

ISA 419: Data-Driven Security

05: Cleaning and Combining Data in Pandas

Fadel M. Megahed, PhD

Professor
Farmer School of Business
Miami University

 @FadelMegahed

 fmegahed

 fmegahed@miamioh.edu

 Automated Scheduler for Office Hours

Spring 2025

Quick Refresher of Last Class

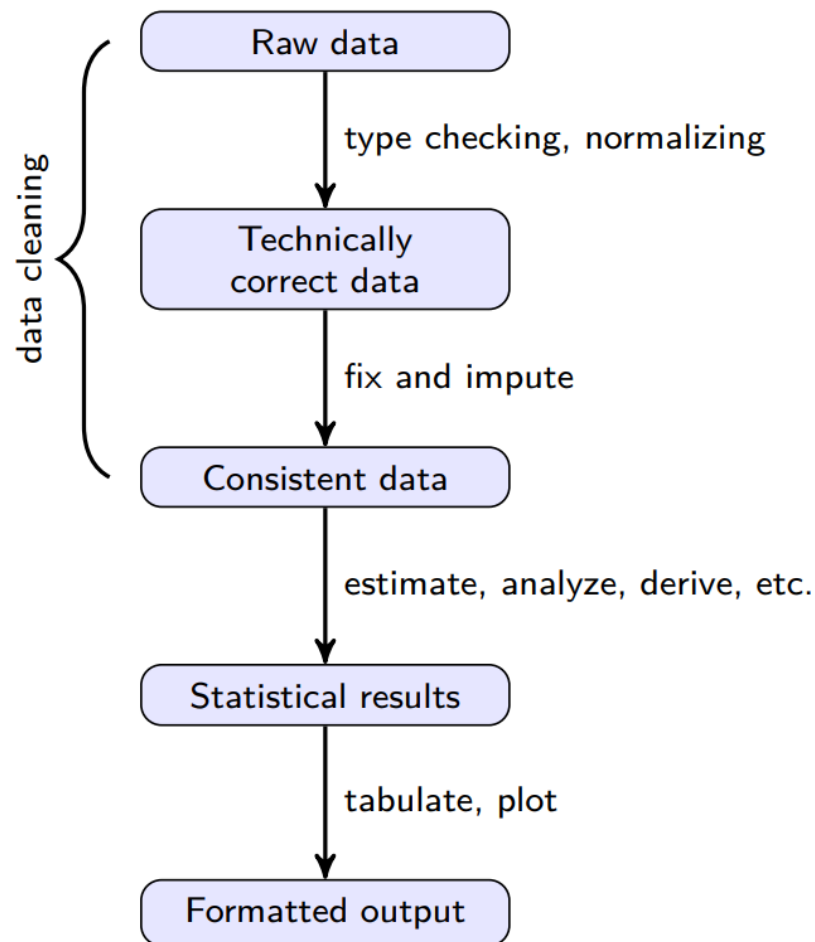
- ✓ Describe and create data frames
- ✓ Import data using pandas
- ✓ Utilize the `head()`, `tail()`, `shape()`, `info()`, `dtypes()`, `count()`, `value_count()`, `describe()`, `isnull().sum()` functions to quickly explore and summarize your dataset
- ✓ Use the `loc[]` and `iloc[]` to select rows and columns
- ✓ Use `assign()` to create new columns on your existing data frame.

Learning Objectives for Today's Class

- Ensure that your imported data is **technically correct** (rename columns and fix `dtypes`)
- Clean data to ensure that your data is consistent
- Understand the difference between `concatenate`, `merge`, and `join`.

Technically Correct Data

The Data Analysis Value Chain



Technically Correct Data

Raw data files may lack headers, contain wrong data types (e.g. numbers stored as strings), wrong category labels, unknown or unexpected character encoding and so on.

Technically correct data is the state in which data can be read into an R data.frame, with correct **name**, **types** and **labels**, without further trouble. However, that **does not mean that the values are error-free or complete**. --- De Jonge and Van Der Loo (2013)

Functions for Cleaning/Renaming Variables

type	package	function()	description
cleaning names	janitor	clean_names()	Resulting names are unique, consisting only of the _ character, numbers, and letters
renaming	pandas	df.rename(columns={'dept2': 'econ', 'dept3': 'isa'})	Will rename the columns dept2 and dept3 to econ and isa

An Example of Cleaning and Renaming Names

```
import pandas as pd
import janitor

# Create a dataframe
df = pd.DataFrame(
    {'dept 1': [1, 2, 3], 'dept 2': [4, 5, 6], 'dept 3': [7, 8, 9]}
)

df = (
    df
    # clean the column names
    .clean_names()
    # note that the column names are now separated with an underscore
    .rename(columns={'dept_2': 'econ', 'dept_3': 'isa'})
)

df.head(n=2)
```

```
##      dept_1  econ  isa
## 0         1     4    7
## 1         2     5    8
```

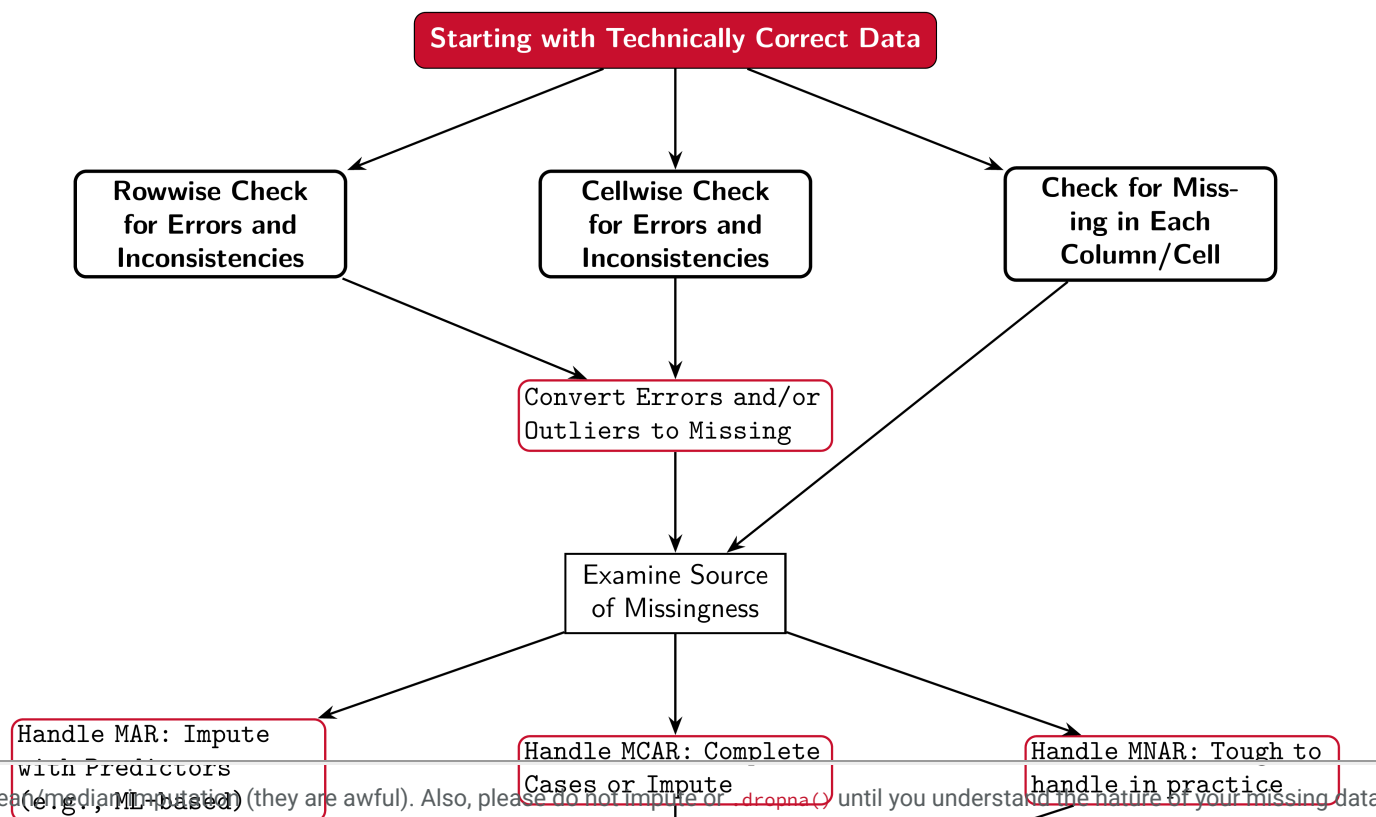

Changing Column/Series Data Types

- **Pandas** has a function called `astype()` that can be used to change the data type of a column. To convert into a different data type, you can use the following syntax:
 - **Int:** `df['column_name'] = df['column_name'].astype('int')`
 - **Categorical:** `df['column_name'] = df['column_name'].astype('category')`
 - **Datetime:** `df['column_name'] = df['column_name'].astype('datetime64')`
 - **Numeric:** `df['column_name'] = df['column_name'].astype('float')`
 - **String:** `df['column_name'] = df['column_name'].astype('str')`
 - **Boolean:** `df['column_name'] = df['column_name'].astype('bool')`
- Obviously, these can also be passed via the `assign()` function in **Pandas**. For example:
`df = df.assign(column_name = df['column_name'].astype('int'))`.

Data Cleaning

My Opinionated Perspective on Data Cleaning

```
tinytex::xelatex('../..//figures/data_cleaning_process.tex')
pdftools::pdf_convert('../..//figures/data_cleaning_process.pdf', dpi = 600,
                      filenames = '../..//figures/data_cleaning_process.png')
```



Reminders: Please do not use mean/median imputation (they are awful). Also, please do not impute or `dropna()` until you understand the nature of your missing data.

Additional Thoughts on Data Cleaning

- **Data cleaning** is the process of ensuring that your data is consistent and correct. This involves checking for errors and inconsistencies, handling missing data, and ensuring the final dataset is consistent and ready for analysis.
- **Data cleaning** is a critical step in the data analysis process. If you don't clean your data, you can't trust your results.
- **Data cleaning** is a time-consuming process. It's not uncommon for data scientists to spend 80% of their time cleaning data and only 20% of their time analyzing it.
- **Data cleaning** is an iterative process. You'll often find that you need to go back and clean your data again after you've started analyzing it.

Data Cleaning in Python

- **Pandas** has a number of functions that can be used to clean data. Some of the most common ones include:
 - `drop_duplicates()`: Remove duplicate rows from a DataFrame.
 - `dropna()`: Remove missing values from a DataFrame.
 - `fillna()`: Fill missing values in a DataFrame (use with caution).
 - `str.replace()`: Replace values in a Series.
 - `str.strip()`: Remove leading and trailing whitespace from a Series.
 - `str.extract()`: Extract values from a Series using a regular expression.
 - `str.get_dummies()`: Convert a Series to dummy variables.
 - `str.pad()`: Pad strings in a Series.
 - `str.zfill()`: Pad strings in a Series with zeros.

Combining Multiple DataFrames

A Motivating Example for merge()

A df of IP addresses and counts:

```
df1 = pd.DataFrame(  
    {'ip': ['134.53.10.20', '134.53.355.200', '129.171.123.45'],  
    'count': [10, 20, 30]}  
)  
  
print(df1)
```

##		ip	count
## 0		134.53.10.20	10
## 1		134.53.355.200	20
## 2		129.171.123.45	30

A df of IP addresses and cities:


```
df2 = pd.DataFrame(  
    {'ip': ['134.53.10.20', '69.12.18.100', '129.171.123.45'],  
    'city': ['Oxford', 'Buffalo', 'Coral Gables']}  
)  
  
print(df2)
```

##		ip	city
## 0		134.53.10.20	Oxford
## 1		69.12.18.100	Buffalo
## 2		129.171.123.45	Coral Gables





What are the differences between the two data frames


Edit me to answer the question.


merge(): API Reference

Getting startedUser Guide**API reference**DevelopmentRelease notes

Search Choose version



Input/output
General functions
Series
DataFrame 
pandas.DataFrame
pandas.DataFrame.index
pandas.DataFrame.columns
pandas.DataFrame.dtypes
pandas.DataFrame.info
pandas.DataFrame.select_dtypes

 > API reference > DataFrame > pandas.DataF...

pandas.DataFrame.merge

DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=None, indicator=False, validate=None) [\[source\]](#)

Merge DataFrame or named Series objects with a database-style join.

A named Series object is treated as a DataFrame with a single named column.

The join is done on columns or indexes. If joining columns on columns, the DataFrame indexes *will be ignored*. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on. When performing a cross merge, no column specifications to merge on

merge(): Syntax

- The `merge()` function in **Pandas** is used to merge two DataFrames. It is similar to the `join()` function in SQL.
- The `merge()` function takes the following arguments:
 - `left`: The first DataFrame to merge.
 - `right`: The second DataFrame to merge.
 - `on`: The column to merge on.
 - `left_on` and `right_on` can be used if the column names are different.
 - `how`: The type of join to perform (e.g., 'inner', 'outer', 'left', 'right').
 - `suffixes`: A tuple of strings to append to overlapping column names.
 - `indicator`: If `True`, adds a column to the merged DataFrame indicating the source of each row.
 - `validate`: If `True`, checks that the merge is unique.

merge()

Default merge():

```
df3 = pd.merge(df1, df2, on='ip')  
print(df3)
```

##		ip	count	city
## 0	134.53.10.20	10	Oxford	
## 1	129.171.123.45	30	Coral Gables	

Changing the how to left:


```
df4 = pd.merge(df1, df2, on='ip', how='left')  
print(df4)
```

##		ip	count	city
## 0	134.53.10.20	10	Oxford	
## 1	134.53.355.200	20	NaN	
## 2	129.171.123.45	30	Coral Gables	


What are the differences between the two data frames?


Edit me to answer the question.


join(): API Reference

Getting startedUser GuideAPI referenceDevelopmentRelease notes

Search Choose version



Input/output
General functions
Series
DataFrame 
pandas.DataFrame
pandas.DataFrame.index
pandas.DataFrame.columns
pandas.DataFrame.dtypes
pandas.DataFrame.info
pandas.DataFrame.select_dtypes

 > API reference > DataFrame > pandas.DataF...

pandas.DataFrame.join

`DataFrame.join(other, on=None, how='left', lsuffix='', rsuffix='', sort=False, validate=None)` [\[source\]](#)

Join columns of another DataFrame.

Join columns with *other* DataFrame either on index or on a key column. Efficiently join multiple DataFrame objects by index at once by passing a list.


Parameters:

other : *DataFrame, Series, or a list containing any combination of them*





join(): Syntax

- The `join()` function is similar to the `merge()` function, but it is used to join DataFrames on their **indices**. The syntax for the `join()` function is as follows:
 - `df1.join(df2)`: Join `df1` and `df2` on their indices.
 - `df1.join(df2, how='left')`: Join `df1` and `df2` on their indices using a left join.
 - `df1.join(df2, how='right')`: Join `df1` and `df2` on their indices using a right join.
 - `df1.join(df2, how='inner')`: Join `df1` and `df2` on their indices using an inner join.
 - `df1.join(df2, how='outer')`: Join `df1` and `df2` on their indices using an outer join.

concat(): API Reference

Getting startedUser GuideAPI referenceDevelopmentRelease notes

Search Choose version



Input/output

General functions ☐

- pandas.melt
- pandas.pivot
- pandas.pivot_table
- pandas.crosstab
- pandas.cut
- pandas.qcut
- pandas.merge
- pandas.merge_ordered
- pandas.merge_asof

[Home](#) > [API reference](#) > [General functions](#) > **pandas.concat**

pandas.concat

pandas.concat(*objs*, *, *axis=0*, *join='outer'*, *ignore_index=False*, *keys=None*, *levels=None*, *names=None*, *verify_integrity=False*, *sort=False*, *copy=None*) [\[source\]](#)

Concatenate pandas objects along a particular axis.

Allows optional set logic along the other axes.

Can also add a layer of hierarchical indexing on the concatenation axis, which may be useful if the labels are the same (or overlapping) on the passed axis number.

Parameters:

concat(): Syntax

- The `concat()` function can be used to concatenate DataFrames along rows or columns. The syntax for the `concat()` function is as follows:
 - `pd.concat([df1, df2], axis=0)`: Concatenate DataFrames along rows, i.e., stack them on top of each other.
 - `pd.concat([df1, df2], axis=1)`: Concatenate DataFrames along columns, i.e., stack them side by side.
 - `pd.concat([df1, df2], axis=0, ignore_index=True)`: Concatenate DataFrames along rows, ignoring the index.
 - `pd.concat([df1, df2], axis=1, join='inner')`: Concatenate DataFrames along columns, using the intersection of the columns.
 - `pd.concat([df1, df2], axis=1, join='outer')`: Concatenate DataFrames along columns, using the union of the columns.

Summary: Merge, Concat, and Join

- **Pandas** has a number of functions that can be used to combine DataFrames. Some of the most common ones include:
 - `merge()`: Merge DataFrames using a database-style join. `merge()` can be used to perform inner, outer, left, and right joins on one or more common columns.
 - `concat()`: Concatenate (stack) DataFrames along rows or columns.
 - `join()`: Join DataFrames using their indexes.

A Demonstration of Combining Data Frames in a Cyber Security Context

An Overview of our Two Datasets

Our [Toxic IP Addresses dataset](#) comes from [Stop Forum Spam](#) and contains a list of IP addresses that have been reported as toxic. The dataset contains the following columns:

- **ip**: The IP address.
- **frequency**: The number of times that the IP address number has been reported during the 90 day period.
- **lastseen**: The date/time (UTC) when the IP address was last seen.

I used the [geoip2](#) library to geolocate the IP addresses in the dataset based on the [Maxmind GeoLite2 City Database](#). I saved the results in a CSV file called [ip_geolocation.csv](#), containing five columns:

- **IP**: The IP address.
- **Country**: The country where the IP address is located.
- **City**: The city where the IP address is located.
- **Latitude**: The latitude of the IP address.
- **Longitude**: The longitude of the IP address.

Analyzing Toxic IPs and their Geolocations

Task	Task 1	Task 2	Task 3	Task 4	Task 5
------	--------	--------	--------	--------	--------

- Load the **Toxic IP Addresses dataset** into a DataFrame called `toxic_ips`.
- Load the **IP Geolocation dataset** into a DataFrame called `geolocation`.
- Answer the questions in the next tabs.
- You can work in groups of 2-3 students.

Recap

Summary of Main Points

By now, you should be able to do the following:

- Ensure that your imported data is **technically correct** (rename columns and fix `dtypes`)
- Clean data to ensure that your data is consistent
- Understand the difference between `concatenate`, `merge`, and `join`.



Review and Clarification



- **Class Notes:** Take some time to revisit your class notes for key insights and concepts.
- **Zoom Recording:** The recording of today's class will be made available on Canvas approximately 3-4 hours after the end of class.
- **Questions:** Please don't hesitate to ask for clarification on any topics discussed in class. It's crucial not to let questions accumulate.