Solution Set - Advanced Pandas Manipulations

1. Advanced Pivoting and Aggregation

Medium Solution

To pivot the dataset such that rows are `Region` and `Year`, columns are `Quarter`, and the values are the sum of `Sales`, we use the `pivot_table` function:

```
pivot_df = df.pivot_table(index=['Region', 'Year'],
    ↪ columns='Quarter', values='Sales', aggfunc='sum
    ↪ ')
```

Explanation: We use `pivot_table` to group by `Region` and `Year` while summing the sales for each combination of `Quarter`. The `aggfunc='sum'` ensures that sales are aggregated correctly.

Hard Solution

To create a pivot table with sum of `Sales` by `Region` and `Year`, including margins:

```
pivot_df = df.pivot_table(index='Region', columns=['
    ↪ Year', 'Quarter'], values='Sales', aggfunc='sum
    ↪ ', margins=True)
```

Explanation: The `pivot_table` function aggregates sales by `Region` and `Year`, with an additional `margins=True` argument to add row and column totals (i.e., the "All" row and column).

Advanced Solution

For calculating the percentage contribution of each product's sales to the region's total sales:

```
pivot_df = df.pivot_table(index='Product', columns=[
   ↪ 'Region', 'Year'], values='Sales', aggfunc='sum
   ↪ ')
total_sales = pivot_df.sum(axis=0)
percentage_contribution = pivot_df / total_sales *
   ↪ 100
```

Explanation: First, we pivot the data to sum the sales by `Product`, `Region`, and `Year`. Then, we calculate the total sales per region and year by summing across the rows. Finally, we calculate the percentage contribution of each product by dividing the product sales by the total sales and multiplying by 100.

—

2. Advanced Melting and Reshaping

Medium Solution

To melt the DataFrame from wide to long format:

```
melted_df = df.melt(id_vars=['Country', 'Year'],
   ↪ var_name='Metric', value_name='Value')
melted_df['Metric'] = melted_df['Metric'].str.split(
   ↪ '_').str[0]
```

Explanation: The `melt` function is used to reshape the DataFrame so that `Population` and `GDP` are combined into a `Metric` column, and their corresponding values into a `Value` column. The `str.split('`$').str[0] operationisusedtoextractthemetric$

Hard Solution

To reshape the melted DataFrame back to wide format:

```
reshaped_df = melted_df.pivot_table(index=['Country'
   ↪ , 'Year'], columns='Metric', values='Value',
   ↪ aggfunc='first')
```

Explanation: The pivot_table function is used to reshape the DataFrame back to a wide format where Population and GDP are the columns, with the values being the corresponding Value for each Country and Year.

Advanced Solution

For reshaping the melted DataFrame and adding a new Metric Type column:

```
melted_df['Metric Type'] = melted_df['Metric'].map({
    ↪ 'Population': 'Demographic', 'GDP': 'Economic'
    ↪ })
reshaped_df = melted_df.pivot_table(index=['Country'
    ↪ , 'Year'], columns=['Metric Type', 'Metric'],
    ↪ values='Value', aggfunc='first')
```

Explanation: The map function is used to create a new column Metric Type, categorizing the metrics as either Demographic or Economic. Then, we reshape the data using pivot_table with a multi-level column index consisting of Metric Type and Metric.

---

3. Multi-Level Index Manipulation

Medium Solution

To unstack the Quarter level and calculate the Profit Margin:

```
df_unstacked = df.unstack(level='Quarter')
df_unstacked['Profit Margin'] = df_unstacked['Profit
    ↪ '] / df_unstacked['Sales']
```

Explanation: We use unstack to pivot the Quarter level, and then we calculate the Profit Margin by dividing the Profit by the Sales.

Hard Solution

To normalize the Sales values for each region so that they sum to 1:

```
df_normalized = df.copy()
df_normalized['Sales'] = df_normalized.groupby('
    ↪ Region')['Sales'].transform(lambda x: x / x.sum
    ↪ ())
```

Explanation: We use groupby to group by Region and then apply transform to normalize the Sales by dividing each value by the sum of Sales within each region.

Advanced Solution

For reshaping and analyzing the multi-indexed DataFrame:

```
stacked_df = df.stack().reset_index()
stacked_df.columns = ['Region', 'Quarter', 'Metric',
    ↪ 'Value']
filtered_df = stacked_df[stacked_df['Metric'] == '
    ↪ Profit']
filtered_df = filtered_df[filtered_df['Value'] > 50]
```

Explanation: We use stack to move the columns back into the index, followed by reset_index to flatten the DataFrame. The columns are renamed, and then we filter the data to include only Profit rows where the value is greater than 50.

---

4. Method Chaining and Advanced Transformations

Medium Solution

To calculate the total Amount per Customer, sorted in descending order:

```
result_df = df.groupby('Customer')['Amount'].sum().
    ↪ sort_values(ascending=False)
```

Explanation: We use groupby to aggregate the total Amount per Customer and then sort the result in descending order.

Hard Solution

To calculate the percentage contribution of each month's Amount to total sales for the year:

```
df['Year'] = pd.to_datetime(df['Date']).dt.year
df['Month'] = pd.to_datetime(df['Date']).dt.month
monthly_sales = df.groupby(['Year', 'Month'])['
    ↪ Amount'].sum()
df['Percentage'] = df['Amount'] / df['Year'].map(
    ↪ monthly_sales) * 100
```

Explanation: First, we extract the Year and Month from the Date column. Then, we calculate the total sales for each month using groupby. Finally, we calculate the percentage contribution of each sale to the total monthly sales.

Advanced Solution

To identify the month with the highest Amount for each Year:

```
monthly_sales = df.groupby(['Year', 'Month'])['
    ↪ Amount'].sum()
highest_sales_month = monthly_sales.groupby('Year').
    ↪ idxmax()
result_df = monthly_sales.loc[highest_sales_month].
    ↪ reset_index()
```

Explanation: We first calculate the total Amount per Year and Month. Then, for each year, we use idxmax to find the index of the month with the highest sales. Finally, we extract the corresponding values and reset the index.
