Pandas Column Repositioning Guide

> **Note:** This guide covers various scenarios for repositioning columns in pandas DataFrames, from simple swaps to complex multi-column reorganizations.

1. Swap Two Specific Columns

Method 1: Using Column Names (Most Common)

```
def swap_columns(df, col1, col2):
    cols = df.columns.tolist()
    idx1, idx2 = cols.index(col1), cols.index(col2)
    cols[idx1], cols[idx2] = cols[idx2], cols[idx1]
    return df[cols]

# Usage
df = swap_columns(df, 'column_A', 'column_B')
```

Method 2: Direct Assignment

```
def swap_columns_direct(df, col1, col2):
    df[[col1, col2]] = df[[col2, col1]]
    return df

# Usage
df = swap_columns_direct(df, 'column_A', 'column_B')
```

2. Move Column to Specific Position

Move to Exact Position

```
def move_column(df, column_name, new_position):
    """
    Move column to specific index position
    """
    cols = df.columns.tolist()
    cols.remove(column_name)
    cols.insert(new_position, column_name)
    return df[cols]

# Move column to front
def move_to_front(df, column_name):
    return move_column(df, column_name, 0)

# Move column to end
def move_to_end(df, column_name):
    return move_column(df, column_name, len(df.columns)-1)

# Usage examples
df = move_column(df, 'important_col', 2)  # Move to 3rd position (0-
indexed)
df = move_to_front(df, 'priority_col')
df = move_to_end(df, 'metadata_col')
```

3. Move Column Before/After Another Column

Move Column Before Reference Column

```
def move_column_before(df, column_to_move, reference_column):
    """
    Move column_to_move before reference_column
    """
    cols = df.columns.tolist()
    cols.remove(column_to_move)
    ref_idx = cols.index(reference_column)
    cols.insert(ref_idx, column_to_move)
    return df[cols]
```

Move Column After Reference Column

```
def move_column_after(df, column_to_move, reference_column):
    """
    Move column_to_move after reference_column
    """
    cols = df.columns.tolist()
    cols.remove(column_to_move)
    ref_idx = cols.index(reference_column) + 1
    cols.insert(ref_idx, column_to_move)
    return df[cols]

# Usage
df = move_column_before(df, 'new_col', 'existing_col')
df = move_column_after(df, 'new_col', 'existing_col')
```

4. Move Multiple Columns Together

Move Multiple Columns as a Group

```
def move_multiple_columns(df, columns_to_move, new_position,
keep_order=True):
    """
    Move multiple columns as a group to new position
    """
    cols = df.columns.tolist()

    # Remove columns to move
    for col in columns_to_move:
        cols.remove(col)

    # Insert them at new position
    if keep_order:
        # Insert in original order
        for i, col in enumerate(reversed(columns_to_move)):
            cols.insert(new_position, col)
    else:
        # Insert in the order provided
        for col in reversed(columns_to_move):
            cols.insert(new_position, col)

    return df[cols]
```

Move Multiple Columns Before Reference Column

```
def move_multiple_before(df, columns_to_move, reference_column):
    """
    Move multiple columns before reference column
    """
    cols = df.columns.tolist()
    ref_idx = cols.index(reference_column)

    # Remove columns to move
    for col in columns_to_move:
        cols.remove(col)

    # Insert before reference column
    for col in reversed(columns_to_move):
        cols.insert(ref_idx, col)

    return df[cols]
```

Move Multiple Columns After Reference Column

```
def move_multiple_after(df, columns_to_move, reference_column):
    """
    Move multiple columns after reference column
    """
    cols = df.columns.tolist()
    ref_idx = cols.index(reference_column) + 1

    # Remove columns to move
    for col in columns_to_move:
        cols.remove(col)

    # Insert after reference column
    for col in reversed(columns_to_move):
        cols.insert(ref_idx, col)

    return df[cols]

# Usage
df = move_multiple_before(df, ['col1', 'col2', 'col3'], 'target_col')
```

5. Advanced: Flexible Column Reordering

Advanced Column Reordering Function

```
def reorder_columns(df, column_order, position='exact'):
    """
    Advanced column reordering

    Parameters:
    - column_order: list of columns to reorder
    - position: 'exact', 'before', 'after', or index position
    """
    if position == 'exact':
        # Set exact column order (keeping other columns in original order)
        remaining_cols = [col for col in df.columns if col not in
column_order]
        new_cols = column_order + remaining_cols
        return df[new_cols]

    elif position in ['before', 'after']:
        raise ValueError("Use move_multiple_before/after for
'before'/'after' positioning")

    else:
        # Position is an integer index
        return move_multiple_columns(df, column_order, position)
```

Organize Columns by Logical Groups

```
def organize_columns_by_groups(df, column_groups, group_order):
    """
    Organize columns by logical groups

    Parameters:
    - column_groups: dict {group_name: [list_of_columns]}
    - group_order: list defining order of groups
    """
    final_column_order = []

    for group in group_order:
        if group in column_groups:
            final_column_order.extend(column_groups[group])
```

```
    # Add any columns not in groups at the end
    remaining_cols = [col for col in df.columns if col not in
final_column_order]
    final_column_order.extend(remaining_cols)

    return df[final_column_order]

# Usage example for grouping
column_groups = {
    'identifiers': ['id', 'name', 'email'],
    'dates': ['created_at', 'updated_at', 'deleted_at'],
    'metrics': ['score', 'rating', 'value']
}
group_order = ['identifiers', 'metrics', 'dates']

df = organize_columns_by_groups(df, column_groups, group_order)
```

6. Handling Large DataFrames Efficiently

Efficient Method for Large DataFrames

```
def efficient_column_reorder(df, new_column_order):
    """
    Efficient method for large DataFrames - avoids creating new DataFrame
    """
    # This modifies the DataFrame in-place
    df.columns = new_column_order
    # Note: This requires you to specify the complete column order
```

Apply Multiple Operations at Once

```
def smart_column_move(df, operations):
    """
    Apply multiple column move operations at once

    Parameters:
    - operations: list of tuples (method, *args)
      Example: [('before', 'colA', 'colB'), ('after', 'colC', 'colD')]
    """
```

```
    temp_df = df.copy()

    for operation in operations:
        method = operation[0]
        if method == 'before':
            temp_df = move_column_before(temp_df, operation[1],
operation[2])
        elif method == 'after':
            temp_df = move_column_after(temp_df, operation[1],
operation[2])
        elif method == 'swap':
            temp_df = swap_columns(temp_df, operation[1], operation[2])
        elif method == 'to_position':
            temp_df = move_column(temp_df, operation[1], operation[2])

    return temp_df

# Usage for multiple operations
operations = [
    ('before', 'new_col1', 'existing_col1'),
    ('after', 'new_col2', 'existing_col2'),
    ('swap', 'colA', 'colB')
]
df = smart_column_move(df, operations)
```

7. One-Liner Solutions for Common Cases

```
# Move single column to front
df = df[['column_name'] + [col for col in df.columns if col != 'column_name']]

# Move single column to end
df = df[[col for col in df.columns if col != 'column_name'] + ['column_name']]

# Move multiple columns to front
cols_to_front = ['col1', 'col2', 'col3']
df = df[cols_to_front + [col for col in df.columns if col not in
cols_to_front]]

# Move multiple columns to end
cols_to_end = ['col1', 'col2', 'col3']
df = df[[col for col in df.columns if col not in cols_to_end] + cols_to_end]
```

Complete Example with All Methods

Working Example

```
import pandas as pd
import numpy as np

# Create sample DataFrame
df = pd.DataFrame({
    'A': range(10),
    'B': range(10, 20),
    'C': range(20, 30),
    'D': range(30, 40),
    'E': range(40, 50),
    'F': range(50, 60)
})

print("Original columns:", df.columns.tolist())

# Swap columns B and D
df = swap_columns(df, 'B', 'D')
print("After swapping B and D:", df.columns.tolist())

# Move column A to position 3
df = move_column(df, 'A', 3)
print("After moving A to position 3:", df.columns.tolist())

# Move column C before E
df = move_column_before(df, 'C', 'E')
print("After moving C before E:", df.columns.tolist())

# Move multiple columns B and D after F
df = move_multiple_after(df, ['B', 'D'], 'F')
print("After moving B,D after F:", df.columns.tolist())
```

Important: Always test these functions on a copy of your DataFrame first when working with important data. Some methods create new DataFrames while others modify in-place.