Pandas Tutorials

**Data Cleaning in Pandas**

Data cleaning is an essential step in data preprocessing to ensure accuracy and consistency. Here are some articles to know more about it:

* [Handling Missing Data](https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/)

**1. Using isnull()**

[**isnull()**](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/) returns a DataFrame of Boolean value where **True** represents missing data (**NaN**). This is simple if we want to find and fill missing data in a dataset.

**2. Checking for Non-Missing Values Using notnull()**

[**notnull()**](https://www.geeksforgeeks.org/python-pandas-series-notnull/) function returns a DataFrame with Boolean values where **True**indicates non-missing (valid) data. This function is useful when we want to focus only on the rows that have valid, non-missing values.

**1. Using fillna()**

[**fillna()**](https://www.geeksforgeeks.org/python-pandas-dataframe-fillna-to-replace-null-values-in-dataframe/) used to replace missing values (NaN) with a given value. Lets see various example for this.

**Example2: Fill with Previous Value (Forward Fill)**

The **pad**method is used to fill missing values with the previous value.

df.fillna(method='pad')

FFILL = wine\_data.ffill(inplace=False, limit=None, downcast=None)

BFILL = wine\_data.bfill(inplace=False, limit=None, downcast=None)

**Example3: Fill with Next Value (Backward Fill)**

The **bfill**function is usedto fill it with the next value.

df.fillna(method='bfill')

Now we are going to fill all the null values in Gender column with "No Gender"

d["Gender"].fillna('No Gender', inplace = **True**)

d[10:25]

**2. Using replace()**

Use[**replace()**](https://www.geeksforgeeks.org/python-pandas-series-str-replace-to-replace-text-in-a-series/)function to replace NaN values with a specific value.

Now, we are going to replace the all **NaN**value in the data frame with **-99** value.

data.replace(to\_replace=np.nan, value=-99)

**3. Using interpolate()**

The[**interpolate()**](https://www.geeksforgeeks.org/pandas-dataframe-interpolate/)function fills missing values using interpolation techniques such as the linear method.

**Example**

Let’s interpolate the missing values using Linear method. This method ignore the index and consider the values as equally spaced.

df.interpolate(method ='linear', limit\_direction ='forward')

**Dropping Missing Values in Pandas**

The **[dropna()](https://www.geeksforgeeks.org/python-pandas-dataframe-dropna/" \t "_blank)** function used to removes rows or columns with NaN values. It can be used to drop data based on different conditions.

**1. Dropping Rows with At Least One Null Value**

Remove rows that contain at least one missing value.

**Example**

**3. Dropping Columns with At Least One Null Value**

To remove columns that contain at least one missing value we use **dropna(axis=1).**

**Example**

dict = {'First Score': [100, np.nan, np.nan, 95],

'Second Score': [30, np.nan, 45, 56],

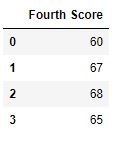
'Third Score': [52, np.nan, 80, 98],

'Fourth Score': [60, 67, 68, 65]}

df = pd.DataFrame(dict)

df.dropna(axis=1)

**Output**



**4. Dropping Rows with Missing Values in CSV Files**

When working with CSV files, we can drop rows with missing values using **dropna()**.

**Example**

**import** **pandas** **as** **pd**

d = pd.read\_csv("/content/employees.csv")

nd = d.dropna(axis=0, how='any')

print("Old data frame length:", len(d))

print("New data frame length:", len(nd))

print("Rows with at least one missing value:", (len(d) - len(nd)))

Handling missing data is a critical step in data cleaning using **Pandas**. Here’s a comprehensive list of **useful methods** to handle missing data, along with **code examples**.

**🔹 1. Detect Missing Data**

import pandas as pd

import numpy as np

df = pd.DataFrame({

'name': ['Alice', 'Bob', None],

'age': [25, None, 30],

'score': [85, np.nan, 90]

})

# Check for missing values (returns DataFrame of booleans)

print(df.isnull())

# Count total missing values per column

print(df.isnull().sum())

# Check for any missing data

print(df.isnull().any())

**🔹 2. Drop Missing Data**

# Drop rows with any NaN

df.dropna()

# Drop rows where all values are NaN

df.dropna(how='all')

# Drop columns with any NaN

df.dropna(axis=1)

# Drop rows where specific column is NaN

df.dropna(subset=['age'])

**🔹 3. Fill Missing Data**

python

CopyEdit

# Fill with a constant value

df.fillna(0)

# Fill with forward fill (previous value)

df.fillna(method='ffill')

# Fill with backward fill (next value)

df.fillna(method='bfill')

# Fill with column mean/median/mode

df['age'].fillna(df['age'].mean(), inplace=True)

df['score'].fillna(df['score'].median(), inplace=True)

# Fill with different values per column

df.fillna({'name': 'Unknown', 'age': df['age'].mean()})

**🔹 4. Replace Missing Values**

# Replace NaN with a specific value

df.replace(to\_replace=np.nan, value=0, inplace=True)

**🔹 5. Interpolate Missing Values**

# Linear interpolation (good for time series)

df['score'] = df['score'].interpolate()

**🔹 6. Check for rows/columns that are fully NA**

python

CopyEdit

# Return rows that are all NaN

df[df.isnull().all(axis=1)]

# Return columns that are all NaN

df.loc[:, df.isnull().all()]

**🔹 7. Drop duplicates before filling**

python

CopyEdit

df.drop\_duplicates(inplace=True)

**🔹 8. Custom function for complex logic**

python

CopyEdit

# Replace NaN in age if name is known

df['age'] = df.apply(lambda row: 25 if pd.isnull(row['age']) and row['name'] == 'Alice' else row['age'], axis=1)

**✅ Summary Table of Key Methods**

| **Method** | **Description** |
| --- | --- |
| isnull() / notnull() | Detect missing values |
| dropna() | Remove missing rows/columns |
| fillna() | Fill missing values with default |
| interpolate() | Fill gaps via interpolation |
| replace() | Replace missing with other values |
| apply() | Custom logic |
| any() / all() | Aggregate missing check |

Let me know if you want to handle missing data in a time series or large-scale dataset — there are optimized techniques for that too.

* [Removing Duplicates](https://www.geeksforgeeks.org/python-pandas-dataframe-drop_duplicates/)

**What Are Duplicates?**

In a DataFrame, a row is considered a **duplicate** if all or specified subset of values are the same as another row.

**🔹 1. Detect Duplicates**

python

CopyEdit

import pandas as pd

df = pd.DataFrame({

'name': ['Alice', 'Bob', 'Alice', 'Charlie', 'Bob'],

'age': [25, 30, 25, 35, 30]

})

# Check for duplicate rows (returns boolean Series)

print(df.duplicated())

# Check for duplicates in a specific column

print(df.duplicated(subset=['name']))

# Get only the duplicate rows

print(df[df.duplicated()])

**🔹 2. Remove Duplicates**

python

CopyEdit

# Remove all duplicate rows, keep the first occurrence

df.drop\_duplicates(inplace=True)

# Resetting the index after dropping

df = df.reset\_index(drop=True)

# Remove duplicates but keep the last occurrence

df.drop\_duplicates(keep='last')

# Remove duplicates based on specific columns

df.drop\_duplicates(subset=['name'], keep='first')

**🔹 3. Keep Only Duplicates (optional step for analysis)**

python

CopyEdit

# Get only the duplicated rows

duplicates = df[df.duplicated(keep=False)]

**🔹 4. Mark Duplicates for Review (instead of dropping)**

python

CopyEdit

# Add a column marking if a row is duplicated

df['is\_duplicate'] = df.duplicated()

**🔹 5. Drop Exact Duplicate Columns (Rare)**

python

CopyEdit

# Transpose, drop duplicate columns, then transpose back

df = df.T.drop\_duplicates().T

**🔹 6. Drop Duplicate Indexes**

python

CopyEdit

df = df[~df.index.duplicated(keep='first')]

**✅ Summary Table of Key Methods**

| **Method** | **Description** |
| --- | --- |
| duplicated() | Returns boolean Series marking duplicates |
| drop\_duplicates() | Drops duplicate rows |
| subset=[...] | Checks duplicates only on given columns |
| `keep='first' | 'last' |
| reset\_index(drop=True) | Resets index after removing duplicates |
| df[df.duplicated()] | Filters only duplicated rows |
| ~df.index.duplicated() | Removes duplicate indexes |
| T.drop\_duplicates().T | Removes duplicate columns |

**🔍 Example – Full Workflow**

python

CopyEdit

df = pd.DataFrame({

'name': ['Alice', 'Bob', 'Alice', 'Charlie', 'Bob'],

'age': [25, 30, 25, 35, 30]

})

# Remove duplicate rows, keeping first

df = df.drop\_duplicates()

# Reset index

df = df.reset\_index(drop=True)

print(df)

Let me know if your data has **nested structures**, **multi-index**, or you want to preserve only certain columns — I can show examples for those cases too.

* [Pandas Change Datatype](https://www.geeksforgeeks.org/pandas-change-datatype/)

**Why Change Data Types?**

* Improve performance & memory usage
* Fix wrong or ambiguous data types
* Enable calculations or date operations

**🔹 1. Check Current Data Types**

python

CopyEdit

import pandas as pd

df = pd.DataFrame({

'age': ['25', '30', '35'],

'price': ['100.5', '200.1', '300.75'],

'joined': ['2022-01-01', '2022-02-01', '2022-03-01']

})

print(df.dtypes)

**🔹 2. Change Column Type Using astype()**

python

CopyEdit

# Convert 'age' from string to integer

df['age'] = df['age'].astype(int)

# Convert 'price' to float

df['price'] = df['price'].astype(float)

# Convert multiple columns at once

df = df.astype({'age': 'int', 'price': 'float'})

**🔹 3. Convert to Datetime Using pd.to\_datetime()**

python

CopyEdit

df['joined'] = pd.to\_datetime(df['joined'])

# Check again

print(df.dtypes)

**🔹 4. Convert to Numeric Using pd.to\_numeric()**

python

CopyEdit

# Useful when data has invalid entries (like 'N/A', '$')

df['price'] = pd.to\_numeric(df['price'], errors='coerce') # invalid values become NaN

**🔹 5. Convert to Category**

python

CopyEdit

# Great for string columns with few unique values (saves memory)

df['gender'] = ['M', 'F', 'M']

df['gender'] = df['gender'].astype('category')

**🔹 6. Convert Boolean or Object Columns**

python

CopyEdit

df['is\_active'] = ['True', 'False', 'True']

df['is\_active'] = df['is\_active'].astype(bool)

**🔹 7. Change Data Type In-place with Error Handling**

python

CopyEdit

# Convert and handle errors gracefully

df['age'] = pd.to\_numeric(df['age'], errors='coerce') # Invalids become NaN

df['joined'] = pd.to\_datetime(df['joined'], errors='coerce') # Invalids become NaT

**✅ Summary Table of Key Methods**

| **Method** | **Description** |
| --- | --- |
| df.dtypes | Show current column types |
| df.astype() | Convert to int, float, str, bool, category |
| pd.to\_numeric() | Convert to number (int/float), with error control |
| pd.to\_datetime() | Convert to datetime format |
| astype('category') | Efficient storage for string labels |
| errors='coerce' | Handle invalid values gracefully |

**✅ Full Example Recap**

python

CopyEdit

df = pd.DataFrame({

'age': ['25', '30', '35'],

'price': ['100.5', '200.1', '300.75'],

'joined': ['2022-01-01', '2022-02-01', '2022-03-01'],

'gender': ['M', 'F', 'M']

})

df['age'] = df['age'].astype(int)

df['price'] = df['price'].astype(float)

df['joined'] = pd.to\_datetime(df['joined'])

df['gender'] = df['gender'].astype('category')

print(df.dtypes)

Let me know if your data comes from CSVs, JSONs, or APIs — and I’ll help you auto-convert types while loading the data.

* [Drop Empty Columns in Pandas](https://www.geeksforgeeks.org/drop-empty-columns-in-pandas/)

Cleaning data is an essential step in data analysis. In this guide we will explore different ways to drop empty, null and zero-value columns in a [Pandas DataFrame](https://www.geeksforgeeks.org/python-pandas-dataframe/) using Python. By the end you'll know how to efficiently clean your dataset using the dropna() and replace() methods.

**Understanding dropna()**

The [dropna() function](https://www.geeksforgeeks.org/python-pandas-dataframe-dropna/" \t "_blank) is a powerful method in Pandas that allows us to remove rows or columns containing missing values (NaN). Depending on the parameters used it can remove rows or columns where at least one value is missing or only those where all values are missing.

***Syntax:*** *DataFrameName.dropna(axis=0, how='any', inplace=False)*

**Parameters:**

* **axis:** axis takes int or string value for rows/columns. Input can be 0 or 1 for Integer and ‘index’ or ‘columns’ for String.
* **how:**how takes string value of two kinds only (‘any’ or ‘all’). ‘any’ drops the row/column if ANY value is Null and ‘all’ drops only if ALL values are null.
* **inplace:** It is a boolean which makes the changes in the data frame itself if True.

**Create a Sample DataFrame:**

This is the sample data frame on which we will use to perform different operations.

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

df = pd.DataFrame({'FirstName': ['Vipul', 'Ashish', 'Milan'],

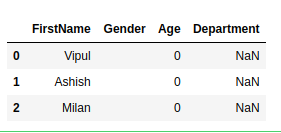
"Gender": ["", "", ""],

"Age": [0, 0, 0]})

df['Department'] = np.nan

print(df)

**Output:**



**Example 1: Remove All Null Value Columns**

This method removes columns where **all** values are NaN. If a column is completely empty (contains only NaN values) it is unnecessary for analysis and can be removed using dropna(how='all', axis=1).

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

df = pd.DataFrame({'FirstName': ['Vipul', 'Ashish', 'Milan'],

"Gender": ["", "", ""],

"Age": [0, 0, 0]})

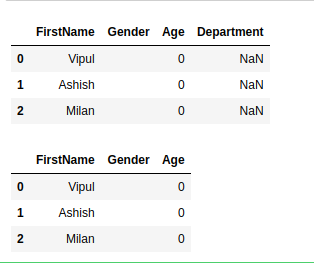
df['Department'] = np.nan

display(df)

df.dropna(how='all', axis=1, inplace=**True**)

display(df)

**Output:**



**Example 2: Replace Empty Strings with Null and Drop Null Columns**

If a column contains empty strings we need to replace them with NaN before dropping the column. Empty strings are not automatically recognized as missing values in Pandas so converting them to NaN ensures they can be handled correctly. After conversion we use dropna(how='all', axis=1) to remove columns that are entirely empty.

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

df = pd.DataFrame({'FirstName': ['Vipul', 'Ashish', 'Milan'],

"Gender": ["", "", ""],

"Age": [0, 0, 0]})

df['Department'] = np.nan

display(df)

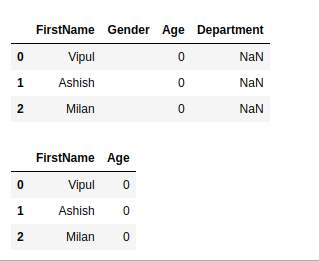
nan\_value = float("NaN")

df.replace("", nan\_value, inplace=**True**)

df.dropna(how='all', axis=1, inplace=**True**)

display(df)

**Output:**



**Example 3: Replace Zeros with Null and Drop Null Columns**

If columns contain only zero values, we convert them to NaN before dropping them.

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

df = pd.DataFrame({'FirstName': ['Vipul', 'Ashish', 'Milan'],

"Gender": ["", "", ""],

"Age": [0, 0, 0]})

df['Department'] = np.nan

display(df)

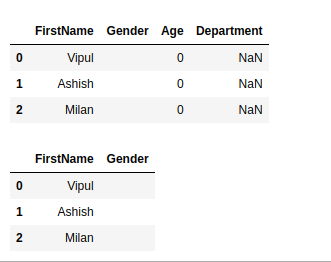
nan\_value = float("NaN")

df.replace(0, nan\_value, inplace=**True**)

df.dropna(how='all', axis=1, inplace=**True**)

display(df)

**Output:**



**Example 4: Replace Both Zeros and Empty Strings with Null and Drop Null Columns**

To clean a dataset fully we may need to replace both zeros and empty strings.

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

df = pd.DataFrame({'FirstName': ['Vipul', 'Ashish', 'Milan'],

"Gender": ["", "", ""],

"Age": [0, 0, 0]})

df['Department'] = np.nan

display(df)

nan\_value = float("NaN")

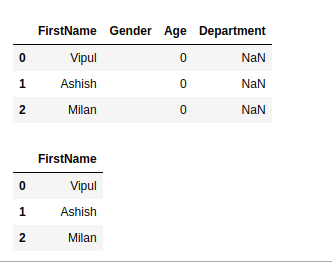
df.replace(0, nan\_value, inplace=**True**)

df.replace("", nan\_value, inplace=**True**)

df.dropna(how='all', axis=1, inplace=**True**)

display(df)

**Output:**



* [String manipulations in Pandas](https://www.geeksforgeeks.org/string-manipulations-in-pandas-dataframe/)

**Setup: Sample DataFrame**

python

CopyEdit

import pandas as pd

df = pd.DataFrame({

'name': [' Alice ', 'BOB', 'ChArLie ', None],

'email': ['alice@example.com', 'BOB@example.COM', 'charlie@EXAMPLE.com', '']

})

**🔹 1. String Case Operations**

python

CopyEdit

df['name\_lower'] = df['name'].str.lower()

df['name\_upper'] = df['name'].str.upper()

df['name\_title'] = df['name'].str.title()

**🔹 2. Trim Whitespace**

python

CopyEdit

df['name\_trimmed'] = df['name'].str.strip() # both sides

df['name\_lstrip'] = df['name'].str.lstrip() # left only

df['name\_rstrip'] = df['name'].str.rstrip() # right only

**🔹 3. Substring Detection**

python

CopyEdit

df['has\_alice'] = df['email'].str.contains('alice', case=False, na=False)

df['starts\_with\_char'] = df['name'].str.startswith('Char', na=False)

df['ends\_with\_com'] = df['email'].str.endswith('.com', na=False)

**🔹 4. Replace & Remove Strings**

python

CopyEdit

df['email\_cleaned'] = df['email'].str.replace('EXAMPLE', 'demo', case=False, regex=True)

df['name\_no\_space'] = df['name'].str.replace(' ', '', regex=False)

**🔹 5. Extract Substrings (Regex)**

python

CopyEdit

# Extract domain from email

df['domain'] = df['email'].str.extract(r'@(\w+\.\w+)', expand=False)

# Extract first 3 letters of name

df['first\_3\_letters'] = df['name'].str[:3]

**🔹 6. Split & Join**

python

CopyEdit

# Split email into user and domain

df[['user', 'domain']] = df['email'].str.split('@', expand=True)

# Join strings

df['joined'] = df['user'].str.cat(df['domain'], sep='@')

**🔹 7. Length & Count**

python

CopyEdit

df['name\_len'] = df['name'].str.len()

df['a\_count'] = df['name'].str.count('a', flags=re.IGNORECASE)

**🔹 8. Check for Empty or Missing**

python

CopyEdit

df['is\_empty'] = df['email'].str.len() == 0

df['is\_null'] = df['email'].isnull()

**🔹 9. String Formatting**

python

CopyEdit

df['formatted'] = df['name'].str.strip().str.title() + ' <' + df['email'] + '>'

**✅ Summary Table of Key Methods**

| **Category** | **Method Examples** |
| --- | --- |
| Case | .str.lower(), .str.upper(), .str.title() |
| Whitespace | .str.strip(), .str.lstrip(), .str.rstrip() |
| Search/Match | .str.contains(), .str.startswith(), .str.endswith() |
| Replace/Remove | .str.replace(), .str.replace('', '') |
| Extract | .str.extract(r'pattern'), .str[:n] |
| Split/Join | .str.split(), .str.cat() |
| Count/Length | .str.len(), .str.count('a') |
| Null/Empty Check | .isnull(), .str.len() == 0 |

**🔍 Full Example Output**

python

CopyEdit

print(df[['name', 'name\_trimmed', 'name\_upper', 'email', 'domain']])

Let me know if you need string manipulations while **reading CSVs**, or before **writing cleaned output** — I can show that too.

4o

**You said:**

can you take some large data set of dataframe and do all needfull operations code example

**ChatGPT said:**

Absolutely! Let's walk through a **realistic example** using a **larger mock dataset** and perform **all major Pandas operations** including:

* Handling missing data
* Removing duplicates
* Changing data types
* Dropping empty columns
* String manipulations

**🔰 Step 1: Create a Sample Large DataFrame**

python

CopyEdit

import pandas as pd

import numpy as np

data = {

'id': range(1, 11),

'name': ['Alice ', 'Bob', 'Charlie', 'Alice ', None, 'bob', 'Dave', 'Eve', 'Frank', 'Charlie'],

'age': [25, 30, None, 25, 40, 30, 35, None, 50, 35],

'email': [

'alice@example.com', 'bob@example.com', None, 'alice@example.com',

'', 'bob@example.com', 'dave@example.com', 'eve@example.com', None, 'charlie@example.com'

],

'join\_date': [

'2021-01-01', '2021-06-15', 'not a date', None,

'2022-03-12', '2021-06-15', '2020-11-05', '2023-08-19', '2023-09-30', 'not a date'

],

'empty\_col': [None] \* 10 # column with all nulls

}

df = pd.DataFrame(data)

**🔹 Step 2: Handle Missing Data**

python

CopyEdit

# Fill missing names with 'Unknown'

df['name'] = df['name'].fillna('Unknown')

# Fill missing ages with the median

df['age'] = df['age'].fillna(df['age'].median())

# Replace empty string in email with NaN, then fill

df['email'].replace('', np.nan, inplace=True)

df['email'].fillna('unknown@example.com', inplace=True)

**🔹 Step 3: Remove Duplicates**

python

CopyEdit

# Drop duplicate rows based on name and email

df = df.drop\_duplicates(subset=['name', 'email'])

**🔹 Step 4: Change Data Types**

python

CopyEdit

# Convert join\_date to datetime, handle errors

df['join\_date'] = pd.to\_datetime(df['join\_date'], errors='coerce')

# Convert age to integer (optional)

df['age'] = df['age'].astype(int)

**🔹 Step 5: Drop Empty Columns**

python

CopyEdit

# Drop columns where all values are NaN

df = df.dropna(axis=1, how='all')

**🔹 Step 6: String Manipulations**

python

CopyEdit

# Clean and normalize names

df['name'] = df['name'].str.strip().str.title()

# Extract domain from email

df['email\_domain'] = df['email'].str.extract(r'@(\w+\.\w+)', expand=False)

# Create a username column

df['username'] = df['email'].str.extract(r'^([^@]+)', expand=False)

# Length of email

df['email\_len'] = df['email'].str.len()

* [String methods in Pandas](https://www.geeksforgeeks.org/top-10-string-methods-in-pandas/)
* [Detect Mixed Data Types and Fix it](https://www.geeksforgeeks.org/pandas-detect-mixed-data-types-and-fix-it/)

Detecting and fixing **mixed data types** in a Pandas DataFrame is crucial for avoiding bugs, improving performance, and ensuring data consistency. Here’s a complete guide with **useful methods** and **code examples**.

**✅ What Are Mixed Data Types?**

A column has **mixed types** when it contains values of different Python types (e.g., strings and integers in the same column). This often results in the column being typed as object or string.

**🔍 Step-by-Step: Detect & Fix Mixed Data Types**

**🧭 Step 1: Create Example DataFrame with Mixed Types**

python

CopyEdit

import pandas as pd

df = pd.DataFrame({

'id': [1, 2, 3, 4],

'age': [25, '30', 35, 'forty'],

'joined': ['2021-01-01', '2022-01-15', pd.NaT, 'not a date'],

'income': [50000.0, '60000', '70000.0', 'eighty thousand']

})

**🔎 Step 2: Detect Mixed Data Types**

**➤ 1. Use df.dtypes**

python

CopyEdit

print(df.dtypes)

**➤ 2. Use applymap with type**

python

CopyEdit

# Show actual Python types in each cell

print(df.applymap(type))

**➤ 3. Detect inconsistent types in a column**

python

CopyEdit

# Show all types present in each column

for col in df.columns:

print(f"{col} types:", df[col].map(type).value\_counts().to\_dict())

**🛠️ Step 3: Fix Mixed Data Types**

**➤ Convert to numeric**

python

CopyEdit

# Convert 'age' and 'income' to numeric; coerce errors to NaN

df['age'] = pd.to\_numeric(df['age'], errors='coerce')

df['income'] = pd.to\_numeric(df['income'], errors='coerce')

**➤ Convert to datetime**

python

CopyEdit

df['joined'] = pd.to\_datetime(df['joined'], errors='coerce')

**➤ Fill or drop problematic rows**

python

CopyEdit

# Fill NaNs with a default

df['age'].fillna(df['age'].median(), inplace=True)

# Or drop rows where conversion failed

df.dropna(subset=['income', 'joined'], inplace=True)

**✅ Step 4: Final Check**

python

CopyEdit

print(df.dtypes)

print(df)

**🧰 Useful Methods Summary**

| **Task** | **Method** |
| --- | --- |
| Check data types | df.dtypes |
| See types of individual cells | df.applymap(type) |
| Count Python types in a col | df[col].map(type).value\_counts() |
| Convert to numeric | pd.to\_numeric(col, errors='coerce') |
| Convert to datetime | pd.to\_datetime(col, errors='coerce') |
| Drop rows with invalid data | df.dropna(subset=[...]) |
| Fill default on error | fillna() |

**🧪 Example Output**

python

CopyEdit

print(df)

# Output after fixing:

# id age joined income

# 0 1 25.0 2021-01-01 50000.0

# 1 2 30.0 2022-01-15 60000.0

Let me know if you’d like to auto-detect **object columns with mixed types** or want this as a utility function to apply to any dataset.

**Replace Values Between 0 and 0.4 with NaN**

**🔸 (Exclusive) 0 < x < 0.4:**

python

CopyEdit

df = df.mask((df > 0) & (df < 0.4), np.nan)

**🔸 (Inclusive) 0 ≤ x ≤ 0.4:**

python

CopyEdit

df = df.mask((df >= 0) & (df <= 0.4), np.nan)

**✅ Final DataFrame**

python

CopyEdit

print(df)