Week 3, Lecture 5

Pandas Fundamentals - DataFrames, Series, Data Loading

Lecture Overview

Pandas is the cornerstone library for data manipulation and analysis in Python. Built on NumPy, it provides high-level data structures and functions designed to make working with structured data fast, easy, and expressive. This lecture introduces pandas' core data structures: Series and DataFrames, and demonstrates how to load, inspect, and perform basic operations on data.

Topics Covered:

• Series: one-dimensional labeled arrays

• DataFrames: two-dimensional labeled data structures

```
• Data loading from CSV, Excel, JSON
```

• Basic data inspection and selection

• Indexing and slicing

• Data cleaning basics

1. Introduction to Pandas

1.1 What is Pandas?

Pandas provides data structures optimized for working with tabular data. The two primary data structures are:

• Series: One-dimensional labeled array (like a column in a spreadsheet)

• DataFrame: Two-dimensional labeled data structure (like a spreadsheet or SQL table)

```
import pandas as pd
import numpy as np
print("Pandas Information")
print("="*70)
print(f"Pandas version: {pd.__version__}")
print(f"NumPy version: {np.__version__}")
```

Pandas Information
== Pandas version: 2.1.0 NumPy version: 1.24.3

2. Series - One-Dimensional Data

2.1 Creating Series

```
# Create Series from list
print("Creating Series")
print("="*70)
# From list
s1 = pd.Series([10, 20, 30, 40, 50])
print(f"Series from list:")
print(s1)
print(f"\nData type: {s1.dtype}")
print(f"Shape: {s1.shape}")
# With custom index
s2 = pd.Series([10, 20, 30, 40, 50],                index=['a', 'b', 'c', 'd', 'e'])
print(f"\nSeries with custom index:")
print(s2)
# From dictionary
data_dict = {'apple': 3, 'banana': 5, 'cherry': 2, 'date': 7}
s3 = pd.Series(data_dict)
print(f"\nSeries from dictionary:")
print(s3)
```

```
Creating Series ======================================================================
Series from list: 0    10 1    20 2    30 3    40 4    50 dtype: int64  Data type:
int64 Shape: (5,)  Series with custom index: a    10 b    20 c    30 d    40 e    50
dtype: int64  Series from dictionary: apple     3 banana    5 cherry    2 date      7
dtype: int64
```

2.2 Series Operations

```
# Series operations s =
pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])
print("Series Operations")
print("="*70)
# Access by index
print(f"Access by label s['b']: {s['b']}")
print(f"Access by position s[1]: {s[1]}")
# Slicing
print(f"\nSlice s['b':'d']:")
print(s['b':'d'])
# Boolean indexing
print(f"\nBoolean indexing s[s > 25]:")
print(s[s > 25])
# Arithmetic operations
print(f"\nArithmetic: s * 2:")
print(s * 2)
# Statistical operations
```

```
print(f"\nStatistics:")
print(f"  Mean: {s.mean()}")
print(f"  Median: {s.median()}")
print(f"  Std: {s.std():.2f}")
print(f"  Sum: {s.sum()}")
print(f"  Min: {s.min()}, Max: {s.max()}")
```

Series Operations
========== Access by label s['b']: 20 Access by position s[1]: 20 Slice s['b':'d']: b 20 c 30 d 40 dtype: int64 Boolean indexing s[s > 25]: c 30 d 40 e 50 dtype: int64 Arithmetic: s * 2: a 20 b 40 c 60 d 80 e 100 dtype: int64 Statistics: Mean: 30.0 Median: 30.0 Std: 15.81 Sum: 150 Min: 10, Max: 50

3. DataFrames - Two-Dimensional Data

3.1 Creating DataFrames

```
# Create DataFrame from dictionary
print("Creating DataFrames")
print("="*70)
# From dictionary of lists
data = {     'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],     'Age': [25, 30, 35, 28, 32],     'City': ['New York', 'Paris', 'London', 'Tokyo', 'Sydney'],     'Salary': [70000, 80000, 75000, 85000, 90000] }
df = pd.DataFrame(data)
print("DataFrame from dictionary:")
print(df)
print(f"\nShape: {df.shape}")
print(f"Columns: {list(df.columns)}")
print(f"Index: {list(df.index)}")
# From list of dictionaries
data_list = [     {'Name': 'Alice', 'Age': 25, 'City': 'New York'},     {'Name': 'Bob', 'Age': 30, 'City': 'Paris'},     {'Name': 'Charlie', 'Age': 35, 'City': 'London'} ]
df2 = pd.DataFrame(data_list)
print(f"\nDataFrame from list of dicts:")
print(df2)
```

Creating DataFrames
========== DataFrame from dictionary: Name Age City Salary 0 Alice 25 New York 70000 1 Bob 30 Paris 80000 2 Charlie 35 London 75000 3 David 28 Tokyo 85000 4 Eve 32 Sydney 90000 Shape: (5, 4) Columns: ['Name', 'Age', 'City', 'Salary'] Index: [0, 1, 2, 3, 4] DataFrame from list of dicts: Name Age City 0 Alice 25 New York 1 Bob 30 Paris 2 Charlie 35 London

3.2 Data Inspection

```
# Data inspection methods
print("Data Inspection")
print("="*70)
```

```
# First/last rows
print("First 3 rows:")
print(df.head(3))
print("\nLast 2 rows:")
print(df.tail(2))
# Basic info
print("\nDataFrame info:")
print(df.info())
# Summary statistics
print("\nSummary statistics:")
print(df.describe())
# Data types
print("\nData types:")
print(df.dtypes)
```

```
Data Inspection ======================================================================
First 3 rows:       Name  Age      City  Salary 0    Alice   25  New York   70000 1
Bob   30     Paris   80000 2  Charlie   35    London   75000  Last 2 rows:     Name
Age    City  Salary 3  David   28   Tokyo   85000 4    Eve   32  Sydney   90000
DataFrame info: <class 'pandas.core.frame.DataFrame'> RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):  #   Column  Non-Null Count  Dtype  ---  ------  ------
--------  -----   0   Name    5 non-null      object  1   Age     5 non-null      int64
2   City    5 non-null      object  3   Salary  5 non-null      int64  dtypes:
int64(2), object(2)
memory usage: 288.0+ bytes  Summary statistics:              Age        Salary count
5.000000      5.000000 mean   30.000000  80000.000000 std     3.807887   7905.694150
min    25.000000  70000.000000 25%    28.000000  75000.000000 50%    30.000000
80000.000000 75%    32.000000  85000.000000 max    35.000000  90000.000000  Data types:
Name      object Age        int64 City      object Salary     int64 dtype: object
```

4. Selecting and Indexing

4.1 Column Selection

```
# Column selection
print("Column Selection")
print("="*70)
# Single column (returns Series)
print("Single column df['Age']:")
print(df['Age'])
print(f"Type: {type(df['Age'])}")
# Multiple columns (returns DataFrame)
print("\nMultiple columns df[['Name', 'Salary']]:")
print(df[['Name', 'Salary']])
print(f"Type: {type(df[['Name', 'Salary']])}")
# Add new column df['Tax'] = df['Salary'] * 0.2
print("\nAfter adding Tax column:")
print(df[['Name', 'Salary', 'Tax']])
```

Column Selection
========== Single column df['Age']: 0 25 1 30 2 35 3 28 4 32 Name: Age, dtype: int64 Type: <class 'pandas.core.series.Series'> Multiple columns df[['Name', 'Salary']]: Name Salary 0 Alice 70000 1 Bob 80000 2 Charlie 75000 3 David 85000 4 Eve 90000 Type: <class 'pandas.core.frame.DataFrame'> After adding Tax column: Name Salary Tax 0 Alice 70000 14000.0 1 Bob 80000 16000.0 2 Charlie 75000 15000.0 3 David 85000 17000.0 4 Eve 90000 18000.0

4.2 Row Selection with loc and iloc

```
# Row selection
print("Row Selection")
print("="*70)
# loc: label-based indexing
print("loc[0] - select by index label:")
print(df.loc[0])
print("\nloc[1:3] - slice by labels:")
print(df.loc[1:3, ['Name', 'Age']])
# iloc: integer position-based indexing
print("\niloc[0] - select by position:")
print(df.iloc[0])
print("\niloc[1:3] - slice by position:")
print(df.iloc[1:3, [0, 1]])
# Boolean indexing
print("\nBoolean indexing - Age > 30:")
print(df[df['Age'] > 30][['Name', 'Age']])
```

Row Selection
========== loc[0] - select by index label: Name Alice Age 25 City New York Salary 70000 Tax 14000 Name: 0, dtype: object loc[1:3] - slice by labels: Name Age 1 Bob 30 2 Charlie 35 3 David 28 iloc[0] - select by position: Name Alice Age 25 City New York Salary 70000 Tax 14000 Name: 0, dtype: object iloc[1:3] - slice by position: Name Age 1 Bob 30 2 Charlie 35 Boolean indexing - Age > 30: Name Age 2 Charlie 35 4 Eve 32

```
5. Loading Data from Files
```

5.1 Reading CSV Files

```
# Create sample CSV import io  csv_data = """Name,Age,Department,Salary
Alice,25,Engineering,70000 Bob,30,Marketing,65000 Charlie,35,Engineering,80000
David,28,Sales,60000 Eve,32,Engineering,75000"""  # Read CSV from string
(simulating file)
df_csv = pd.read_csv(io.StringIO(csv_data))
print("Reading CSV")
print("="*70)
print("Loaded DataFrame:")
print(df_csv)
```

```
# Common parameters
print("\nCSV reading options:")
print("  - pd.read_csv('file.csv')
# Basic read")
print("  - pd.read_csv('file.csv', sep=';')
# Different separator")
print("  - pd.read_csv('file.csv', header=None)
# No header")
print("  - pd.read_csv('file.csv', index_col=0)
# Set index column")
print("  - pd.read_csv('file.csv', usecols=['A','B'])
# Select columns")
print("  - pd.read_csv('file.csv', nrows=100)
# Limit rows")
Reading CSV ========================================================================
Loaded DataFrame:       Name  Age   Department  Salary 0    Alice   25  Engineering
70000 1      Bob   30    Marketing   65000 2  Charlie   35  Engineering   80000 3
David   28        Sales   60000 4      Eve   32  Engineering   75000  CSV reading
options:   - pd.read_csv('file.csv')
# Basic read   -
pd.read_csv('file.csv', sep=';')
# Different separator   -
pd.read_csv('file.csv', header=None)
# No header   -
pd.read_csv('file.csv', index_col=0)
# Set index column   -
pd.read_csv('file.csv',
usecols=['A','B'])
# Select columns   -
pd.read_csv('file.csv', nrows=100)
# Limit rows
```

5.2 Writing Data

```
# Writing data to files
print("Writing Data")
print("="*70)
# Save to CSV df_csv.to_csv('output.csv', index=False)
print("Saved to output.csv")
# Save to Excel (requires openpyxl)
# df_csv.to_excel('output.xlsx', sheet_name='Data', index=False)
# print("Saved to output.xlsx")
# Save to JSON df_csv.to_json('output.json', orient='records', indent=2)
print("Saved to output.json")
# Writing options
print("\nWriting options:")
```

```
print("  - to_csv(index=False)
# Don't write index")
print("  - to_csv(sep='\t')
# Tab-separated")
print("  - to_csv(columns=['A', 'B'])
# Select columns")
print("  - to_excel(sheet_name='Data')
# Excel with sheet name")
print("  - to_json(orient='records')
# JSON format")
Writing Data ========================================================================
Saved to output.csv Saved to output.json  Writing options:   - to_csv(index=False)
# Don't write index   - to_csv(sep='\t')
# Tab-separated   - to_csv(columns=['A', 'B'])
# Select columns   - to_excel(sheet_name='Data')
# Excel with sheet name   - to_json(orient='records')
# JSON format
```

6. Basic Data Cleaning

6.1 Handling Missing Data

```
# Create DataFrame with missing values
data_missing = {     'Name': ['Alice', 'Bob', None, 'David', 'Eve'],     'Age': [25,
None, 35, 28, 32],     'Score': [85.5, 90.0, None, 88.5, 92.0] }
df_missing = pd.DataFrame(data_missing)
print("Handling Missing Data")
print("="*70)
print("Original DataFrame:")
print(df_missing)
# Check for missing values
print("\nMissing values per column:")
print(df_missing.isnull().sum())
print("\nRows with any missing values:")
print(df_missing[df_missing.isnull().any(axis=1)])
# Drop rows with missing values df_dropped = df_missing.dropna()
print("\nAfter dropna():")
print(df_dropped)
# Fill missing values df_filled = df_missing.fillna({'Age':
df_missing['Age'].mean(),                                   'Score':
df_missing['Score'].mean(),                                   'Name':
'Unknown'})
print("\nAfter fillna():")
print(df_filled)
```

Handling Missing Data
========== Original DataFrame: Name Age Score 0 Alice 25.0 85.5 1 Bob NaN 90.0 2 None 35.0 NaN 3 David 28.0 88.5 4 Eve 32.0 92.0 Missing values per column: Name 1 Age 1 Score 1 dtype: int64 Rows with any missing values: Name Age Score 1 Bob NaN 90.0 2 None 35.0 NaN After dropna(): Name Age Score 0 Alice 25.0 85.5 3 David 28.0 88.5 4 Eve 32.0 92.0 After fillna(): Name Age Score 0 Alice 25.0 85.5 1 Bob 30.0 90.0 2 Unknown 35.0 89.0 3 David 28.0 88.5 4 Eve 32.0 92.0

6.2 Removing Duplicates

```
# Create DataFrame with duplicates
data_dup = {     'Name': ['Alice', 'Bob', 'Alice', 'Charlie', 'Bob'],     'Age': [25, 30, 25, 35, 30],     'City': ['NYC', 'Paris', 'NYC', 'London', 'Paris'] }
df_dup = pd.DataFrame(data_dup)
print("Removing Duplicates")
print("="*70)
print("Original DataFrame:")
print(df_dup)
# Check for duplicates
print(f"\nDuplicate rows: {df_dup.duplicated().sum()}")
print("\nDuplicate rows marked:")
print(df_dup[df_dup.duplicated()])
# Remove duplicates df_unique = df_dup.drop_duplicates()
print("\nAfter drop_duplicates():")
print(df_unique)
# Keep last occurrence df_unique_last = df_dup.drop_duplicates(keep='last')
print("\nKeep last occurrence:")
print(df_unique_last)
```

Removing Duplicates
========== Original DataFrame: Name Age City 0 Alice 25 NYC 1 Bob 30 Paris 2 Alice 25 NYC 3 Charlie 35 London 4 Bob 30 Paris Duplicate rows: 2 Duplicate rows marked: Name Age City 2 Alice 25 NYC 4 Bob 30 Paris After drop_duplicates(): Name Age City 0 Alice 25 NYC 1 Bob 30 Paris 3 Charlie 35 London Keep last occurrence: Name Age City 2 Alice 25 NYC 3 Charlie 35 London 4 Bob 30 Paris

Summary

Key Takeaways:

• Series: One-dimensional labeled array for single columns

• DataFrame: Two-dimensional table with labeled rows and columns

```
• pd.read_csv()
and pd.read_excel()
load data from files
```

• loc[] for label-based indexing, iloc[] for position-based

• dropna() removes missing values, fillna() fills them

• drop_duplicates() removes duplicate rows

• head(), tail(), info(), describe() for data inspection

Practice Exercises

1. Load a CSV file and display the first 10 rows and summary statistics

2. Filter a DataFrame to show only rows where a numeric column exceeds a threshold

3. Handle missing values by filling with column means

```
4. Create a new column calculated from existing columns
```

5. Remove duplicate rows based on specific columns

6. Export cleaned data to both CSV and JSON formats

Next Lecture

In Lecture 6, we'll explore advanced data analysis with Pandas including groupby operations, merging datasets, time series, and data visualization with Matplotlib.