Part 1: Pandas Manipulation Cheatsheet

Here is a quick-reference table for common `pandas` data manipulation tasks, focusing on the areas you requested.

Category	Operation/ Goal	Example Code
Data Cleaning	**Standardize Column Names**	`df.columns = df.columns.str.lower().str.replace(' ', '_')`
	Rename Specific Columns	`df.rename(columns={'old_name': 'new_name', 'typo_name': 'co`
	Change Column Data Type	`df['col'] = df['col'].astype(int)` `df['col'] = pd.to_numeric(df['col'], errors='coerce')`
	Clean String Data	`df['price'] = df['price'].str.replace('$', '').astype(float`
Filtering Data	**Filter Rows by a Single Condition**	`df[df['age'] > 30]`

Category	Operation/ Goal	Example Code
	Filter by Multiple Conditions (AND)	`df[(df['age'] > 30) & (df['country'] == 'USA')]`
	Filter by Multiple Conditions (OR)	`df[(df['category'] == 'A') | (df['category'] == 'B')]`
	Filter using a List of Values	`df[df['product_id'].isin([101, 102, 105])]`
Combining Data	**Combine DataFrames Vertically (stacking rows)**	`pd.concat([df1, df2, df3], ignore_index=True)`
	Combine DataFrames Horizontally (joining columns)	`pd.merge(df_left, df_right, on='key_column', how='inner')`
Grouping Data	**Group by a Column and Aggregate**	`df.groupby('category')['sales'].sum()`

Category	Operation/ Goal	Example Code
	Group by Multiple Columns	`df.groupby(['region', 'category'])['sales'].mean()`
	Multiple Aggregations	`df.groupby('category').agg(total_sales=('sales', 'sum'), av`

Part 2: The Practical Case Study

Scenario: You are a data scientist at a global retail company. The sales data for the first quarter is spread across multiple CSV files from different regional offices (North America, Europe, Asia). Additionally, there are separate files for product information and customer details. The data is messy. Your task is to clean and consolidate all this information into a single, master DataFrame for analysis.

The Challenge:

1. Combine sales data from three regional files.
2. Standardize the column names, which have typos and different casing.
3. Clean and convert the `Total Sale` column to a numeric type.
4. Merge the consolidated sales data with product and customer information.
5. Create a final, clean DataFrame ready for analysis.

The 7 "Tables" (as CSV files):

Let's first create the dummy data. In a real scenario, these would be separate `.csv` files.

```
# Setup: Create the dummy CSV files for our case study
import pandas as pd
import os

# Create a directory for our data
if not os.path.exists('sales_data'):
    os.makedirs('sales_data')

# --- Sales Files (with messy columns) ---
sales_na_data = """sale_id,product_id,customer_id,Total Sale
101,A54,C1,$150.50
102,B12,C2,$75.00
103,A54,C3,$140.25
"""
with open('sales_data/sales_north_america.csv', 'w') as f:
    f.write(sales_na_data)

sales_eu_data = """Sale ID,Product ID,CustomerID,Total Sale
201,C78,C4,$25.99
202,B12,C5,$80.00
"""
with open('sales_data/sales_europe.csv', 'w') as f:
    f.write(sales_eu_data)

sales_asia_data = """sale id,product id,customer id,Total_Sale
301,A54,C6,$155.00
302,D99,C7,$200.10
"""
with open('sales_data/sales_asia.csv', 'w') as f:
    f.write(sales_asia_data)

# --- Supporting Info Files ---
products_data = """product_id,product_name,category
A54,Laptop,Electronics
B12,Mouse,Electronics
C78,T-Shirt,Apparel
D99,Keyboard,Electronics
"""
with open('sales_data/products.csv', 'w') as f:
```

```
    f.write(products_data)

customers_data = """id,first_name,last_name,country
C1,John,Doe,USA
C2,Jane,Smith,Canada
C3,Peter,Jones,USA
C4,Hans,Schmidt,Germany
C5,Isabelle,Dubois,France
C6,Kenji,Tanaka,Japan
C7,Li,Wei,China
"""
with open('sales_data/customers.csv', 'w') as f:
    f.write(customers_data)

# --- Bonus Files for another Concat Example ---
promotions_q1_data = """promo_id,discount_percent
P1,10
P2,15
"""
with open('sales_data/promos_q1.csv', 'w') as f:
    f.write(promotions_q1_data)

promotions_q2_data = """promo_id,discount_percent
P3,20
P4,5
"""
with open('sales_data/promos_q2.csv', 'w') as f:
    f.write(promotions_q2_data)

print("Dummy CSV files created successfully in the 'sales_data' directory.")
```

The Code Challenge: Solution

Here is the step-by-step Python code to solve the challenge, applying the concepts from the cheatsheet.

```python
# Import pandas
import pandas as pd

# --- Step 1: Load all regional sales files ---
sales_files = [
    'sales_data/sales_north_america.csv',
    'sales_data/sales_europe.csv',
    'sales_data/sales_asia.csv'
]
# Load them into a list of DataFrames
list_of_dfs = [pd.read_csv(file) for file in sales_files]

# --- Step 2: Standardize Column Names ---
# Let's inspect the columns of the first DataFrame to see the problem
print("Original columns of first file:", list_of_dfs[0].columns)
print("Original columns of second file:", list_of_dfs[1].columns)

# Create a clean, standardized list of column names
clean_columns = ['sale_id', 'product_id', 'customer_id', 'total_sale']

# Loop through the list of DataFrames and apply the new column names
for df in list_of_dfs:
    df.columns = clean_columns

# Let's verify the change
print("\nCleaned columns of first file:", list_of_dfs[0].columns)

# --- Step 3: Concatenate the sales DataFrames ---
sales_df = pd.concat(list_of_dfs, ignore_index=True)

print("\n--- Consolidated Sales DataFrame (Before Cleaning) ---")
print(sales_df)
print("\nData types before cleaning:")
print(sales_df.info())

# --- Step 4: Clean the 'total_sale' column and change data type ---
# The 'total_sale' column is an 'object' (string) because of the '$'
# We use .str.replace() to remove the character and .astype() to convert
sales_df['total_sale'] = sales_df['total_sale'].str.replace('$', '').astype(float)
```

```python
# Also, let's fix the customer_id in the customers table before merging
# The column is called 'id' but we need it to be 'customer_id' to merge
customers_df = pd.read_csv('sales_data/customers.csv')
customers_df.rename(columns={'id': 'customer_id'}, inplace=True)

# --- Step 5: Load supporting files and merge ---
products_df = pd.read_csv('sales_data/products.csv')

# Merge sales with products
master_df = pd.merge(sales_df, products_df, on='product_id', how='left')

# Merge the result with customers
master_df = pd.merge(master_df, customers_df, on='customer_id', how='left')

# --- Step 6: Final Verification ---
print("\n--- Final Master DataFrame ---")
print(master_df)

print("\nFinal data types and info:")
master_df.info()

# --- Example of GroupBy from the cheatsheet ---
print("\n--- Example Analysis: Total Sales by Country ---")
sales_by_country = master_df.groupby('country')['total_sale'].agg(['sum', 'mean', 'count'])
print(sales_by_country)
```

Summary of Actions Taken:

1. **Load Multiple Files:** We loaded 3 separate CSV files into a list of pandas DataFrames.
2. **Standardize Columns (`.columns`)**: We identified inconsistent column names (`sale_id`, `Sale ID`, `sale id`) and programmatically assigned a single, clean set of names to all DataFrames before combining them. We also used `.rename()` on the customers table.
3. **Concatenate (`pd.concat`)**: We stacked the three cleaned sales DataFrames vertically into one, creating a single source of truth for sales transactions.
4. **Clean and Convert Type (`.str.replace`, `.astype`)**: The `total_sale` column

was a string with dollar signs. We removed the character and converted the column to a `float` type, making it usable for mathematical operations.

5. **Merge (`pd.merge`)**: We performed two database-style "left joins" to enrich our sales data. We first added product details (name, category) by merging on `product_id`, and then added customer details (name, country) by merging on `customer_id`.
6. **Group and Aggregate (`.groupby`, `.agg`)**: As a final bonus, we demonstrated how the newly cleaned data could be used for analysis by grouping by `country` to calculate total and average sales.

This case study directly mirrors a very common real-world task and showcases how the functions in the cheatsheet work together to transform messy, disparate data into a powerful, analysis-ready dataset.

Of course! This is an excellent topic. Mastering `pandas` data manipulation is the foundation of almost every data science task.

Let's build a comprehensive guide, starting with a clear cheatsheet and then applying it to a realistic, multi-file case study.

Pandas Data Manipulation Cheatsheet

Here is a quick-reference table for common data wrangling tasks in pandas.

Task/Goal	Pandas Function/ Method	Example Code
Grouping & Aggregating	`df.groupby()`	`df.groupby('category')['sales'].sum()`
		`df.groupby(['region', 'product']).agg({'sales': ['m`
Renaming Columns	`df.rename()`	`df.rename(columns={'old_name': 'new_name', 'another`
	Programmatic Cleaning	`df.columns = [col.lower().strip().replace(' ', '_')`
Changing	`df['col'].astype()`	`df['price'] = df['price'].astype(float)`

Task/Goal	Pandas Function/ Method	Example Code
Data Types		`df['user_id'] = df['user_id'].astype(str)`
	To Numeric (with errors)	`pd.to_numeric(df['col'], errors='coerce')` (Turns nor
	To Datetime	`pd.to_datetime(df['date_col'])`
Combining DataFrames	`pd.concat()` (Stacking)	`pd.concat([df1, df2, df3], ignore_index=True)`
Filtering (Boolean Indexing)	`[]` with boolean series	`df[df['sales'] > 1000]` `df[(df['region'] == 'West') & (df['sales'] > 500)]`
Filtering (Query Method)	`df.query()`	`df.query('sales > 1000')` `df.query("region == 'West' and sales > 500")`
Handling String Data	`.str` accessor	`df['price_str'].str.replace('$', '').str.replace(',`

The Practical Application: Merging Messy Sales Reports

Scenario: You are a data analyst for a retail company. Each region emails you their sales report as a separate CSV file. Your task is to combine these 7 files into a single, clean master dataset for analysis. However, the files are messy: column names have typos and inconsistent formatting, and data types are incorrect.

The 7 Messy "CSV" Files (Simulated as DataFrames)

Here is the Python code to generate our 7 messy DataFrames.

```python
import pandas as pd
import numpy as np

# File 1: New York (The "correct" reference format)
sales_ny = pd.DataFrame({
    'Date': ['2023-10-01', '2023-10-01', '2023-10-02'],
    'Product_ID': ['A101', 'B202', 'A101'],
    'Units_Sold': [5, 3, 2],
    'Sale_Amount': [50.00, 45.00, 20.00]
})

# File 2: Los Angeles (Inconsistent column name)
sales_la = pd.DataFrame({
    'Date': ['2023-10-01', '2023-10-03'],
    'Product_ID': ['C303', 'B202'],
    'Units_Sold': [10, 4],
    'Sale Amount': [150.00, 60.00] # <-- Note the space in 'Sale Amount'
})

# File 3: Chicago (Data type is string with symbols)
sales_chicago = pd.DataFrame({
    'Date': ['2023-10-02', '2023-10-03'],
    'Product_ID': ['A101', 'C303'],
    'Units_Sold': [1, 5],
    'Sale_Amount': ['$10.00', '$75.50'] # <-- String with '$'
})

# File 4: Houston (Another column name typo)
sales_houston = pd.DataFrame({
    'Date': ['2023-10-02'],
    'product id': ['B202'], # <-- Lowercase and space
    'Units_Sold': [8],
    'Sale_Amount': [120.00]
})

# File 5: Seattle (Units sold is a float)
sales_seattle = pd.DataFrame({
    'Date': ['2023-10-04'],
    'Product_ID': ['C303'],
```

```
    'Units_Sold': [2.0], # <-- Float instead of int
    'Sale_Amount': [30.00]
})

# File 6: Boston (Inconsistent date format and column name)
sales_boston = pd.DataFrame({
    'sale_date': ['10/01/2023', '10/04/2023'], # <-- Different format and name
    'Product_ID': ['A101', 'B202'],
    'Units_Sold': [3, 6],
    'Sale_Amount': [30.00, 90.00]
})

# File 7: Miami (Missing data)
sales_miami = pd.DataFrame({
    'Date': ['2023-10-05', '2023-10-05'],
    'Product_ID': ['A101', 'C303'],
    'Units_Sold': [4, np.nan], # <-- Missing units
    'Sale_Amount': [40.00, np.nan] # <-- Missing sales
})

# Put all our "files" into a list
all_sales_files = [
    sales_ny, sales_la, sales_chicago, sales_houston,
    sales_seattle, sales_boston, sales_miami
]
```

The Code Challenge: Cleaning and Combining the Data

Your Task: Write the Python code to perform the following steps:

1. **Standardize Column Names:** Create a uniform format for all column names across all DataFrames (e.g., `sale_date`, `product_id`, `units_sold`, `sale_amount`).
2. **Combine Data:** Concatenate all 7 DataFrames into a single master DataFrame.
3. **Correct Data Types:**
    - Convert the date column to a proper `datetime` object.

    - Ensure `sale_amount` is a `float`. This will require removing characters like '$'.
    - Ensure `units_sold` is a nullable integer type to handle missing values.
4. **Perform Analysis:** Use the final, clean DataFrame to answer:
    - What were the total sales for each `Product_ID`?
    - Show all sales records where more than 5 units were sold.

Solution

Here is the step-by-step solution implementing the concepts from our cheatsheet.

```python
# Create a list to hold the cleaned dataframes
cleaned_dfs = []

# --- Step 1: Standardize Column Names and Clean Each DataFrame ---
# We loop through each dataframe and apply a consistent cleaning logic.

for df in all_sales_files:
    # Make a copy to avoid SettingWithCopyWarning
    df_clean = df.copy()

    # Programmatic column renaming
    # Renames 'Sale Amount' to 'sale_amount', 'product id' to 'product_id', etc.
    df_clean.columns = [
        col.lower().strip().replace(' ', '_').replace('sale_amount', 'sale_amount').repla
        for col in df_clean.columns
    ]
    # A specific rename for 'product_id' just in case the above is not perfect
    if 'product_id' not in df_clean.columns and 'product id' in df_clean.columns:
         df_clean.rename(columns={'product id': 'product_id'}, inplace=True)

    # A specific rename for the unique date column name
    if 'sale_date' not in df_clean.columns and 'date' in df_clean.columns:
        df_clean.rename(columns={'date': 'sale_date'}, inplace=True)

    cleaned_dfs.append(df_clean)

# --- Step 2: Combine Data ---
# Use pd.concat to stack all the cleaned dataframes vertically
master_sales = pd.concat(cleaned_dfs, ignore_index=True)

print("--- Data After Concatenation (Types are still messy) ---")
print(master_sales)
print("\n")
master_sales.info()


# --- Step 3: Correct Data Types in the Master DataFrame ---

# Convert 'sale_date' to datetime objects
```

```
master_sales['sale_date'] = pd.to_datetime(master_sales['sale_date'])

# Clean 'sale_amount' by removing '$' and converting to float
# The .astype(str) is a safety measure if a column contains mixed types like numbers and s
master_sales['sale_amount'] = master_sales['sale_amount'].astype(str).str.replace('$', ''
# 'coerce' will turn any problematic values (like 'nan' string) into a real NaN (Not a Nu
master_sales['sale_amount'] = pd.to_numeric(master_sales['sale_amount'], errors='coerce')

# Convert 'units_sold' to a nullable integer type.
# 'Int64' (capital 'I') can handle NaN values, whereas 'int' cannot.
master_sales['units_sold'] = master_sales['units_sold'].astype('Int64')


print("\n\n--- Final Cleaned Master DataFrame ---")
print(master_sales.head())
print("\n")
master_sales.info()


# --- Step 4: Perform Analysis ---

# 4a: What were the total sales for each Product_ID? (Using groupby)
total_sales_by_product = master_sales.groupby('product_id')['sale_amount'].sum()
print("\n\n--- Total Sales by Product ID ---")
print(total_sales_by_product)


# 4b: Show all sales records where more than 5 units were sold. (Using .query())
large_sales = master_sales.query('units_sold > 5')
print("\n\n--- Sales Records with More Than 5 Units Sold ---")
print(large_sales)
```

This walkthrough covers a highly realistic data cleaning pipeline, leveraging the core pandas functions for renaming, type casting, string manipulation, concatenation, and finally, analysis with `groupby` and `query`.