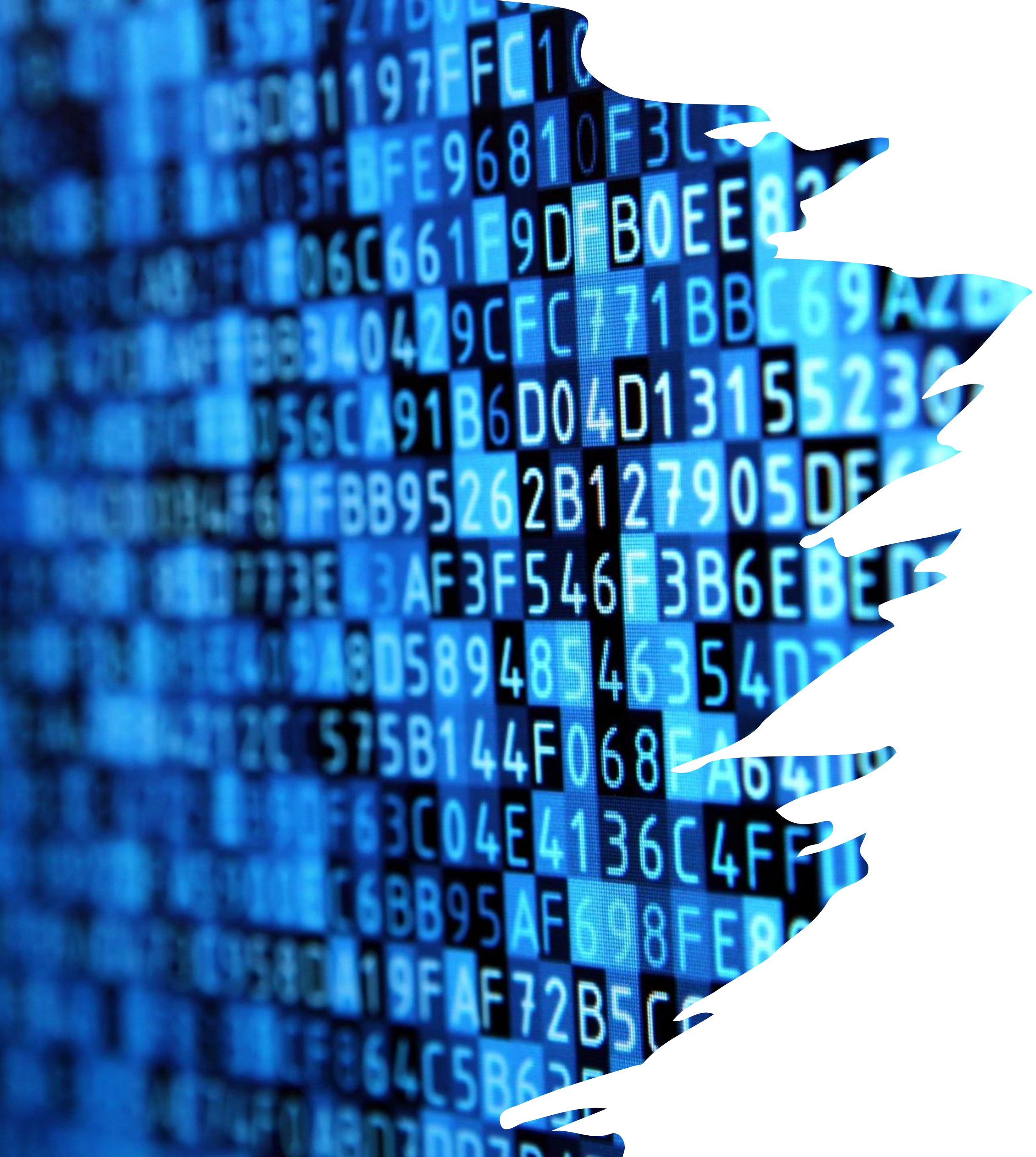
Data cleaning and transformation

Agenda

- Handle Missing Data & Duplicates
- Merge, Join, Concatenate DataFrames
- Apply Aggregation & GroupBy
- Generate Summary Stats & Insights



What is "Messy" Data

- Real-world data is almost never clean. It often has:
- **Missing Data:** Represented as NaN (Not a Number). This can be from data entry errors, or data that was never collected.
- **Duplicates:** The same row appearing more than once.

Why does it matter?

Missing data can break your calculations (e.g., mean(), sum()).

Duplicates can skew your results (e.g., making sales look higher than they are).

Sample data

- data = {'A': [1, 2, np.nan, 4],
  'B': [5, np.nan, 7, 8],
‘ C': [9, 10, 11, 12]}

df = pd.DataFrame(data)

print(f"Original Data:\n{df}\n")

```
df.isnull()
```

Find missing data (returns True/False)

count of missing data per column

```
df.isnull().sum()
```

Drop rows with any NaN

df.dropna()

Fill NaN with a value

```
df.fillna(value=0)
```

Fill with the column's mean

- mean_b = df['B'].mean()
- df['B'] = df['B'].fillna(value=mean_b)

Finding & Handling Duplicates

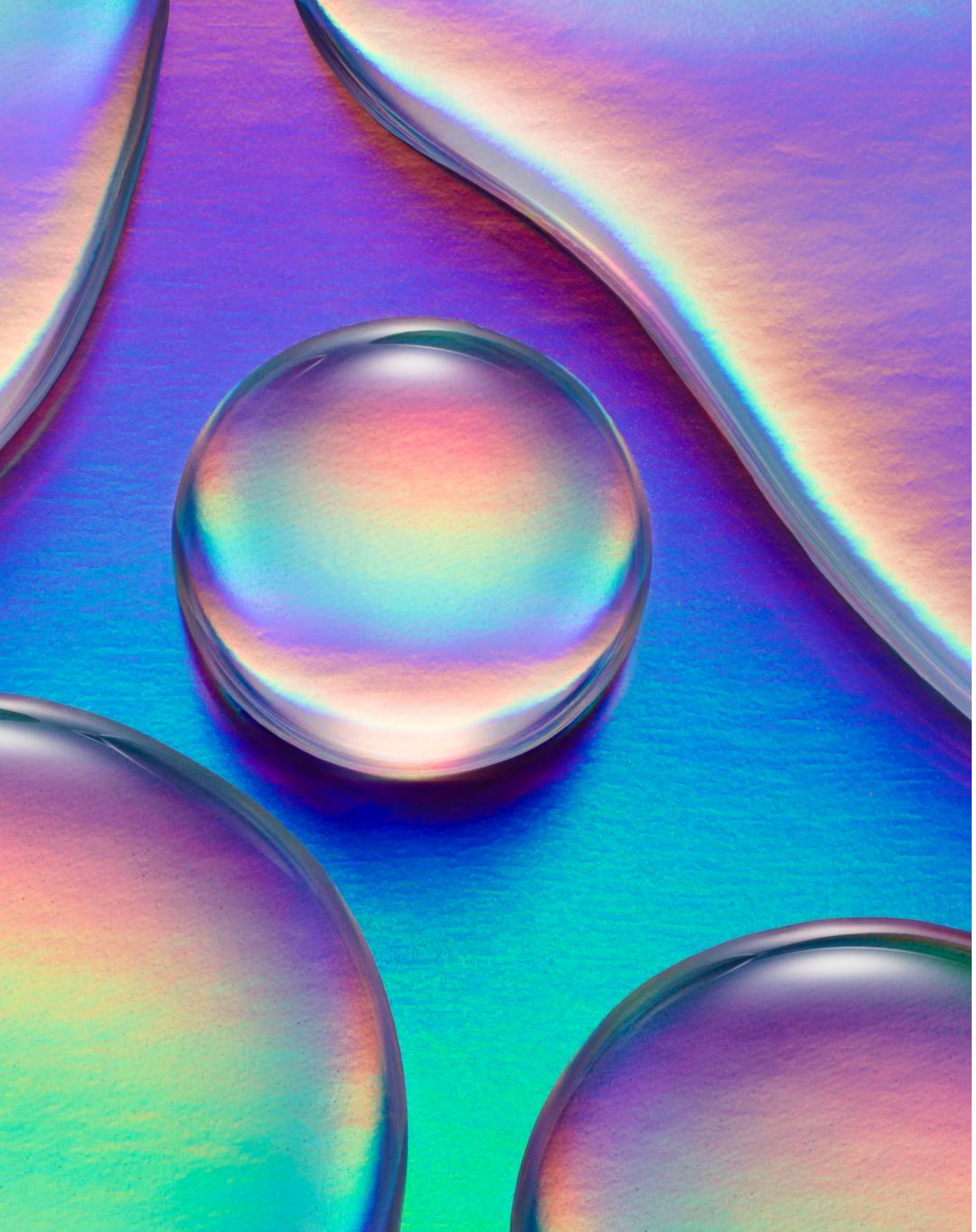
- DataFrame.duplicated(subset=None, keep='first)

- Return boolean Series denoting duplicate rows.

duplicated(subset=None, keep='first')

- **Subset *column label or sequence of labels, optional***
- Only consider certain columns for identifying duplicates, by default use all of the columns.

- **Keep *{'first', 'last', False}, default 'first'***
- Determines which duplicates (if any) to mark.
- first : Mark duplicates as True except for the first occurrence.
- last : Mark duplicates as True except for the last occurrence.
- False : Mark all duplicates as True.



Drop duplicate rows

- **DataFrame.drop_duplicates(*subset*=*None*, *, *keep*=*'first'*, *inplace*=*False*, *ignore_index*=*False*)**

- df.drop_duplicates()

.drop_duplicates()

- **inplace*bool, default False***
- Whether to modify the DataFrame rather than creating a new one.

- **ignore_index*bool, default False***
- If True, the resulting axis will be labeled 0, 1, …, n - 1.

Hands-On

```
raw_data = {
    'Employee': ['John', 'Anna', 'Peter', 'John', 'Linda', 'Anna'],
    'Department': ['Sales', 'IT', 'Sales', 'Sales', 'HR', 'IT'],
    'Salary': [50000, 75000, 52000, 50000, 60000, np.nan]
}
emp_df = pd.DataFrame(raw_data)
```

- Task 1:Find and print the total count of missing values for each column.
- Task 2:Drop the duplicate rows (Hint: 'John' and 'Anna' appear twice).
- Task 3 (Bonus):Fill the missing Salary with the average salary of the other employees.

Aggregation and GroupBy

“Split-Apply-Combine" Strategy

- Split: You groupby color (whites, darks, colors).
- Apply: You apply a function to each group
- Combine: You combine the results

Split

Split: df.groupby('Department') -> splits data into groups (Sales, IT, HR).

Apply

['Salary'].mean() -> applies the "mean" function to the 'Salary' of each group.

Example

- data = {'Employee': ['John', 'Anna', 'Peter', 'Linda'], 'Department': ['Sales', 'IT', 'Sales', 'HR'], 'Salary': [50000, 75000, 52000, 60000]}
- df = pd.DataFrame(data)
- # 1. Find the AVERAGE salary for each department
- # 2. Find the SUM of salaries for each department
- # 3. Find the COUNT of employees in each department
- # 4. Using .agg() for multiple operations

aggregate

- it's used to apply one or more aggregation functions to your grouped data.

- Apply **multiple functions** at once (e.g., mean, max, count)

- Rename the output columns

Hands-On Code: Groupby

- sales_data = {
-     'Region': ['North', 'South', 'North', 'South', 'East', 'East'],
-     'Product': ['A', 'A', 'B', 'B', 'A', 'B'],
-     'Amount': [100, 150, 200, 250, 50, 300]
- }
- df_sales = pd.DataFrame(sales_data)
- 1. What is the *total* sales Amount for each Region?
- 2. What is the *average* sales Amount for each Product?
- 3. What is the *total* sales Amount for each *combination* of Region and Product? (Hint: You can groupby a list: ['Region', 'Product'])

Getting Quick Insights

- Now that your data is clean and organized, you can get high-level insights.

.describe() - Stats for all numeric columns

- Gives count, mean, std dev, min, max, quartiles

.value_counts()

Frequency of unique values #
Great for categorical data

.info()

.corr() - Correlation

- Shows how strongly numeric columns are related
- Need more numeric columns for this to be useful
- df.corr()