Data Manipulation with Pandas

Hrachya Asatryan

What This Lecture Is (and Isn't)

This lecture is about

- Understanding data as an object
- How pandas represents and manipulates data
- How to *reason* about datasets

This lecture is NOT

- A pandas API reference
- A tutorial on every method
- Machine learning

Why Pandas Exists

Before pandas:

- NumPy → fast but structureless
- SQL → structured but rigid
- Excel → flexible but dangerous

Pandas exists to:

- Work with **structured, labeled data**
- Iterate quickly
- Preserve meaning

Pandas is a library for labeled, tabular data and the operations that make sense on it.

What “Labeled” Means

- Rows and columns have **names**
- Operations align by **labels**, not positions
- Labels are part of the data

This changes how computations behave.

Pandas Core Abstractions

Two Fundamental Objects

Pandas has two core data structures:

- **Series** (1D, labeled)

- **DataFrame** (2D, labeled)

Everything else builds on these.

pd.Series: Labeled 1D Data

A Series is:

- A 1D array
- With an **index**

```
s = pd.Series([72, 85, 90], index=["math", "physics", "cs"])
```

pd.Series is both an array and a mapping

- Array-like: supports vectorized operations
- Dictionary-like: keys → values

```
s["physics"]
  s.iloc[1]
```

Index ≠ position.

DataFrame: Aligned Series

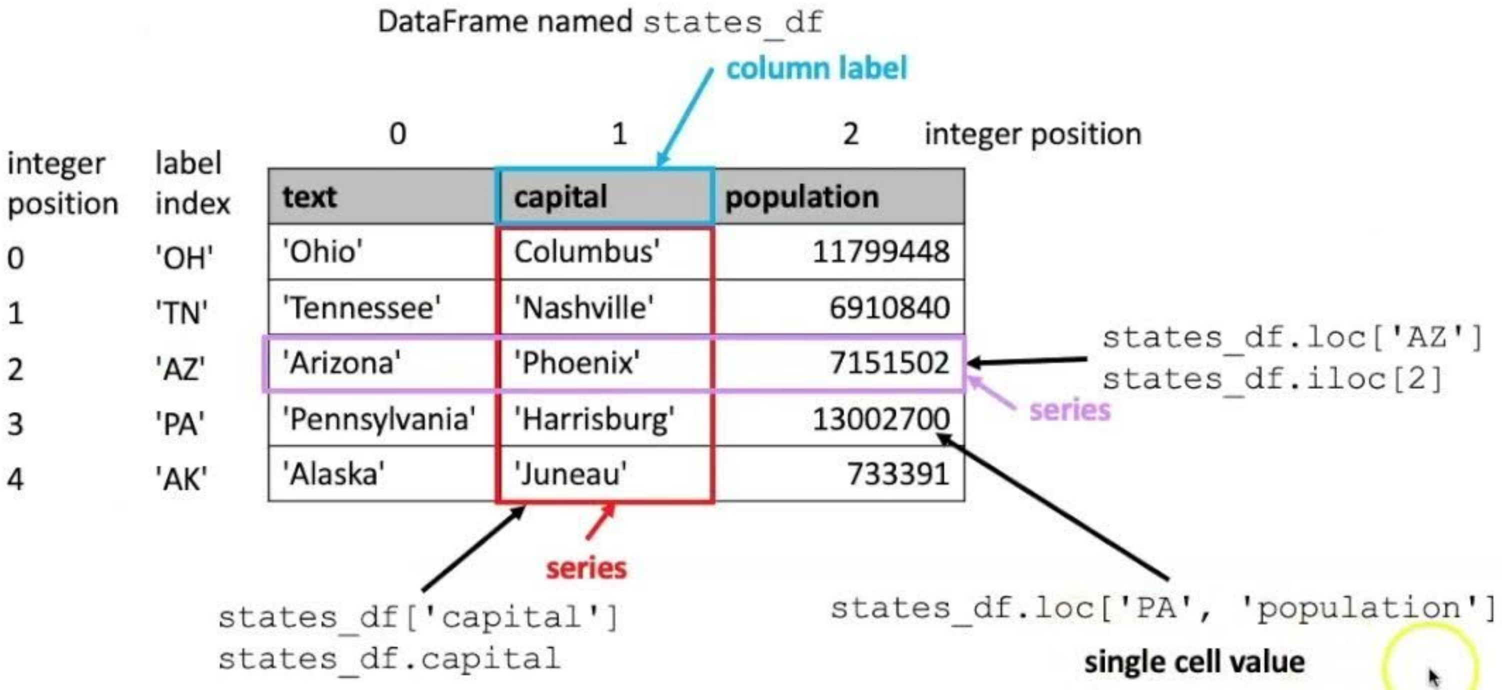
A DataFrame is:

- A collection of Series
- Sharing the same index

```
df["math"]  # Series
```

Each column is its own labeled object.

DataFrame



DataFrame named `states_df`

column label

		0	1	2 integer position
integer position	label index	text	capital	population
0	'OH'	'Ohio'	Columbus'	11799448
1	'TN'	'Tennessee'	'Nashville'	6910840
2	'AZ'	'Arizona'	'Phoenix'	7151502
3	'PA'	'Pennsylvania'	'Harrisburg'	13002700
4	'AK'	'Alaska'	'Juneau'	733391

`states_df.loc['AZ']`
`states_df.iloc[2]`

series

series

`states_df['capital']`
`states_df.capital`

`states_df.loc['PA', 'population']`

single cell value

- **Data frames** are essentially tables.
- The **values** of columns or rows are **series.**

Columns vs Rows

Which axis do you think it is natural to do operations on?

Columns vs Rows

Which axis do you think it is natural to do operations on?

- If I want to calculate statistics for the data, it makes sense to do it per-column
- Column-wise operations are natural
- Row-wise operations require intent (`axis`)

```
df.mean()
df.mean(axis=1)
```

This is not accidental.

Indexing and Alignment

The Index Is Not Optional

The index:

- Identifies data
- Controls alignment
- Can be anything (strings, dates, categories)

Two Kinds of Indexing

- `.loc` → label-based
- `.iloc` → position-based

```
df.loc["Alice"]
df.iloc[0]
```

Never confuse them.

Why Pandas Forces This Separation

- Labels and positions are not the same thing
- **Silent mistakes are worse than errors**
- Alignment is the core idea
  - Pandas aligns data by labels
- `s1 + s2`
  - Matching labels combine
  - Non-matching labels -> `NaN`

Alignment Example

```
s1 = pd.Series([1,2,3], index=["a","b","c"])
s2 = pd.Series([10,20,30], index=["b","c","d"])
```

Result:

- Correct
- Dangerous
- Silent

Alignment in DataFrames

```
df - df.mean()
```

- Mean is computed per column

- Subtraction aligns on column names

This is **intentional broadcasting**.

Pandas Will Not Save You

Pandas assumes you know what you’re doing.

- No warnings
- No complaints
- Just NaNs

Data Types

Common pandas dtypes

- int64, float64
- object
- category
- datetime64

If you see object, be suspicious. It usually means

“ pandas gave up on understanding your data “

Missing Data Is Information

Missing data:

- Is not zero
- Is not false
- Has meaning

Represented as NaN.

Missing Data Operations

```
df.isna()
df.dropna()
df.fillna(0)
```

Handling missing data is a decision, not a fix.

Working with Data

What is data?

Data Is Not Reality

Data is:

- A measurement
- A proxy
- Often incomplete or biased

Reality → measurement → data

Data Is a Table of Claims

Every row is a claim about the world.
Every column is a type of claim.

Example:

- Row: “This person exists”

- Column: “This person’s age is 23”

If a value is missing or wrong → the claim is weak or false.

Data Isn’t Just Numbers

Data includes:

- Measurements
- Categories
- Identifiers
- Timestamps
- Flags

Pandas treats these differently **on purpose**.

This is why dtype matters.

Columns Are Not Equal

Columns play different roles:

- Identifiers (IDs)
- Features (measurements)
- Targets (labels)
- Metadata (timestamps, source)

Pandas doesn’t know this — **you must**.

Pandas Assumes Tabular Semantics

Pandas assumes:

- Rows are observations
- Columns are variables
- Columns have meaning across rows

This is why:

- vectorization works
- groupby makes sense
- alignment exists

Pandas Assumes Comparability

If values are in the same column:

- They are comparable
- They share units
- They can be summarized

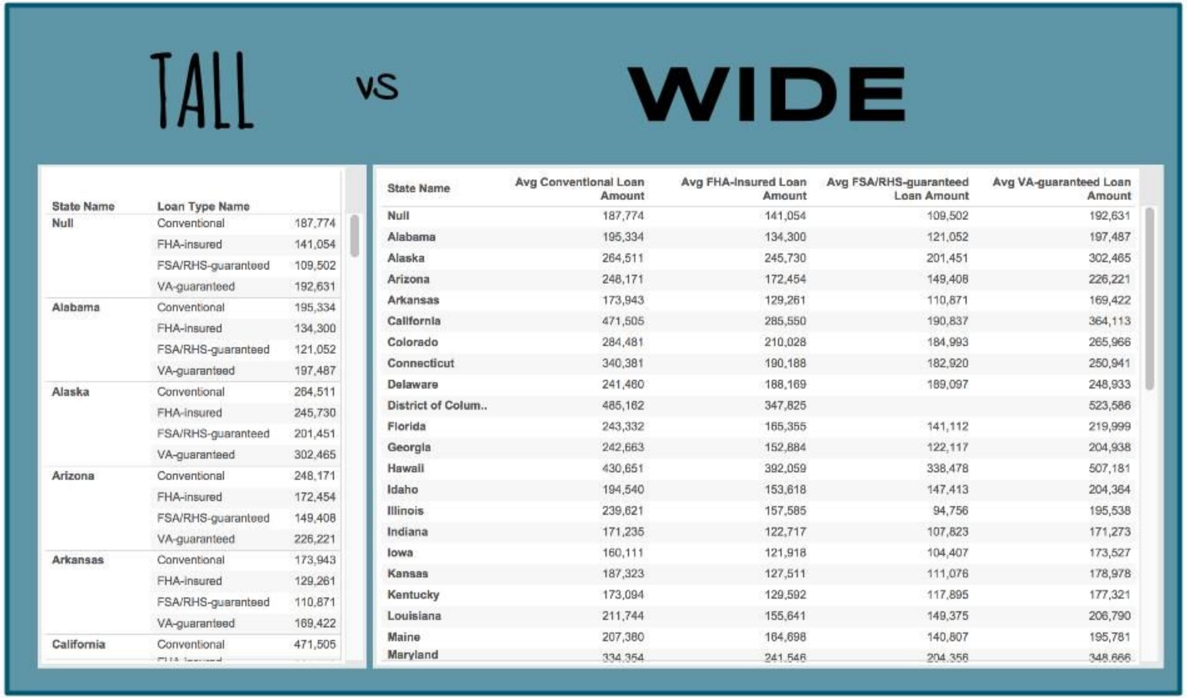
If this is false, **your analysis is invalid**.

Pandas Is Column-Oriented

- Columns are first-class
- Operations default to column-wise
- Rows are secondary

This is why `df.mean()` works without arguments.

Tall vs Wide Data


TALL
vs
WIDE

State Name	Loan Type Name	
Null	Conventional	187,774
	FHA-insured	141,054
	FSA/RHS-guaranteed	109,502
	VA-guaranteed	192,631
Alabama	Conventional	195,334
	FHA-insured	134,300
	FSA/RHS-guaranteed	121,052
	VA-guaranteed	197,487
Alaska	Conventional	264,511
	FHA-insured	245,730
	FSA/RHS-guaranteed	201,451
	VA-guaranteed	302,465
Arizona	Conventional	248,171
	FHA-insured	172,454
	FSA/RHS-guaranteed	149,408
	VA-guaranteed	226,221
Arkansas	Conventional	173,943
	FHA-insured	129,261
	FSA/RHS-guaranteed	110,871
	VA-guaranteed	169,422
California	Conventional	471,505
	FHA-insured	

State Name	Avg Conventional Loan Amount	Avg FHA-Insured Loan Amount	Avg FSA/RHS-guaranteed Loan Amount	Avg VA-guaranteed Loan Amount
Null	187,774	141,054	109,502	192,631
Alabama	195,334	134,300	121,052	197,487
Alaska	264,511	245,730	201,451	302,465
Arizona	248,171	172,454	149,408	226,221
Arkansas	173,943	129,261	110,871	169,422
California	471,505	285,550	190,837	364,113
Colorado	284,481	210,028	184,993	265,966
Connecticut	340,381	190,188	182,920	250,941
Delaware	241,460	188,169	189,097	248,933
District of Colum..	485,162	347,825		523,586
Florida	243,332	165,355	141,112	219,999
Georgia	242,663	152,884	122,117	204,938
Hawaii	430,651	392,059	338,478	507,181
Idaho	194,540	153,618	147,413	204,364
Illinois	239,621	157,585	94,756	195,538
Indiana	171,235	122,717	107,823	171,273
Iowa	160,111	121,918	104,407	173,527
Kansas	187,323	127,511	111,076	178,978
Kentucky	173,094	129,592	117,895	177,321
Louisiana	211,744	155,641	149,375	206,790
Maine	207,380	164,698	140,807	195,781
Maryland	334,354	241,546	204,356	348,666

Pandas can handle both, but GroupBy prefers tall data while Visualization prefers tall data.

One Variable per Column

Rule:

> **One column = one variable.**

Violations:

- “Jan_sales”, “Feb_sales”, …
- Multiple units in one column
- Encoded categories in strings

These force awkward pandas code later.

Index as a Coordinate System

Index defines:

- What makes a row unique
- How data aligns
- What joins mean

Index is **not decoration**.

The Real Data Loop

Load → Inspect → Question → Transform → Summarize → Repeat

not

Load → Model → Profit

Inspect Before You Touch Anything

Mandatory first steps:

```
df.shape
df.columns
df.dtypes
df.head()
df.info()
```

If you skip this, everything after is suspect.

Questions Come First, Code Second

Bad workflow:

> **“Which pandas method should I use?”**

Good workflow:

> **“What question am I answering?”**

Pandas methods are answers, not goals.

Typical Questions and Pandas Thinking

Question: “Are Groups Different?”

Translation:

- Identify grouping variable

- Summarize numeric columns

```
df.groupby("group").mean()
```

Question: “Is Something Unusual?”

Translation:

- Look at ranges
- Look at counts
- Look at missingness

```
df.describe()
```

Question: “Does This Change Over Time?”

Translation:

- Time-based index
- Sorting
- Aggregation

Pandas is designed for this.

Question: “Do Variables Move Together?”

Translation:

- Numeric columns
- Summaries or correlations

Later → visualization.

Why Visualization Comes Next

Tables hide:

- Distribution
- Outliers
- Structure

Visualization reveals them.

Summary

- Pandas is about **labeled data**
- Indexing and alignment matter
- Data analysis is iterative
- Pandas helps you ask questions

What Comes Next

- Exploratory Data Analysis
- Visualization
- Seeing patterns instead of guessing

Questions?

Jupyter Notebook Demo

In this jupyter notebook, we will be demoing the core ideas of Pandas: indexing and alignment, as well as inspecting and questioning data.

Resources

- Pandas Official Documentation
- Jake VanderPlas Python Data Science Handbook