Pandas: Complete Lesson Summary (Concise but Comprehensive)

A compact walkthrough of all lessons, subtopics, and applied examples from the course material.

# Preface

This document summarizes every lesson and subtopic from the provided course material on Python’s pandas library. Each section preserves full topical coverage while reducing verbosity. Every concept includes a short explanation and a minimal, runnable code snippet to reinforce understanding.

# Introduction to Pandas & NumPy

Pandas is a high-level data analysis library built on top of NumPy. Its core objects are Series (1D) and DataFrame (2D tabular data). NumPy provides n-dimensional arrays and vectorized math; pandas layers labeling, alignment, and rich table operations.

Install and import:

pip install numpy pandas  
  
import numpy as np  
import pandas as pd

NumPy arrays vs Python lists: vectorized math without manual loops.

toyPrices = np.array([5, 8, 3, 6])  
print(toyPrices - 2) # [3 6 1 4]

Series and custom indices:

ages = np.array([13, 25, 19])  
ser = pd.Series(ages, index=['Emma', 'Swetha', 'Serajh'])  
print(ser)

Create a DataFrame and set an index:

df = pd.DataFrame([  
 ['John Smith','123 Main St',34],  
 ['Jane Doe', '456 Maple Ave',28],  
 ['Joe Schmo', '789 Broadway',51]  
], columns=['name','address','age'])  
  
print(df)  
print(df.set\_index('name'))

# Lesson 1 — Lambda Functions (Python refresher)

Lambda functions are single-line, anonymous functions—useful for concise transformations and simple conditionals.

## Syntax & basics

Define in one line; pass input after ':' and return the expression value.

add\_two = lambda x: x + 2  
print(add\_two(3)) # 5

## Substring checks with 'in'

Use membership tests on strings.

contains\_a = lambda word: 'a' in word  
print(contains\_a('banana')) # True

## Length checks

Use len() in expressions.

long\_string = lambda s: len(s) > 12  
print(long\_string('photosynthesis')) # True

## Indexing & suffix tests

Access last char with s[-1].

ends\_in\_a = lambda s: s[-1] == 'a'  
print(ends\_in\_a('data')) # True

## Conditionals in lambdas

Use `A if cond else B` inline.

double\_or\_zero = lambda n: n\*2 if n > 10 else 0  
print(double\_or\_zero(15)) # 30

## Even / odd via modulo

Modulo 2 determines parity.

even\_or\_odd = lambda n: 'even' if n % 2 == 0 else 'odd'

## Multiples & comparisons

Test multiples with modulo; use comparators (>, >=, ==, !=).

multiple\_of\_three = lambda n: 'multiple of three' if n % 3 == 0 else 'not a multiple'

## Ones’ place & powers

Use % 10 and \*\* for exponent.

ones\_place = lambda n: n % 10  
  
double\_square = lambda n: 2 \* (n\*\*2)

## Random additions

Produce variability with randint.

import random  
add\_random = lambda n: n + random.randint(1, 10)  
print(add\_random(5))

# Lesson 2 — Creating, Loading, and Selecting Data

## Import pandas

Conventional alias is pd.

import pandas as pd

## Create a DataFrame (from dict)

Keys become column names, values are equal-length lists.

df = pd.DataFrame({  
 'name': ['John Smith', 'Jane Doe', 'Joe Schmo'],  
 'address': ['123 Main St.', '456 Maple Ave.', '789 Broadway'],  
 'age': [34, 28, 51]  
})  
print(df)

## Create a DataFrame (from list of lists)

Control column order explicitly via columns=.

df = pd.DataFrame([  
 ['San Diego', 100],  
 ['Los Angeles', 120],  
 ['San Francisco', 90]  
], columns=['Location', 'Employees'])

## CSV basics

CSV is text with comma-separated columns; header row then data rows.

name,cake\_flavor,frosting\_flavor,topping  
Chocolate Cake,chocolate,chocolate,chocolate shavings

## Load & save CSV

Use read\_csv and to\_csv.

df = pd.read\_csv('sample.csv')  
print(df.head())  
  
df.to\_csv('out.csv', index=False)

## Inspect a DataFrame

Peek at shape, columns, dtypes.

print(df.head())  
print(df.info())

## Select columns

Select one as Series; multiple as DataFrame.

ages = df['age'] # Series  
subset = df[['name','age']] # DataFrame

## Select rows by position (iloc)

Zero-based indexing; slicing excludes stop.

row2 = df.iloc[2]  
last3 = df.iloc[-3:]

## Filter rows with logic

Boolean masks with ==, >, <, !=; combine with & and |; use .isin().

jan = df[df.month == 'January']  
late\_winter = df[df.month.isin(['January','February','March'])]  
march\_april = df[(df.month=='March') | (df.month=='April')]

## Resetting indices

After filtering, use reset\_index to make them consecutive.

subset = df.loc[[1,3,5]]  
subset = subset.reset\_index(drop=True)

# Lesson 3 — Modifying DataFrames

## Add constant or list-based columns

Assign lists of equal length or a single scalar.

df['In Stock?'] = True  
  
df['Sold in Bulk?'] = ['Yes','Yes','No','No']

## Add computed columns

Vectorized math across columns.

df['Sales Tax'] = df['Price'] \* 0.075  
  
df['Margin'] = df['Price'] - df['Cost to Manufacture']

## Column-wise apply

Transform values with Series.apply and Python/string functions.

df['Lowercase Name'] = df['Name'].apply(str.lower)

## Row-wise apply (axis=1)

Access multiple columns per row; great for conditional logic.

total\_earned = lambda row: (  
 40\*row['hourly\_wage'] + (row['hours\_worked']-40)\*1.5\*row['hourly\_wage']  
) if row['hours\_worked']>40 else row['hours\_worked']\*row['hourly\_wage']  
  
df['total\_earned'] = df.apply(total\_earned, axis=1)

## Renaming columns (.columns)

Replace all column names—ensure correct order.

df.columns = ['ID','Title','Category','Year Released','Rating']

## .rename (safer targeted renames)

Map old→new; use inplace=True to modify the same frame.

df.rename(columns={'name': 'movie\_title'}, inplace=True)

# Lesson 4 — Aggregates in Pandas

## Column statistics

Compute summaries like mean/median/std/nunique/unique.

print(df['age'].mean())  
print(df['state'].nunique())  
print(df['color'].unique())

## GroupBy aggregation

Aggregate by categories; always reset\_index for a flat result.

orders = pd.read\_csv('orders.csv')  
shoe\_counts = orders.groupby(['shoe\_type','shoe\_color']).id.count().reset\_index()  
print(shoe\_counts.head())

## Pivot tables

Reshape long→wide for 2D comparisons; pivot then reset\_index.

pivoted = shoe\_counts.pivot(  
 index='shoe\_type', columns='shoe\_color', values='id'  
).reset\_index()  
print(pivoted.head())

# Lesson 5 — Working with Multiple DataFrames

## Merges (SQL-style joins)

Combine on shared columns/keys; default is inner. Use how='left'|'right'|'outer'.

visits = pd.read\_csv('visits.csv', parse\_dates=[1])  
checkouts = pd.read\_csv('checkouts.csv', parse\_dates=[1])  
  
v\_to\_c = visits.merge(checkouts) # inner on matching columns  
v\_to\_c['time'] = v\_to\_c['checkout\_time'] - v\_to\_c['visit\_time']  
print(v\_to\_c['time'].mean())

## Specify join keys

Use left\_on/right\_on for differently named keys.

a.merge(b, left\_on='user\_id', right\_on='id', how='left')

## Concatenate rows

Stack frames with identical schemas.

menu = pd.concat([bakery, ice\_cream])  
print(menu.head())

# Overview of Lessons & Subtopics Included

• Lesson 1 — Lambda Functions: syntax & basics; substring checks; length checks; endswith; inline conditionals; even/odd; multiples; ones’ place & powers; randomness.

• Lesson 2 — Creating, Loading, and Selecting Data: import; create from dict/list; CSV basics; read\_csv/to\_csv; inspect (head/info); select cols; select rows (iloc/slices); logical filters (&, |, isin); reset\_index.

• Lesson 3 — Modifying DataFrames: add constant/list columns; computed columns; column-wise apply; row-wise apply (axis=1); rename with .columns; targeted .rename.

• Lesson 4 — Aggregates in Pandas: column statistics; groupby aggregations; pivot tables.

• Lesson 5 — Working with Multiple DataFrames: merges (inner/left/right/outer); specifying keys; concatenation; time deltas example.