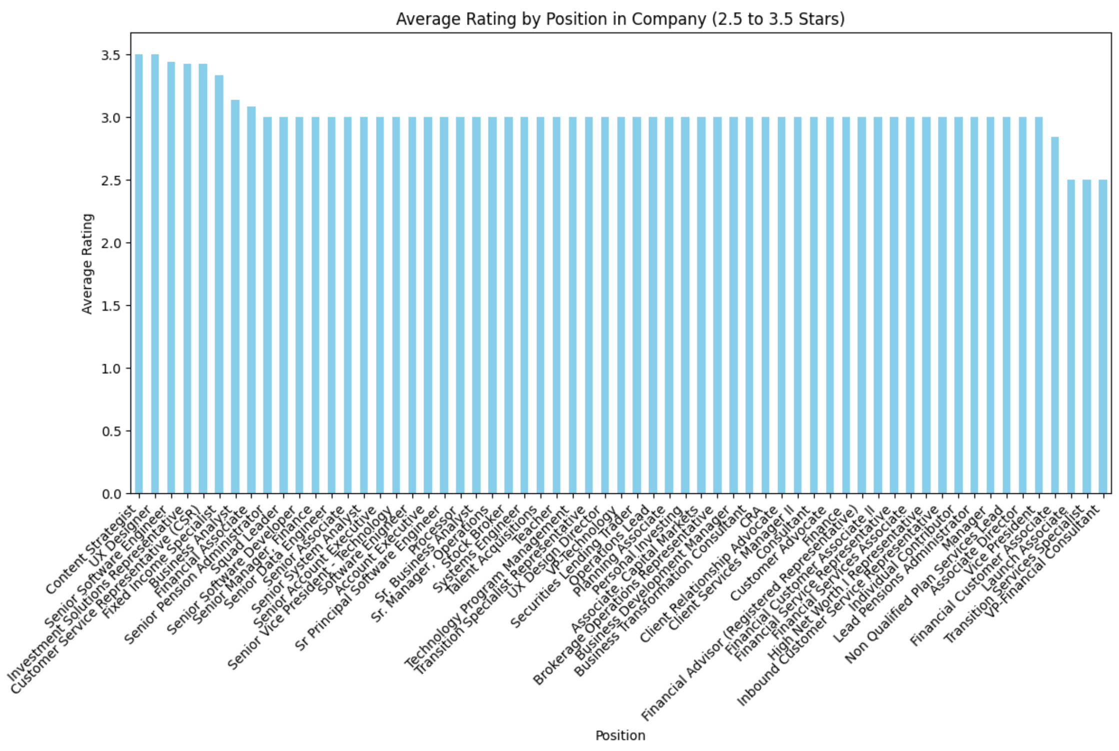
```python
# Convert 'Rating' to a numeric type
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Group by the 'Position' and calculate the average 'Rating'
average_ratings_by_position = df.groupby('Position')['Rating'].mean()

# Filter out positions with an average rating greater than 2
average_ratings_below_two =␣
 ↪average_ratings_by_position[average_ratings_by_position <= 2].
 ↪sort_values(ascending=False)

# Plotting the average ratings by position for ratings of 2 or below
plt.figure(figsize=(12, 8))
average_ratings_below_two.plot(kind='bar', color='skyblue')
plt.title('Average Rating by Position in Company (2 Stars or Below)')
plt.xlabel('Position')
plt.ylabel('Average Rating')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()  # Adjusts the plot to ensure everything fits without␣
 ↪overlapping
plt.show()
```



```
[19]: import pandas as pd
      import matplotlib.pyplot as plt


      # Convert 'Rating' to a numeric type
      df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

      # Group by the 'Position' and calculate the average 'Rating'
      average_ratings_by_position = df.groupby('Position')['Rating'].mean()

      # Filter out positions with an average rating less than 4
      average_ratings_four_or_above =␣
       ↪average_ratings_by_position[average_ratings_by_position >= 4].
       ↪sort_values(ascending=False)

      # Plotting the average ratings by position for ratings of 4 or above
      plt.figure(figsize=(12, 8))
      average_ratings_four_or_above.plot(kind='bar', color='skyblue')
      plt.title('Average Rating by Position in Company (4 Stars or Above)')
      plt.xlabel('Position')
      plt.ylabel('Average Rating')
      plt.xticks(rotation=45, ha='right')
      plt.tight_layout()  # Adjusts the plot to ensure everything fits without␣
       ↪overlapping
      plt.show()
```

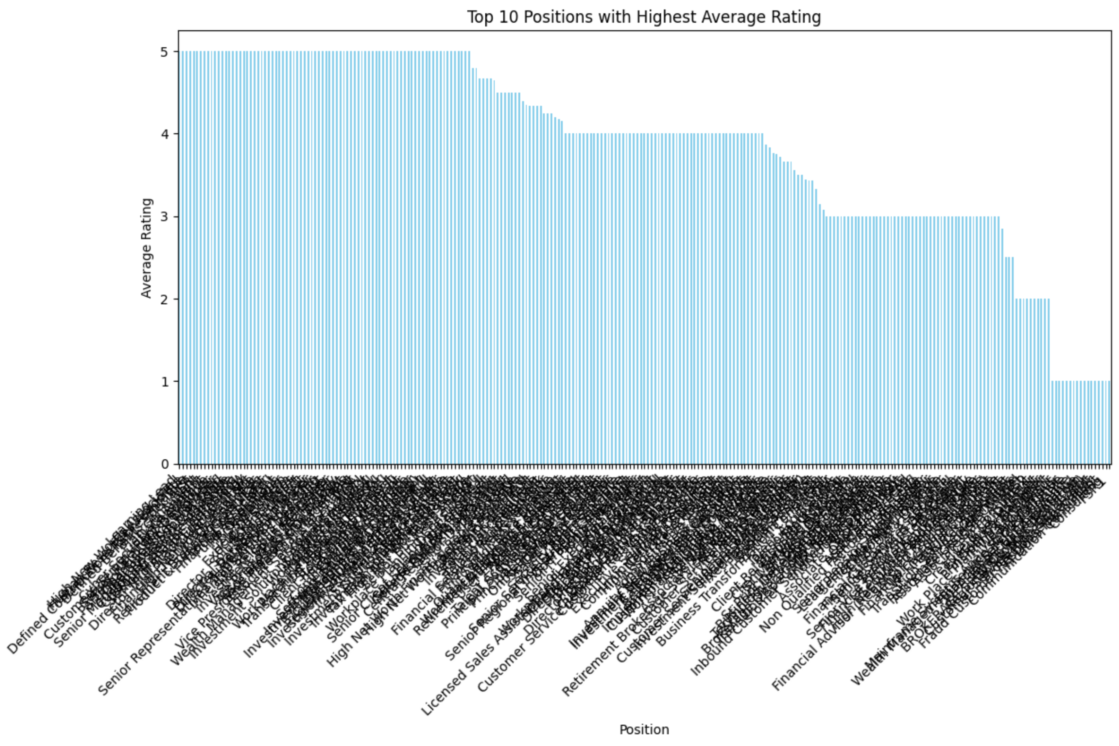


[20]:
```
import pandas as pd
import matplotlib.pyplot as plt


# Convert 'Rating' to a numeric type
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Group by the 'Position' and calculate the average 'Rating'
average_ratings_by_position = df.groupby('Position')['Rating'].mean()

# Filter out positions with an average rating between 2.5 and 3.5
average_ratings_between =␣
 ↪average_ratings_by_position[(average_ratings_by_position >= 2.5) &␣
 ↪(average_ratings_by_position <= 3.5)].sort_values(ascending=False)

# Plotting the average ratings by position for ratings between 2.5 and 3.5
plt.figure(figsize=(12, 8))
average_ratings_between.plot(kind='bar', color='skyblue')
plt.title('Average Rating by Position in Company (2.5 to 3.5 Stars)')
plt.xlabel('Position')
plt.ylabel('Average Rating')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()  # Adjusts the plot to ensure everything fits without␣
 ↪overlapping
plt.show()
```



[42]:
```
import pandas as pd
import matplotlib.pyplot as plt


# Convert 'Rating' to a numeric type
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Group by the 'Position' and calculate the average 'Rating'
average_ratings_by_position = df.groupby('Position')['Rating'].mean()


top_positions = average_ratings_by_position.round(10).
 ↪sort_values(ascending=False)
print(top_positions)


plt.figure(figsize=(12, 8))
top_positions.plot(kind='bar', color='skyblue')
plt.title('Top 10 Positions with Highest Average Rating')
plt.xlabel('Position')
plt.ylabel('Average Rating')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()  # Adjusts the plot to ensure everything fits without␣
 ↪overlapping
plt.show()
```

```
Position
Learning Lead                               5.0
High New Worth Associate I                  5.0
Human Resources Associate                   5.0
Defined Contributions Representative        5.0
Defined Benefits Specialist                 5.0
                                           …
Fraud Customer Response Specialist          1.0
IT Does Not Matter                          1.0
Communication Consultant                    1.0
Consulting                                  1.0
FSR1                                        1.0
Name: Rating, Length: 261, dtype: float64
```

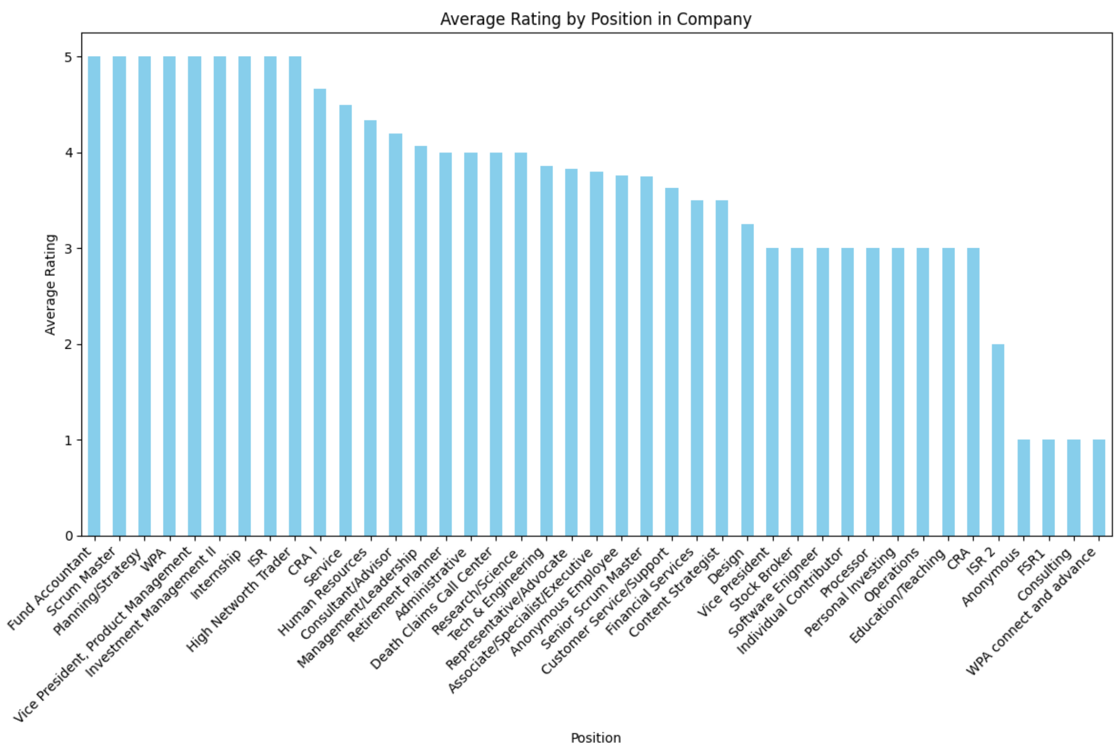


[50]:
```
import pandas as pd



def categorize_position(position):
    position_lower = position.lower()
    if any(keyword in position_lower for keyword in ['consultant', 'advisor']):
        return 'Consultant/Advisor'
    elif any(keyword in position_lower for keyword in ['associate',␣
↪'specialist', 'executive']):
        return 'Associate/Specialist/Executive'
    elif any(keyword in position_lower for keyword in ['analyst', 'programmer',␣
↪'engineer', 'developer', 'it', 'technology']):
        return 'Tech & Engineering'
    elif 'customer service' in position_lower or 'help desk' in position_lower␣
↪or 'support' in position_lower:
        return 'Customer Service/Support'
    elif any(keyword in position_lower for keyword in ['representative',␣
↪'advocate']):
        return 'Representative/Advocate'
    elif any(keyword in position_lower for keyword in ['manager', 'supervisor',␣
↪'director', 'leader', 'lead', 'head']):
        return 'Management/Leadership'
    elif 'financial' in position_lower or 'finance' in position_lower:
        return 'Financial Services'
    elif 'marketing' in position_lower or 'sales' in position_lower:
        return 'Marketing & Sales'
    elif 'human resources' in position_lower or 'hr' in position_lower:
        return 'Human Resources'
    elif 'admin' in position_lower or 'secretary' in position_lower or␣
↪'assistant' in position_lower:
        return 'Administrative'
    elif 'operations' in position_lower:
        return 'Operations'
    elif 'design' in position_lower or 'ux' in position_lower:
        return 'Design'
    elif 'intern' in position_lower or 'internship' in position_lower:
        return 'Internship'
    elif 'service' in position_lower:
        return 'Service'
    elif 'planning' in position_lower or 'strategy' in position_lower:
        return 'Planning/Strategy'
    elif 'compliance' in position_lower or 'legal' in position_lower:
        return 'Compliance/Legal'
    elif 'education' in position_lower or 'teacher' in position_lower or␣
↪'instructor' in position_lower:
        return 'Education/Teaching'
    elif 'research' in position_lower or 'scientist' in position_lower:
        return 'Research/Science'
    else:
        words = position.split()
        # Return the last three words joined by spaces, or the entire string if␣
 ↪it has less than three words
        return ' '.join(words[-4:]) if len(words) >= 4 else ' '.join(words)
# Apply the function to create a new column
df['general_position'] = df['Position'].apply(categorize_position)

# Display the DataFrame to verify the new column
print(df['general_position'].unique())
```

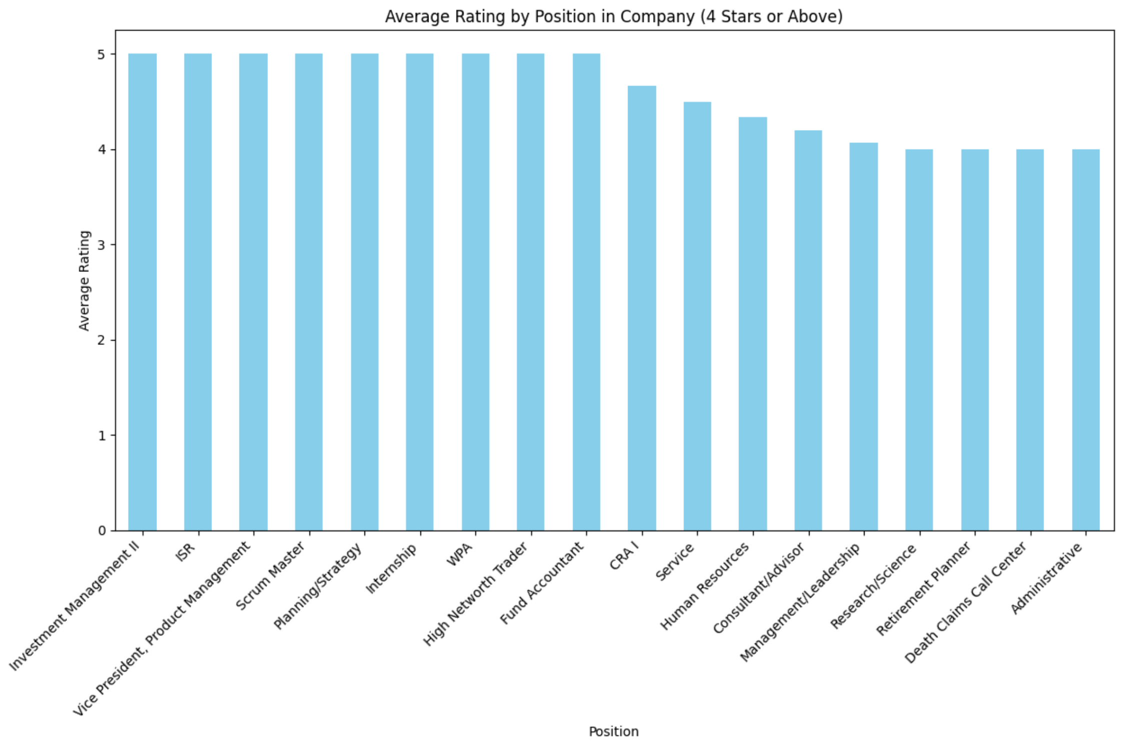
```
['Anonymous Employee' 'Tech & Engineering' 'Customer Service/Support'
 'Associate/Specialist/Executive' 'Consultant/Advisor'
 'Representative/Advocate' 'Management/Leadership' 'Design'
 'Financial Services' 'Retirement Planner' 'CRA' 'Stock Broker'
 'Senior Scrum Master' 'Internship' 'Vice President, Product Management'
 'CRA I' 'Investment Management II' 'ISR' 'WPA' 'Software Enigneer'
 'Consulting' 'FSR1' 'Personal Investing' 'Scrum Master'
 'Content Strategist' 'Vice President' 'Operations' 'Service'
 'Fund Accountant' 'Administrative' 'Death Claims Call Center'
 'Human Resources' 'Planning/Strategy' 'Individual Contributor' 'ISR 2'
 'Education/Teaching' 'High Networth Trader' 'WPA connect and advance'
 'Research/Science' 'Anonymous' 'Processor']
```

[52]:
```
import pandas as pd
import matplotlib.pyplot as plt


# Convert 'Rating' to a numeric type
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Group by the 'Position' and calculate the average 'Rating'
average_ratings_by_position = df.groupby('general_position')['Rating'].mean()


top_positions = average_ratings_by_position.round(10).
 ↪sort_values(ascending=False)
print(top_positions)


plt.figure(figsize=(12, 8))
top_positions.plot(kind='bar', color='skyblue')
plt.title('Average Rating by Position in Company')
plt.xlabel('Position')
plt.ylabel('Average Rating')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()  # Adjusts the plot to ensure everything fits without␣
 ↪overlapping
plt.show()
```

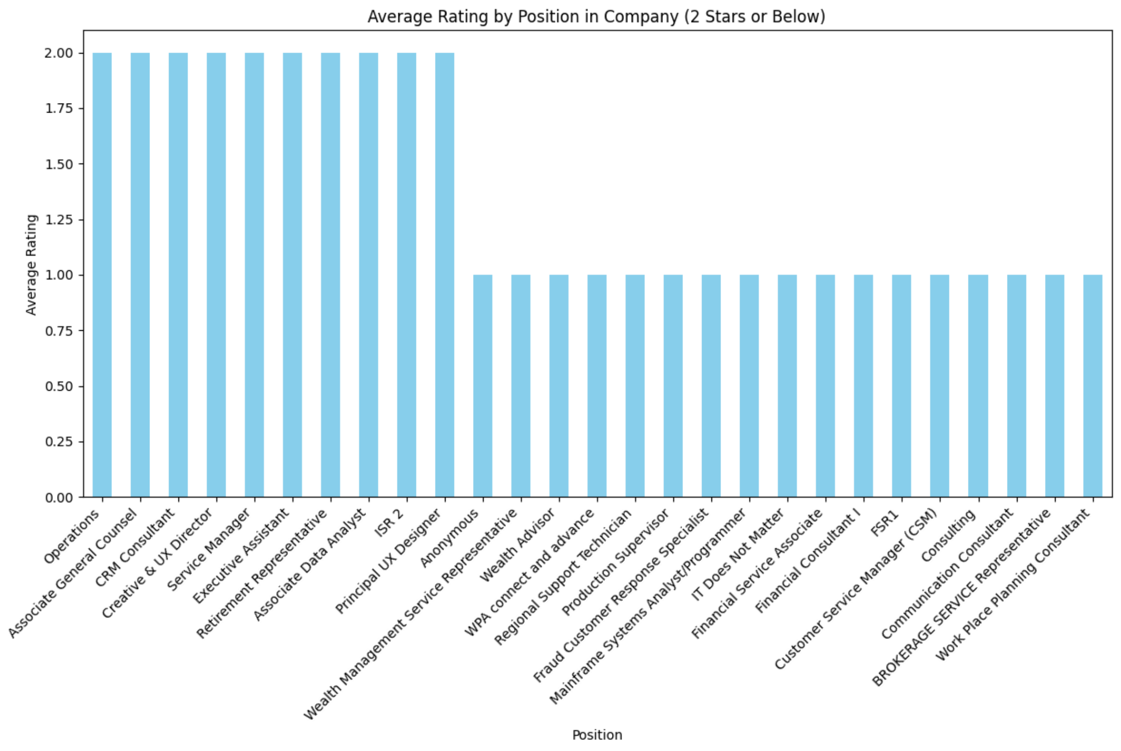
```
general_position
Fund Accountant                        5.000000
Scrum Master                           5.000000
Planning/Strategy                      5.000000
WPA                                    5.000000
Vice President, Product Management     5.000000
Investment Management II               5.000000
Internship                             5.000000
ISR                                    5.000000
High Networth Trader                   5.000000
CRA I                                  4.666667
Service                                4.500000
Human Resources                        4.333333
Consultant/Advisor                     4.202703
Management/Leadership                  4.073529
Retirement Planner                     4.000000
Administrative                         4.000000
Death Claims Call Center               4.000000
Research/Science                       4.000000
Tech & Engineering                     3.858824
Representative/Advocate                3.832061
Associate/Specialist/Executive         3.804511
Anonymous Employee                     3.760000
Senior Scrum Master                    3.750000
Customer Service/Support               3.628571
Financial Services                     3.500000
Content Strategist                     3.500000
Design                                 3.250000
Vice President                         3.000000
Stock Broker                           3.000000
Software Enigneer                      3.000000
Individual Contributor                 3.000000
Processor                              3.000000
Personal Investing                     3.000000
Operations                             3.000000
Education/Teaching                     3.000000
CRA                                    3.000000
ISR 2                                  2.000000
Anonymous                              1.000000
FSR1                                   1.000000
Consulting                             1.000000
WPA connect and advance                1.000000
Name: Rating, dtype: float64
```



[53]:
```
import pandas as pd
import matplotlib.pyplot as plt


# Convert 'Rating' to a numeric type
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Group by the 'Position' and calculate the average 'Rating'
average_ratings_by_position = df.groupby('general_position')['Rating'].mean()

# Filter out positions with an average rating less than 4
average_ratings_four_or_above =␣
↪average_ratings_by_position[average_ratings_by_position >= 4].
↪sort_values(ascending=False)

# Plotting the average ratings by position for ratings of 4 or above
plt.figure(figsize=(12, 8))
average_ratings_four_or_above.plot(kind='bar', color='skyblue')
plt.title('Average Rating by Position in Company (4 Stars or Above)')
plt.xlabel('Position')
plt.ylabel('Average Rating')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()  # Adjusts the plot to ensure everything fits without␣
↪overlapping
plt.show()
```



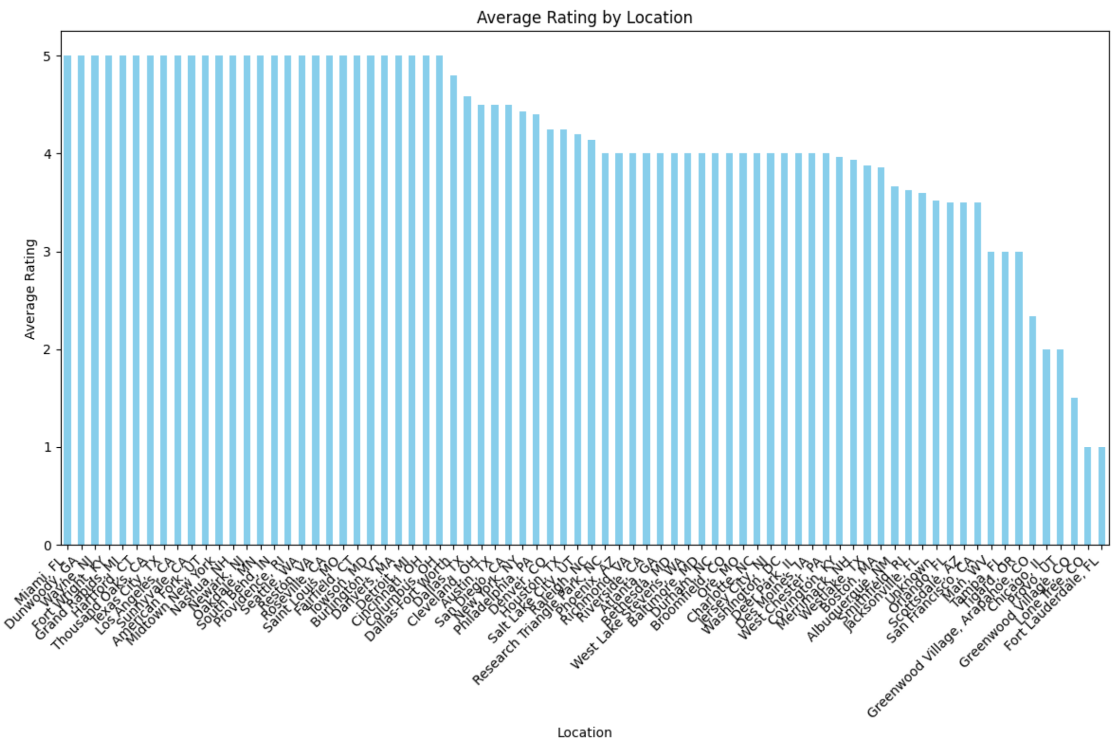
[54]:
```
import pandas as pd
import matplotlib.pyplot as plt


# Convert 'Rating' to a numeric type
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Group by the 'Position' and calculate the average 'Rating'
average_ratings_by_position = df.groupby('Position')['Rating'].mean()

# Filter out positions with an average rating greater than 2
average_ratings_below_two =␣
↪average_ratings_by_position[average_ratings_by_position <= 2].
↪sort_values(ascending=False)

# Plotting the average ratings by position for ratings of 2 or below
plt.figure(figsize=(12, 8))
average_ratings_below_two.plot(kind='bar', color='skyblue')
plt.title('Average Rating by Position in Company (2 Stars or Below)')
plt.xlabel('Position')
plt.ylabel('Average Rating')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()  # Adjusts the plot to ensure everything fits without␣
 ↪overlapping
plt.show()
```



[55]:
```
import pandas as pd
import matplotlib.pyplot as plt


# Ensure 'Rating' is a numeric type
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Group by 'Location' and calculate the average 'Rating'
average_ratings_by_location = df.groupby('Location')['Rating'].mean()

# You can sort the results if desired
average_ratings_by_location_sorted = average_ratings_by_location.
 ↪sort_values(ascending=False)

# Display the average ratings by location
print(average_ratings_by_location_sorted)
```

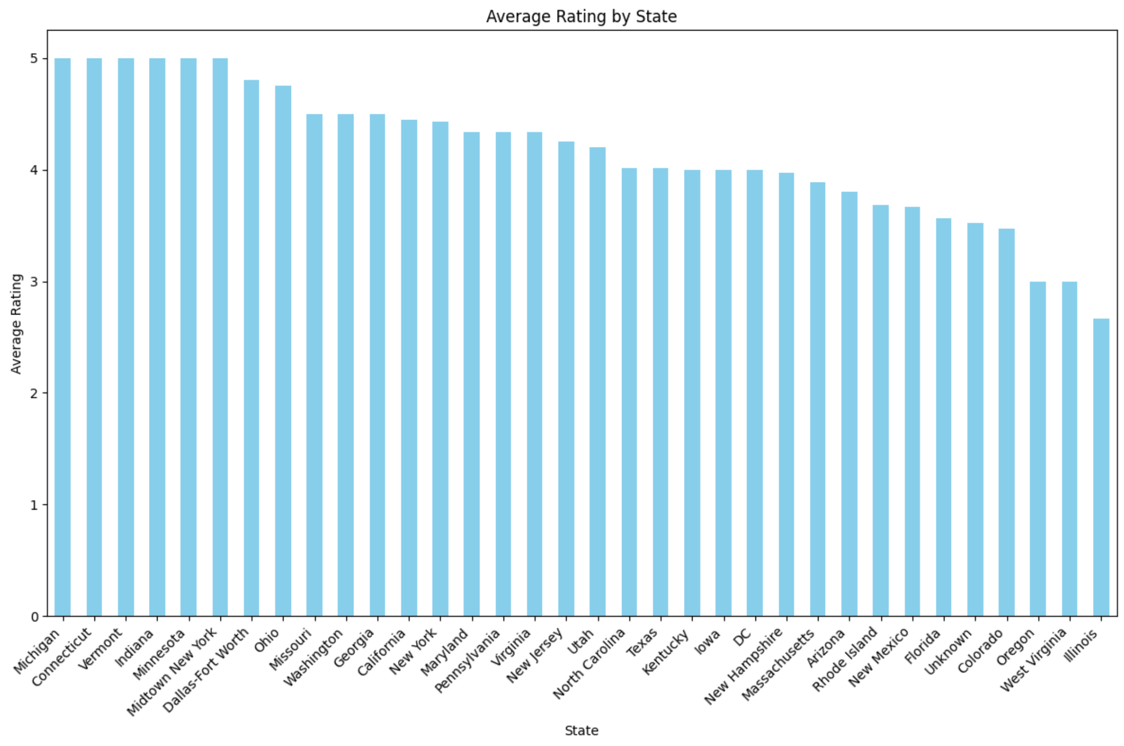
```python
# Optionally, you can plot this data
plt.figure(figsize=(12, 8))
average_ratings_by_location_sorted.plot(kind='bar', color='skyblue')
plt.title('Average Rating by Location')
plt.xlabel('Location')
plt.ylabel('Average Rating')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()  # Adjusts the plot to ensure everything fits without␣
 ↪overlapping
plt.show()
```

```
Location
Miami, FL                 5.0
Dunwoody, GA              5.0
Wayne, NJ                 5.0
Fort Wright, KY           5.0
Grand Rapids, MI          5.0
                         …
Chicago, IL               2.0
Provo, UT                 2.0
Greenwood Village, CO     1.5
Lone Tree, CO             1.0
Fort Lauderdale, FL       1.0
Name: Rating, Length: 76, dtype: float64
```



[57]:
```
import pandas as pd
import matplotlib.pyplot as plt


# Ensure 'Rating' is a numeric type
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Function to extract state initials
def extract_state(location):
    parts = location.split(', ')
    return parts[-1] if len(parts) > 1 else location

# Extract state initials to a new column
df['location_state'] = df['Location'].apply(extract_state)

# Mapping of state initials to full names
state_mapping = {
    'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona', 'AR': 'Arkansas',
    'CA': 'California', 'CO': 'Colorado', 'CT': 'Connecticut', 'DE': 'Delaware',
    'FL': 'Florida', 'GA': 'Georgia', 'HI': 'Hawaii', 'ID': 'Idaho',
    'IL': 'Illinois', 'IN': 'Indiana', 'IA': 'Iowa', 'KS': 'Kansas',
    'KY': 'Kentucky', 'LA': 'Louisiana', 'ME': 'Maine', 'MD': 'Maryland',
    'MA': 'Massachusetts', 'MI': 'Michigan', 'MN': 'Minnesota', 'MS':␣
 ↪'Mississippi',
    'MO': 'Missouri', 'MT': 'Montana', 'NE': 'Nebraska', 'NV': 'Nevada',
    'NH': 'New Hampshire', 'NJ': 'New Jersey', 'NM': 'New Mexico', 'NY': 'New␣
 ↪York',
    'NC': 'North Carolina', 'ND': 'North Dakota', 'OH': 'Ohio', 'OK':␣
 ↪'Oklahoma',
    'OR': 'Oregon', 'PA': 'Pennsylvania', 'RI': 'Rhode Island', 'SC': 'South␣
 ↪Carolina',
    'SD': 'South Dakota', 'TN': 'Tennessee', 'TX': 'Texas', 'UT': 'Utah',
    'VT': 'Vermont', 'VA': 'Virginia', 'WA': 'Washington', 'WV': 'West␣
 ↪Virginia',
    'WI': 'Wisconsin', 'WY': 'Wyoming'
}

# Replace state initials with full names, retain original if no mapping found
df['location_state'] = df['location_state'].apply(lambda x: state_mapping.
 ↪get(x, x))

# Group by the new 'location_state' and calculate the average 'Rating'
average_ratings_by_location_state = df.groupby('location_state')['Rating'].
 ↪mean().sort_values(ascending=False)
```

```
# Optionally, plot this data
plt.figure(figsize=(12, 8))
average_ratings_by_location_state.plot(kind='bar', color='skyblue')
plt.title('Average Rating by State')
plt.xlabel('State')
plt.ylabel('Average Rating')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

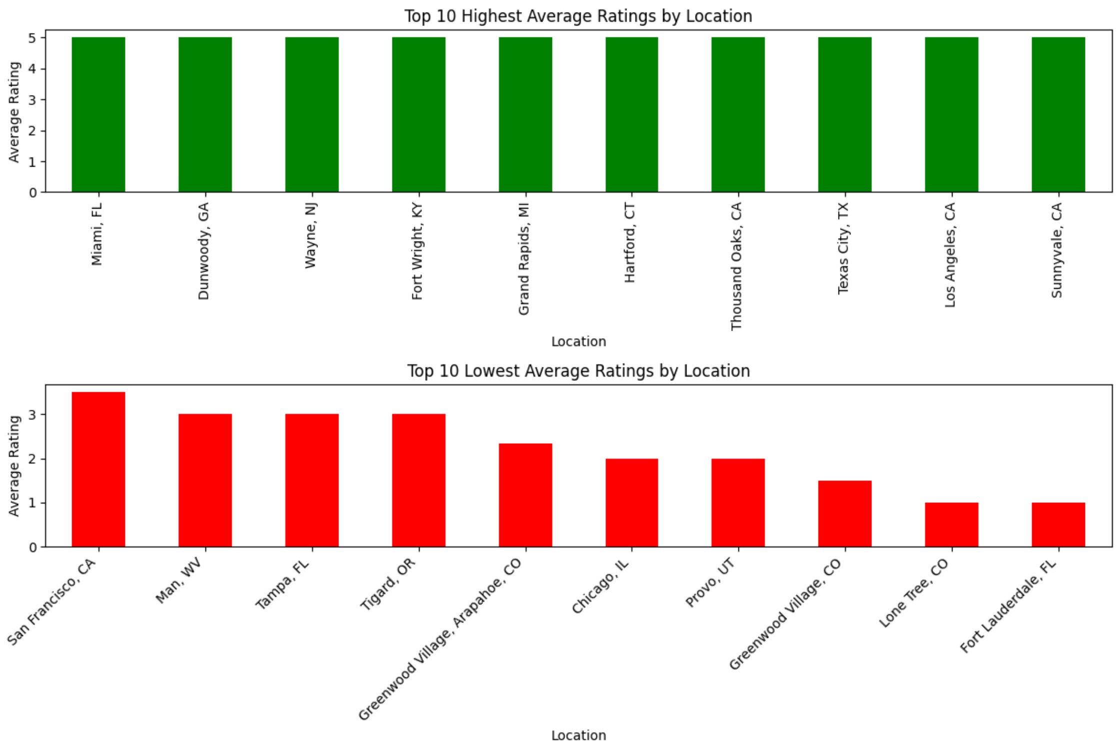


[56]:
```
import pandas as pd
import matplotlib.pyplot as plt


# Ensure 'Rating' is a numeric type
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Group by 'Location' and calculate the average 'Rating'
average_ratings_by_location = df.groupby('Location')['Rating'].mean()

# Sort the average ratings
```

```
average_ratings_by_location_sorted = average_ratings_by_location.
 ↪sort_values(ascending=False)

# Get the top 10 highest average ratings
top_10_highest = average_ratings_by_location_sorted.head(10)
print("Top 10 Highest Average Ratings by Location:")
print(top_10_highest)

# Get the top 10 lowest average ratings
top_10_lowest = average_ratings_by_location_sorted.tail(10)
print("\nTop 10 Lowest Average Ratings by Location:")
print(top_10_lowest)

# Plotting for visualization
plt.figure(figsize=(12, 8))

# Plot for top 10 highest average ratings
plt.subplot(2, 1, 1)
top_10_highest.plot(kind='bar', color='green')
plt.title('Top 10 Highest Average Ratings by Location')
plt.ylabel('Average Rating')

# Plot for top 10 lowest average ratings
plt.subplot(2, 1, 2)
top_10_lowest.plot(kind='bar', color='red')
plt.title('Top 10 Lowest Average Ratings by Location')
plt.ylabel('Average Rating')
plt.xticks(rotation=45, ha='right')

plt.tight_layout()
plt.show()
```

```
Top 10 Highest Average Ratings by Location:
Location
Miami, FL           5.0
Dunwoody, GA        5.0
Wayne, NJ           5.0
Fort Wright, KY     5.0
Grand Rapids, MI    5.0
Hartford, CT        5.0
Thousand Oaks, CA   5.0
Texas City, TX      5.0
Los Angeles, CA     5.0
Sunnyvale, CA       5.0
Name: Rating, dtype: float64

Top 10 Lowest Average Ratings by Location:
Location
```

```
San Francisco, CA                  3.500000
Man, WV                            3.000000
Tampa, FL                          3.000000
Tigard, OR                         3.000000
Greenwood Village, Arapahoe, CO    2.333333
Chicago, IL                        2.000000
Provo, UT                          2.000000
Greenwood Village, CO              1.500000
Lone Tree, CO                      1.000000
Fort Lauderdale, FL                1.000000
Name: Rating, dtype: float64
```



[17]:
```
import pandas as pd
from collections import Counter
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download necessary NLTK data
nltk.download('punkt')
nltk.download('stopwords')

# Assuming your data is loaded into a DataFrame called df
# Convert 'Rating' to a numeric type and filter rows with rating 3 or below
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')
filtered_df = df[df['Rating'] <= 3]

# Convert 'con' to string type
filtered_df['con'] = filtered_df['con'].astype(str)

# Extend the stopwords list with your custom words
custom_stop_words = {'like', 'even', 'people', 'get', 'fidelity'}
stop_words = set(stopwords.words('english')).union(custom_stop_words)

def word_frequency(text_column):
    words = word_tokenize(' '.join(text_column).lower())
    filtered_words = [word for word in words if word.isalpha() and word not in␣
  ↪stop_words]
    return Counter(filtered_words)

# Counting word frequency in 'con'
con_freq = word_frequency(filtered_df['con'])

# Displaying the most common words in 'con' for ratings 3 or below, excluding␣
  ↪custom stop words
print("Most common words in 'con' for ratings 3 or below (excluding certain␣
  ↪words):")
for word, count in con_freq.most_common(10):
    print(f"{word}: {count}")
```

```
Most common words in 'con' for ratings 3 or below (excluding certain words):
work: 89
company: 60
pay: 55
management: 51
job: 49
time: 46
call: 39
office: 37
many: 35
employees: 35
```

```
[nltk_data] Downloading package punkt to /Users/jeet/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/jeet/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
/var/folders/f6/tr_gxbkd2510_wbt4rl50ndh0000gn/T/ipykernel_9549/577463723.py:17:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_df['con'] = filtered_df['con'].astype(str)
```