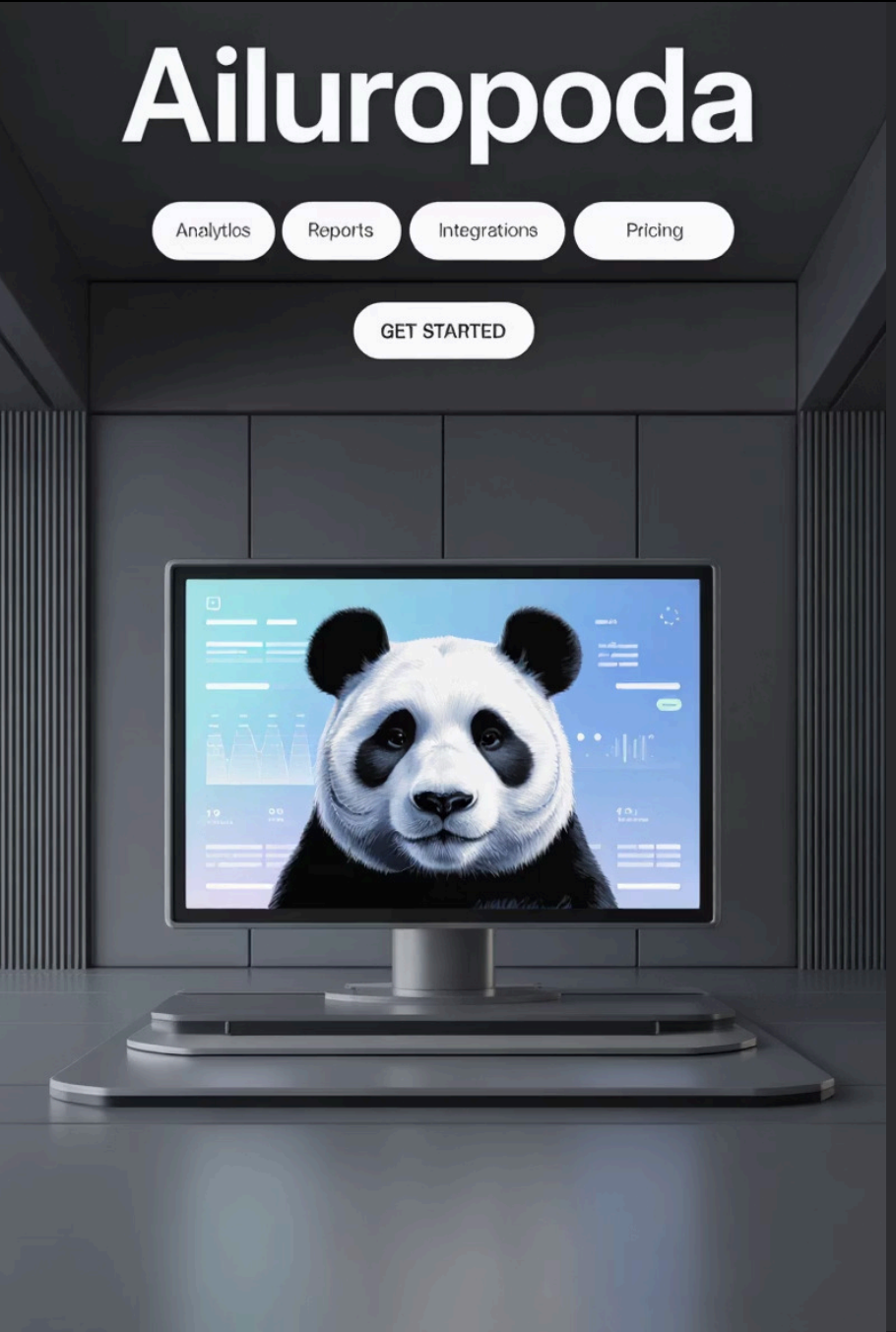


Pandas in Python

Introduction

What is Pandas?

Python's Data Powerhouse

Pandas is an open-source data analysis and manipulation library built on NumPy. It provides high-performance, easy-to-use data structures and data analysis tools for Python programming.

Originally developed by Wes McKinney in 2008, pandas has become the go-to tool for data scientists, analysts, and developers working with structured data.

Key Features:

- Fast and efficient data operations
- Flexible data alignment
- Integrated handling of missing data
- Powerful grouping and aggregation
- Easy data import/export

The Three Pillars of Pandas

Index

The backbone that labels and organizes your data. Think of it as row/column labels that make data retrieval fast and intuitive.

Series

One-dimensional labeled array capable of holding any data type. It's like a column in a spreadsheet with smart indexing.

DataFrame

Two-dimensional labeled data structure with columns of potentially different types. The heart of pandas data analysis.

Understanding the Index Object

The Index is pandas' labeling system that provides fast lookups and data alignment. It's immutable and hashable, making it perfect for efficient data operations.

01

Create Basic Index

```
import pandas as pd
index = pd.Index([1, 2, 3, 4, 5])
print(index)
```

02

Named Index

```
named_index = pd.Index(['A', 'B', 'C'], name='letters')
print(named_index)
```

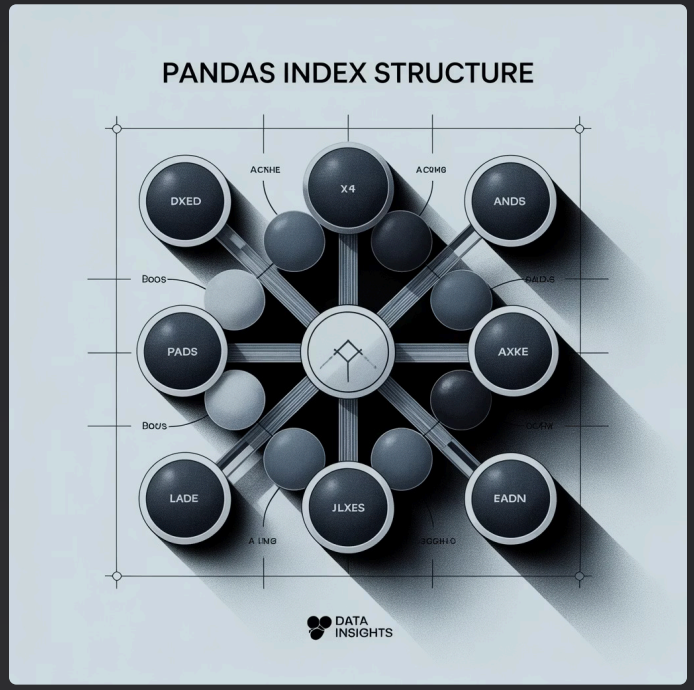
03

Date Index

```
date_index = pd.date_range('2024-01-01', periods=5)
print(date_index)
```

Key Properties

- index.name - Index name
- index.dtype - Data type
- index.shape - Dimensions
- index.is_unique - Uniqueness check



Mastering Series Objects

A Series is a one-dimensional labeled array that can hold any data type. It combines the power of NumPy arrays with intelligent indexing, making it perfect for time series, categorical data, and numerical analysis.

1

Creation

```
# From list
s1 = pd.Series([1, 2, 3, 4])

# From dictionary
s2 = pd.Series({'a': 1, 'b': 2, 'c': 3})

# With custom index
s3 = pd.Series([10, 20, 30],
               index=['x', 'y', 'z'])
```

2

Access

```
# By label
print(s3['x'])  # 10

# By position
print(s3.iloc[0])  # 10

# Multiple values
print(s3[['x', 'z']])
```

3

Operations

```
# Mathematical operations
result = s3 * 2

# Boolean indexing
filtered = s3[s3 > 15]

# Statistical methods
print(s3.mean(), s3.std())
```

Essential Series Methods

Data Inspection

```
series.head()     # First 5 elements
series.tail()     # Last 5 elements
series.info()     # Data types & memory
series.describe()  # Statistical summary
```

Data Cleaning

```
series.dropna()      # Remove NaN values
series.fillna(0)     # Fill NaN with 0
series.drop_duplicates()  # Remove duplicates
series.astype('int')  # Change data type
```

Data Analysis

```
series.value_counts()  # Count unique values
series.unique()       # Get unique values
series.sort_values()   # Sort by values
series.sort_index()    # Sort by index
```

These methods form the foundation of data exploration and cleaning workflows. Practice using them with different data types to build muscle memory for real-world analysis tasks.

DataFrame: The Heart of Pandas

DataFrames are two-dimensional labeled data structures with columns of potentially different types. Think of them as spreadsheets or SQL tables, but with much more functionality and flexibility.

From Dictionary

```
data = {
'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25, 30, 35],
'City': ['NYC', 'LA', 'Chicago']
}
df = pd.DataFrame(data)
```

From File

```
# CSV files
df = pd.read_csv('data.csv')

# Excel files
df = pd.read_excel('data.xlsx')

# JSON files
df = pd.read_json('data.json')
```

1 2 3

From Lists

```
data = [['Alice', 25, 'NYC'],
['Bob', 30, 'LA'],
['Charlie', 35, 'Chicago']]
df = pd.DataFrame(data,
columns=['Name', 'Age', 'City'])
```

DataFrame Anatomy

- **Columns:** Each column is a Series
- **Index:** Row labels (default: 0, 1, 2...)
- **Values:** The actual data in 2D array
- **dtypes:** Data type of each column

DataFrame Selection and Indexing

Accessing data in DataFrames requires understanding multiple selection methods. Each approach serves different use cases, from simple column selection to complex conditional filtering.

1 Column Selection

```
# Single column (returns Series)
ages = df['Age']

# Multiple columns (returns DataFrame)
subset = df[['Name', 'Age']]

# All columns except one
without_city = df.drop('City', axis=1)
```

2 Row Selection with .loc[]

```
# By label/index
first_row = df.loc[0]

# Multiple rows
first_two = df.loc[0:1]

# Conditional selection
adults = df.loc[df['Age'] >= 30]
```

3 Position-based with .iloc[]

```
# By integer position
first_row = df.iloc[0]

# Slice rows and columns
subset = df.iloc[0:2, 1:3]

# Last row
last_row = df.iloc[-1]
```

Advanced DataFrame Operations

Filtering & Querying

```
# Boolean indexing
young_people = df[df['Age'] < 30]

# Multiple conditions
ny_young = df[(df['Age'] < 30) &
              (df['City'] == 'NYC')]

# Using query method
result = df.query('Age > 25 and City == "LA"')
```

Grouping & Aggregation

```
# Group by city
city_groups = df.groupby('City')

# Aggregate functions
avg_age_by_city = df.groupby('City')['Age'].mean()

# Multiple aggregations
stats = df.groupby('City').agg({
    'Age': ['mean', 'max', 'min']
})
```

Sorting & Ranking

```
# Sort by values
sorted_df = df.sort_values('Age',
                           ascending=False)

# Sort by multiple columns
multi_sort = df.sort_values(['City', 'Age'])

# Rank values
df['Age_Rank'] = df['Age'].rank()
```

These operations form the core of data analysis workflows. Master them to efficiently explore, clean, and transform your datasets for meaningful insights.

Your Pandas Journey Continues

Foundation Built

You now understand the core pandas objects: Index for labeling, Series for 1D data, and DataFrame for 2D analysis. These are your building blocks for any data project.

Practice Essential

Load real datasets and experiment with selection, filtering, and grouping operations. The more you practice these fundamentals, the more intuitive they become.

Next Steps

Explore additional topics. Your pandas foundation makes these concepts accessible.

Key Takeaways

- Index objects provide fast, labeled data access
- Series combine arrays with intelligent indexing
- DataFrames are your primary tool for structured data
- Selection methods (.loc, .iloc, []) serve different needs
- Practice builds intuition and confidence

> "Data is the new oil, and pandas is your refinery. Master these fundamentals, and you'll transform raw data into valuable insights."