Data Reshaping with Pandas Pivot Methods

A Comprehensive Guide for Data Analysis & Visualization

📊 Introduction to Data Reshaping

Data reshaping is the process of transforming data from one format to another to make it more suitable for analysis. In pandas, the primary tools for this are **pivot()** and **pivot_table()** functions.

Long Format (Current Data)

Company	Year	GRI	ESRS
A2A S.P.A.	2022	1	0
A2A S.P.A.	2023	1	0
A2A S.P.A.	2024	1	1
ENEL S.P.A.	2022	1	0

Characteristics: Repeated company entries, time as rows

Wide Format (After Pivot)

Company	GRI_2022	GRI_2023	GRI_2024	ESRS_2024
A2A S.P.A.	1	1	1	1
ENEL S.P.A.	1	1	1	1

Characteristics: One row per company, time as columns

🛠️ Understanding Pivot Functions

pivot() - Basic Reshaping

```
df.pivot(index, columns, values)
```

Parameters:

- **index**: Column(s) to use as row identifiers
- **columns**: Column whose values become new column headers
- **values**: Column(s) to populate the new columns

pivot_table() - Advanced with Aggregation

```
df.pivot_table(values, index, columns, aggfunc, fill_value)
```

Additional Parameters:

- **aggfunc**: Aggregation function for duplicates ('mean', 'sum', 'first')
- **fill_value**: Value to replace missing data
- **margins**: Add subtotals (True/False)

💼 Practical Example: Sustainability Reporting Data

Original Dataset Structure

```
# Check original data structure print("Original Data Shape:", df.shape) print("Companies:", df['Name'].nunique()) print("Years:", df['Anno'].unique())
```

Output:

Original Data Shape: (750, 54) 750 rows, 54 columns

Companies: 250 250 unique companies

Years: [2022 2023 2024] 3 years of data

Applying Pivot Transformation

```
# Reshape sustainability reporting data df_wide = df.pivot_table( index=['Name', 'ATECO_2007_descrizione'], # Grouping variables columns='Anno', # Years become columns values=['GRI', 'ESRS', 'SASB'], # Metrics to spread aggfunc='first', # Handle any duplicates fill_value=0 # Fill missing with 0 ).reset_index()
```

Column Name Cleaning

```
# Flatten MultiIndex columns df_wide.columns = ['_'.join(filter(None, col)).strip() for col in df_wide.columns.values] # Rename identifier columns df_wide = df_wide.rename(columns={ 'Name_': 'Company', 'ATECO_2007_descrizione_': 'Industry' })
```

🚀 Benefits of Wide Format

📈 **Time Series Analysis**
Compare year-over-year changes easily with all data in one view

💾 **Reduced Redundancy**
From 750 rows to 250 rows (66% reduction) - one row per company

👁️ **Improved Readability**
All related data visible without scrolling through repeated entries

⚡ **Efficient Analysis**
Perfect for statistical operations and visualization

Visualization Example

```
# Easy year comparison plotting import matplotlib.pyplot as plt # Plot GRI adoption over years gri_columns = ['GRI_2022', 'GRI_2023', 'GRI_2024'] df_wide[gri_columns].mean().plot(kind='bar') plt.title('GRI Adoption Over Time') plt.show()
```

⚠️ Troubleshooting Guide

Common Errors & Solutions

Error: "Index contains duplicate entries"

```
# Solution: Use pivot_table with aggregation df.pivot_table( index=['Company', 'Industry'], columns='Year', values='Metric', aggfunc='first' # or 'mean', 'max', etc. )
```

Error: "No numeric types to aggregate"

```
# Solution: Ensure values are numeric df['Metric'] = pd.to_numeric(df['Metric'], errors='coerce')
```

Issue: MultiIndex columns confusion

```
# Solution: Flatten column names df.columns = [f'{level0}_{level1}' if level1 else level0 for level0, level1 in df.columns]
```

🎯 Best Practices

✅ Do This

- Check for duplicates before pivoting
- Use descriptive column names
- Handle missing values appropriately
- Validate results against original data
- Use pivot_table() for real-world data

❌ Avoid This

- Ignoring duplicate entries
- Using confusing column names
- Leaving NaN values unhandled
- Not testing the transformation
- Using pivot() with potential duplicates

🎓 Key Takeaways

💡 **Core Concepts:**

Pivoting transforms data from long to wide format
pivot_table() is more robust than pivot() for real-world data
Wide format enables efficient time-series analysis
Proper column naming is crucial for usability
Always validate your transformations

Next Steps to Master

- Explore **melt()** for reverse transformations
- Practice with different **aggregation functions**
- Learn about **MultiIndex operations**
- Experiment with **real datasets**