

## Lesson 01: Python Setup

Data Science with Python – BSc Course

Data Science Program

45 Minutes

## After this lesson, you will be able to:

- Install Anaconda and launch Jupyter Notebook
- Create and execute Python code cells
- Understand and use basic data types (int, float, str, bool)
- Store stock prices and financial data in variables

**Finance Application:** Store and manipulate stock prices using Python variables.

---

Foundation lesson – everything builds on these basics

# The Jupyter Notebook Environment

## Why Jupyter?

- Interactive code execution
- Mix code, output, and documentation
- Industry standard for data science
- Perfect for financial analysis

## Getting Started:

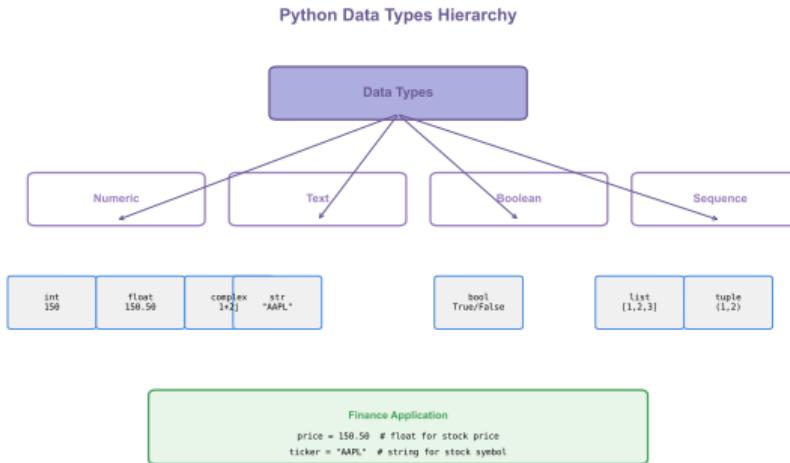
- ① Install Anaconda from [anaconda.com](https://www.anaconda.com)
- ② Launch Jupyter Notebook
- ③ Create New → Python 3

The screenshot shows the Jupyter Notebook interface. At the top, there's a purple header bar with the title "Jupyter Notebook Interface" and a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Help. Below the header is a code cell labeled "In [1]". The code in the cell is:# Calculate stock price change
initial\_price = 150.00
final\_price = 165.50
change = final\_price - initial\_price
print(f"Change: \${change:.2f}")The output cell below it is labeled "Out[1]". It displays the result of the calculation: "Change: \$15.50". A vertical blue arrow points from the "Input Code" label to the code cell, and another blue arrow points from the "Output Result" label to the output cell.

---

Jupyter = Julia + Python + R – supports multiple languages

# Python Data Types



Use `type(variable)` to check any variable's type

## Four Basic Types:

`int` – Integers (whole numbers)

```
price_shares = 100
```

`float` – Decimals

```
stock_price = 185.50
```

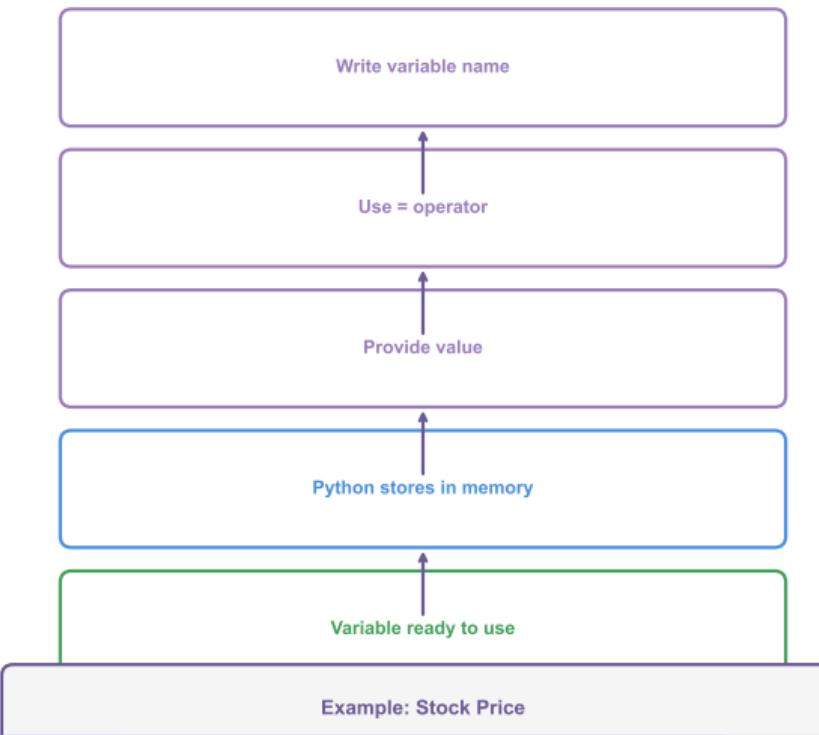
`str` – Text strings

```
ticker = "AAPL"
```

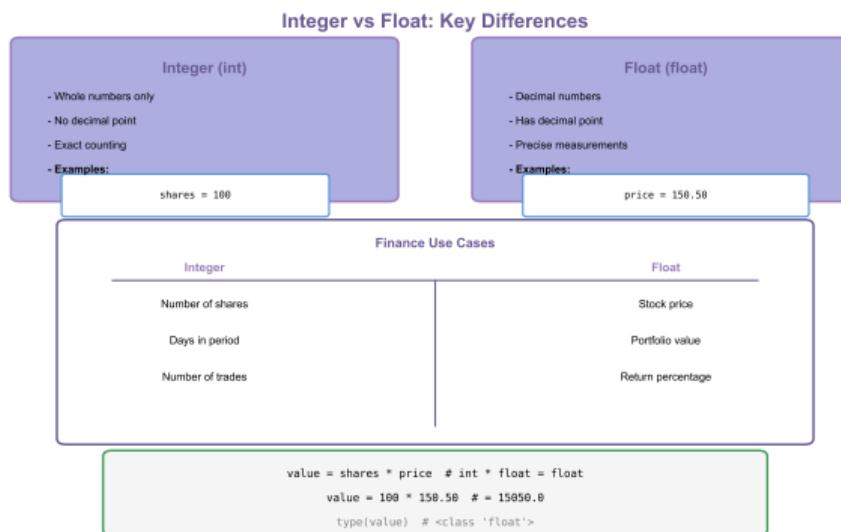
`bool` – True/False

```
is_profitable = True
```

## Variable Assignment Process



# Integers vs Floats in Finance



## When to use integers:

- Number of shares: `shares = 100`
- Trading days: `days = 252`
- Position count: `positions = 5`

## When to use floats:

- Stock prices: `price = 185.50`
- Returns: `ret = 0.0523`
- Percentages: `pct = 5.23`

Division always returns float:  $10 / 3 = 3.333\dots$

# String Operations for Finance

String Operations in Python			
Operation	Syntax	Example	Result
Concatenation	<code>ticker1 + ticker2</code>	<code>"AAPL" + "MSFT"</code>	<code>"AAPLMSFT"</code>
Repetition	<code>ticker * 3</code>	<code>"XYZ" * 3</code>	<code>"XYZXYZXYZ"</code>
Upper/Lower	<code>ticker.upper()</code>	<code>"aapl".upper()</code>	<code>"AAPL"</code>
Slicing	<code>ticker[0:2]</code>	<code>"APPLE"[0:2]</code>	<code>"AP"</code>
Length	<code>len(ticker)</code>	<code>len("AAPL")</code>	<code>4</code>
Format	<code>f-string</code>	<code>f"Price: \${price}"</code>	<code>"Price: \$150.50"</code>

## Common String Operations:

`ticker = "AAPL"`

`ticker.upper() → "AAPL"`

`ticker.lower() → "aapl"`

## String Formatting:

`f"Price: ${price}"`

## Concatenation:

`"NASDAQ:" + ticker`

F-strings (`f"..."`) are the modern way to format strings

# Boolean Logic for Trading Decisions

## Boolean Logic & Truth Tables

AND Operator		A and B
A	B	
True	True	True
True	False	False
False	True	False
False	False	False

OR Operator		A or B
A	B	
True	True	True
True	False	True
False	True	True
False	False	False

NOT Operator	
A	not A
True	False
False	True

### Finance Example: Buy Signal

```
price = 145.00
volume = 1000000
buy = (price < 150) and (volume > 500000) # True
print("Buy signal: (buy)") # Buy signal: True
```

## Comparison Operators:

- `>` greater than
- `<` less than
- `>=` greater or equal
- `==` equal to
- `!=` not equal

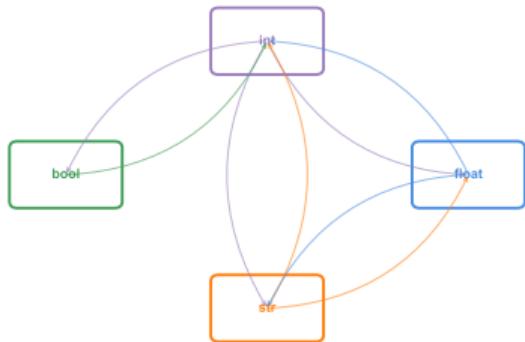
## Example:

`price > 200 → True/False`

**Booleans are essential for trading rule logic**

# Type Conversion

## Type Conversion (Casting)



### Conversion Examples

```
int("150")      # "150" -> 150  
float("150.50") # "150.50" -> 150.5  
str(150)        # 150 -> "150"
```

```
int(150.99)    # 150.99 -> 150 (truncates!)  
bool(0)         # 0 -> False  
bool(150)       # 150 -> True
```

## Converting Between Types:

`int("100") → 100`

`float("185.5") → 185.5`

`str(185.5) → "185.5"`

`bool(1) → True`

## Finance Use Case:

Reading prices from CSV files  
(data comes as strings)

**Be careful: `int("185.5")` fails – convert to float first**

# Why Python Instead of Excel?

## Python vs Excel for Finance

Feature	Excel	Python
Data Size	Limited (1M rows)	Unlimited
Automation	Manual/Macros	Full Scripts
Reproducibility	Low	High
Version Control	Difficult	Git Integration
Visualization	Built-in Charts	Custom Libraries
Speed	Slow (large data)	Fast
Learning Curve	Easy	Moderate

Best for Excel:

- Quick calculations

Best for Python:

- Large datasets (>100K rows)

## Hands-on Exercise (25 min)

Create a Jupyter notebook and complete:

① Create variables for a stock portfolio:

- `ticker = "AAPL"` (string)
- `shares = 50` (integer)
- `buy_price = 150.25` (float)
- `current_price = 185.50` (float)

② Calculate portfolio metrics:

- Total investment: `shares * buy_price`
- Current value: `shares * current_price`
- Profit: current value - investment
- Return %: `(profit / investment) * 100`

③ Create a boolean: `is_profitable = profit > 0`

④ Print results using f-strings

---

Save your notebook – we'll build on this next lesson

# Lesson Summary

## Key Takeaways:

- Jupyter Notebook is our development environment
- Four basic types: int, float, str, bool
- Variables store values with descriptive names
- Type conversion needed when reading external data
- Python handles large datasets better than Excel

**Next Lesson:** Data Structures (Lists and Dictionaries)

---

**Practice:** Experiment with different variable types in Jupyter

## Lesson 02: Data Structures

Data Science with Python – BSc Course

Data Science Program

45 Minutes

# Learning Objectives

**After this lesson, you will be able to:**

- Create and manipulate Python lists
- Access elements using indexing and slicing
- Build dictionaries for key-value data storage
- Apply list comprehensions for efficient data processing

**Finance Application:** Store portfolio holdings as lists and dictionaries.

---

**Data structures are containers for organizing information**

# List Indexing



## Creating Lists:

```
prices = [185, 190, 188, 195]
```

## Accessing Elements:

prices[0] → 185 (first)

prices[-1] → 195 (last)

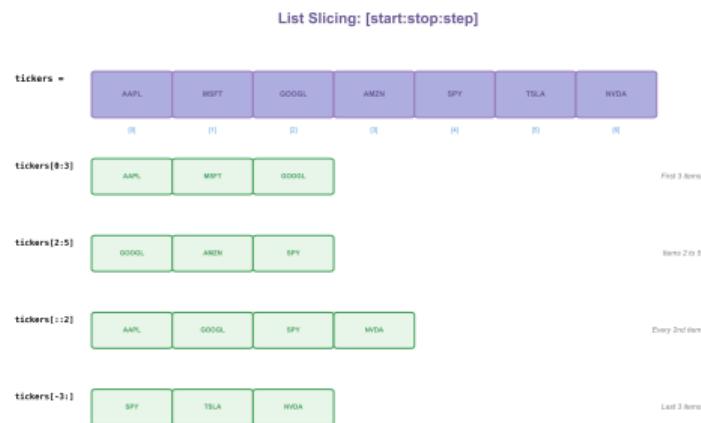
prices[1] → 190 (second)

## Remember:

Python indexing starts at 0!

Negative indices count from the end: -1 is last element

# Slicing Notation



## Slice Syntax: list[start:end:step]

`prices = [185, 190, 188, 195, 182]`

`prices[1:4] → [190, 188, 195]`

`prices[:3] → [185, 190, 188]`

`prices[2:] → [188, 195, 182]`

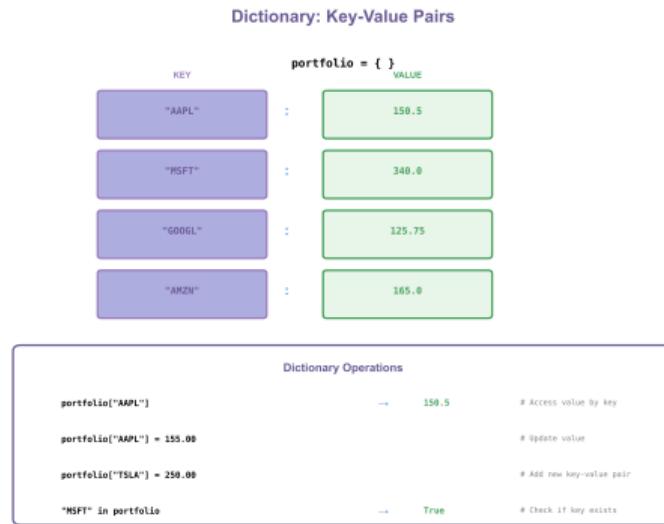
`prices[::-2] → [185, 188, 182]`

**Key:** End index is exclusive

---

Slicing creates a new list – original unchanged

# Dictionary Structure



## Key-Value Pairs:

```
portfolio = {  
    "AAPL": 50,  
    "MSFT": 30,  
    "GOOGL": 20  
}
```

## Access by Key:

```
portfolio["AAPL"] → 50
```

```
portfolio.keys()
```

```
portfolio.values()
```

Dictionaries provide O(1) lookup – very fast access

## Nested Data Structures

```
portfolio = {
```

"AAPL" :

```
{"price": 150.5, "shares": 100, "value": 15050.0}
```

"MSFT" :

```
{"price": 340.0, "shares": 50, "value": 17000.0}
```

"GOOGL" :

```
{"price": 125.75, "shares": 75, "value": 9431.25}
```

```
}
```

### Accessing Nested Data

```
portfolio["AAPL"]           → {"price": 150.5, "shares": 100, ...} # Full nested dict

portfolio["AAPL"]["price"]   → 150.5                                # Specific value

portfolio["MSFT"]["shares"] → 50                                    # Shares for MSFT

portfolio["GOOGL"]["value"] → 9431.25                            # Total value
```

# List Methods

Common List Methods			
Method	Description	Example	Result
append()	Add item to end	prices.append(200)	[150, 165, 175]
insert()	Add item at position	prices.insert(1, 168)	[150, 168, 165]
remove()	Remove first occurrence	prices.remove(165)	[150]
pop()	Remove and return item	prices.pop()	Returns: 165
sort()	Sort list in place	prices.sort()	[150, 165, 175]
reverse()	Reverse list order	prices.reverse()	[175, 165]
count()	Count occurrences	prices.count(150)	1

## Adding Elements:

`prices.append(200)`

`prices.insert(0, 180)`

## Removing:

`prices.remove(188)`

`prices.pop()` – removes last

## Sorting:

`prices.sort()`

`prices.reverse()`

Methods modify the list in-place (except sorted())

# Portfolio as Dictionary

Portfolio Representation: Dictionary

```
portfolio = {  
    "AAPL": {"shares": 100, "buy": 145.0, "current": 159.5},  
    "MSFT": {"shares": 50, "buy": 320.0, "current": 340.0},  
    "GOOGL": {"shares": 75, "buy": 120.0, "current": 125.75}  
}
```

Portfolio Calculations

```
# Calculate total value  
total_value = 0  
  
for ticker in portfolio:  
    shares = portfolio[ticker]["shares"]  
    price = portfolio[ticker]["current_price"]  
    total_value += shares * price  
  
print(f"Total portfolio value: ${total_value:.2f}")  
# Output: Total portfolio value: $31,481.25
```

## Portfolio Dictionary:

```
prices = {  
    "AAPL": 185.50,  
    "MSFT": 378.20,  
    "GOOGL": 141.80  
}
```

## Calculate Total:

```
total = sum(prices.values())
```

## Check Existence:

```
"AAPL" in prices → True
```

Dictionaries are ideal for ticker-to-data mappings

# List Comprehension

List Comprehension: Concise List Creation

The diagram illustrates the transition from a traditional loop to a more concise list comprehension. It consists of three colored boxes: an orange box for the 'Traditional Loop', a green box for 'More Concise', and a light blue box for 'List Comprehension'. Arrows point from left to right, indicating the progression.

**Traditional Loop:**

```
prices = [150, 165, 140, 172]
doubled = []
for price in prices:
    doubled.append(price * 2)

# Result: [300, 330, 280, 344]
```

**More Concise:**

```
prices = [150, 165, 140, 172]
doubled = [price * 2
           for price in prices]

# Result: [300, 330, 280, 344]
```

**List Comprehension:**

```
prices = [150, 165, 140, 172]
doubled = [price * 2
           for price in prices]

# Result: [300, 330, 280, 344]
```

List Comprehension Examples

Basic transformation:  
[x \* 2 for x in prices]  
# Double all prices

With condition (filter):  
[x for x in prices if x > 190]  
# Only prices > 190

String manipulation:  
[t.lower() for t in tickers]  
# Lowercase all tickers

Math operations:  
[x\*\*2 for x in [1,2,3,4]]  
# Squares: [1,4,9,16]

With If else:  
[x if x > 150 else 0 for x in prices]  
# Set low prices to 0

## Traditional Loop:

```
returns = []
for p in prices:
    returns.append(p * 1.05)
```

## List Comprehension:

```
returns = [p * 1.05 for p in prices]
```

## With Condition:

```
high = [p for p in prices if p > 190]
```

Comprehensions are more Pythonic and often faster

# Choosing the Right Structure

## Choosing the Right Data Structure



Feature Comparison

List	Feature	Dictionary
Ordered	Order	Unordered
<code>prices[0]</code>	Access	<code>portfolio["AAPL"]</code>
$O(n)$ search	Speed	$O(1)$ lookup
Duplicates OK	Duplicates	Unique keys
Integer indices	Keys	Any immutable

## Hands-on Exercise (25 min)

### Build a portfolio tracker:

- ① Create a list of stock tickers:

```
tickers = ["AAPL", "MSFT", "GOOGL", "AMZN"]
```

- ② Create a dictionary with shares owned:

```
shares = {"AAPL": 50, "MSFT": 30, ...}
```

- ③ Create a dictionary with current prices

- ④ Calculate portfolio value using list comprehension:

```
values = [shares[t] * prices[t] for t in tickers]
```

- ⑤ Find total portfolio value: sum(values)

- ⑥ Filter stocks worth more than \$5000

---

Save your work – we'll add more features next lesson

## Key Takeaways:

- Lists store ordered sequences (accessed by index)
- Dictionaries store key-value pairs (accessed by key)
- Slicing extracts portions: `list[start:end]`
- List comprehensions create lists efficiently
- Choose structure based on access pattern

**Next Lesson:** Control Flow (if/else, loops)

---

Data structures + control flow = programming logic

## Lesson 03: Control Flow

Data Science with Python – BSc Course

Data Science Program

45 Minutes

## After this lesson, you will be able to:

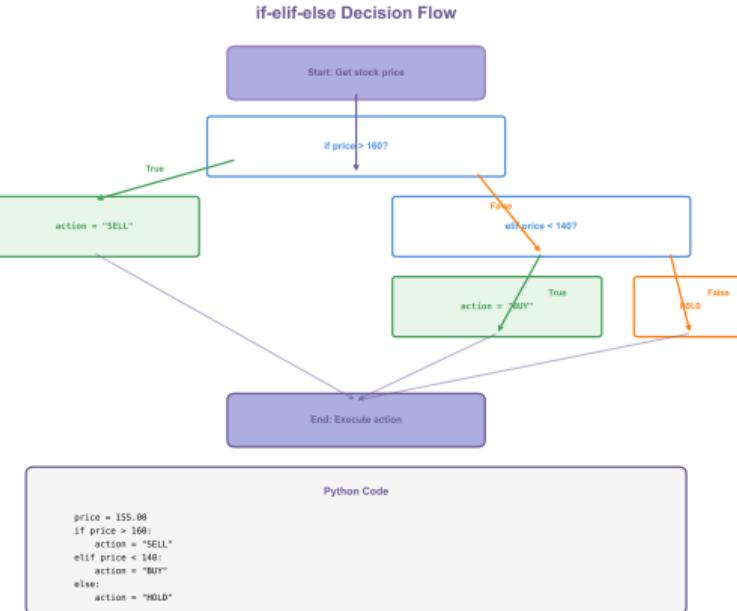
- Write conditional statements with if/elif/else
- Create loops to iterate over data
- Implement trading rules using control flow
- Use break and continue for loop control

**Finance Application:** Implement trading signals and position sizing rules.

---

**Control flow determines which code executes based on conditions**

# If-Else Statements



Indentation (4 spaces) defines code blocks in Python

## Basic Structure:

```
if price > 200:
    signal = "SELL"
elif price < 150:
    signal = "BUY"
else:
    signal = "HOLD"
```

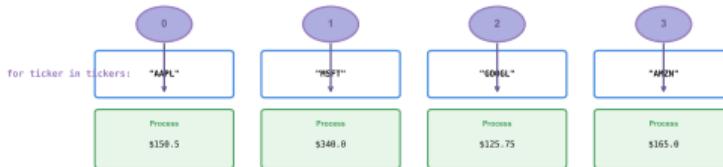
## Key Points:

- Colon after condition
- Indentation matters!
- elif is optional

# For Loops

for Loop: Iteration Over Sequence

```
tickers = ["AAPL", "MSFT", "GOOGL", "AMZN"]
```



Loop Example: Calculate Total Portfolio Value

```
portfolio = {"AAPL": 150.50, "MSFT": 340.80, "GOOGL": 125.75}
shares = {"AAPL": 100, "MSFT": 50, "GOOGL": 75}

total_value = 0
for ticker in portfolio:
    price = portfolio[ticker]
    num_shares = shares[ticker]
    value = price * num_shares
    total_value += value
    print(f"({ticker}): ${value:.2f}")

print(f"\nTotal: ${total_value:.2f}")
# Output: Total: $31,481.25
```

**Iterate Over List:**

```
for price in prices:  
    print(price)
```

**With Index:**

```
for i, p in enumerate(prices):  
    print(f"Day {i}: {p}")
```

**Range:**

```
for i in range(5):  
    print(i) # 0,1,2,3,4
```

For loops iterate a known number of times

03\_while\_loop\_diagram/chart.pdf

## Basic While:

```
balance = 10000
while balance > 5000:
    balance *= 0.95
    print(balance)
```

## Use Cases:

- Unknown iterations
- Waiting for condition
- Simulation until target

**Warning:** Infinite loops!

# Nested Loops

## Nested Loops: Loop Within a Loop

```
Outer Loop: for ticker in ["AAPL", "MSFT", "GOOGL"]  
    Inner loop: for day in range(5)  
        Process each ticker for each day  
Total Iterations: 3 tickers × 5 days = 15
```

### Nested Loop Example: Price Matrix

```
tickers = ["AAPL", "MSFT", "GOOGL"]  
days = ["Mon", "Tue", "Wed", "Thu", "Fri"]  
  
for ticker in tickers:          # Outer loop (3 iterations)  
    print(f"\n{ticker} prices:")  
    for day in days:           # Inner loop (5 iterations)  
        price = get_price(ticker, day)  # Called 15 times total  
        print(f" {day}: ${price:.2f}")  
  
# Output:  
# AAPL prices:  
#   Mon: $150.50  
#   Tue: $151.25  
#   ...
```

# Break and Continue

## break vs continue: Loop Control

```
break: Exit Loop Immediately
prices = [150, 165, 148, 175, 162]

for price in prices:
    if price < 150:
        print("Stop! Low: ${price}")
        break # Exit loop
    print("OK: ${price}")

# Output:
# OK: $150
# Stop! Low: $148
# Stop! Low: $162
# (loop ends, 175 and 165 not processed)
```

```
continue: Skip to Next Iteration
prices = [150, 165, 148, 175, 162]

for price in prices:
    if price < 150:
        print("Skip: ${price}")
        continue # Skips rest
    print("Process: ${price}")

# Output:
# Process: $150
# Process: $165
# Skip: $148
# Process: $175
# Process: $162
```

**Break:** Exit loop entirely

for price in prices:

if price > 200:

    break # stop now

**Continue:** Skip to next iteration

for price in prices:

if price < 0:

    continue # skip this

process(price)

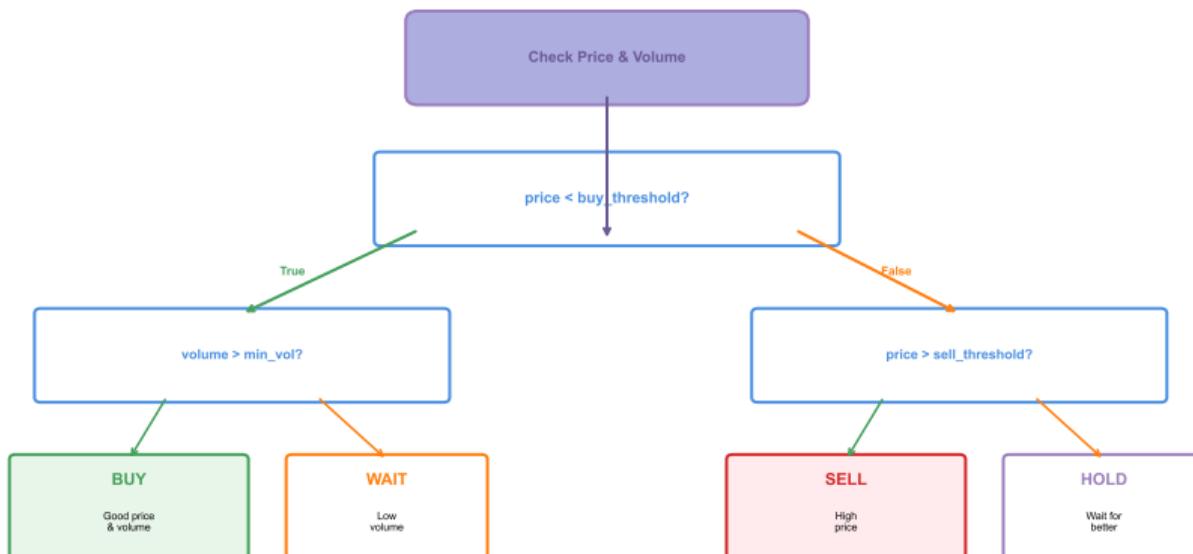
### Key Differences

<b>break</b>	Exits loop completely	Used when condition met
<b>continue</b>	Skips current iteration	Used to filter items
<b>break</b>	No more iterations	Loop terminates
<b>continue</b>	Continues with next	Loop continues

**Break stops loop; continue skips current iteration**

# Trading Rules Decision Tree

## Trading Rules: Decision Tree



### Python Implementation

```
price, volume = 145.00, 1200000
buy_threshold, sell_threshold = 150.00, 170.00
min_volume = 1000000
```

# Loop Comparison

## for vs while: Loop Comparison

for Loop	Aspect	while Loop	
Iterate over sequence	Use Case	Repeat until condition	
Known iterations	Duration	Unknown iterations	
for x in sequence:	Syntax	while condition:	
Automatic	Increment	Manual	
List, range(), dict	Common With	Counters, flags	
More readable	Readability	More flexible	
Portfolio analysis	Finance Example	Price convergence	
for Example		while Example	
<pre>prices = [150, 165, 148] total = 0 for price in prices:     total += price avg = total / len(prices)</pre>		<pre>price = 100 target = 150 while price &lt; target:     price *= 1.05     years += 1</pre>	

# Control Flow Patterns

## Common Control Flow Patterns

### Pattern 1: Guard Clause

```
def buy_stock(price, balance):
    # Guard: Check preconditions
    if price <= 0:
        return "Invalid price"
    if balance < price:
        return "Insufficient funds"

    # Main logic
    execute_buy(price)
```

### Pattern 2: Accumulator

```
prices = [150, 165, 148, 172]
total = 0 # Accumulator

for price in prices:
    total += price

average = total / len(prices)
print(f"Avg: ${average:.2f}")
```

### Pattern 3: Search & Break

```
prices = [150, 165, 148, 172]
found = False

for price in prices:
    if price < 150:
        print(f"Found: ${price}")
        found = True
        break # Stop searching
```

### Pattern 4: Filter Pattern

```
all_prices = [150, 165, 148, 172]
high_prices = [] # Filtered list

for price in all_prices:
    if price > 160:
        high_prices.append(price)

# Result: [165, 172]
```

### Pattern 5: Counter

```
prices = [150, 165, 148, 172, 145]
count_low = 0 # Counter

for price in prices:
    if price < 150:
        count_low += 1
```

### Pattern 6: Find Min/Max

```
prices = [150, 165, 148, 172]
max_price = prices[0] # Initialize

for price in prices:
    if price > max_price:
        max_price = price
```

## Hands-on Exercise (25 min)

### Implement a simple trading system:

- ① Create price list: `prices = [180, 185, 195, 188, 205, 198]`
- ② Implement trading rules:
  - If price  $<$  200: SELL
  - If price  $>$  185: BUY
  - Else: HOLD
- ③ Loop through prices and generate signals
- ④ Count total BUY, SELL, HOLD signals
- ⑤ Find first price that triggers SELL (use break)
- ⑥ Skip negative prices if any (use continue)

---

This forms the basis of algorithmic trading

# Lesson Summary

## Key Takeaways:

- if/elif/else for conditional execution
- for loops iterate over sequences
- while loops continue until condition is false
- break exits loop, continue skips iteration
- Indentation defines code blocks

## Next Lesson: Functions

---

**Control flow + functions = modular trading systems**

## Lesson 04: Functions

Data Science with Python – BSc Course

Data Science Program

45 Minutes

## After this lesson, you will be able to:

- Define and call functions with parameters
- Use return statements to output values
- Understand variable scope (local vs global)
- Write docstrings for documentation

**Finance Application:** Create reusable functions for return calculations and risk metrics.

---

Functions are building blocks of modular code

# Function Anatomy

## Function Anatomy

```
def calculate_return(price_old, price_new):
    """Calculate percentage return."""
    return (price_new - price_old) / price_old * 100
```

The diagram shows a Python function definition within a blue-bordered box. Annotations with arrows point to specific parts of the code:

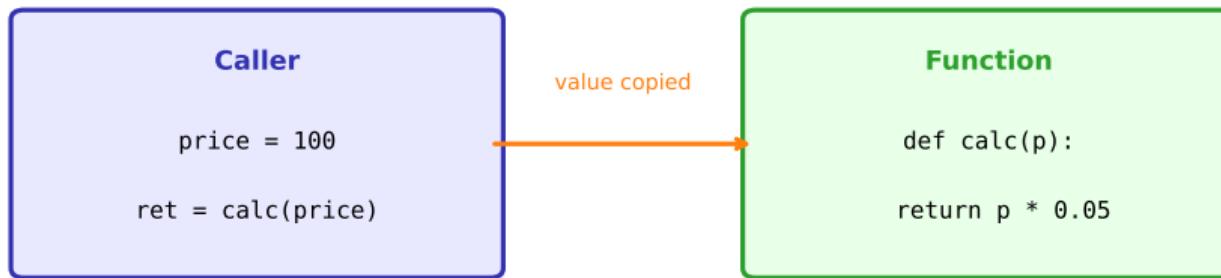
- An orange arrow labeled "keyword" points to the `def` keyword.
- A blue arrow labeled "parameters" points to the parameters `price_old` and `price_new`.
- A green arrow labeled "return value" points to the `return` statement.
- A grey arrow labeled "docstring" points to the multi-line string starting with `"""`.

*Functions encapsulate reusable logic*

---

**def keyword, name, parameters, colon, indented body, return**

## Parameter Passing



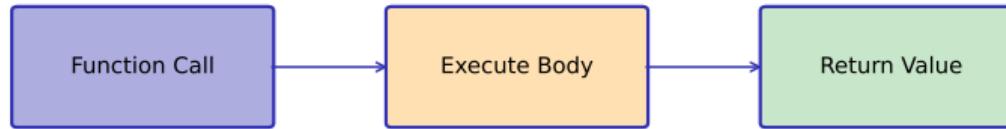
Positional: func(a, b)

Keyword: func(x=1, y=2)

Default: def f(x=10)

\*args, \*\*kwargs

## Return Value Flow



### Single return:

```
return price * 1.05
```

### Multiple returns:

```
return mean, std
```

### No return (None):

```
print("Hello")
```

### Early return:

```
if x < 0: return 0
```

Functions without return statement return None

## Variable Scope: Local vs Global

### Global Scope

```
tax_rate = 0.15
```

### Local Scope (inside function)

```
def calc_tax(income):  
    tax = income * tax_rate
```

*Local variables exist only during function execution*

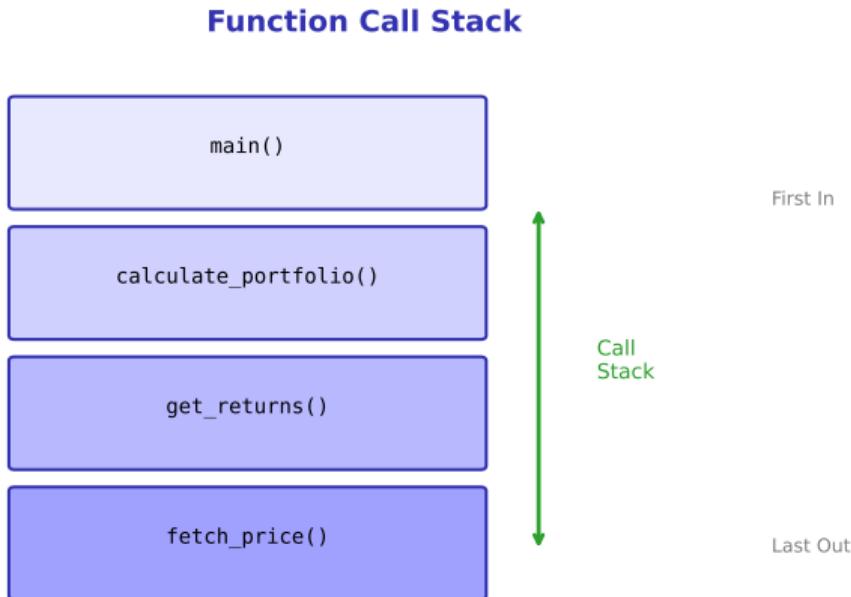
## Docstring Best Practices

```
def calculate_sharpe(returns, rf_rate=0.02):
    """
    Calculate the Sharpe ratio for a series of returns.

    Parameters:
        returns (array): Daily return values
        rf_rate (float): Risk-free rate (default: 0.02)

    Returns:
        float: Annualized Sharpe ratio
    """
    excess = returns.mean() - rf_rate/252
    return excess / returns.std() * np.sqrt(252)
```

# Function Call Stack



Stack grows with nested calls, shrinks as functions return

## Pure vs Impure Functions

### Pure Function

Same input -> Same output

No side effects

```
def add(a, b):  
    return a + b
```

### Impure Function

Modifies external state

May have side effects

```
def update(lst, x):  
    lst.append(x)
```

*Prefer pure functions for predictable, testable code*

## Essential Finance Functions

```
calculate_return(p1, p2)
```

Price change %

```
annualize_return(daily_ret)
```

Convert to yearly

```
calculate_volatility(returns)
```

Standard deviation

```
sharpe_ratio(ret, rf)
```

Risk-adjusted return

```
max_drawdown(prices)
```

Largest peak-to-trough

```
beta(stock, market)
```

Market sensitivity

## Hands-on Exercise (25 min)

### Build a finance functions library:

- ① `calculate_return(buy, sell)` – percentage return
- ② `annualize_return(daily_ret, days=252)` – annualized
- ③ `calculate_volatility(returns)` – standard deviation
- ④ `sharpe_ratio(returns, rf=0.02)` – risk-adjusted return
- ⑤ Test each function with sample data
- ⑥ Add docstrings to all functions

---

These functions will be used throughout the course

## Key Takeaways:

- Functions encapsulate reusable logic
- Parameters pass data in, return sends data out
- Local scope: variables exist only inside function
- Docstrings document function purpose and usage
- Pure functions are predictable and testable

**Next Lesson:** DataFrames Introduction

---

Functions + pandas = powerful financial analysis

## Lesson 05: DataFrames Introduction

Data Science with Python – BSc Course

Data Science Program

45 Minutes

## After this lesson, you will be able to:

- Import pandas and create DataFrames
- Load data from CSV files
- Explore data with head(), tail(), info(), describe()
- Understand DataFrame structure (index, columns, values)

**Finance Application:** Load and explore stock price data.

---

pandas is THE library for data manipulation in Python

**DataFrame Structure**

Index	Date	AAPL	MSFT	GOOGL
0	2024-01-02	185.2	376.1	140.9
1	2024-01-03	184.8	374.2	139.5
2	2024-01-04	186.1	378.5	141.2

Rows (observations)  Columns (features) 

2D labeled data structure with rows and columns

## Series vs DataFrame

### Series (1D)

Single column

0	185.2
1	184.8
2	186.1

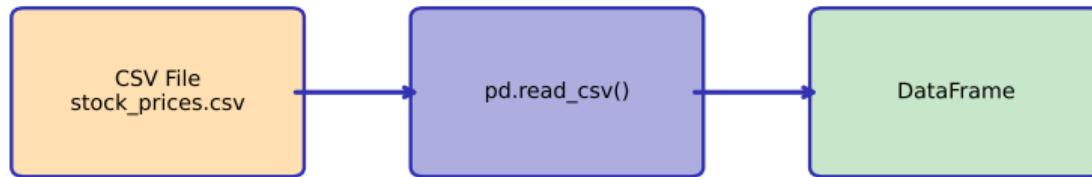
### DataFrame (2D)

Multiple columns

	AAPL	MSFT	VOL
0	185.2	376.1	1.2M
1	184.8	374.2	1.1M
2	186.1	378.5	1.3M

*DataFrame = Collection of Series sharing an index*

## Loading CSV Data



### Common Parameters:

```
filepath: "data/prices.csv"  
index_col: "Date"  
parse_dates: True  
usecols: ["AAPL", "MSFT"]
```

## Viewing Data: head() and tail()

`df.head(3)`

*First 3 rows*

```
2024-01-02  185.2  
2024-01-03  184.8  
2024-01-04  186.1
```

`df.tail(3)`

*Last 3 rows*

```
2024-12-27  195.8  
2024-12-30  196.2  
2024-12-31  197.1
```

Default: 5 rows | Customize: `head(10)`, `tail(20)`

## DataFrame Info: df.info()

```
<class pandas.DataFrame>

RangeIndex: 252 entries, 0 to 251

Data columns (5 columns):

 Date      252 non-null datetime64
 AAPL      252 non-null float64
 MSFT      252 non-null float64
 GOOGL     250 non-null float64 (2 missing)

memory usage: 10.0 KB
```

## Summary Statistics: df.describe()

Stat	AAPL	MSFT
count	252	252
mean	189.5	385.2
std	8.2	12.5
min	175.1	355.8
25%	183.4	375.6
50%	188.9	384.1
75%	195.2	394.8
max	210.3	420.5

## Index and Columns

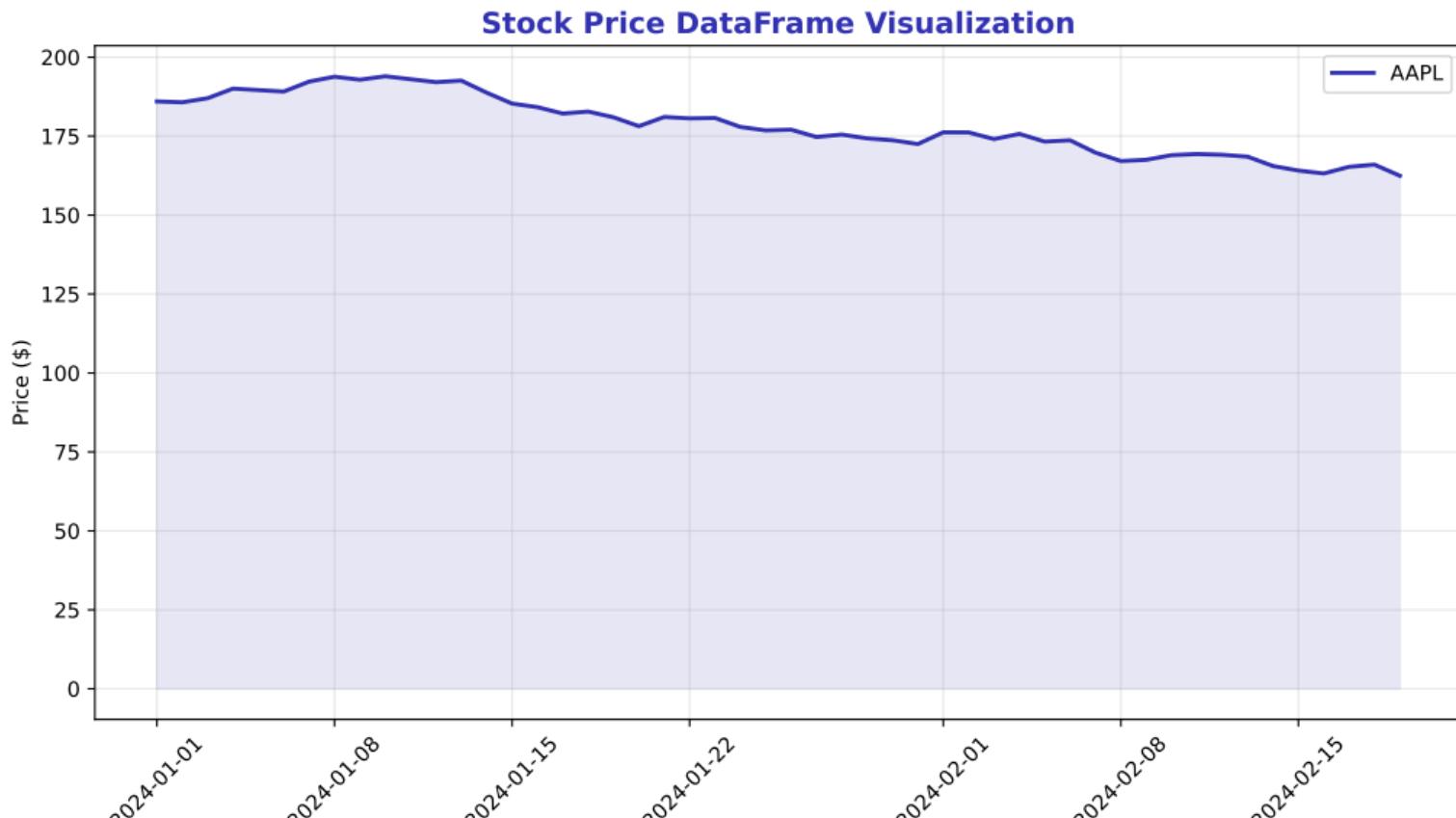
```
df.columns: ["Date", "AAPL", "MSFT", "GOOGL"]
```

df.index

Data

```
df.shape: (252, 4) | df.dtypes: column data types
```

## Stock Data Example



## Hands-on Exercise (25 min)

### Explore stock price data:

- ① Load the stock data:

```
df = pd.read_csv("../datasets/stock_prices.csv")
```

- ② View first 10 rows: df.head(10)

- ③ Check data types: df.info()

- ④ Get statistics: df.describe()

- ⑤ Access column names: df.columns

- ⑥ Check shape: df.shape

- ⑦ Find which stock has highest mean price

---

Exploration before analysis prevents costly mistakes

# Lesson Summary

## Key Takeaways:

- pandas DataFrame is the core data structure
- pd.read\_csv() loads CSV files easily
- head()/tail() show first/last rows
- info() shows data types and missing values
- describe() provides statistical summary

**Next Lesson:** Selection and Filtering

---

Loading data is step 1 – now we'll learn to slice it

## Lesson 06: Selection and Filtering

Data Science with Python – BSc Course

Data Science Program

45 Minutes

## After this lesson, you will be able to:

- Select columns using bracket and dot notation
- Access rows with iloc (position) and loc (label)
- Filter data using boolean conditions
- Combine multiple conditions with & and —

**Finance Application:** Screen stocks by price, volume, and other criteria.

---

**Selection and filtering extract relevant data for analysis**

## Column Selection Methods

```
df['AAPL']
```

Single column (Series)

```
df[['AAPL', 'MSFT']]
```

Multiple columns (DataFrame)

```
df.AAPL
```

Attribute access (simple names)

```
df.loc[:, 'AAPL':'GOOGL']
```

Range of columns

## iloc vs loc

### iloc (Integer Location)

Position-based indexing

```
df.iloc[0]
```

```
df.iloc[0:5, 1:3]
```

Uses: 0, 1, 2, ...

### loc (Label Location)

Label-based indexing

```
df.loc['2024-01-02']
```

```
df.loc[:, 'AAPL']
```

Uses: dates, names

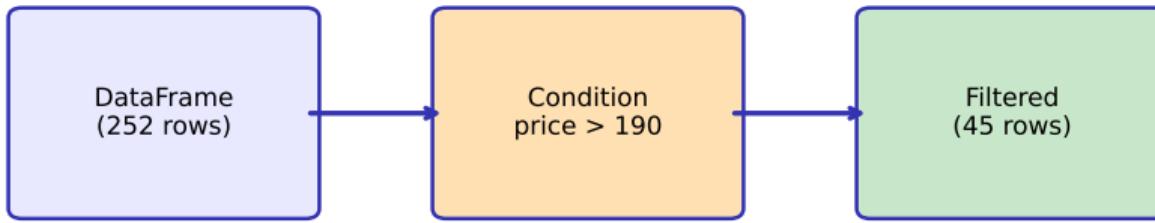
*iloc: exclusive end | loc: inclusive end*

## Boolean Masking

AAPL	df["AAPL"] > 188	Mask	Result
185		False	190
190		True	195
188	→	False	→
195		True	
182		False	

*Boolean mask filters rows where condition is True*

## Conditional Filtering Flow



```
df_filtered = df[df["AAPL"] > 190]
```

## Multiple Conditions

### AND: &

```
(df["AAPL"] > 185) &  
(df["MSFT"] > 380)
```

### OR: |

```
(df["AAPL"] > 200) |  
(df["MSFT"] > 400)
```

### NOT: ~

```
~(df["AAPL"] > 190)
```

*Always use parentheses around each condition!*

## Chained Filtering with query()

### Traditional

```
df[(df["AAPL"] > 185) &  
    (df["Volume"] > 1e6)]
```

### query() Method

```
df.query("AAPL > 185 and  
         Volume > 1e6")
```

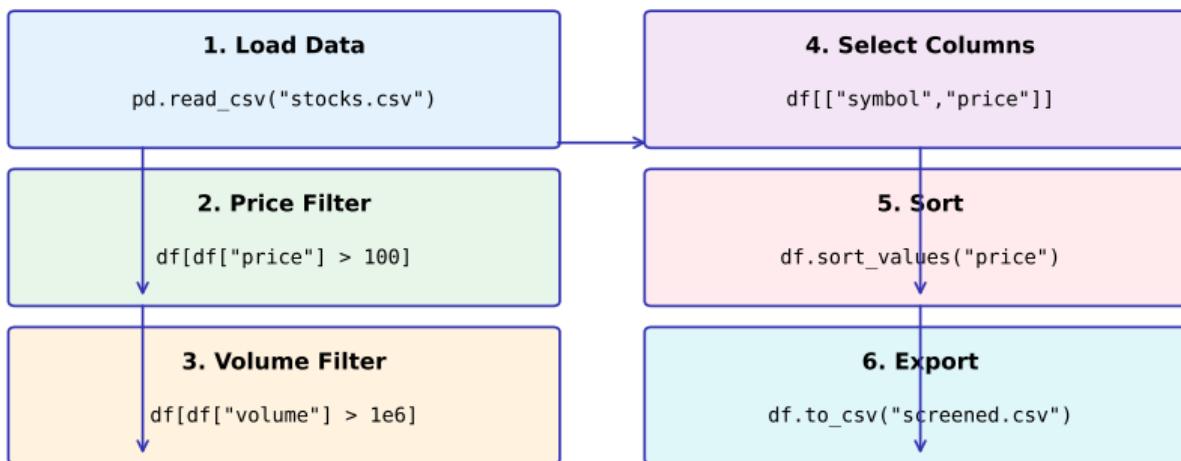
*query() is more readable for complex filters*

### Membership: isin()

```
df[df["Symbol"].isin(["AAPL", "MSFT", "GOOGL"])]
```

## Selection Methods Comparison

Method	Use Case	Returns
<code>df["col"]</code>	Single column	Series
<code>df[["col1", "col2"]]</code>	Multiple columns	DataFrame
<code>df.iloc[0]</code>	Row by position	Series
<code>df.loc["date"]</code>	Row by label	Series
<code>df[df.col &gt; x]</code>	Filter rows	DataFrame



*Combine filters to build powerful stock screeners*

## Hands-on Exercise (25 min)

### Build a stock screener:

- ① Load stock data from CSV
- ② Select only AAPL and MSFT columns
- ③ Filter rows where AAPL > 185
- ④ Filter rows where AAPL > 185 AND MSFT > 375
- ⑤ Use query() for the same filter
- ⑥ Select first 10 trading days using iloc
- ⑦ Sort by AAPL price descending

---

Stock screeners are fundamental tools in finance

# Lesson Summary

## Key Takeaways:

- `df["col"]` selects single column as Series
- `iloc` uses integer positions; `loc` uses labels
- Boolean conditions create True/False masks
- Combine conditions with `&` (and) and `—` (or)
- `query()` is cleaner for complex filters

**Next Lesson:** Missing Data and Cleaning

---

**Week 1 complete! You can now load, explore, and filter data**

## Lesson 07: Missing Data and Cleaning

Data Science with Python – BSc Course

45 Minutes

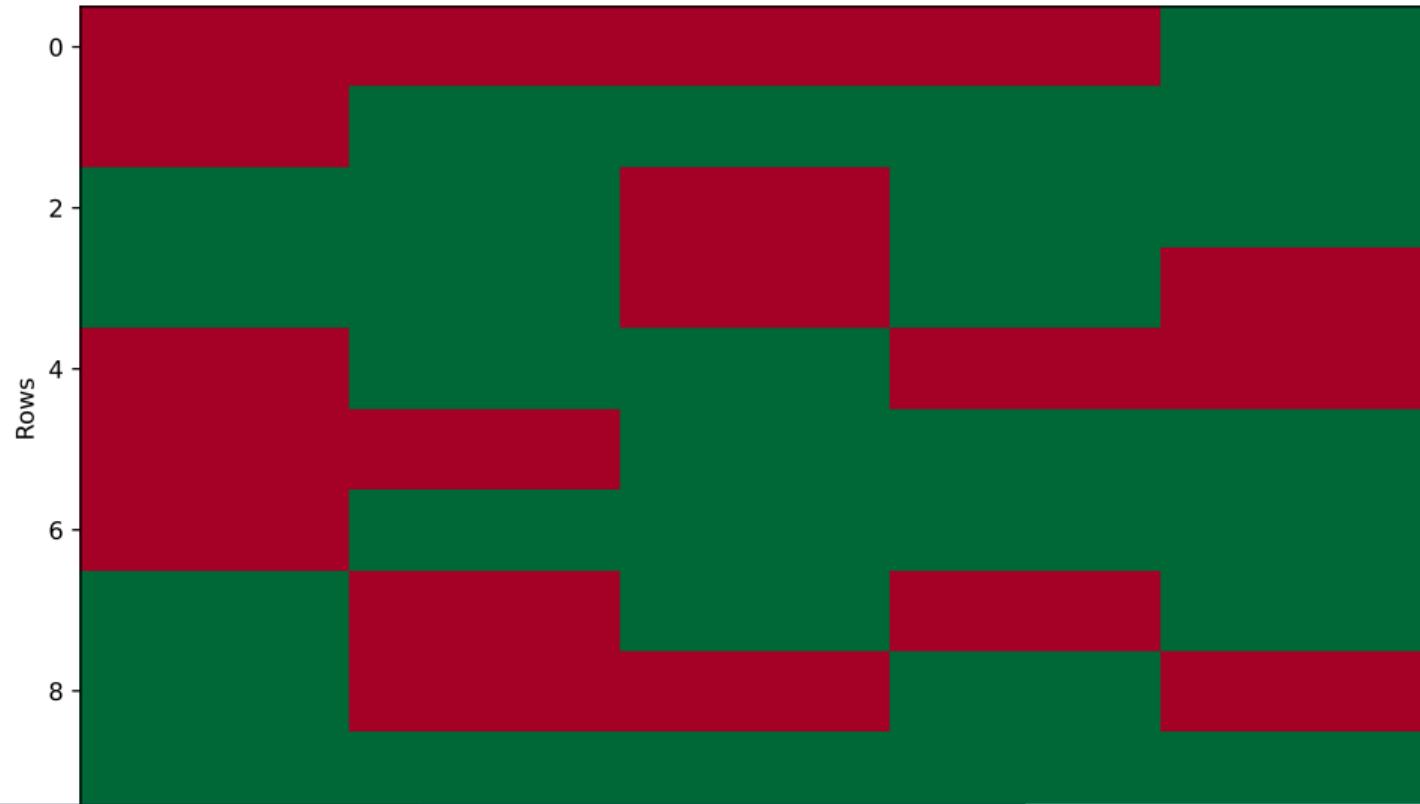
# Learning Objectives

**After this lesson, you will be able to:**

- isna()/isnull() for detection
- fillna() methods (ffill, bfill, mean)
- dropna() to remove missing
- Handling duplicates
- Data type conversion

**Finance application: Stock data processing and analysis**

**Missing Data Pattern (Red = Missing)**



## fillna() Methods Comparison

`fillna(0)`

Fill with constant value

`fillna(method="ffill")`

Forward fill (last valid)

`fillna(method="bfill")`

Backward fill (next valid)

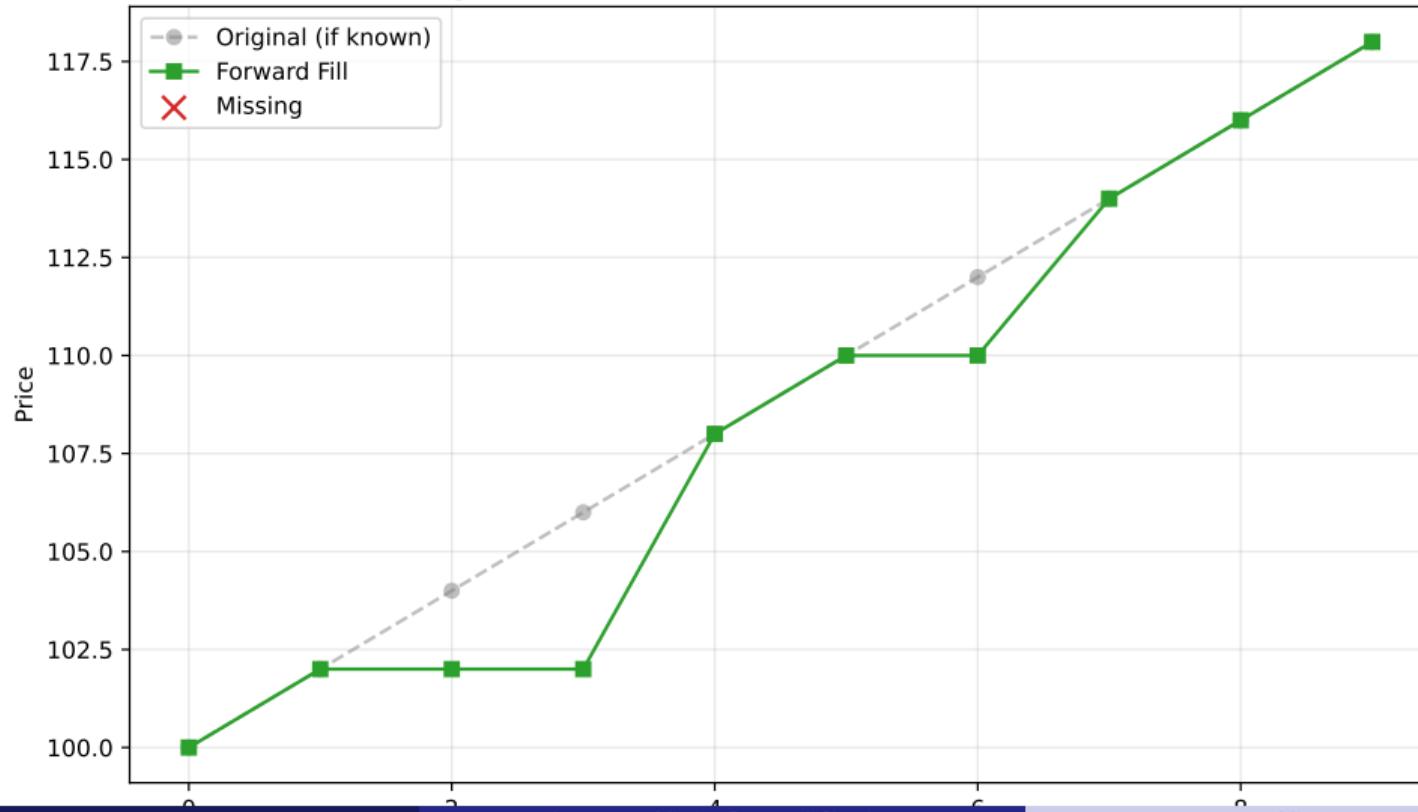
`fillna(df.mean())`

Fill with column mean

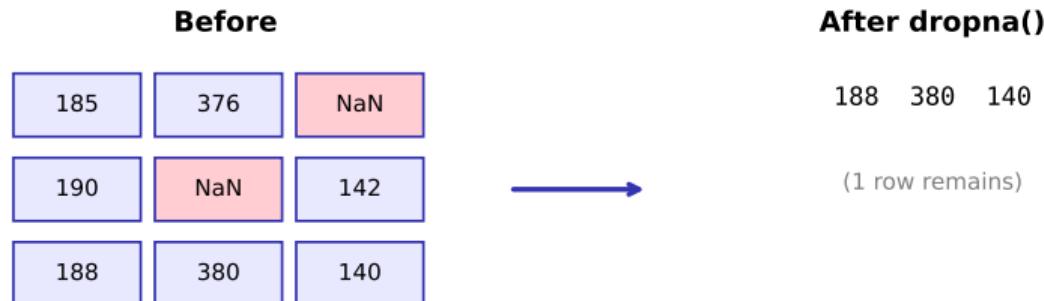
### Data Quality Checklist

- [ ] Check for missing values (isna())
- [ ] Identify duplicates (duplicated())
- [ ] Verify data types (dtypes)
- [ ] Check value ranges (describe())
- [ ] Validate dates (date parsing)
- [ ] Look for outliers

### Imputation: Forward Fill for Stock Prices



## dropna() Behavior



### Detecting Duplicates

```
df.duplicated()
```

Returns boolean mask

```
df.drop_duplicates()
```

Removes duplicate rows

```
df.drop_duplicates(subset=['Date'])
```

Check specific columns only

### Data Cleaning Workflow

1. Load raw data

2. Check info() and describe()

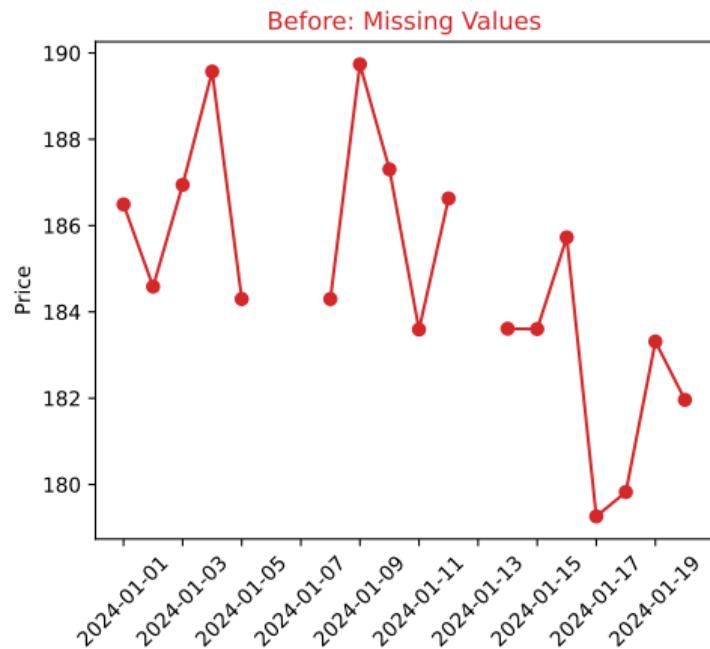
3. Handle missing values

4. Remove duplicates

5. Fix data types

6. Validate ranges

### Data Cleaning: Before vs After



Key concept for financial data analysis

## Lesson Summary

### Key Takeaways:

- `isna()`/`isnull()` for detection
- `fillna()` methods (`ffill`, `bfill`, `mean`)
- `dropna()` to remove missing
- Handling duplicates
- Data type conversion

**Practice:** Apply these concepts to the stock price dataset.

## Lesson 08: Basic Operations

### Data Science with Python – BSc Course

45 Minutes

## After this lesson, you will be able to:

- Creating new columns
- `apply()` for transformations
- Arithmetic operations
- Sorting with `sort_values()`
  
- Calculating returns and moving averages

**Finance application: Stock data processing and analysis**

## Creating New Columns

```
df["Return"] = df["Close"].pct_change()
```

Calculate returns

```
df["MA20"] = df["Close"].rolling(20).mean()
```

Moving average

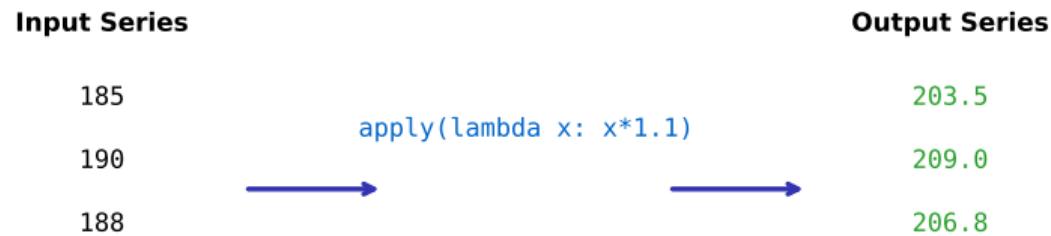
```
df["High_Low"] = df["High"] - df["Low"]
```

Price range

```
df["Signal"] = np.where(df["Return"]>0, 1, -1)
```

Conditional

## apply() Function



## DataFrame Arithmetic

```
df["A"] + df["B"]
```

Element-wise addition

```
df["A"] * 100
```

Scalar multiplication

```
df["A"] / df["B"]
```

Division

```
df.sum()
```

Column sums

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

Row means

### Sorting DataFrames

```
df.sort_values("Price")
```

Sort by single column (ascending)

```
df.sort_values("Price", ascending=False)
```

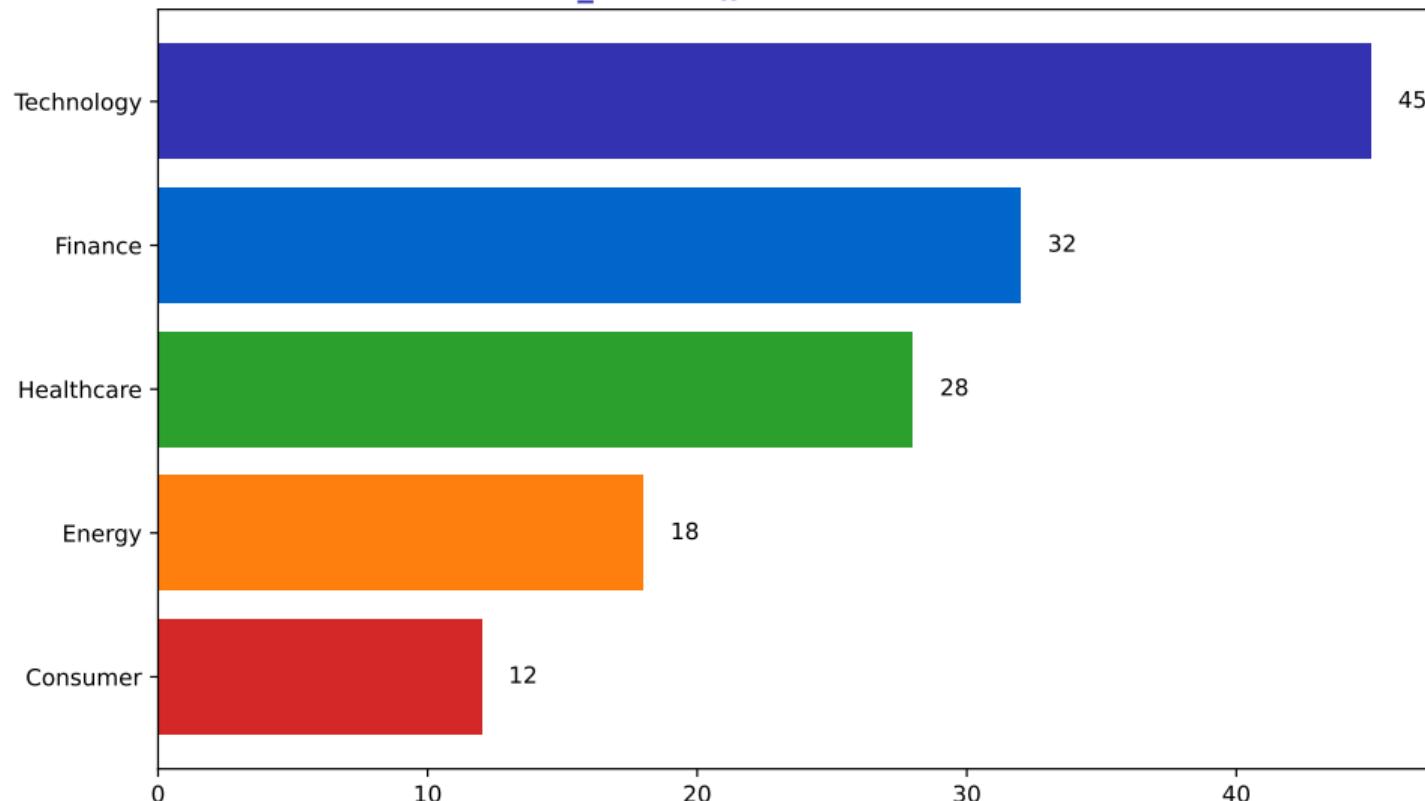
Sort descending

```
df.sort_values(["Sector", "Price"])
```

Sort by multiple columns

## 05 Value Counts

`value_counts(): Sector Distribution`



## Calculating Returns

Simple Return:  $r_t = \frac{P_t - P_{t-1}}{P_{t-1}}$

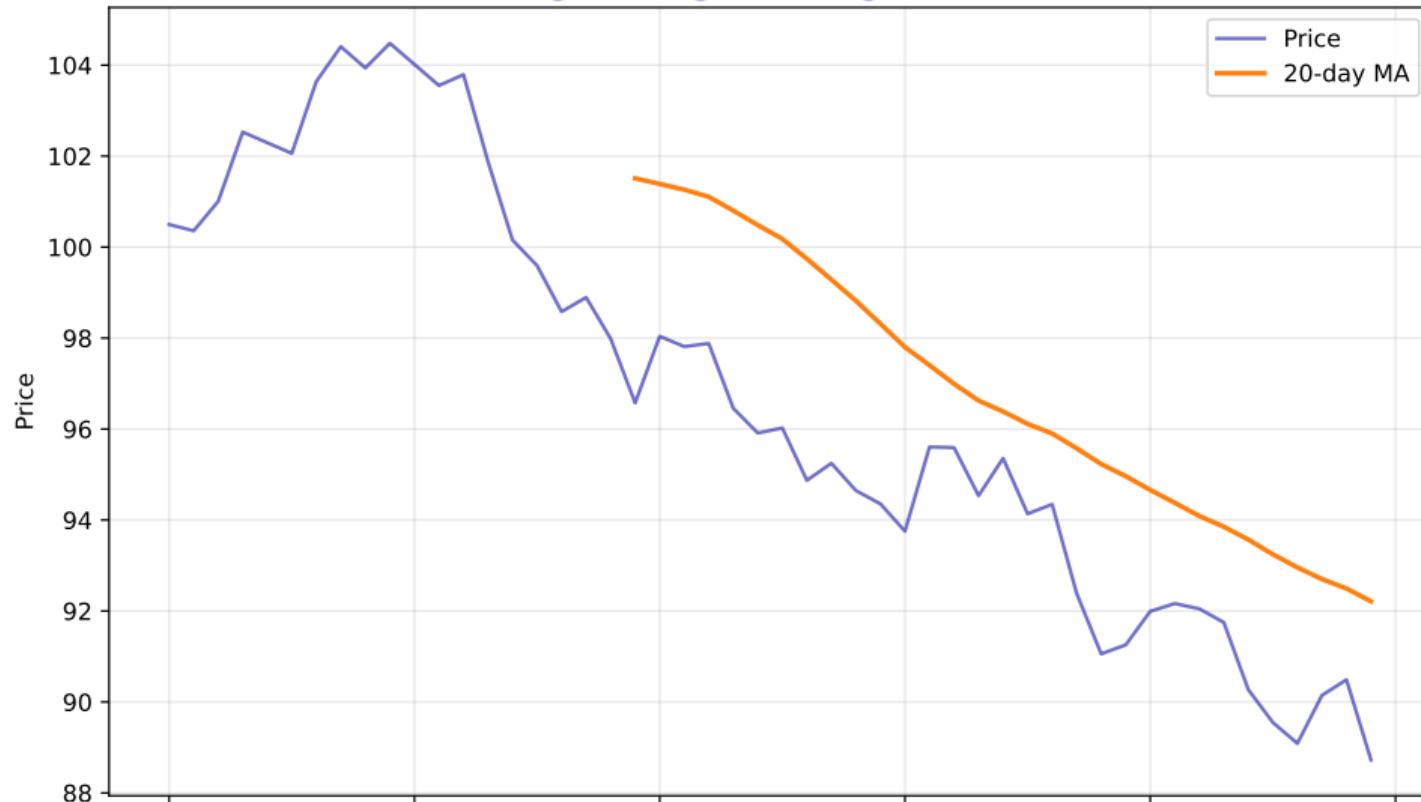
```
df["Return"] = df["Price"].pct_change()
```

Log Return:  $r_t = \ln(P_t) - \ln(P_{t-1})$

```
df["LogRet"] = np.log(df["Price"]).diff()
```

## 07 Moving Average

Moving Average: `rolling(20).mean()`



## Operations Cheat Sheet

`pct_change()` - Returns

`diff()` - Differences

`cumsum()` - Cumulative sum

`cumprod()` - Cumulative product

`rolling(n)` - Rolling window

`shift(n)` - Lag values

`rank()` - Rankings

`clip(lower, upper)` - Bound values

### Key Takeaways:

- Creating new columns
- `apply()` for transformations
- Arithmetic operations
- Sorting with `sort_values()`
  
- Calculating returns and moving averages

**Practice:** Apply these concepts to the stock price dataset.

## Lesson 09: GroupBy Operations

Data Science with Python – BSc Course

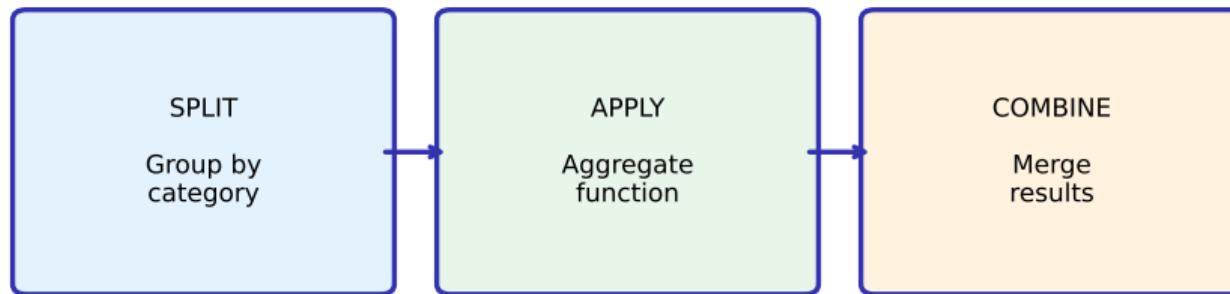
45 Minutes

## After this lesson, you will be able to:

- Split-apply-combine paradigm
- groupby() basics
- Aggregation functions
- transform() vs agg()
- Multi-column grouping

Finance application: Stock data processing and analysis

## Split-Apply-Combine Paradigm



## GroupBy Workflow

```
df.groupby("Sector")["Return"].mean()
```



Group

Select

Aggregate

### Aggregation Functions

**mean()**

Average value

**sum()**

Total sum

**count()**

Number of values

**std()**

Standard deviation

**min()/max()**

Extremes

**first()/last()**

First/last value

### agg() vs transform()

#### agg()

Returns ONE value  
per group

Result: smaller

#### transform()

Returns SAME shape  
as input

Result: same size

### Multi-Column GroupBy

```
df.groupby(["Sector", "Year"])["Return"].mean()
```

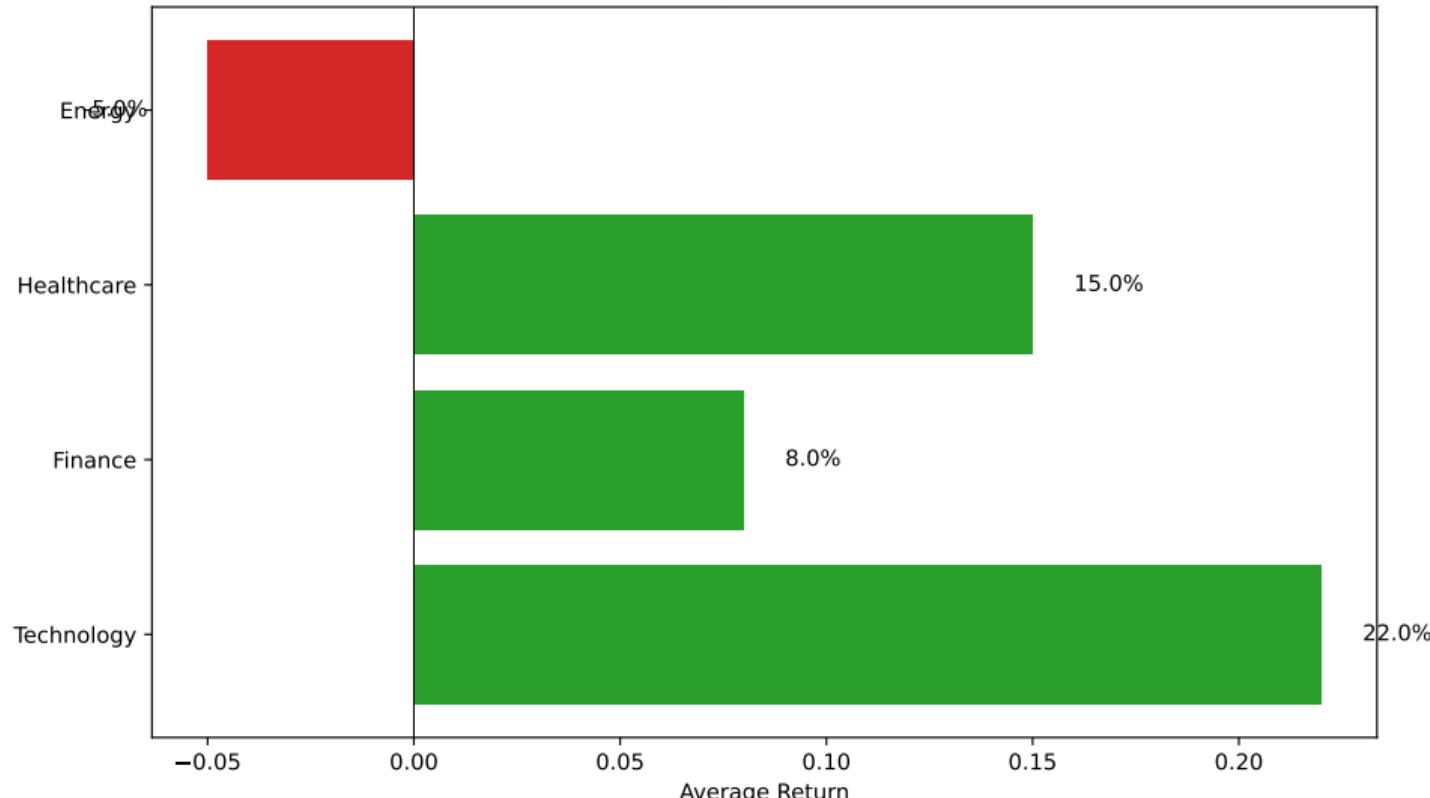
Creates hierarchical grouping:

Technology, 2023 -> 0.15

Technology, 2024 -> 0.22

Finance, 2023 -> 0.08

Sector Returns: `groupby("Sector")["Return"].mean()`



### Common GroupBy Patterns

```
df.groupby("X")["Y"].agg(["mean", "std"])
```

```
df.groupby("X").agg({"A": "sum", "B": "mean"})
```

```
df.groupby("X")["Y"].transform("mean")
```

```
df.groupby("X").apply(custom_function)
```

### GroupBy in Finance

#### Sector returns

```
groupby("Sector")["Return"].mean()
```

#### Monthly aggregation

```
groupby(df.index.month).sum()
```

#### Portfolio weights

```
groupby("Asset")["Value"].transform(lambda x: x/x.sum())
```

#### Risk by category

```
groupby("Rating")["Volatility"].mean()
```

## Lesson Summary

### Key Takeaways:

- Split-apply-combine paradigm
- groupby() basics
- Aggregation functions
- transform() vs agg()
- Multi-column grouping

**Practice:** Apply these concepts to the stock price dataset.

## Lesson 10: Merging and Joining

Data Science with Python – BSc Course

45 Minutes

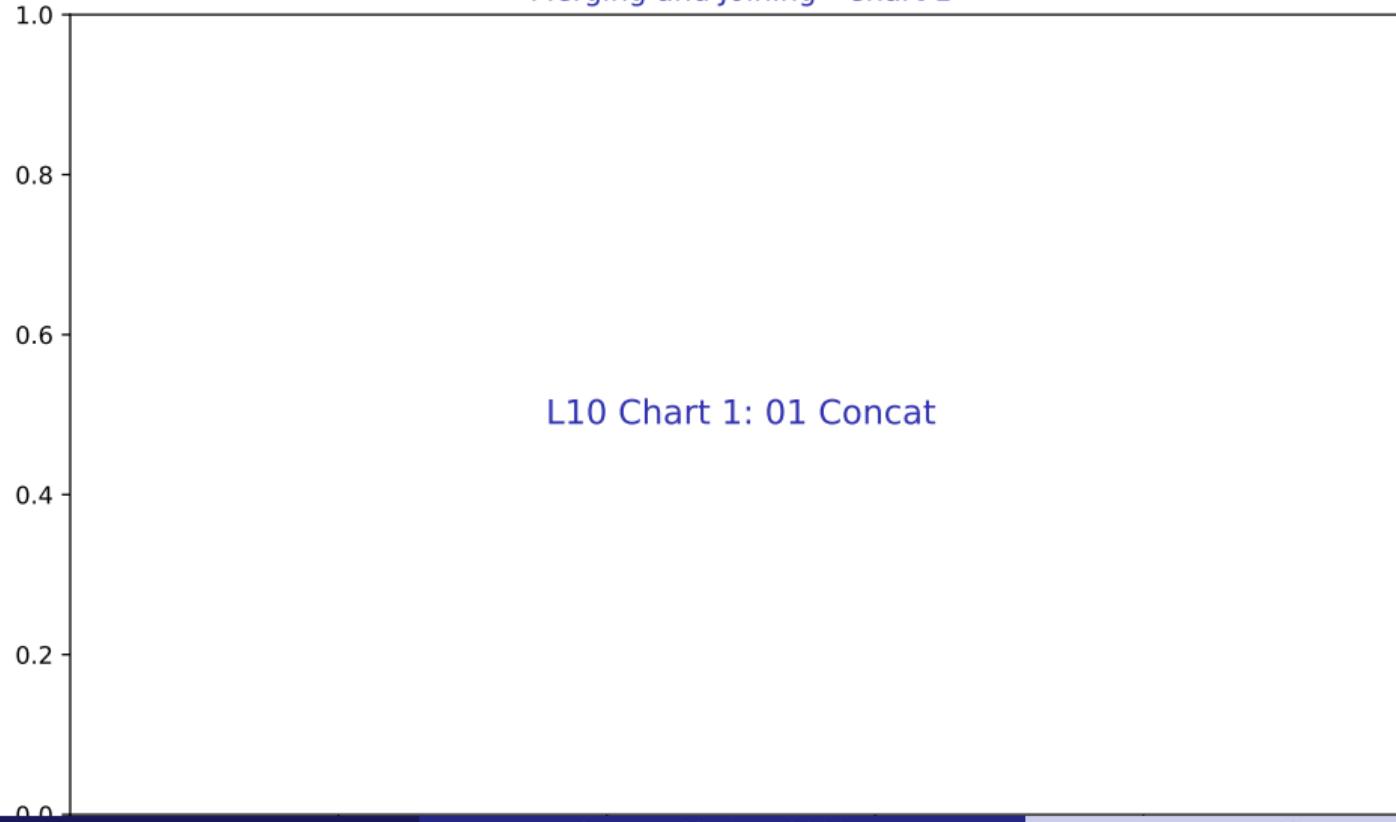
# Learning Objectives

**After this lesson, you will be able to:**

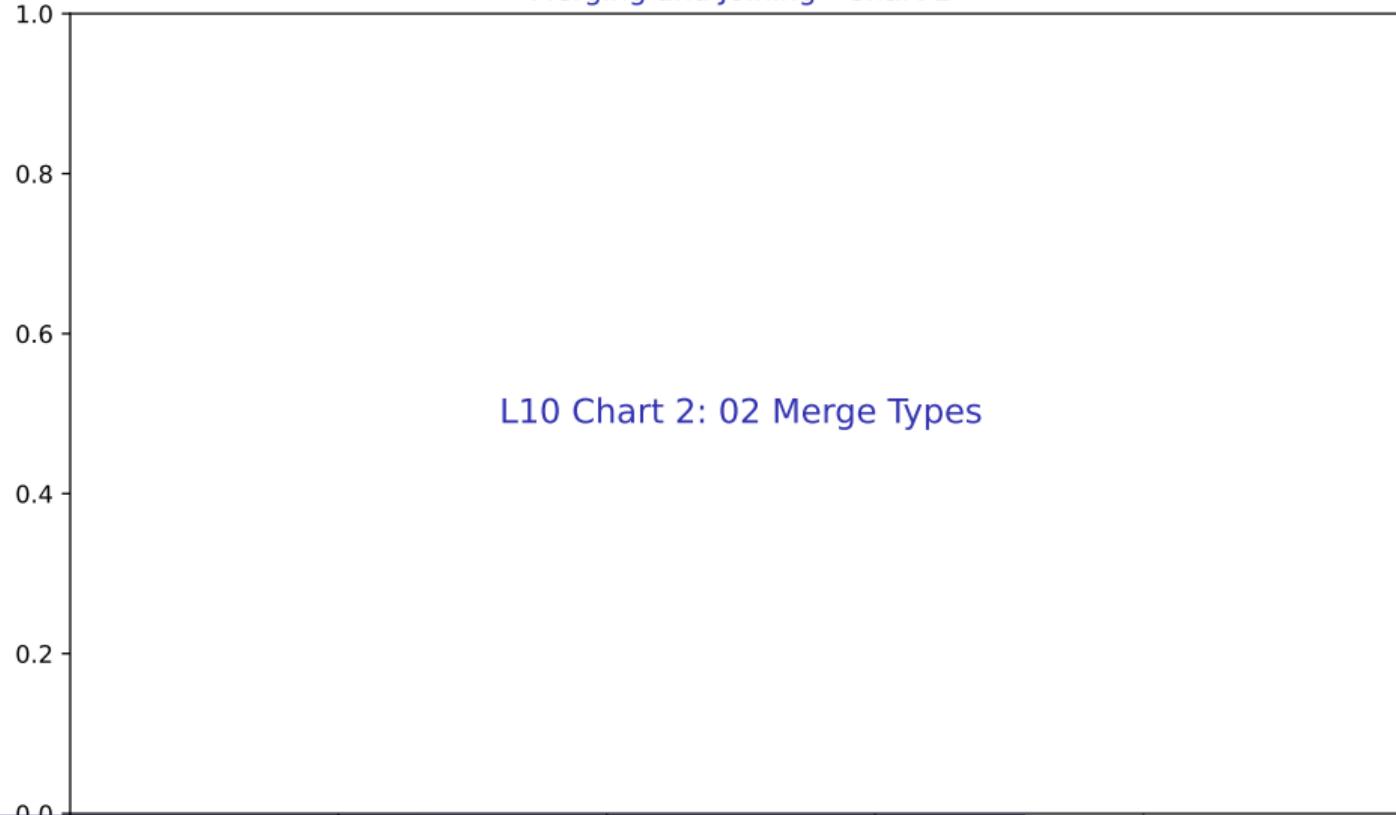
- pd.concat() for stacking
- pd.merge() for SQL-style joins
- Join types: inner, outer, left, right
- Handling key columns

**Finance application: Stock data processing and analysis**

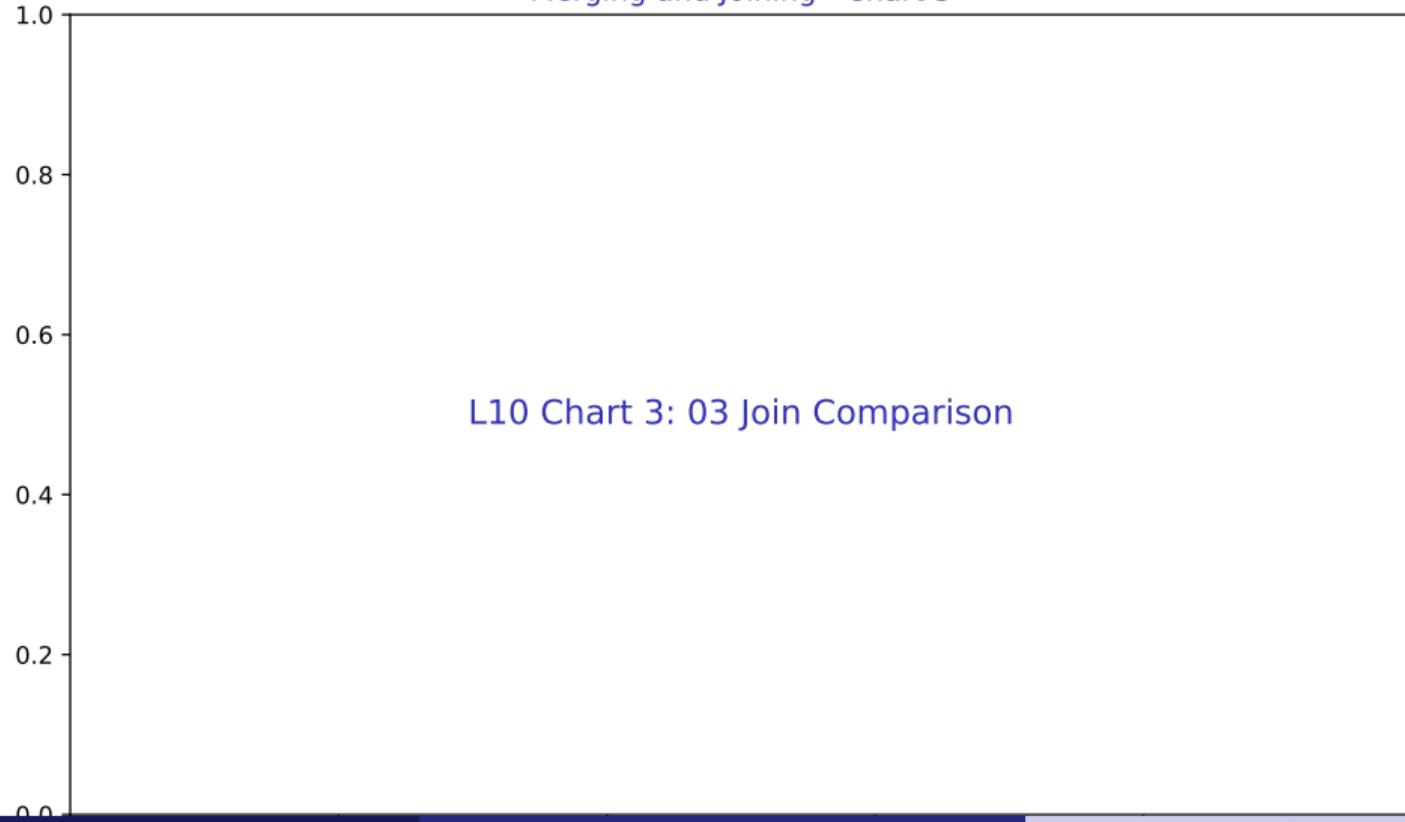
## Merging and Joining - Chart 1



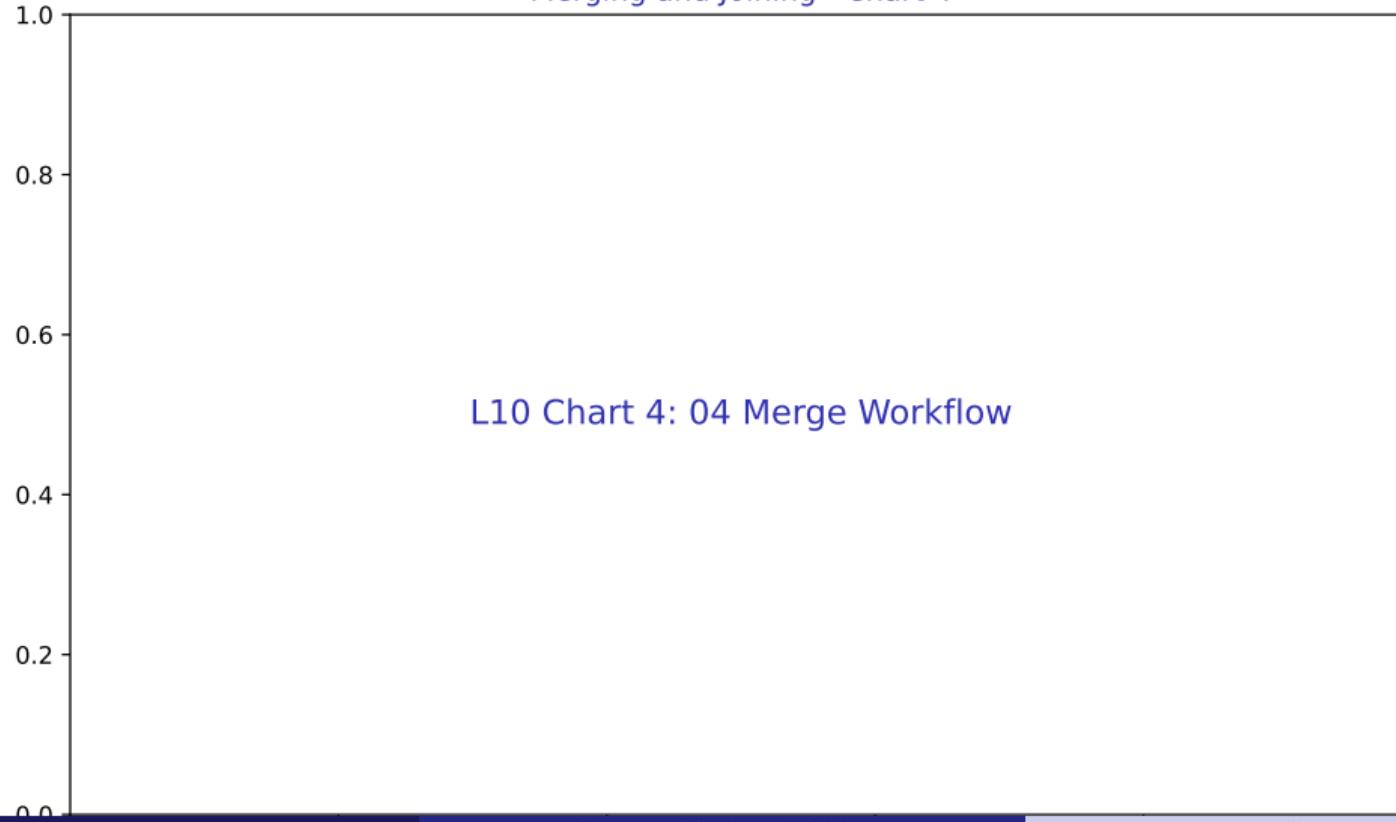
Merging and Joining - Chart 2



Merging and Joining - Chart 3

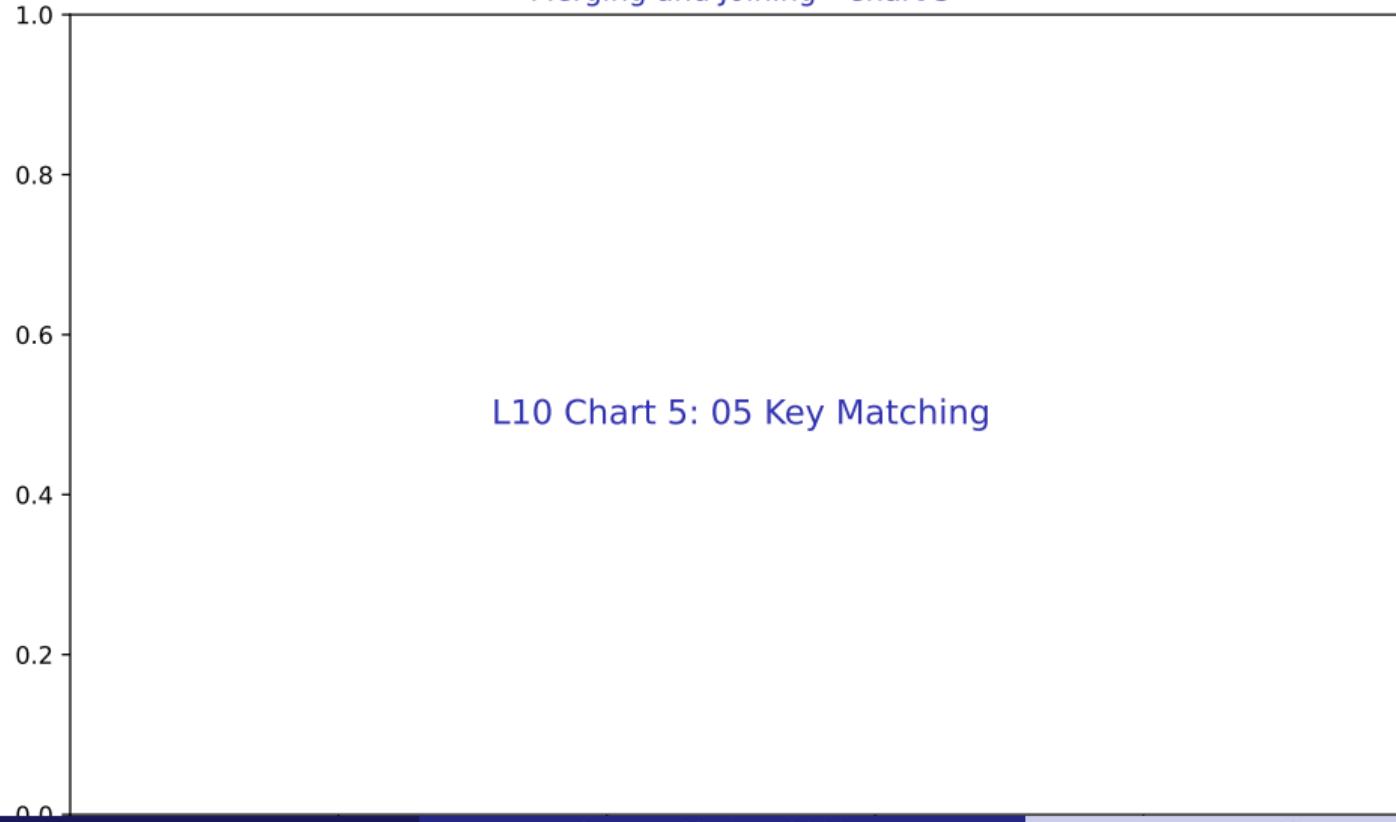


Merging and Joining - Chart 4

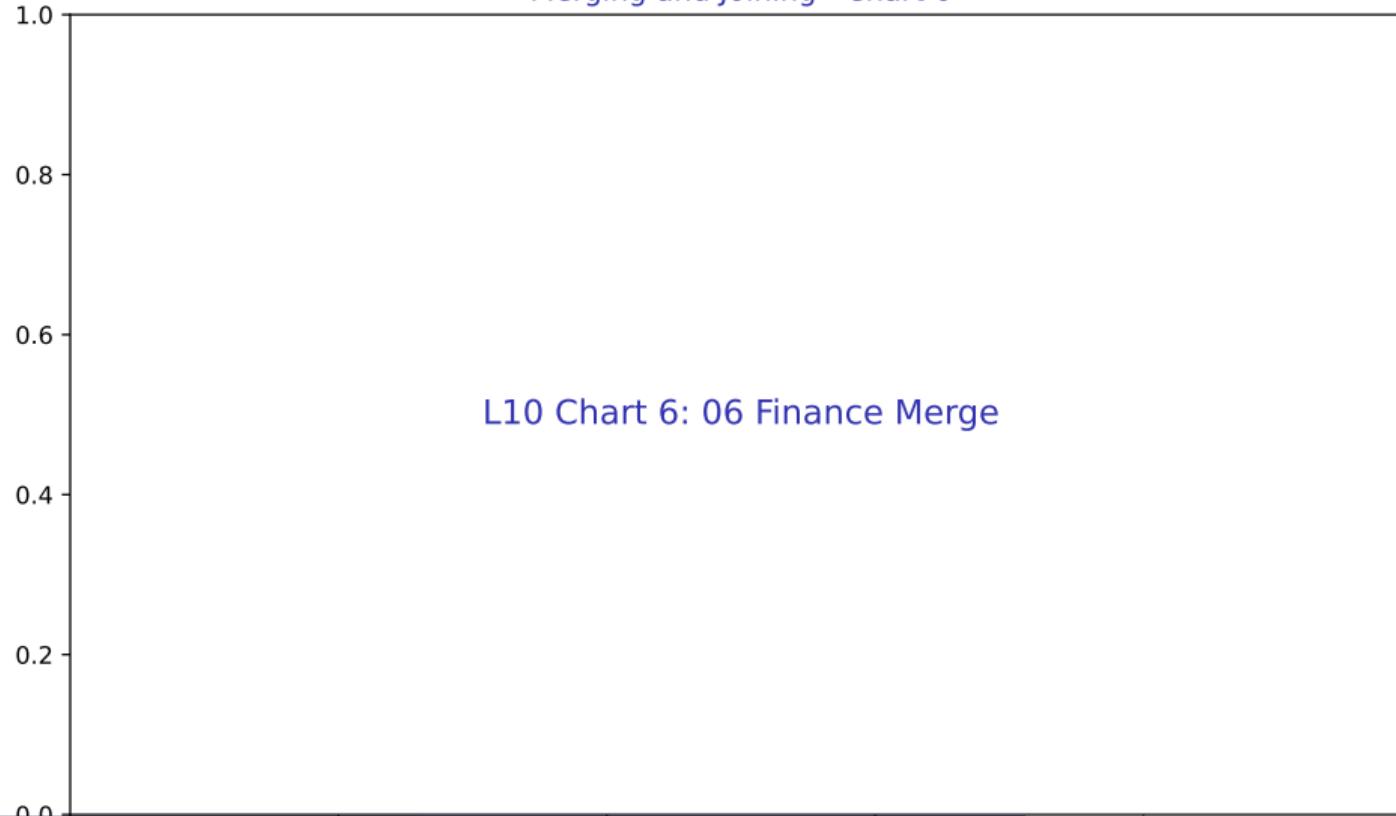


## 05 Key Matching

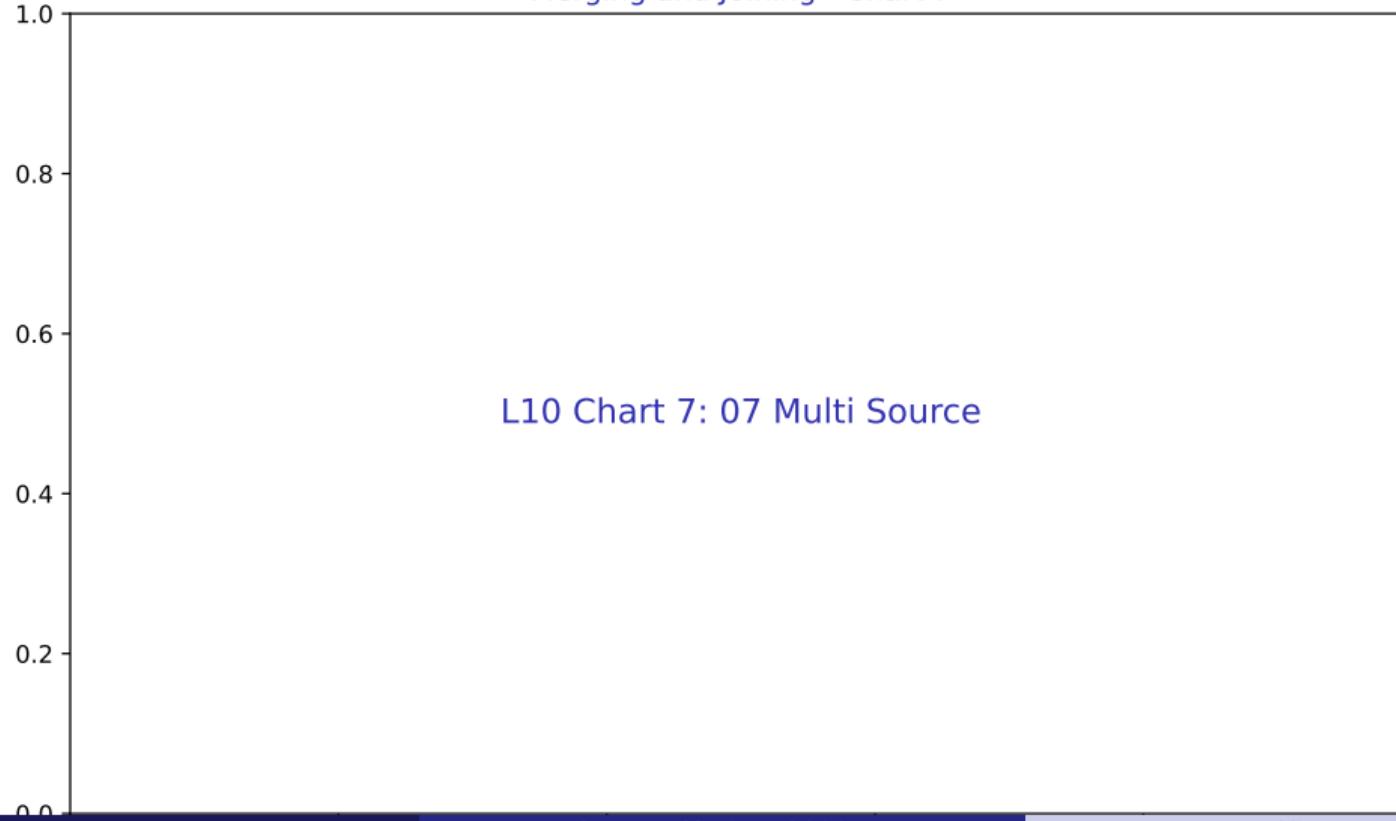
Merging and Joining - Chart 5



Merging and Joining - Chart 6



Merging and Joining - Chart 7



### Merging and Joining - Chart 8



### Key Takeaways:

- pd.concat() for stacking
- pd.merge() for SQL-style joins
- Join types: inner, outer, left, right
- Handling key columns

**Practice:** Apply these concepts to the stock price dataset.

## Lesson 11: NumPy Basics

Data Science with Python – BSc Course

45 Minutes

**After this lesson, you will be able to:**

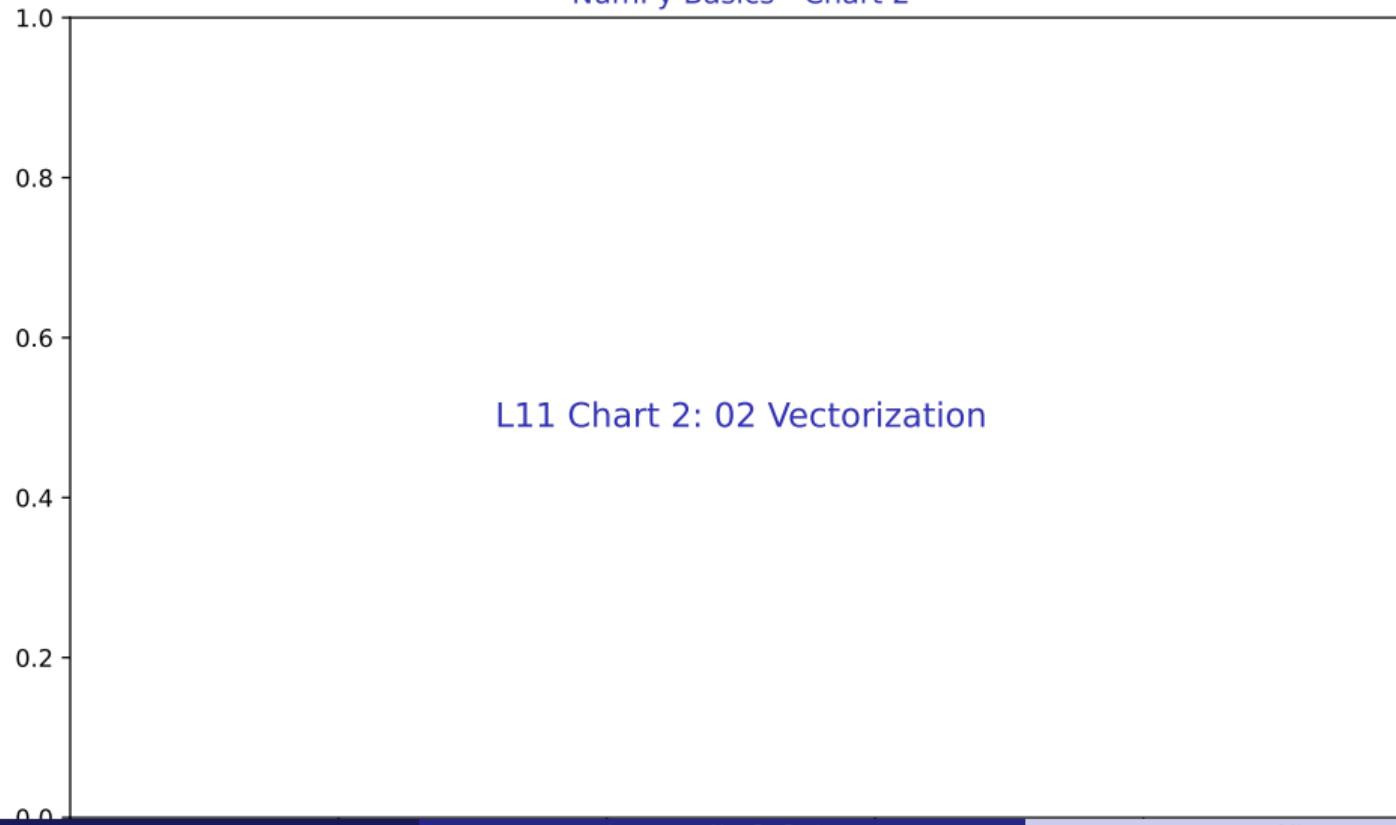
- Arrays vs lists
- Vectorized operations
- Broadcasting
- Mathematical functions
- Portfolio calculations

**Finance application: Stock data processing and analysis**

NumPy Basics - Chart 1

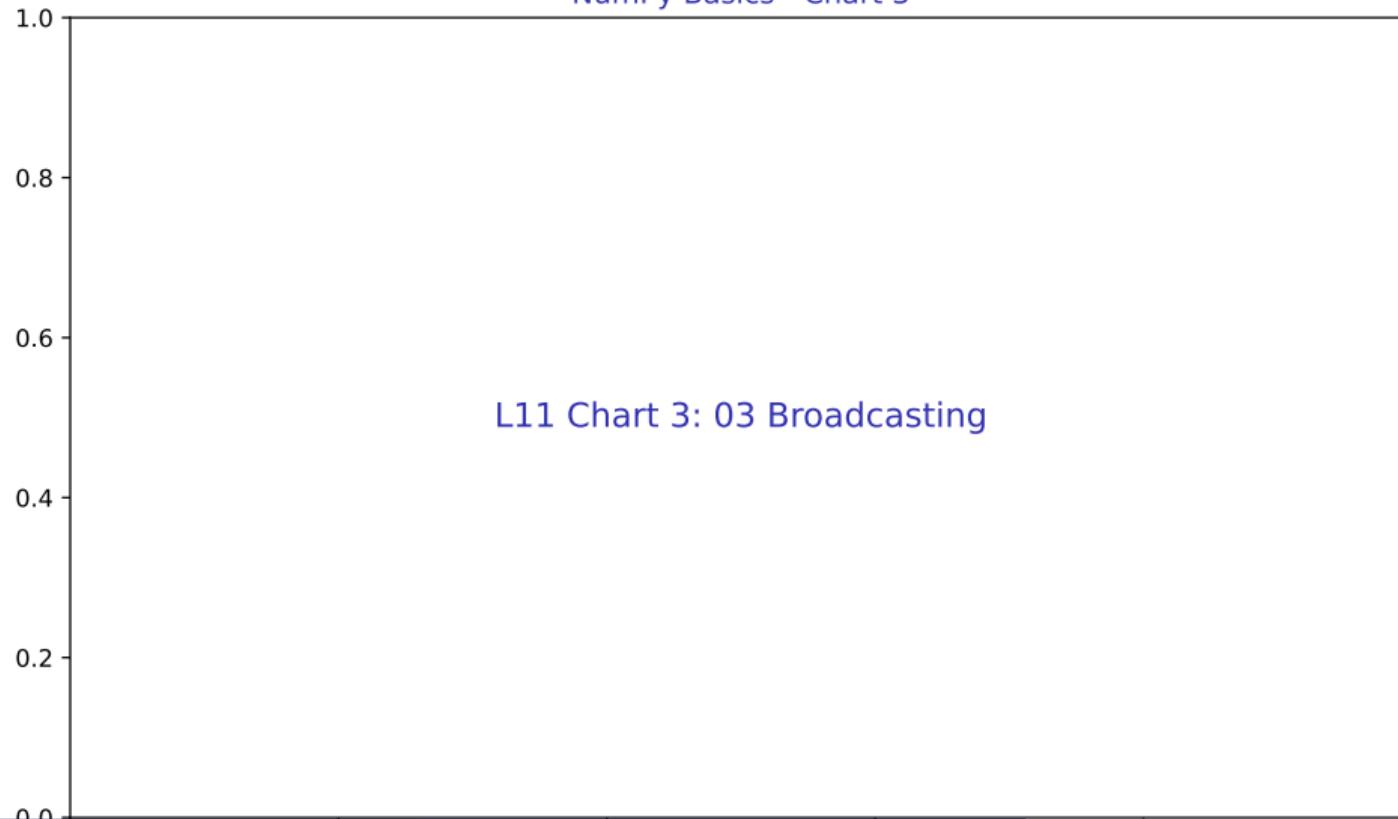


NumPy Basics - Chart 2



## 03 Broadcasting

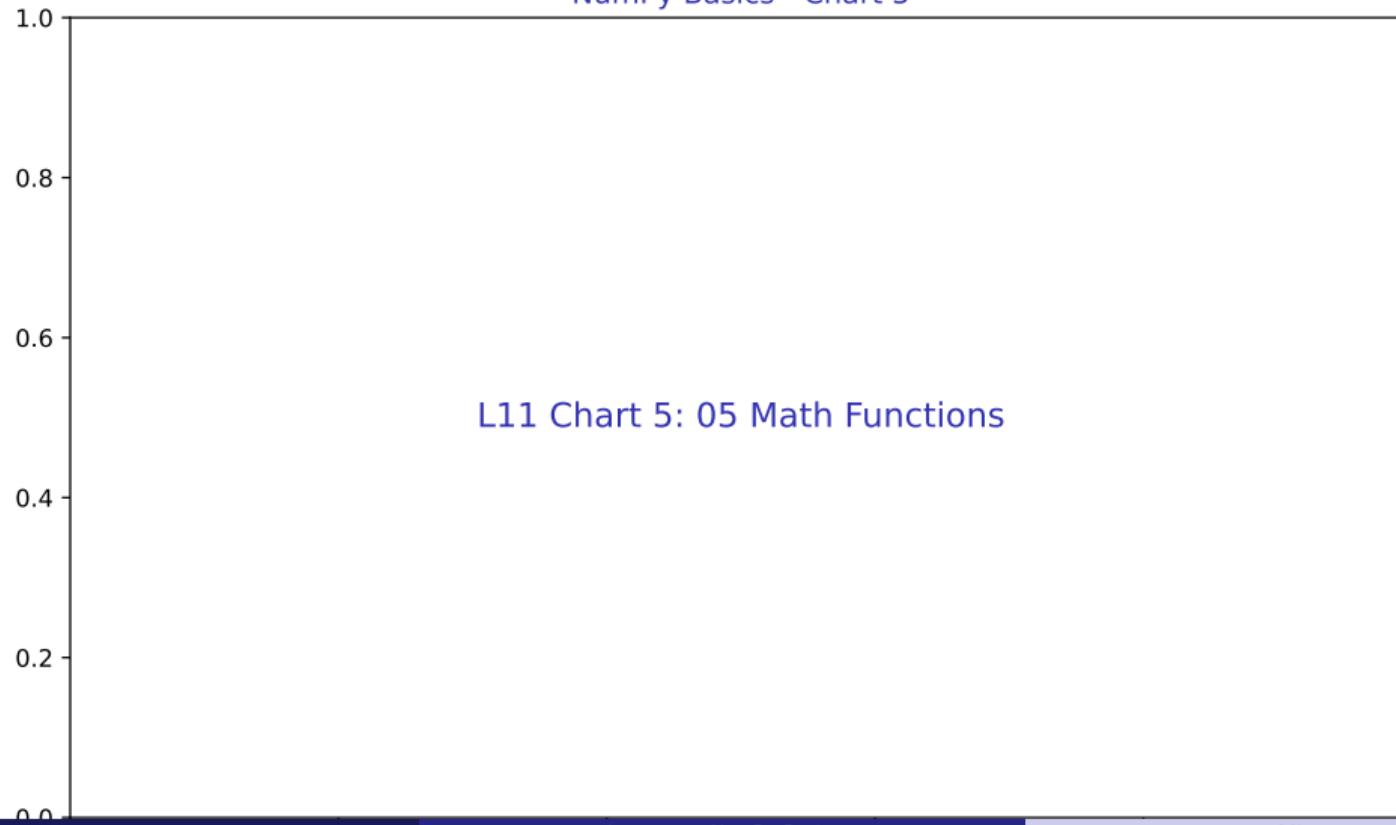
NumPy Basics - Chart 3



NumPy Basics - Chart 4

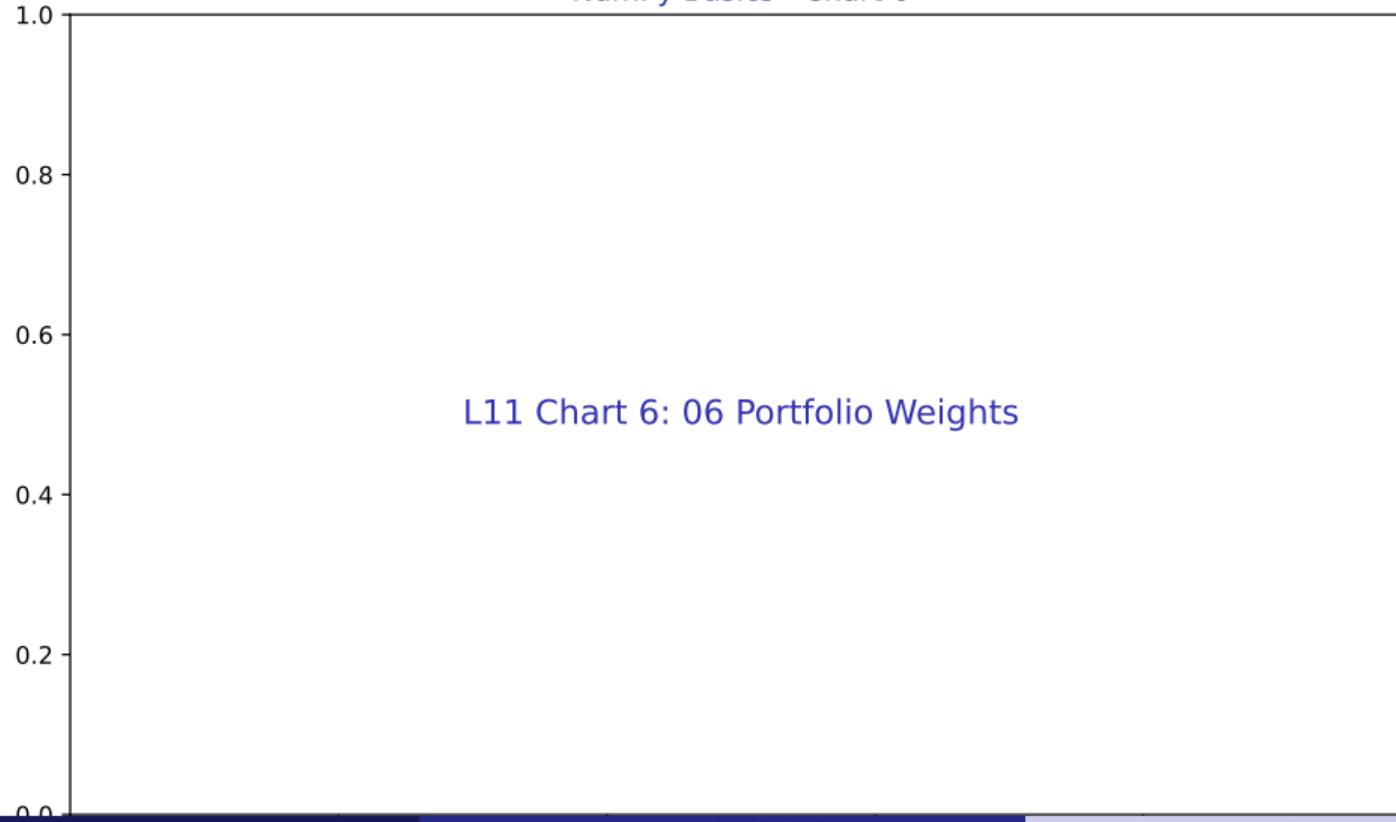


NumPy Basics - Chart 5

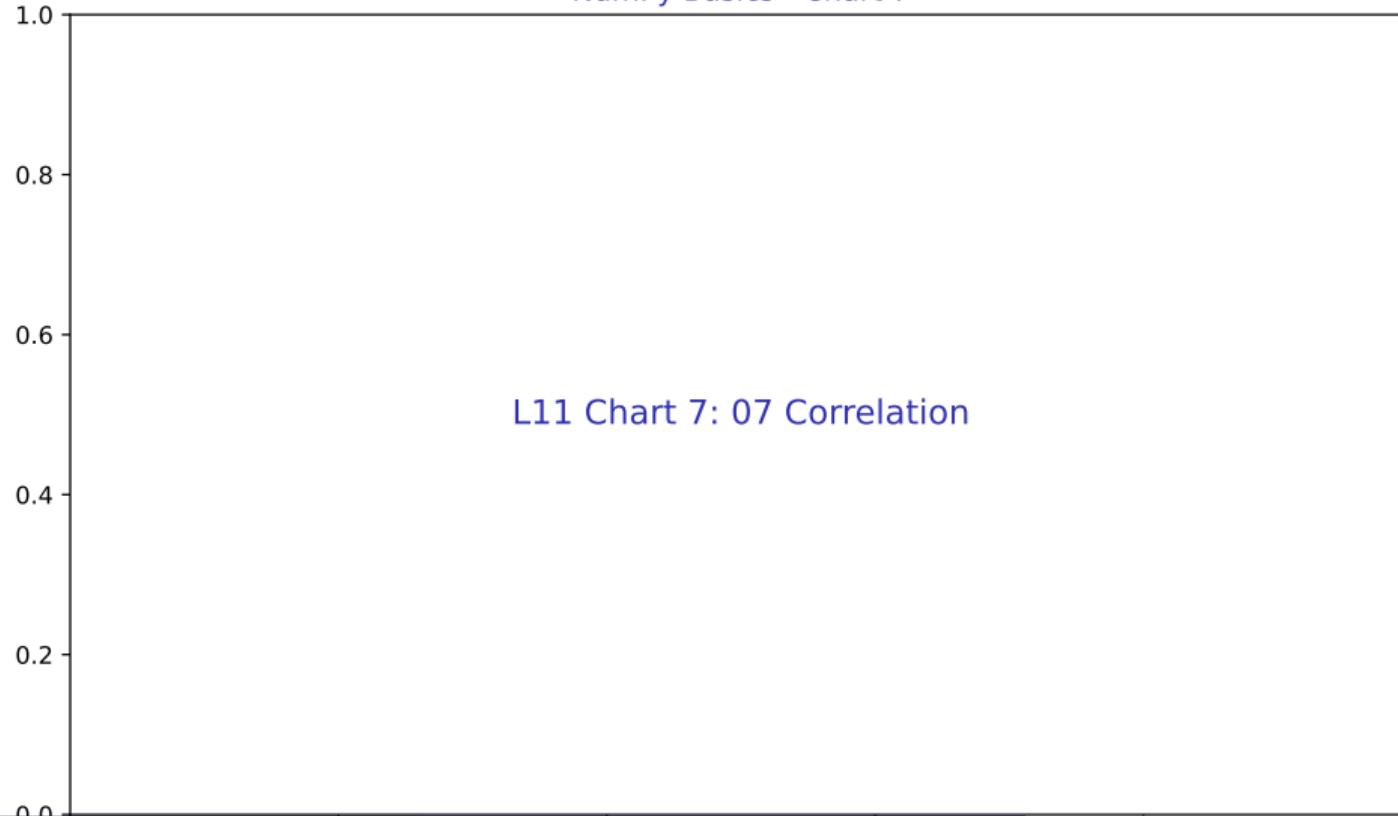


## 06 Portfolio Weights

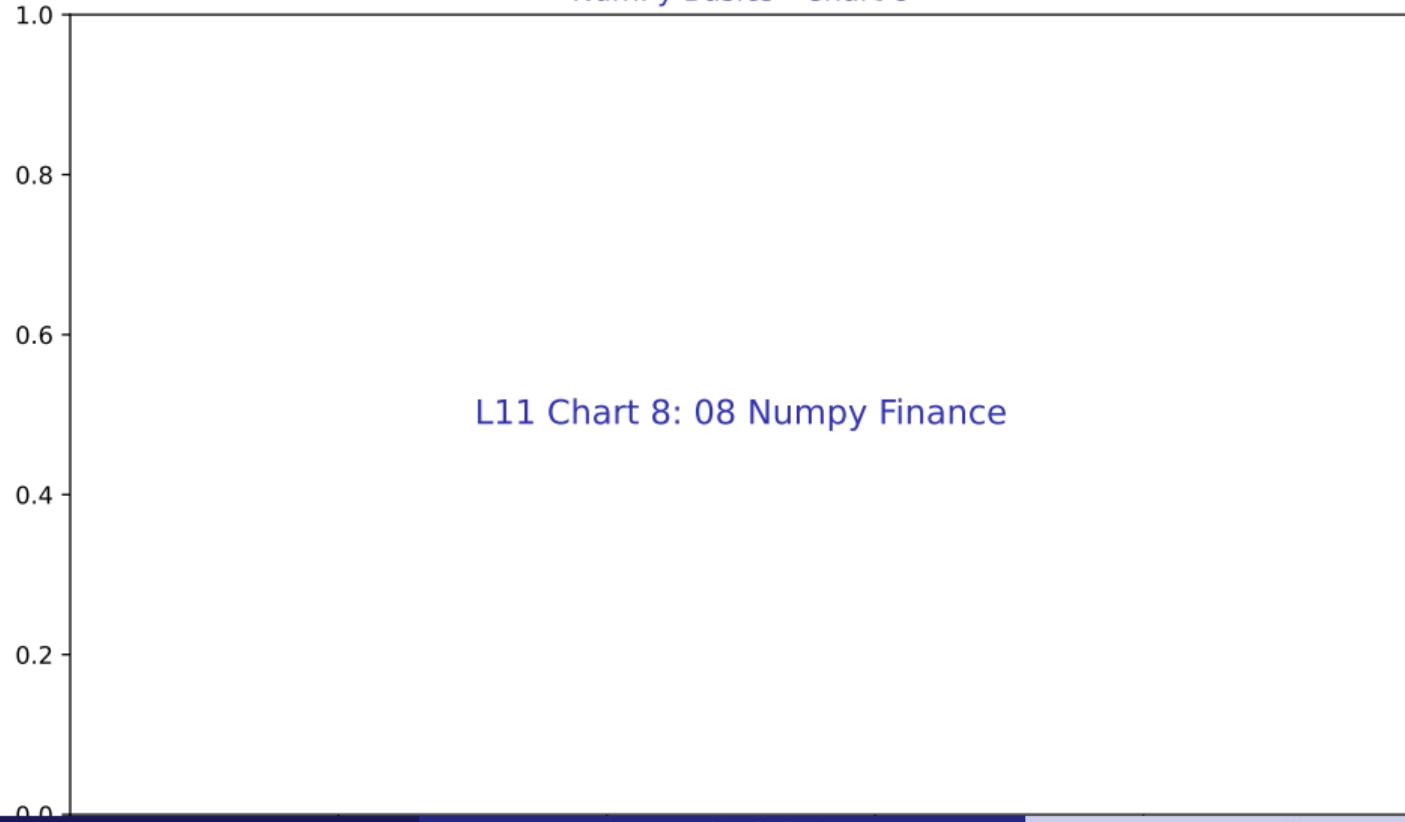
NumPy Basics - Chart 6



NumPy Basics - Chart 7



NumPy Basics - Chart 8



### Key Takeaways:

- Arrays vs lists
- Vectorized operations
- Broadcasting
- Mathematical functions
- Portfolio calculations

**Practice:** Apply these concepts to the stock price dataset.

## Lesson 12: Time Series Basics

Data Science with Python – BSc Course

45 Minutes

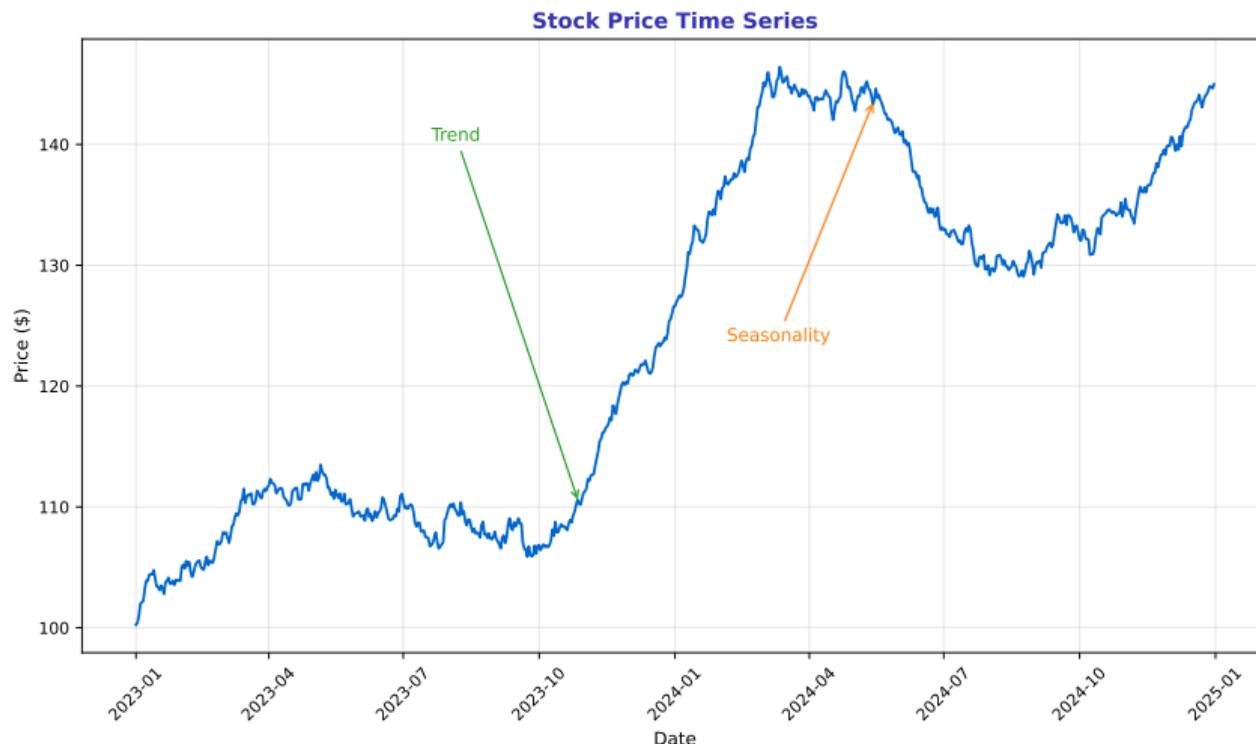
# Learning Objectives

**After this lesson, you will be able to:**

- DateTime index
- Resampling (daily to monthly)
- Rolling windows
- shift() and pct\_change()
- Time series patterns in finance

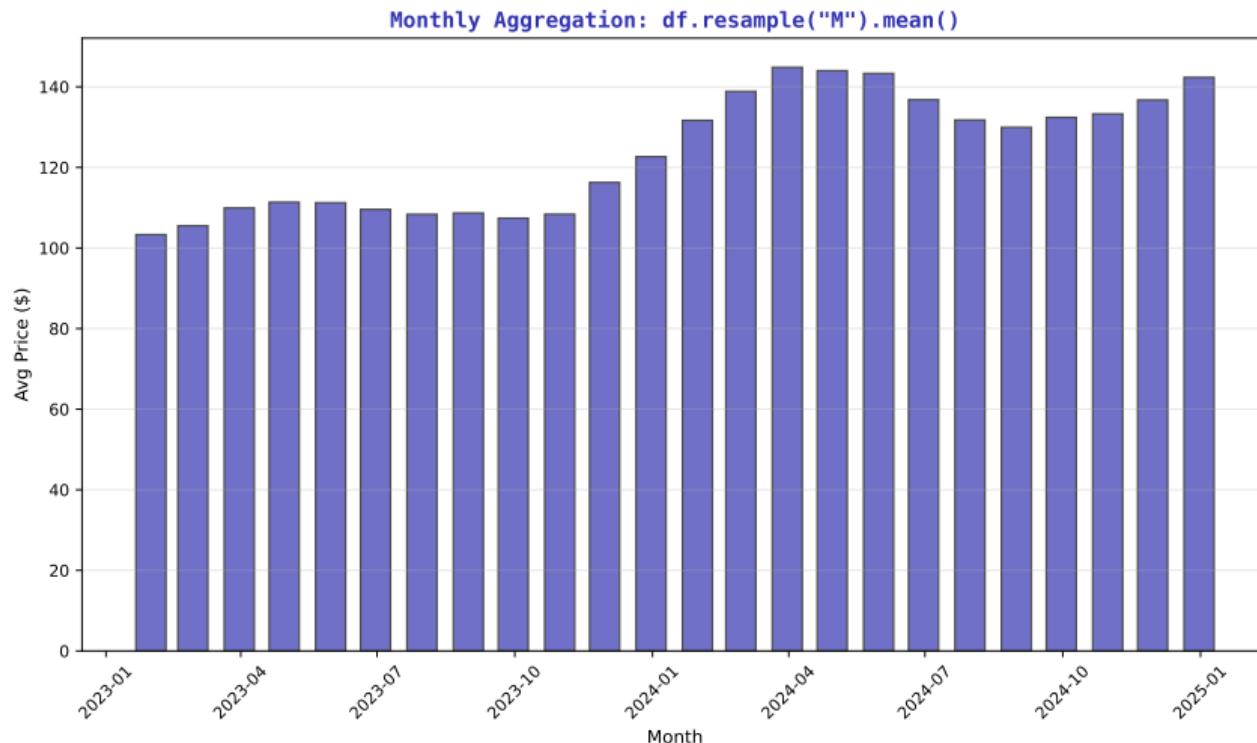
**Finance application: Stock data processing and analysis**

# Basic Time Series



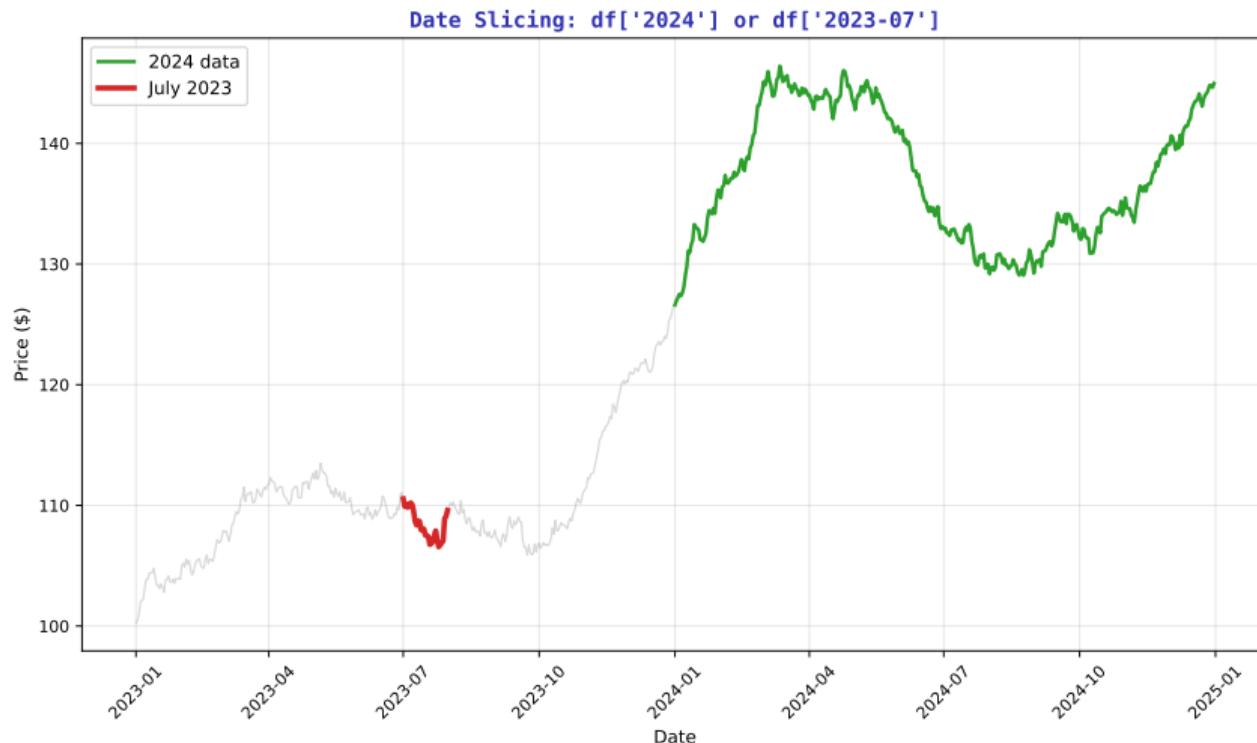
Plotting time-indexed data

# Monthly Aggregation



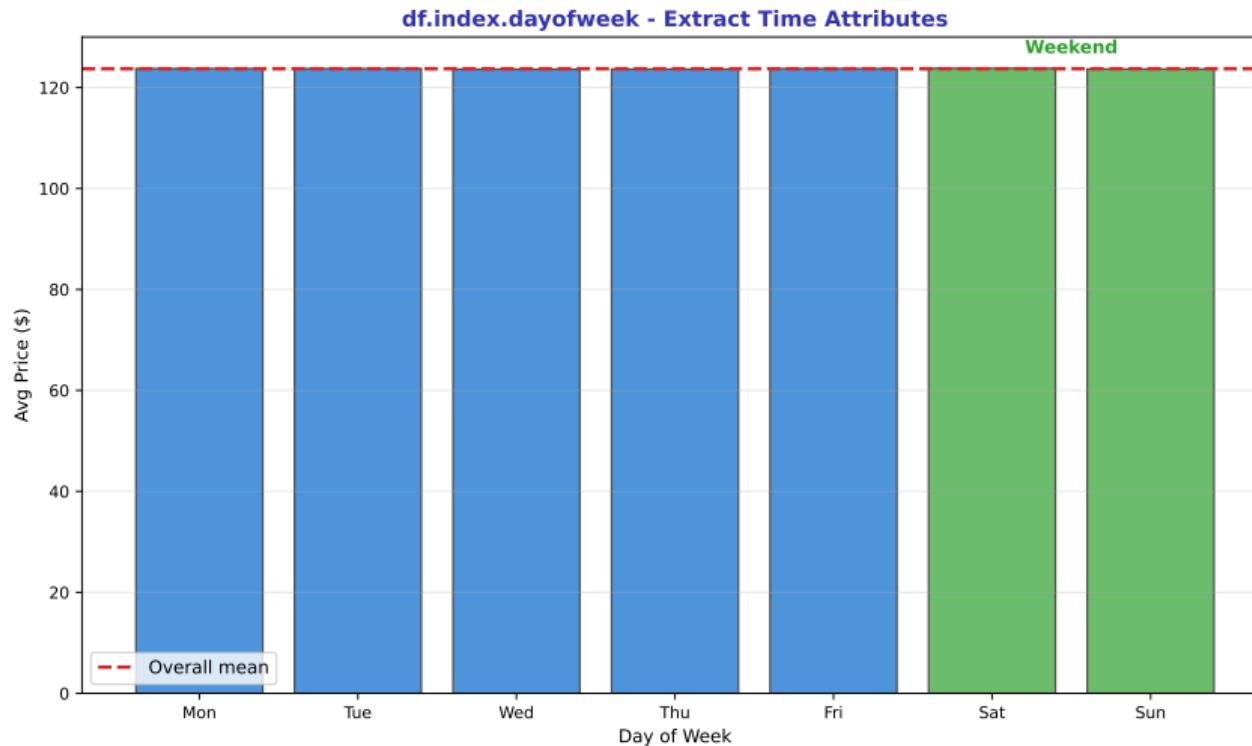
Summarizing daily data into monthly periods

## Date Slicing



Selecting data by date range

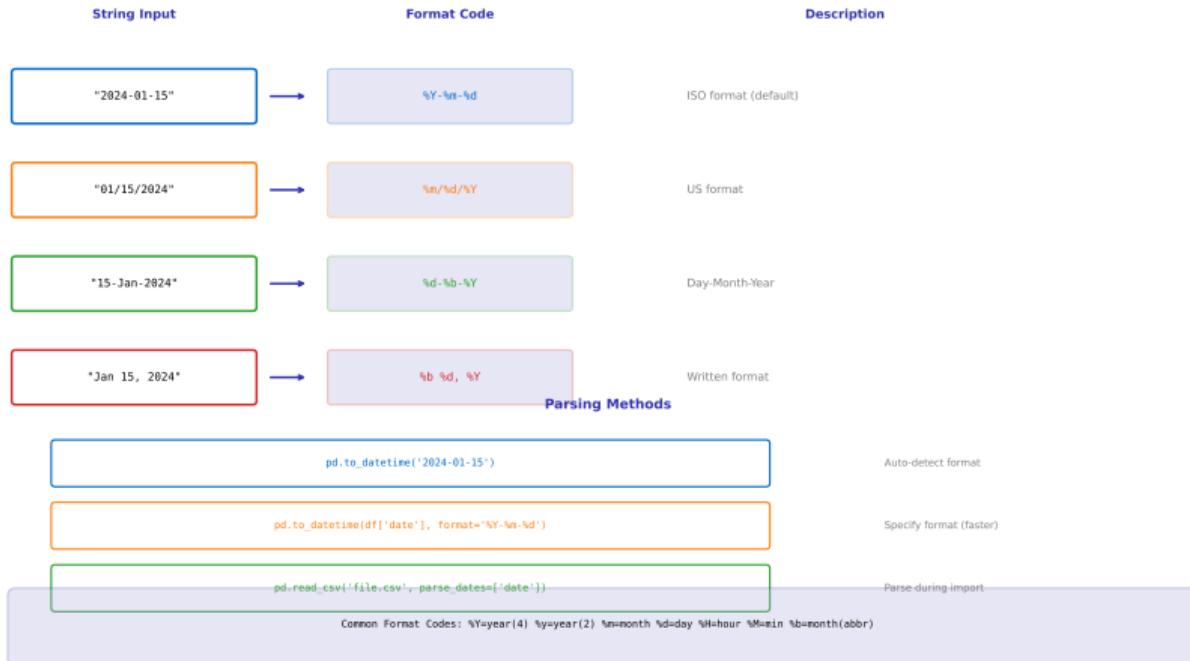
## Day of Week Patterns



Analyzing weekday effects

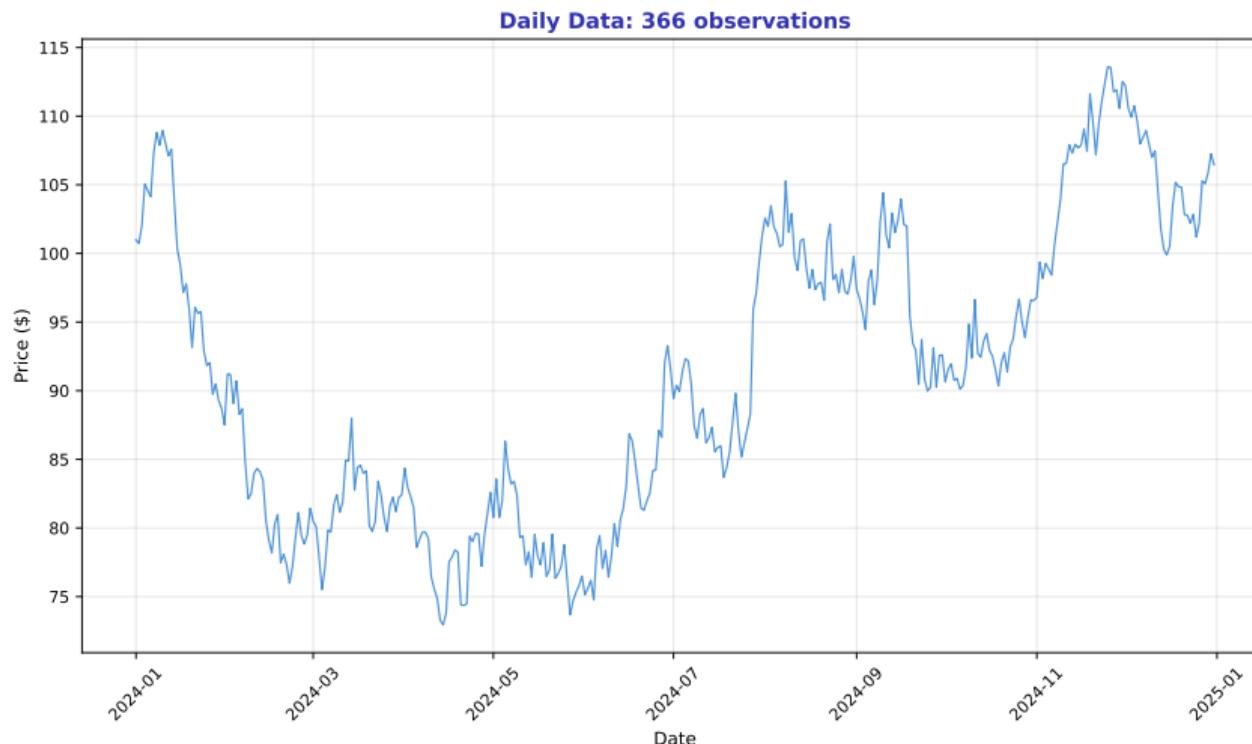
# Datetime Parsing

## Datetime Parsing in pandas



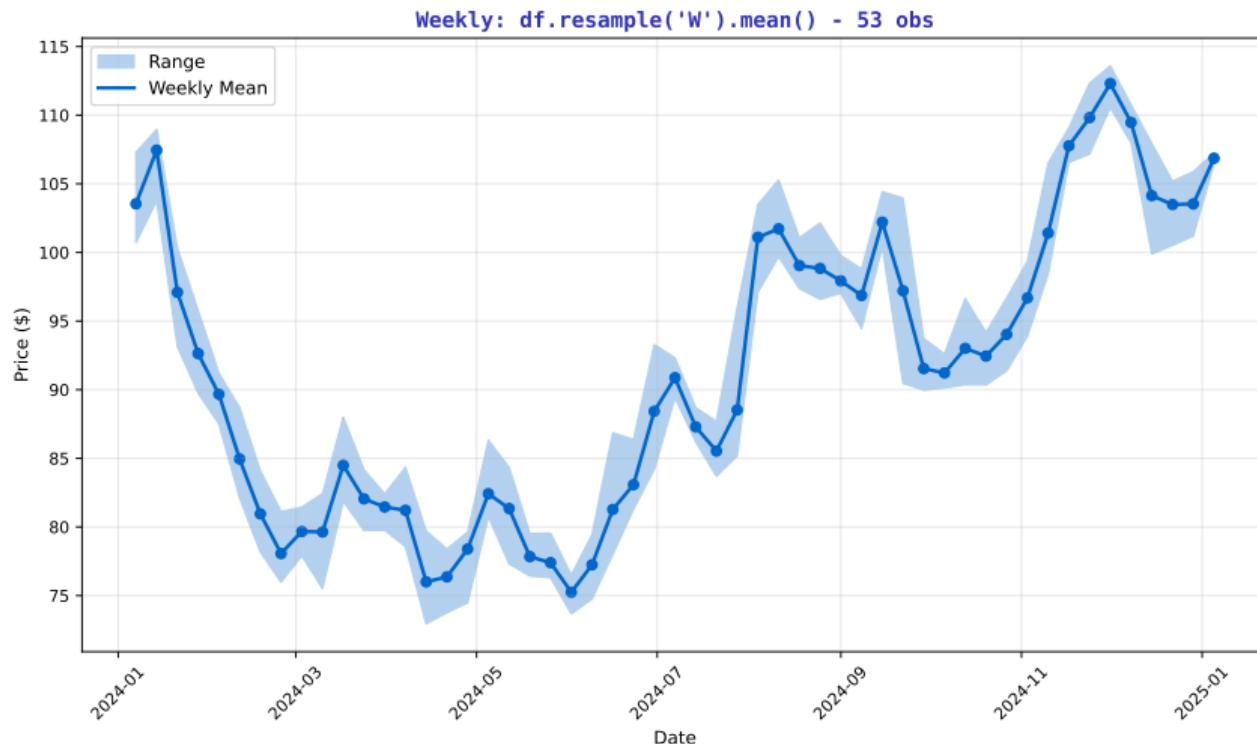
## Converting strings to datetime objects

## Daily Data



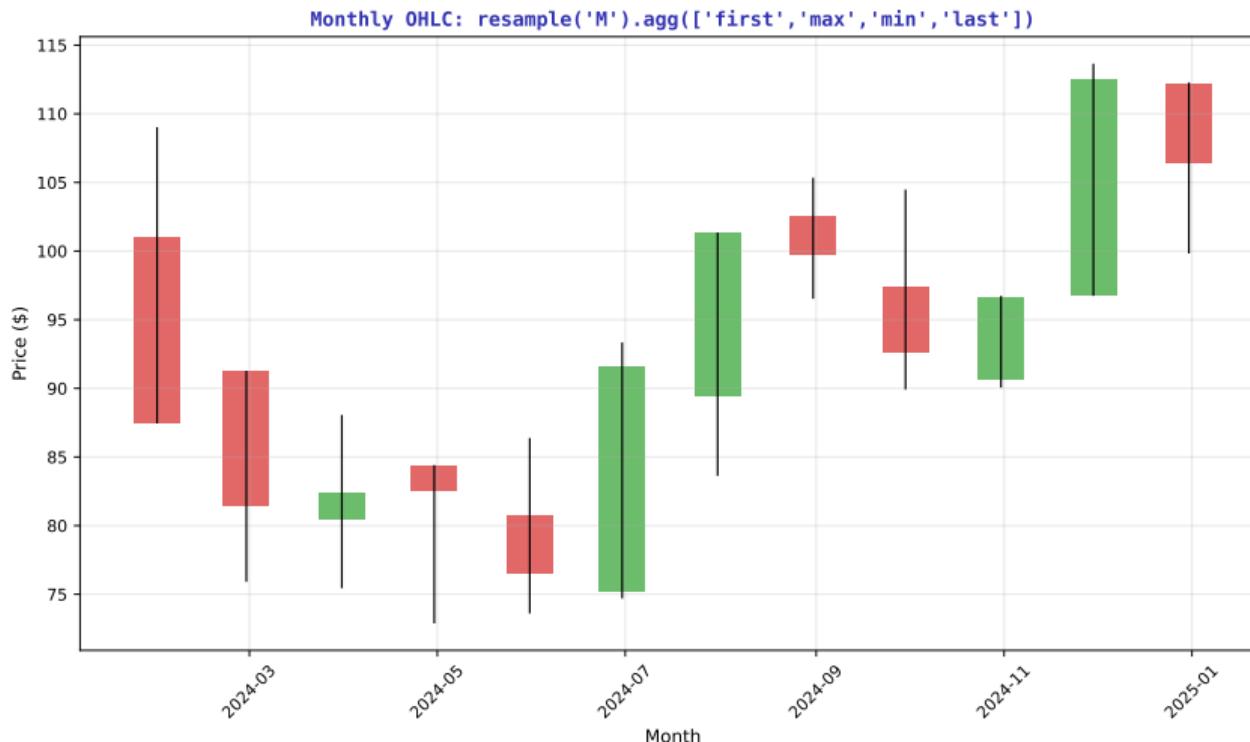
Working with daily frequency

## Weekly Resampling



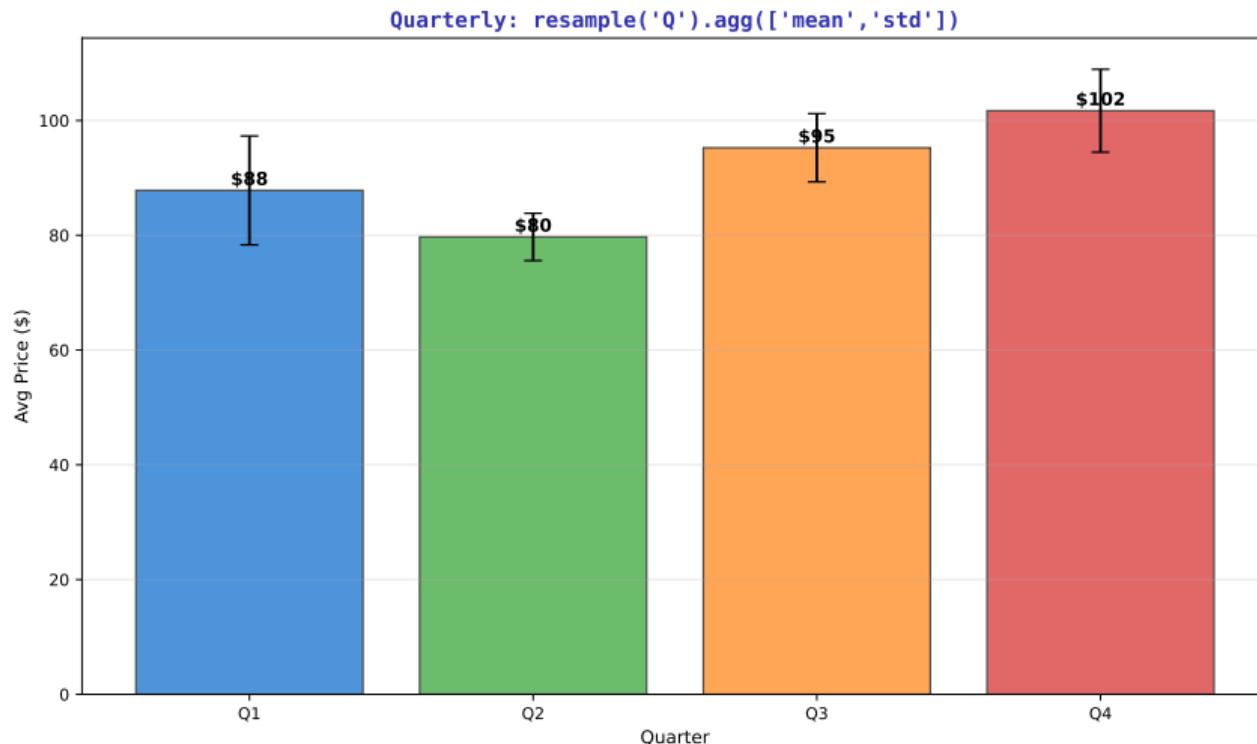
Aggregating to weekly frequency

# Monthly OHLC



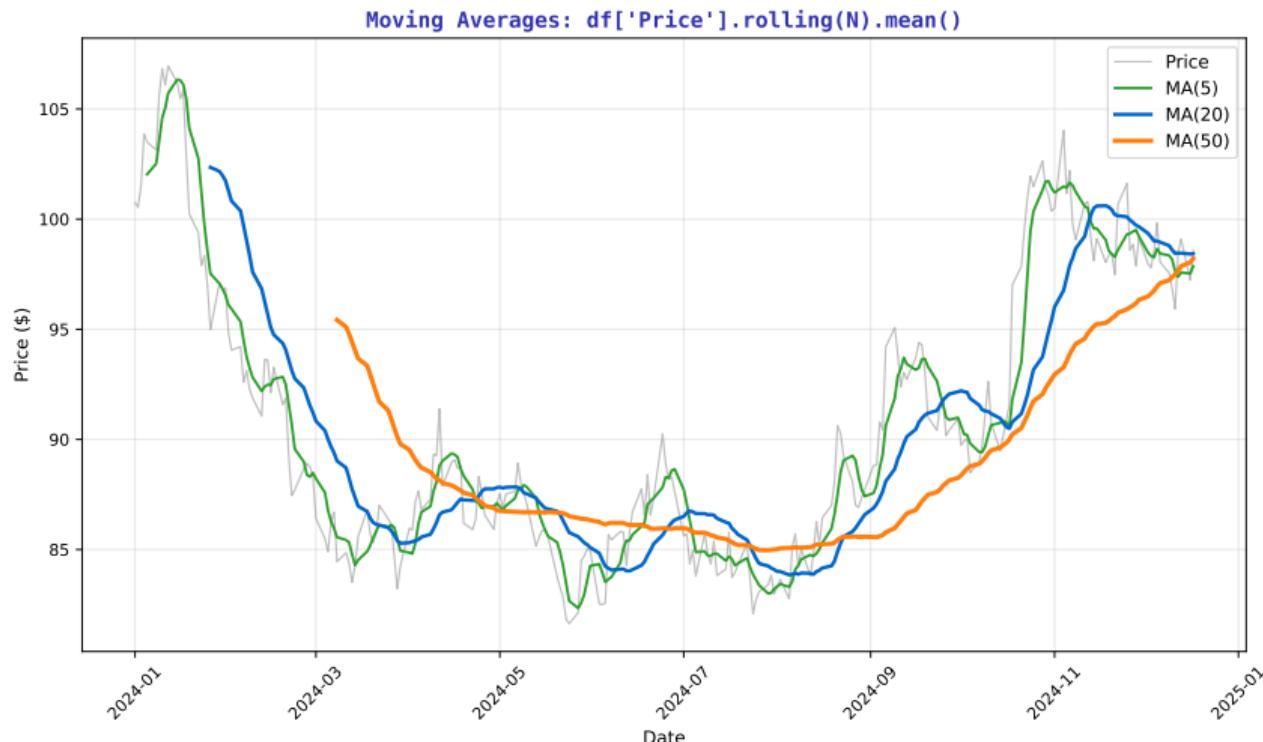
Open-High-Low-Close monthly bars

## Quarterly Resampling



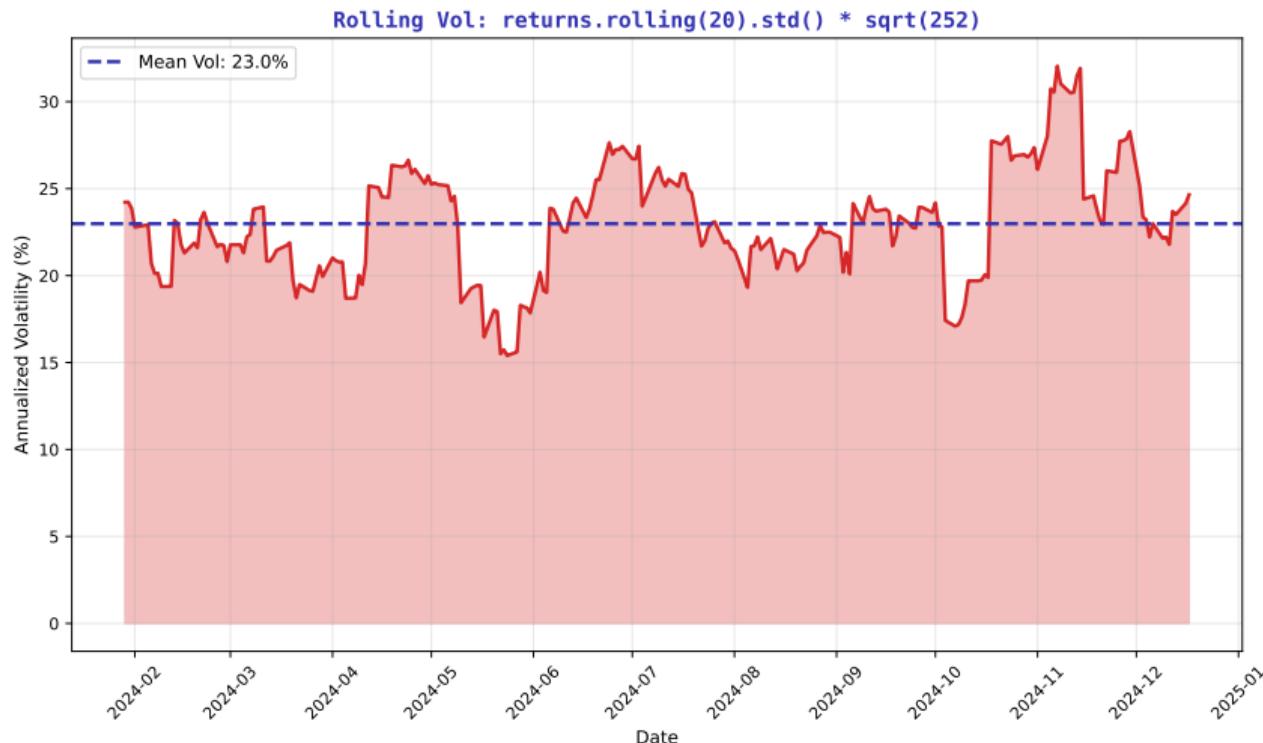
Aggregating to quarterly frequency

# Moving Averages



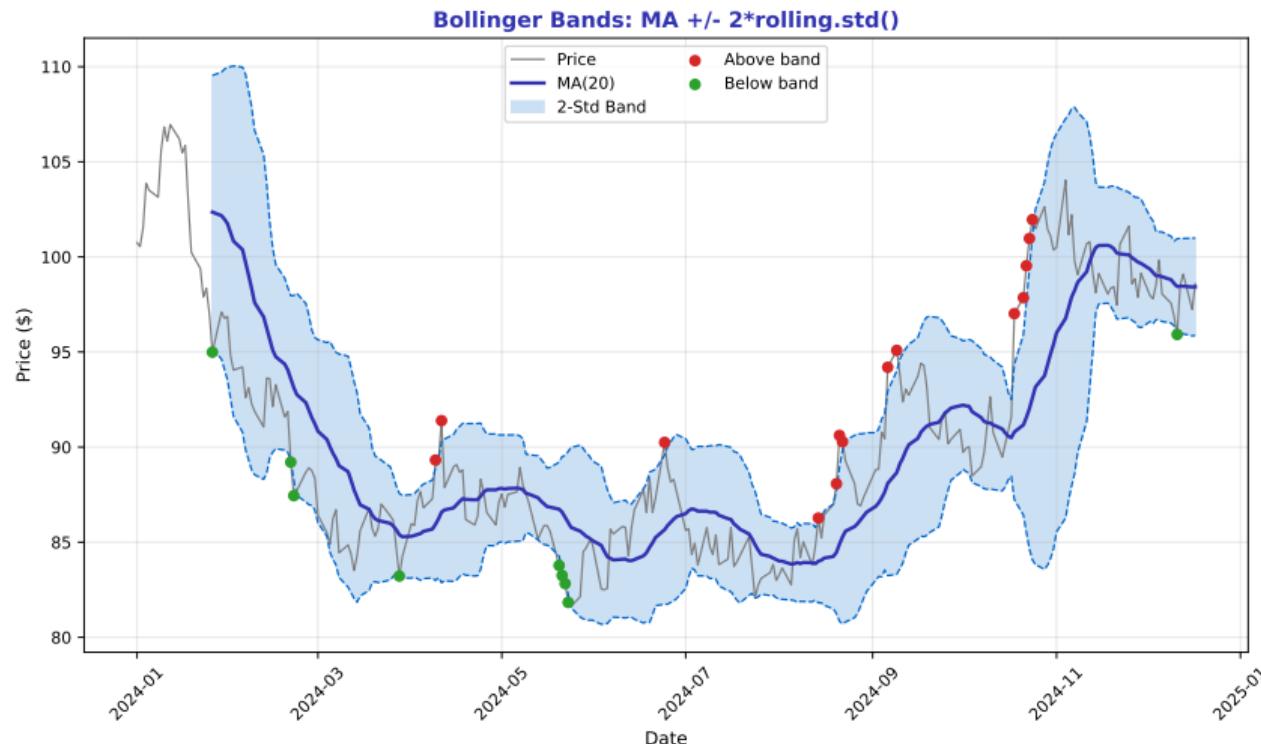
## Smoothing with rolling mean

## Rolling Volatility



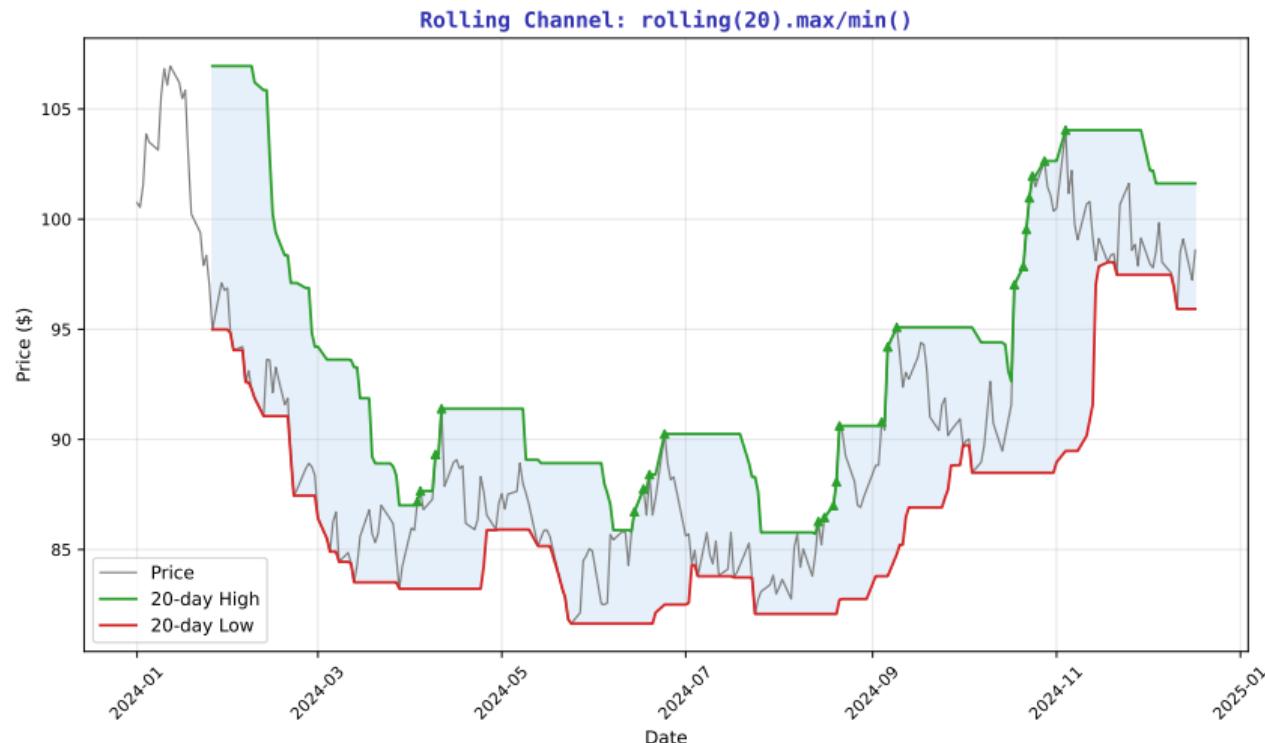
Measuring time-varying risk

# Bollinger Bands



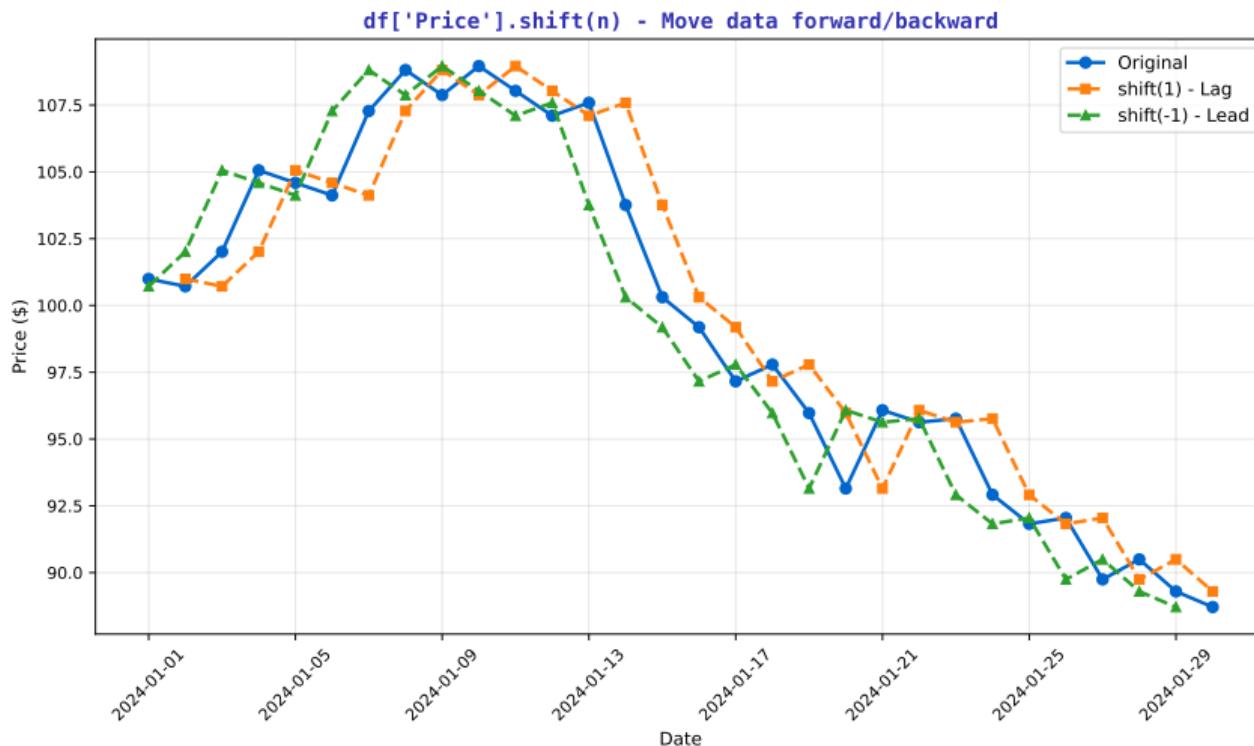
Trading bands using rolling statistics

## Rolling Channel



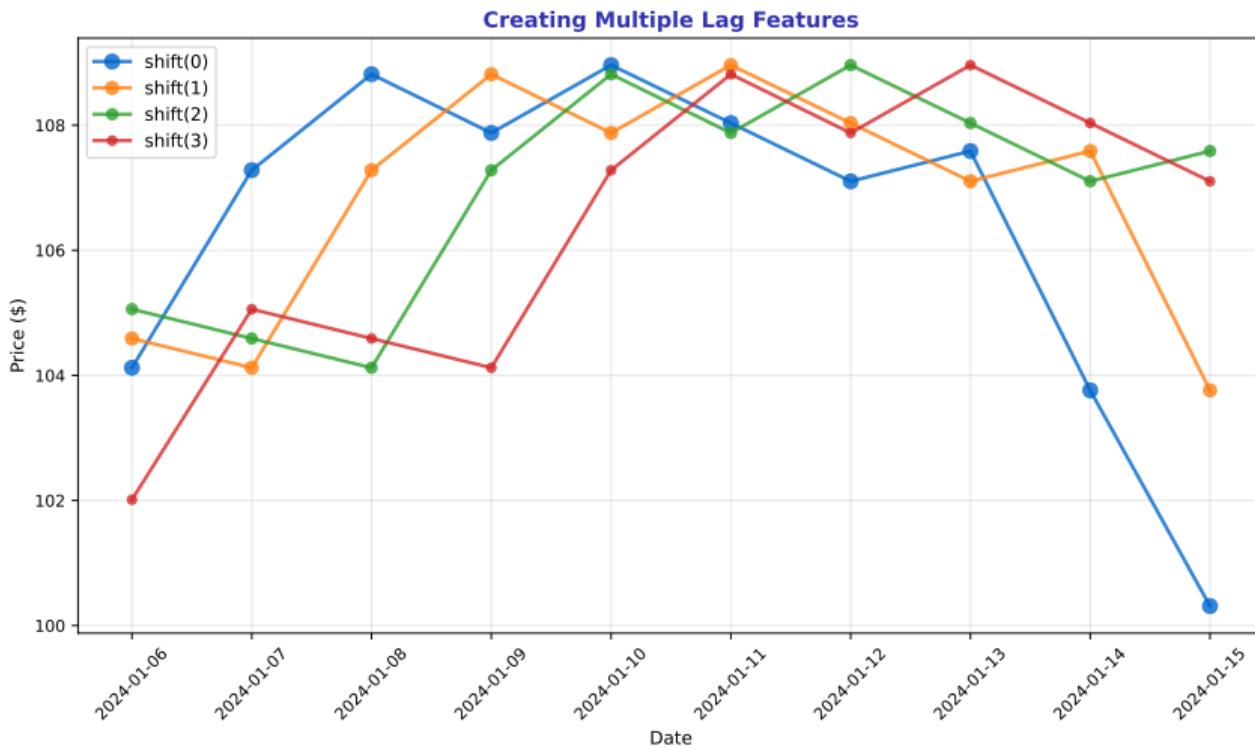
Price channels with rolling min/max

## Shift Demo



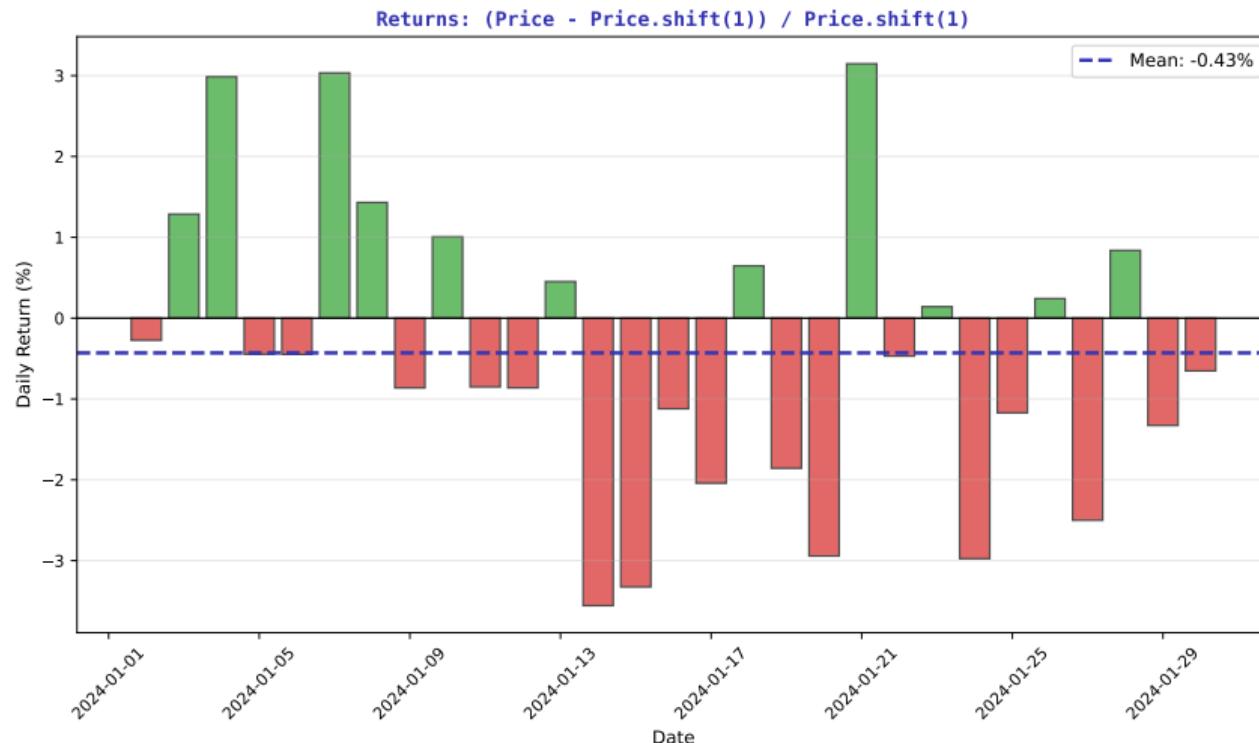
### Basic lag operation

# Multiple Lags



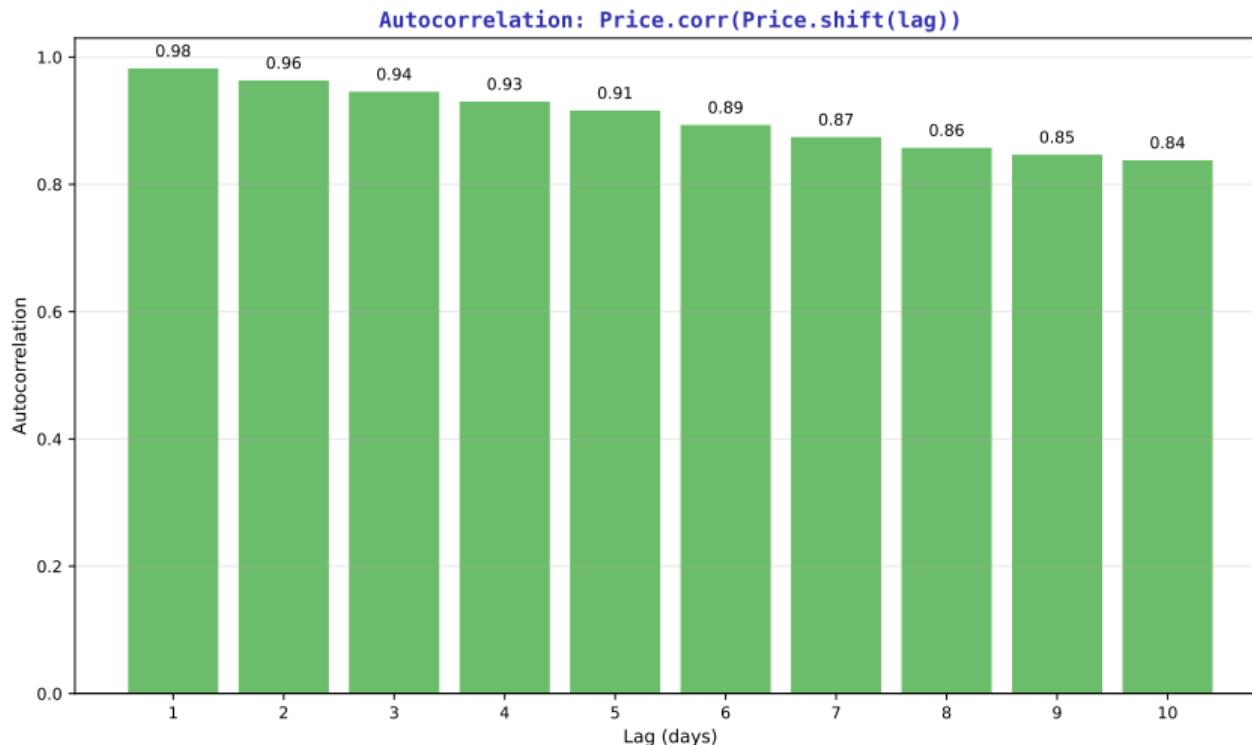
Creating multiple lagged features

## Returns with Shift



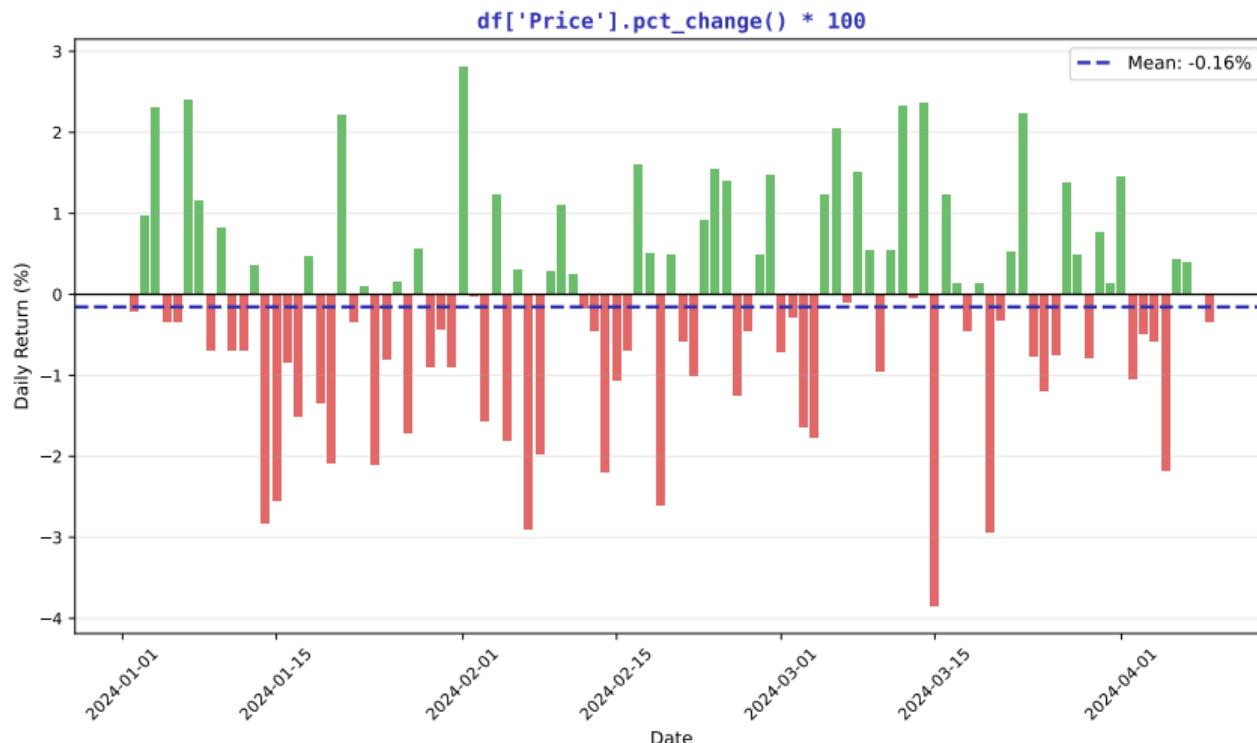
Computing returns using shift

# Autocorrelation



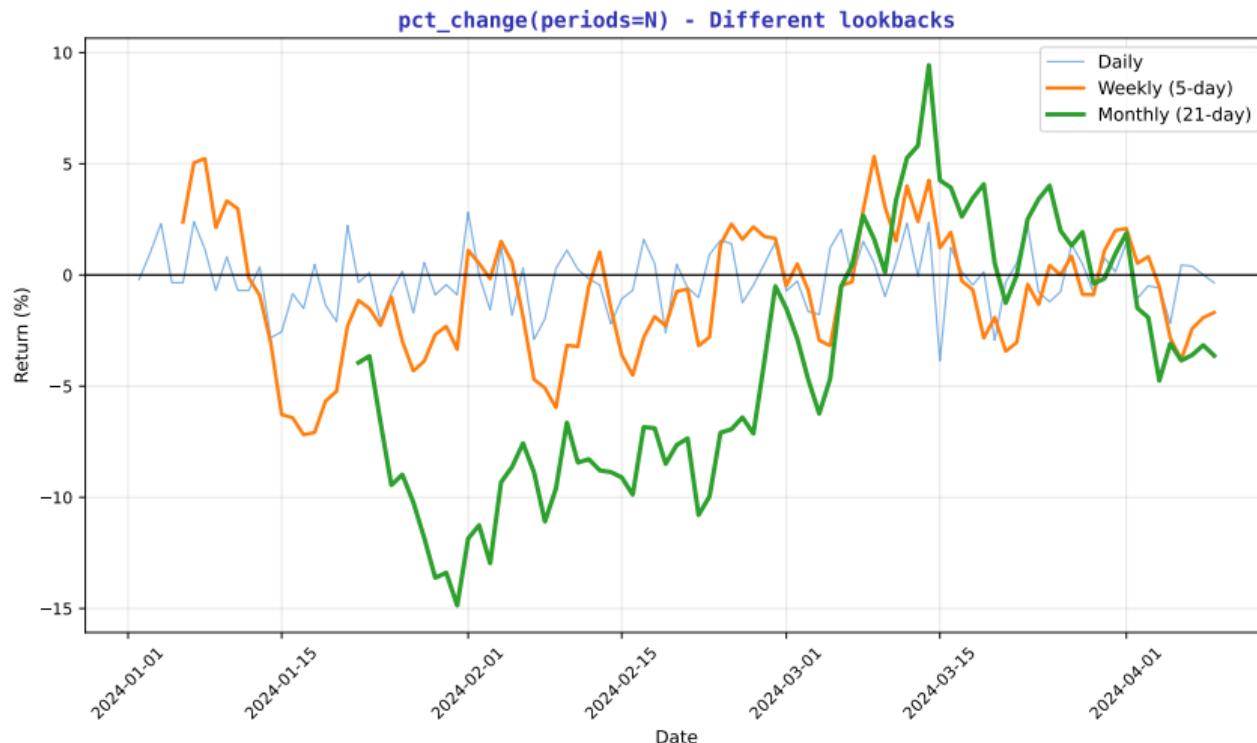
Serial correlation in returns

## Daily Returns



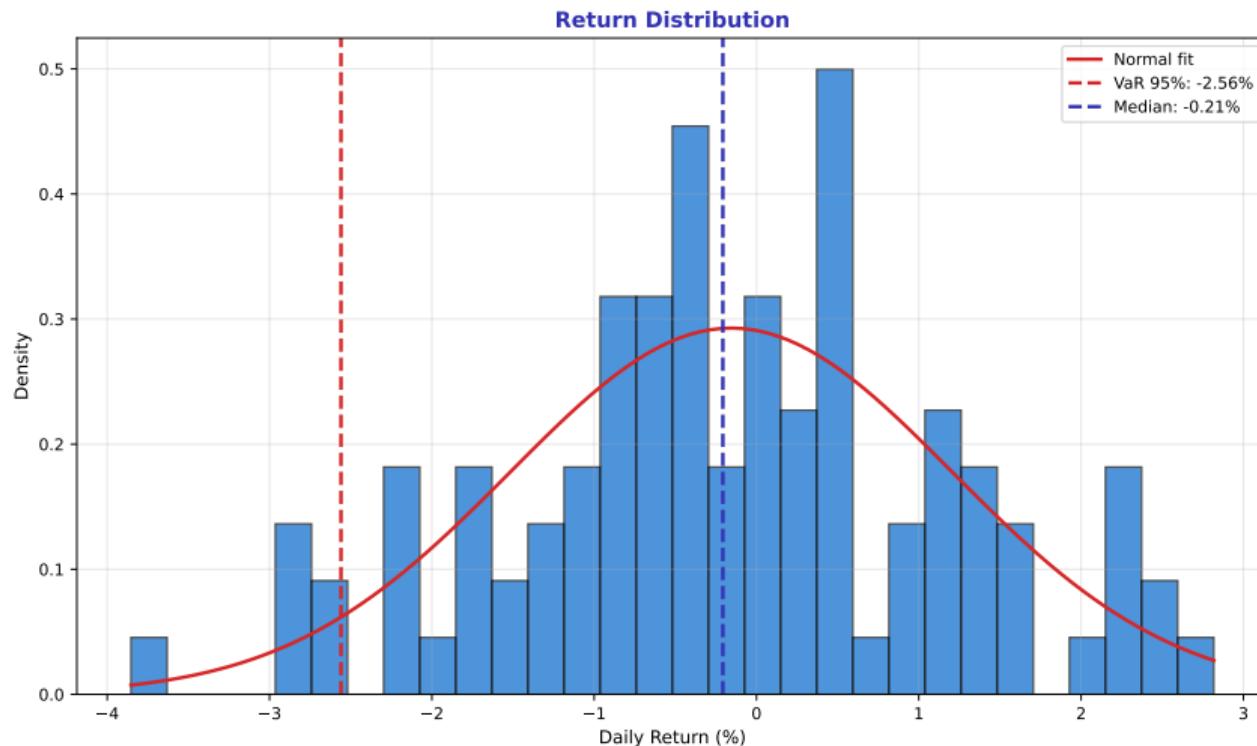
Computing percentage changes

## Periods Comparison



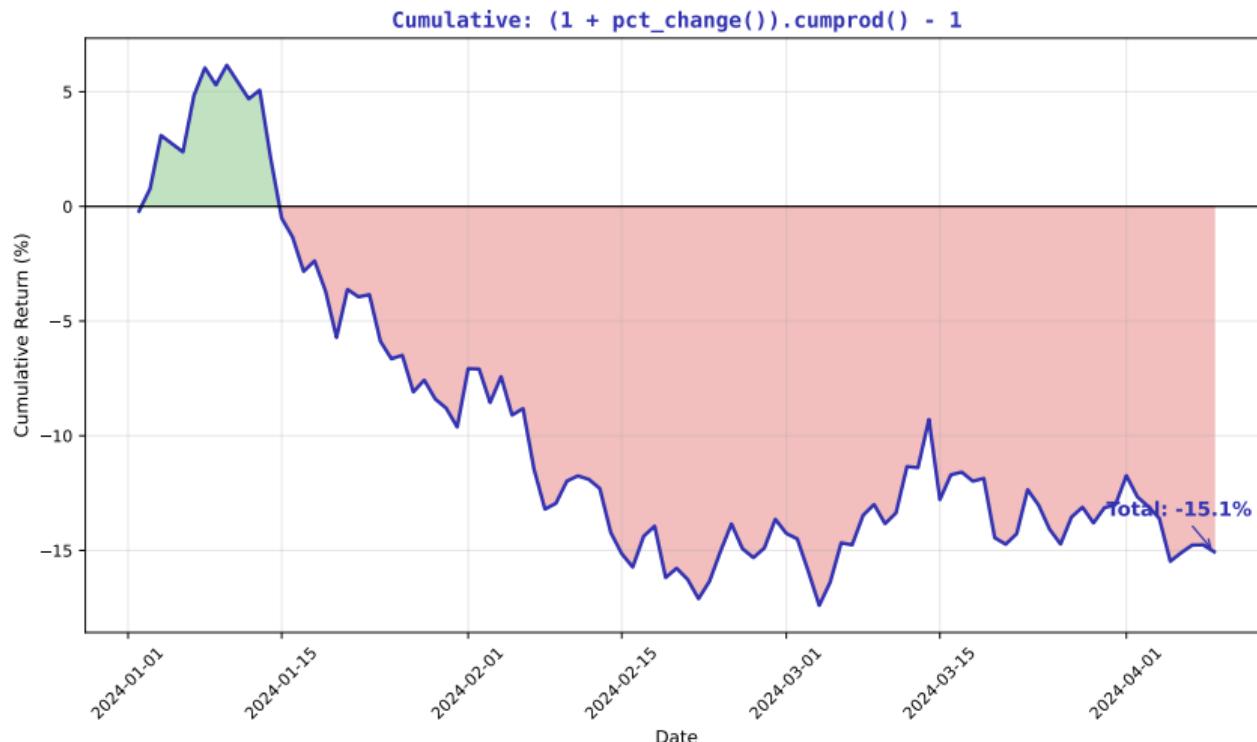
Different return periods

# Return Distribution



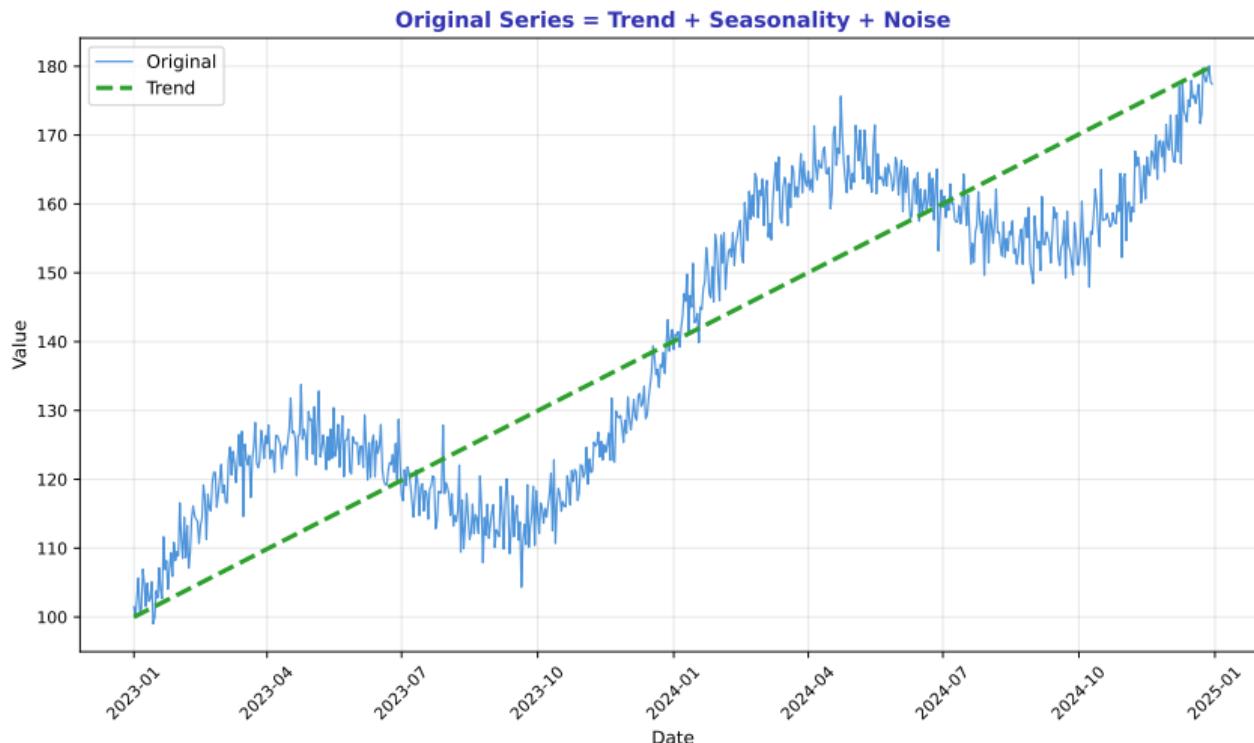
## Statistical properties of returns

## Cumulative Returns



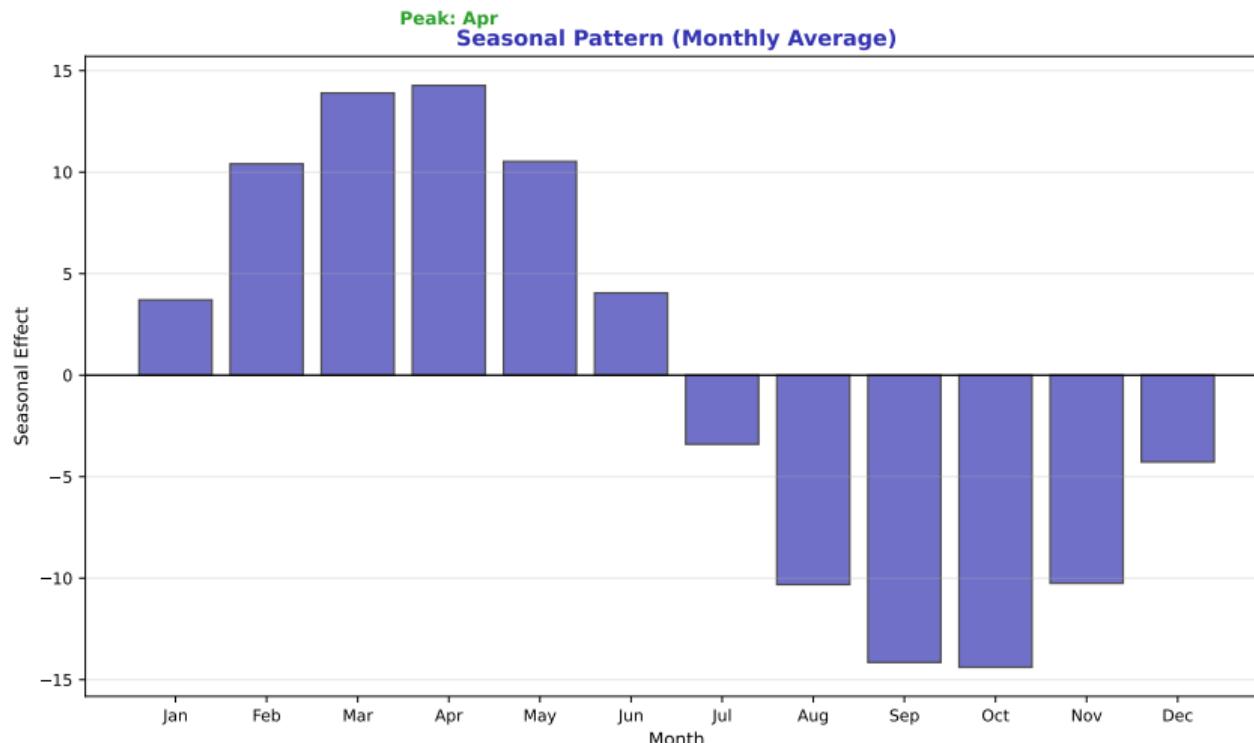
Growth of investment over time

## Original Trend



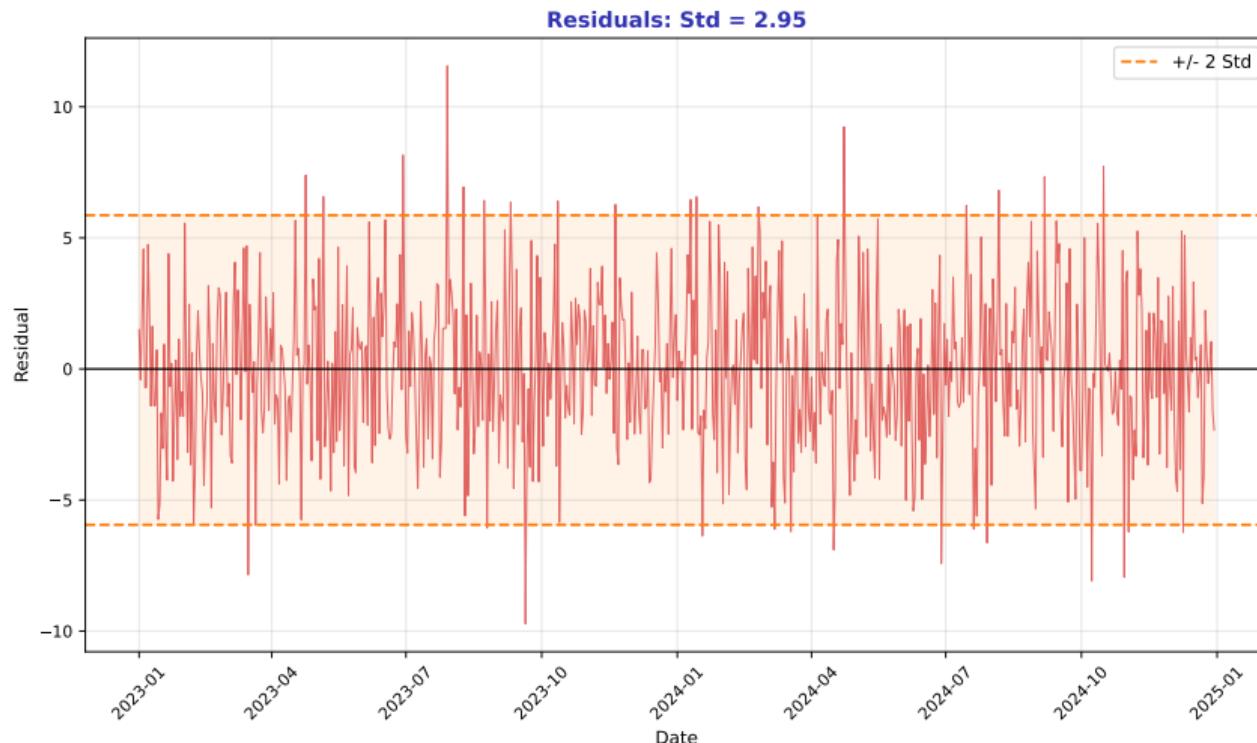
Identifying the long-term trend

## Seasonal Pattern



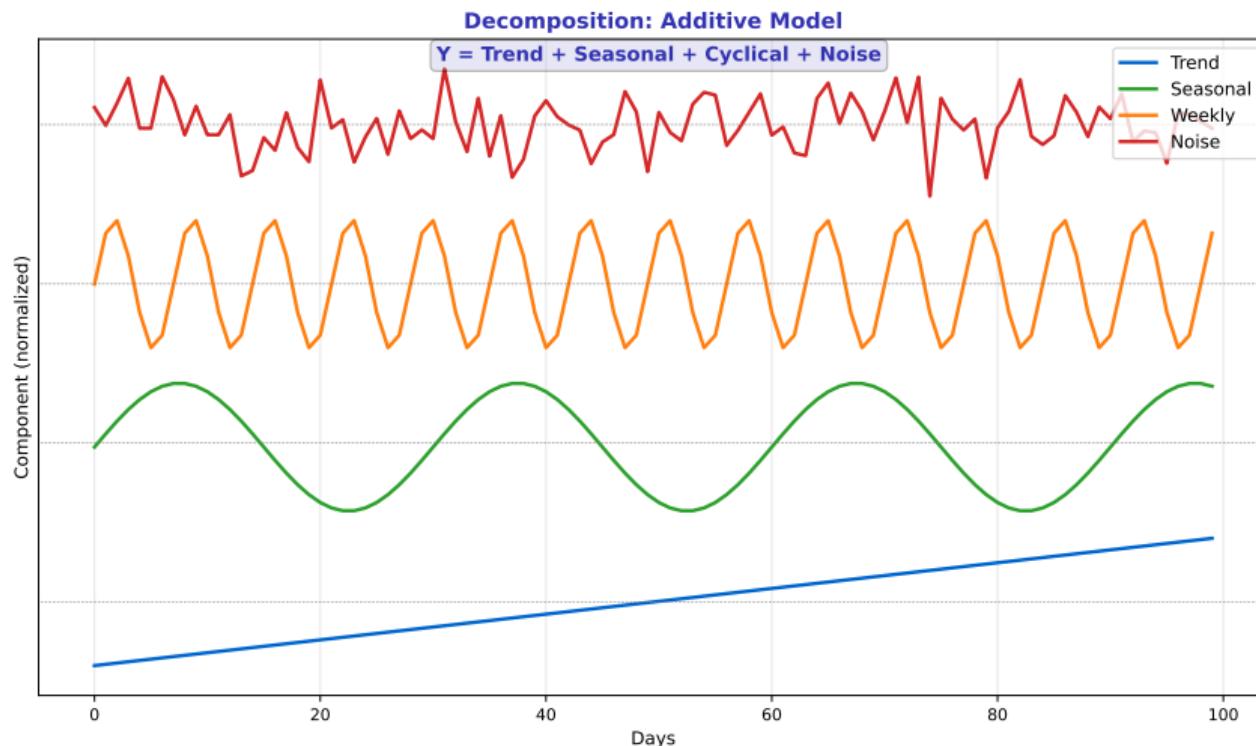
Recurring patterns within periods

## Residuals



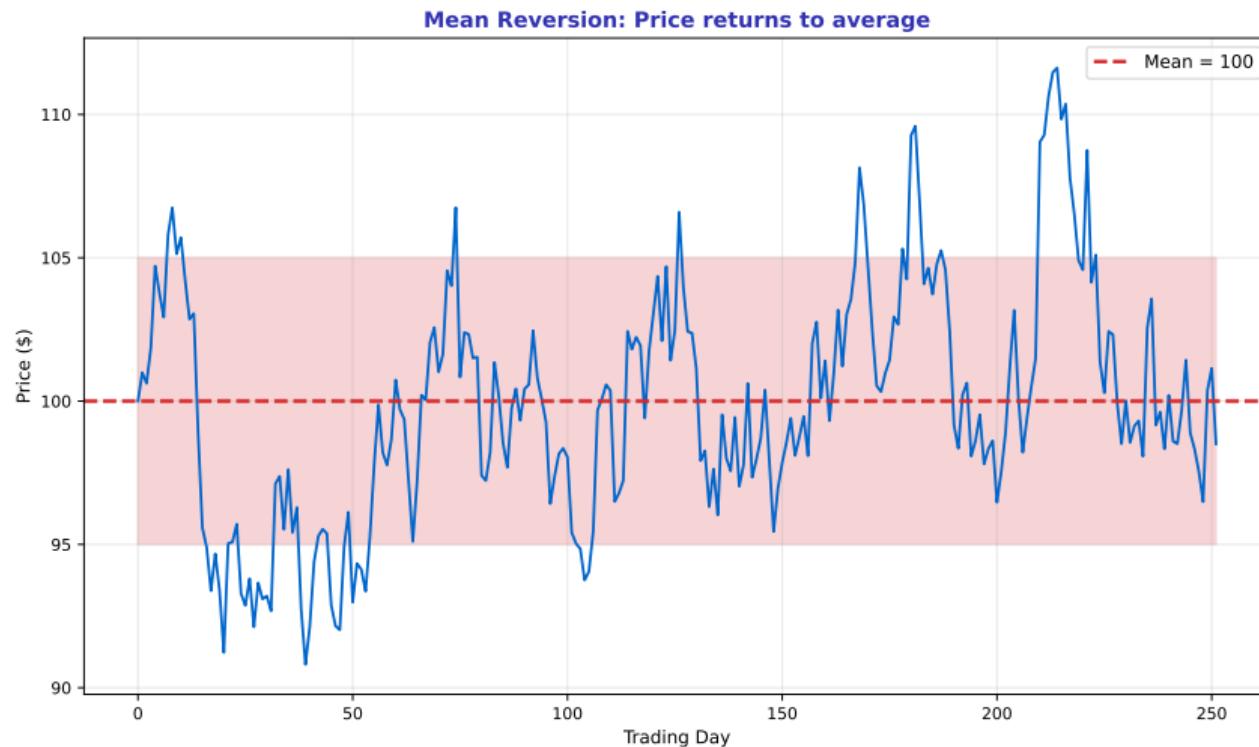
**Random component after decomposition**

## Full Decomposition



Trend + Seasonal + Residual

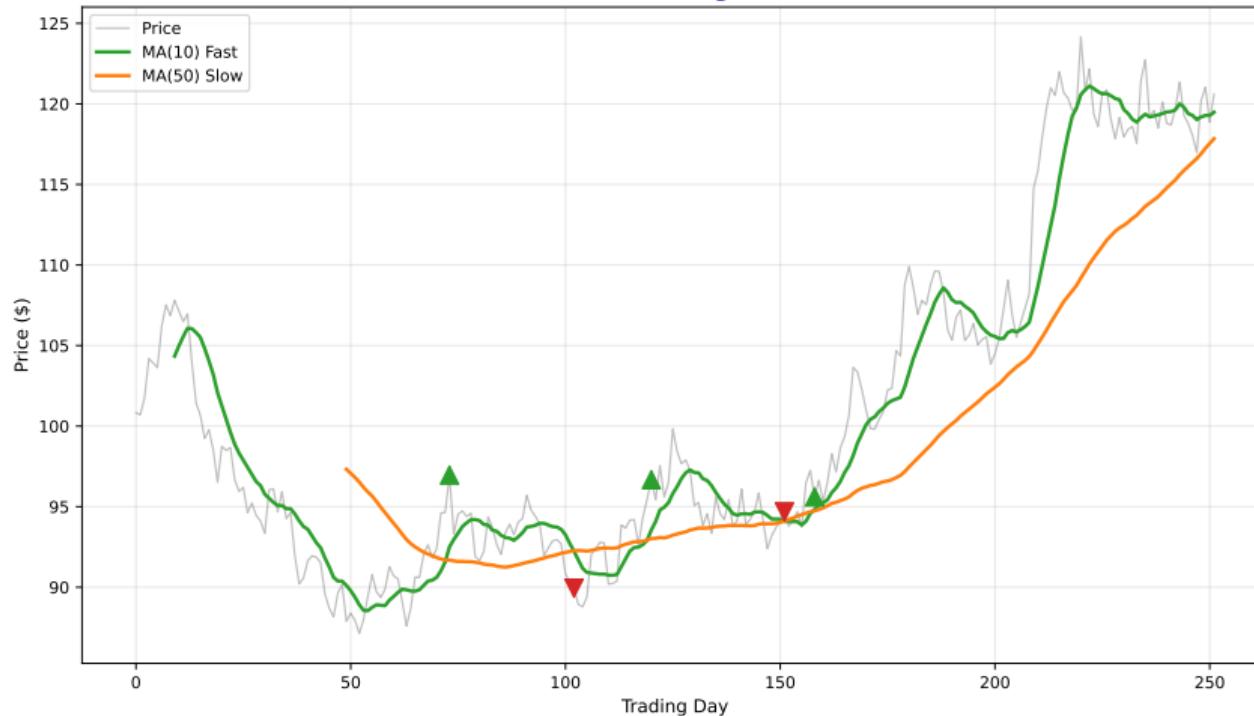
## Mean Reversion



Price returning to average

# Momentum

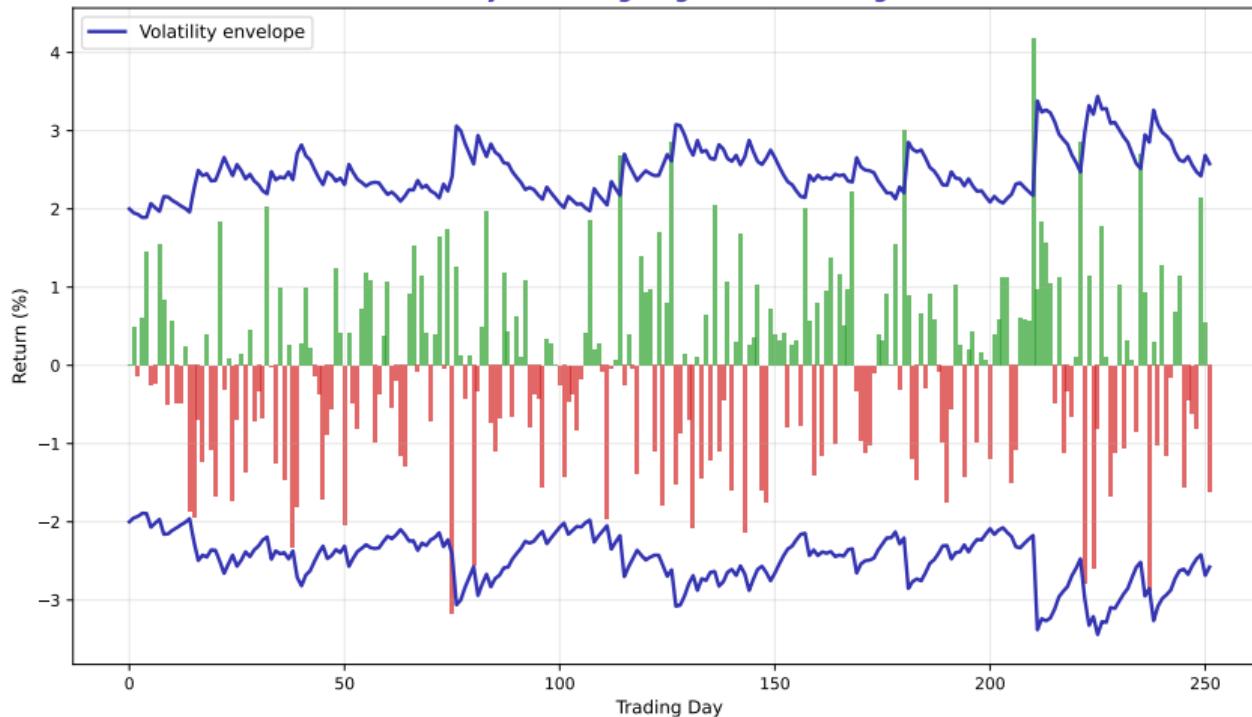
Momentum: Trend following with MA crossover



## Trend-following patterns

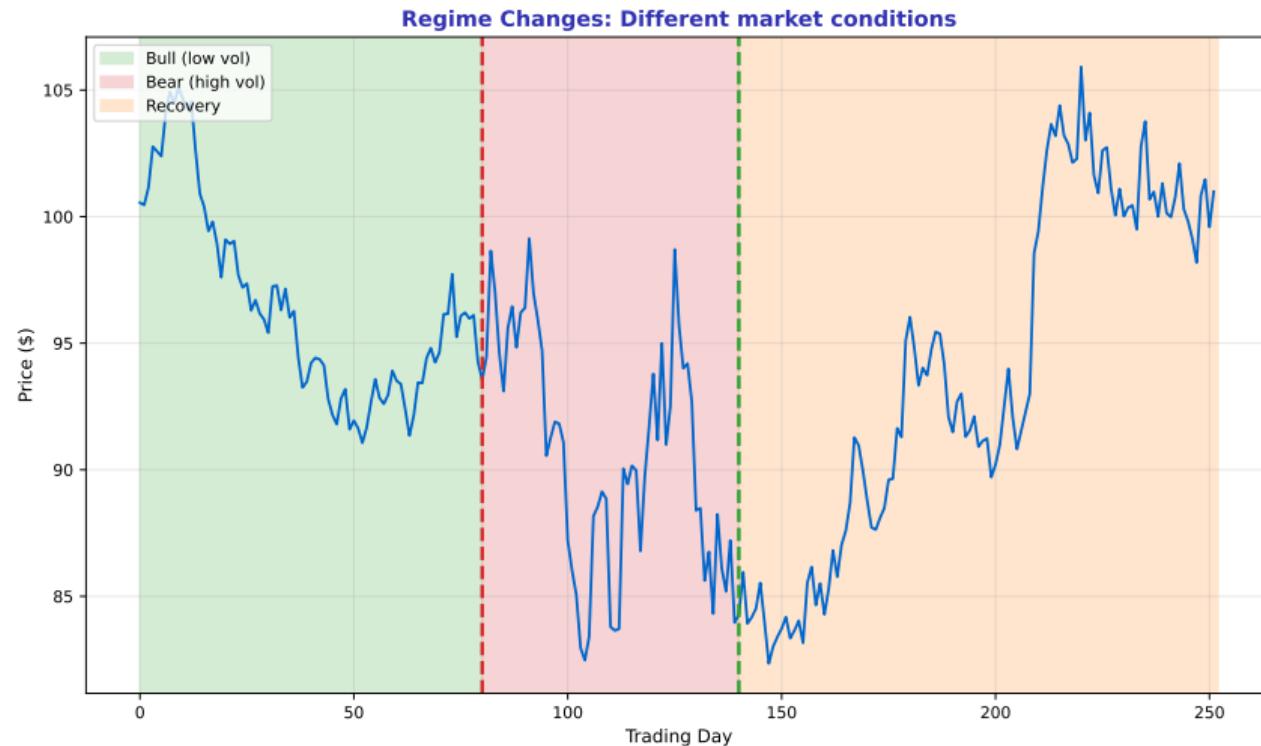
# Volatility Clustering

Volatility Clustering: High vol follows high vol



High volatility follows high volatility

# Regime Changes



Different market conditions

## Lesson Summary

### Key Takeaways:

- DateTime index for time series data
- Resampling changes data frequency
- Rolling windows for moving statistics
- shift() creates lags, pct\_change() computes returns
- Recognize patterns: trend, seasonality, mean reversion, momentum

**Practice:** Apply these concepts to the stock price dataset.

## Lesson 13: Descriptive Statistics

Data Science with Python – BSc Course

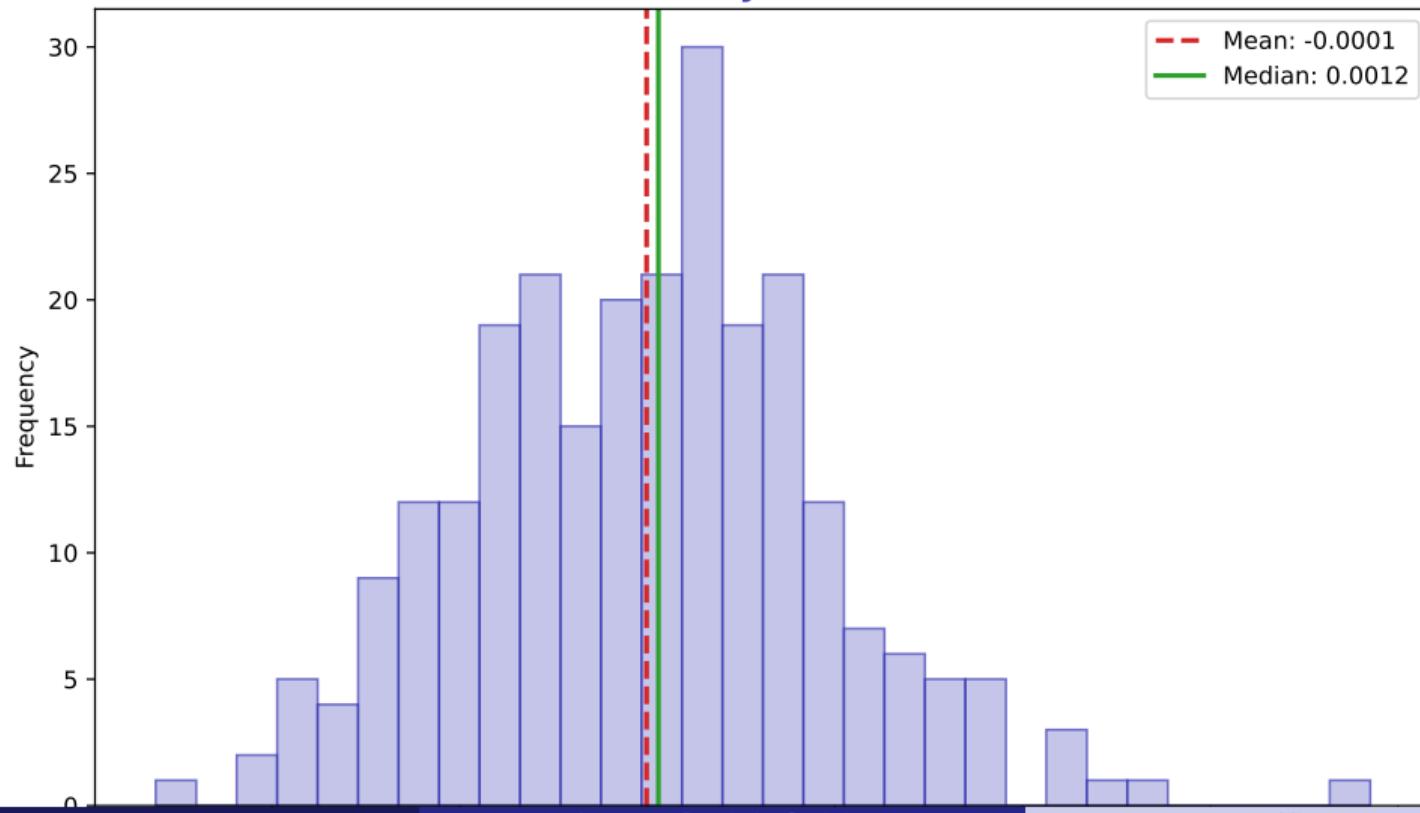
45 Minutes

## After this lesson, you will be able to:

- Calculate mean, median, mode
- Measure dispersion (std, variance, range)
- Interpret quartiles and percentiles
- Analyze skewness and kurtosis

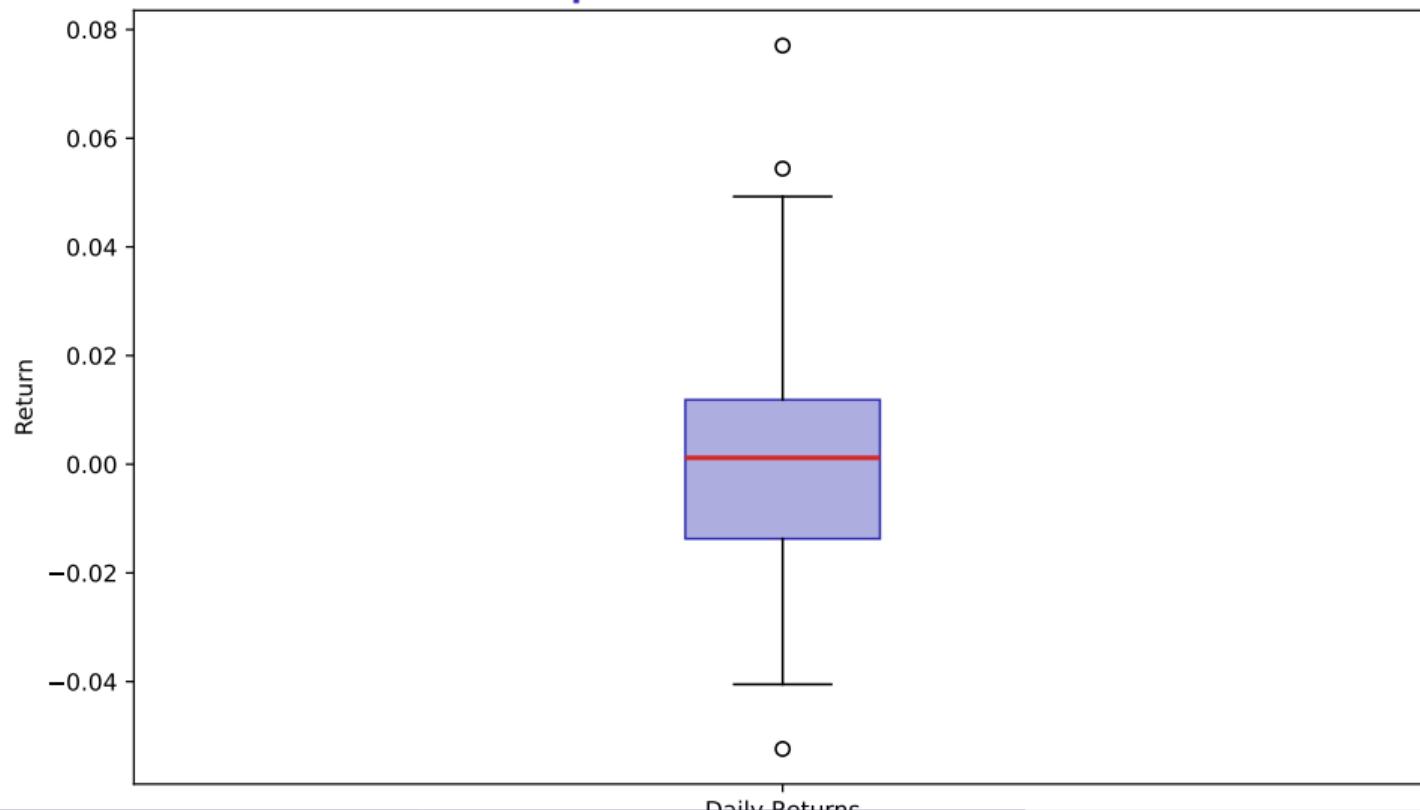
Finance application: Statistical analysis of market data

## Central Tendency: Mean vs Median



## Dispersion

**Dispersion: Std Dev = 0.0193**



## Quartiles and Percentiles

Minimum: -0.0524

Q1 (25th): -0.0137

Median (50th): 0.0012

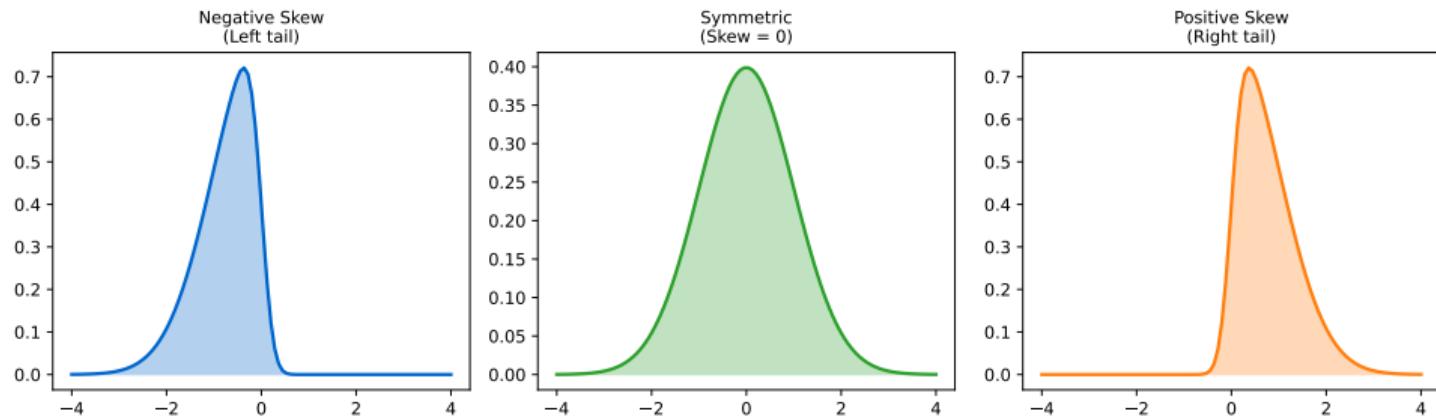
Q3 (75th): 0.0119

Maximum: 0.0771

IQR: 0.0256

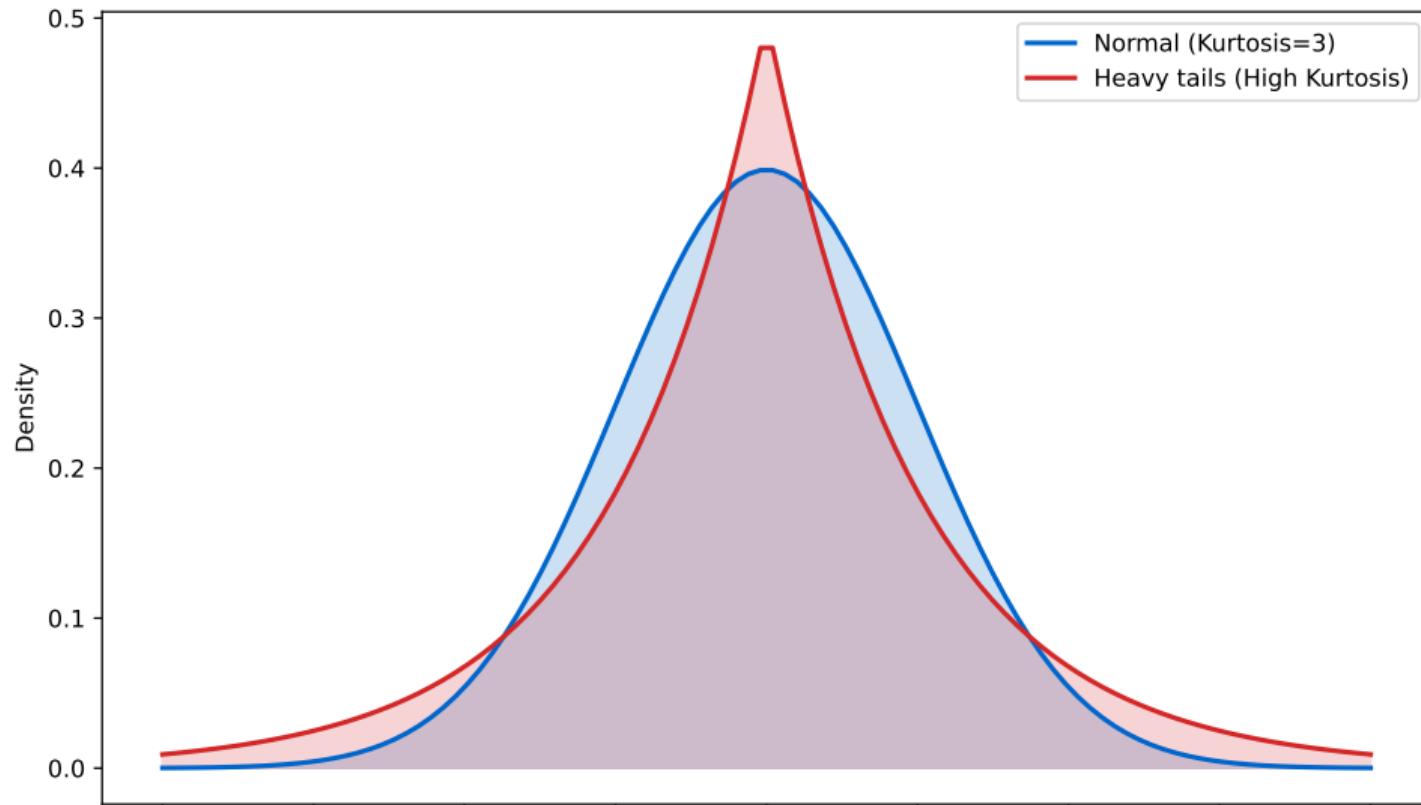
# Skewness

## Skewness in Return Distributions



Statistical foundation for data-driven decisions

### Kurtosis: Tail Thickness



## Summary Table

### df.describe() Output

Statistic	Value
Count	252
Mean	-0.0001
Std	0.0193
Min	-0.0524
25%	-0.0137
50%	0.0012
75%	0.0119
Max	0.0771

Statistical foundation for data-driven decisions

## Key Finance Statistics

**Annualized Return**

mean \* 252

**-1.90%****Annualized Volatility**

std \* sqrt(252)

**30.65%****Sharpe Ratio**

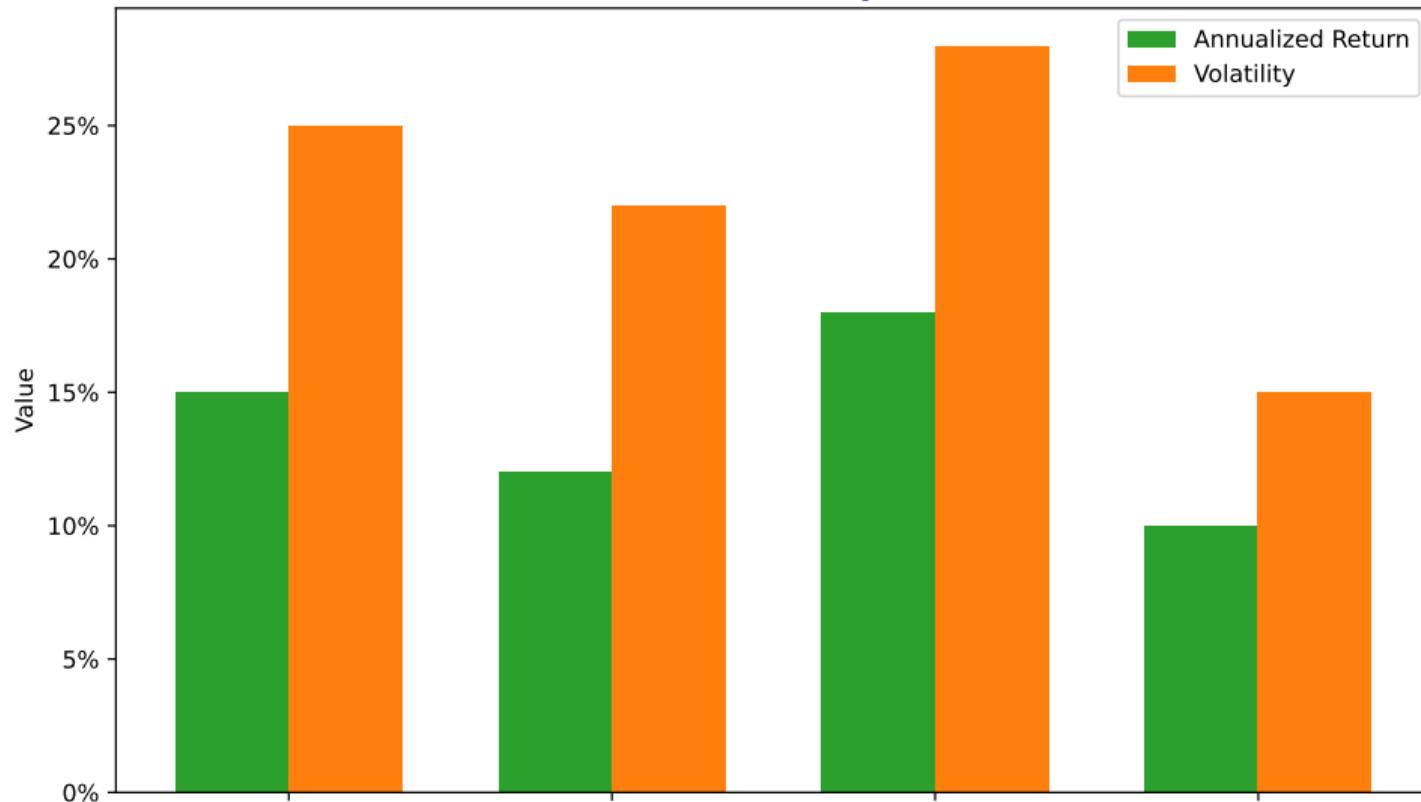
(ret - rf) / vol

**-0.13****Max Drawdown**

Largest peak-to-trough

**-12.5%**

## Risk-Return Comparison



## Key Takeaways:

- Calculate mean, median, mode
- Measure dispersion (std, variance, range)
- Interpret quartiles and percentiles
- Analyze skewness and kurtosis

**Statistics + Visualization = Data Science foundation**

## Lesson 14: Probability Distributions

Data Science with Python – BSc Course

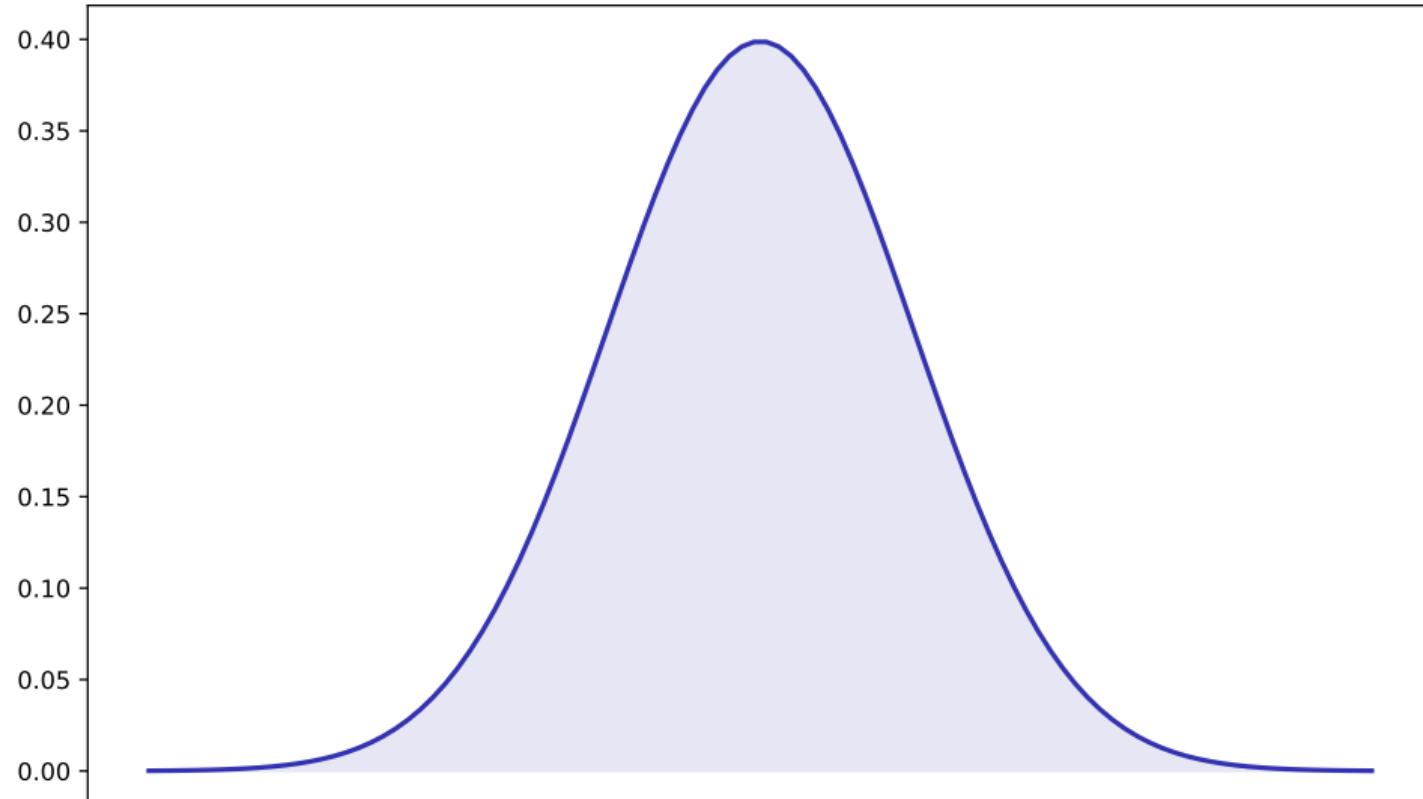
45 Minutes

## After this lesson, you will be able to:

- Understand normal distribution properties
- Apply distributions to stock returns
- Use QQ-plots for normality testing
- Recognize fat tails in finance

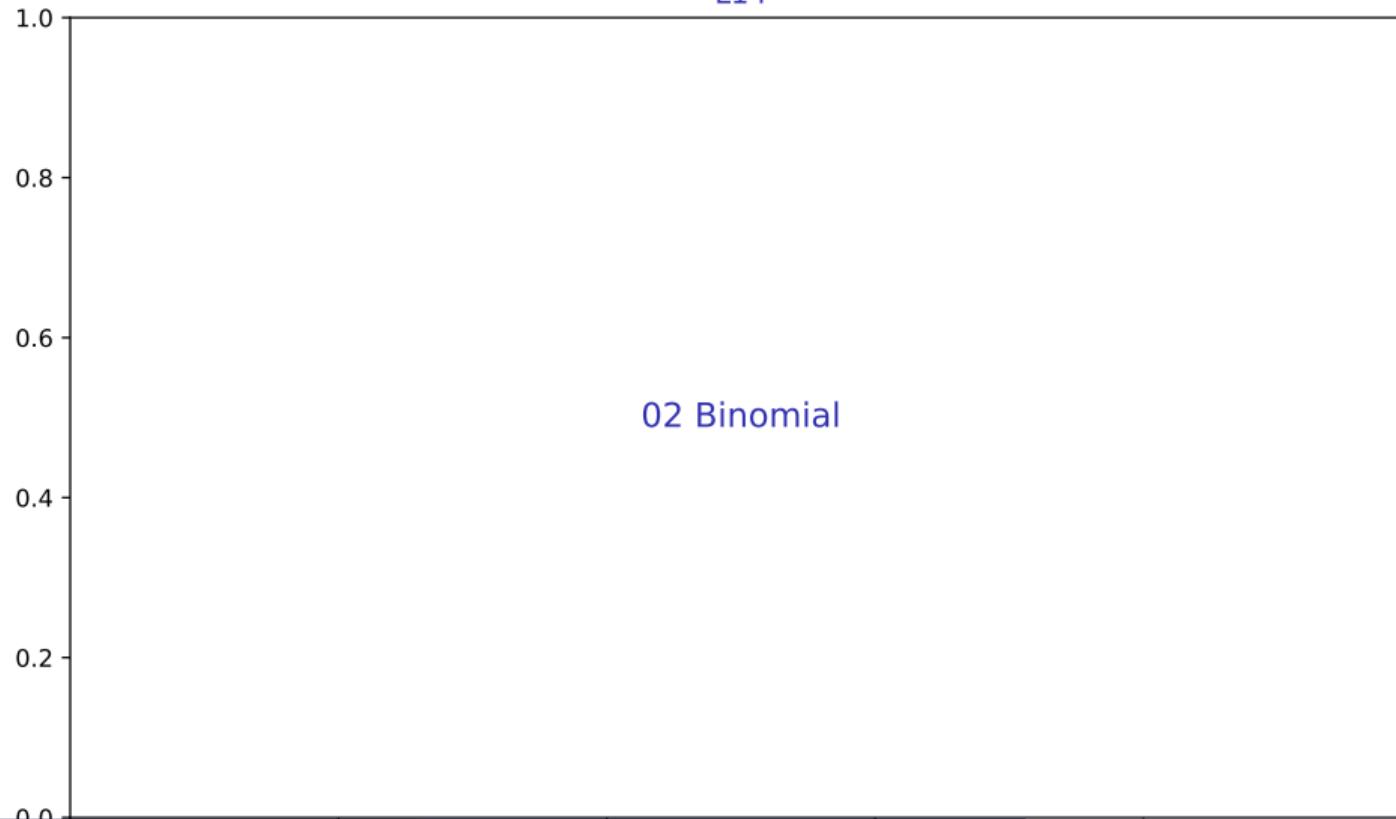
Finance application: Statistical analysis of market data

## 01 Normal Distribution



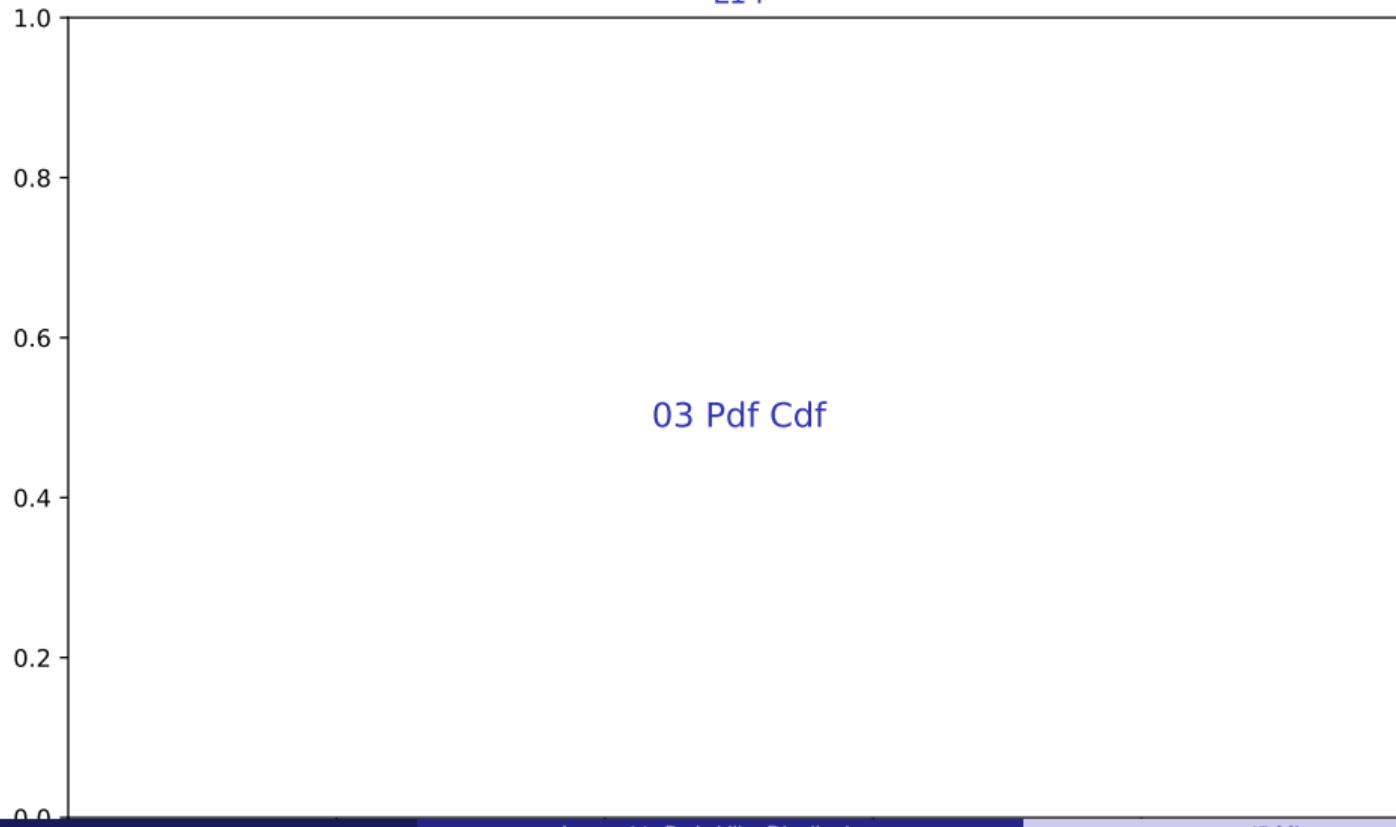
L14

02 Binomial



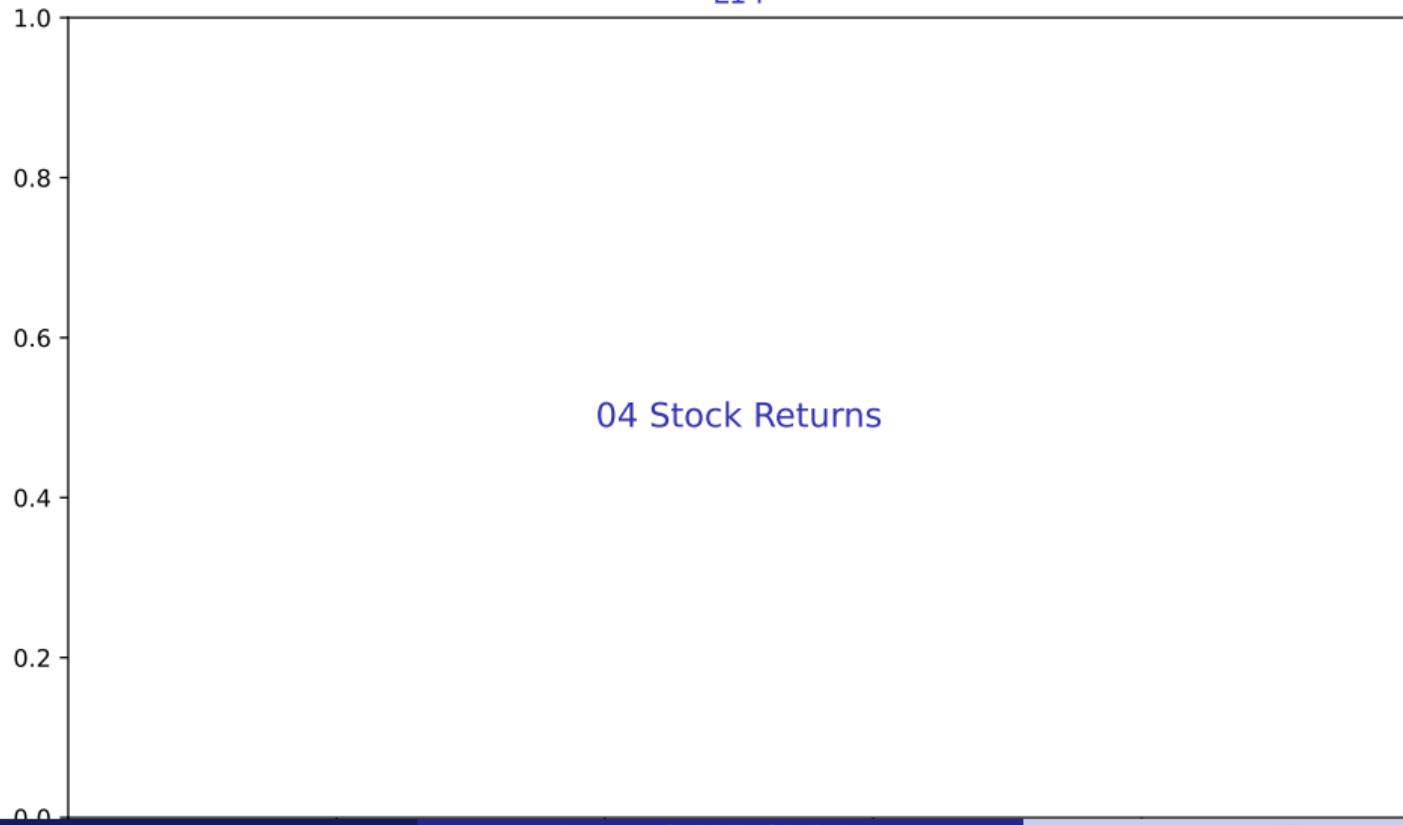
L14

03 Pdf Cdf



L14

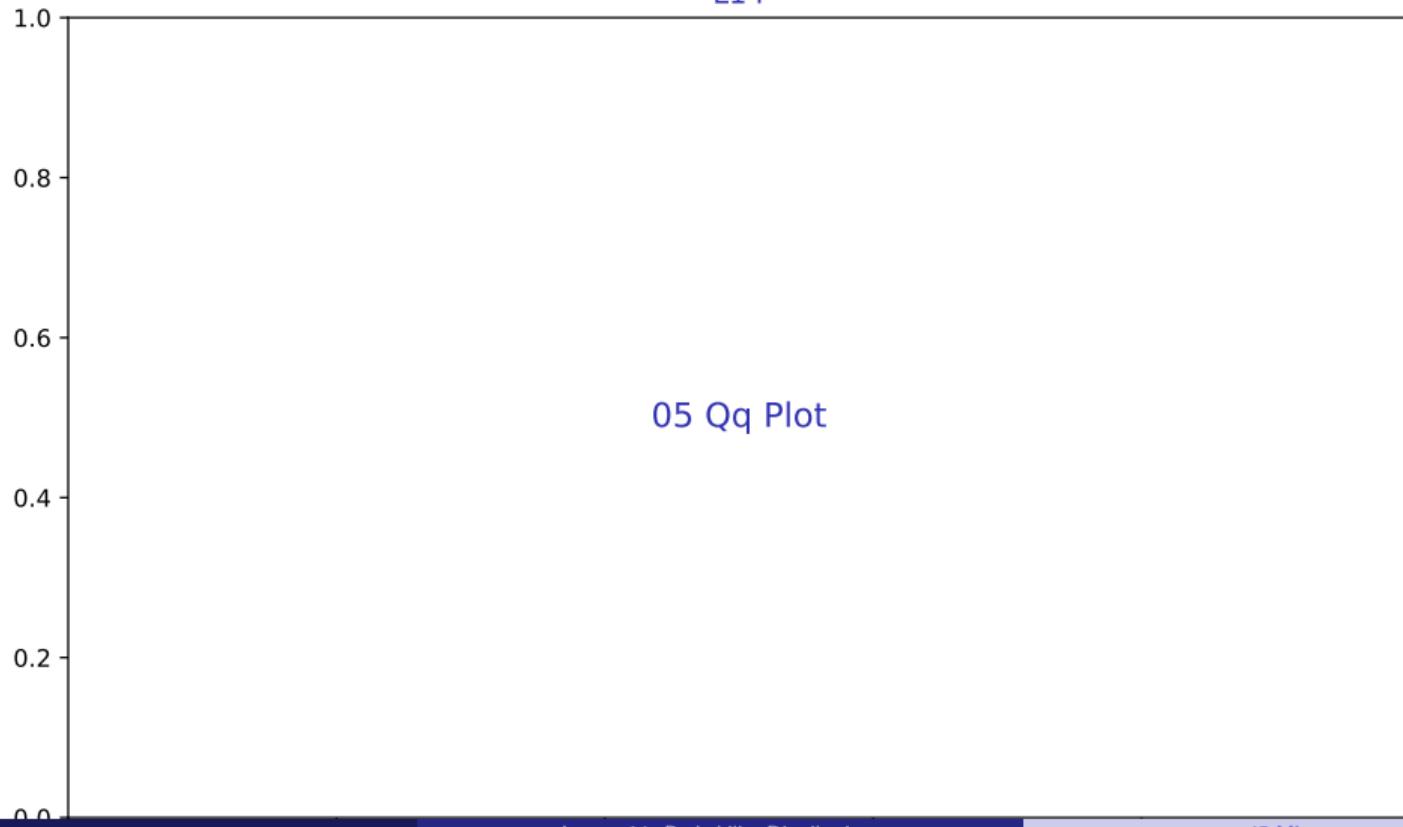
04 Stock Returns



## Qq Plot

L14

05 Qq Plot

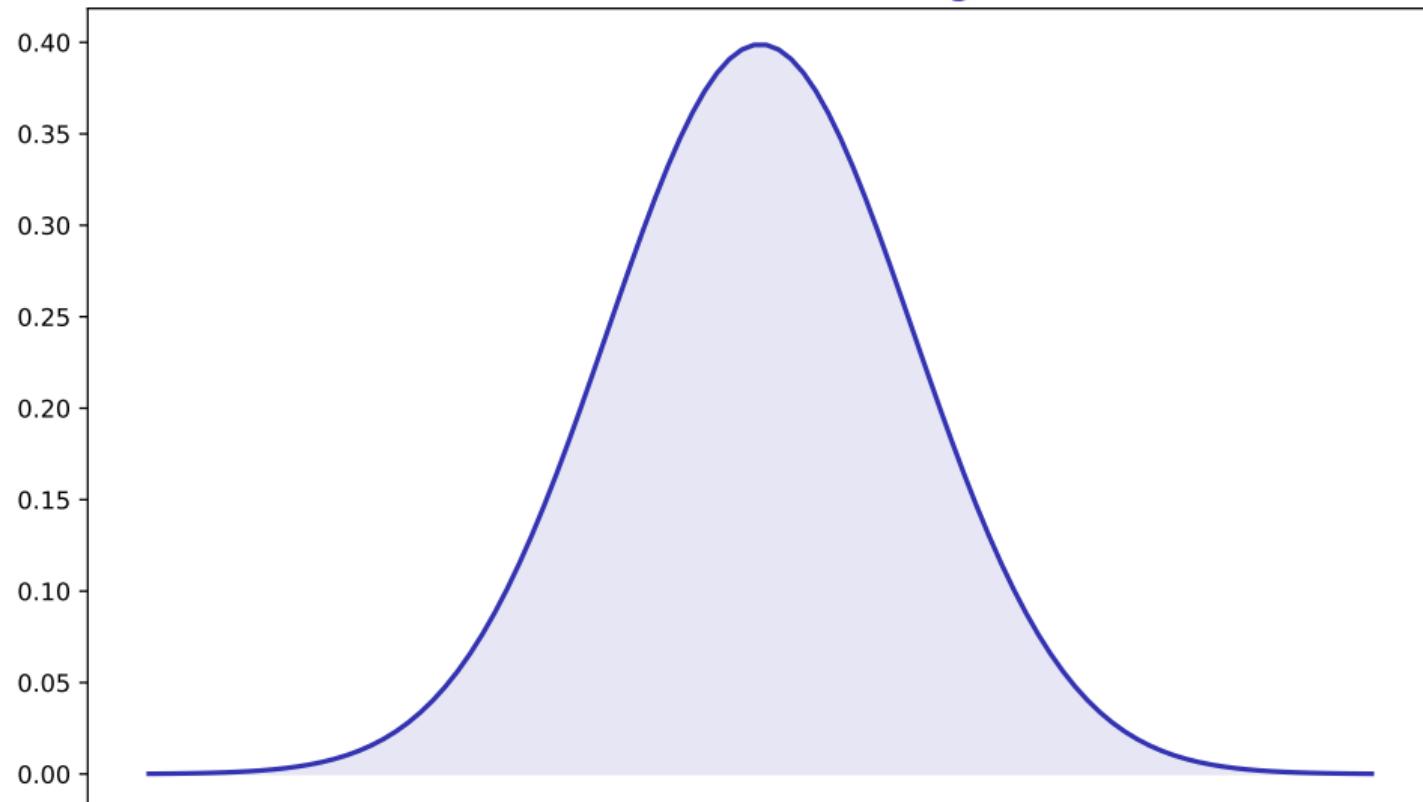


L14

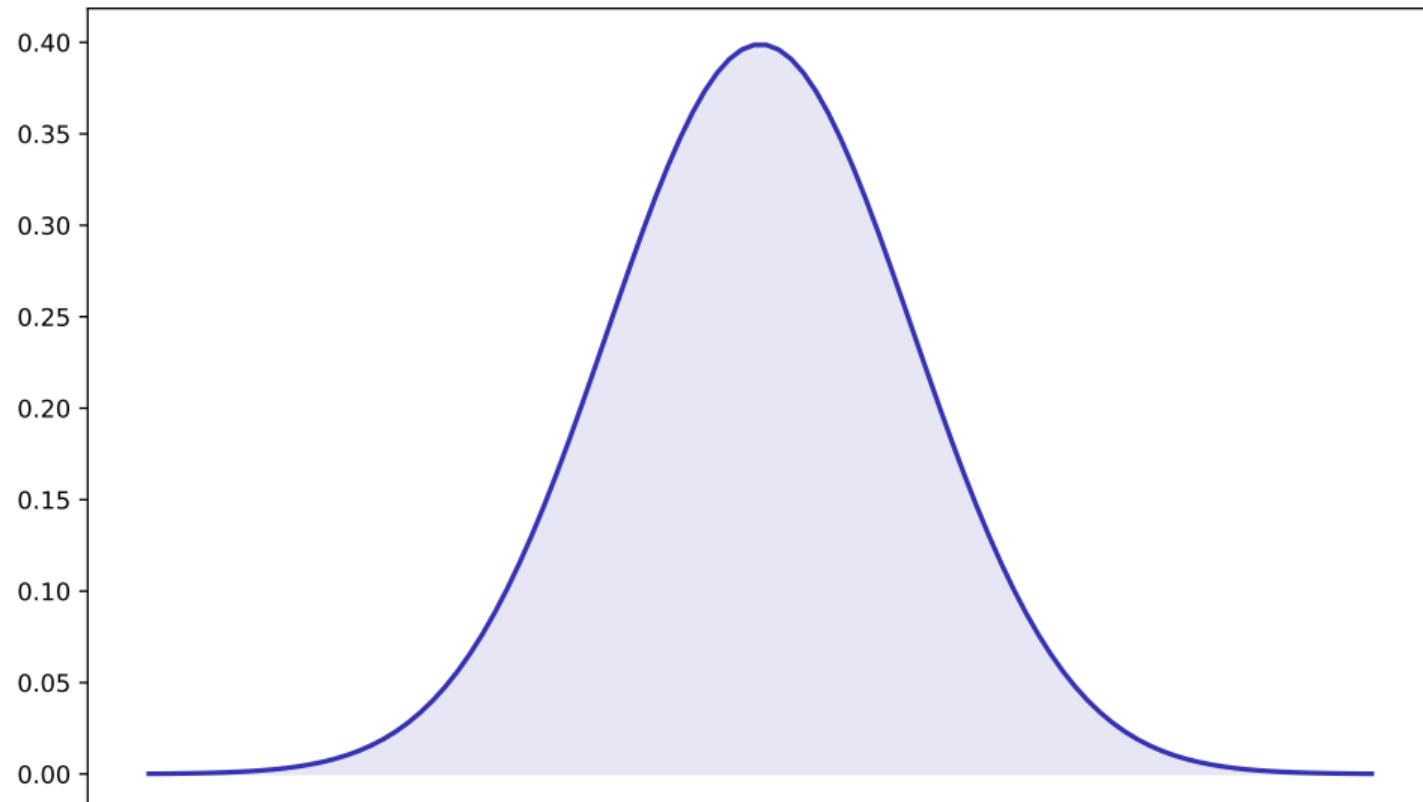
06 Fat Tails



## 07 Distribution Fitting



## 08 Finance Distributions



## Key Takeaways:

- Understand normal distribution properties
- Apply distributions to stock returns
- Use QQ-plots for normality testing
- Recognize fat tails in finance

Statistics + Visualization = Data Science foundation

## Lesson 15: Hypothesis Testing

Data Science with Python – BSc Course

45 Minutes

## After this lesson, you will be able to:

- Formulate null and alternative hypotheses
- Perform t-tests
- Interpret p-values correctly
- Construct confidence intervals

Finance application: Statistical analysis of market data

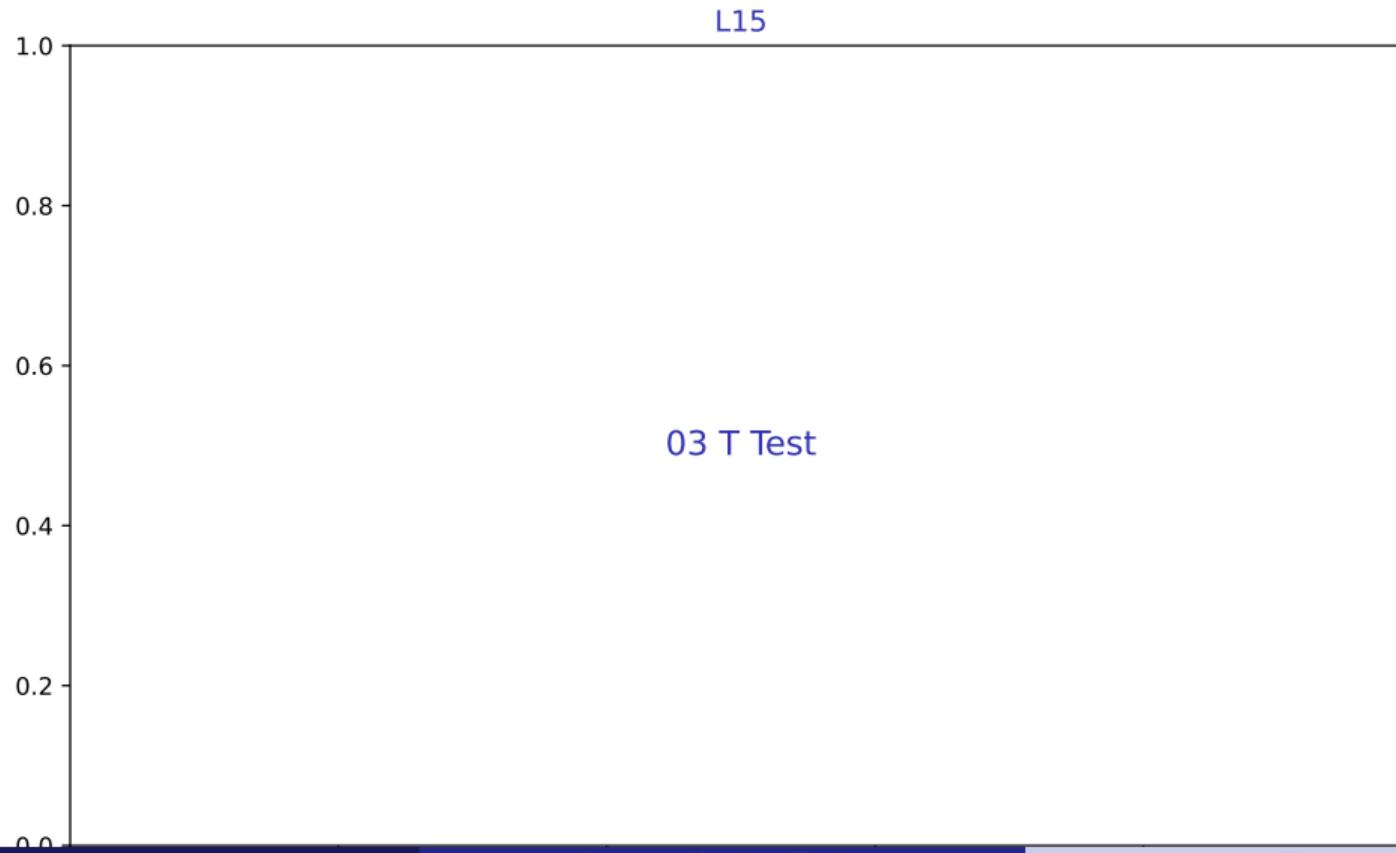
L15

01 Hypothesis Concept

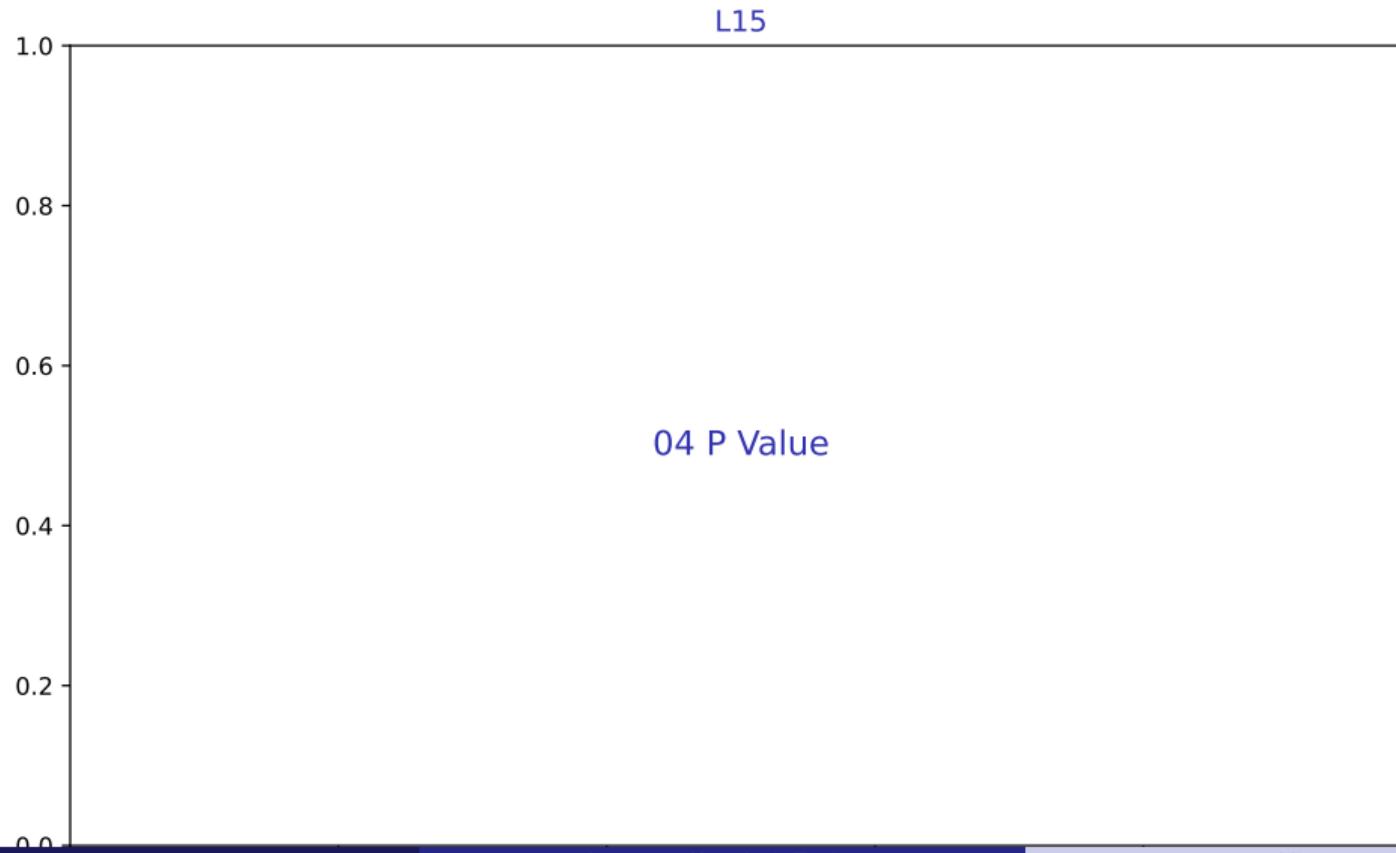
## Null Alternative



# T Test

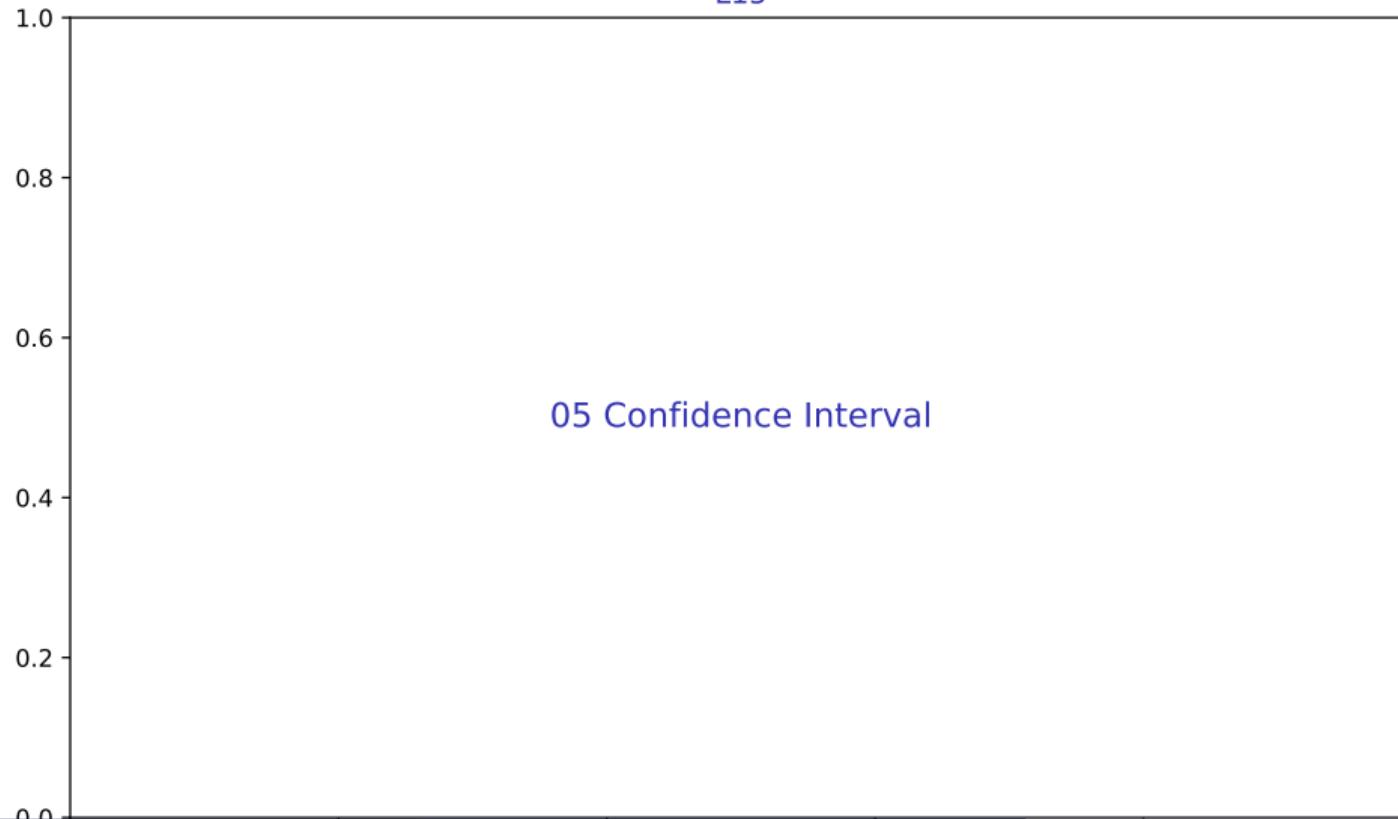


## P Value

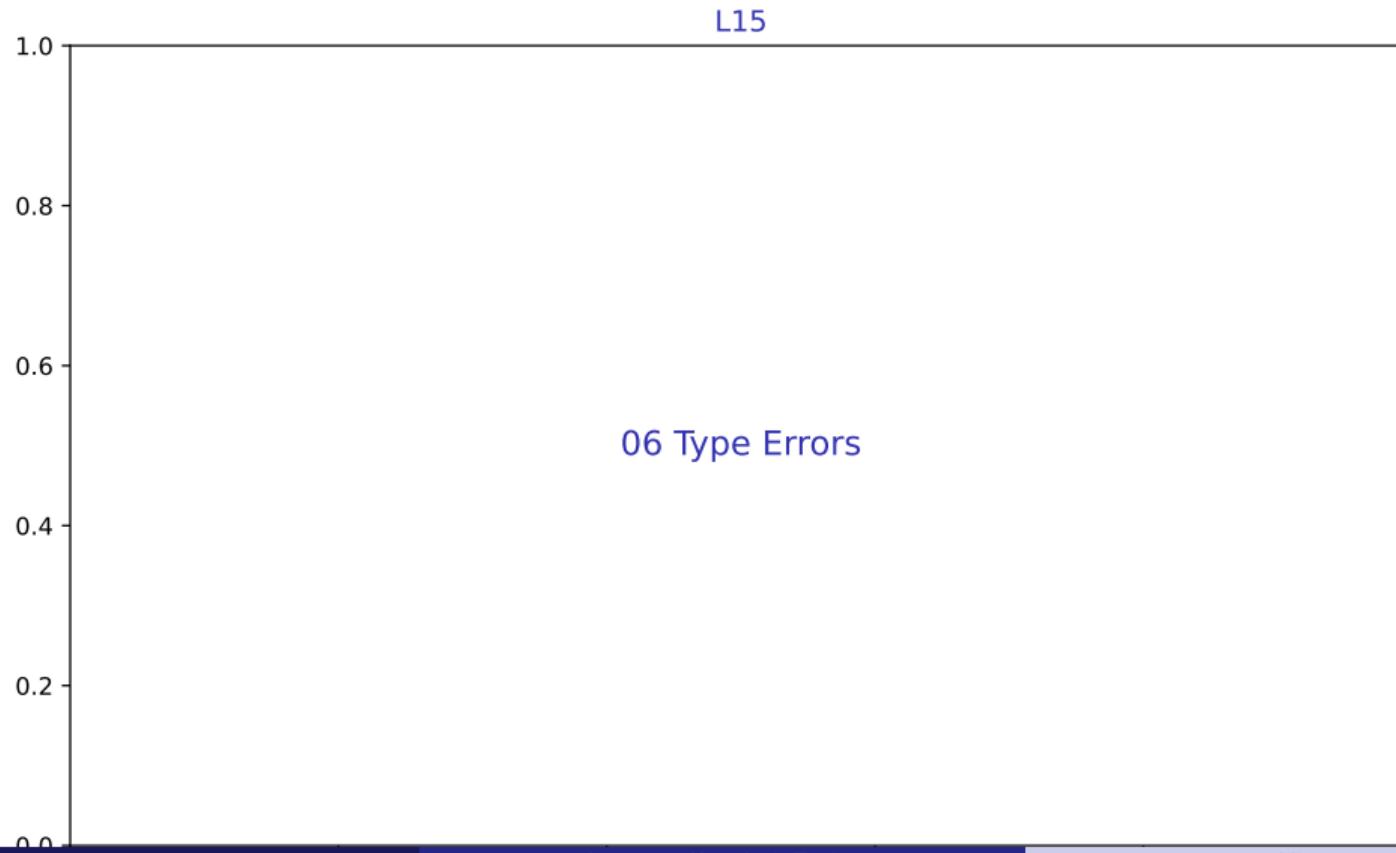


## Confidence Interval

L15



# Type Errors



# Ab Testing

L15

07 Ab Testing

L15

08 Finance Tests

## Key Takeaways:

- Formulate null and alternative hypotheses
- Perform t-tests
- Interpret p-values correctly
- Construct confidence intervals

**Statistics + Visualization = Data Science foundation**

## Lesson 16: Correlation Analysis

Data Science with Python – BSc Course

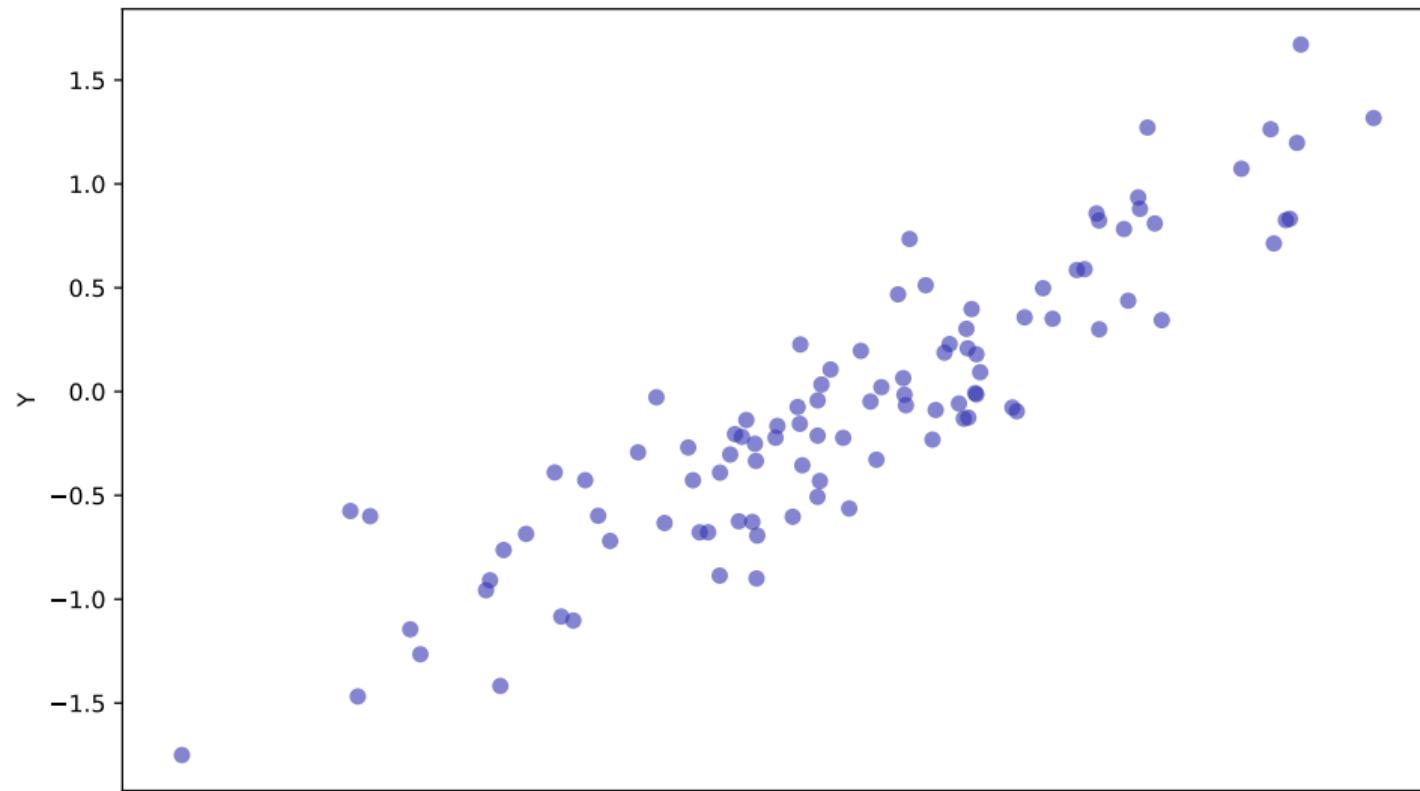
45 Minutes

**After this lesson, you will be able to:**

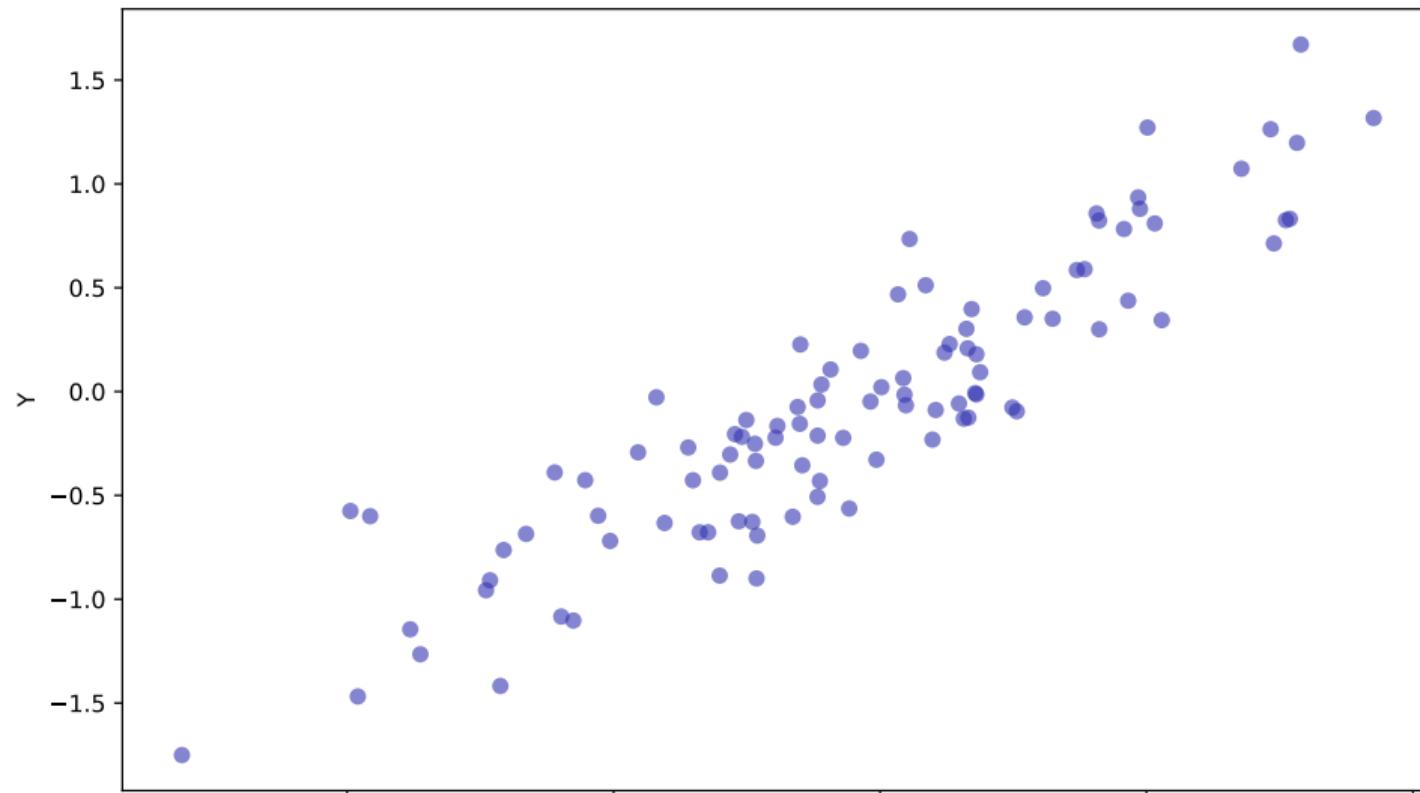
- Calculate Pearson and Spearman correlation
- Create correlation heatmaps
- Distinguish correlation from causation
- Apply to portfolio analysis

**Finance application: Statistical analysis of market data**

### 01 Correlation Scatter



### 02 Correlation Values



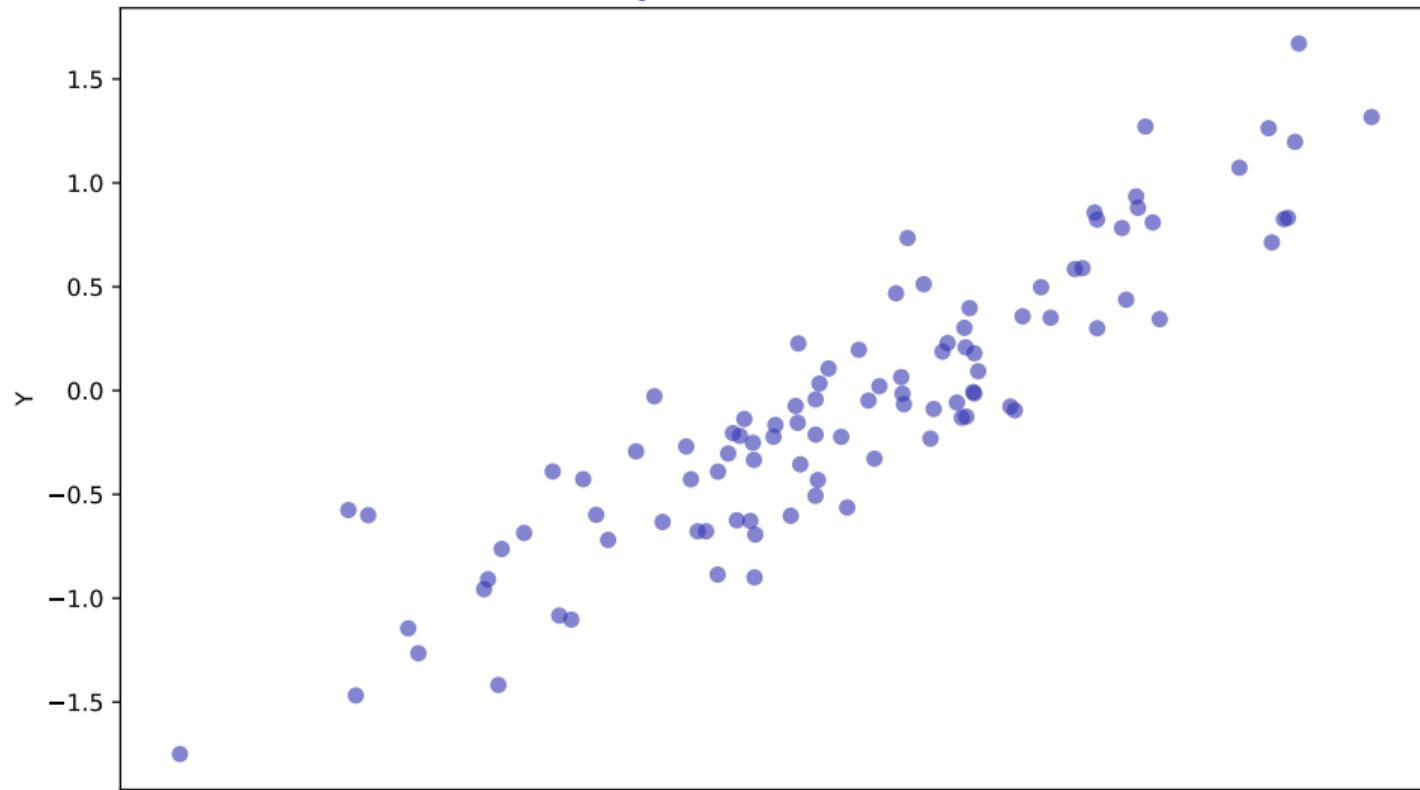
L16



## 04 Heatmap



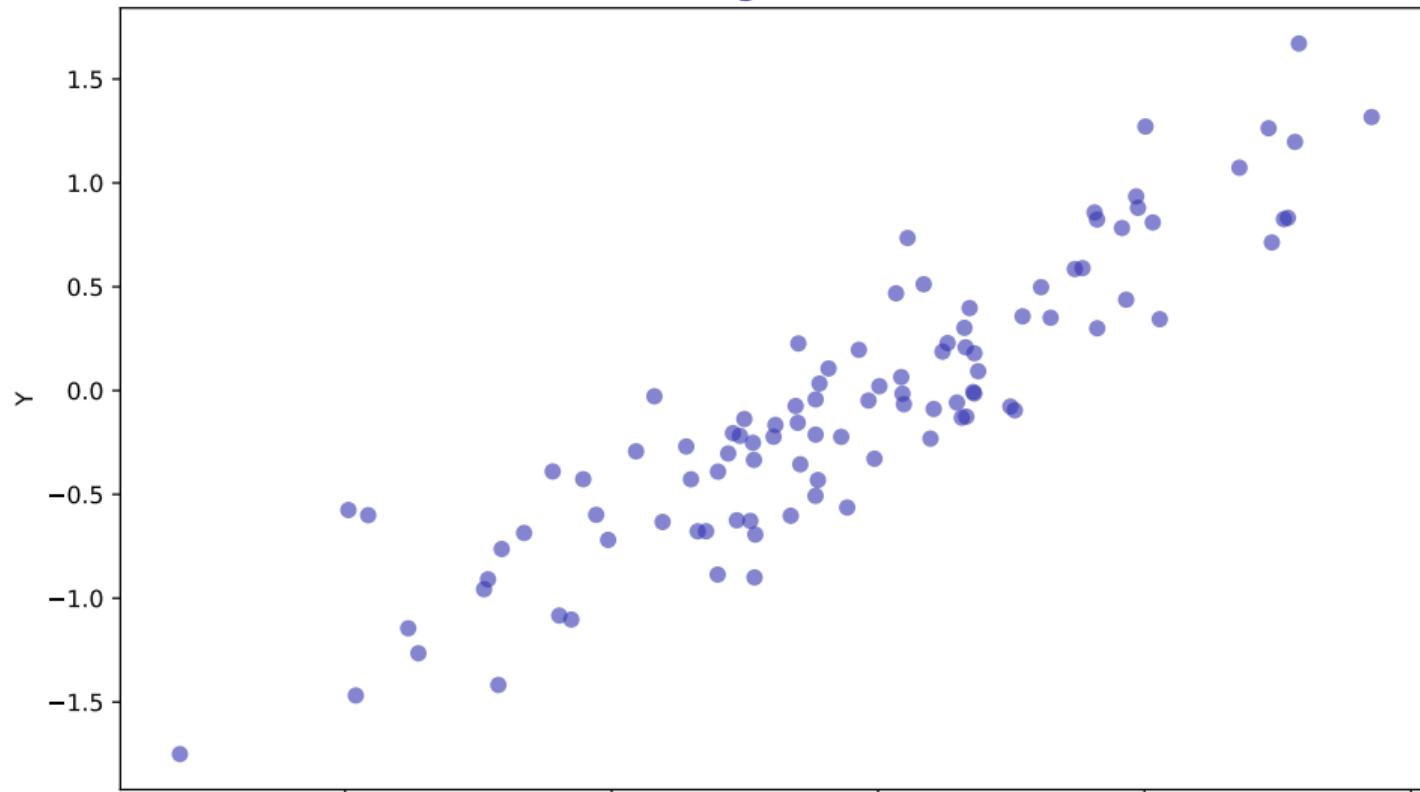
## 05 Spurious Correlation



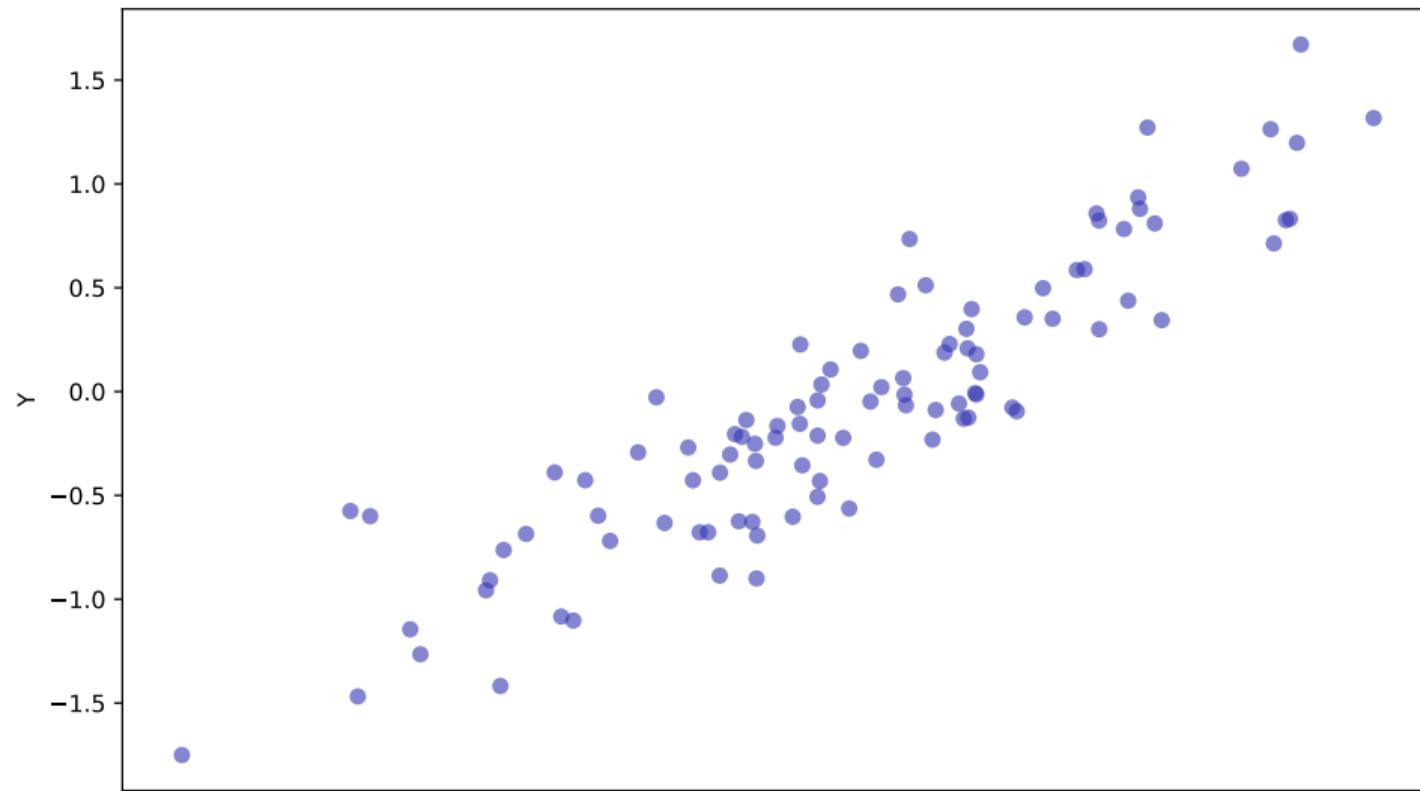
L16

06 Causation

### 07 Rolling Correlation



## 08 Portfolio Correlation



## Key Takeaways:

- Calculate Pearson and Spearman correlation
- Create correlation heatmaps
- Distinguish correlation from causation
- Apply to portfolio analysis

**Statistics + Visualization = Data Science foundation**

## Lesson 17: Matplotlib Basics

Data Science with Python – BSc Course

45 Minutes

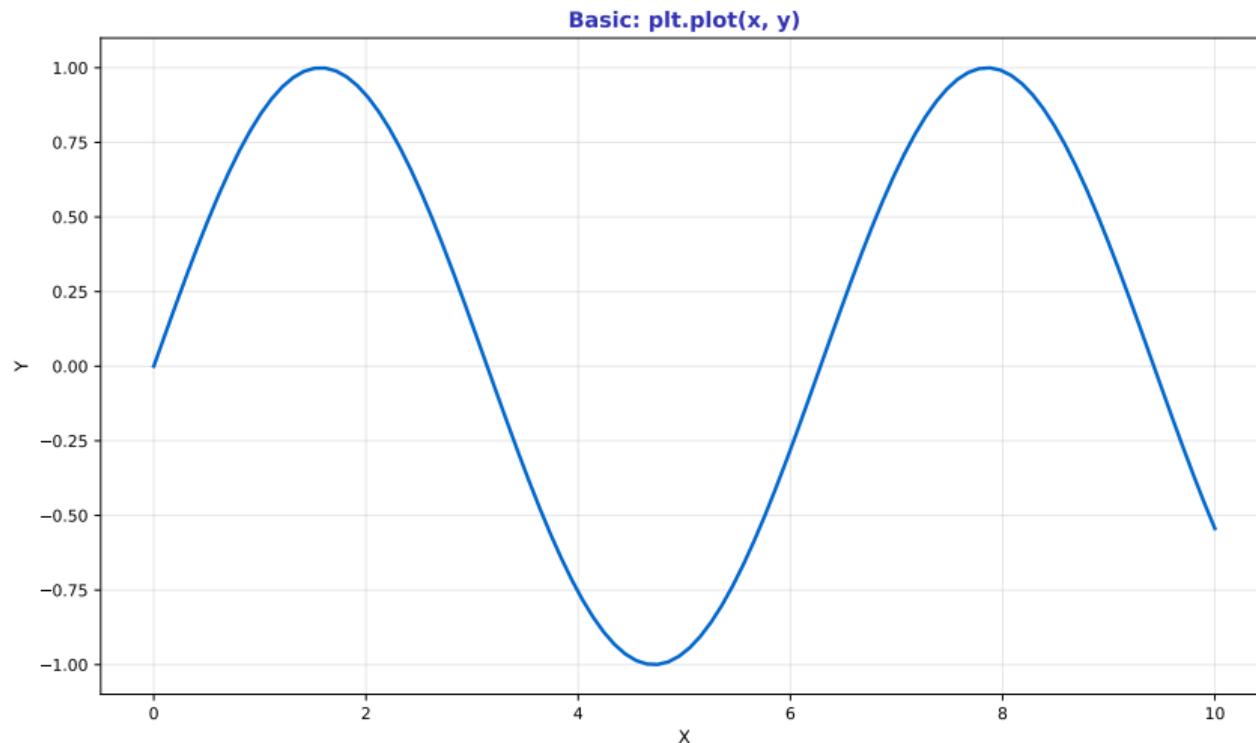
# Learning Objectives

**After this lesson, you will be able to:**

- Create line, bar, and scatter plots
- Customize colors, labels, legends
- Build multi-panel figures
- Add annotations and formatting

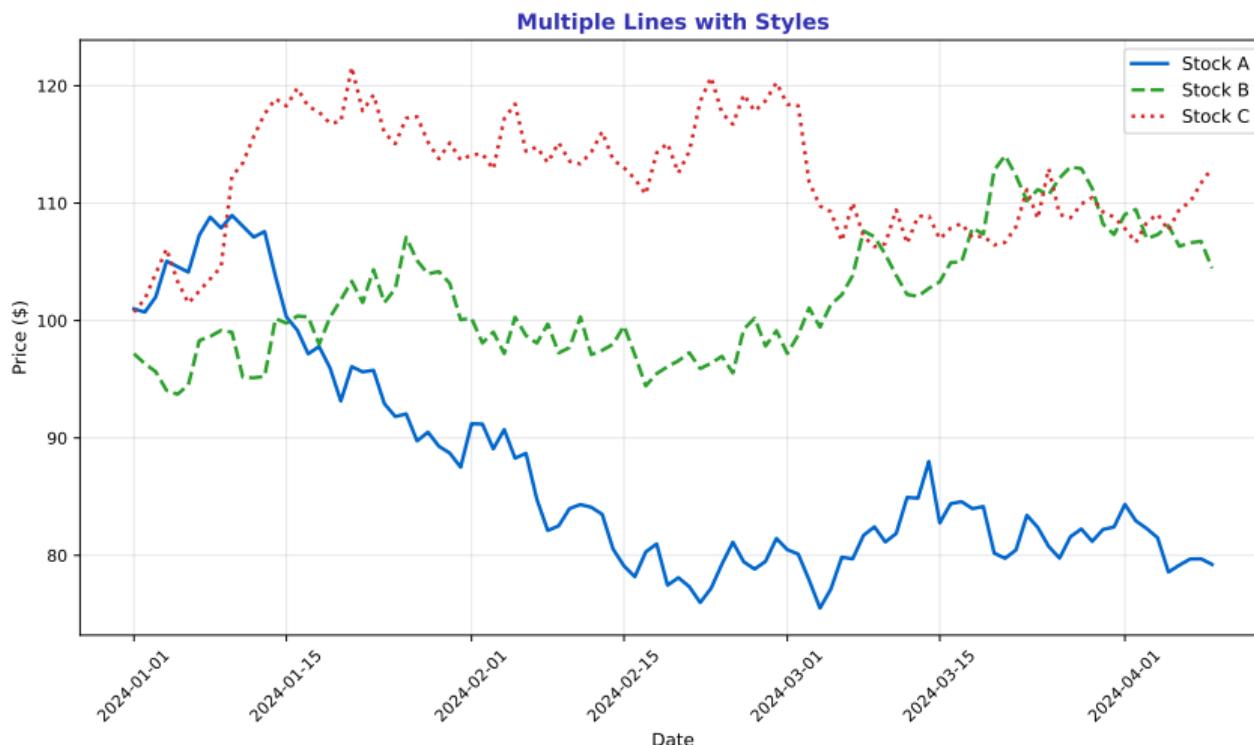
**Finance application: Statistical analysis of market data**

## Basic Line Plot



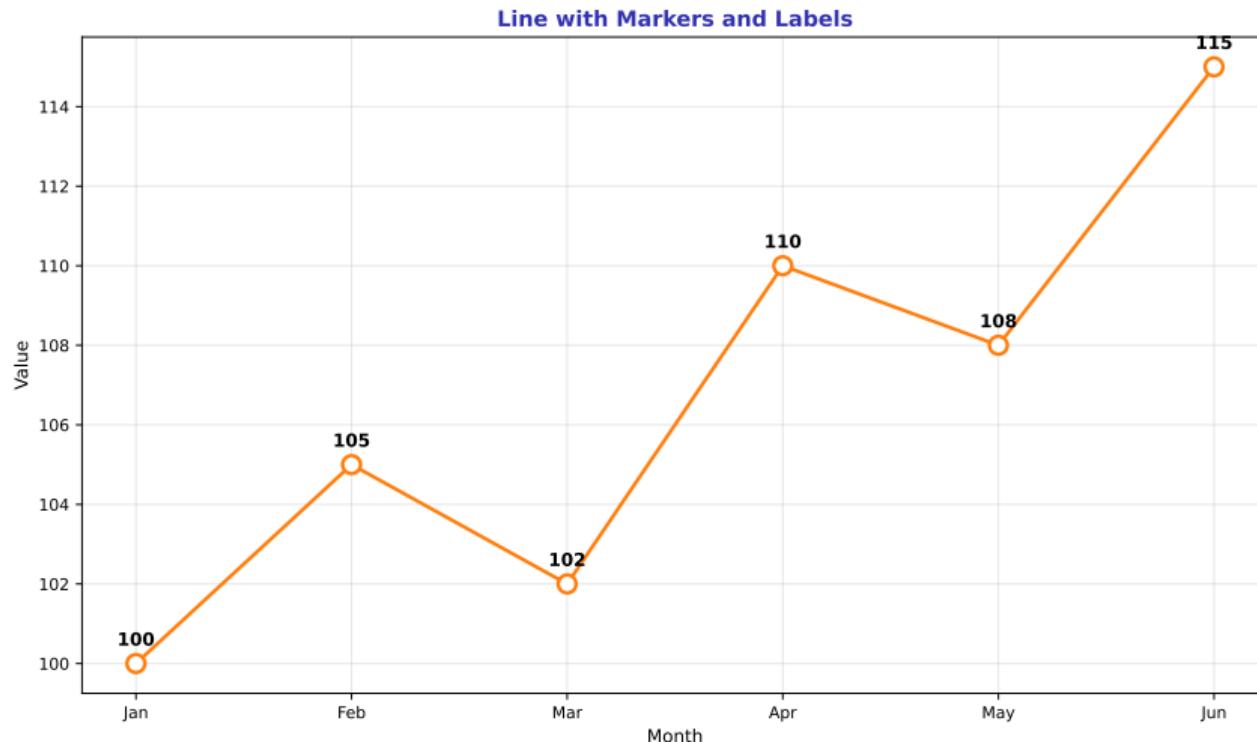
`plt.plot(x, y)` for simple line charts

## Multiple Lines



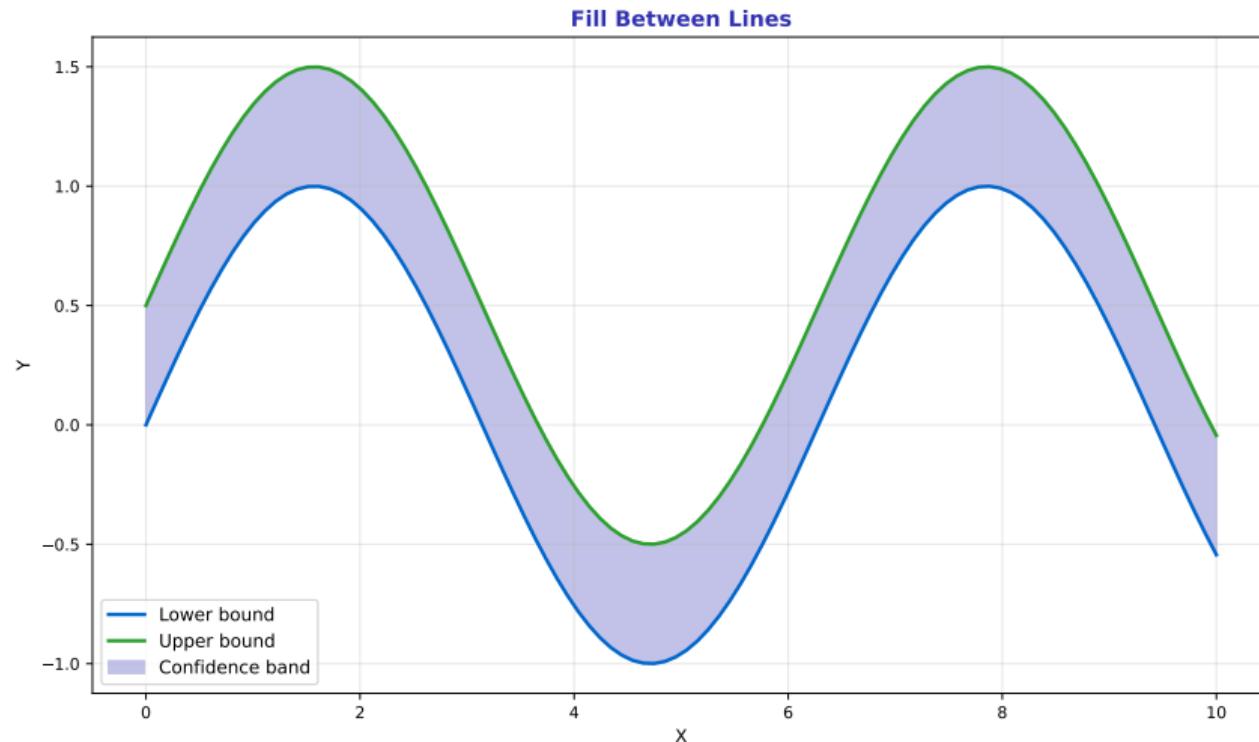
Comparing multiple time series

## Line with Markers



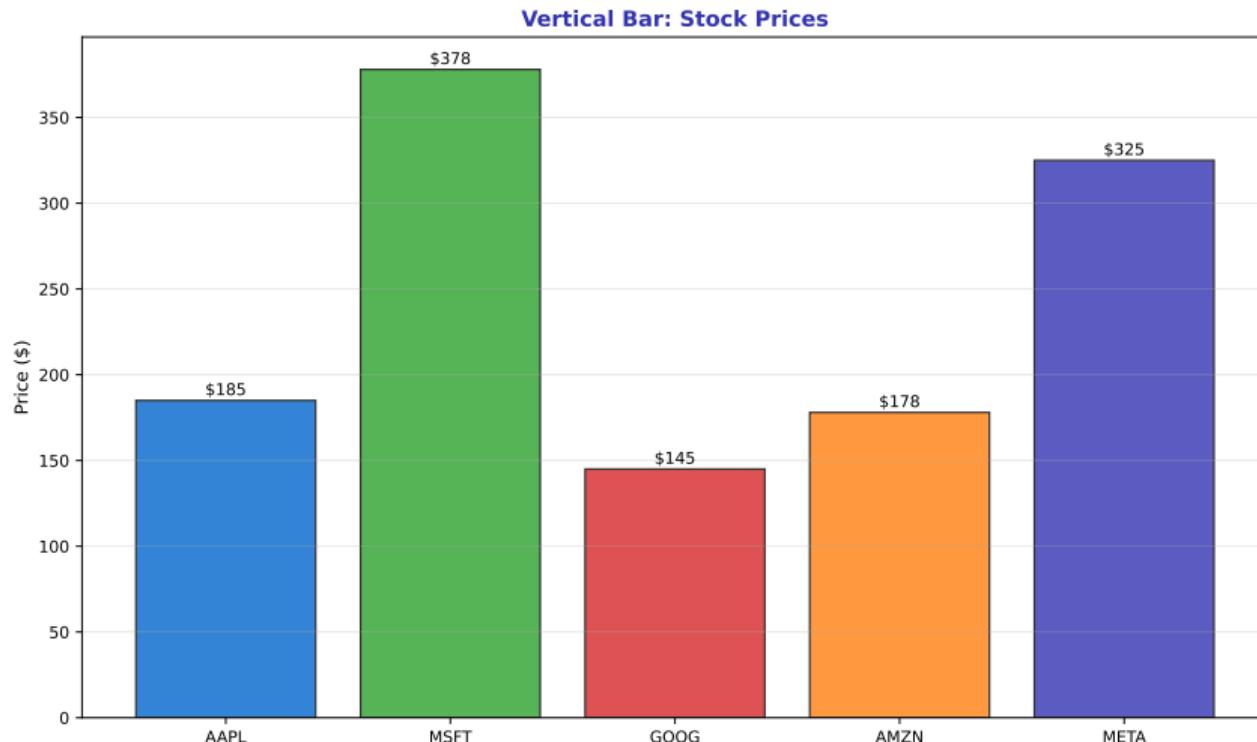
Adding data point markers

## Fill Between



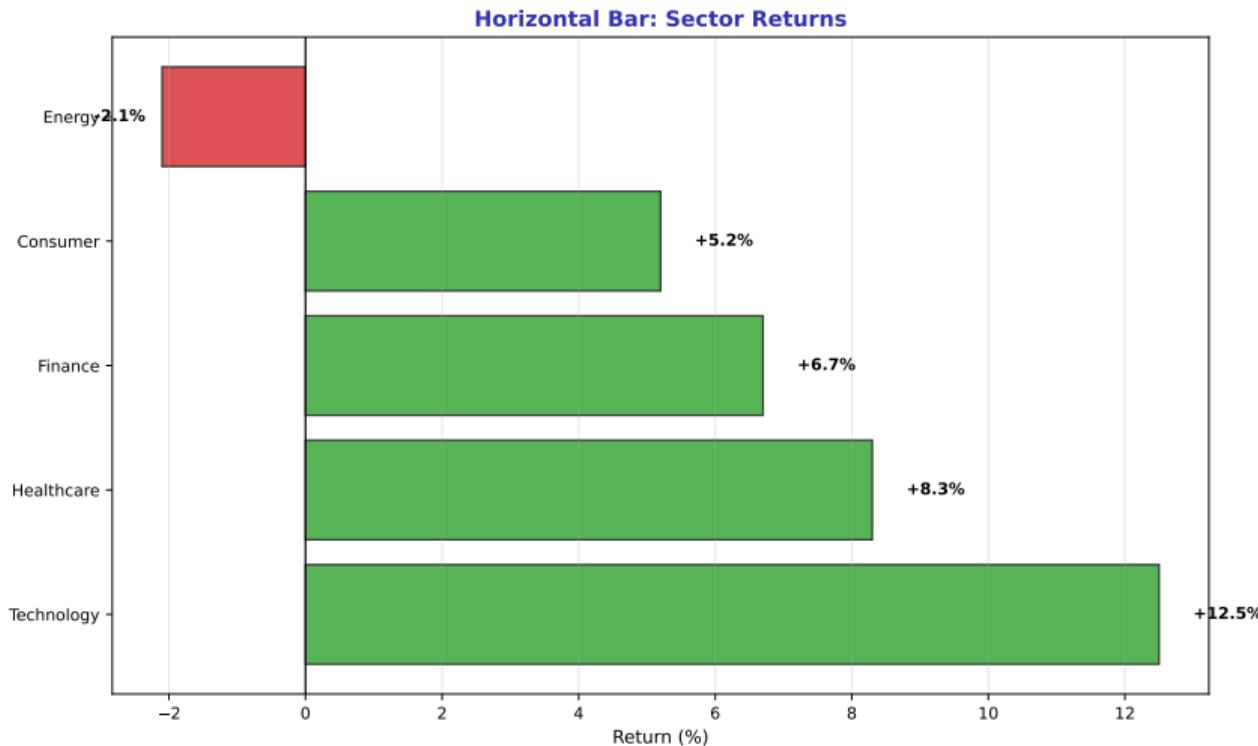
**Shading area between lines**

## Vertical Bar



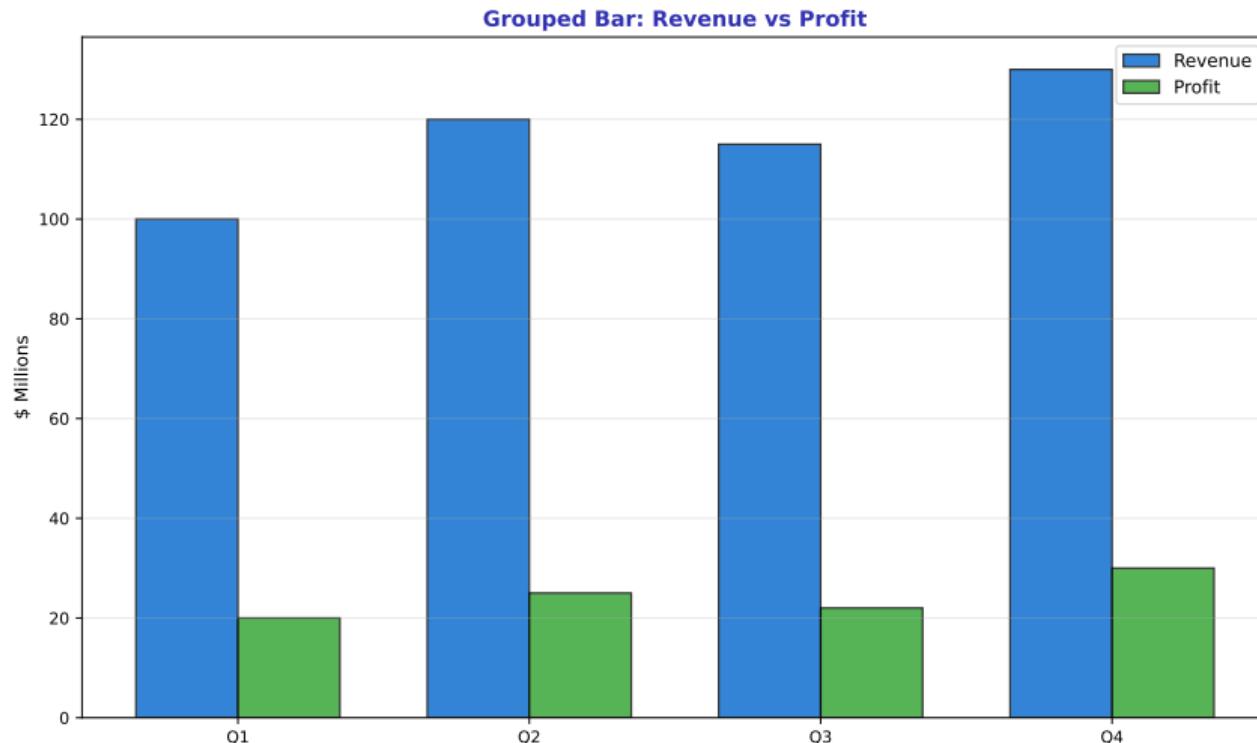
Comparing categorical values

## Horizontal Bar



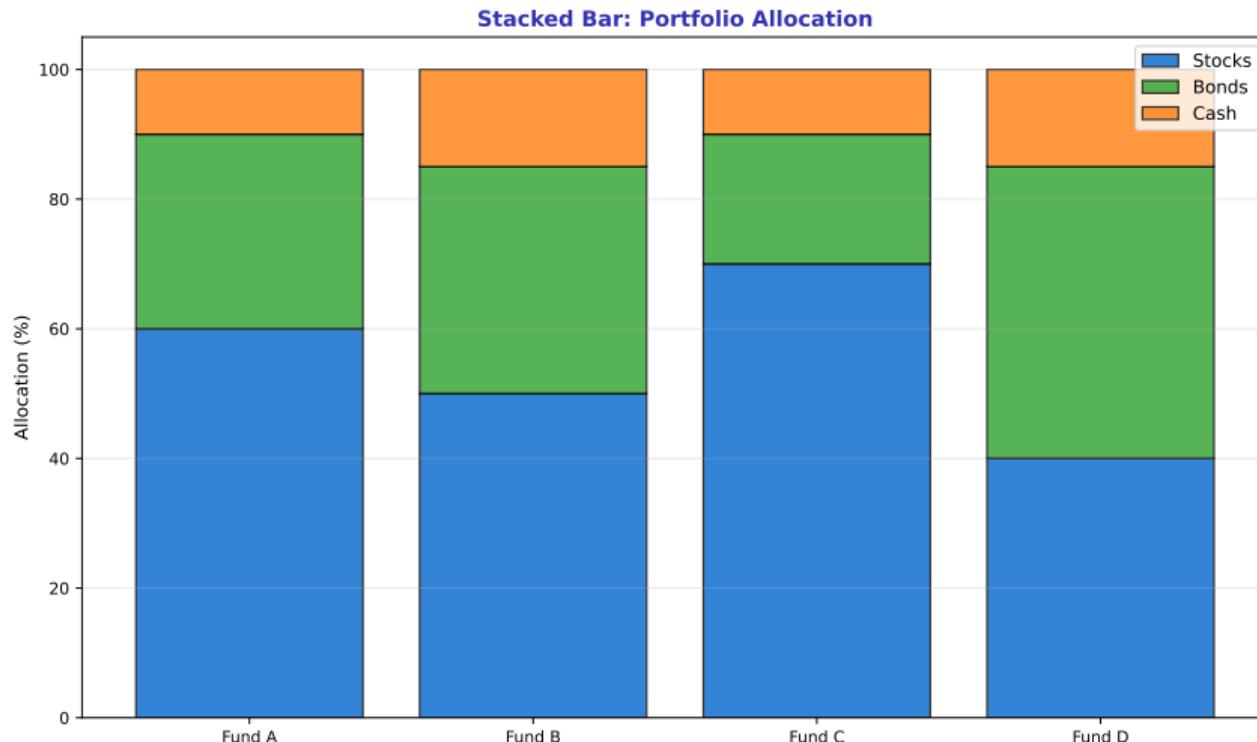
Better for long category names

## Grouped Bar



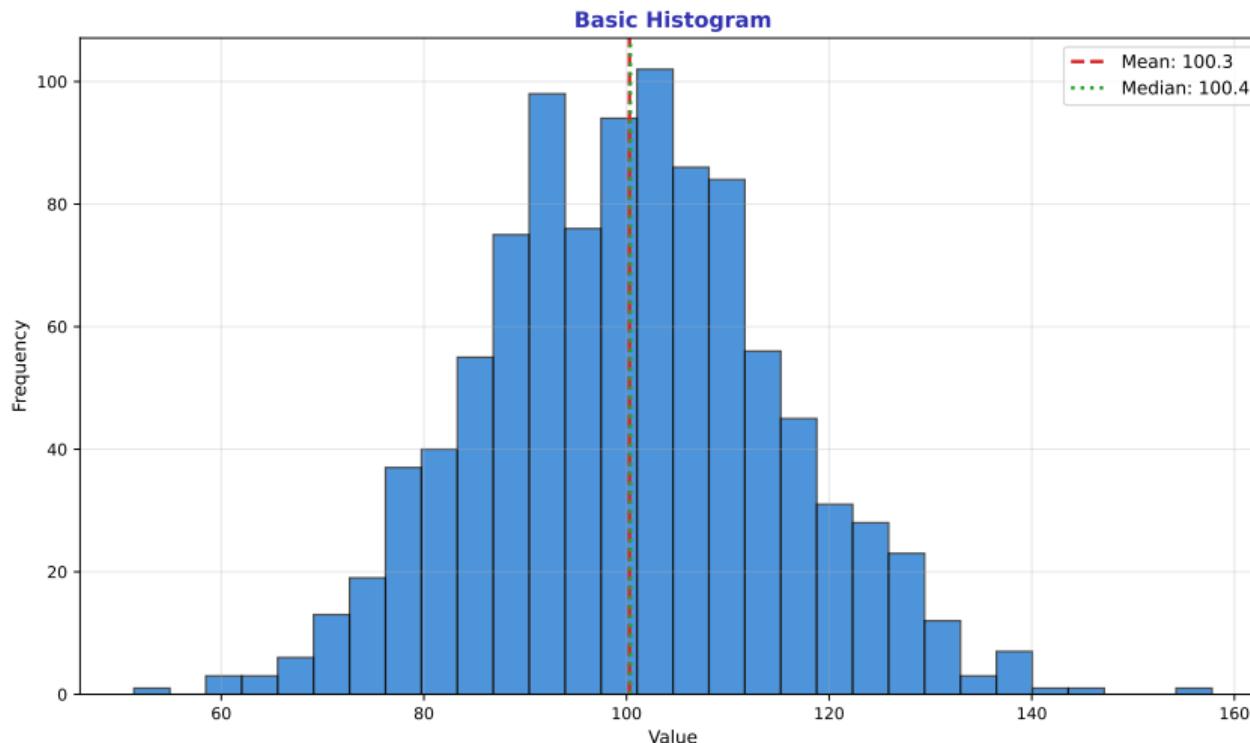
Comparing multiple series by category

## Stacked Bar



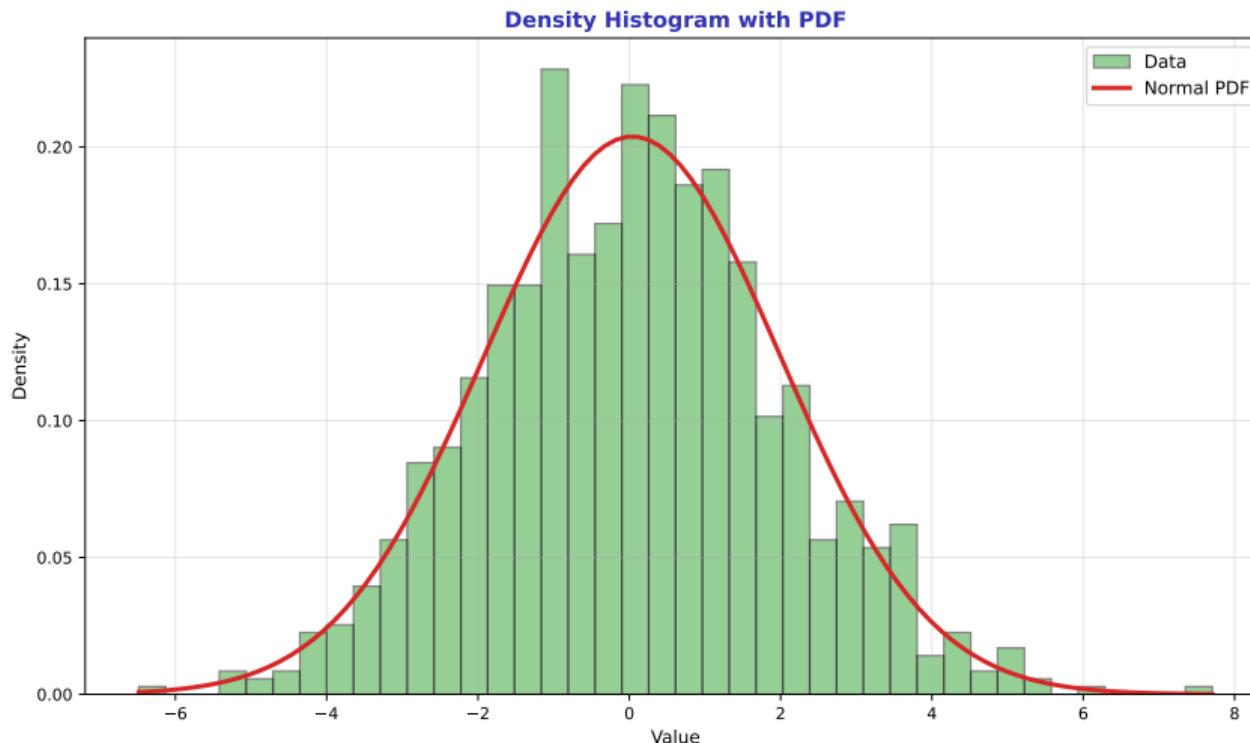
Showing composition of totals

# Basic Histogram



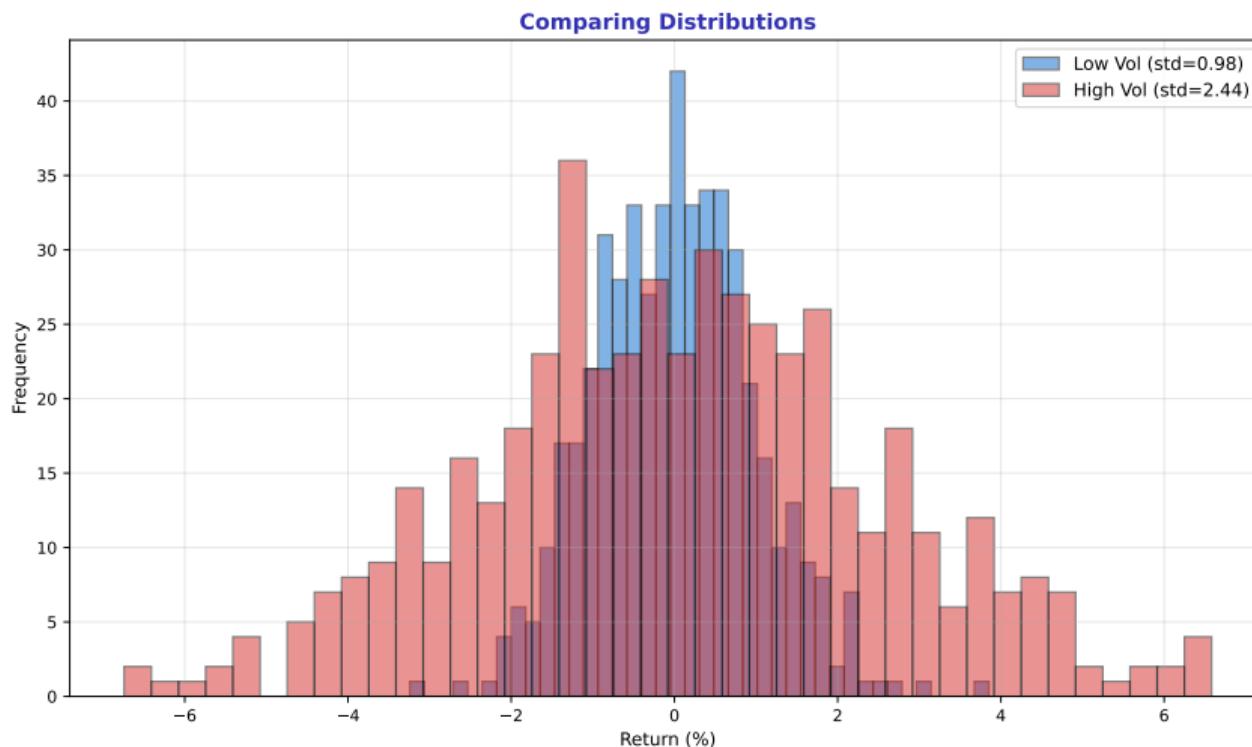
Distribution with mean and median

# Density Histogram



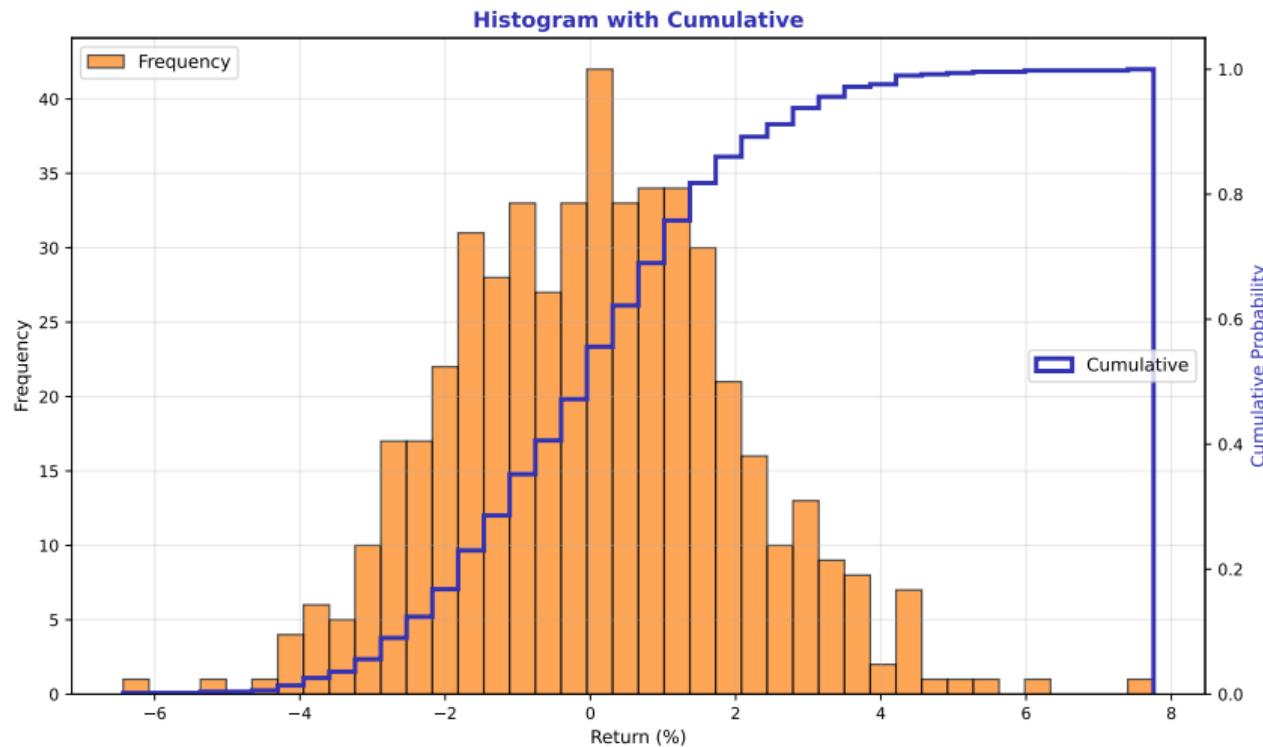
Normalized with PDF overlay

# Comparing Distributions



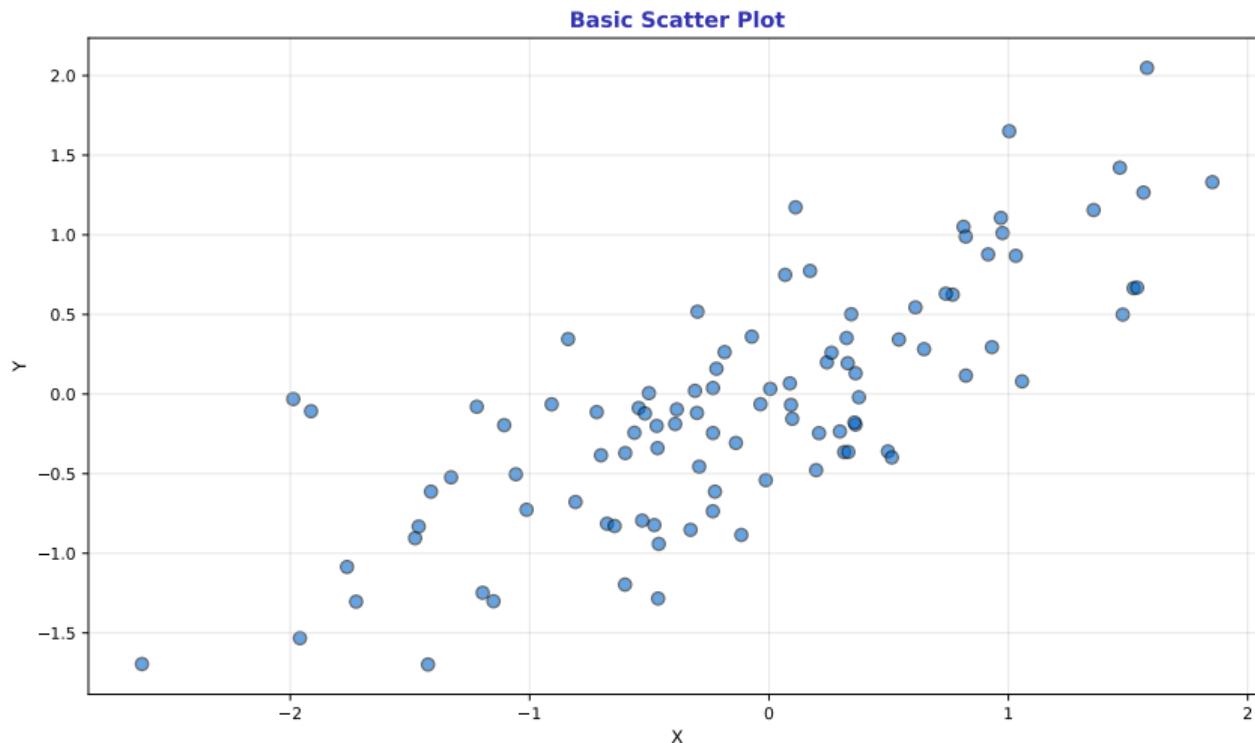
Multiple overlapping histograms

# Cumulative Histogram



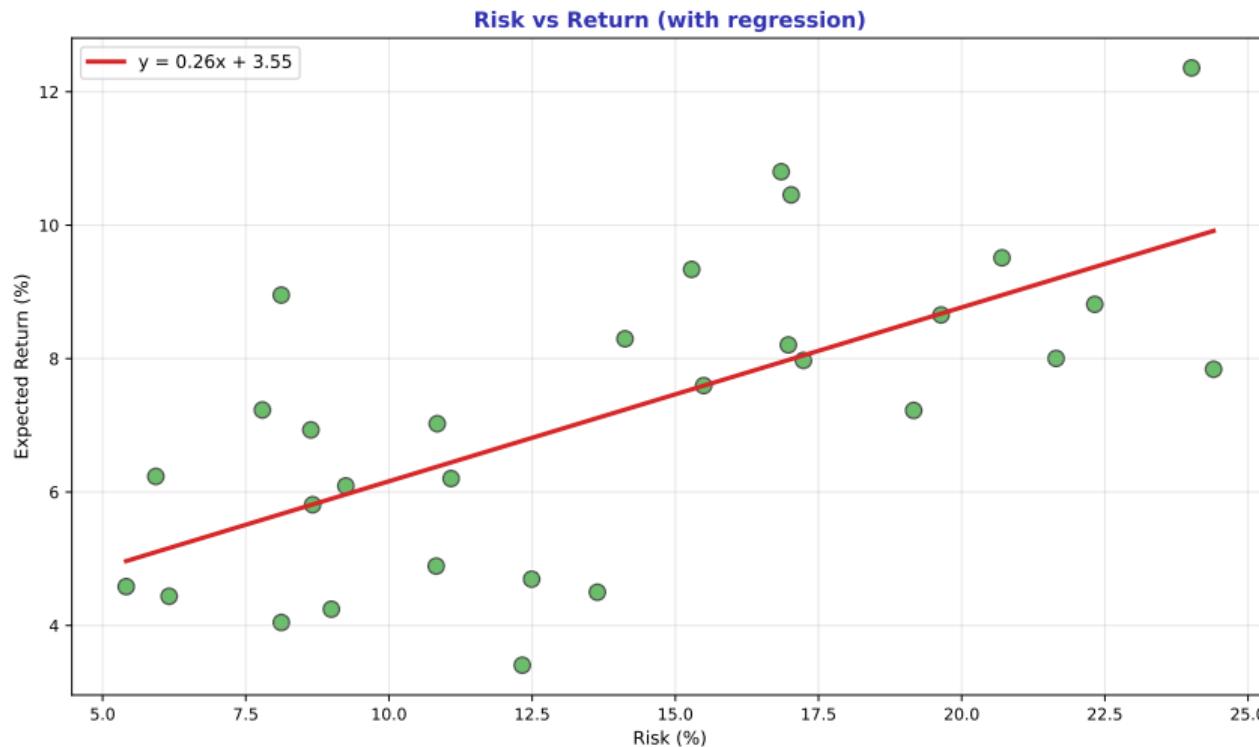
Frequency with CDF

## Basic Scatter



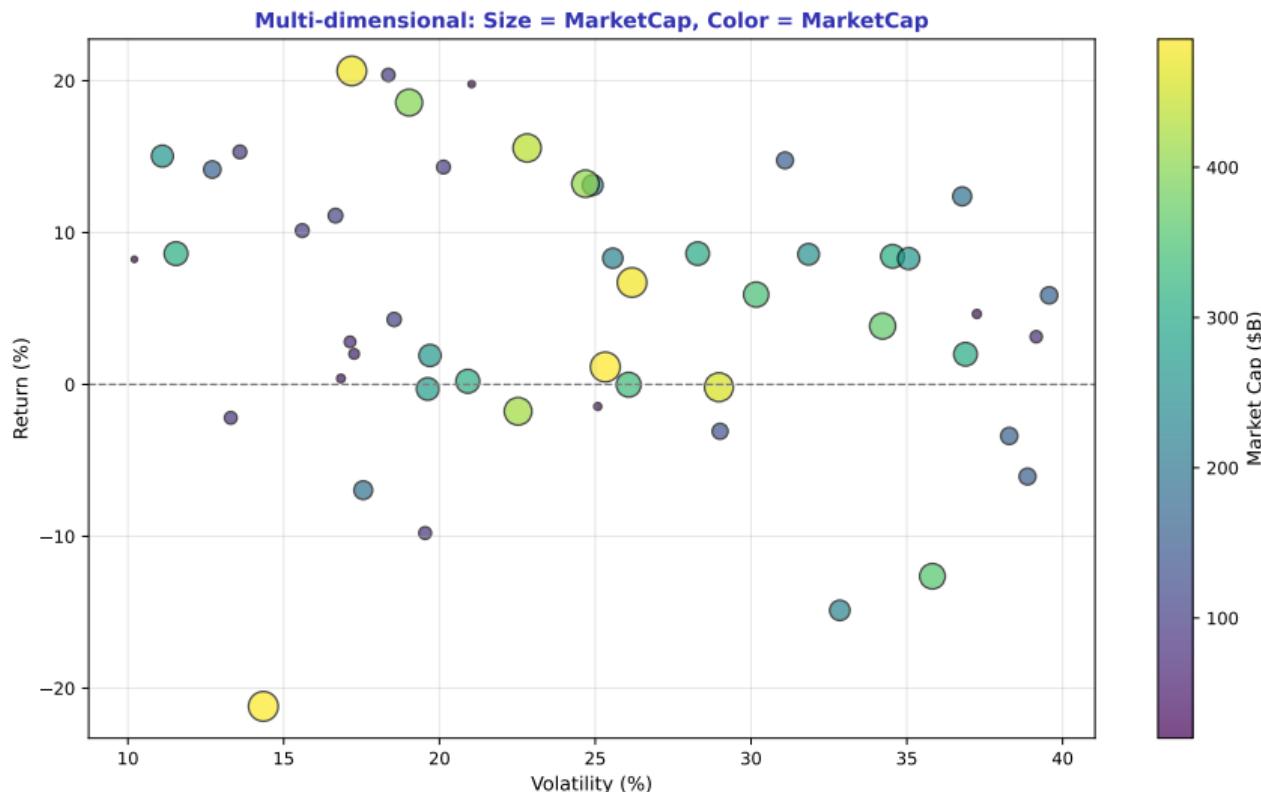
Simple X vs Y relationship

## Scatter with Regression



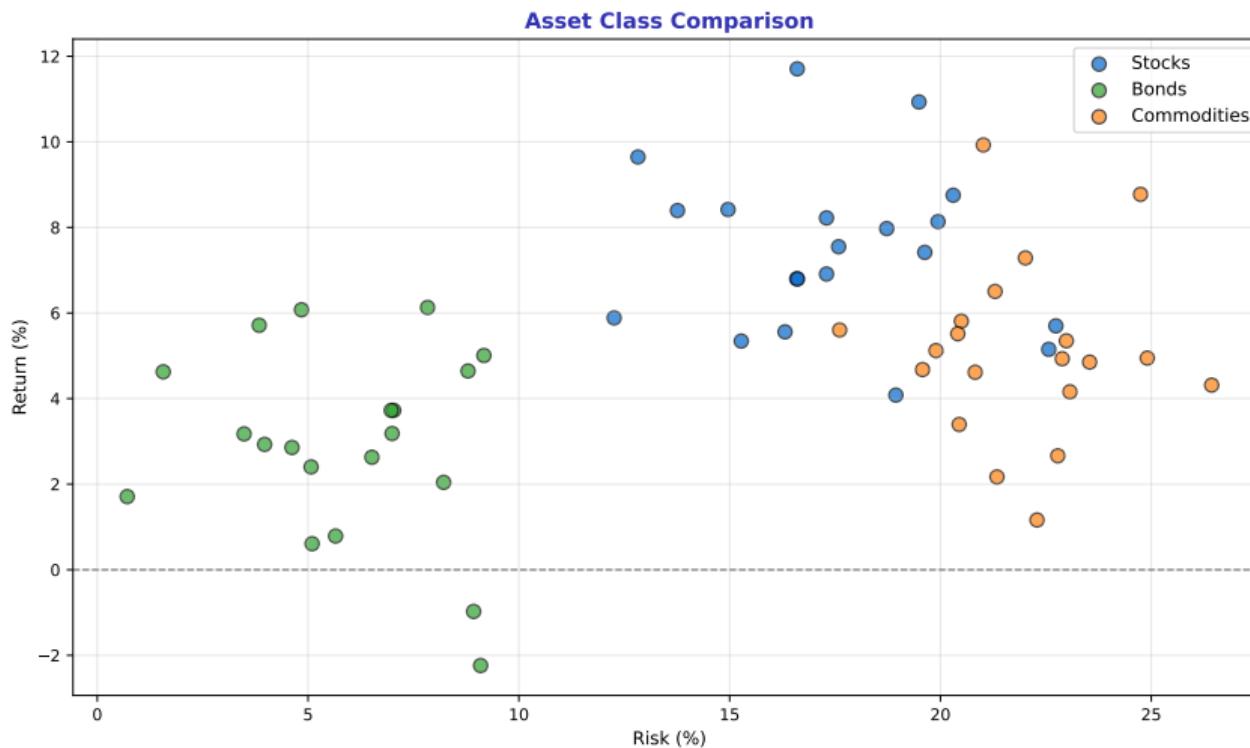
Adding trend line

# Multidimensional Scatter



Size and color encoding

## Grouped Scatter



Comparing categories

add\_subplot (sin)

fig.add\_subplot(2, 2, 1): sin(x)

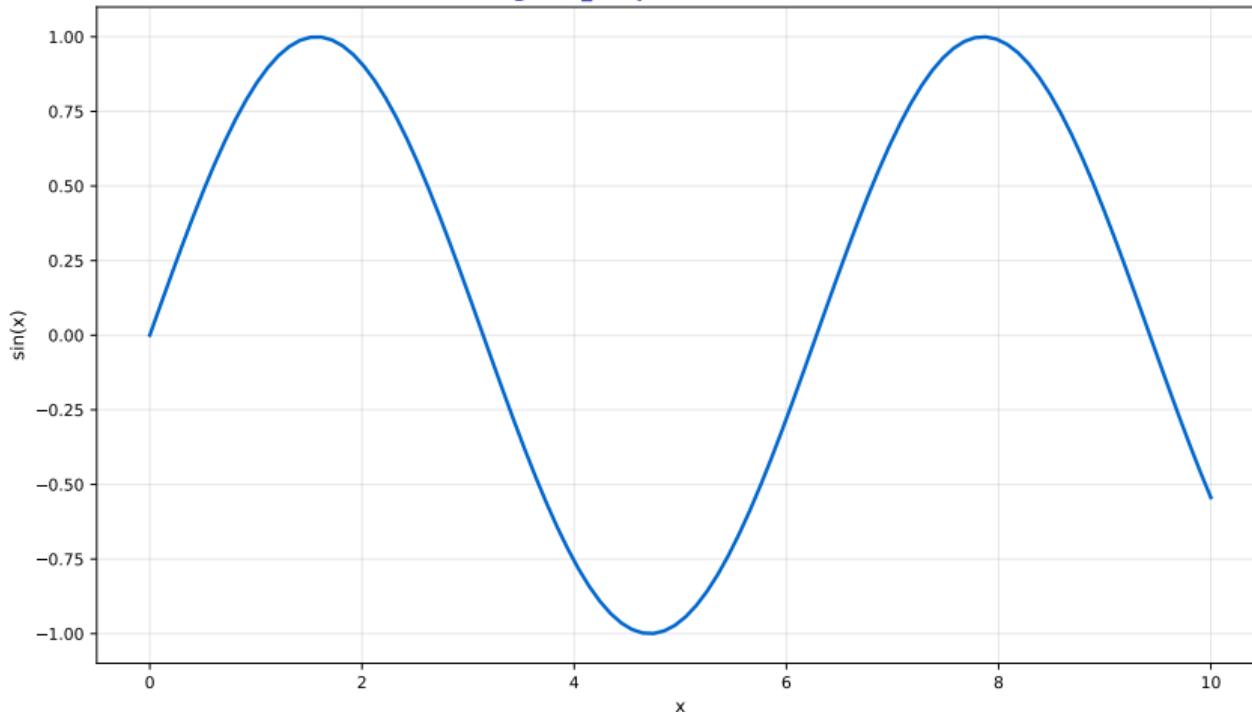
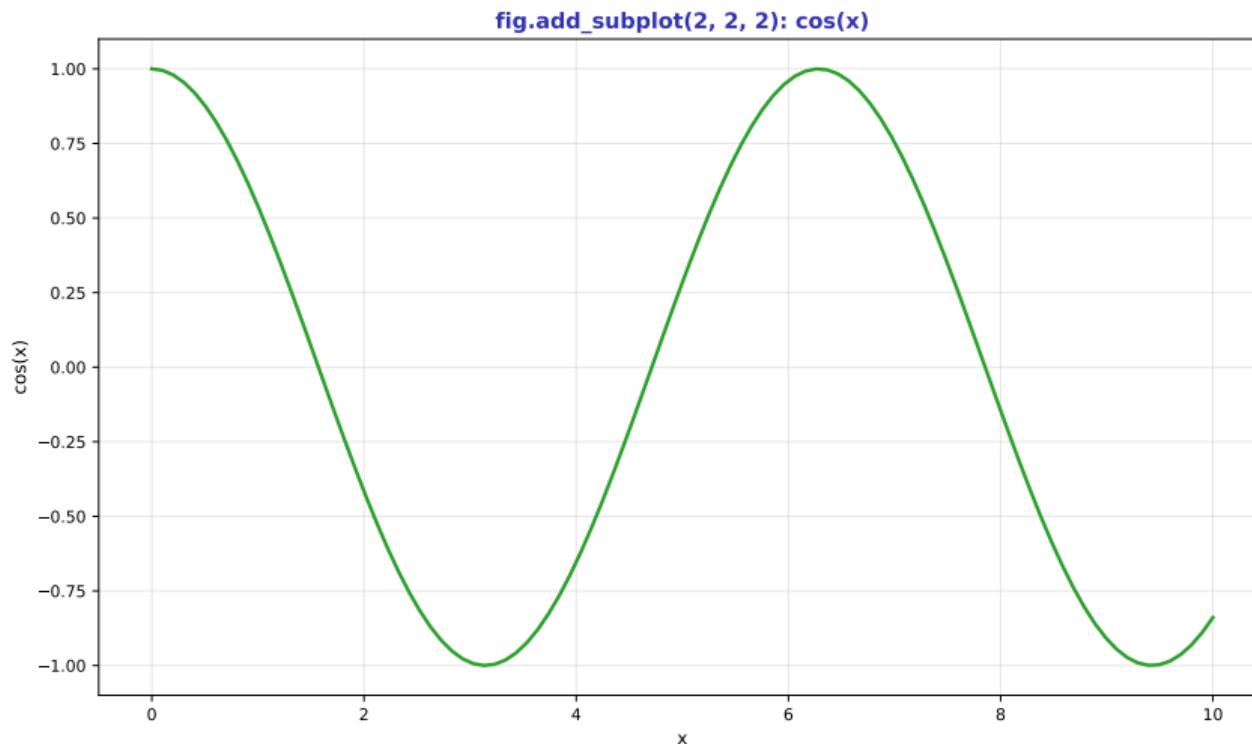


fig.add\_subplot(rows, cols, index)

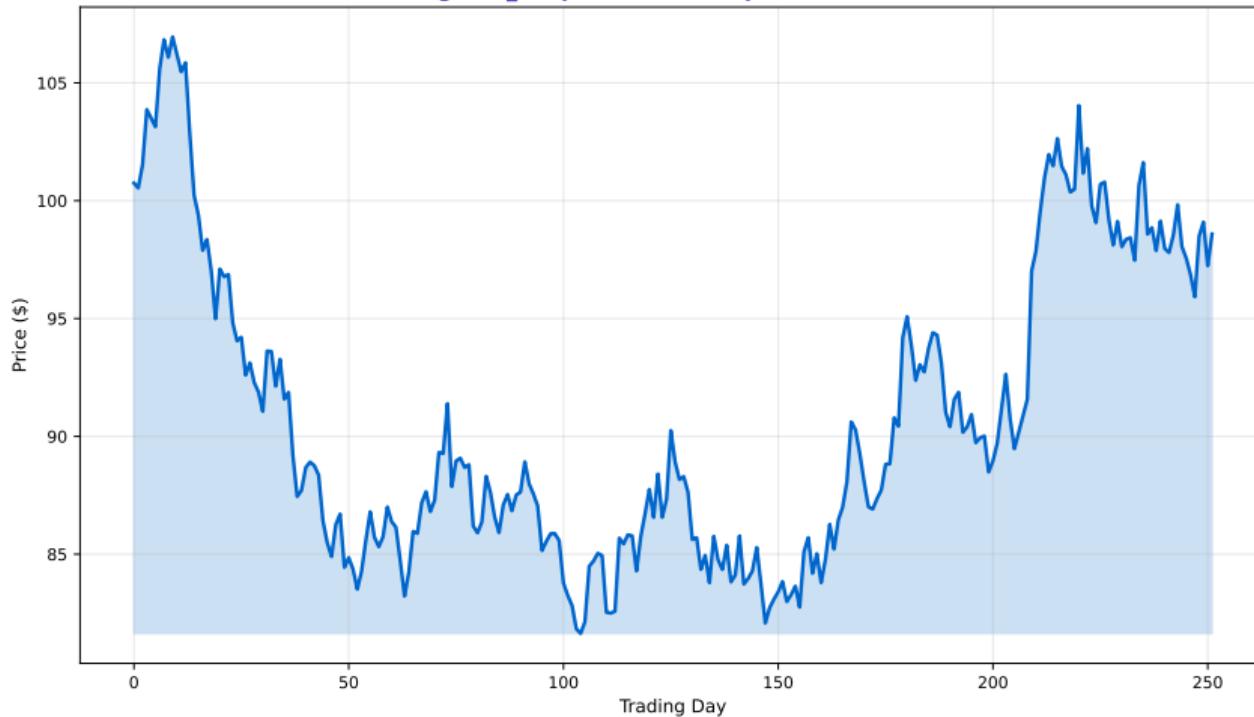
## add\_subplot (cos)



Multiple axes in one figure

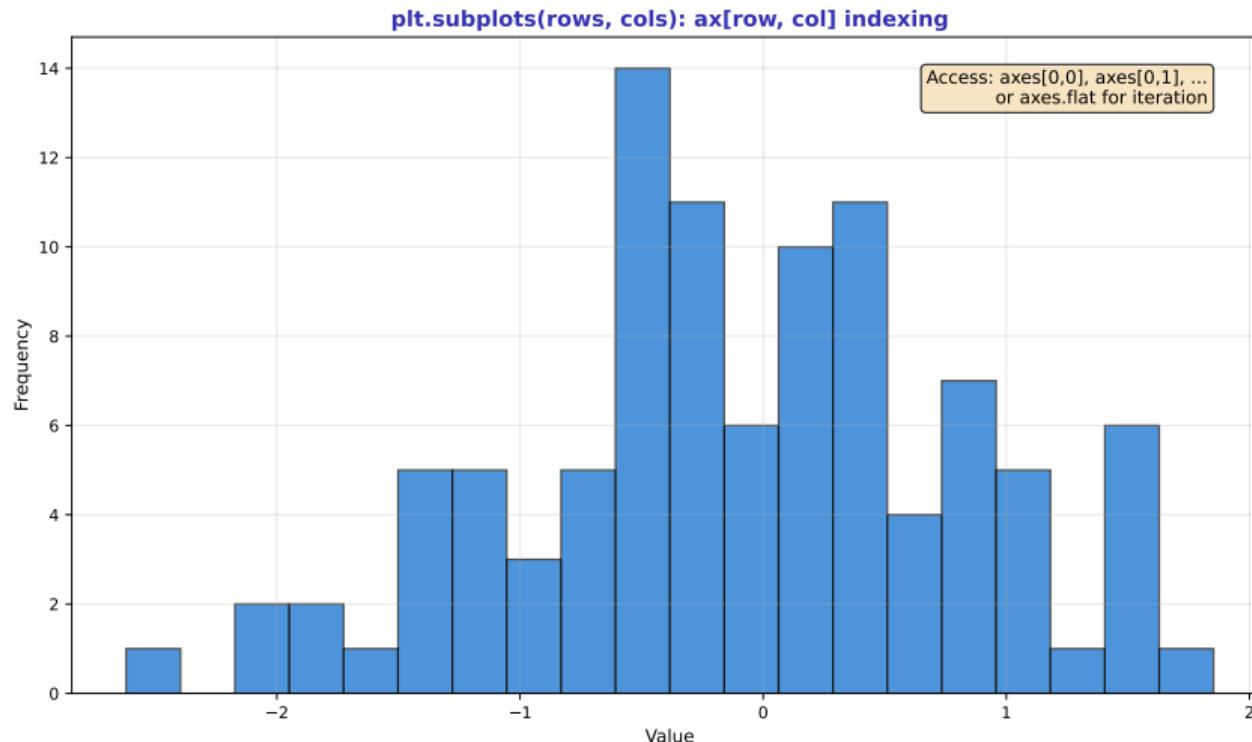
## Spanning Subplot

`fig.add_subplot(2, 1, 2) - Spans full width`



Subplot spanning multiple positions

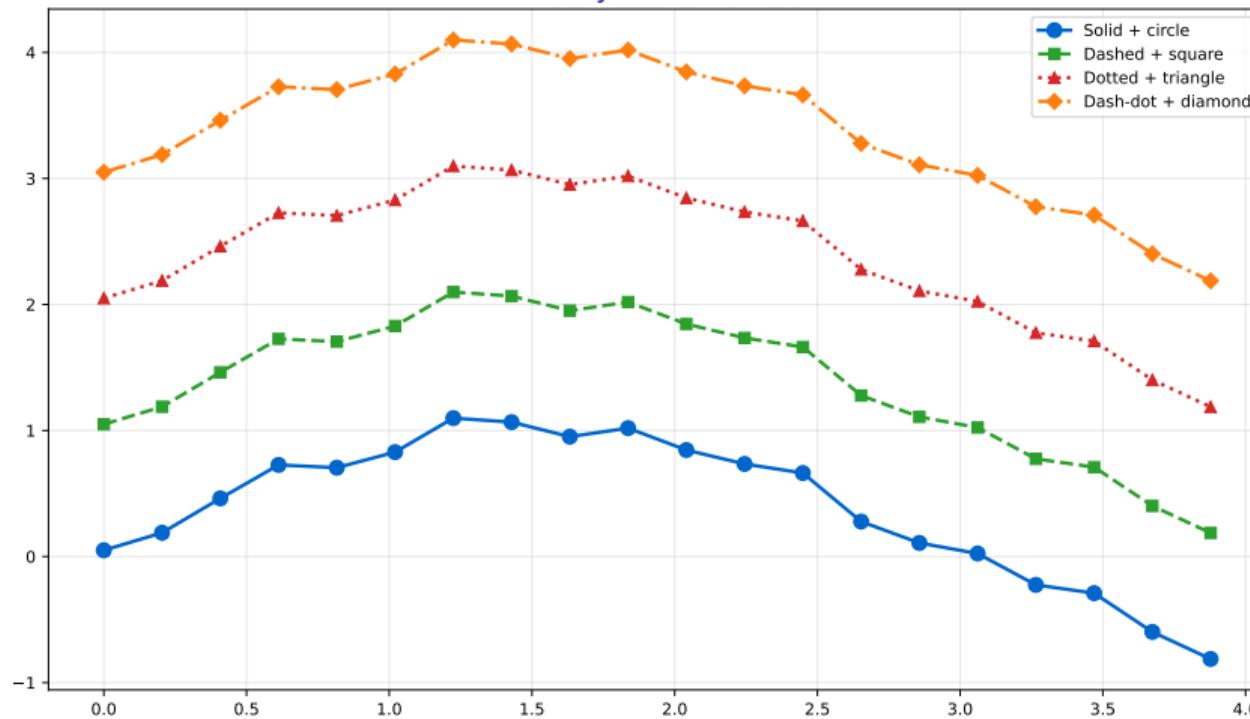
## Subplots Iteration



`plt.subplots(rows, cols) and indexing`

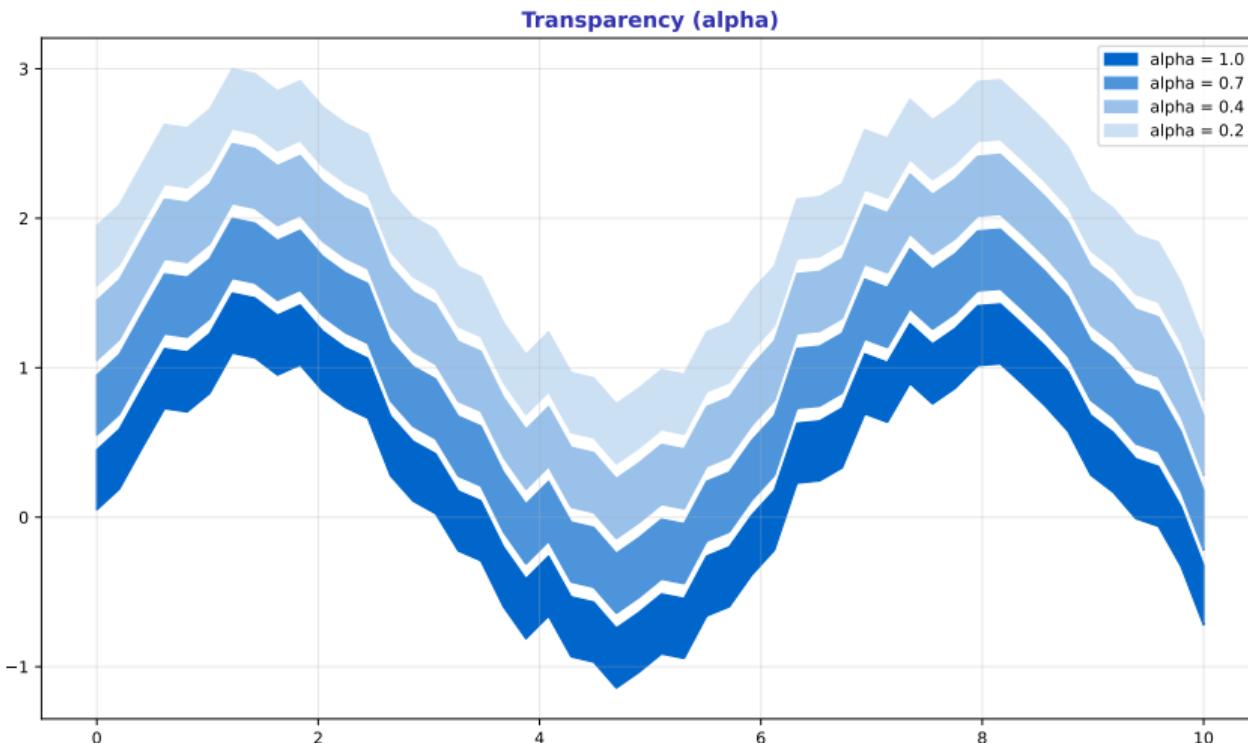
# Line Styles

Line Styles and Markers



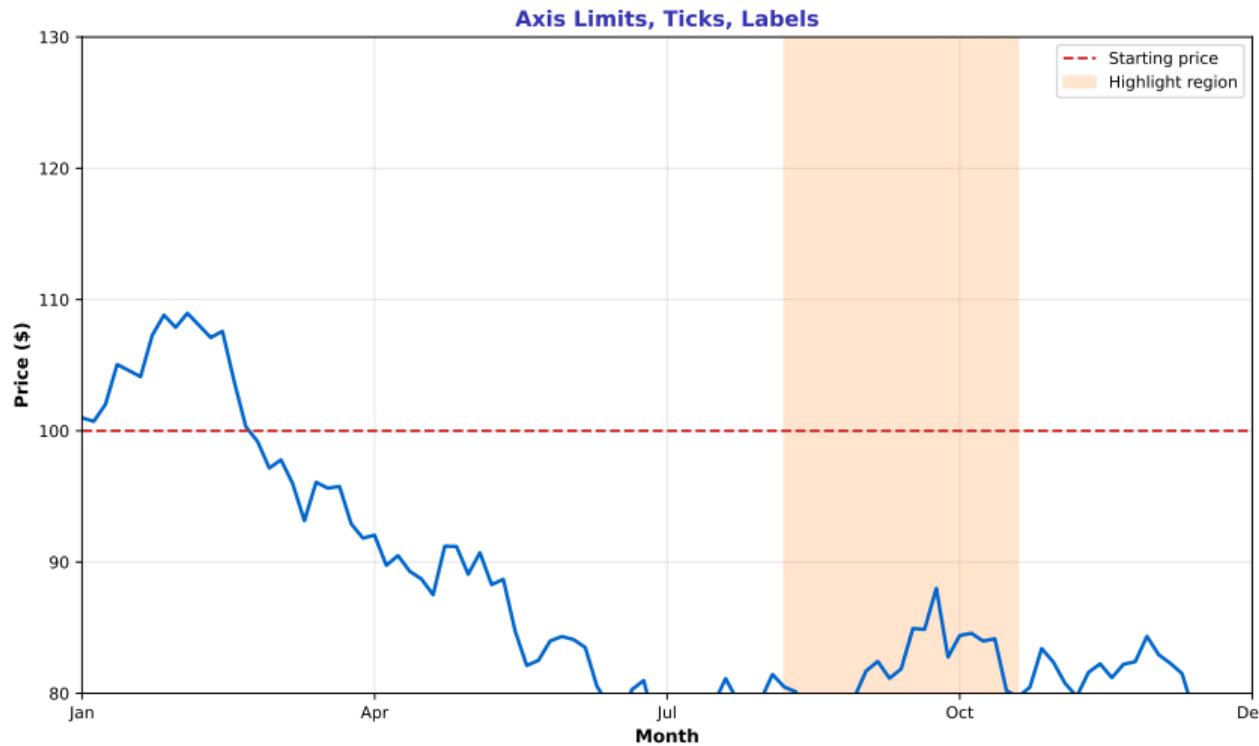
Solid, dashed, dotted, markers

# Transparency



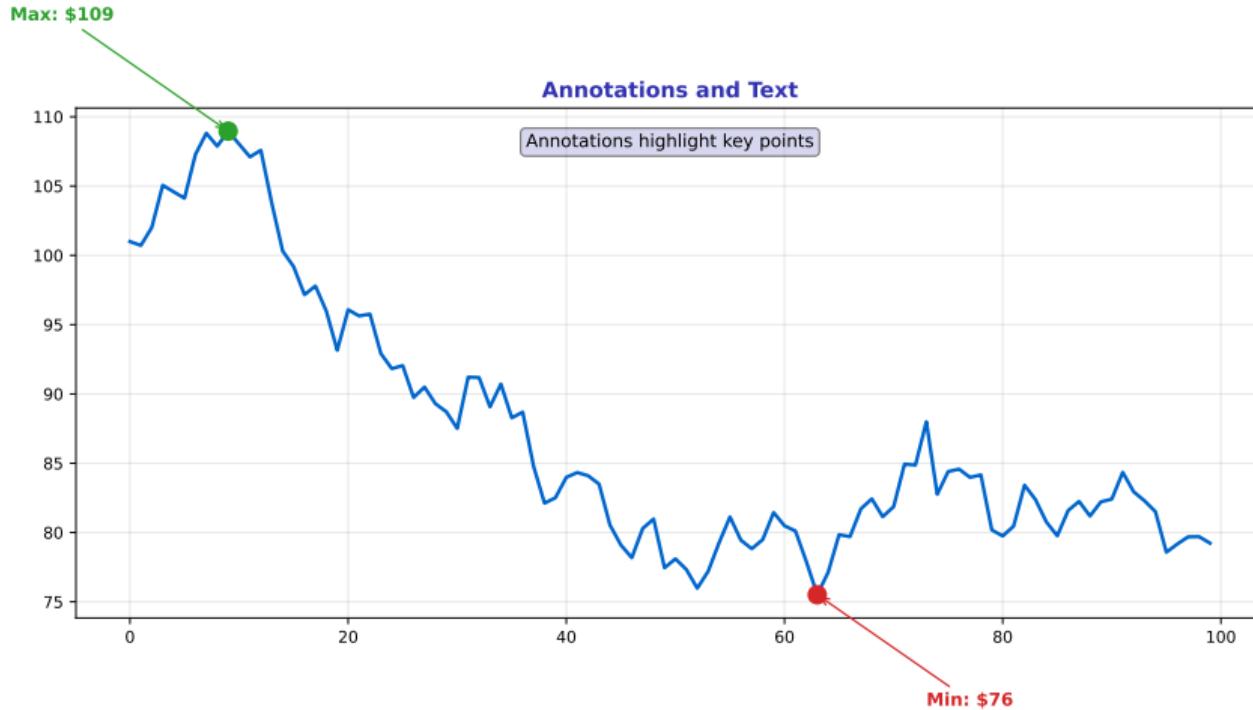
Using alpha for layering

## Axis Customization



Limits, ticks, labels

## Annotations and Text



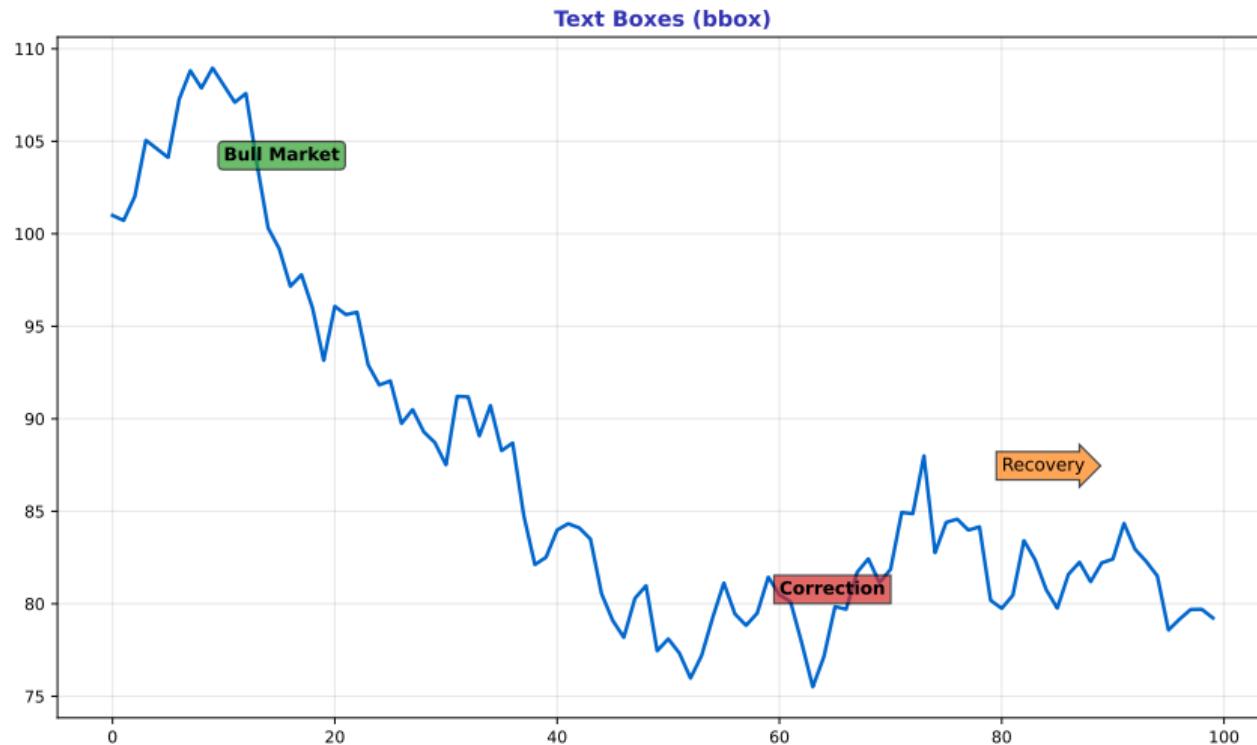
Highlighting key points

# Arrow Styles



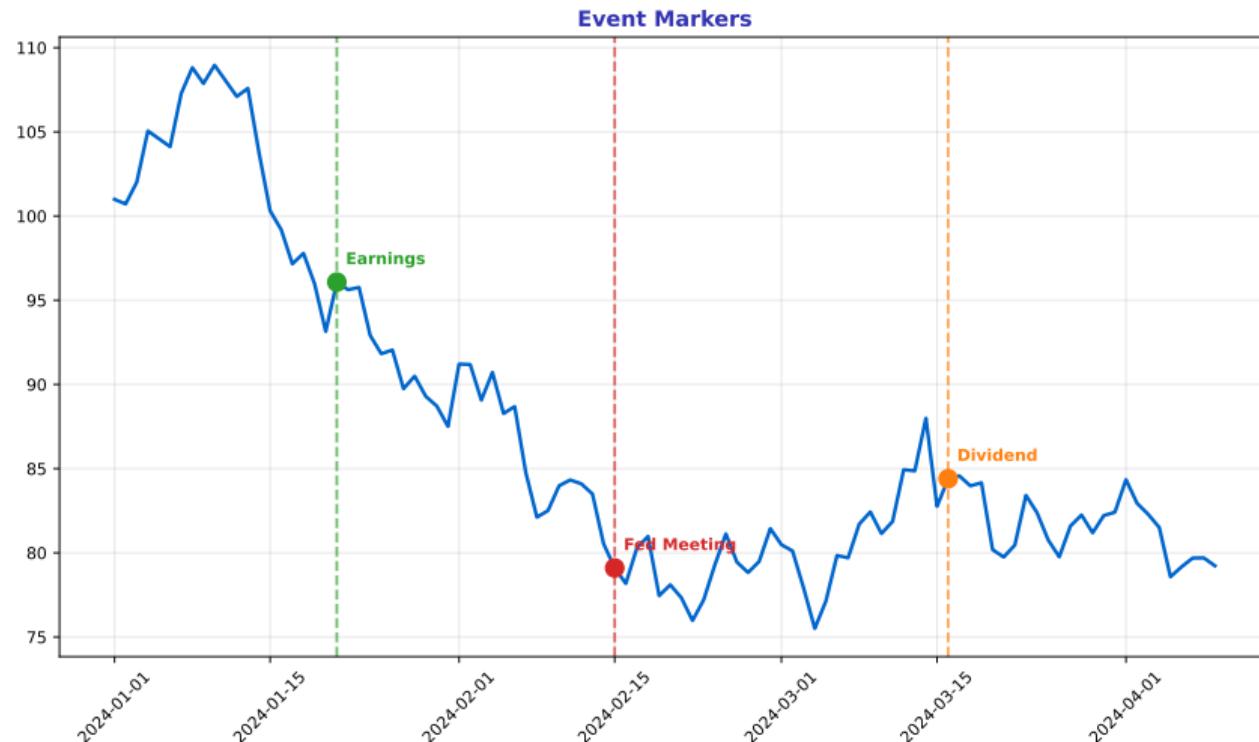
Different annotation arrows

## Text Boxes

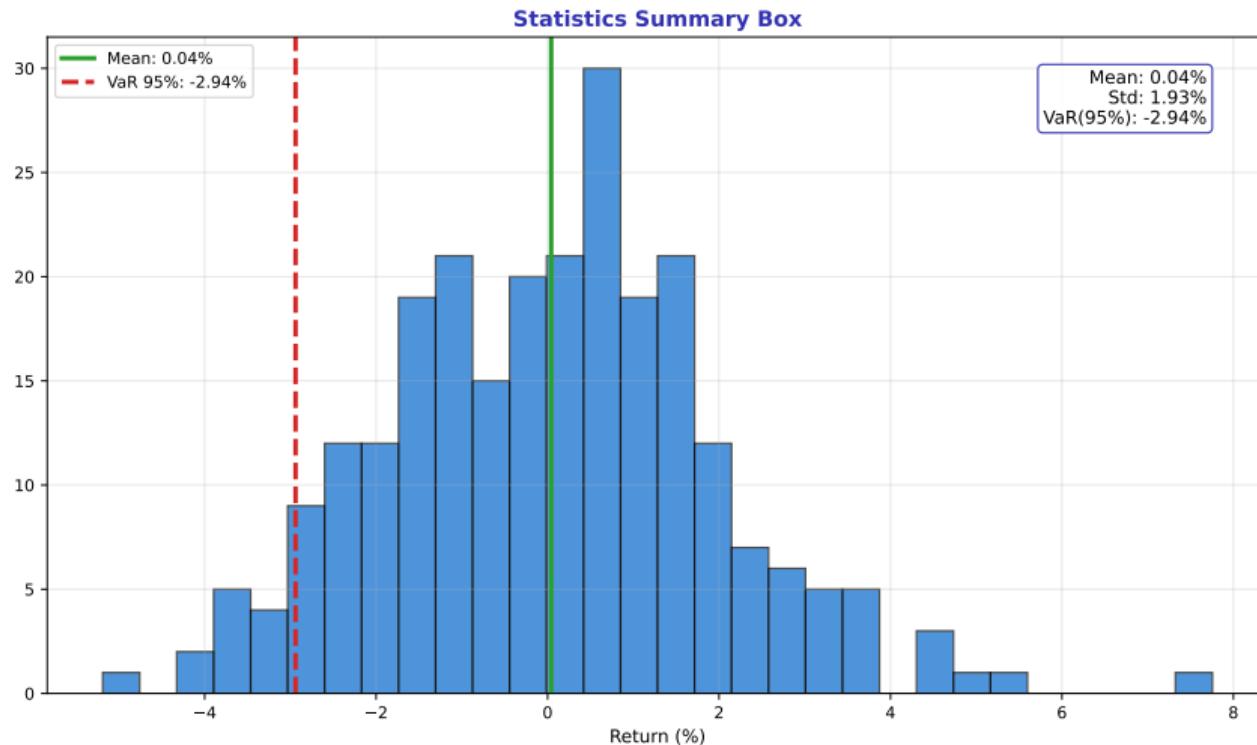


bbox styles for annotations

## Event Markers

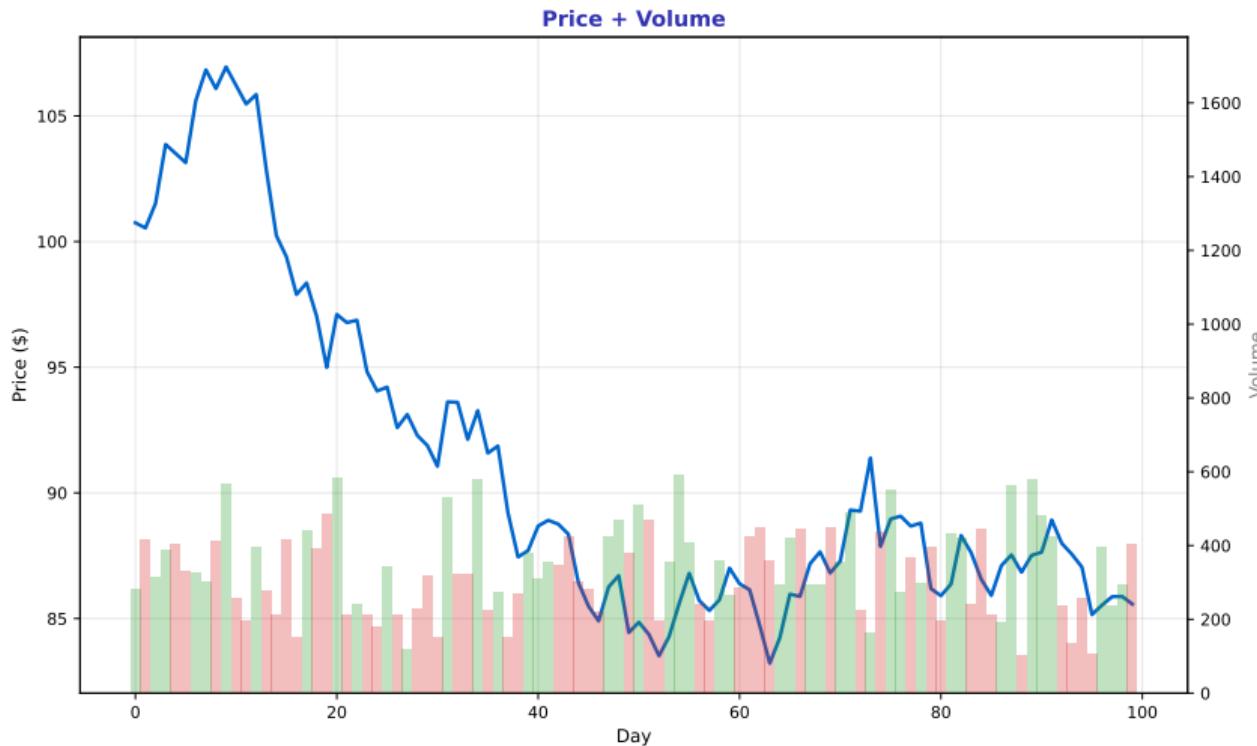


Highlighting specific dates



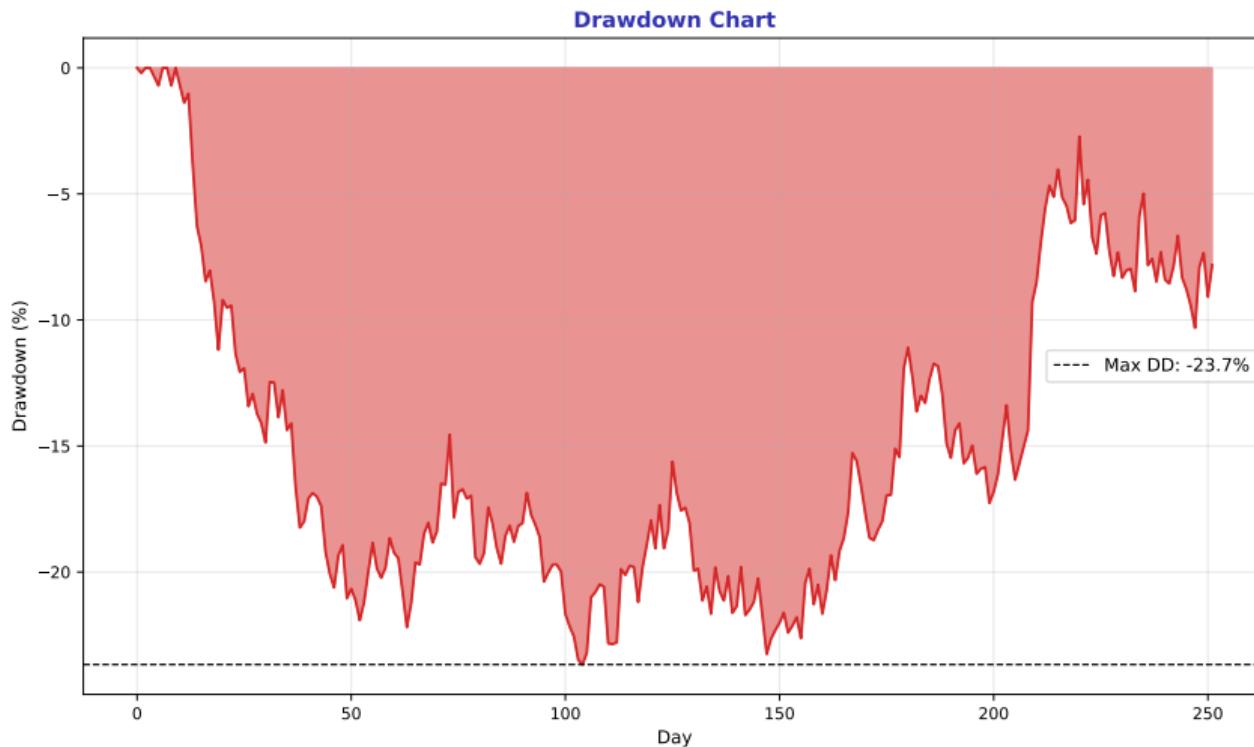
Key metrics annotation

## Price + Volume



Dual axis chart

# Drawdown



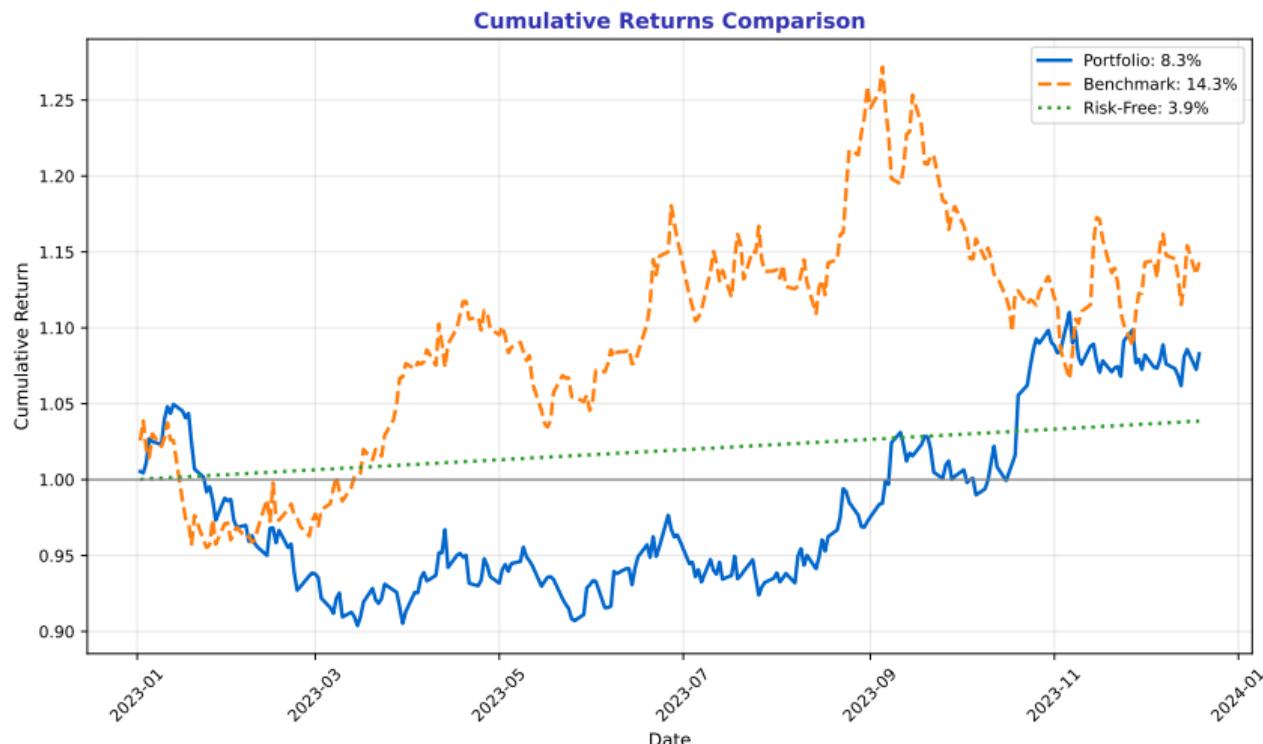
Visualizing losses from peak

## Rolling Sharpe



Performance over time

## Cumulative Returns



## Strategy comparison

## Key Takeaways:

- Line plots for time series, bar charts for categories
- Histograms show distributions, scatter plots show relationships
- Subplots combine multiple views
- Customization: colors, styles, annotations, text boxes
- Finance applications: price/volume, drawdown, Sharpe, returns

**Statistics + Visualization = Data Science foundation**

## Lesson 18: Seaborn Statistical Plots

Data Science with Python – BSc Course

45 Minutes

## After this lesson, you will be able to:

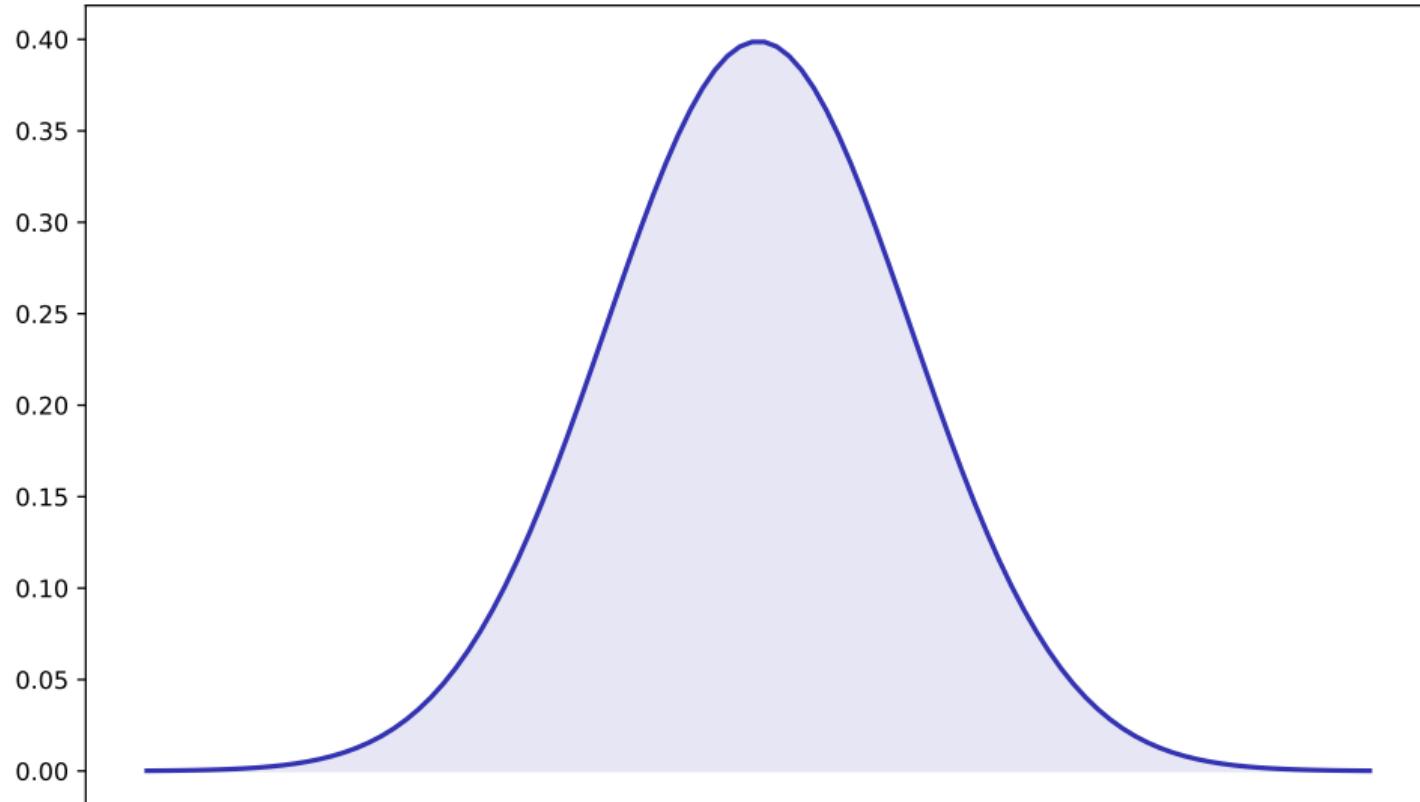
- Use seaborn for statistical visualization
- Create distribution and categorical plots
- Build correlation heatmaps
- Apply professional styling

**Finance application: Statistical analysis of market data**

L18

01 Seaborn Intro

## 02 Distribution Plots



L18

03 Categorical Plots

L18

04 Regression Plots

## 05 Heatmaps



# Pairplot

L18

06 Pairplot

L18

07 Styling

L18

08 Finance Seaborn

## Lesson Summary

### Key Takeaways:

- Use seaborn for statistical visualization
- Create distribution and categorical plots
- Build correlation heatmaps
- Apply professional styling

**Statistics + Visualization = Data Science foundation**

## Lesson 19: Multi-Panel Figures

Data Science with Python – BSc Course

45 Minutes

# Learning Objectives

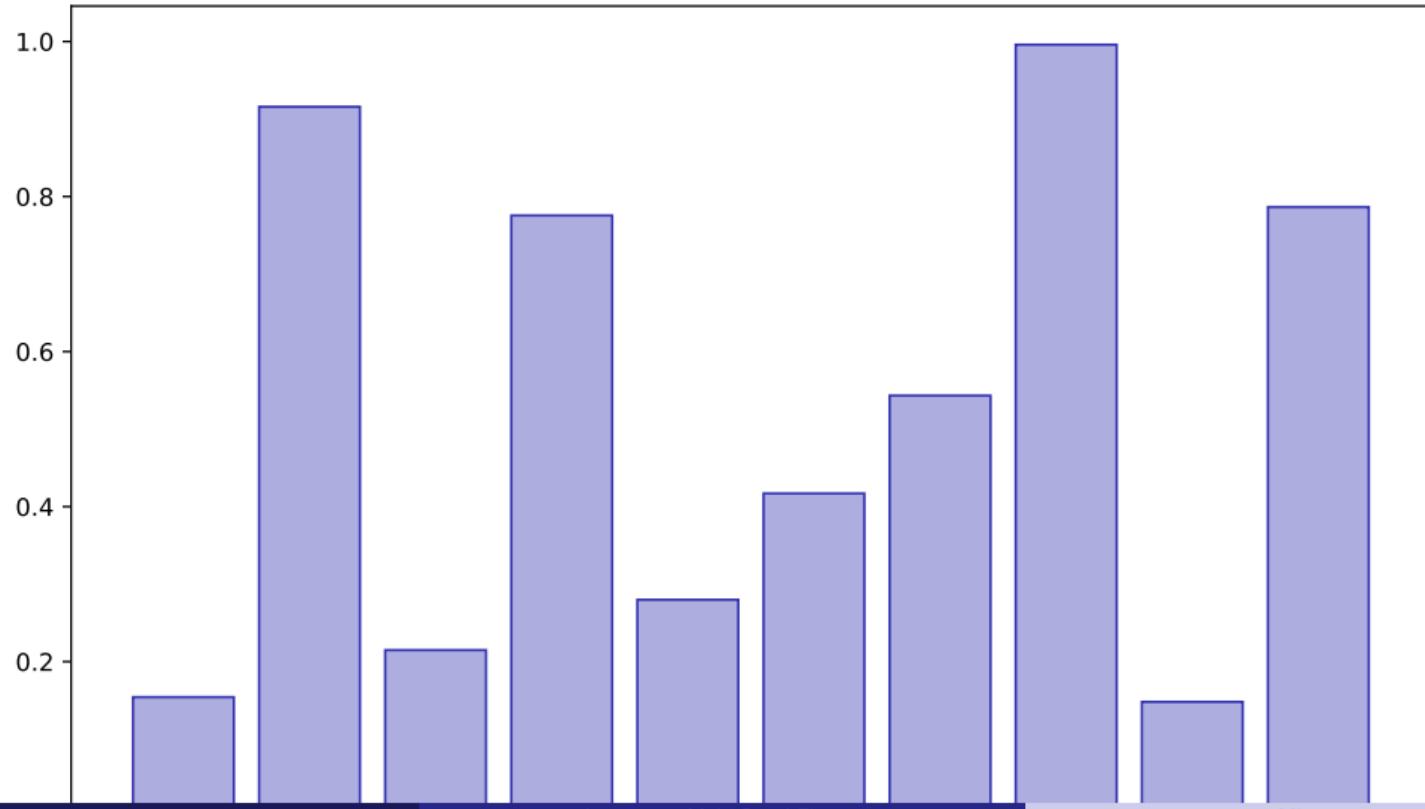
**After this lesson, you will be able to:**

- Create subplots with plt.subplots()
- Arrange multiple visualizations
- Share axes and legends
- Build financial dashboards

**Building towards your final project**

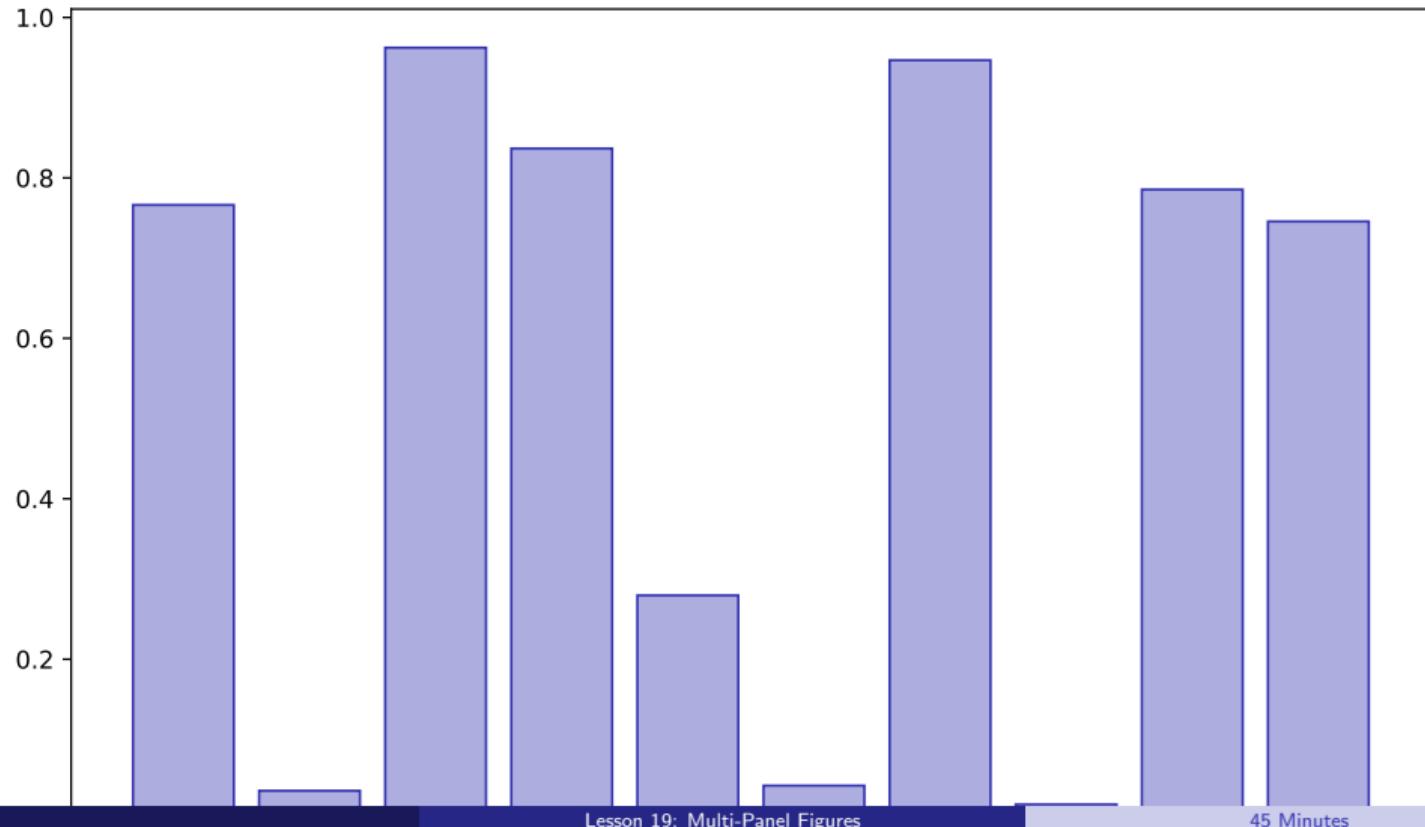
## Subplots Grid

**Subplots Grid**



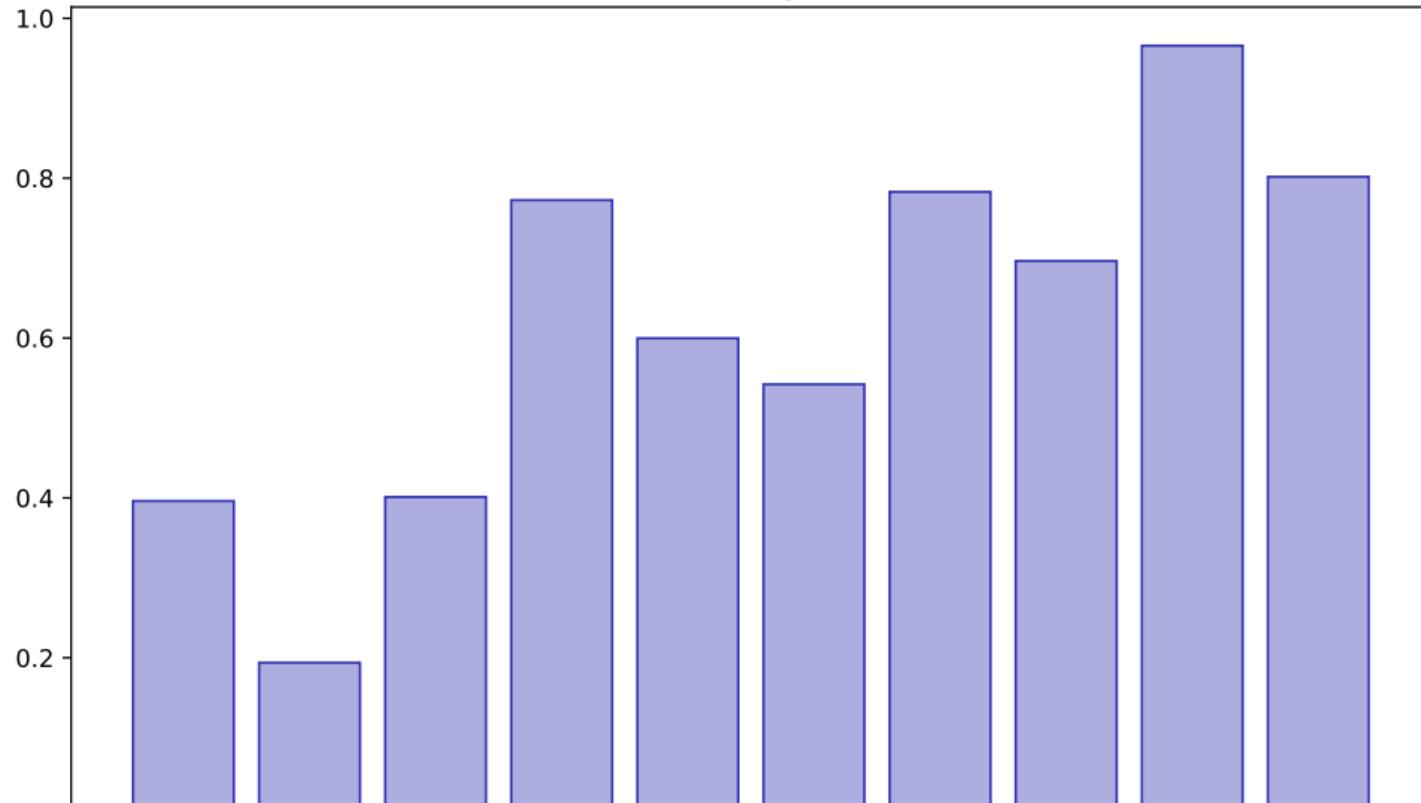
## Shared Axes

### Shared Axes

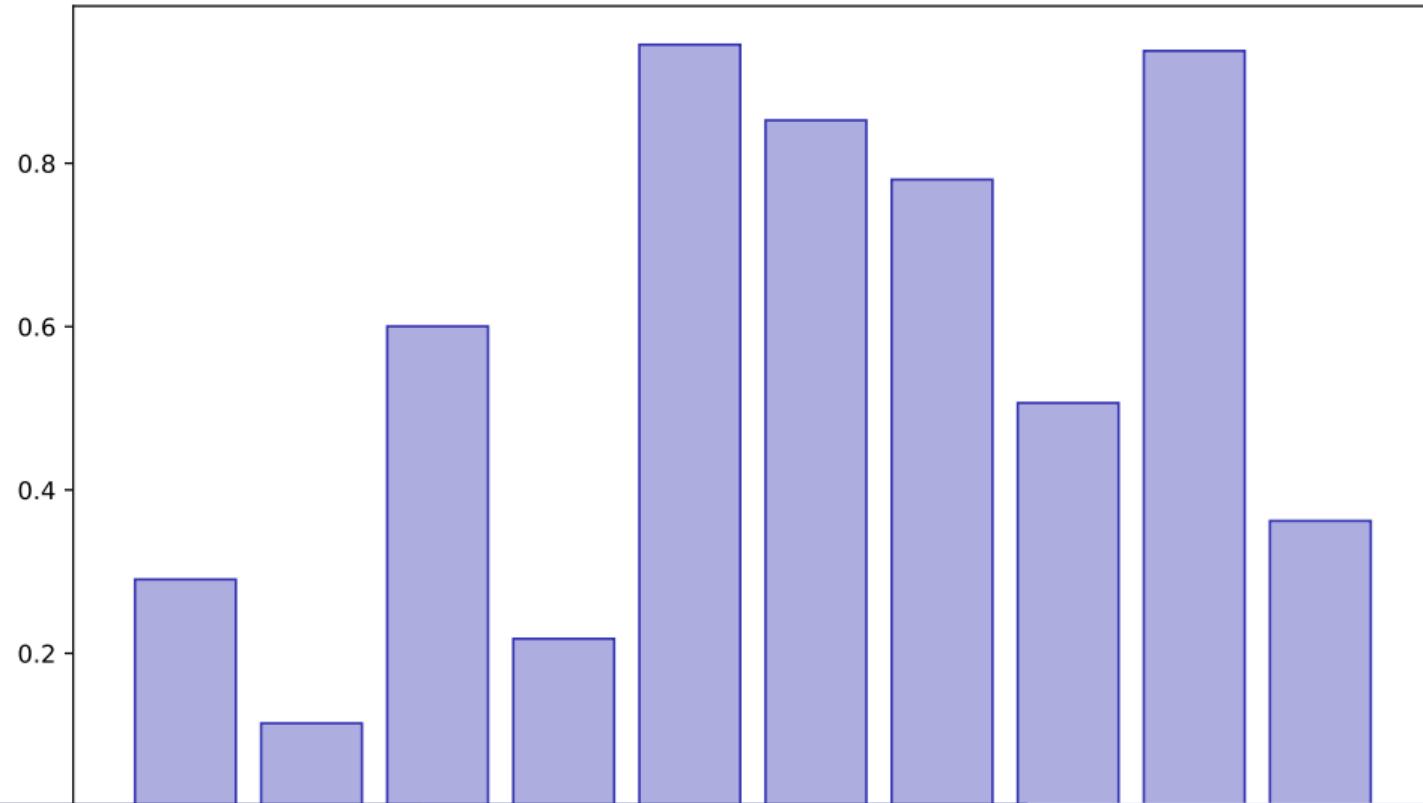


## Mixed Layouts

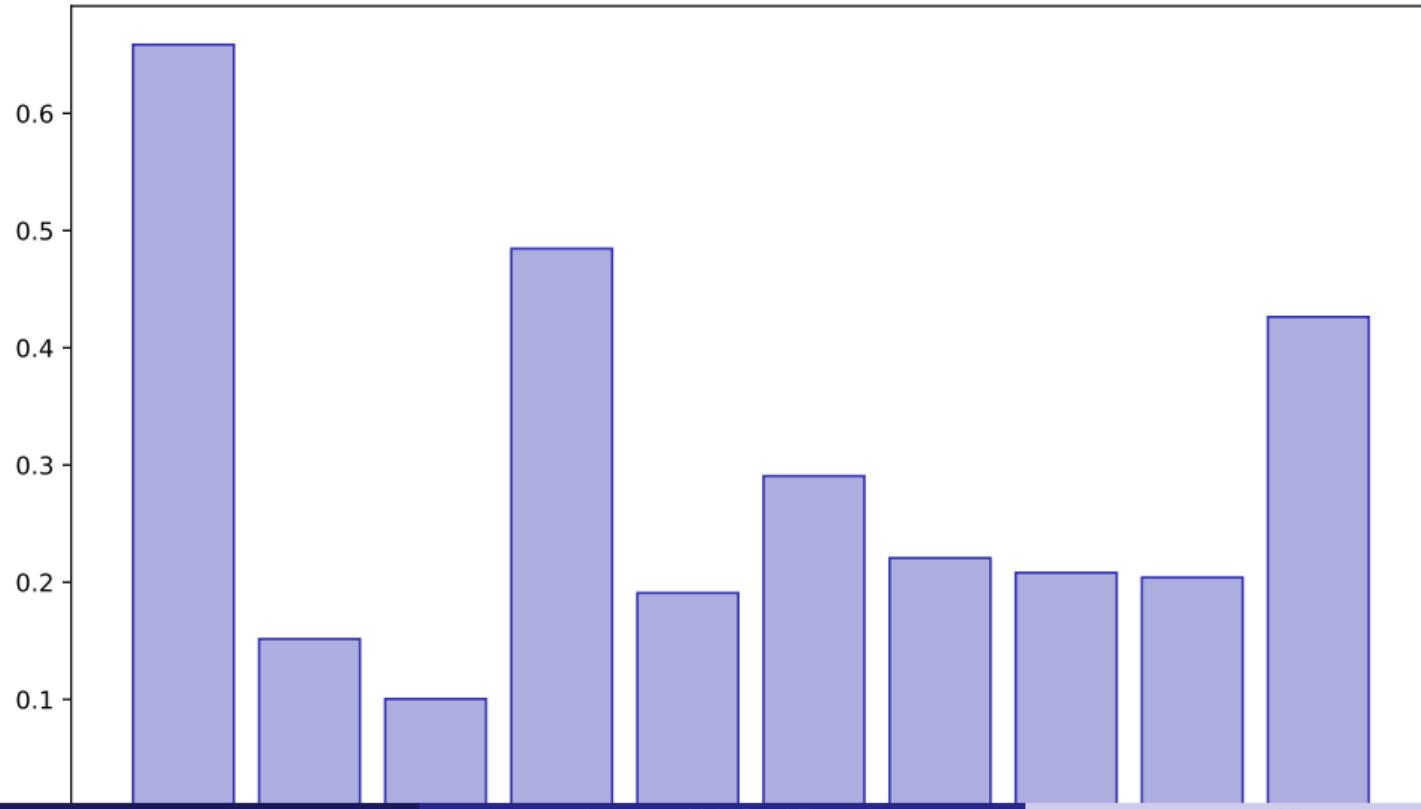
**Mixed Layouts**



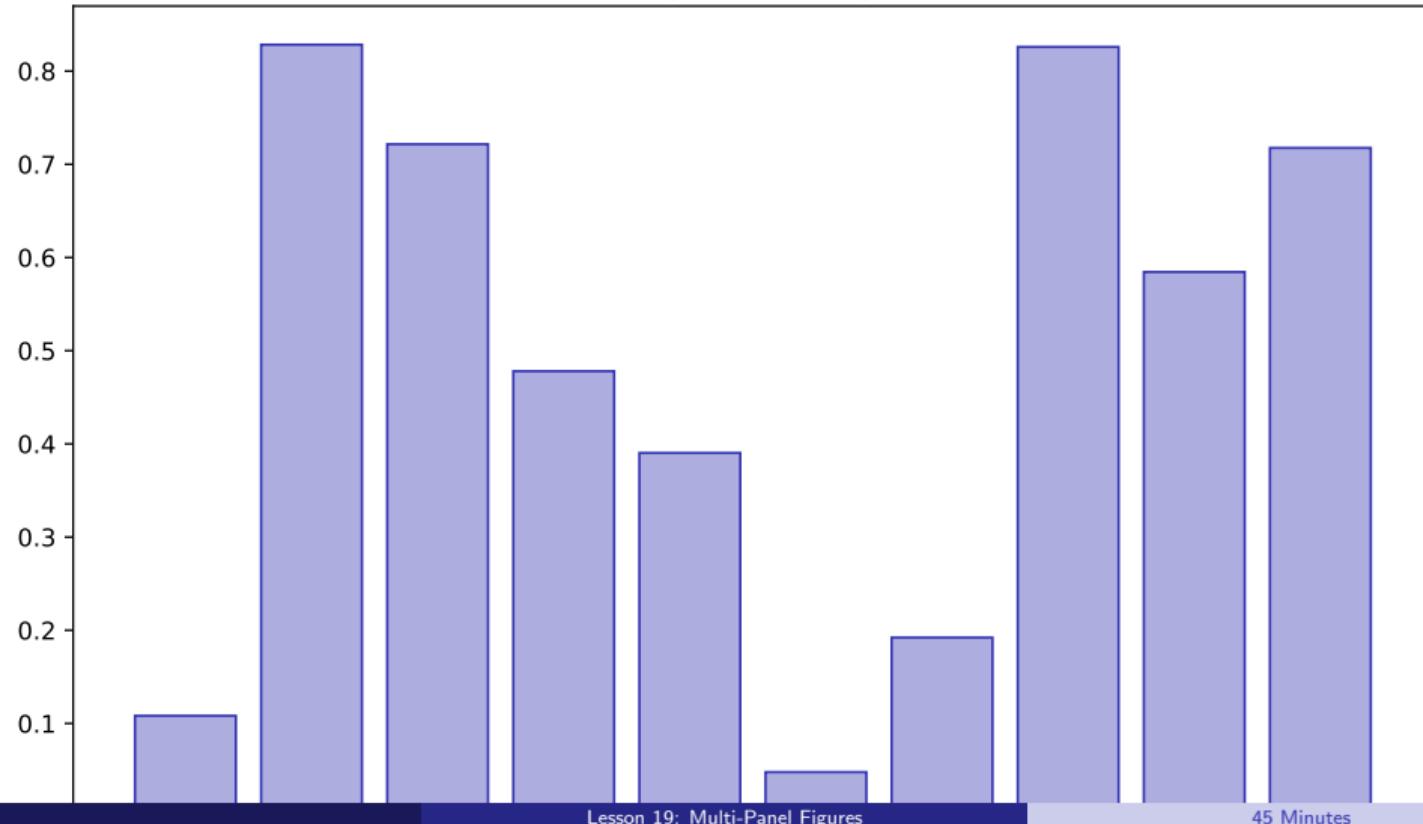
## Gridspec



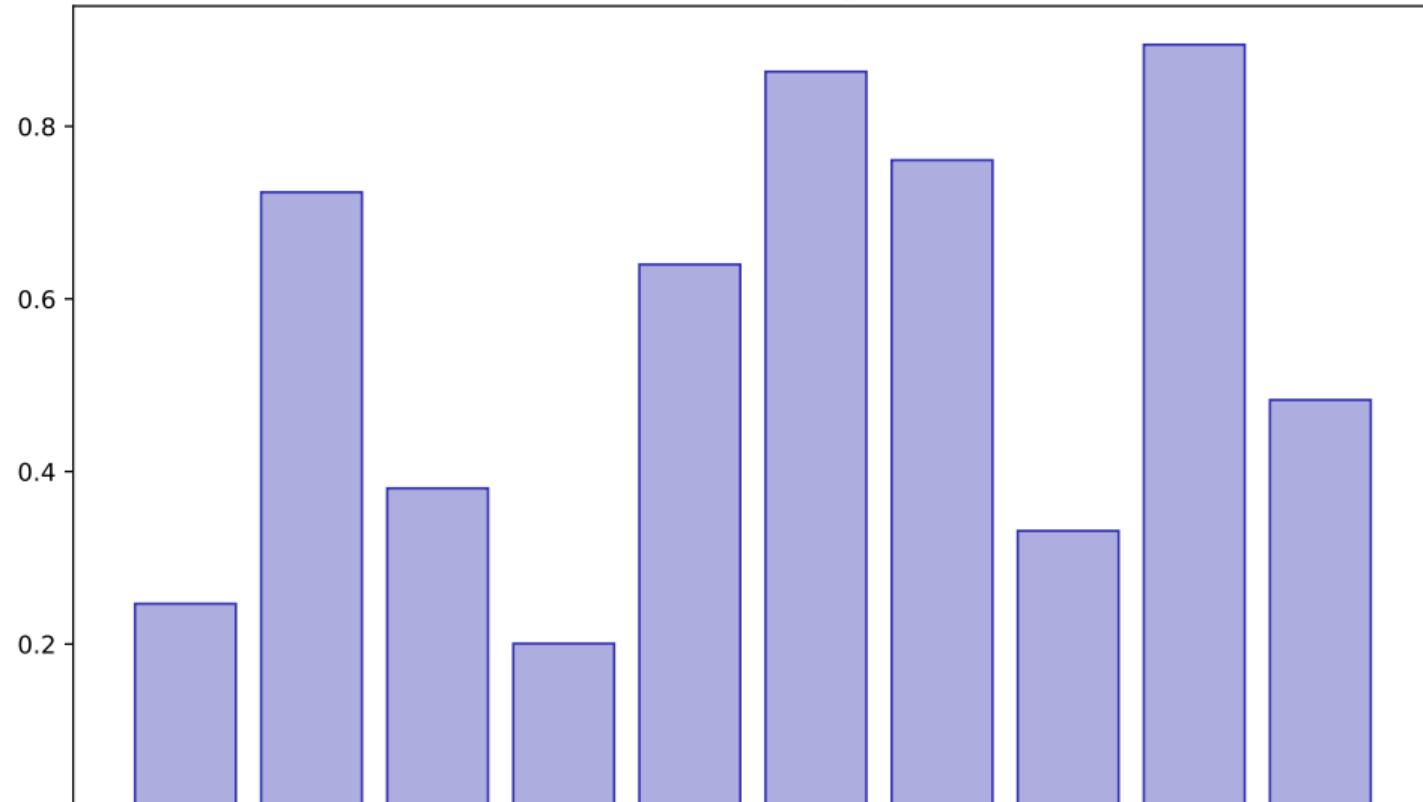
### Nested Plots



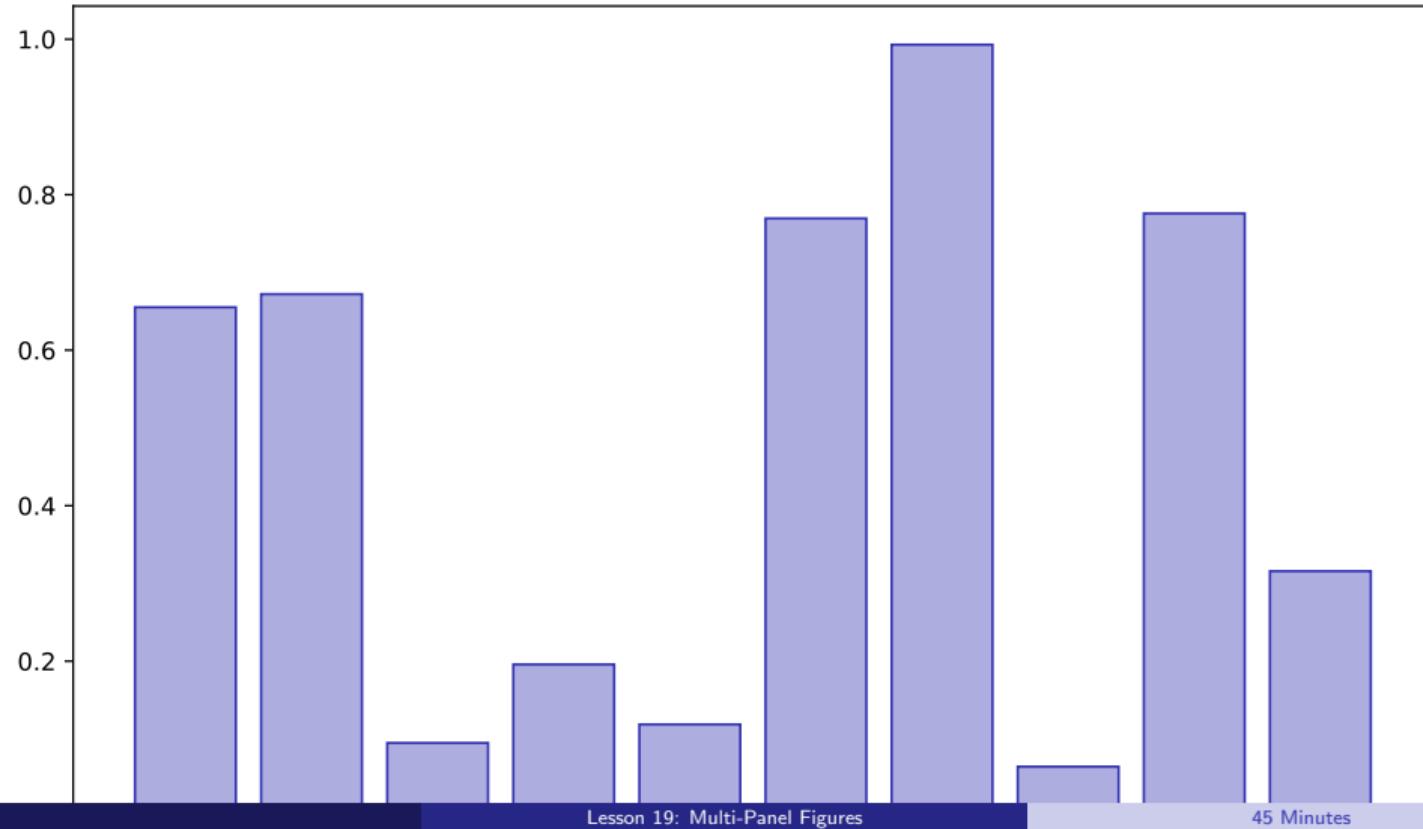
### Figure Sizing



## Dashboard Layout



## Finance Dashboard



# Lesson Summary

## Key Takeaways:

- Create subplots with plt.subplots()
- Arrange multiple visualizations
- Share axes and legends
- Build financial dashboards

Apply these skills in your final project

## Lesson 20: Data Storytelling

Data Science with Python – BSc Course

45 Minutes

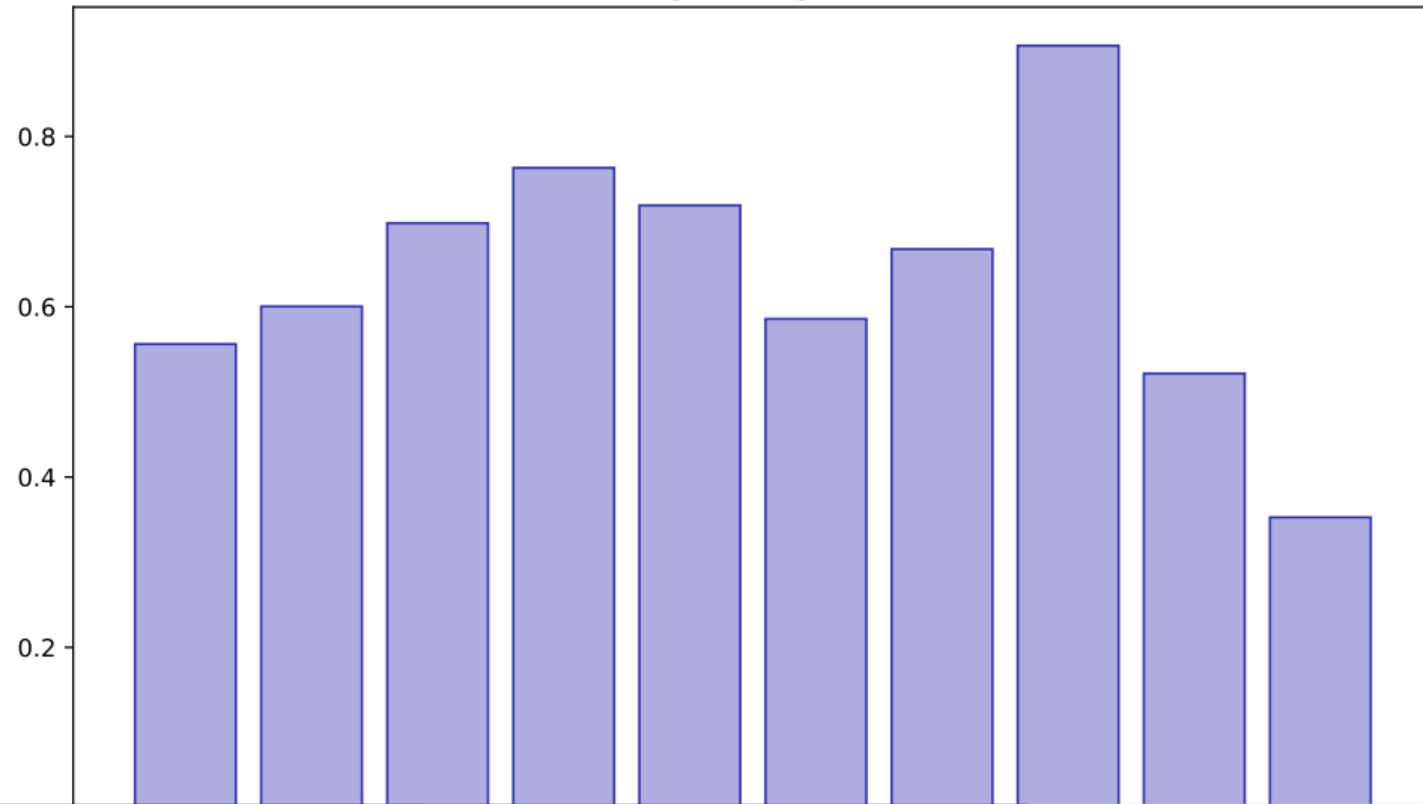
# Learning Objectives

**After this lesson, you will be able to:**

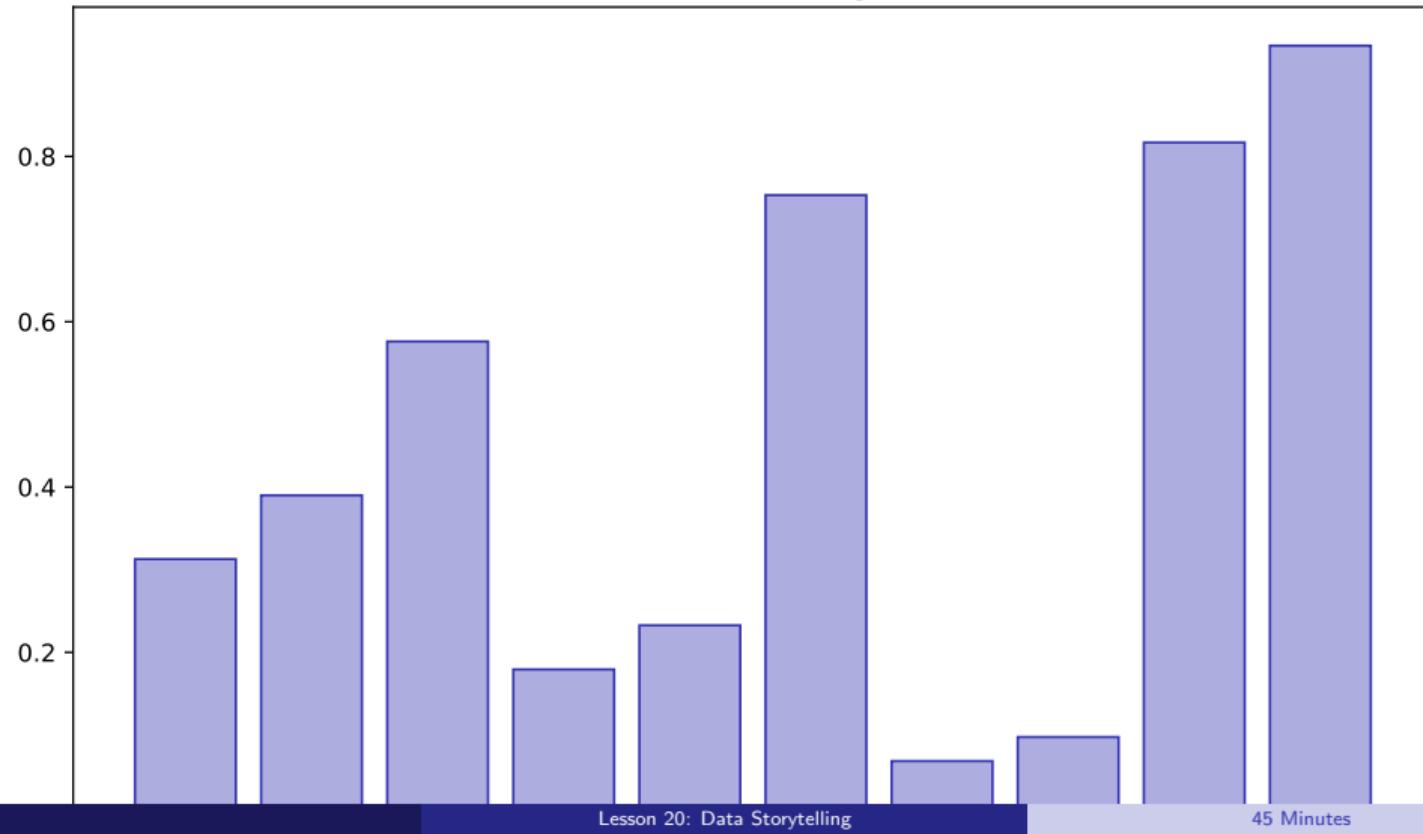
- Design visualizations for communication
- Apply color theory
- Create narrative flow
- Present to stakeholders

**Building towards your final project**

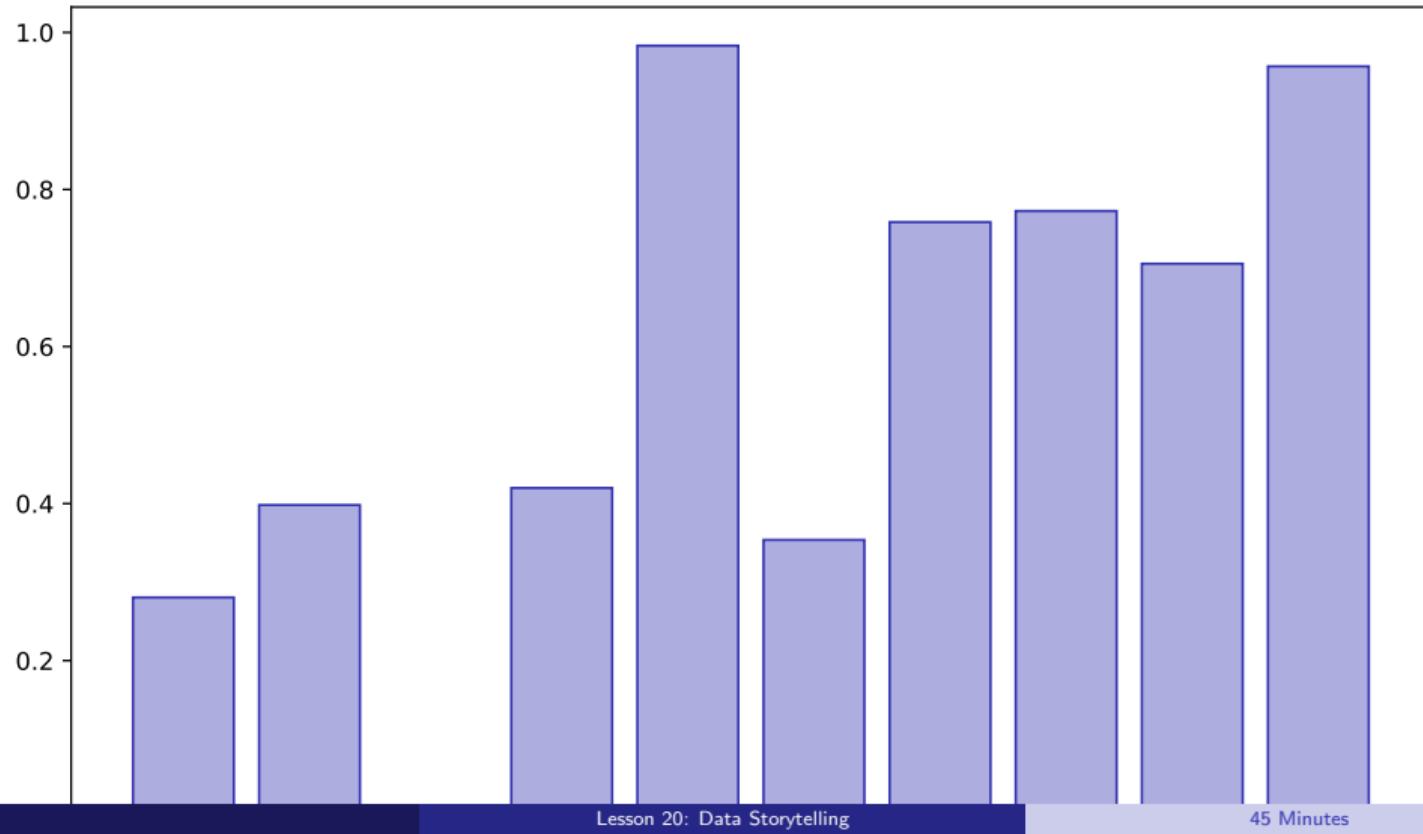
## Storytelling Flow



## Color Theory

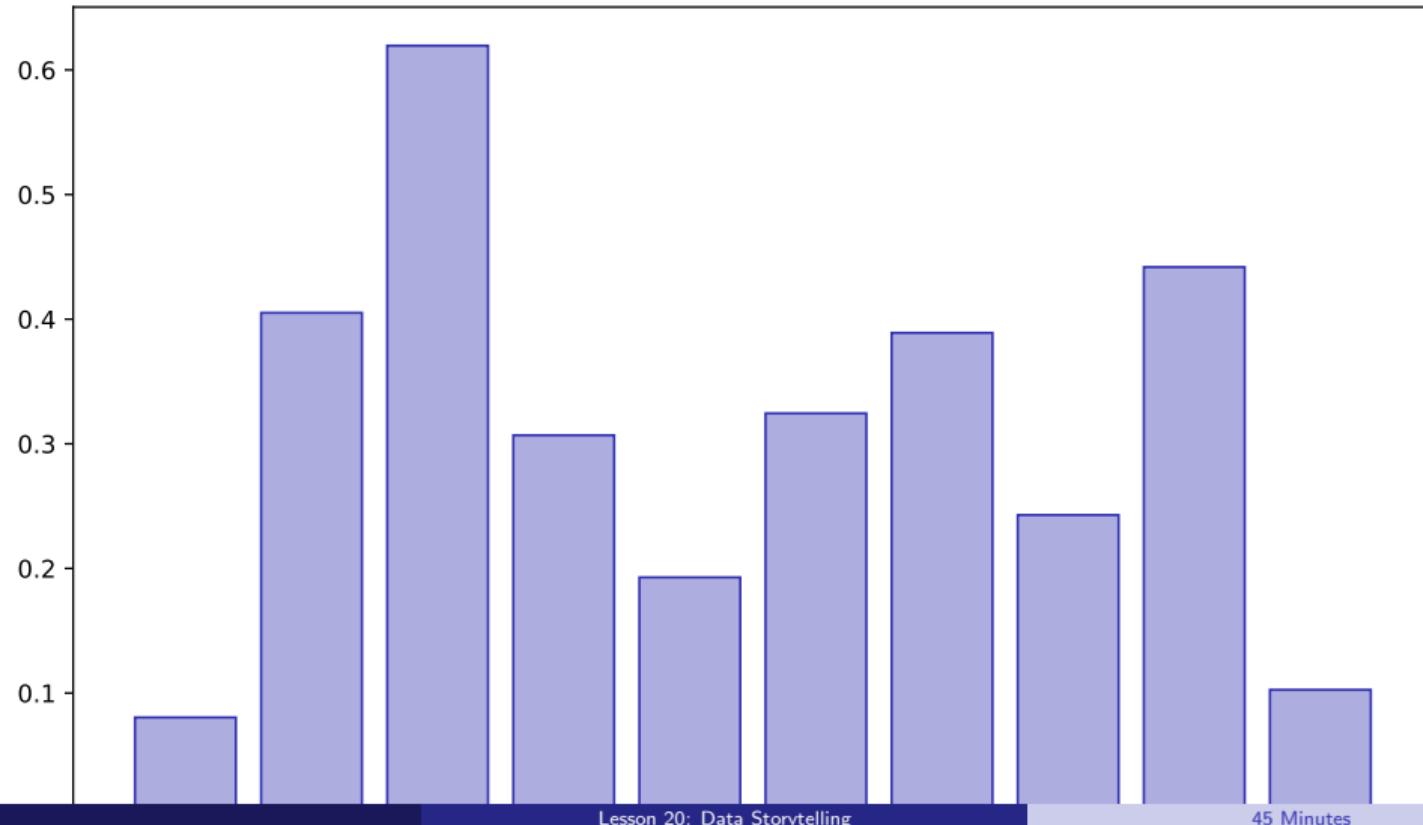


## Chart Selection

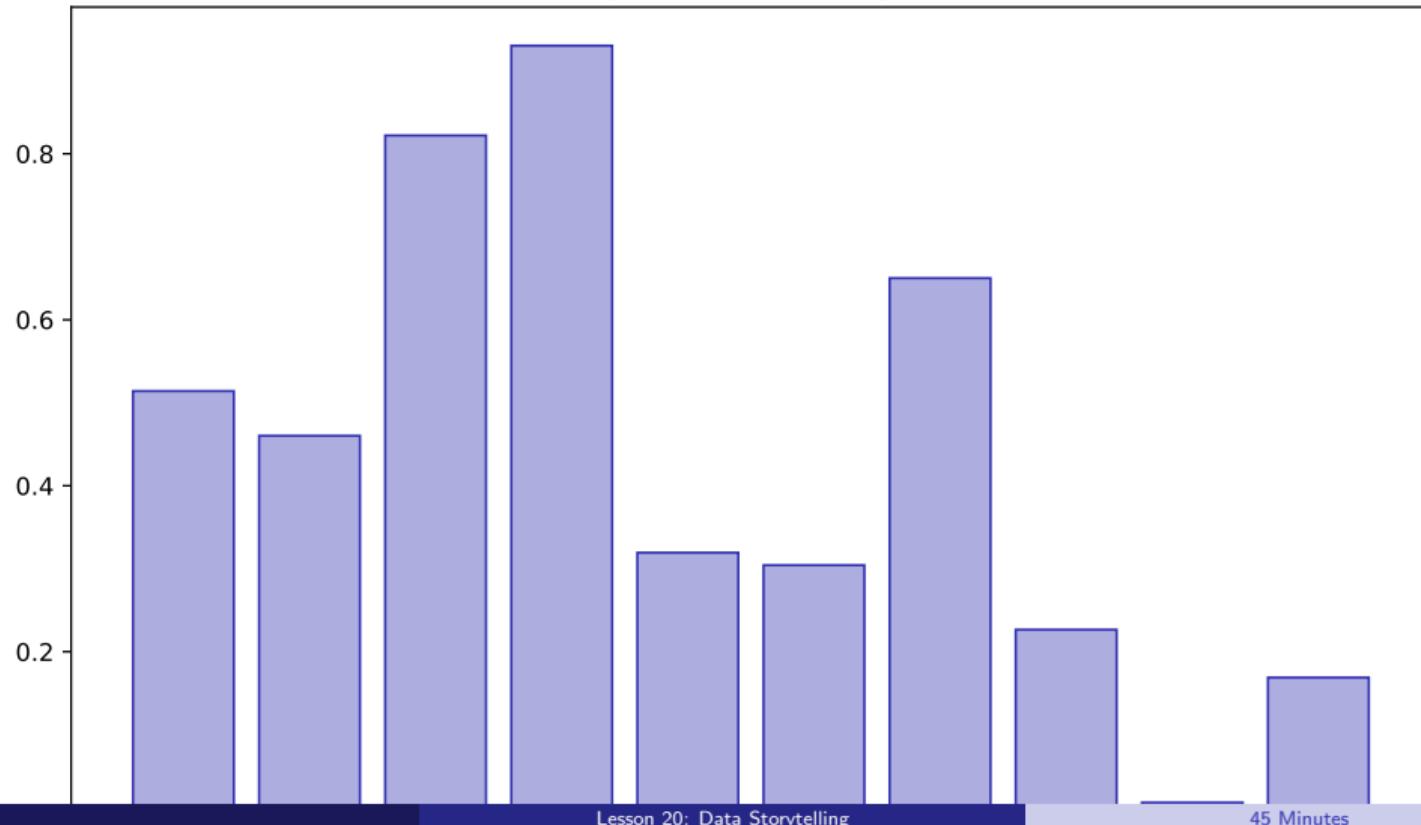


## Annotations

### Annotations

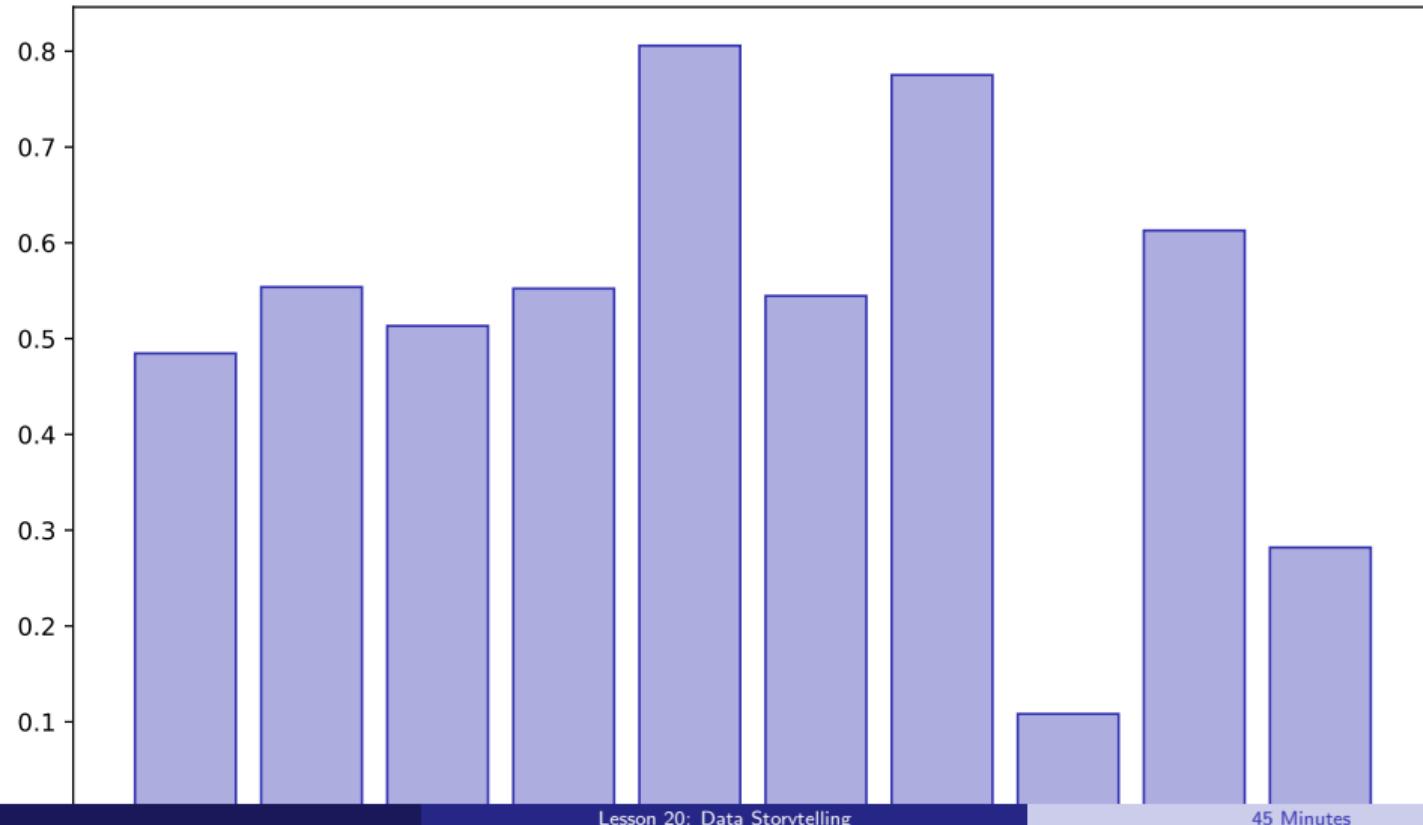


## Emphasis Techniques

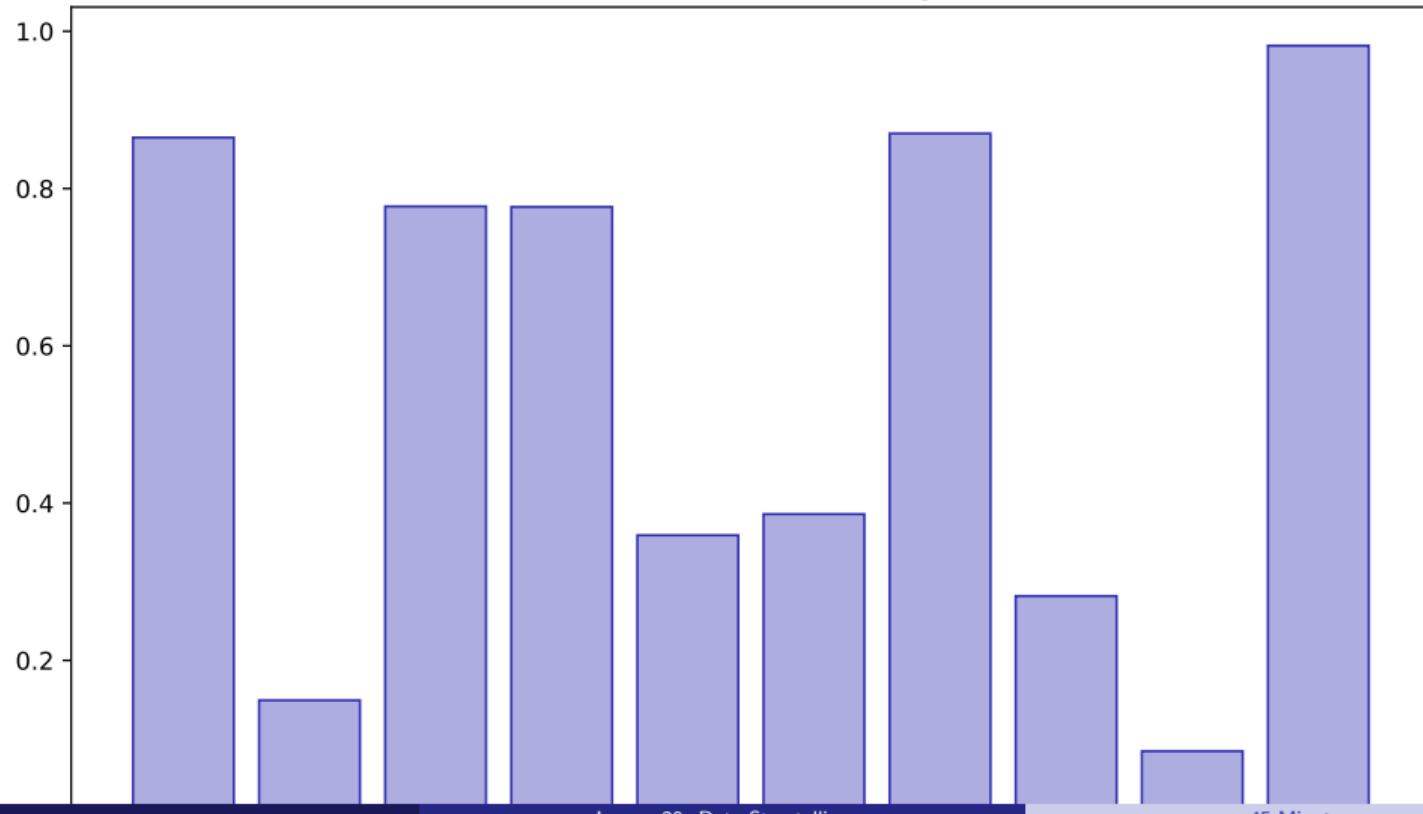


Before After

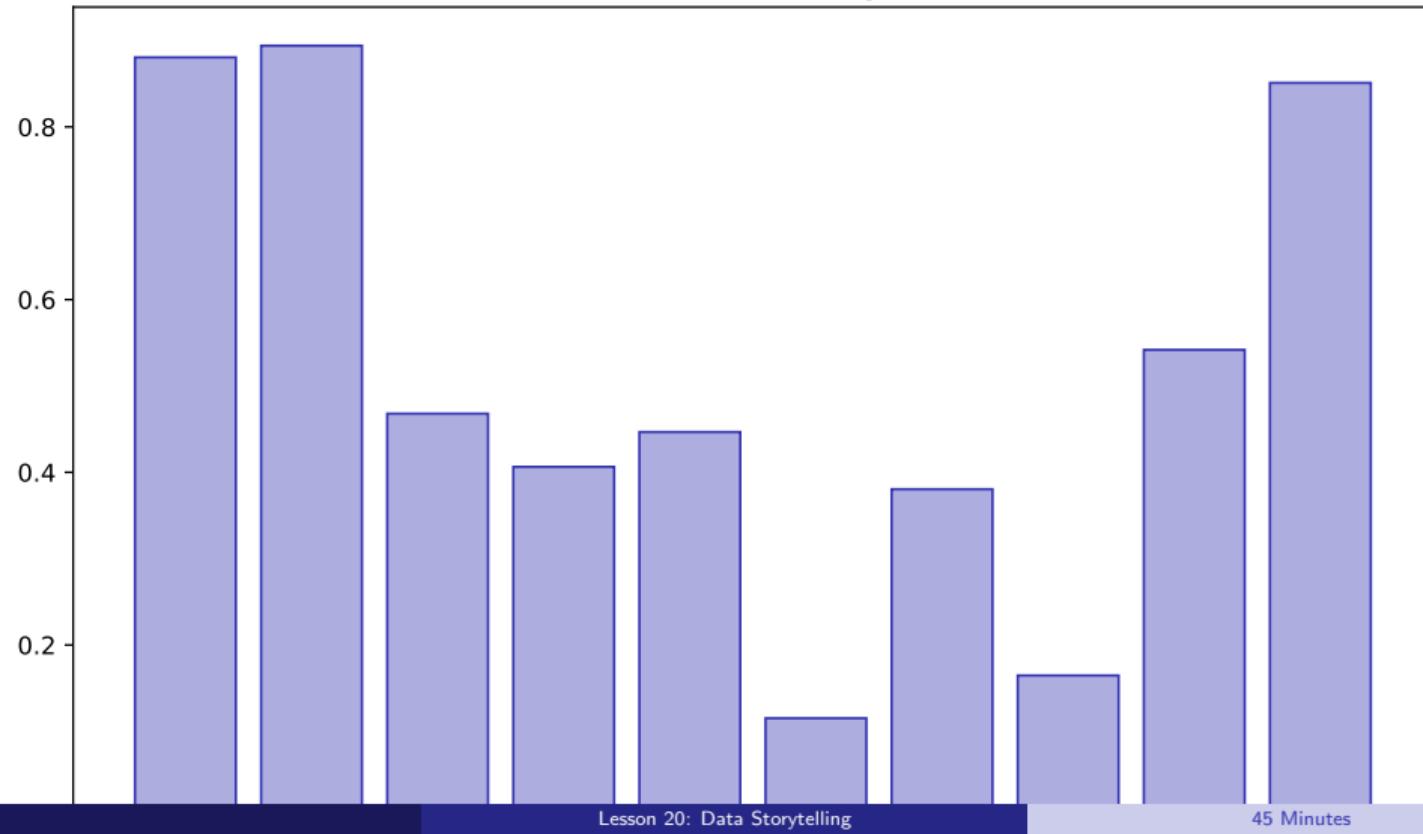
Before After



### Executive Summary



## Finance Story



## Key Takeaways:

- Design visualizations for communication
- Apply color theory
- Create narrative flow
- Present to stakeholders

Apply these skills in your final project

## Lesson 21: Linear Regression

Data Science with Python – BSc Course

45 Minutes

## Learning Objectives

**The Problem:** A portfolio manager needs to understand how stocks respond to market movements. How do we quantify systematic risk?

**After this lesson, you will be able to:**

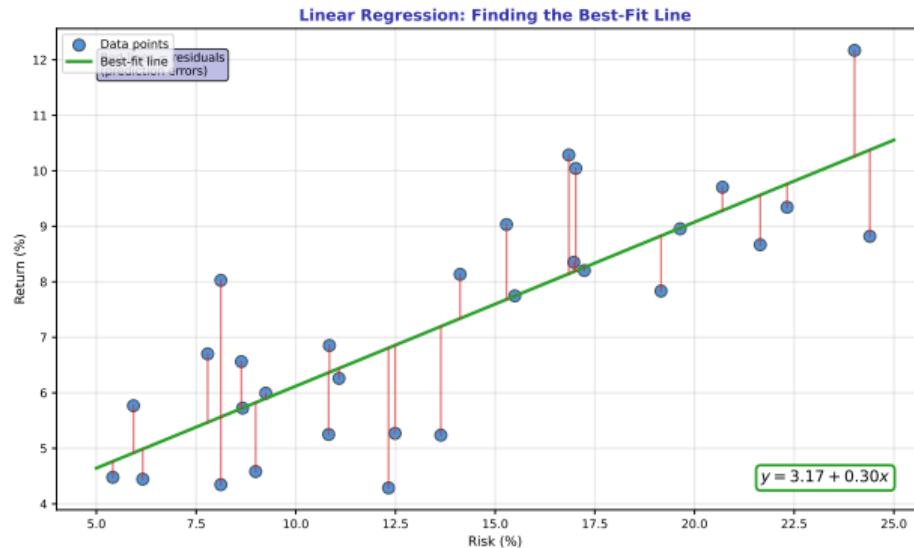
- Understand OLS estimation and the least squares principle
- Fit linear models using sklearn's LinearRegression
- Interpret coefficients (slope as beta, intercept as alpha)
- Estimate CAPM beta to classify stocks by risk profile

**Finance Application: Stock classification for portfolio construction**

# Regression Concept

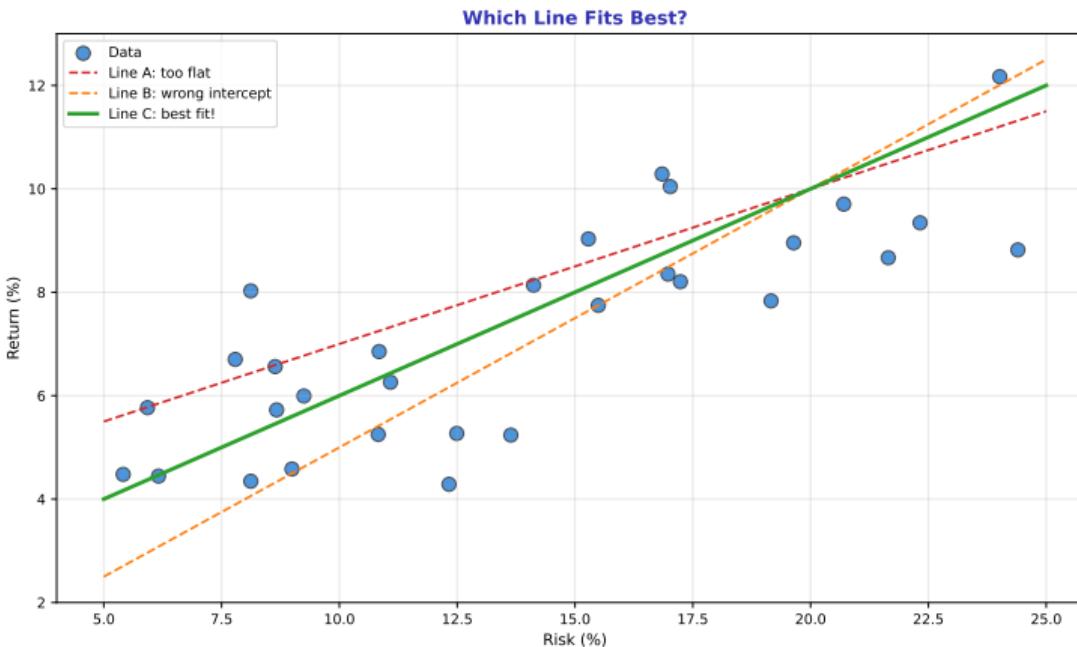
## Finding the Best-Fit Line

- Linear regression finds the line that best describes the relationship
- In finance: How does stock return respond to market return?



The "best" line minimizes the sum of squared vertical distances (residuals)

# Which Line Fits Best?

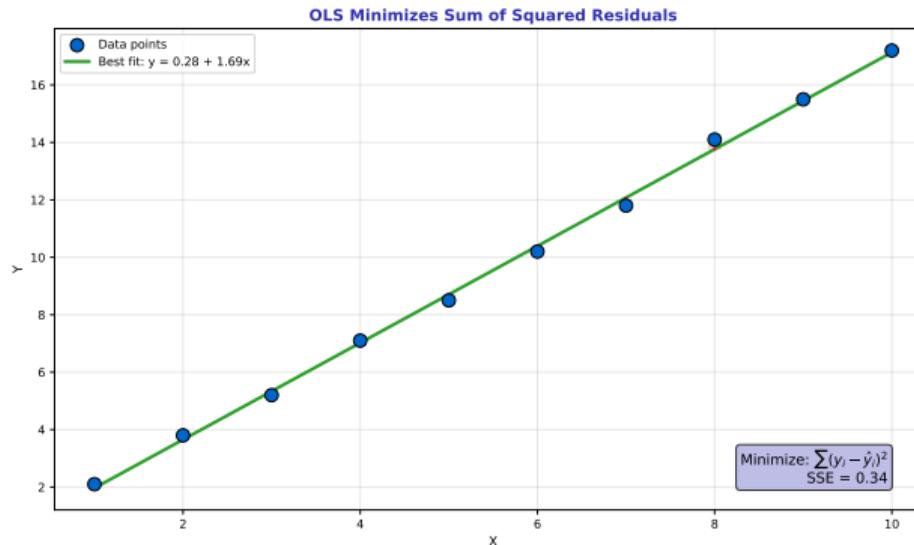


OLS finds the unique line that minimizes total squared error

# OLS Formula

## Ordinary Least Squares: The Math

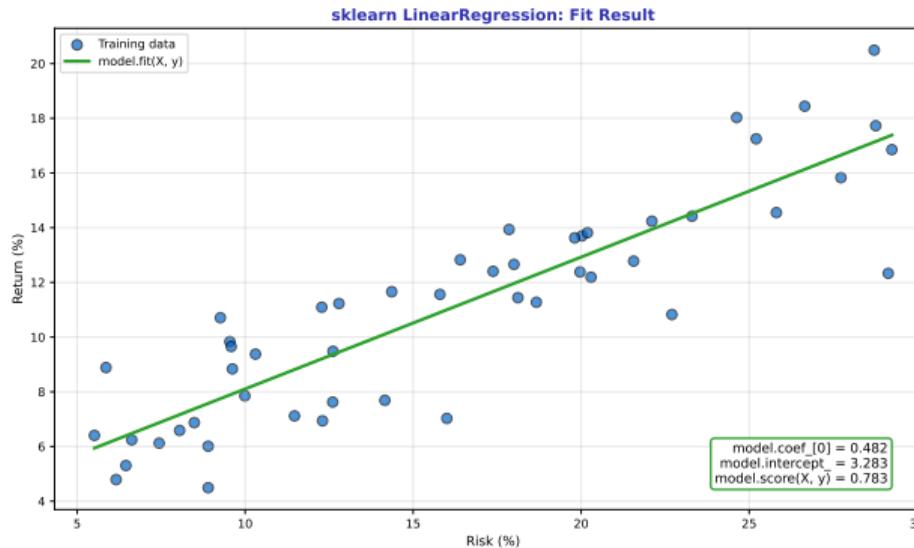
- Goal: Find  $\beta_0, \beta_1$  that minimize  $\sum(y_i - \hat{y}_i)^2$
- Solution:  $\beta_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$



Why squares? (1) Makes errors positive, (2) Penalizes large errors more

## Implementation in Python

- `from sklearn.linear_model import LinearRegression`
- `model = LinearRegression().fit(X, y)`
- Access: `model.coef_` (slope), `model.intercept_`

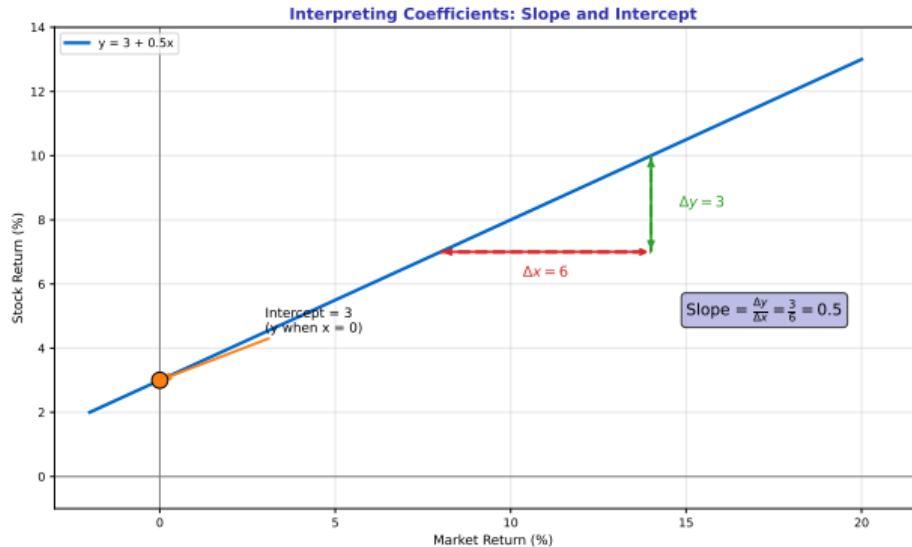


Pattern: `model.fit(X, y)` then `model.predict(X_new)` – works for all sklearn models

# Coefficient Interpretation

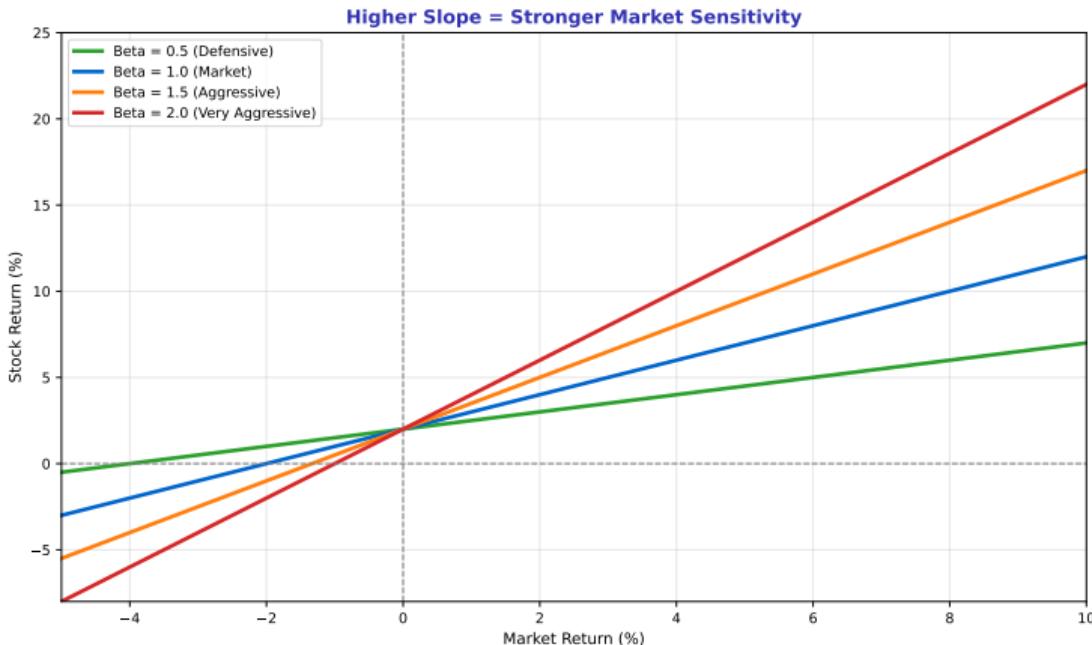
## What Do the Numbers Mean?

- **Slope ( $\beta_1$ ):** For each 1% market move, stock moves  $\beta_1$ %
- **Intercept ( $\beta_0$ ):** Stock's return when market return is zero



Finance translation: Slope = beta (systematic risk), Intercept = alpha (skill)

## Different Slopes = Different Betas

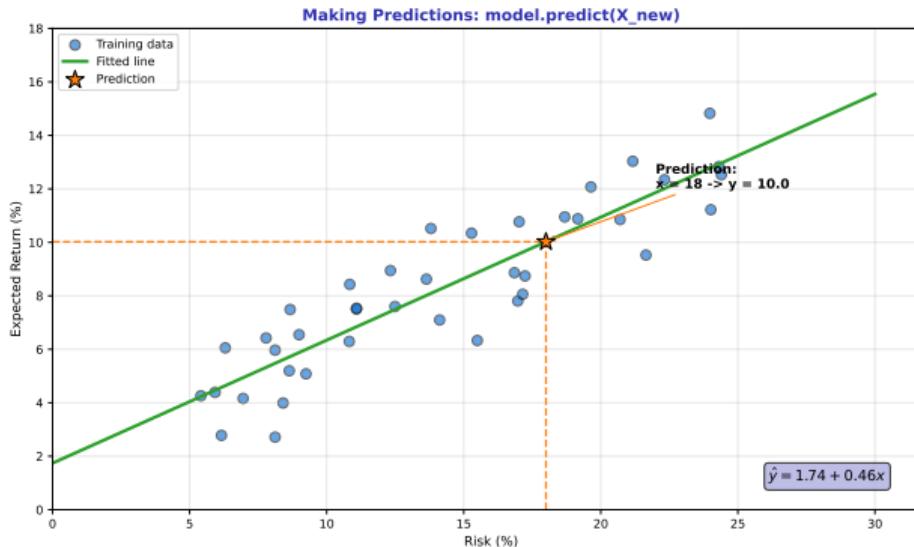


**Higher beta = stock amplifies market moves more**

# Making Predictions

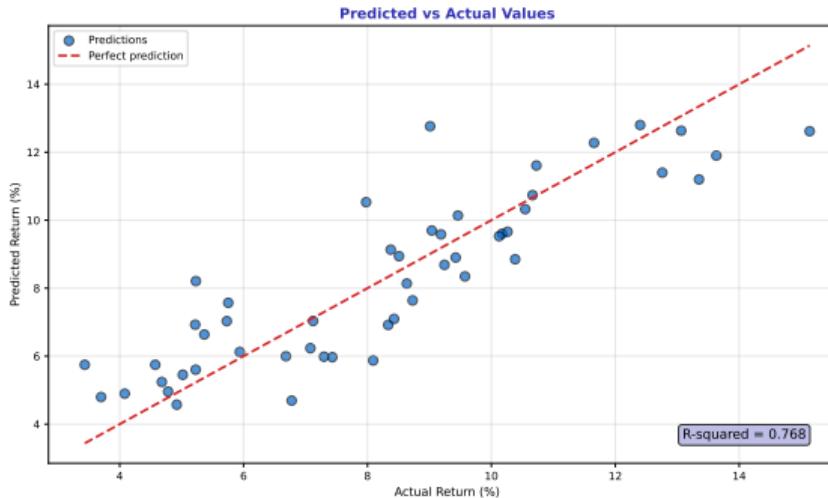
## Using the Model for Forecasting

- Once fitted, predict stock return for any market scenario
- `predicted = model.predict([[market_return]])`



Caution: Predictions assume the relationship stays stable

## Predicted vs Actual



Points on the diagonal = perfect predictions. R-squared measures fit quality.

# Extrapolation Warning

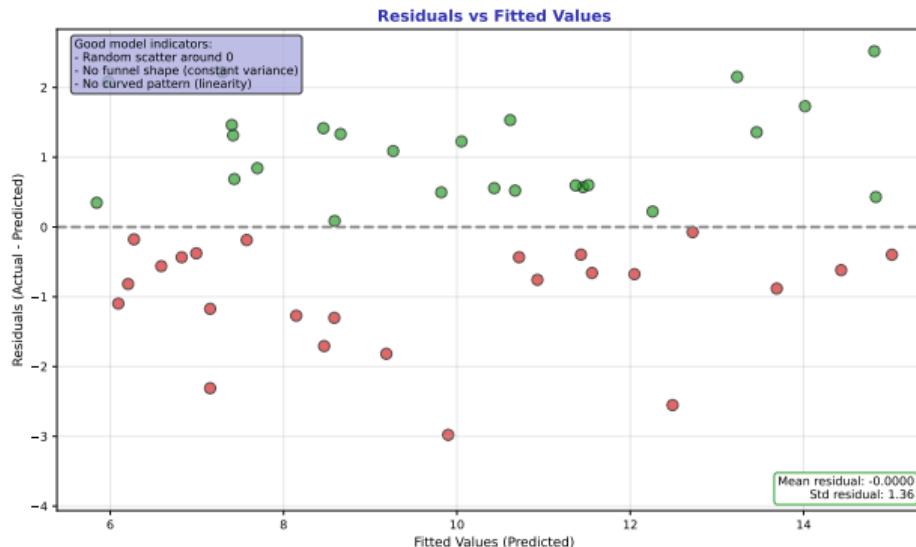


Never predict outside your training data range!

# Residuals

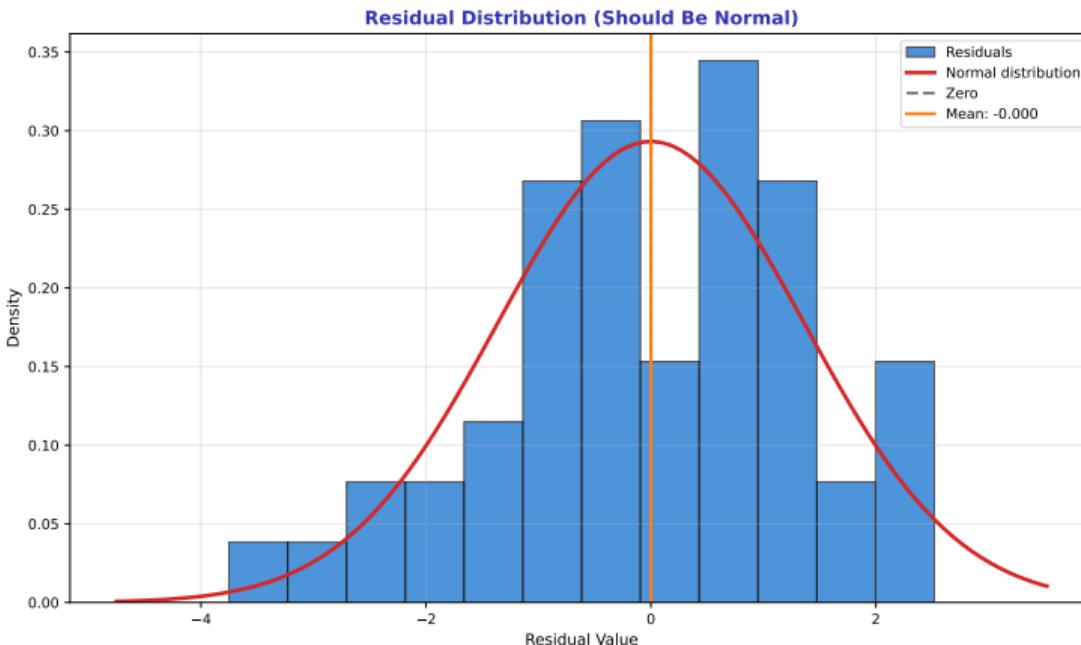
## Checking Prediction Quality

- Residual = Actual - Predicted ( $e_i = y_i - \hat{y}_i$ )
- Good model: residuals should be random (no pattern)



Plot residuals vs predicted: if you see a pattern, the model is missing something

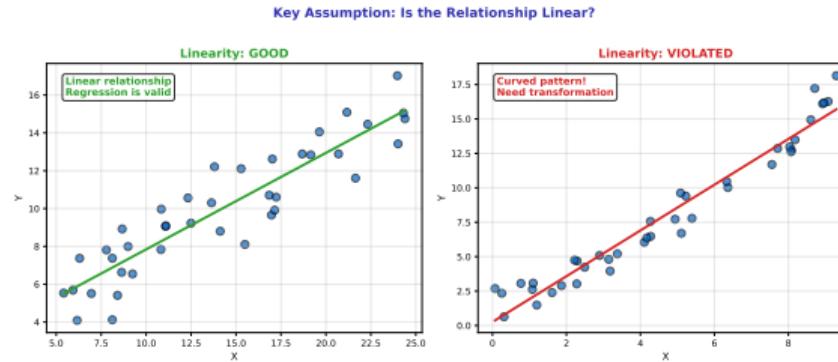
# Residual Distribution



**Normality assumption:** residuals should follow a bell curve centered at zero

## When Does Linear Regression Work?

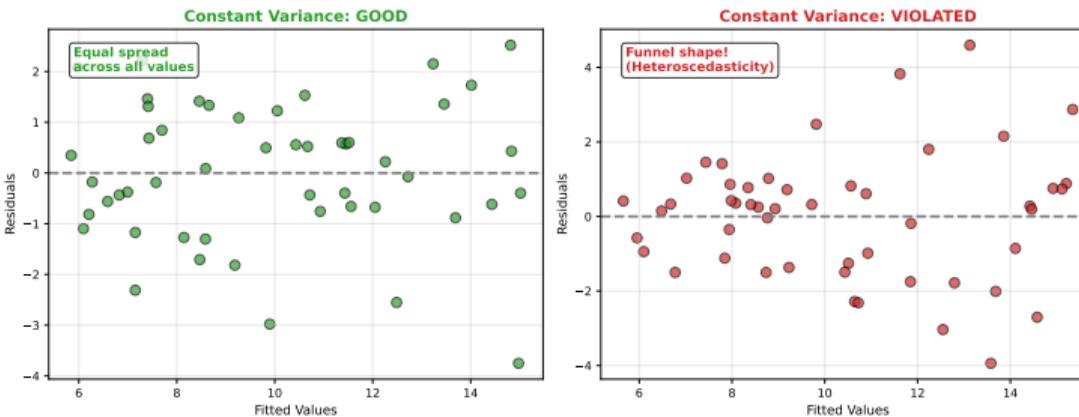
- **Linearity:** Relationship is actually linear (not curved)
- **Homoscedasticity:** Variance of errors is constant
- **Normality:** Residuals are normally distributed



Finance reality: Stock returns often violate these – check residuals

# Homoscedasticity Check

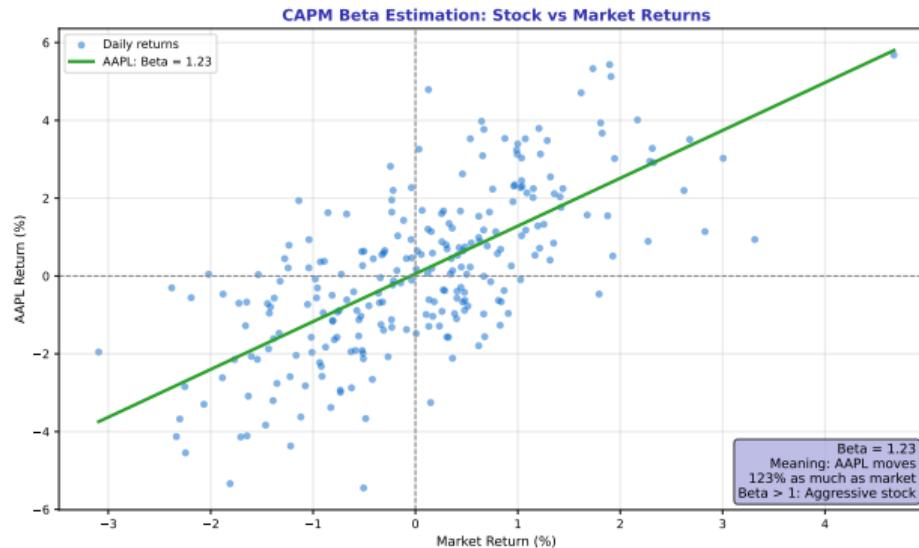
Homoscedasticity Check: Is Variance Constant?



Funnel shape = heteroscedasticity. Fix: weighted least squares or log transform

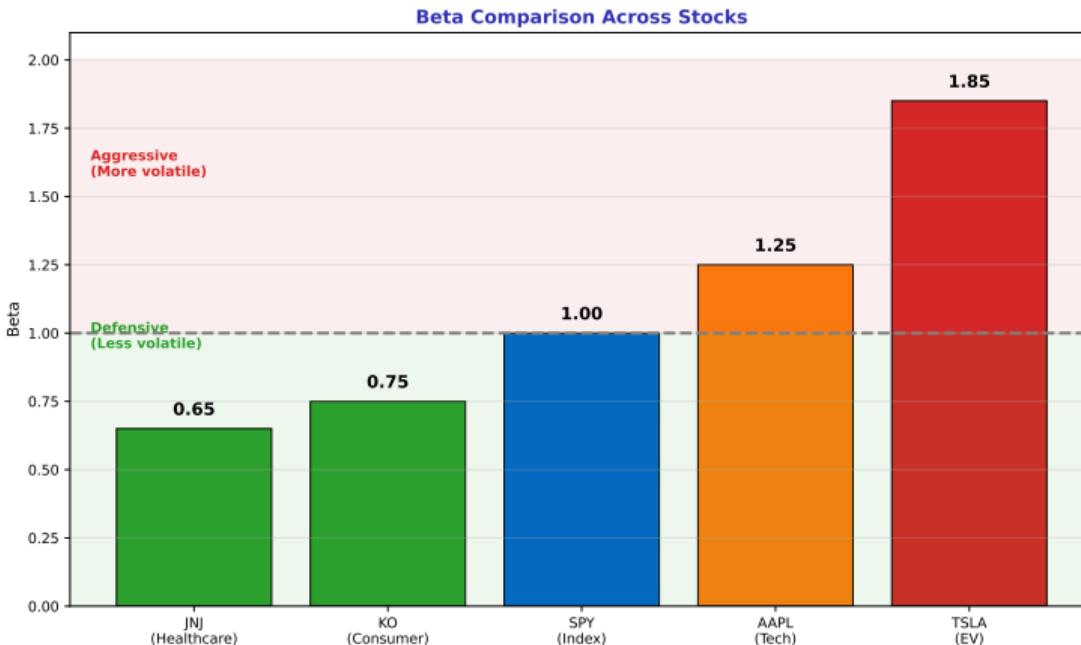
## The Solution: Stock Classification by Systematic Risk

- **Beta > 1:** Aggressive stock – amplifies market moves
- **Beta < 1:** Defensive stock – dampens volatility



Alpha ( $\beta_0$ ): Outperformance after risk adjustment

## Beta Comparison



Mix defensive (low beta) and aggressive (high beta) stocks based on risk tolerance

## Hands-On Exercise (25 min)

### Task: Estimate Beta for Your Favorite Stock

- ① Download 1 year of daily returns for a stock (e.g., MSFT) and SPY
- ② Fit: `model.fit(spy_returns, stock_returns)`
- ③ Extract and interpret: What is the beta? What is the alpha?
- ④ Plot the regression line with actual data points

**Deliverable:** Scatter plot with regression line, annotated with beta value.

**Extension:** Compare beta estimates using different time periods (1yr vs 5yr)

## Lesson Summary

**Problem Solved:** We can now quantify systematic risk using CAPM beta via linear regression.

**Key Takeaways:**

- OLS finds the line that minimizes squared errors
- sklearn: `LinearRegression().fit(X, y)` – three lines of code
- Slope = beta (market sensitivity), Intercept = alpha (skill)

**Next Lesson:** Regularization (L22) – what happens with too many features?

**Memory:** Beta = slope of stock vs market regression. High beta = high volatility.

## Lesson 22: Regularization

Data Science with Python – BSc Course

45 Minutes

**The Problem:** With many predictors, our model can memorize noise instead of learning patterns. How do we build models that generalize to unseen data?

**After this lesson, you will be able to:**

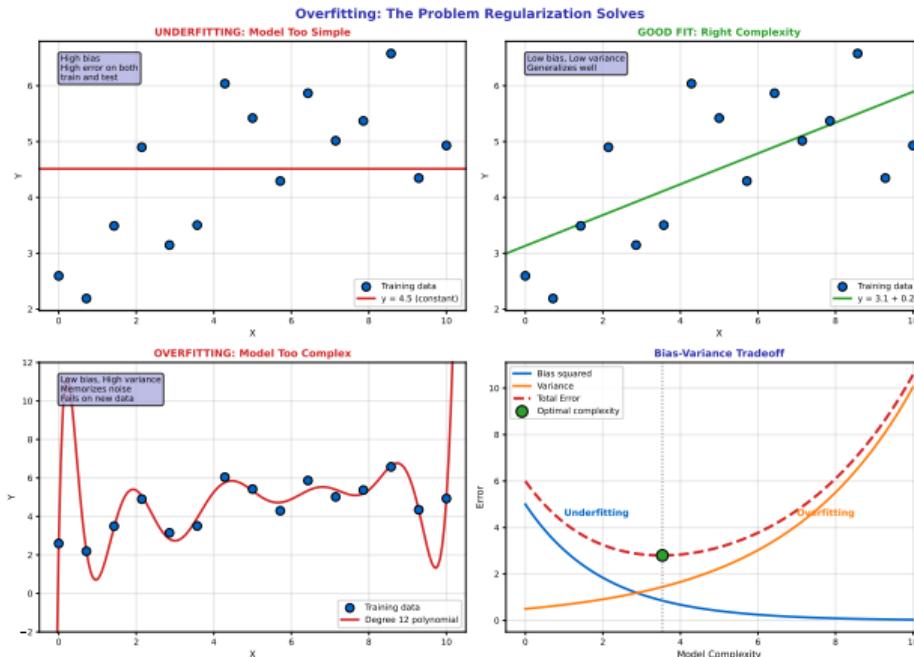
- Recognize overfitting and its causes
- Apply Ridge (L2) regularization to shrink coefficients
- Apply Lasso (L1) for automatic feature selection
- Tune the regularization strength with cross-validation

**Finance Application: Building robust factor models with many correlated predictors**

# The Overfitting Problem

## When Models Memorize Instead of Learn

- High training accuracy but poor test performance
- Complex models fit noise in the training data

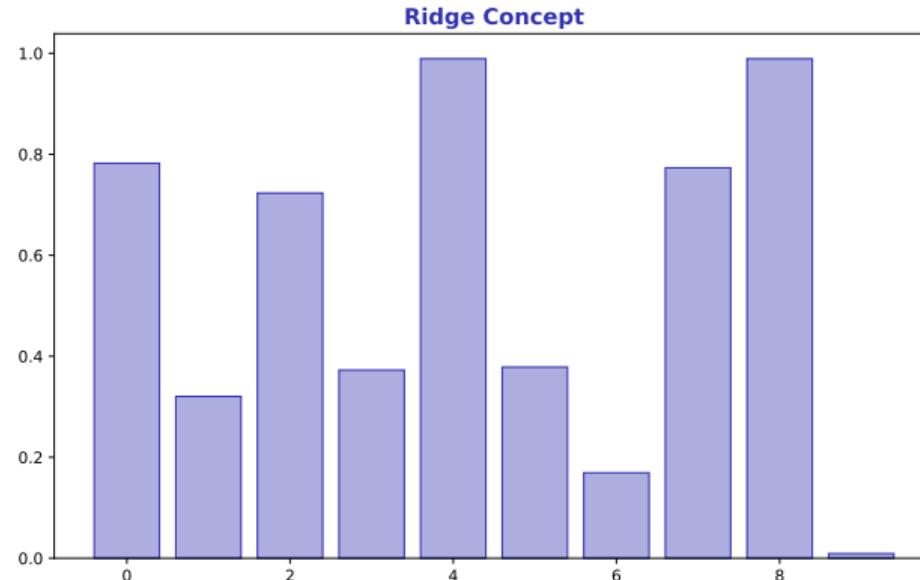


Red flag: If train error keeps dropping but test error rises, you're overfitting

# Ridge Regression (L2)

## Shrink All Coefficients Toward Zero

- Add penalty:  $\text{Loss} = \sum(y - \hat{y})^2 + \lambda \sum \beta_j^2$
- Large  $\lambda$  = stronger shrinkage, simpler model

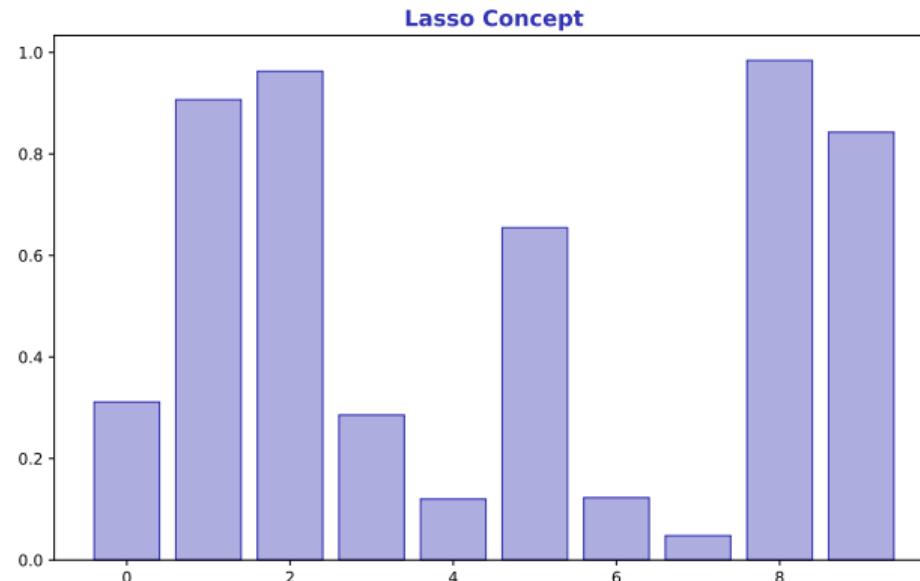


Ridge keeps all features but reduces their influence – good for multicollinearity

# Lasso Regression (L1)

## Some Coefficients Go to Exactly Zero

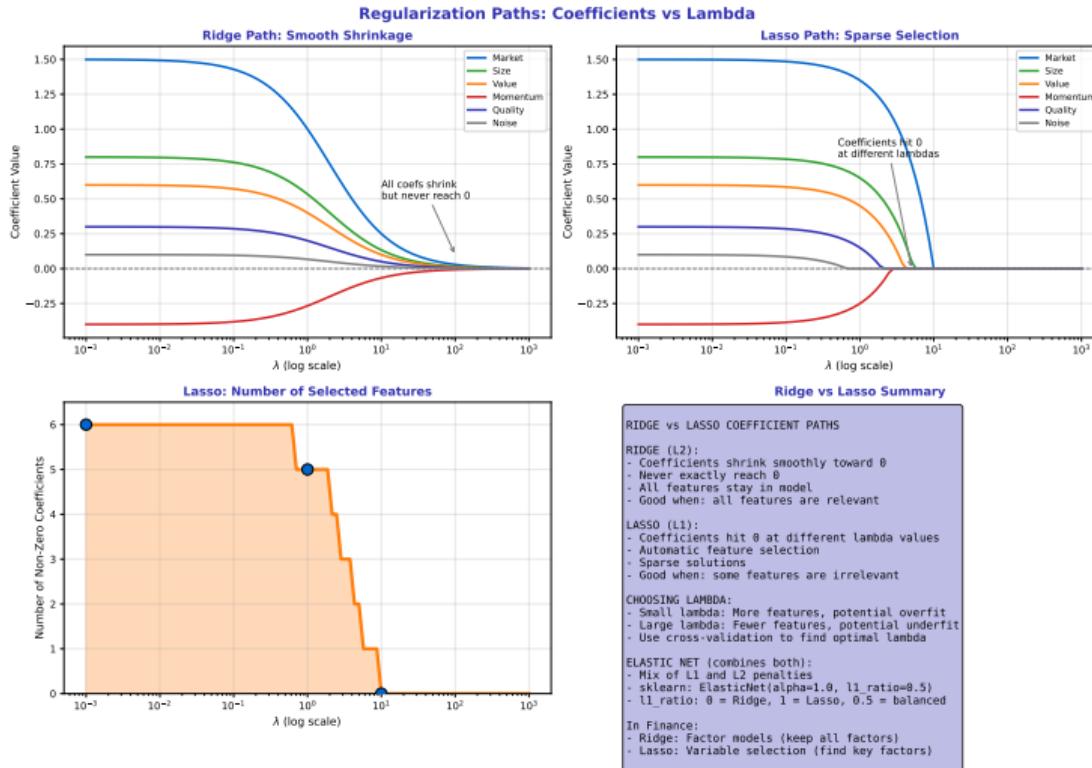
- Add penalty:  $\text{Loss} = \sum(y - \hat{y})^2 + \lambda \sum |\beta_j|$
- L1 penalty creates sparse solutions (automatic feature selection)



Lasso eliminates irrelevant features – use when you suspect many predictors are useless

# Coefficient Paths

## How Coefficients Change with Lambda

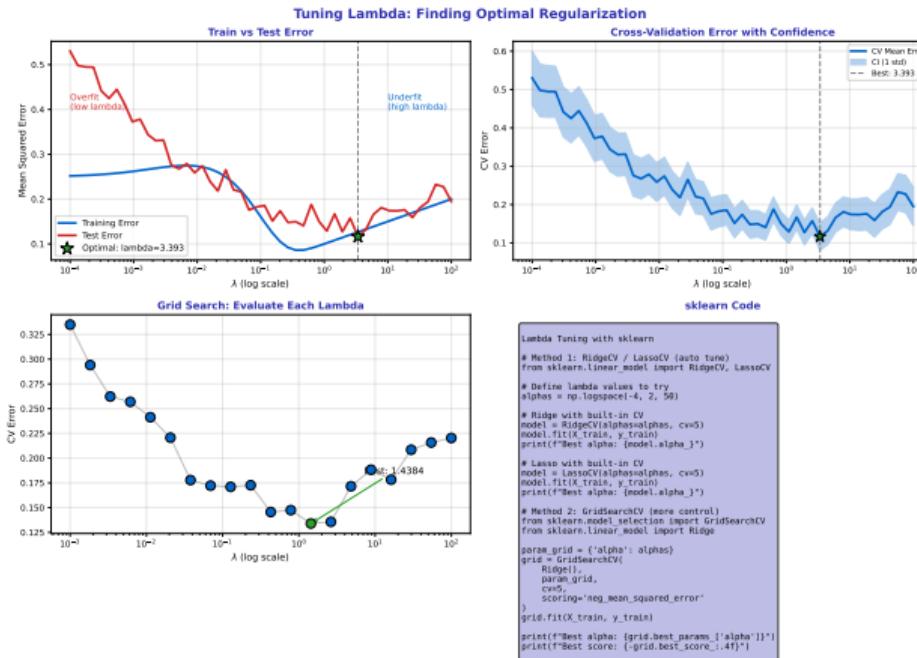


As  $\lambda$  increases, coefficients shrink. Lasso drives some to zero. Ridge does not.

# Tuning Lambda

## Finding the Right Penalty Strength

- Too small: overfitting (model too complex)
- Too large: underfitting (model too simple)

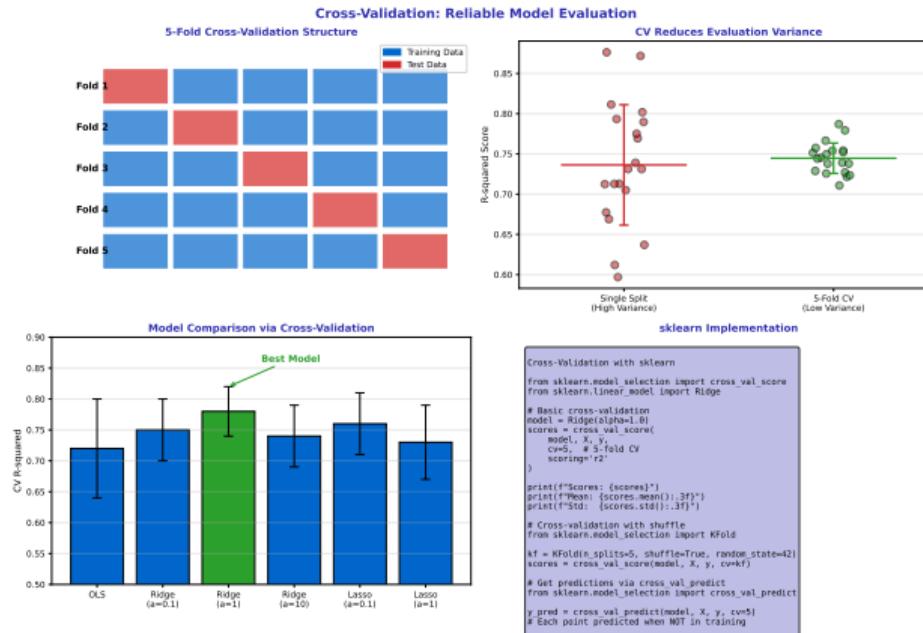


Use cross-validation to find the lambda that minimizes test error

# Cross-Validation for Lambda

## sklearn Makes It Easy

- RidgeCV and LassoCV automatically search lambda values
- K-fold CV: split data K ways, train on K-1, test on 1, average

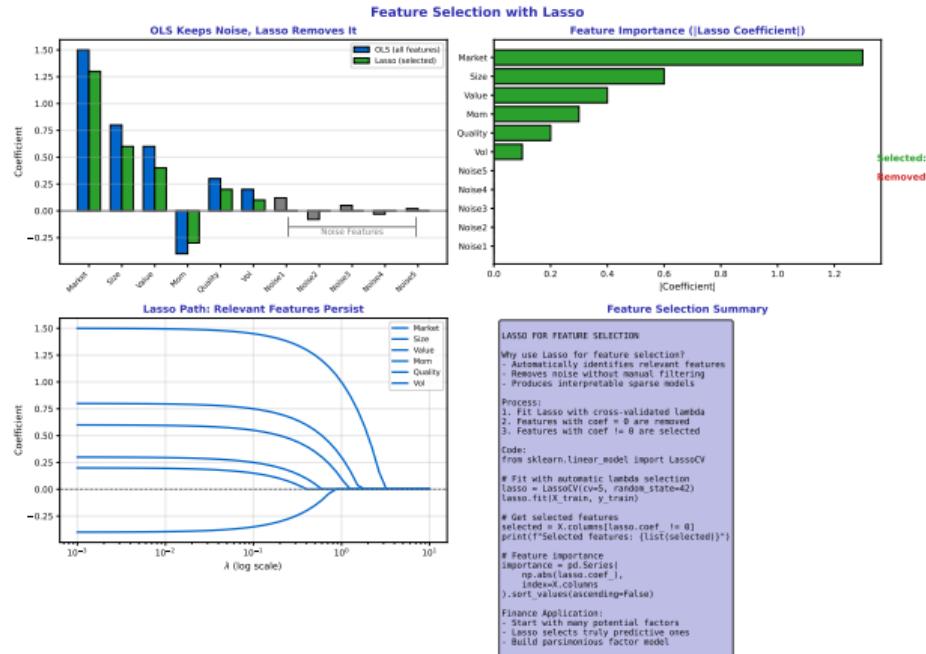


Rule: Use `RidgeCV(alphas=[0.1, 1, 10, 100])` to search logarithmically

# Feature Selection with Lasso

## Which Predictors Actually Matter?

- Non-zero Lasso coefficients = selected features
- Zero coefficients = features eliminated by the model

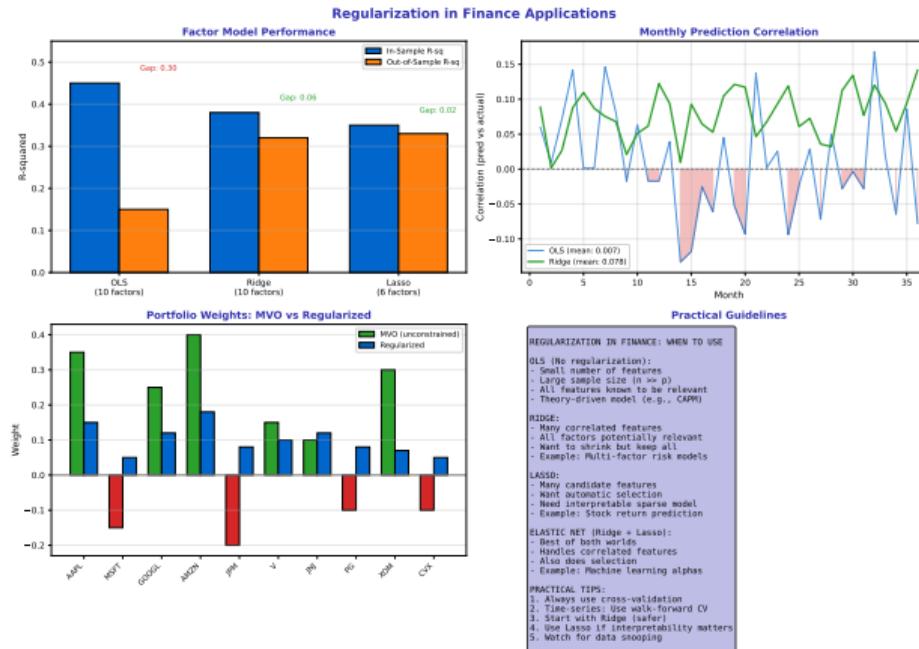


Finance insight: Lasso often keeps 3-5 factors from a candidate set of 20+

# Finance Application: Factor Models

## Building Robust Return Predictions

- Many candidate factors are correlated (value, quality, momentum...)
- Regularization prevents unstable, extreme factor weights



Industry practice: Regularized regression for combining alpha signals

## Hands-On Exercise (25 min)

### Task: Compare Ridge vs Lasso on Multi-Factor Data

- ① Create synthetic data with 20 features (only 5 are truly predictive)
- ② Fit OLS, Ridge, and Lasso models
- ③ Compare test set  $R^2$  for each model
- ④ Plot Lasso coefficients – which features were selected?

**Deliverable:** Bar chart of coefficients comparing OLS vs Ridge vs Lasso.

**Extension:** Use LassoCV to find optimal lambda and report selected features

## Lesson Summary

**Problem Solved:** Regularization prevents overfitting when we have many predictors or limited data.

**Key Takeaways:**

- Ridge (L2) shrinks all coefficients – handles multicollinearity
- Lasso (L1) sets some coefficients to zero – automatic feature selection
- Use cross-validation (RidgeCV, LassoCV) to tune  $\lambda$

**Next Lesson:** Regression Metrics (L23) – how do we measure model quality?

**Memory:** Ridge = Ridge keeps all features. Lasso = Lasso Loses features (L for Lose).

## Lesson 23: Regression Metrics

Data Science with Python – BSc Course

45 Minutes

## Learning Objectives

**The Problem:** We've built a regression model, but how good is it? How do we compare different models objectively?

**After this lesson, you will be able to:**

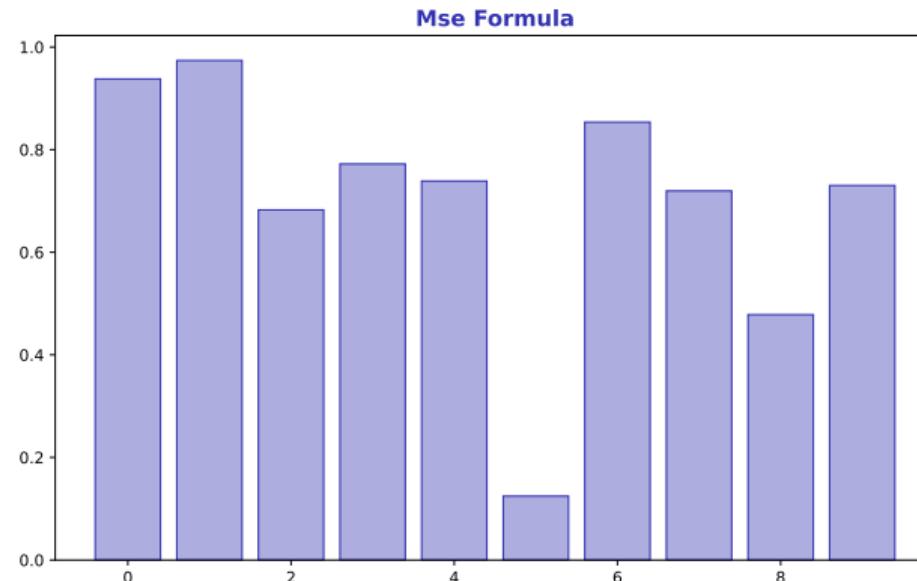
- Calculate MSE, RMSE, and MAE for prediction quality
- Interpret  $R^2$  as variance explained
- Use adjusted  $R^2$  to penalize model complexity
- Apply time series cross-validation for financial data

**Finance Application:** Evaluating return prediction models before deployment

# Mean Squared Error (MSE)

## The Foundation Metric

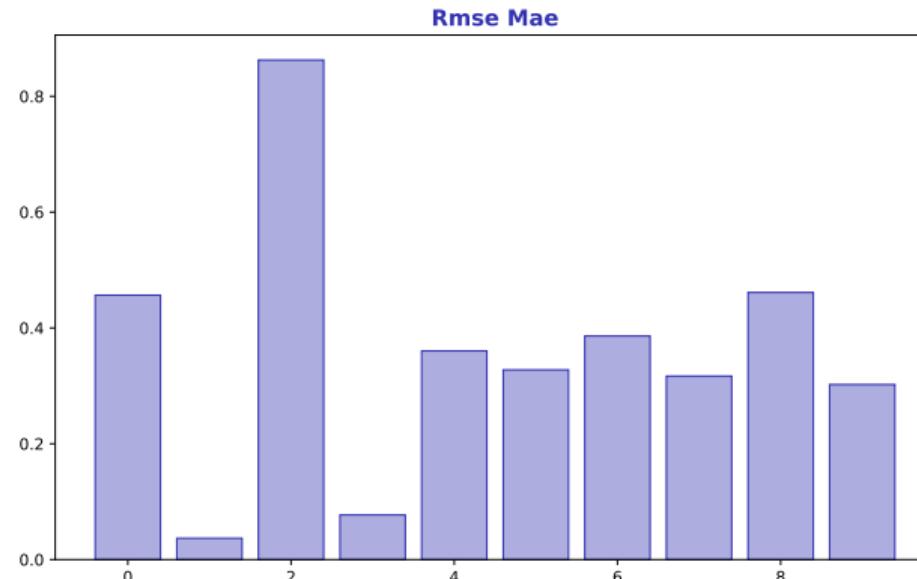
- $MSE = \frac{1}{n} \sum (y_i - \hat{y}_i)^2$
- Units are squared (e.g., dollars squared) – hard to interpret directly



MSE penalizes large errors heavily due to squaring – sensitive to outliers

## Interpretable Error Measures

- $\text{RMSE} = \sqrt{\text{MSE}}$  – same units as  $y$ , penalizes large errors
- $\text{MAE} = \frac{1}{n} \sum |y_i - \hat{y}_i|$  – average absolute error, robust to outliers

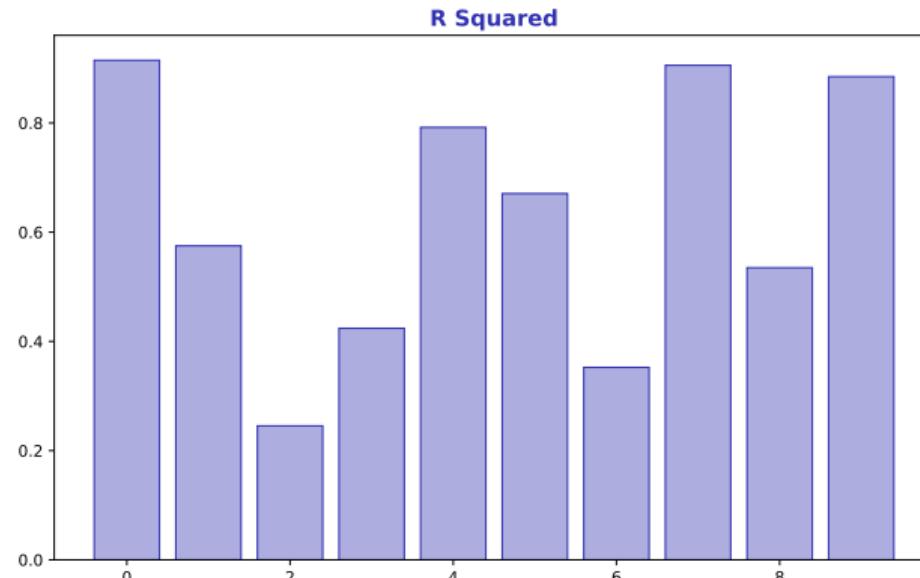


Rule:  $\text{RMSE} \geq \text{MAE}$  always. If  $\text{RMSE} \gg \text{MAE}$ , you have large outliers.

# R-Squared ( $R^2$ )

## Proportion of Variance Explained

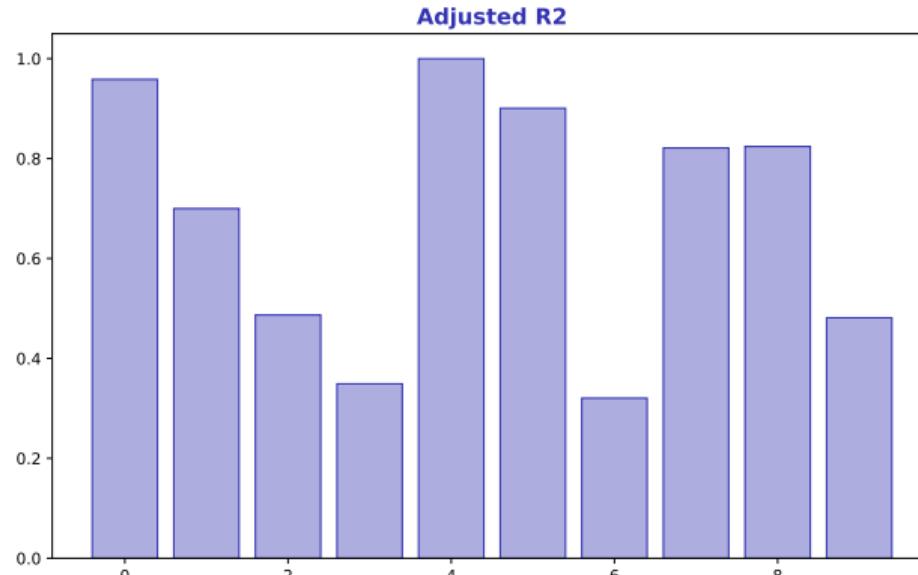
- $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$  – ranges from 0 to 1
- $R^2 = 0.7$  means model explains 70% of variance in  $y$



Warning:  $R^2$  always increases with more features – can be misleading

## Penalizing Model Complexity

- Adjusted  $R^2 = 1 - \frac{(1-R^2)(n-1)}{n-p-1}$  where  $p = \text{number of features}$
- Only increases if new feature improves fit more than expected by chance

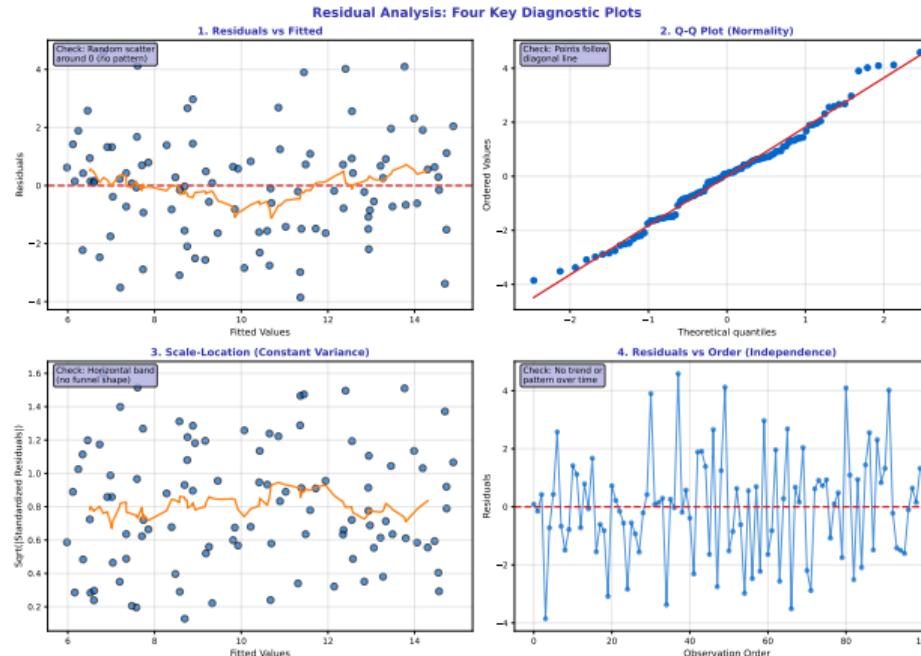


Use adjusted  $R^2$  when comparing models with different numbers of features

# Residual Analysis

## Checking Model Assumptions

- Plot residuals vs fitted values – should show no pattern
- Patterns indicate missing nonlinearity or heteroscedasticity

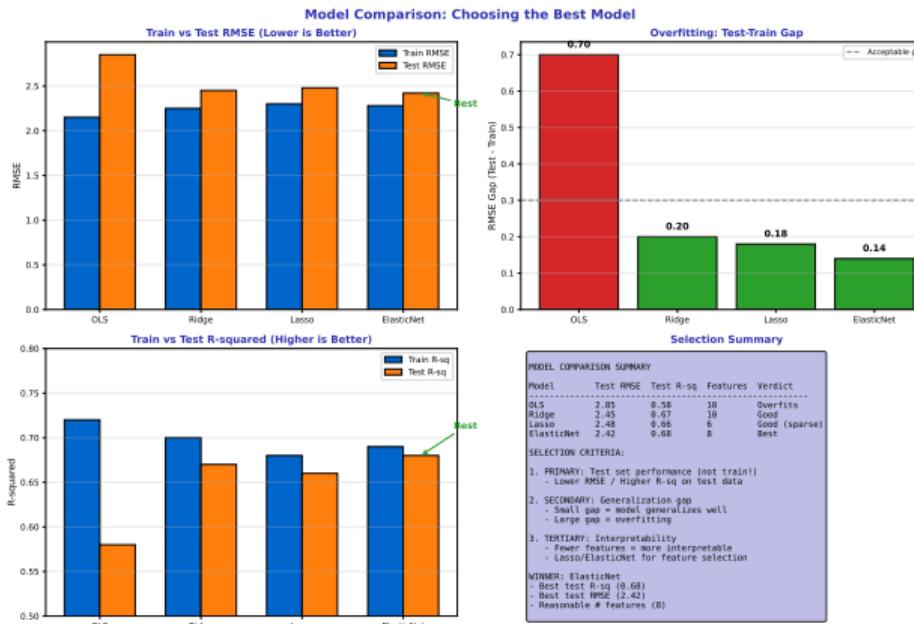


Good practice: Always plot residuals before trusting your model

# Model Comparison

## Fair Comparison Requires Same Data

- Compare on held-out test set, not training set
- Use cross-validation for robust comparison

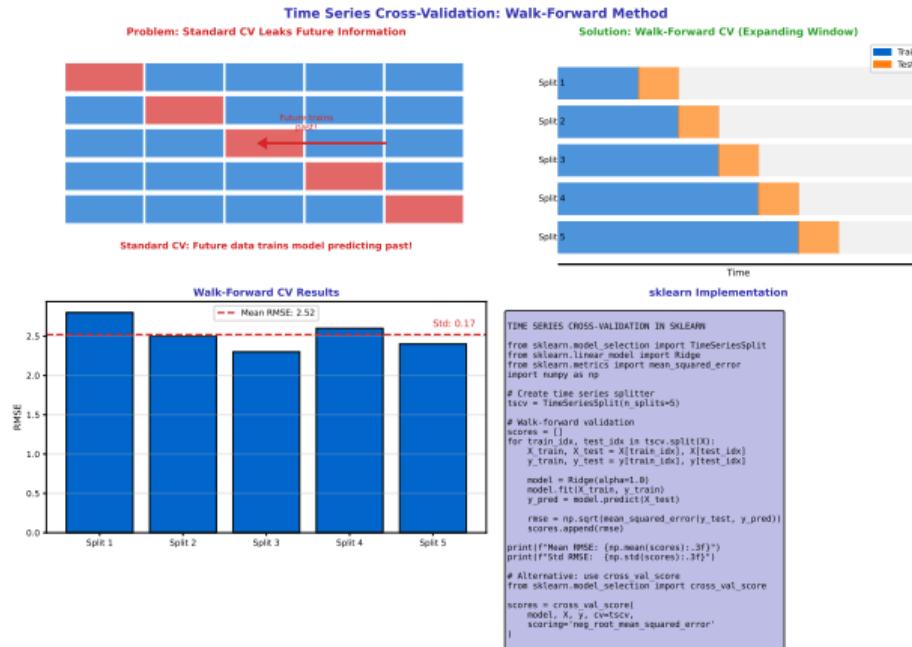


Never compare training  $R^2$  values – always use test set performance

# Time Series Cross-Validation

## Respecting Temporal Order

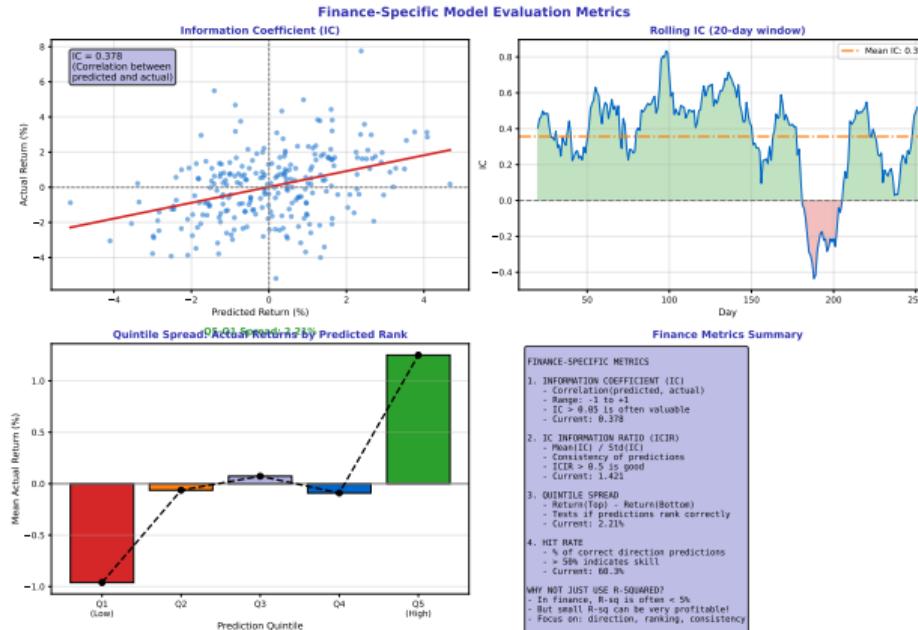
- Standard CV shuffles data – invalid for time series (future leaks into past)
- Rolling window: train on  $[t_1, t_n]$ , test on  $[t_{n+1}, t_{n+k}]$



Finance rule: Never train on data from the future – use `TimeSeriesSplit`

## Beyond Standard Regression Metrics

- Information Coefficient (IC): correlation between predicted and actual returns
- Hit Rate: percentage of correct direction predictions



In trading, predicting direction matters more than magnitude

## Hands-On Exercise (25 min)

### Task: Evaluate a Stock Return Prediction Model

- ① Fit a linear regression predicting next-day returns from lagged features
- ② Calculate MSE, RMSE, MAE, and  $R^2$  on a test set
- ③ Plot residuals vs predicted values – any patterns?
- ④ Use TimeSeriesSplit for proper cross-validation

**Deliverable:** Summary table of metrics + residual plot.

**Extension:** Calculate Information Coefficient and compare to  $R^2$

## Lesson Summary

**Problem Solved:** We can now objectively measure and compare regression model quality.

**Key Takeaways:**

- RMSE/MAE: interpretable error in original units
- $R^2$ : proportion of variance explained (0 to 1)
- Adjusted  $R^2$ : penalizes unnecessary complexity
- Time series requires special validation (no data leakage)

**Next Lesson:** Factor Models (L24) – multi-factor return prediction

**Memory:** RMSE punishes outliers, MAE treats all errors equally,  $R^2$  is % explained

## Lesson 24: Factor Models

Data Science with Python – BSc Course

45 Minutes

**The Problem:** CAPM uses only market beta, but stocks also respond to size, value, and momentum. How do we capture multiple sources of systematic risk?

**After this lesson, you will be able to:**

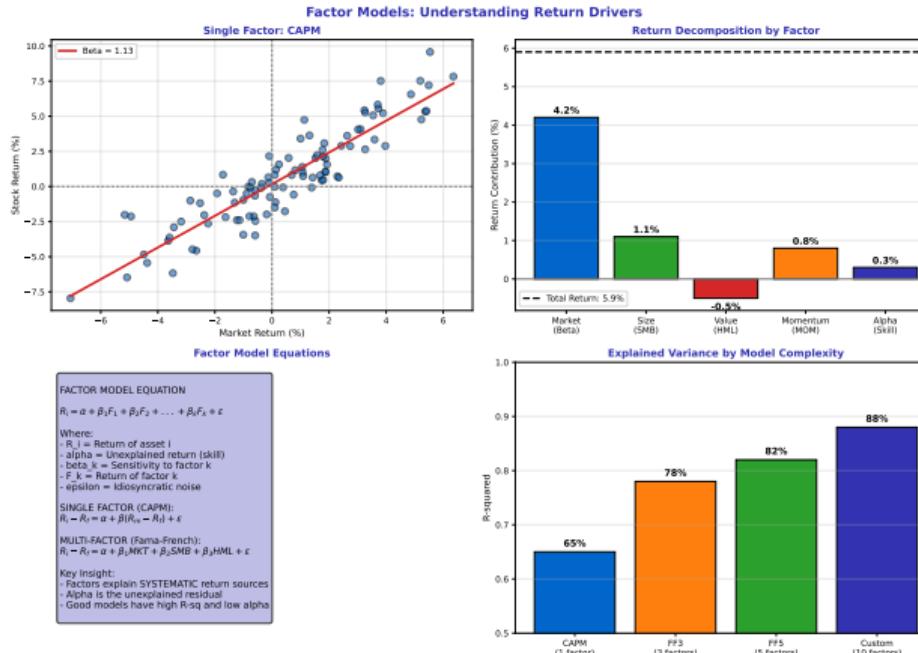
- Build multi-factor regression models
- Understand Fama-French factors (SMB, HML)
- Interpret factor loadings and alpha
- Create complete ML pipelines with sklearn

**Finance Application:** Decomposing returns into systematic factors for attribution

# Factor Concept

## From Single to Multiple Risk Sources

- CAPM:  $R_i - R_f = \alpha + \beta_M(R_M - R_f)$  – only market risk
- Multi-factor: Add size, value, momentum as additional explanatory variables

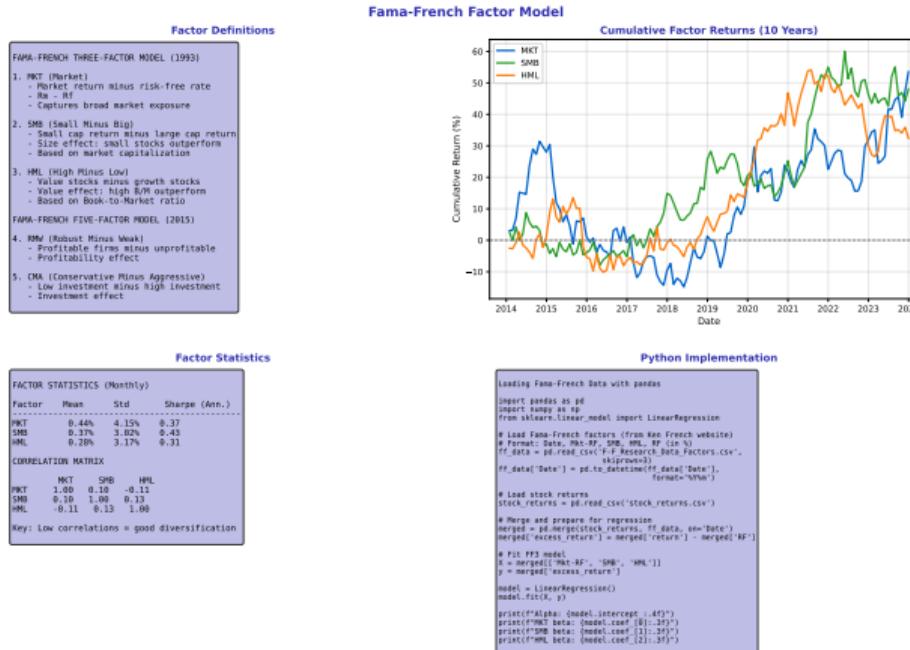


Key insight: Different stocks have different exposures to different risk factors

# Fama-French Factors

## The Classic Three-Factor Model

- **SMB** (Small Minus Big): Small caps outperform large caps historically
- **HML** (High Minus Low): Value stocks outperform growth stocks

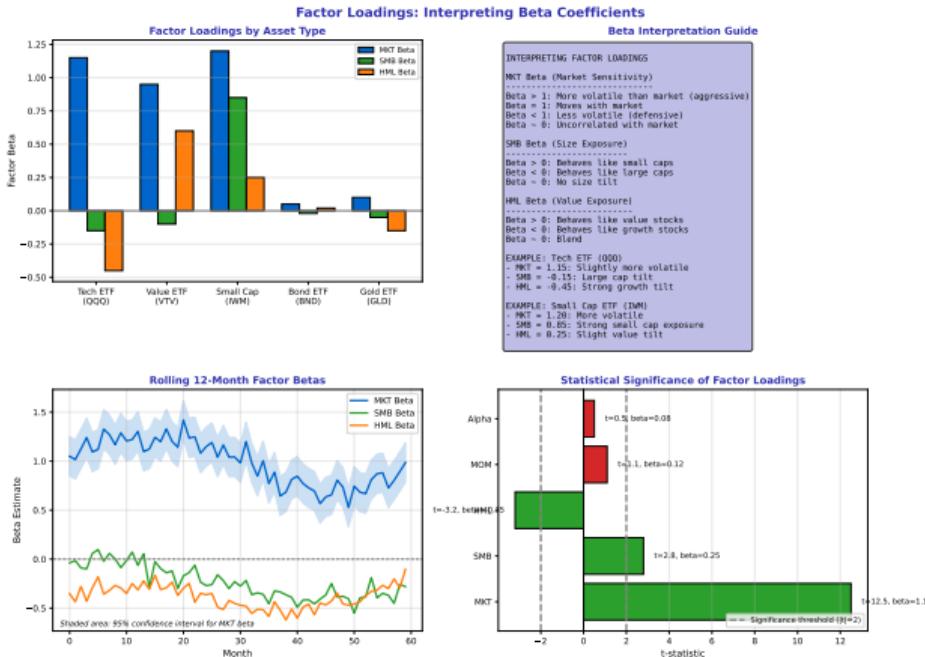


Nobel Prize (2013): Fama showed these factors explain returns better than CAPM alone

# Factor Loadings

## How Much Exposure Does a Stock Have?

- Each stock has different sensitivities to each factor
- Loading = regression coefficient on that factor

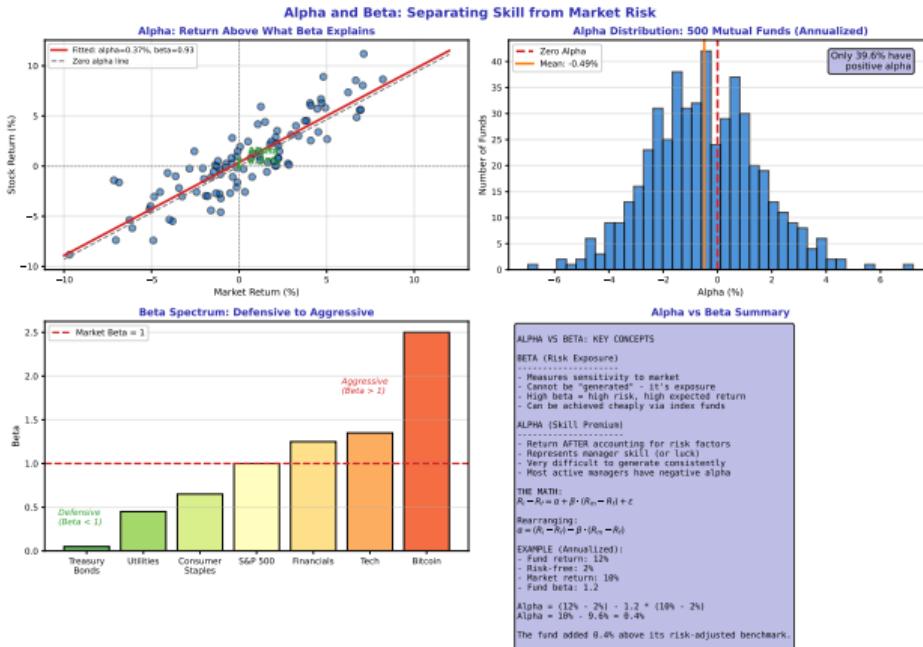


Example: TSLA has high market beta but negative HML (growth stock, not value)

# Alpha and Beta Decomposition

## Separating Skill from Risk Exposure

- $\alpha$ : Return not explained by factors (manager skill or mispricing)
- Multi-factor alpha is “purer” than CAPM alpha

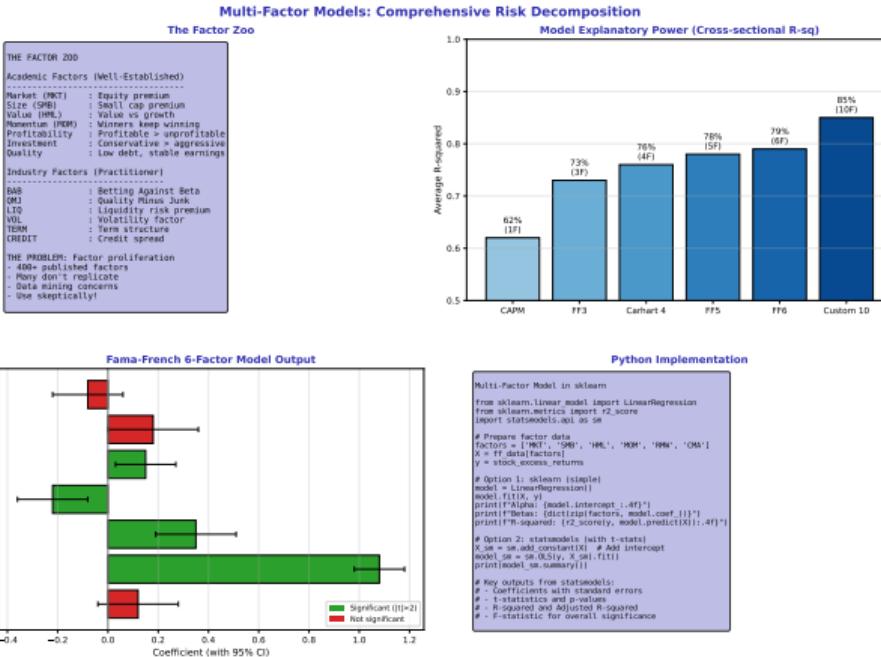


Industry standard: Report alpha after controlling for Fama-French factors

# Multi-Factor Regression

## Implementation in Python

- Same sklearn API: `LinearRegression().fit(X, y)` where X has multiple columns
- Each coefficient is a factor loading

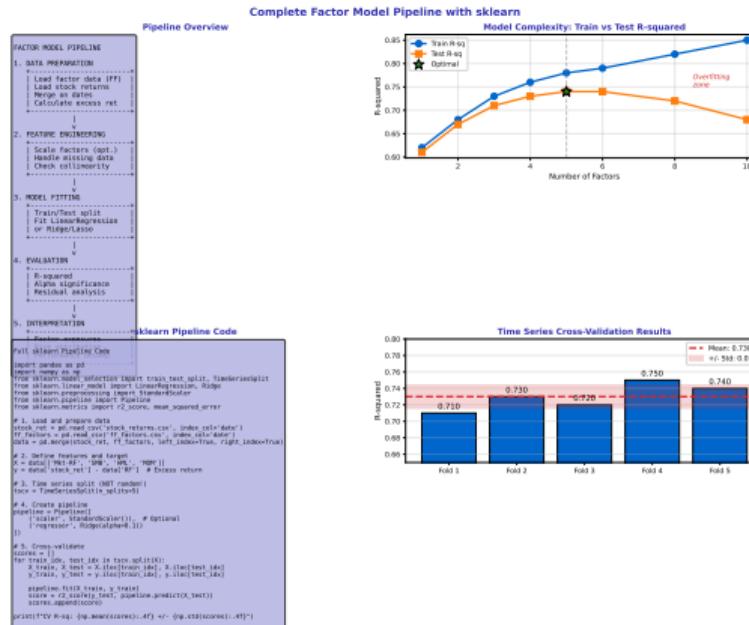


X matrix columns: [Mkt-RF, SMB, HML] – intercept is alpha

# sklearn Pipeline

## Combining Preprocessing and Modeling

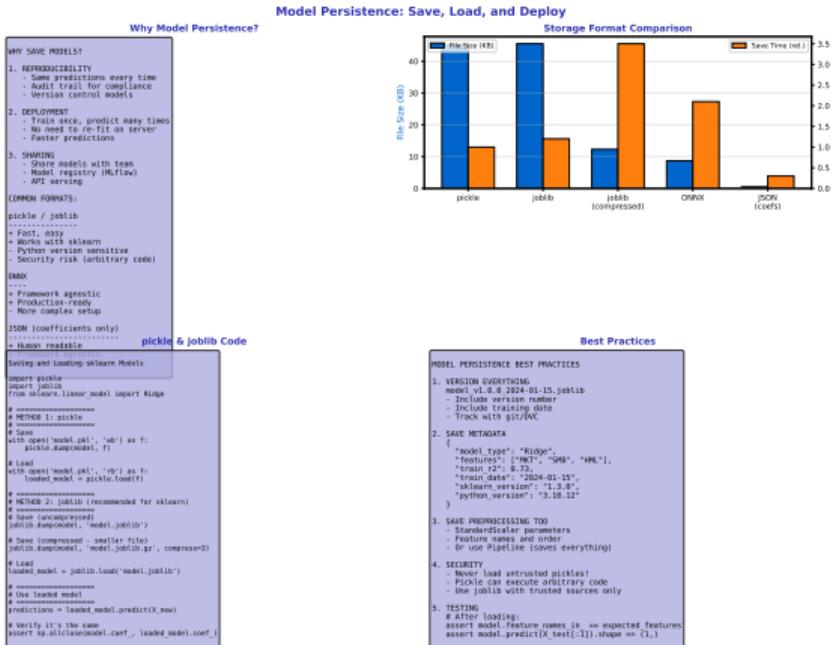
- `Pipeline([('scaler', StandardScaler()), ('reg', Ridge())])`
- Prevents data leakage: scaler fits only on training data



Best practice: Always use pipelines for reproducible workflows

## Saving and Loading Trained Models

- `joblib.dump(model, 'model.pkl')` – save to disk
- `model = joblib.load('model.pkl')` – reload for production

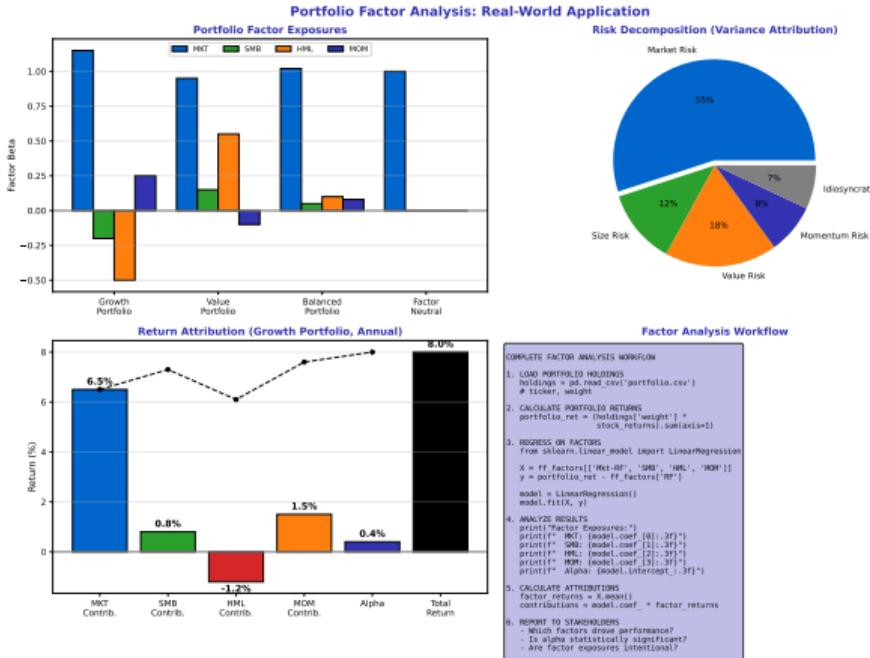


Deployment: Train once, deploy saved model to production

# Portfolio Factor Attribution

## Where Did Your Returns Come From?

- Decompose portfolio returns into factor contributions
- Shows whether performance came from market timing, factor bets, or alpha



**Risk management: Understand your factor exposures before they surprise you**

## Hands-On Exercise (25 min)

### Task: Build a Fama-French Factor Model

- ① Download Fama-French 3-factor data from Kenneth French's website
- ② Merge with your stock returns (AAPL or similar)
- ③ Fit multi-factor regression: stock returns vs [Mkt-RF, SMB, HML]
- ④ Interpret: What are the factor loadings? Is there alpha?
- ⑤ Compare  $R^2$  to single-factor CAPM model

**Deliverable:** Table of factor loadings + comparison of  $R^2$  values.

**Extension:** Add momentum (UMD) as a fourth factor – does  $R^2$  improve?

## Lesson Summary

**Problem Solved:** We can now decompose stock returns into multiple systematic factors, giving better risk attribution than CAPM alone.

### Key Takeaways:

- Fama-French: Market + SMB (size) + HML (value)
- Factor loadings = regression coefficients on each factor
- Alpha after factors = true outperformance
- sklearn pipelines ensure reproducible workflows

**Next Lesson:** Classification (L25) – predicting categories instead of numbers

**Memory:** SMB = Small Minus Big (size), HML = High Minus Low (value)

## Lesson 25: Logistic Regression

Data Science with Python – BSc Course

45 Minutes

## Learning Objectives

**The Problem:** Linear regression predicts continuous values, but what if we need to predict categories? How do we classify stocks as "buy" or "sell" based on features?

**After this lesson, you will be able to:**

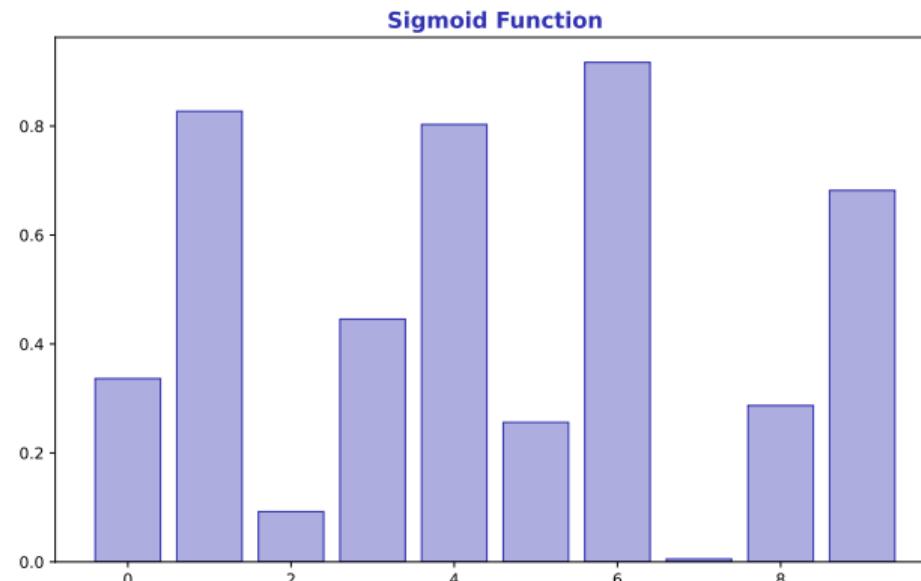
- Understand the sigmoid function and probability output
- Build binary classifiers with sklearn
- Interpret coefficients as odds ratios
- Predict market direction (up/down)

**Finance Application:** Predicting whether tomorrow's return is positive or negative

# Sigmoid Function

## From Linear to Probability

- $\sigma(z) = \frac{1}{1+e^{-z}}$  – squashes any value to (0, 1)
- Output is interpreted as  $P(y = 1|x)$  – probability of positive class

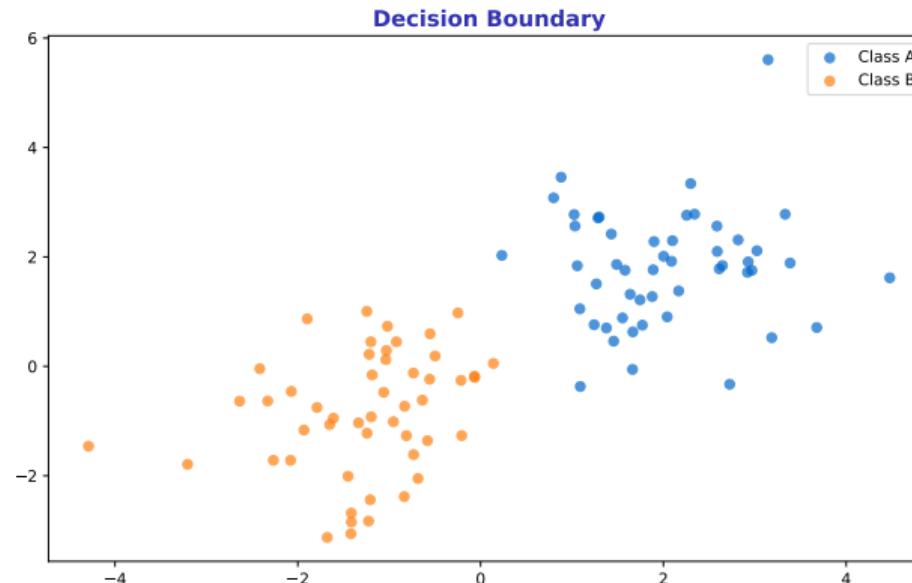


Key insight: Logistic regression = linear regression passed through sigmoid

# Decision Boundary

## Where Do We Draw the Line?

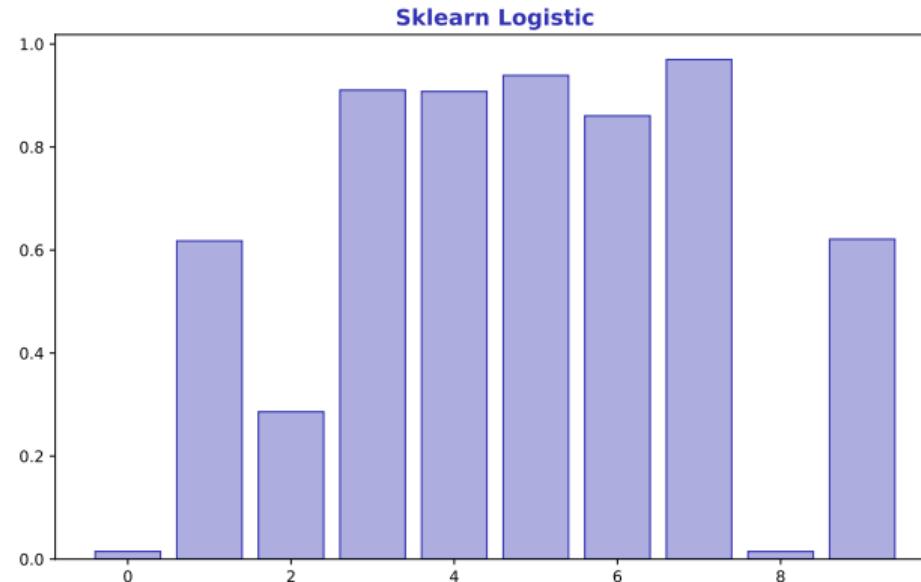
- Default threshold: predict class 1 if  $P(y = 1) > 0.5$
- Decision boundary is where  $\beta_0 + \beta_1 x = 0$



The boundary can be adjusted based on costs of false positives vs false negatives

## Same API as Linear Regression

- `from sklearn.linear_model import LogisticRegression`
- `model.fit(X, y) then model.predict(X_new)`

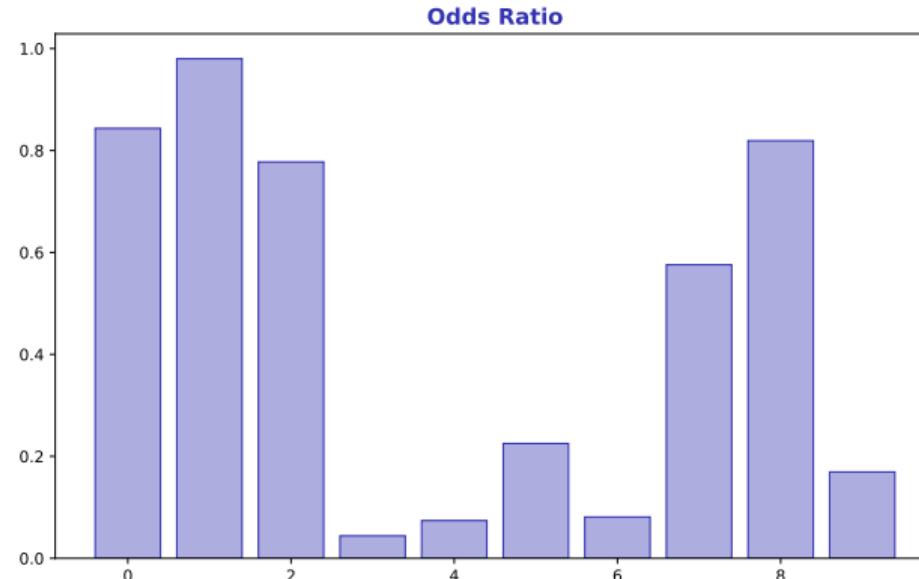


Use `model.predict_proba(X)` to get probabilities instead of 0/1

# Odds Ratio Interpretation

## What Do the Coefficients Mean?

- $e^{\beta_j}$  = multiplicative change in odds per unit increase in  $x_j$
- $e^{\beta} = 2$  means odds double when feature increases by 1

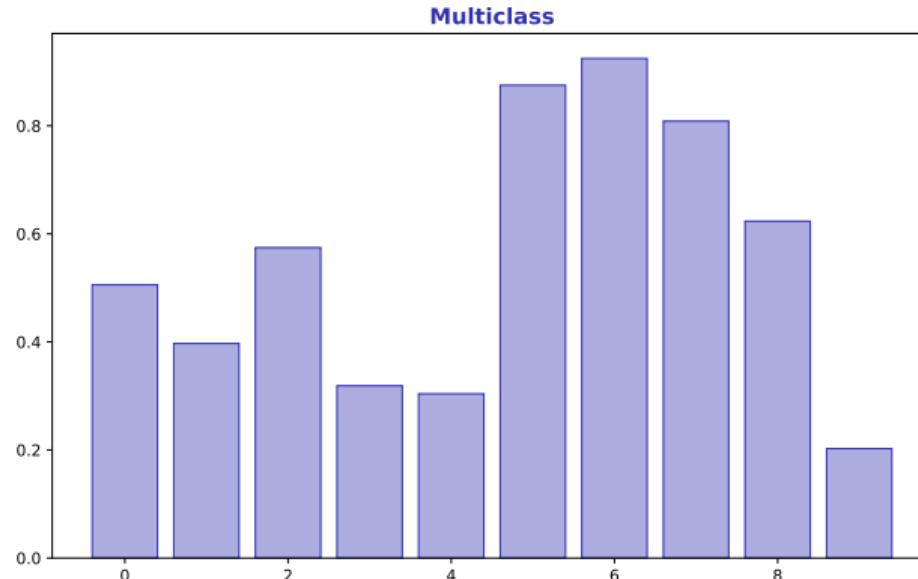


Finance: “Each 1% increase in momentum doubles the odds of positive return”

# Multiclass Classification

## More Than Two Classes

- One-vs-Rest (OvR): train K binary classifiers, pick highest probability
- Softmax/Multinomial: direct extension of sigmoid to K classes

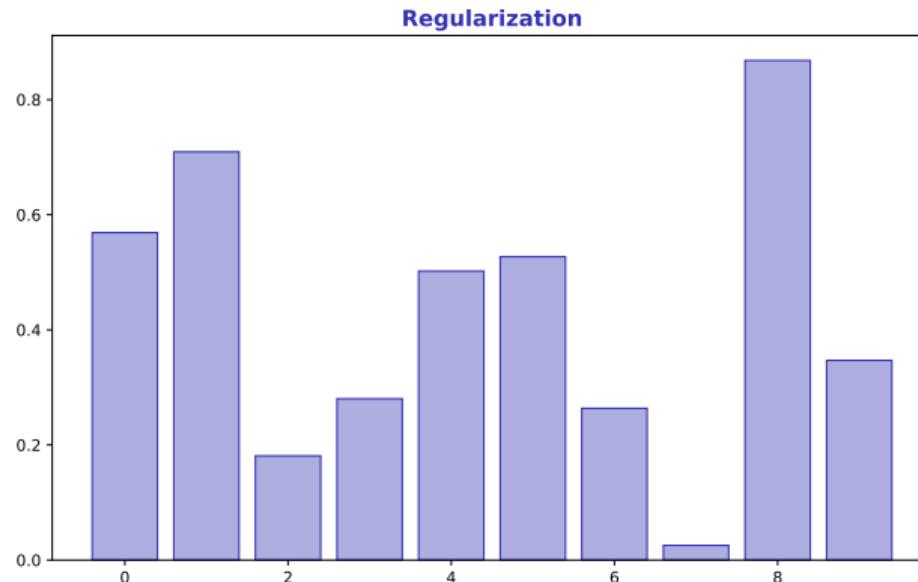


**sklearn default: `multi_class='auto'` chooses automatically**

# Regularization in Logistic

## Preventing Overfitting

- Same L1 (Lasso) and L2 (Ridge) penalties apply
- sklearn default: L2 with  $C=1.0$  (inverse of  $\lambda$ )

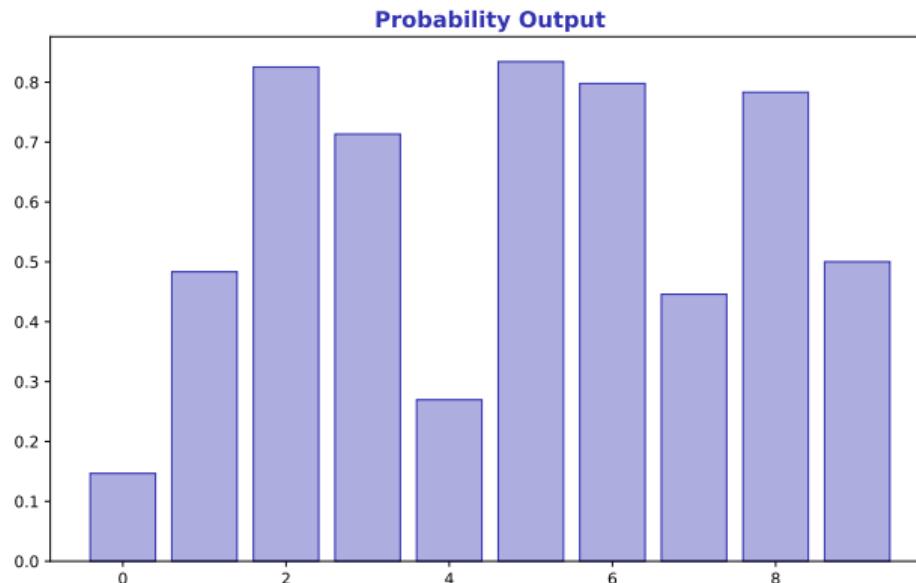


Lower  $C$  = stronger regularization (opposite of  $\lambda$  convention)

# Probability Calibration

## Are the Probabilities Accurate?

- Model says 70% – does event happen 70% of the time?
- Logistic regression is generally well-calibrated

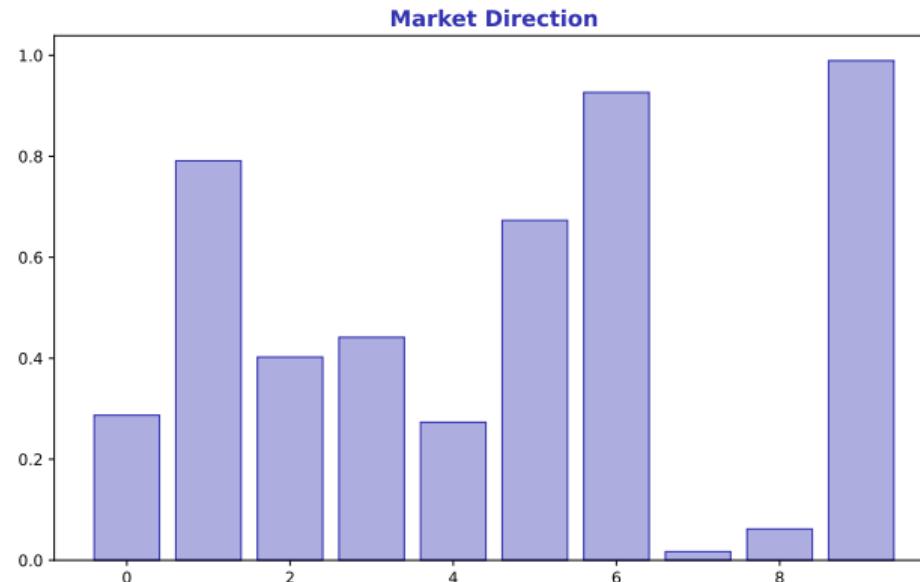


Calibration matters for risk management – you need accurate probabilities

# Market Direction Prediction

## Finance Application: Up or Down?

- Features: lagged returns, volume, volatility
- Target: 1 if next-day return  $> 0$ , else 0



Reality check: Even 52% accuracy can be profitable with proper sizing

## Hands-On Exercise (25 min)

### Task: Predict Stock Direction

- ① Create binary target: 1 if next-day return  $> 0$ , else 0
- ② Features: 5-day momentum, 20-day volatility, volume ratio
- ③ Fit logistic regression and examine coefficients
- ④ Calculate accuracy on held-out test set
- ⑤ Interpret: Which features increase odds of positive return?

**Deliverable:** Coefficient table with odds ratios + test accuracy.

**Extension:** Try different thresholds (0.4, 0.6) – how does accuracy change?

## Lesson Summary

**Problem Solved:** We can now predict binary outcomes (up/down, buy/sell) and get probability estimates.

**Key Takeaways:**

- Sigmoid transforms linear output to probability
- Coefficients:  $e^{\beta}$  = odds ratio
- Same sklearn API: `fit()`, `predict()`, `predict_proba()`
- Regularization via C parameter (lower = stronger)

**Next Lesson:** Decision Trees (L26) – non-linear classification

**Memory:** Logistic = Linear + Sigmoid. Output is probability, not continuous value.

## Lesson 26: Decision Trees

Data Science with Python – BSc Course

45 Minutes

**The Problem:** Logistic regression assumes linear decision boundaries. What if the relationship between features and class is more complex?

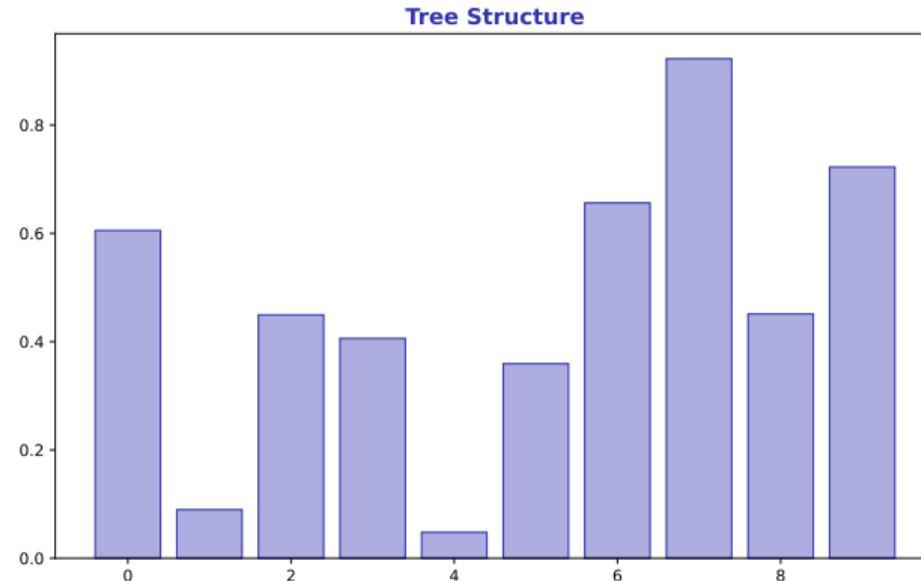
**After this lesson, you will be able to:**

- Build decision tree classifiers
- Understand splitting criteria (Gini, Entropy)
- Apply Random Forest for better generalization
- Interpret feature importance from tree models

**Finance Application:** Rule-based credit scoring and trading signals

## If-Then Rules as a Tree

- Root node: first split on most informative feature
- Leaf nodes: final class predictions

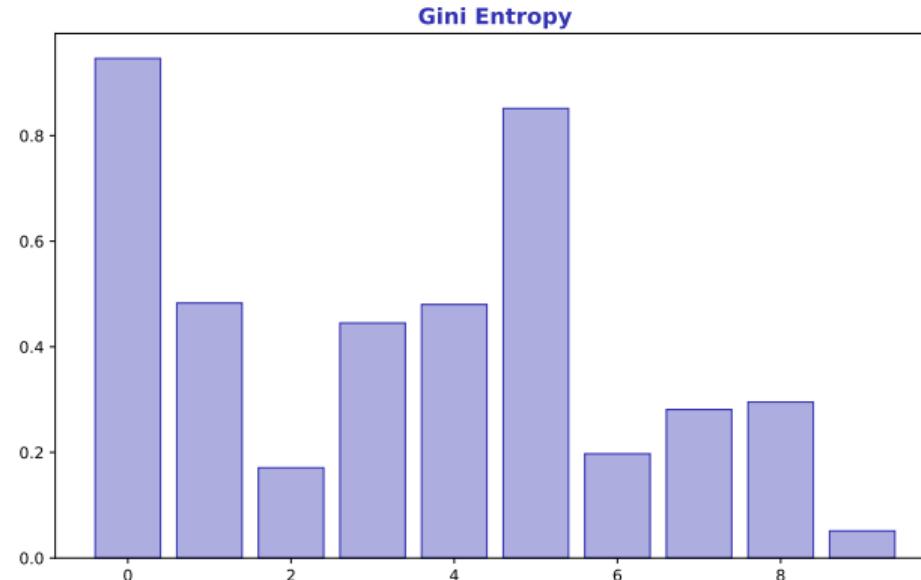


Trees are interpretable: you can explain exactly why a prediction was made

# Gini and Entropy

## How to Choose the Best Split?

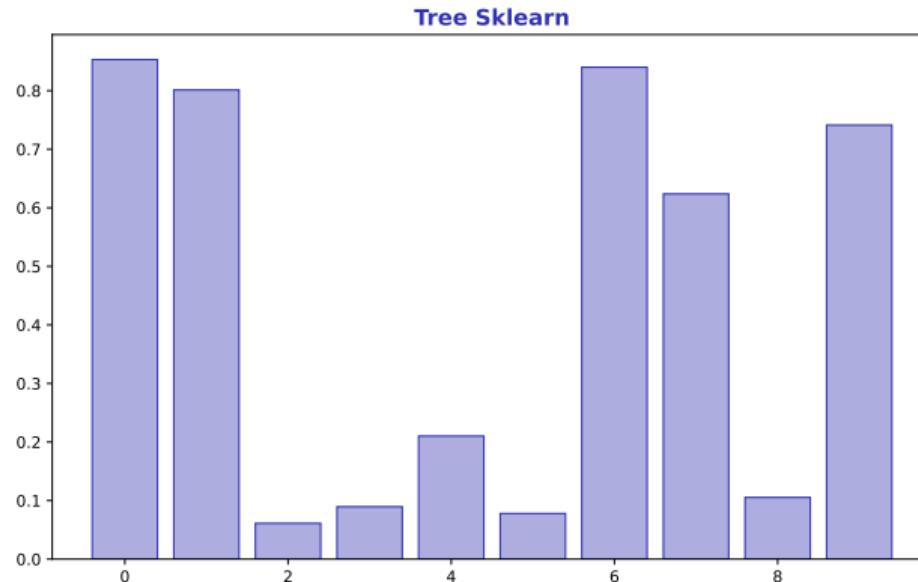
- Gini:  $1 - \sum p_i^2$  – measures impurity (lower = purer)
- Entropy:  $-\sum p_i \log p_i$  – information gain criterion



In practice: Gini and Entropy give similar results. sklearn default is Gini.

## Building Trees in Python

- `from sklearn.tree import DecisionTreeClassifier`
- Key params: `max_depth, min_samples_leaf`

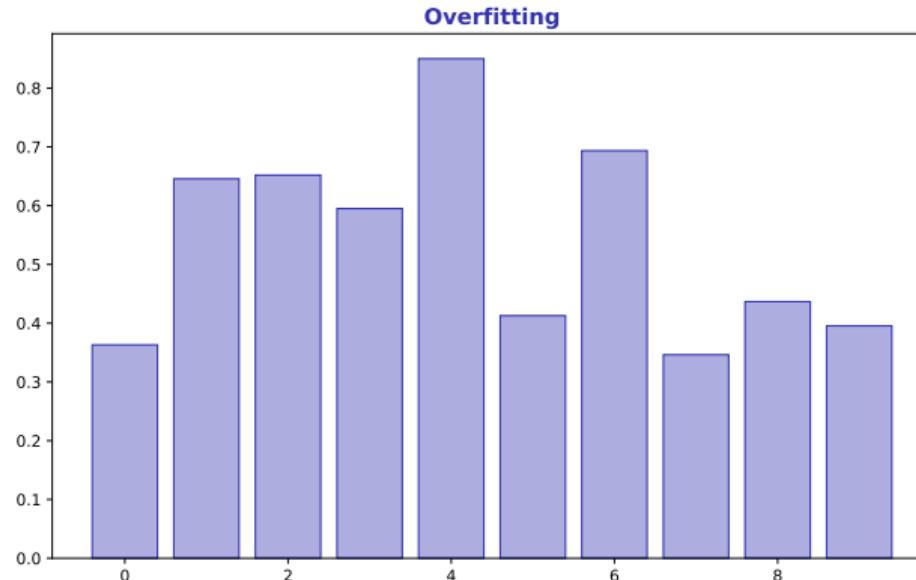


Always set `max_depth` to prevent trees from memorizing training data

# Overfitting in Trees

## The Danger of Deep Trees

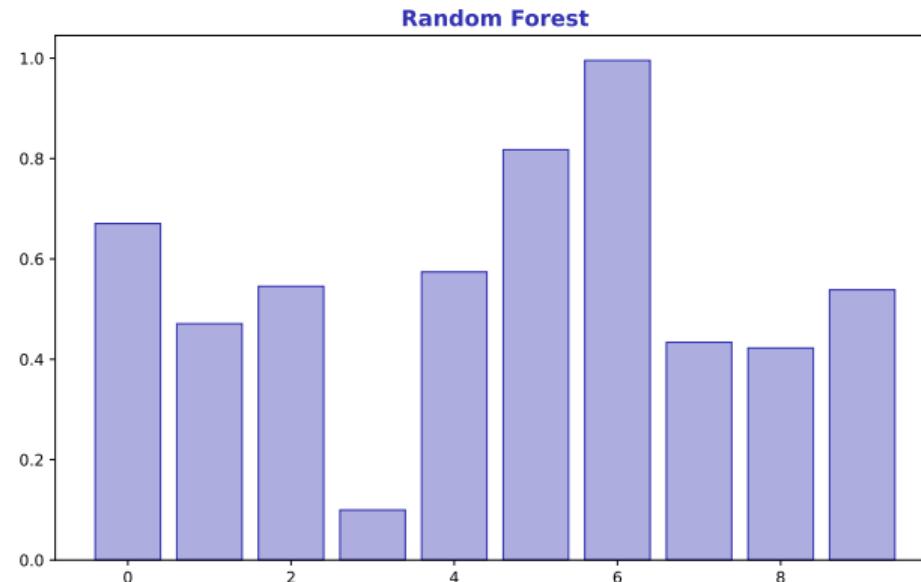
- Unlimited depth: tree can perfectly fit training data (100% accuracy)
- Test accuracy often much worse – memorization, not learning



Rule: Start with `max_depth=3`, increase only if underfitting

## Ensemble of Trees

- Train many trees on random subsets of data and features
- Final prediction: majority vote (classification) or average (regression)

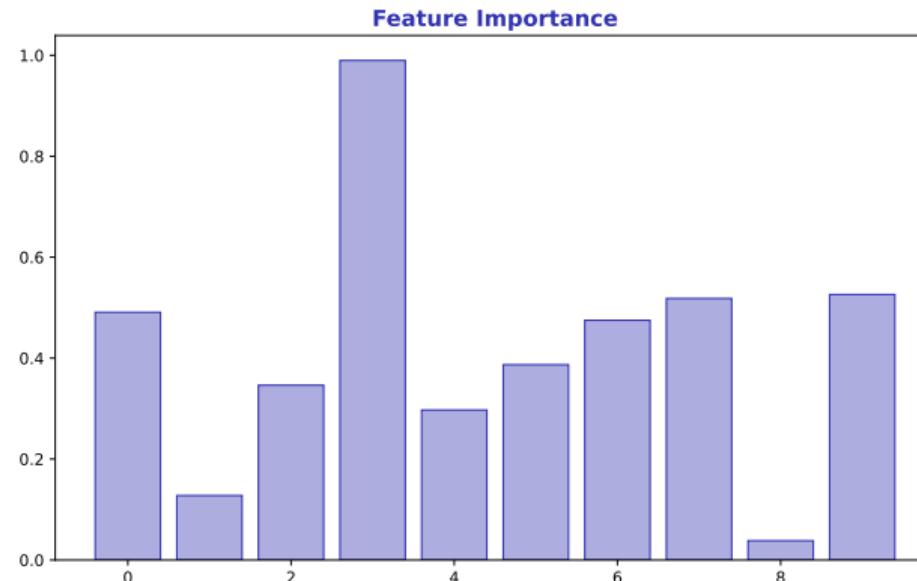


Random Forest = bagging + feature randomization. Much more robust than single tree.

# Feature Importance

## Which Features Matter Most?

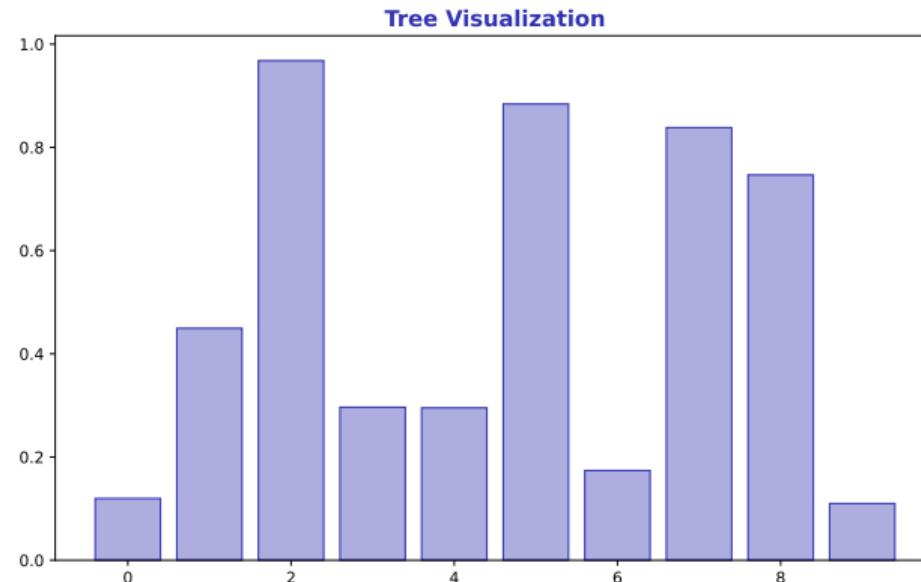
- Importance = total reduction in impurity from splits on that feature
- Normalized to sum to 1.0 across all features



Access via `model.feature_importances_` – useful for feature selection

## Making Trees Interpretable

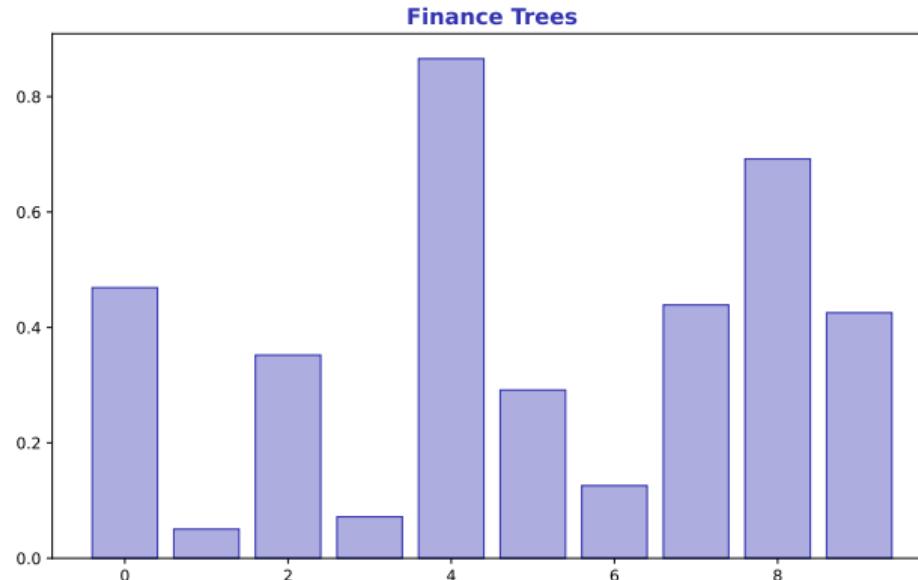
- `sklearn.tree.plot_tree()` for inline visualization
- Export to Graphviz for publication-quality diagrams



Visualization helps explain model decisions to non-technical stakeholders

## Trading Rules from Trees

- Trees naturally create rule-based strategies
- Example: "If RSI < 30 AND volume > 2×avg, then BUY"



**Caution:** Trees overfit easily on financial data – use cross-validation

## Hands-On Exercise (25 min)

### Task: Build a Trading Signal Classifier

- ① Features: RSI, MACD, Bollinger Band position, volume ratio
- ② Target: 1 if next-5-day return  $> 2\%$ , else 0
- ③ Train DecisionTree with `max_depth=4` and RandomForest
- ④ Compare test accuracy – which generalizes better?
- ⑤ Plot feature importance for Random Forest

**Deliverable:** Tree visualization + feature importance bar chart.

**Extension:** Try different `max_depth` values – plot train vs test accuracy

## Lesson Summary

**Problem Solved:** Decision trees capture non-linear relationships and produce interpretable rules.

**Key Takeaways:**

- Trees split on features that maximize information gain
- Control complexity via max\_depth, min\_samples\_leaf
- Random Forest: many trees > one tree (reduces overfitting)
- Feature importance reveals which variables drive predictions

**Next Lesson:** Classification Metrics (L27) – beyond accuracy

**Memory:** Single tree overfits. Random Forest averages many trees for stability.

## Lesson 27: Classification Metrics

Data Science with Python – BSc Course

45 Minutes

**The Problem:** Our classifier has 95% accuracy – is that good? What if 95% of samples are one class? How do we properly evaluate classification models?

**After this lesson, you will be able to:**

- Build and interpret confusion matrices
- Calculate precision, recall, and F1 score
- Plot and interpret ROC curves and AUC
- Choose metrics appropriate for your problem

**Finance Application: Evaluating fraud detection and default prediction models**

# Confusion Matrix

## The Foundation of Classification Metrics

- 2x2 table: TP, TN, FP, FN (True/False Positive/Negative)
- All other metrics derive from these four numbers

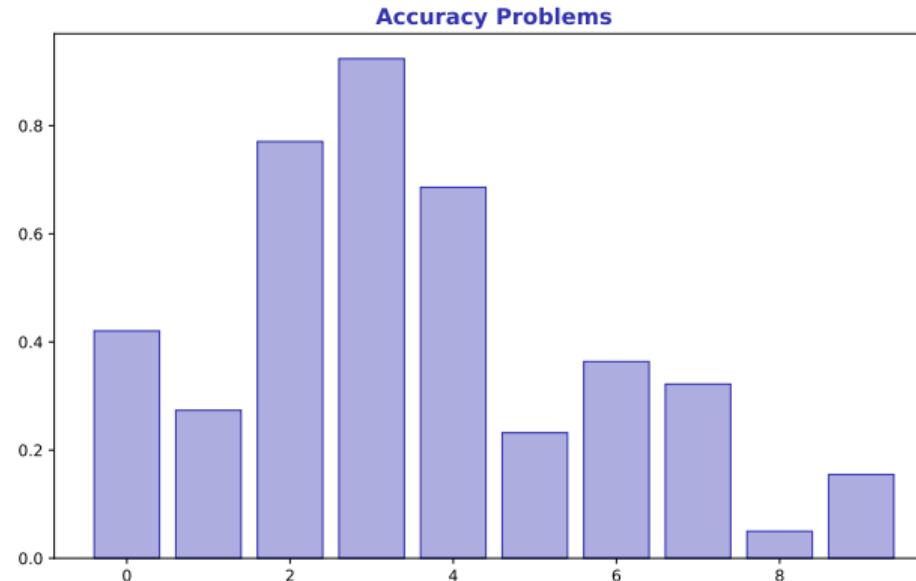
		Confusion Matrix	
		Pred 0	Pred 1
True 0	True 0	45	5
	True 1	8	42

TP = correct positive, FP = false alarm (Type I), FN = missed positive (Type II)

# The Accuracy Trap

## When Accuracy Misleads

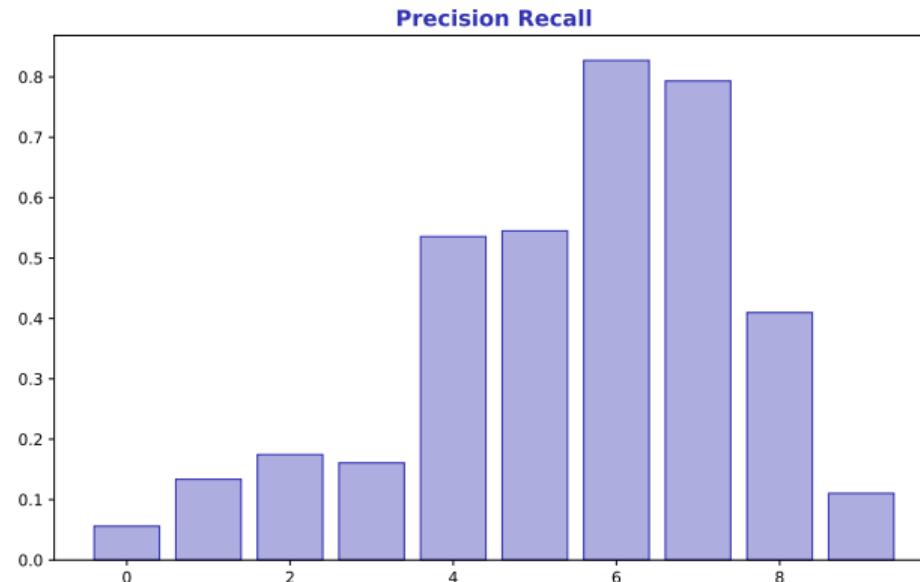
- 1% fraud rate: predicting “no fraud” gives 99% accuracy
- But you catch zero frauds – useless model



**Rule:** Never use accuracy alone when classes are imbalanced

## Two Perspectives on Model Quality

- Precision =  $TP / (TP + FP)$  – “Of predicted positives, how many correct?”
- Recall =  $TP / (TP + FN)$  – “Of actual positives, how many found?”

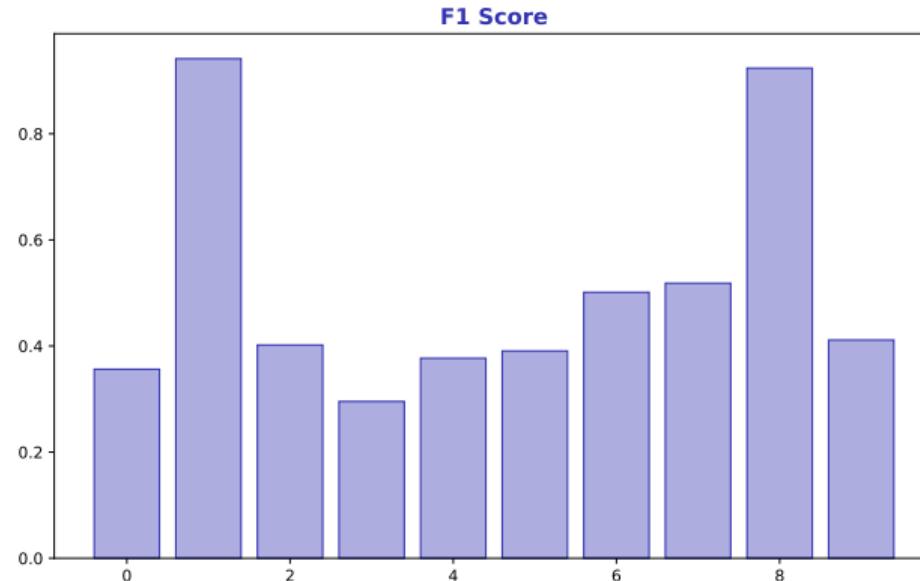


**Trade-off:** High threshold = high precision, low recall. Low threshold = opposite.

# F1 Score

## Balancing Precision and Recall

- $F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$  – harmonic mean
- Single number when you need both precision and recall

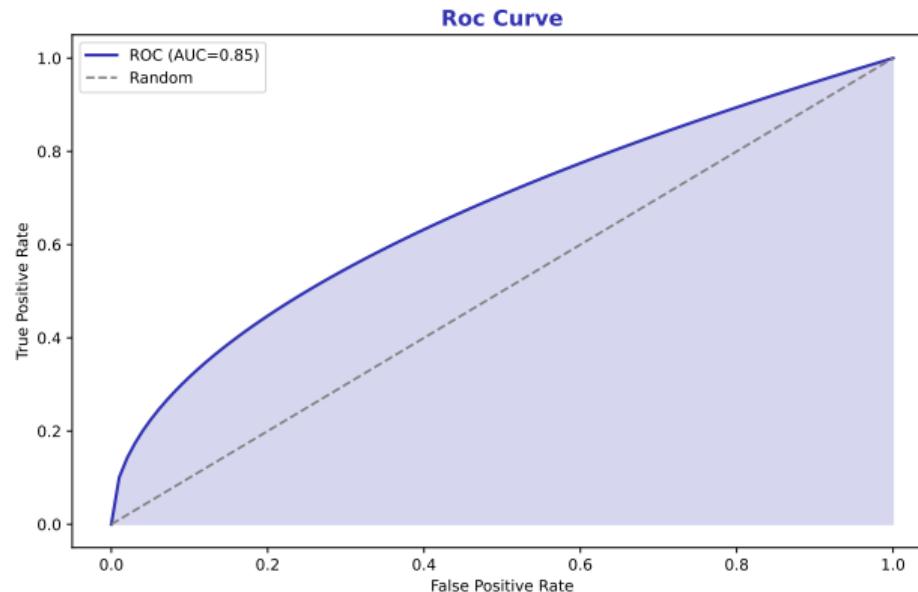


F1 = 0 if either precision or recall is 0. Max F1 = 1 (perfect).

# ROC Curve

## Performance Across All Thresholds

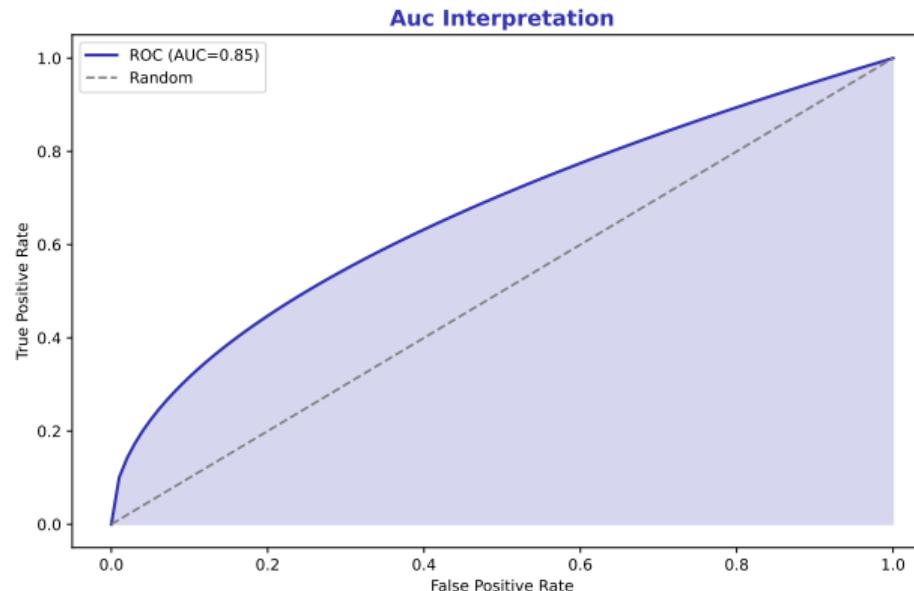
- X-axis: False Positive Rate (FPR). Y-axis: True Positive Rate (Recall)
- Each point is a different decision threshold



Perfect classifier: curve goes to top-left corner (TPR=1, FPR=0)

## Area Under the ROC Curve

- AUC = 0.5: random guessing. AUC = 1.0: perfect separation
- Interpretation: probability that model ranks positive higher than negative

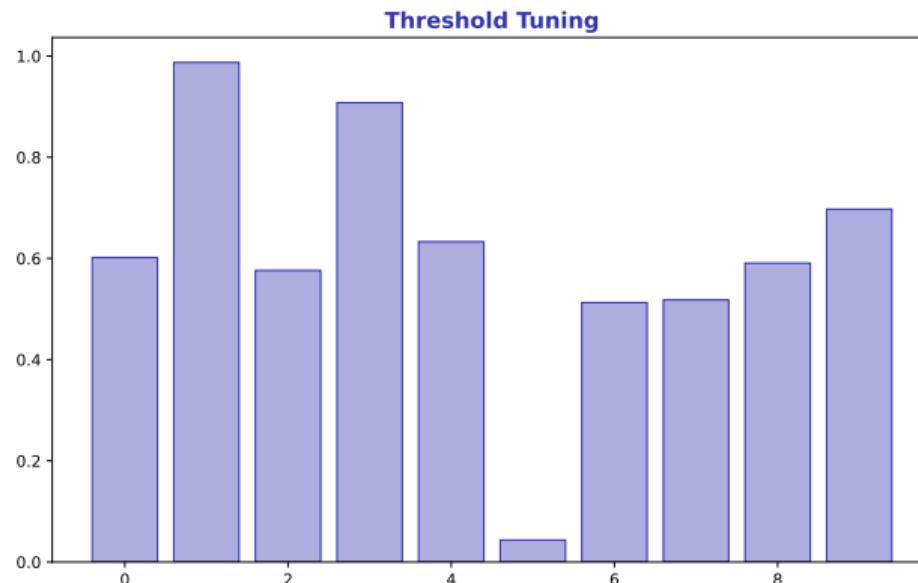


Industry benchmarks: AUC > 0.7 acceptable, > 0.8 good, > 0.9 excellent

# Threshold Tuning

## Choosing the Right Operating Point

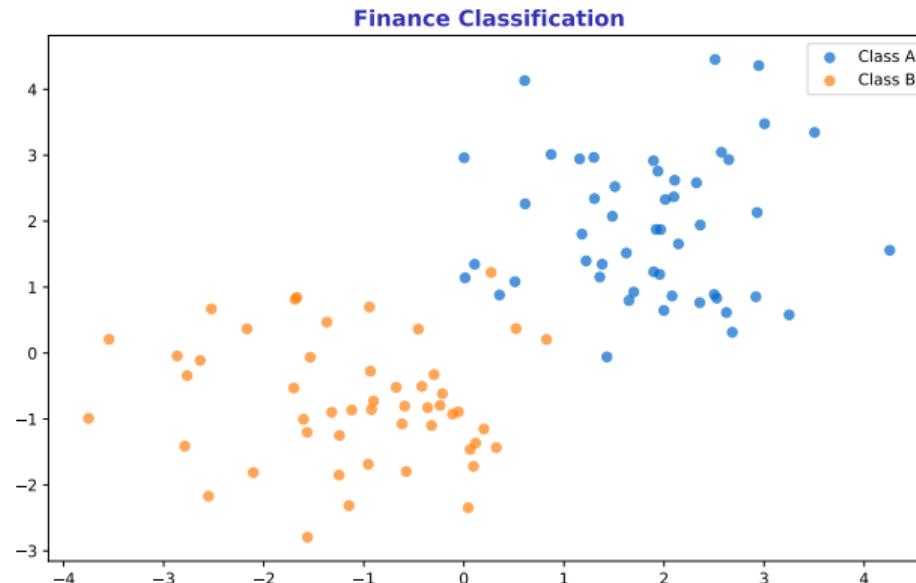
- Default threshold (0.5) is rarely optimal
- Tune based on business costs: cost of FP vs cost of FN



Example: Fraud costs \$1000, investigation costs \$10 – lower threshold is better

## Metrics for Financial Problems

- Default prediction: high recall (catch all defaults), accept lower precision
- Trading signals: high precision (avoid false signals), accept lower recall



Match metric to cost structure: what's worse, missing a default or false alarm?

## Hands-On Exercise (25 min)

### Task: Evaluate a Default Prediction Model

- ① Fit logistic regression on credit data (target: default yes/no)
- ② Generate confusion matrix with `confusion_matrix()`
- ③ Calculate precision, recall, F1 using `classification_report()`
- ④ Plot ROC curve and calculate AUC
- ⑤ Try thresholds 0.3 and 0.7 – how do metrics change?

**Deliverable:** Confusion matrix heatmap + ROC curve with AUC annotation.

**Extension:** Calculate expected cost at each threshold using custom cost matrix

## Lesson Summary

**Problem Solved:** We now have a toolkit of metrics beyond accuracy for proper model evaluation.

**Key Takeaways:**

- Confusion matrix: TP, TN, FP, FN – foundation of all metrics
- Precision = quality of positives, Recall = coverage of positives
- ROC/AUC: threshold-independent performance measure
- Choose metrics based on business costs

**Next Lesson:** Class Imbalance (L28) – handling rare events

**Memory:** Precision = Positive predictions that are correct. Recall = Positives that we Recovered.

## Lesson 28: Class Imbalance

Data Science with Python – BSc Course

45 Minutes

**The Problem:** Fraud occurs in 0.1% of transactions. Default in 2% of loans. How do we train models when positive cases are extremely rare?

**After this lesson, you will be able to:**

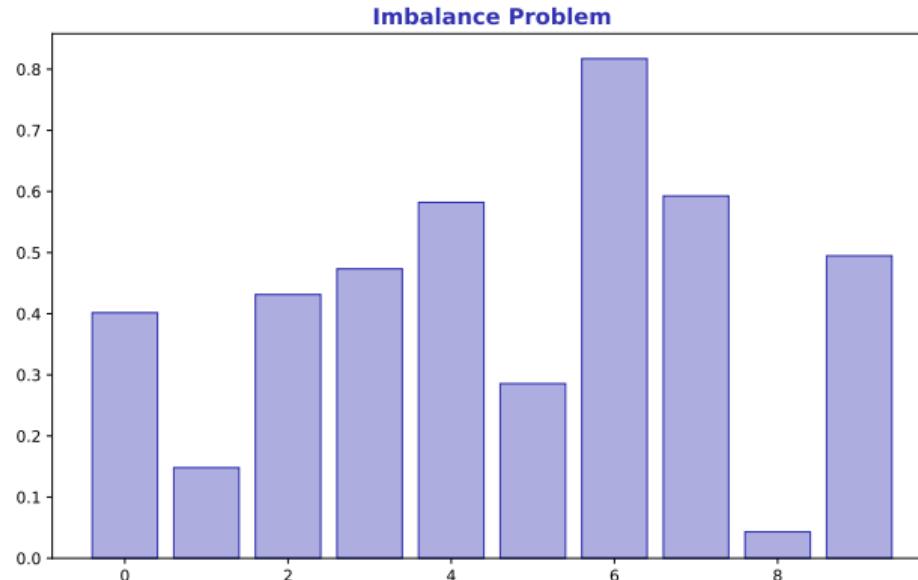
- Identify and diagnose imbalanced datasets
- Apply SMOTE and other oversampling techniques
- Use class weights to rebalance training
- Evaluate models fairly on imbalanced data

**Finance Application:** Fraud detection, default prediction, rare event modeling

# The Imbalance Problem

## Why Standard Models Fail

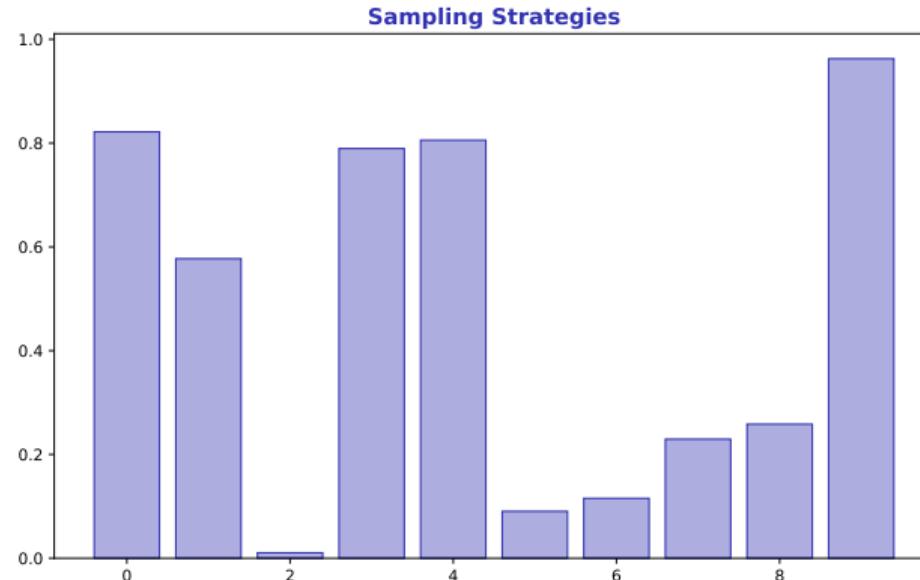
- Model learns to predict majority class (easy 99% accuracy)
- Minority class is ignored because errors are “cheap”



**Check:** If your model predicts same class 95%+ of the time, you have a problem

## Rebalancing the Training Data

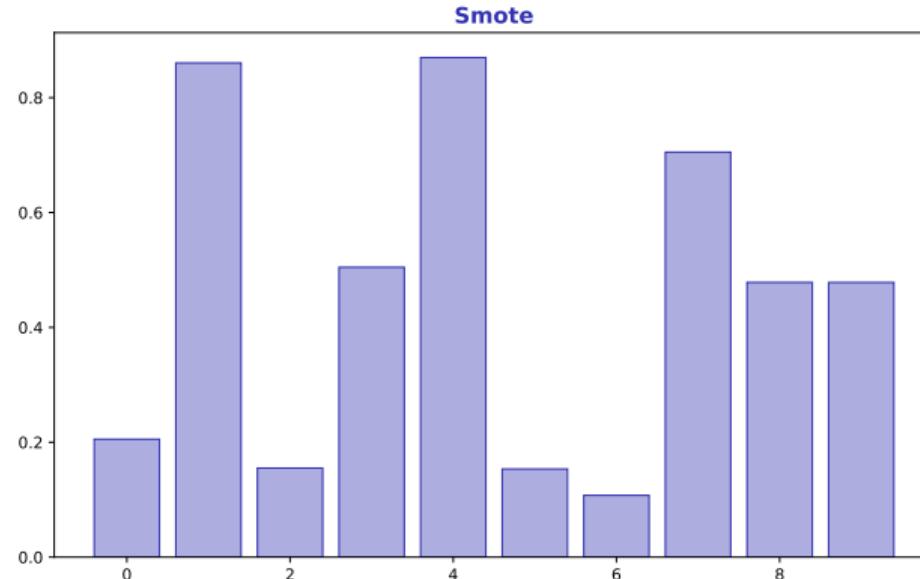
- Undersampling: remove majority class samples (loses information)
- Oversampling: duplicate or synthesize minority samples



**Rule:** Only resample training data, never test data – test must reflect reality

## Synthetic Minority Over-sampling Technique

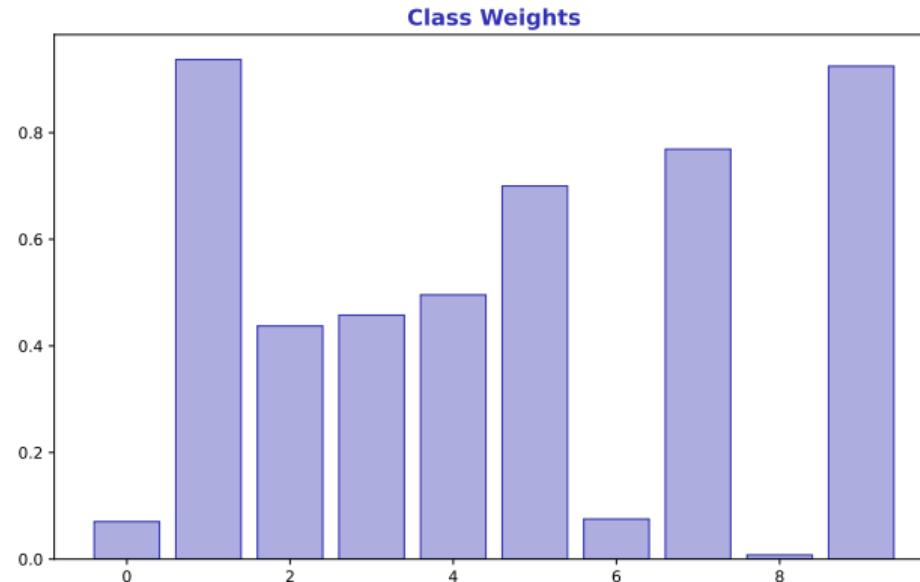
- Creates synthetic samples by interpolating between minority neighbors
- from imblearn.over\_sampling import SMOTE



SMOTE creates new points along lines between existing minority samples

## Rebalancing Without Resampling

- Give higher weight to minority class errors in loss function
- sklearn: `class_weight='balanced'` or custom weights

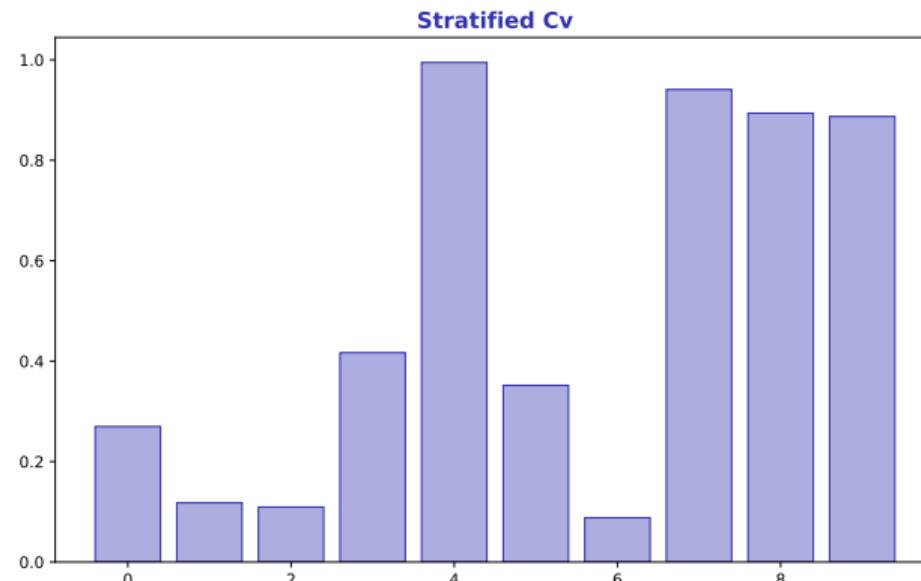


Balanced weights: inversely proportional to class frequency

# Stratified Cross-Validation

## Preserving Class Proportions

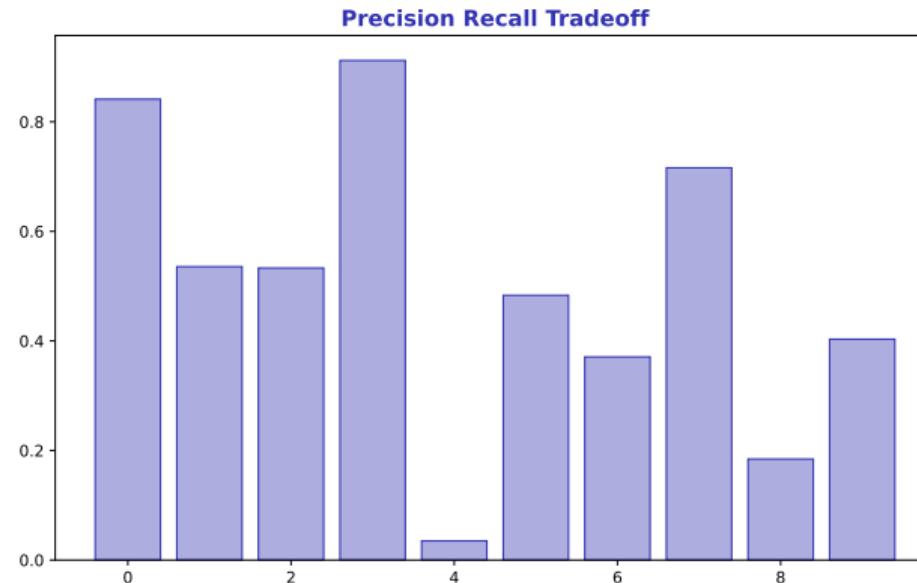
- Regular CV may put all rare events in one fold
- Stratified CV ensures each fold has same class ratio



Always use `StratifiedKFold` for imbalanced classification

## Finding the Right Balance

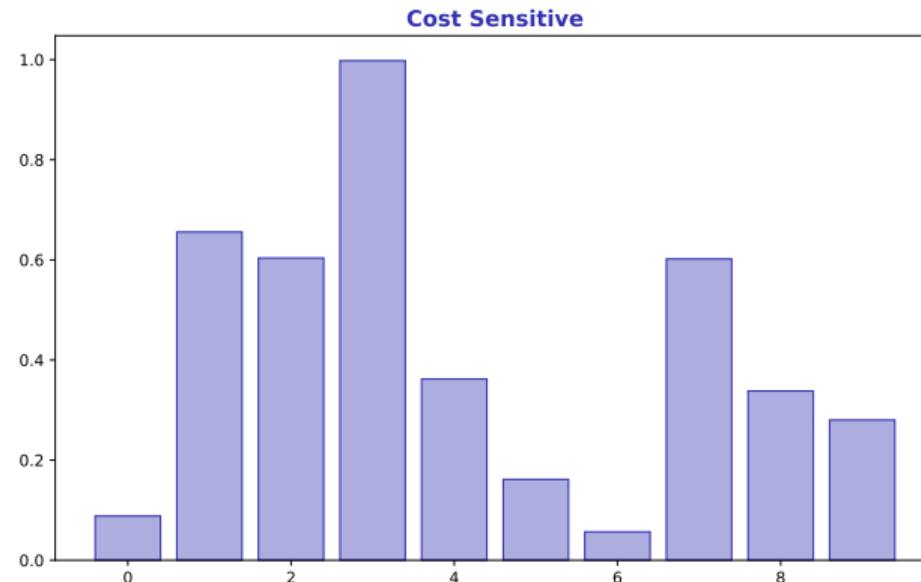
- Precision-Recall curve better than ROC for imbalanced data
- Area under PR curve (AP) is more informative than AUC



PR curve focuses on positive class – what we care about in imbalanced problems

## Encoding Business Costs

- FN (missed fraud) costs \$1000, FP (false alarm) costs \$10
- Optimal threshold minimizes expected total cost

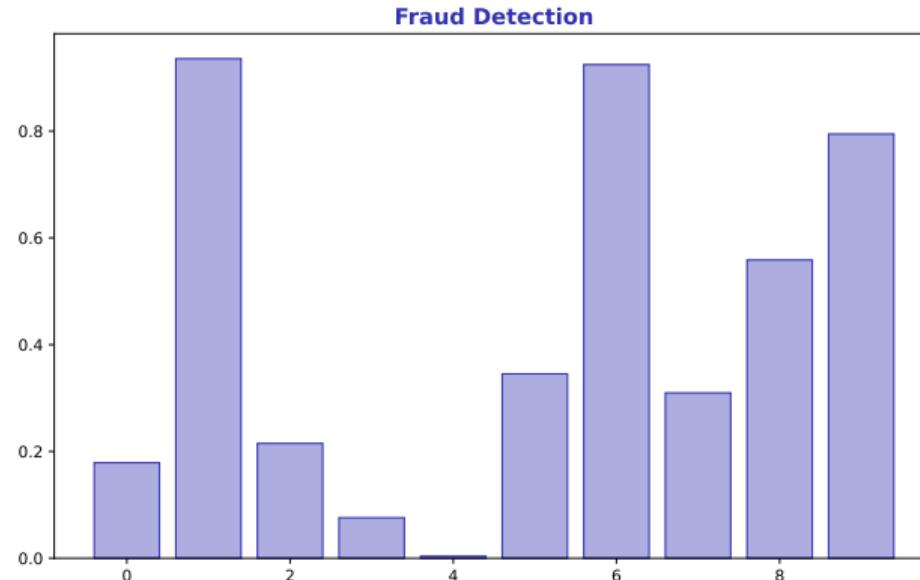


Formula:  $\text{Expected Cost} = \text{FN} \times \text{Cost}_{FN} + \text{FP} \times \text{Cost}_{FP}$

# Fraud Detection Example

## Putting It All Together

- Real-world fraud rates: 0.1-1% positive
- Pipeline: SMOTE + class weights + stratified CV + PR evaluation



Industry insight: Ensemble methods (XGBoost, LightGBM) work well for fraud

## Hands-On Exercise (25 min)

### Task: Build a Fraud Detection Model

- ① Create synthetic imbalanced data (1% fraud rate)
- ② Train baseline model – observe accuracy trap
- ③ Apply SMOTE and retrain – compare recall
- ④ Use `class_weight='balanced'` – compare to SMOTE
- ⑤ Plot Precision-Recall curve for best model

**Deliverable:** Comparison table of recall/precision across methods.

**Extension:** Implement cost-sensitive threshold selection

## Lesson Summary

**Problem Solved:** We can now handle imbalanced datasets common in finance (fraud, default, rare events).

**Key Takeaways:**

- Accuracy misleads on imbalanced data – use precision/recall
- SMOTE creates synthetic minority samples
- Class weights: alternative to resampling
- Always use stratified CV and PR curves

**Next Lesson:** KMeans Clustering (L29) – unsupervised learning begins

**Memory:** SMOTE = Synthetic samples. Class weights = penalize minority errors more.

## Lesson 29: K-Means Clustering

Data Science with Python – BSc Course

45 Minutes

## Learning Objectives

**The Problem:** We have 500 stocks with dozens of features. How do we group similar stocks together without predefined categories? This is unsupervised learning.

**After this lesson, you will be able to:**

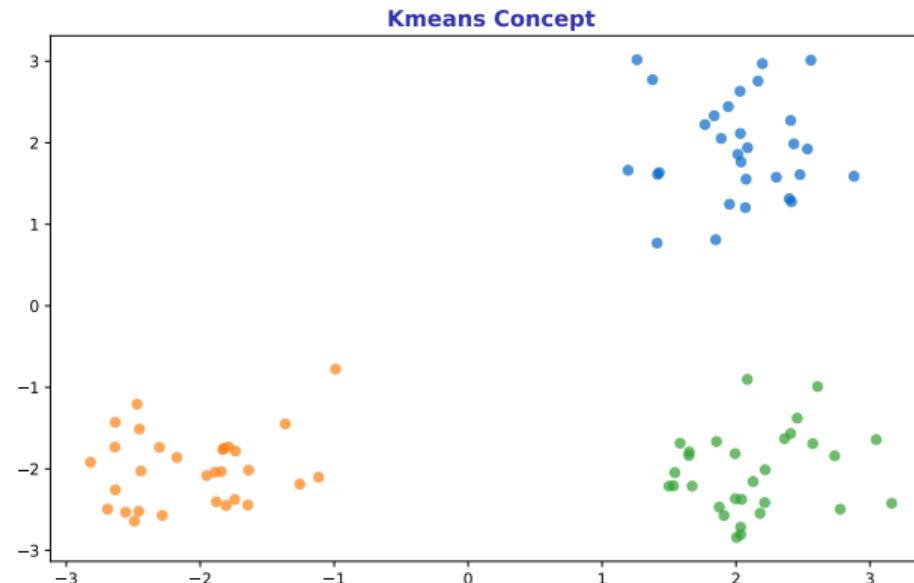
- Apply K-Means clustering algorithm
- Choose optimal K using elbow method and silhouette score
- Interpret cluster centers (centroids)
- Segment financial assets by behavior

**Finance Application:** Stock segmentation, customer clustering, regime detection

# K-Means Concept

## Partitioning Data into K Groups

- Goal: minimize within-cluster variance (inertia)
- Each point belongs to cluster with nearest centroid

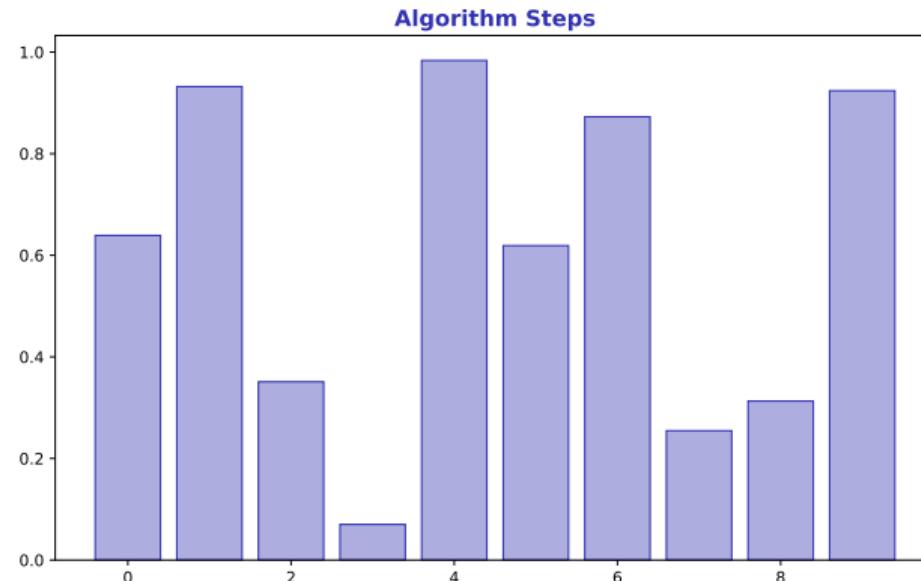


K-Means finds compact, spherical clusters – works well when clusters are well-separated

# Algorithm Steps

## Iterative Refinement

- Initialize: randomly place K centroids
- Repeat: (1) assign points to nearest centroid, (2) update centroids to cluster means

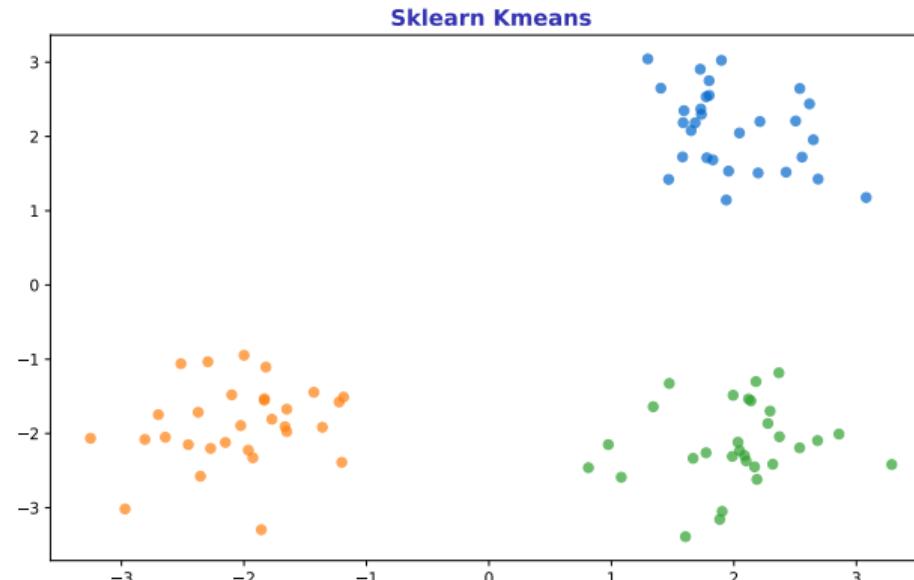


Convergence guaranteed but may find local minimum – use multiple random starts

# sklearn KMeans

## Implementation in Python

- `from sklearn.cluster import KMeans`
- `kmeans = KMeans(n_clusters=K, n_init=10).fit(X)`

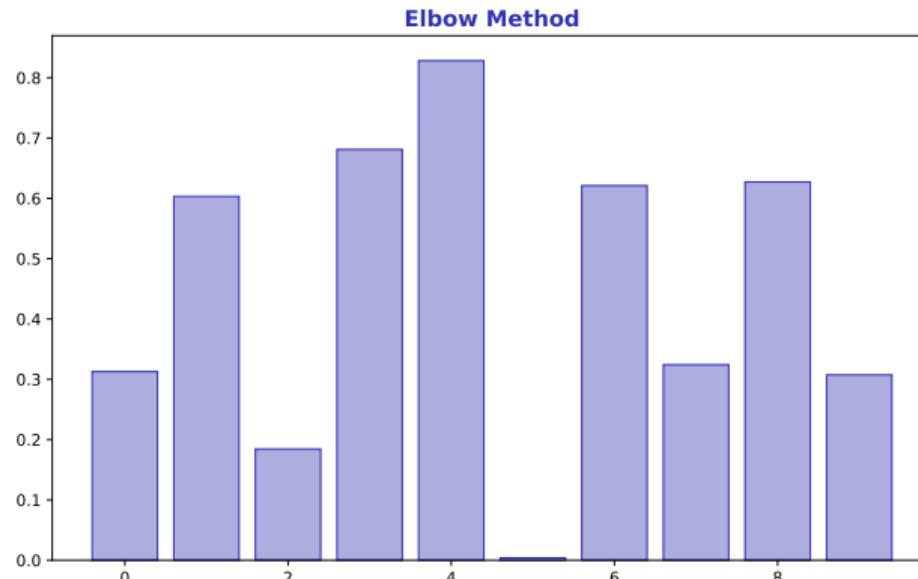


Access labels via `kmeans.labels_` and centers via `kmeans.cluster_centers_`.

# Elbow Method

## How Many Clusters?

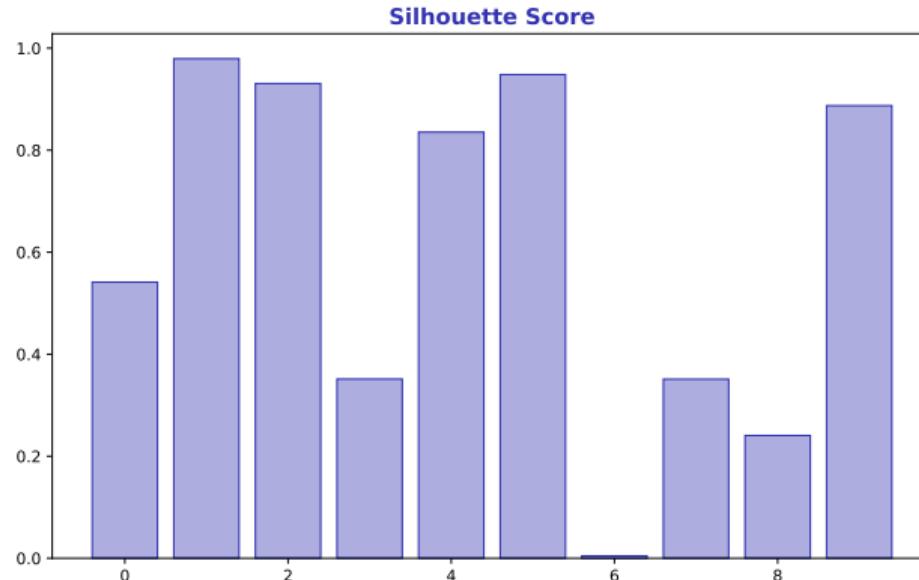
- Plot inertia vs K – look for “elbow” where improvement slows
- Inertia always decreases with K; find diminishing returns



The elbow is subjective – combine with domain knowledge and silhouette score

## Measuring Cluster Quality

- Range: -1 to 1. Higher = better-defined clusters
- Compares within-cluster distance to nearest-cluster distance

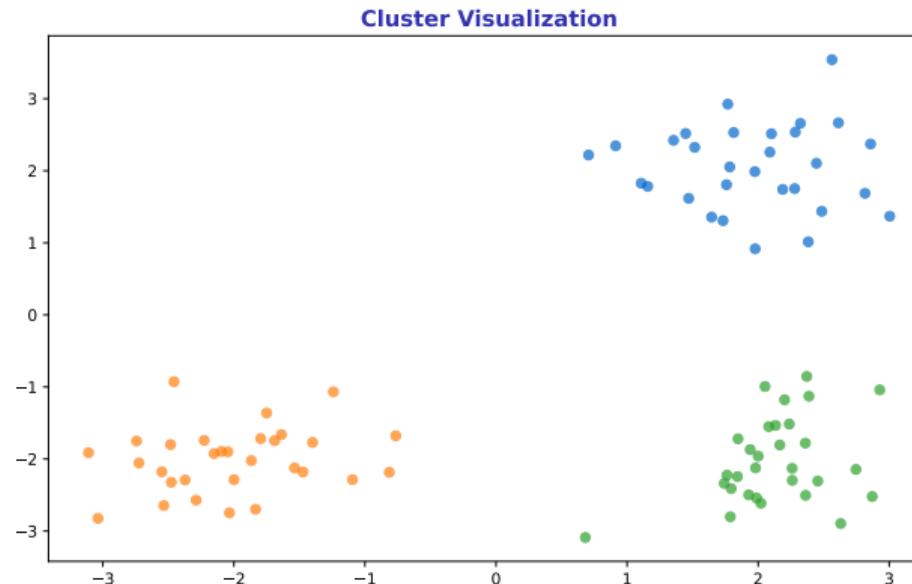


Silhouette > 0.5 indicates reasonable structure. Negative = probably wrong cluster.

# Cluster Visualization

## Seeing the Results

- 2D scatter plot with cluster colors
- For high-dimensional data: use PCA first, then plot

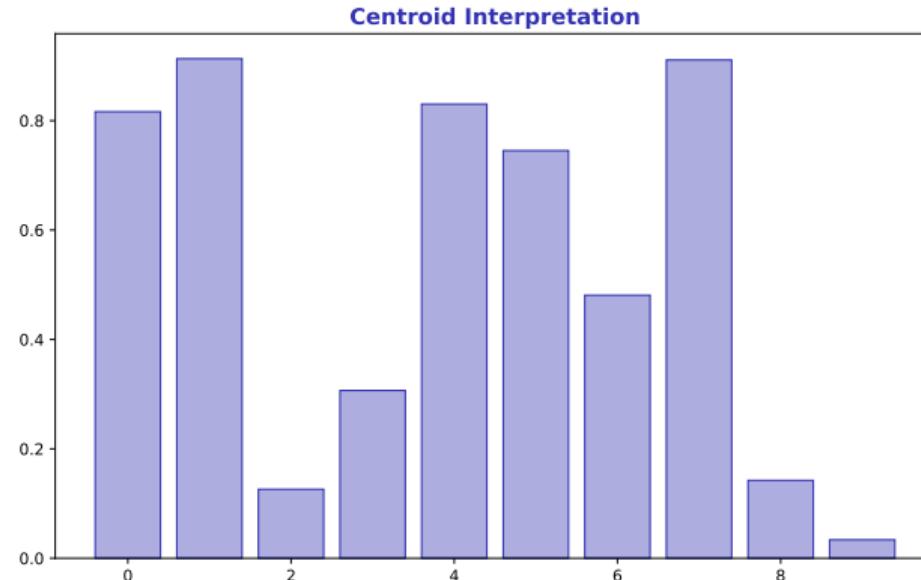


Always visualize clusters to sanity-check results

# Centroid Interpretation

## What Does Each Cluster Represent?

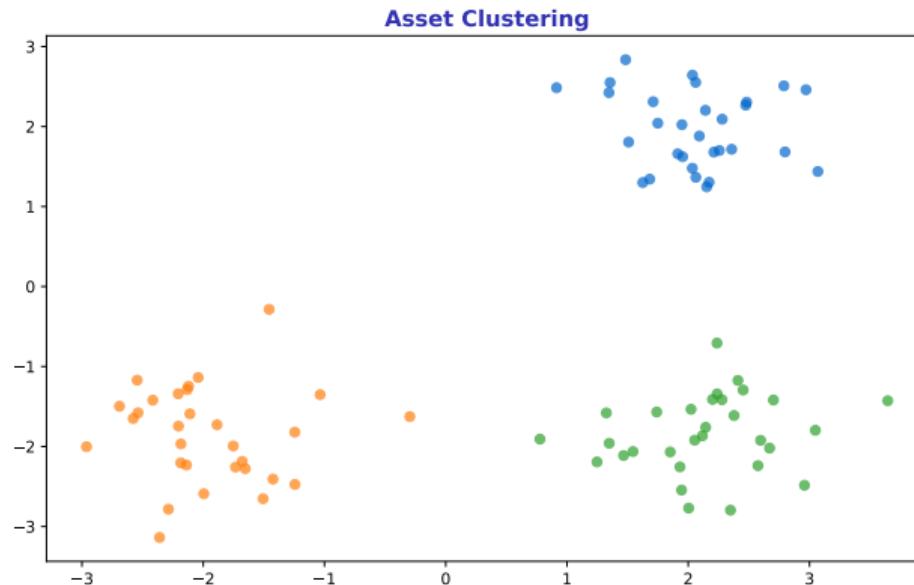
- Centroid = average feature values for that cluster
- Compare centroids to understand cluster characteristics



Example: Cluster 1 = high volatility, low volume; Cluster 2 = low volatility, high volume

## Finance Application: Stock Segmentation

- Features: returns, volatility, beta, market cap, sector
- Clusters reveal natural groupings beyond traditional sectors



Use for diversification: select one stock per cluster for uncorrelated portfolio

## Hands-On Exercise (25 min)

### Task: Cluster Stocks by Behavior

- ① Calculate features: 1-year return, volatility, beta for 50 stocks
- ② Standardize features (important for K-Means!)
- ③ Run K-Means with  $K=2,3,4,5$  – plot elbow curve
- ④ Choose best  $K$  using elbow + silhouette
- ⑤ Interpret centroids: what characterizes each cluster?

**Deliverable:** Elbow plot + scatter plot of clusters with labels.

**Extension:** Compare clusters to GICS sectors – do they align?

## Lesson Summary

**Problem Solved:** We can now discover natural groupings in data without predefined labels.

**Key Takeaways:**

- K-Means: iteratively assign points to K cluster centers
- Choose K: elbow method + silhouette score
- Always standardize features before clustering
- Interpret centroids to understand cluster meaning

**Next Lesson:** Hierarchical Clustering (L30) – clusters within clusters

**Memory:** K-Means minimizes inertia. Elbow = where curve bends. Silhouette = cluster quality.

## Lesson 30: Hierarchical Clustering

Data Science with Python – BSc Course

45 Minutes

**The Problem:** K-Means requires choosing K upfront. What if we want to see the full hierarchy of cluster relationships at all levels?

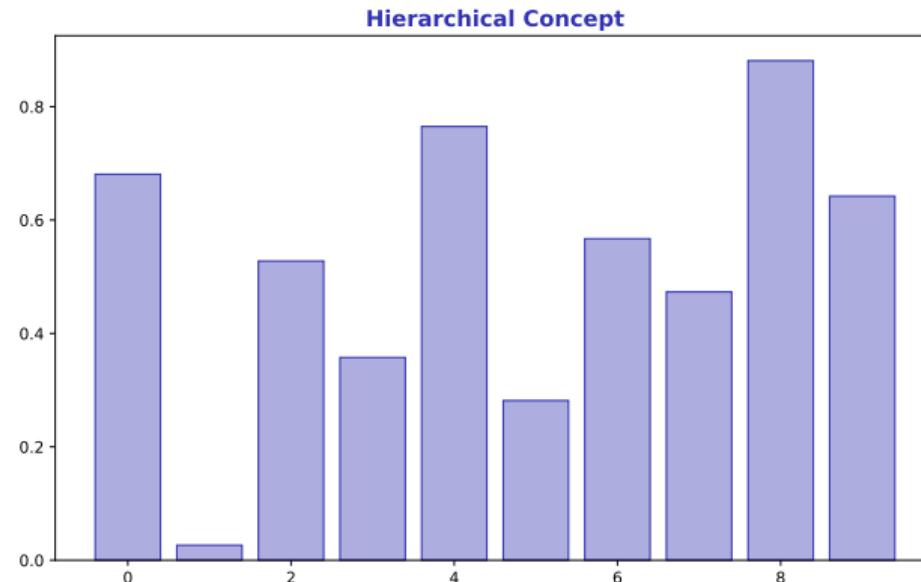
**After this lesson, you will be able to:**

- Build and interpret dendrograms
- Choose between linkage methods (single, complete, ward)
- Cut dendrograms to obtain flat clusters
- Apply hierarchical clustering to portfolio construction

**Finance Application: Hierarchical Risk Parity (HRP) portfolio optimization**

## Building a Tree of Clusters

- Agglomerative: start with N clusters, merge closest pairs
- Result: nested hierarchy showing relationships at all scales

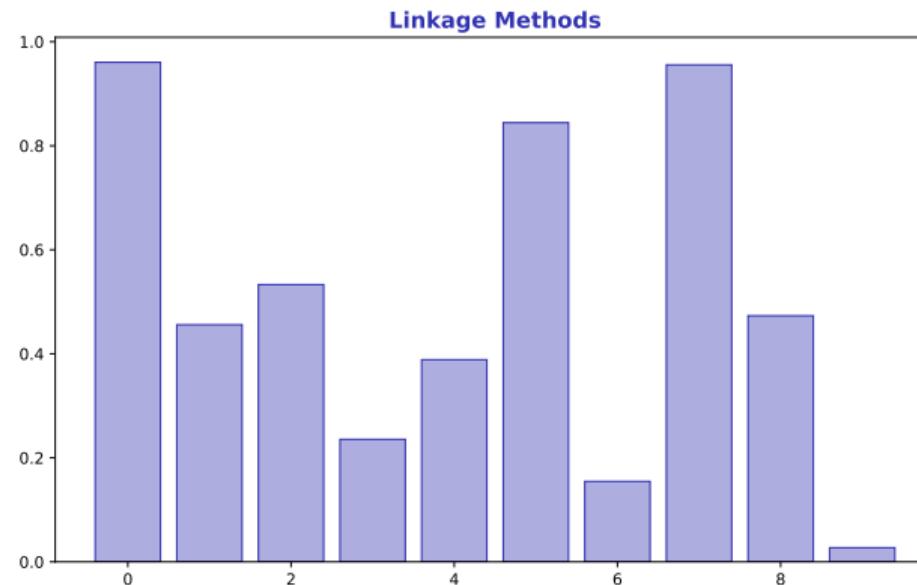


Advantage over K-Means: no need to specify K in advance

# Linkage Methods

## How to Measure Cluster Distance?

- Single: min distance between clusters (chains)
- Complete: max distance (compact clusters)
- Ward: minimize variance increase (balanced)

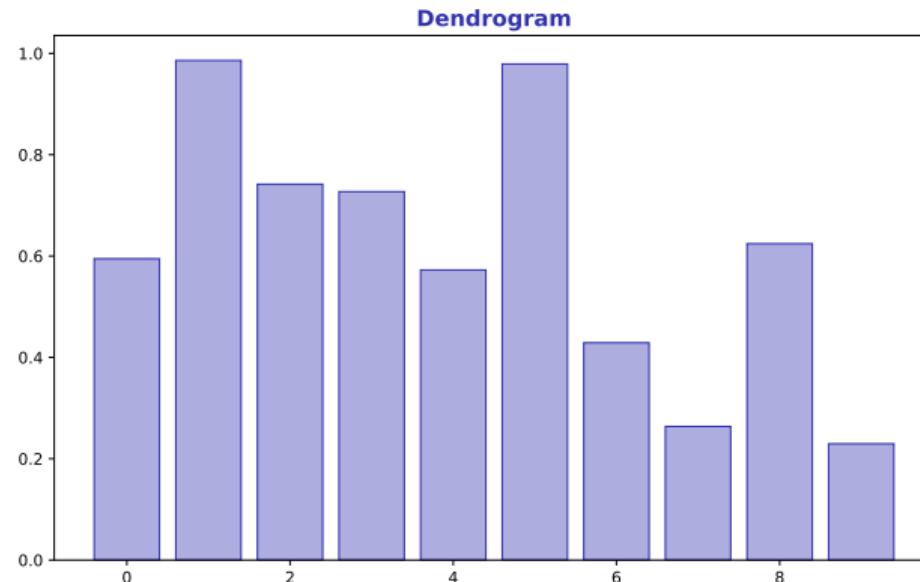


Default recommendation: Ward linkage for most applications

# Dendrogram

## Visualizing the Hierarchy

- Y-axis: distance at which clusters merge
- X-axis: individual observations (leaves)

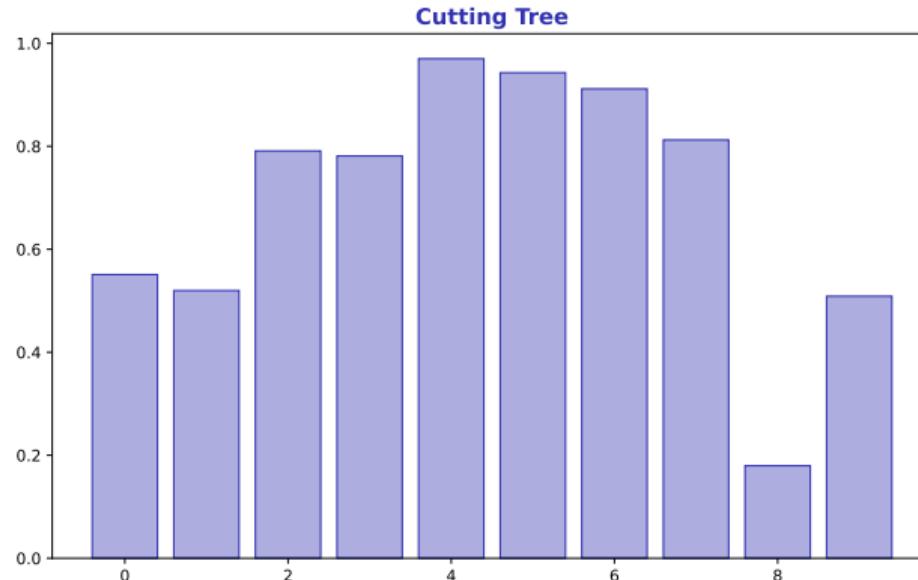


Read dendograms bottom-up: similar items merge early, different items merge late

# Cutting the Tree

## From Hierarchy to Flat Clusters

- Draw horizontal line at chosen height – clusters below are groups
- Higher cut = fewer clusters, lower cut = more clusters

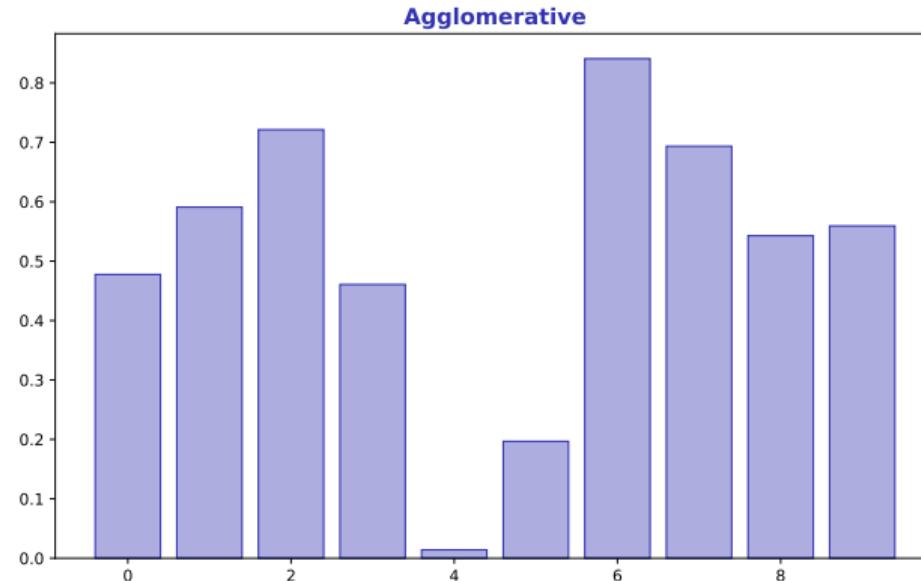


Use `fcluster(Z, t=height, criterion='distance')` to cut in scipy

# Agglomerative Clustering

## sklearn Implementation

- `from sklearn.cluster import AgglomerativeClustering`
- Specify `n_clusters` or `distance_threshold`

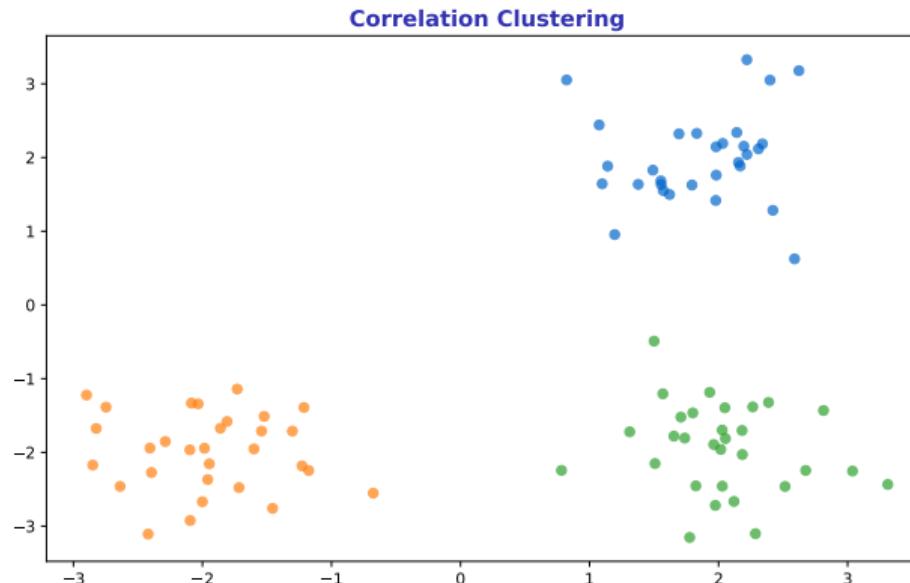


For dendrogram plotting, use `scipy.cluster.hierarchy` instead

# Correlation-Based Clustering

## Using Correlation as Similarity

- Distance =  $1 - \text{correlation}$  (or  $\sqrt{2(1 - \rho)}$ )
- Cluster assets by return correlation patterns



Finance standard: cluster correlation matrix to find asset groups

# Comparison with K-Means

## When to Use Which?

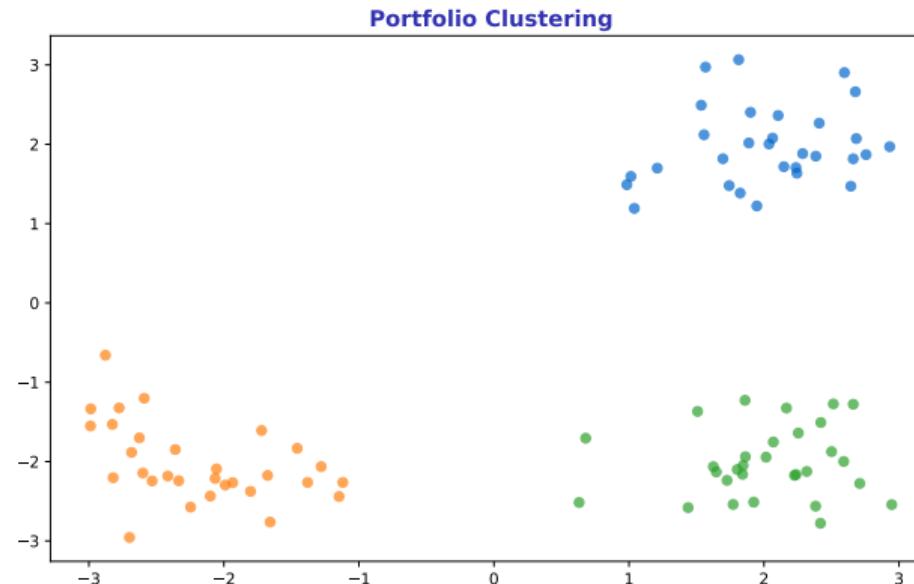
- K-Means: faster, need K, spherical clusters
- Hierarchical: slower, visual hierarchy, any cluster shape



Use hierarchical for exploratory analysis, K-Means for production

## Hierarchical Risk Parity (HRP)

- Cluster assets by correlation, then allocate within/across clusters
- More robust than mean-variance optimization



HRP: Modern portfolio construction using hierarchical clustering

## Hands-On Exercise (25 min)

### Task: Build Asset Hierarchy

- ① Calculate correlation matrix for 20 stocks (1 year daily returns)
- ② Convert to distance matrix:  $d = \sqrt{2(1 - \rho)}$
- ③ Build dendrogram with Ward linkage
- ④ Cut at 2-3 different heights – compare resulting clusters
- ⑤ Label clusters by dominant sector

**Deliverable:** Dendrogram with cluster cut lines annotated.

**Extension:** Implement simple HRP allocation based on your clusters

## Lesson Summary

**Problem Solved:** We can now discover hierarchical relationships and create clusters at any granularity.

**Key Takeaways:**

- Hierarchical clustering builds a tree (dendrogram)
- Linkage method matters: Ward for balanced clusters
- Cut dendrogram at desired height to get flat clusters
- Finance: correlation-based clustering for portfolio construction

**Next Lesson:** PCA (L31) – reducing dimensions while preserving information

**Memory:** Dendrogram = tree. Cut horizontally to get clusters. Ward = balanced.

## Lesson 31: PCA Dimensionality Reduction

Data Science with Python – BSc Course

45 Minutes

**The Problem:** We have 100 features but many are correlated. How do we reduce dimensions while preserving the most important information?

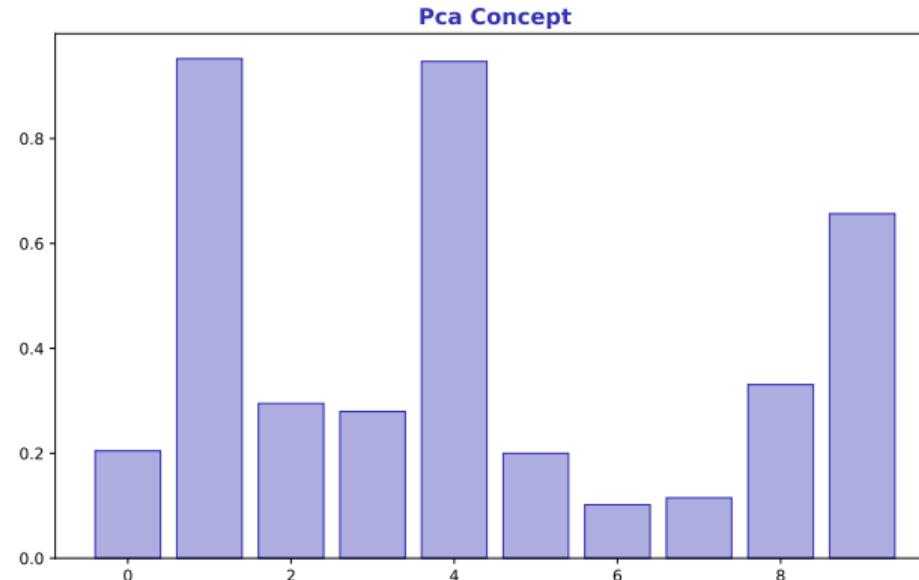
**After this lesson, you will be able to:**

- Understand principal components as new axes
- Apply PCA with sklearn
- Interpret explained variance ratio
- Reduce feature dimensions for visualization and modeling

**Finance Application:** Factor discovery, risk decomposition, noise reduction

## Finding New Axes

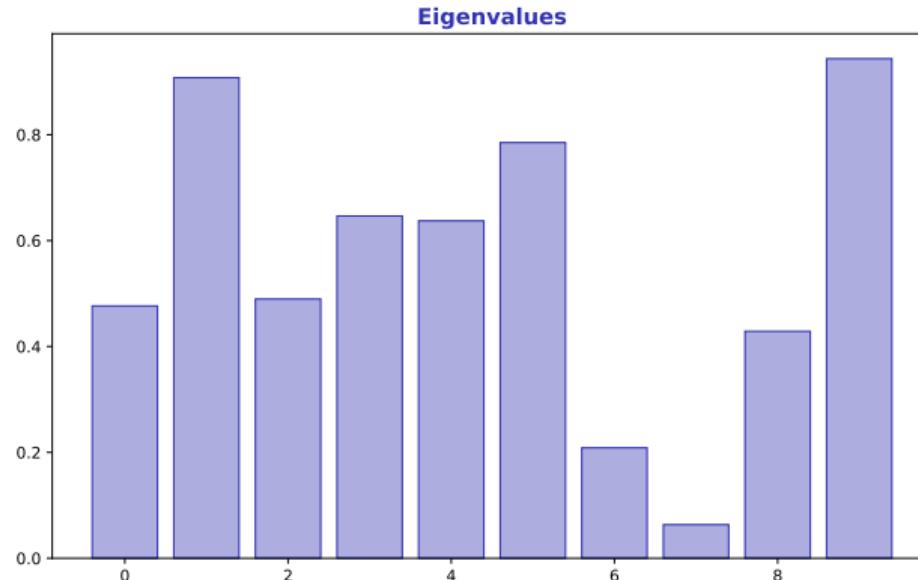
- PC1: direction of maximum variance
- PC2: orthogonal to PC1, captures next most variance



PCA rotates coordinate system to align with data's natural directions

## The Math Behind PCA

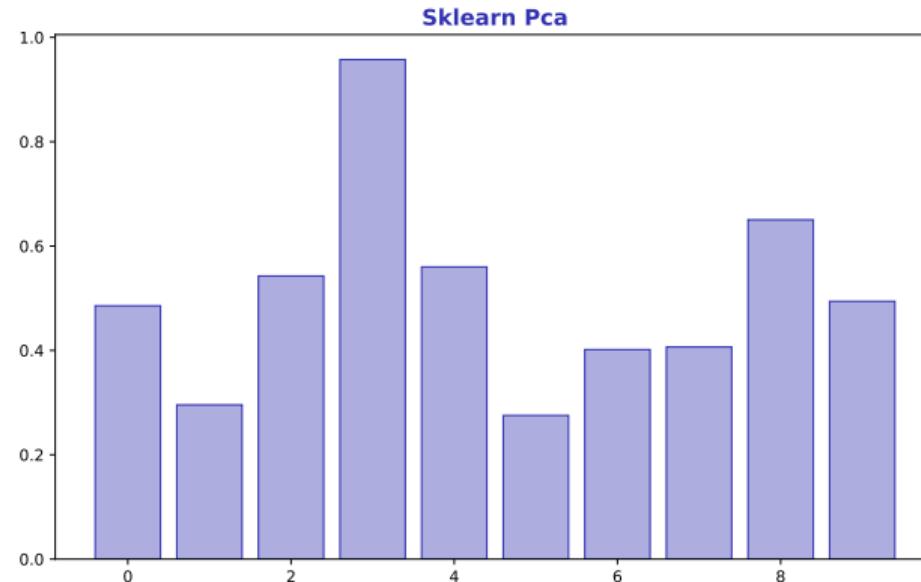
- Eigenvectors of covariance matrix = principal component directions
- Eigenvalues = variance explained by each component



Larger eigenvalue = more important component. Sum of eigenvalues = total variance.

## Implementation in Python

- `from sklearn.decomposition import PCA`
- `pca = PCA(n_components=2).fit_transform(X)`

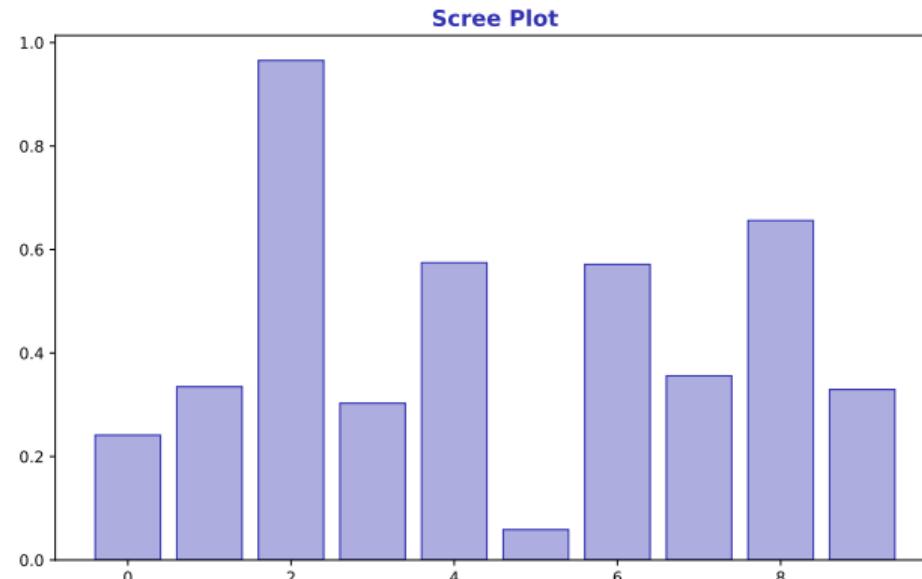


Always standardize features first! PCA is sensitive to scale.

# Scree Plot

## How Many Components to Keep?

- Plot eigenvalues in decreasing order
- Look for “elbow” where values level off

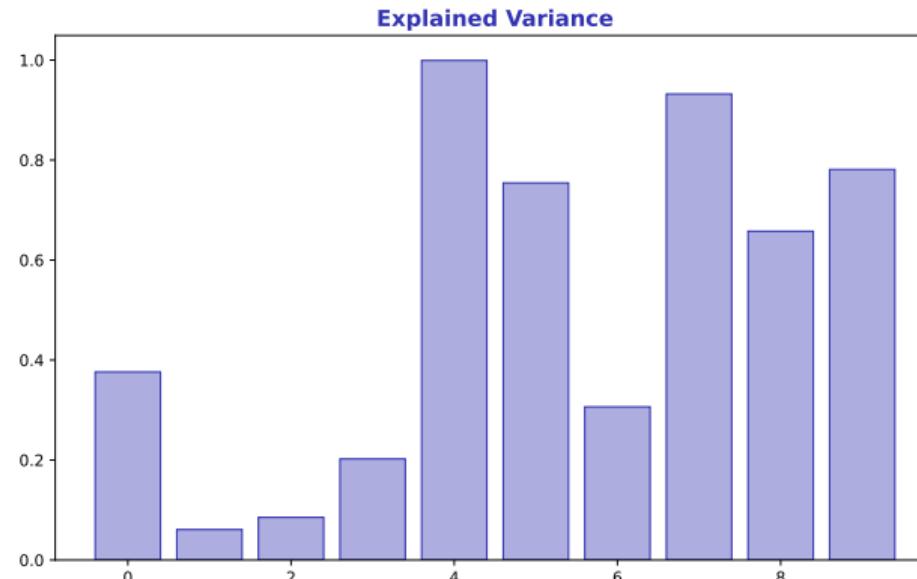


Alternative: keep components until cumulative variance > 90%

# Explained Variance

## Cumulative Information Retained

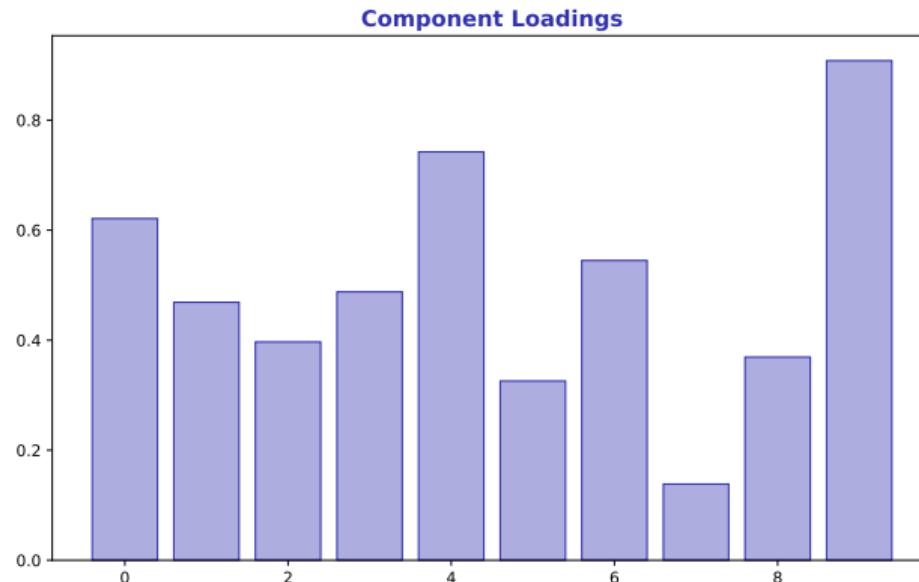
- `pca.explained_variance_ratio_` shows each component's share
- Cumulative sum tells total information retained



Example: 3 components explain 85% of variance – 97 features were mostly redundant

## Interpreting What Components Mean

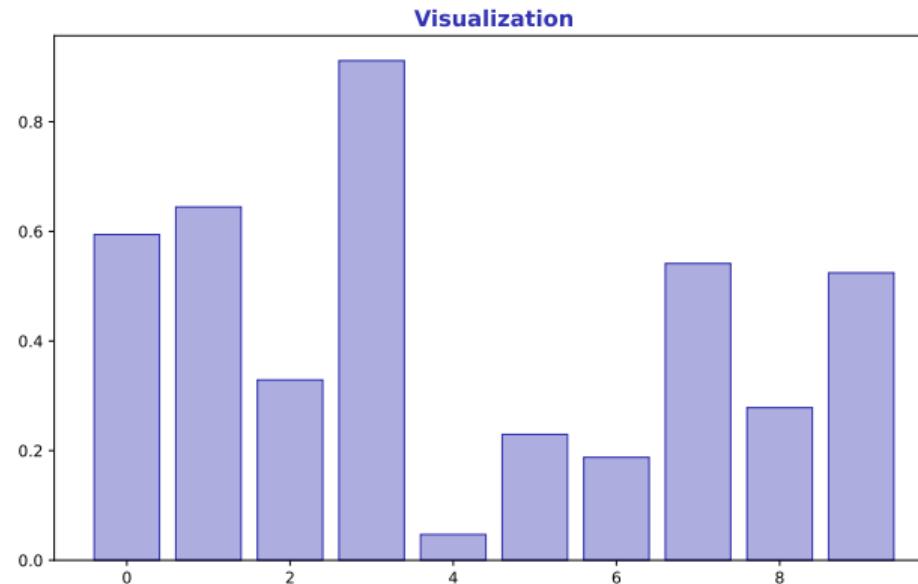
- Loadings = correlations between original features and components
- High loading = feature strongly influences that component



Finance: PC1 often = "market factor", PC2 = "size" or "value"

## Seeing High-Dimensional Data

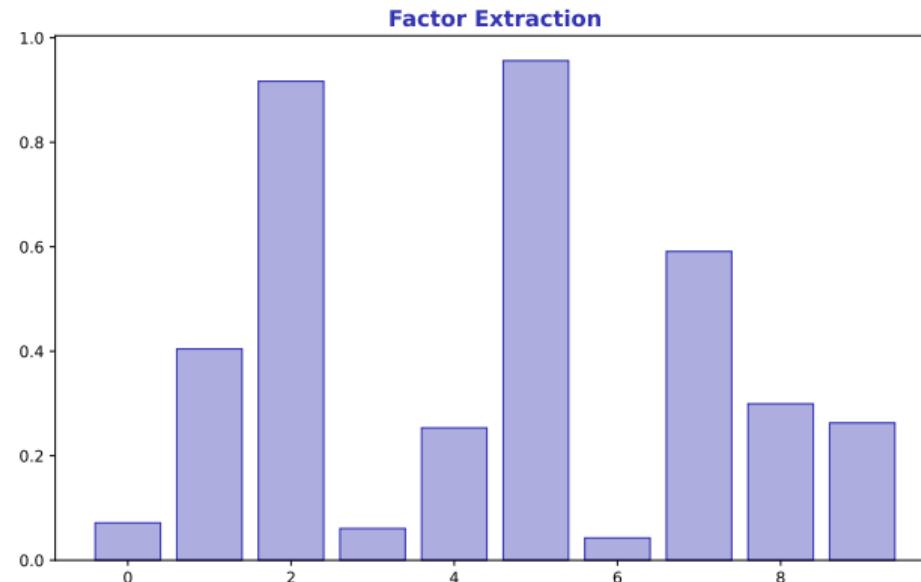
- Project to PC1 vs PC2 for 2D scatter plot
- Reveals clusters and outliers invisible in original space



PCA visualization is exploratory – always check explained variance

## Finance Application: Statistical Factors

- PCA on stock returns finds latent market factors
- First few PCs often correspond to market, sector, and style factors



Statistical PCA vs economic factors (Fama-French) – different but related

## Hands-On Exercise (25 min)

### Task: Discover Factors in Stock Returns

- ① Calculate daily returns for 30 stocks (1 year)
- ② Standardize returns and fit PCA
- ③ Plot scree plot – how many components needed for 80% variance?
- ④ Examine PC1 loadings – what does it represent?
- ⑤ Project stocks to PC1 vs PC2 and color by sector

**Deliverable:** Scree plot + 2D projection with sector colors.

**Extension:** Compare PC1 to S&P 500 returns – are they correlated?

## Lesson Summary

**Problem Solved:** We can now reduce high-dimensional data while preserving most information.

**Key Takeaways:**

- PCA finds orthogonal directions of maximum variance
- Scree plot and cumulative variance guide component selection
- Always standardize before PCA
- Finance: PCA extracts statistical factors from returns

**Next Lesson:** ML Pipeline (L32) – putting it all together

**Memory:** PCA = rotate to max variance axes. PC1 = most important direction.

## Lesson 32: Complete ML Pipeline

Data Science with Python – BSc Course

45 Minutes

## Learning Objectives

**The Problem:** We've learned many techniques separately. How do we combine preprocessing, feature engineering, and modeling into a reproducible, leak-free workflow?

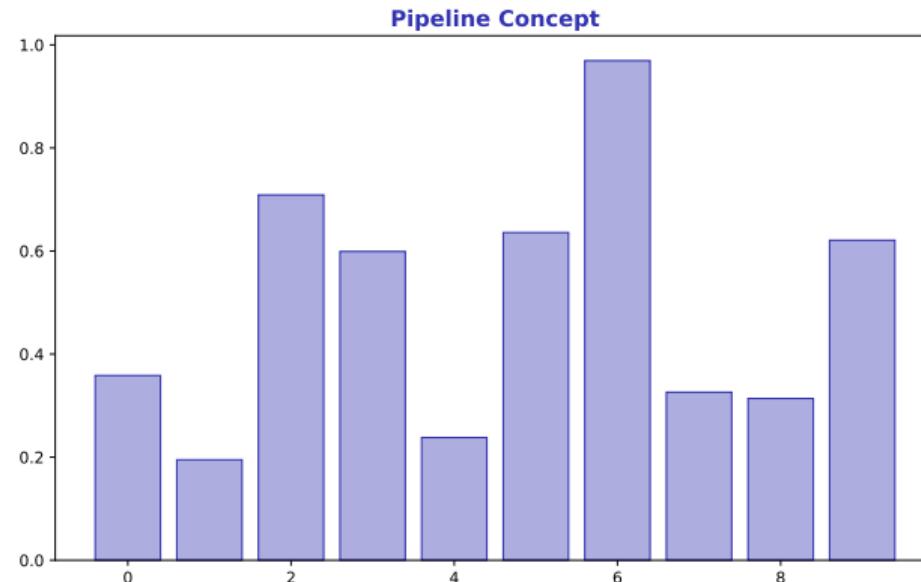
**After this lesson, you will be able to:**

- Build sklearn pipelines for end-to-end workflows
- Apply cross-validation correctly (avoiding data leakage)
- Tune hyperparameters with GridSearchCV and RandomizedSearchCV
- Handle time series data with proper train/test splits

**Finance Application:** Production-ready ML systems for trading and risk

## Chaining Steps Together

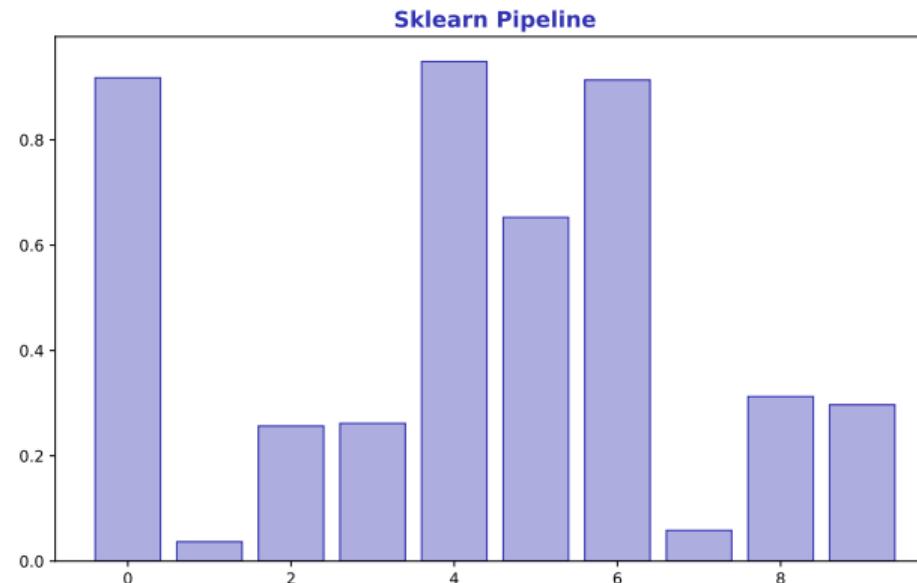
- Pipeline = sequence of transformers + final estimator
- Each step's output becomes next step's input



Pipelines ensure transformations are applied consistently to train and test data

## Building Your First Pipeline

- `Pipeline([('scaler', StandardScaler()), ('model', Ridge())])`
- Call `.fit(X_train, y_train)` and `.predict(X_test)`

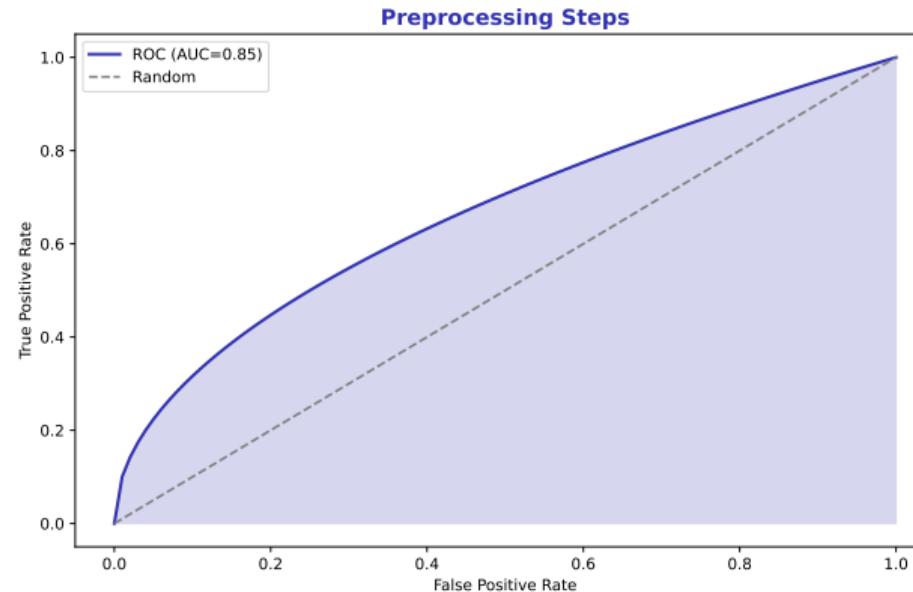


Naming convention: `('step_name', transformer_object)`

# Preprocessing Steps

## Common Transformers

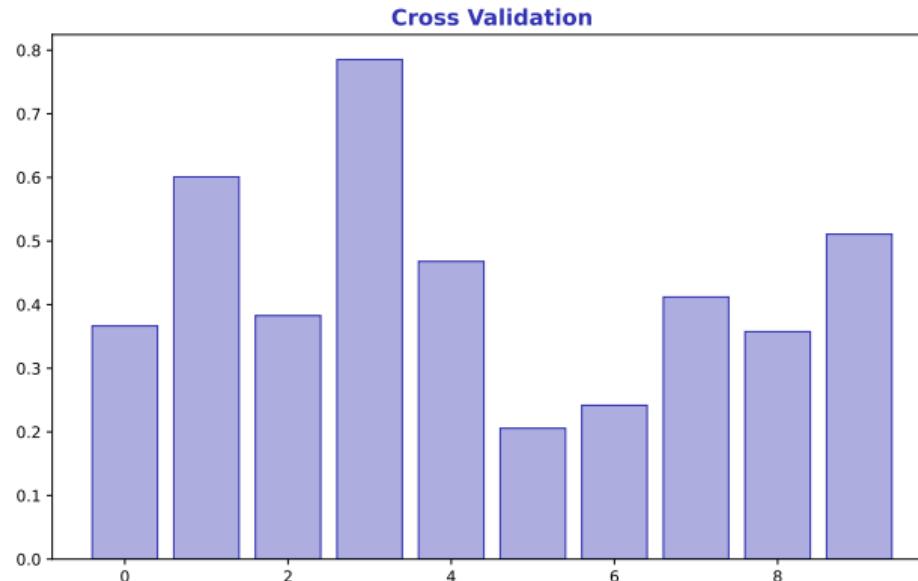
- StandardScaler, MinMaxScaler for numeric features
- OneHotEncoder for categorical features
- SimpleImputer for missing values



Use ColumnTransformer to apply different transforms to different columns

## Robust Performance Estimation

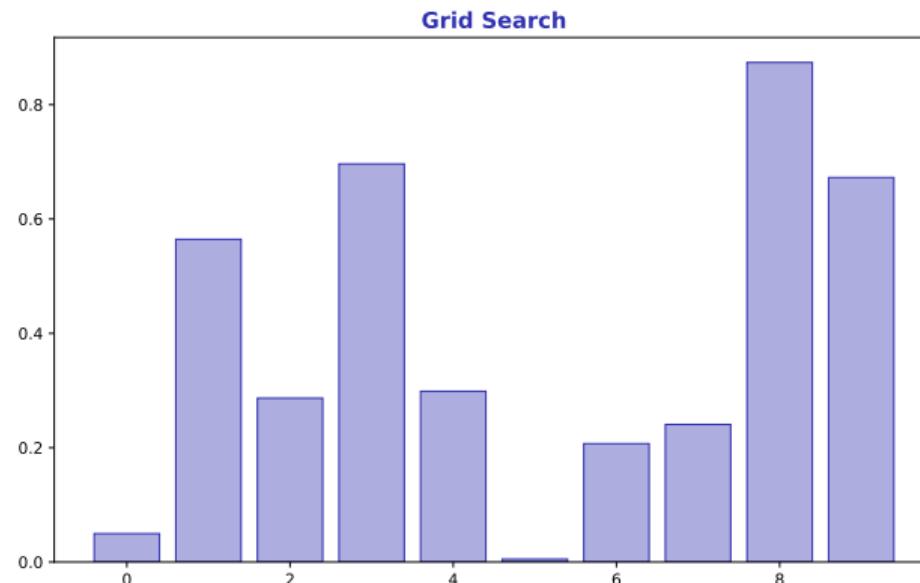
- K-Fold: split data K ways, train on K-1, test on 1, rotate
- `cross_val_score(pipeline, X, y, cv=5)` returns K scores



CV inside pipeline = transformers fit only on training fold (no leakage)

## Exhaustive Hyperparameter Search

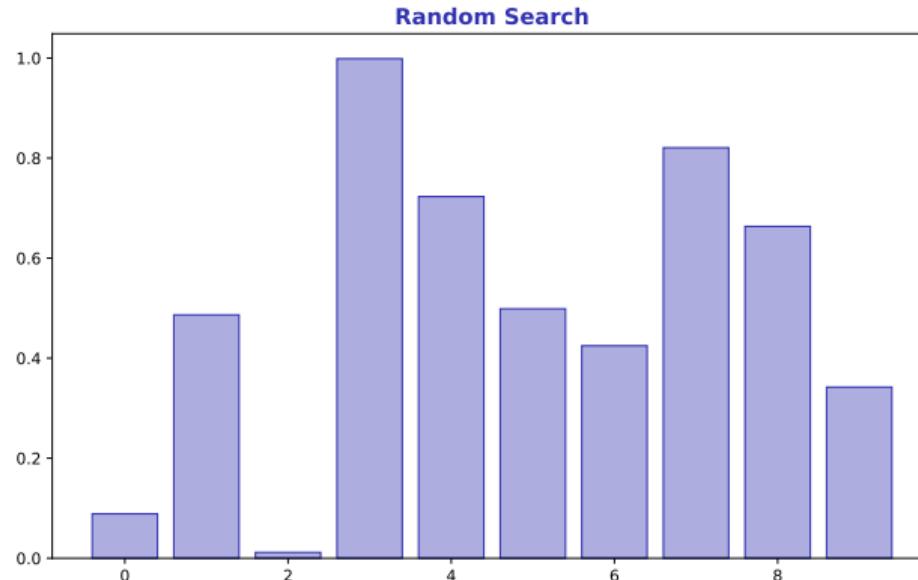
- Define parameter grid: `{'model_alpha': [0.1, 1, 10]}`
- GridSearchCV tries all combinations with CV



Access best params via `grid.best_params_` and best score via `grid.best_score_`.

## Efficient Search for Large Spaces

- Sample random combinations instead of exhaustive grid
- Specify distributions: `uniform(0.01, 10)` for continuous params

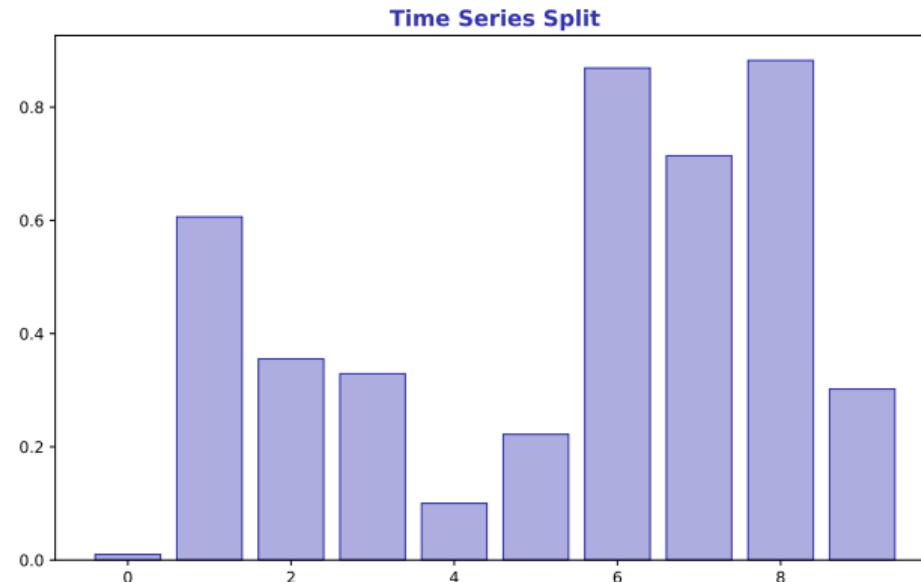


Rule: Use random search when grid has  $>100$  combinations

# Time Series Split

## Respecting Temporal Order

- Standard CV shuffles – invalid for time series
- TimeSeriesSplit: train on past, test on future (rolling)

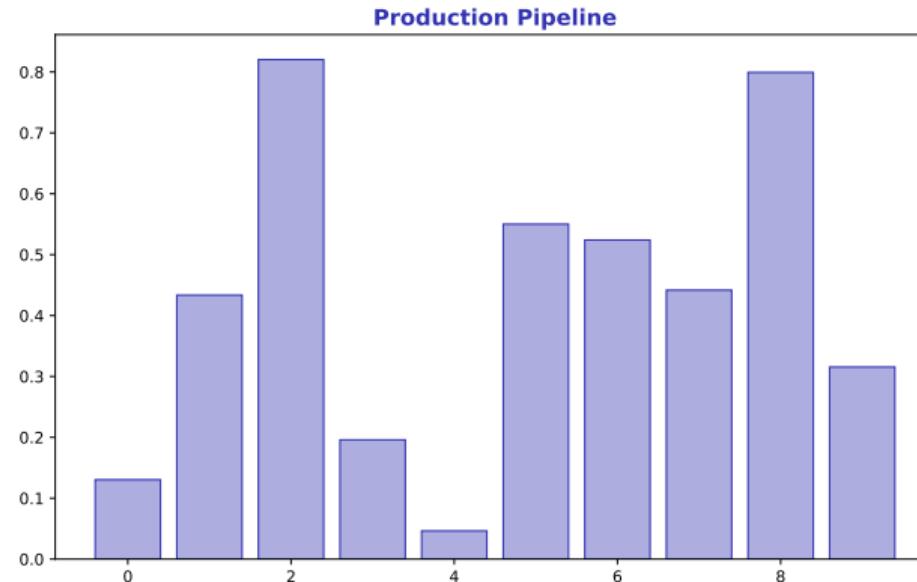


**Finance critical:** Never let future data leak into training

# Production Pipeline

## From Development to Deployment

- Save entire pipeline with `joblib.dump(pipe, 'model.pkl')`
- Load and predict: `pipe = joblib.load('model.pkl')`



Pipeline saves all preprocessing steps – deploy once, predict anywhere

## Hands-On Exercise (25 min)

### Task: Build End-to-End Prediction Pipeline

- ① Create pipeline: StandardScaler → PCA(5) → Ridge
- ② Use GridSearchCV to tune `pca_n_components` and `ridge_alpha`
- ③ Evaluate with TimeSeriesSplit (5 splits)
- ④ Print best parameters and cross-validation score
- ⑤ Save best pipeline to disk with joblib

**Deliverable:** Best params + CV score + saved model file.

**Extension:** Add `ColumnTransformer` for mixed numeric/categorical features

## Lesson Summary

**Problem Solved:** We can now build complete, reproducible ML workflows that prevent data leakage.

**Key Takeaways:**

- Pipelines chain preprocessing and modeling together
- Cross-validation estimates generalization performance
- GridSearchCV/RandomizedSearchCV for hyperparameter tuning
- TimeSeriesSplit for financial data – never leak future

**Next Lesson:** Perceptron (L33) – introduction to neural networks

**Memory:** Pipeline = chain. GridSearchCV = try all. TimeSeriesSplit = respect time.

## Lesson 33: Perceptron

Data Science with Python – BSc Course

45 Minutes

## Learning Objectives

**The Problem:** Traditional ML models have fixed architectures. How do neural networks learn? Understanding the perceptron is the first step toward deep learning.

**After this lesson, you will be able to:**

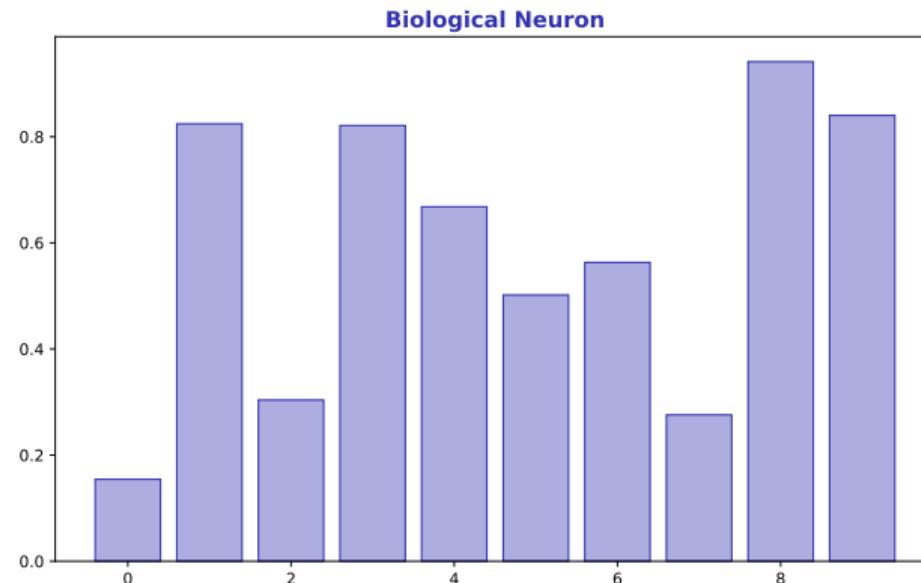
- Understand the biological inspiration for neural networks
- Build and train a single perceptron
- Recognize the limitation: only linearly separable problems
- Prepare for multi-layer networks that overcome this

**Finance Application: Foundation for deep learning models in quantitative finance**

# Biological Neuron

## Inspiration from the Brain

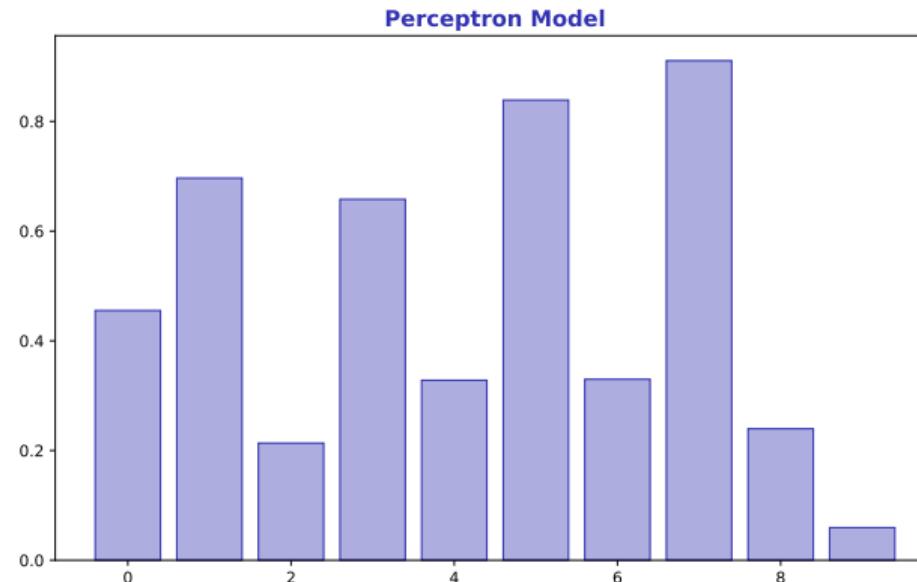
- Dendrites receive signals, soma processes, axon outputs
- Neuron “fires” when input exceeds threshold



Perceptron: simplified mathematical model of biological neuron

## The Simplest Neural Network

- Output:  $y = \sigma(\sum w_i x_i + b)$  where  $\sigma$  is activation
- Weights  $w_i$  control importance of each input

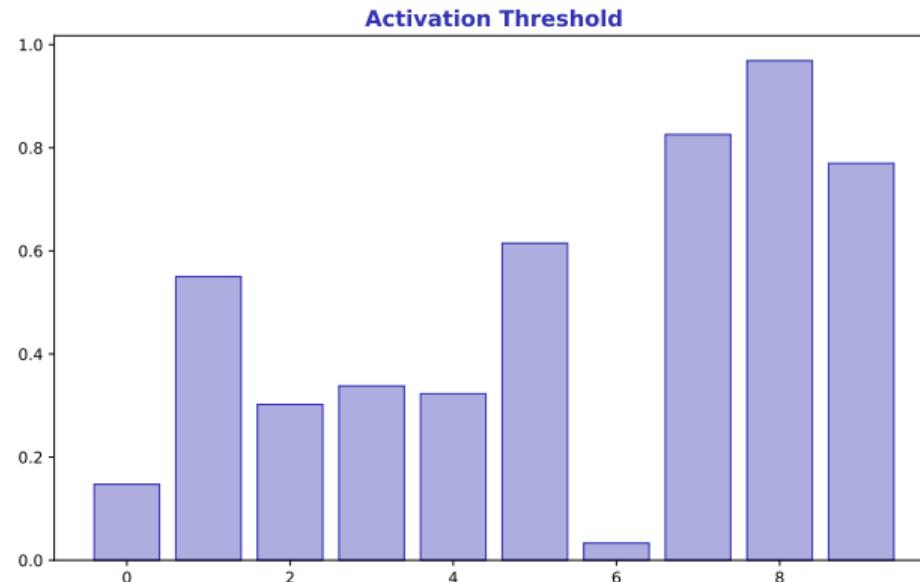


Single perceptron = logistic regression (same math, different framing)

# Activation Threshold

## When Does the Neuron Fire?

- Step function: output 1 if weighted sum  $> 0$ , else 0
- Modern: sigmoid, ReLU for smooth gradients

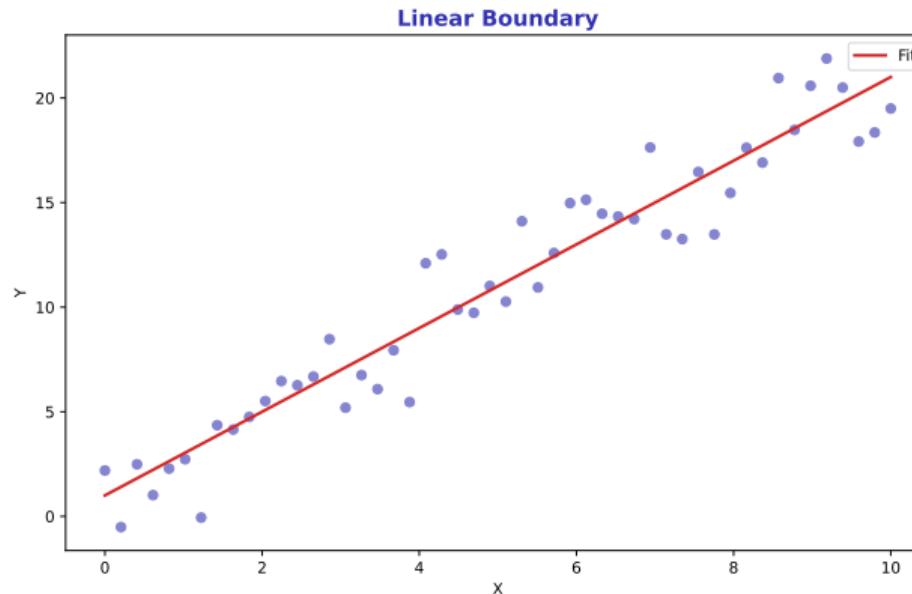


Step function: original perceptron. Sigmoid: smooth for gradient descent.

# Linear Decision Boundary

## What Can a Perceptron Learn?

- Single perceptron creates a linear boundary (hyperplane)
- Can separate AND, OR but not more complex patterns

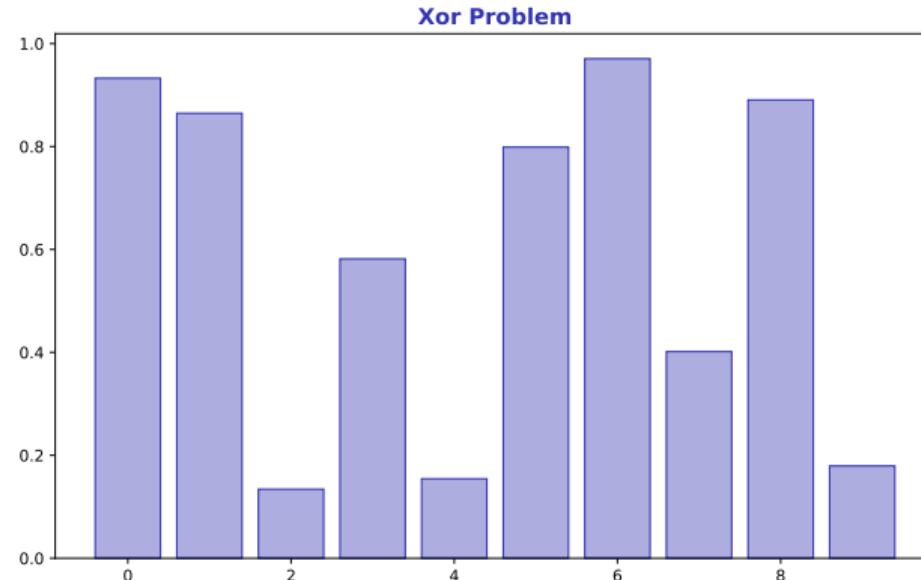


Perceptron = linear classifier. Same limitation as logistic regression.

# The XOR Problem

## A Famous Limitation

- XOR is not linearly separable – no single line can divide it
- This limitation stalled neural network research for years

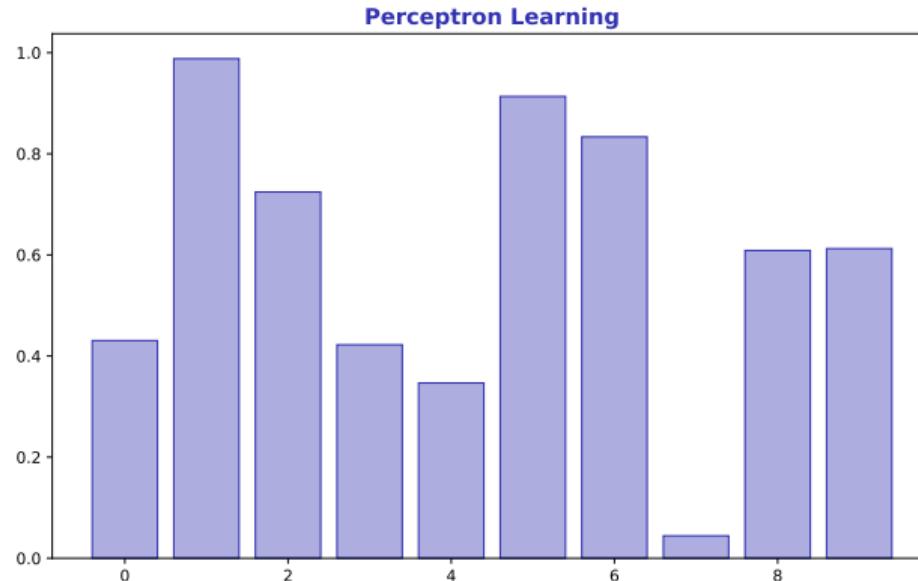


Solution: add hidden layers (multi-layer perceptron) to learn non-linear boundaries

# Perceptron Learning Rule

## How Weights Are Updated

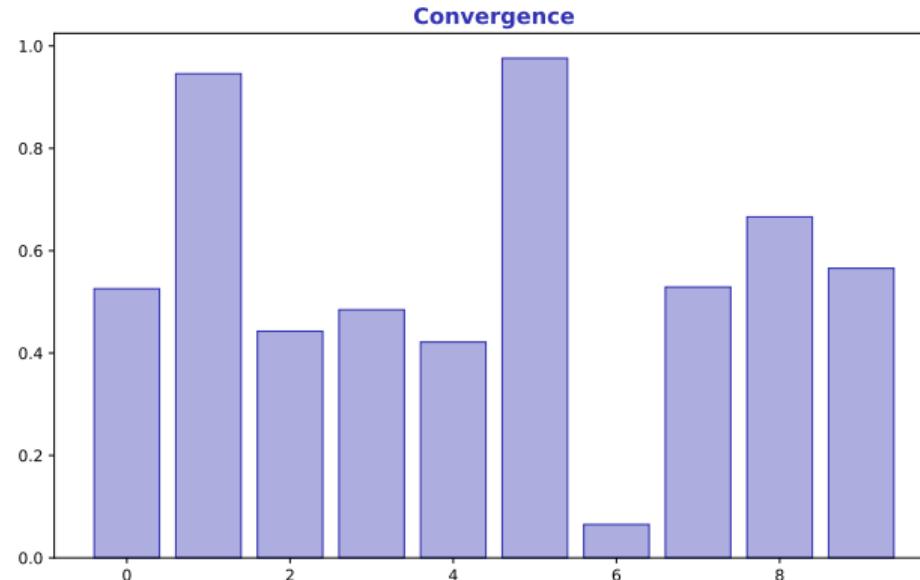
- If wrong:  $w_{new} = w_{old} + \eta \cdot (y_{true} - y_{pred}) \cdot x$
- Learning rate  $\eta$  controls step size



Perceptron learning rule: simple but powerful for linearly separable data

## When Will Training Stop?

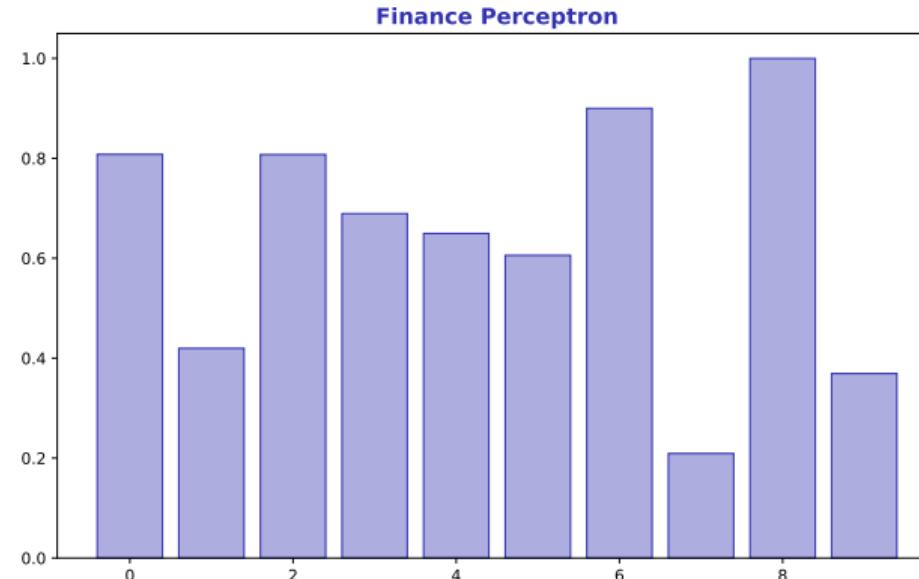
- If data is linearly separable, perceptron converges in finite steps
- If not separable, it oscillates forever (need multi-layer)



Perceptron Convergence Theorem (1962): guaranteed for separable problems

## Simple Trading Signals

- Inputs: momentum, volatility, volume signals
- Output: buy (1) or sell (0) decision



Reality: Financial patterns are rarely linearly separable – need deeper networks

## Hands-On Exercise (25 min)

### Task: Implement Perceptron from Scratch

- ① Create synthetic linearly separable 2D data
- ② Implement perceptron learning rule in Python (no sklearn)
- ③ Train and plot decision boundary at each epoch
- ④ Try on XOR data – observe failure to converge
- ⑤ Compare to sklearn's Perceptron class

**Deliverable:** Animation of boundary evolution + XOR failure plot.

**Extension:** Modify learning rate – how does convergence speed change?

## Lesson Summary

**Problem Solved:** We understand the building block of neural networks and its limitations.

**Key Takeaways:**

- Perceptron: weighted sum + activation = binary output
- Only learns linearly separable patterns
- XOR problem: needs hidden layers to solve
- Foundation for multi-layer perceptrons (MLPs)

**Next Lesson:** MLPs and Activations (L34) – adding hidden layers

**Memory:** Perceptron = single neuron = linear boundary. XOR needs hidden layers.

## Lesson 34: MLPs and Activations

Data Science with Python – BSc Course

45 Minutes

**The Problem:** Single perceptrons can only learn linear boundaries. How do we build networks that learn complex, non-linear patterns?

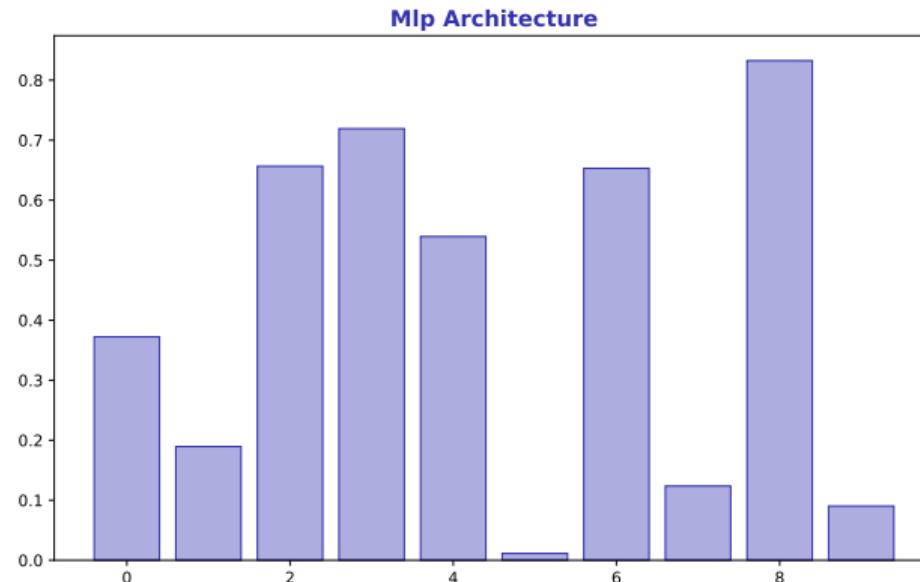
**After this lesson, you will be able to:**

- Design multi-layer perceptron (MLP) architectures
- Choose appropriate activation functions (ReLU, sigmoid, softmax)
- Build neural networks with Keras
- Apply MLPs to non-linear classification problems

**Finance Application:** Non-linear regime detection and pattern recognition

## Adding Hidden Layers

- Input layer → Hidden layer(s) → Output layer
- Hidden layers learn intermediate representations

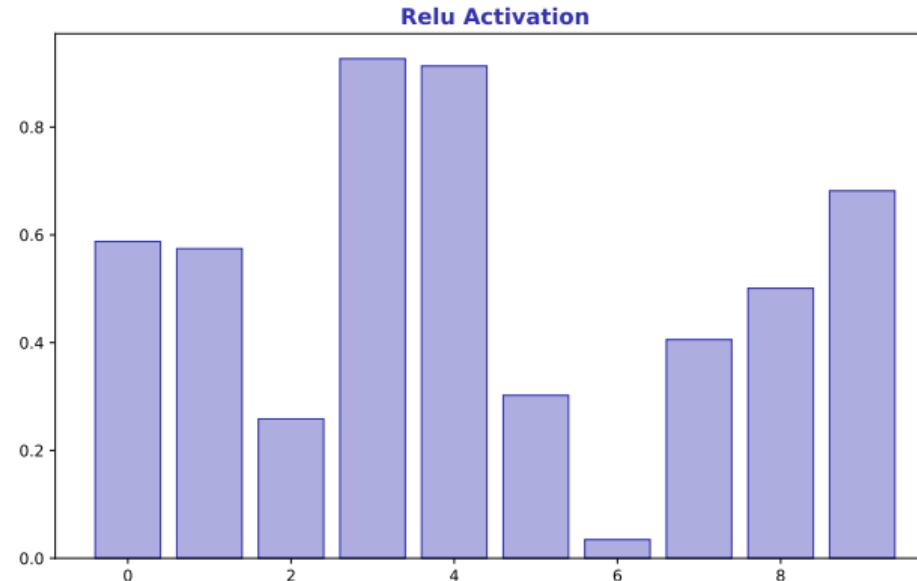


More layers = more complex patterns, but also more parameters to train

# ReLU Activation

## The Modern Default

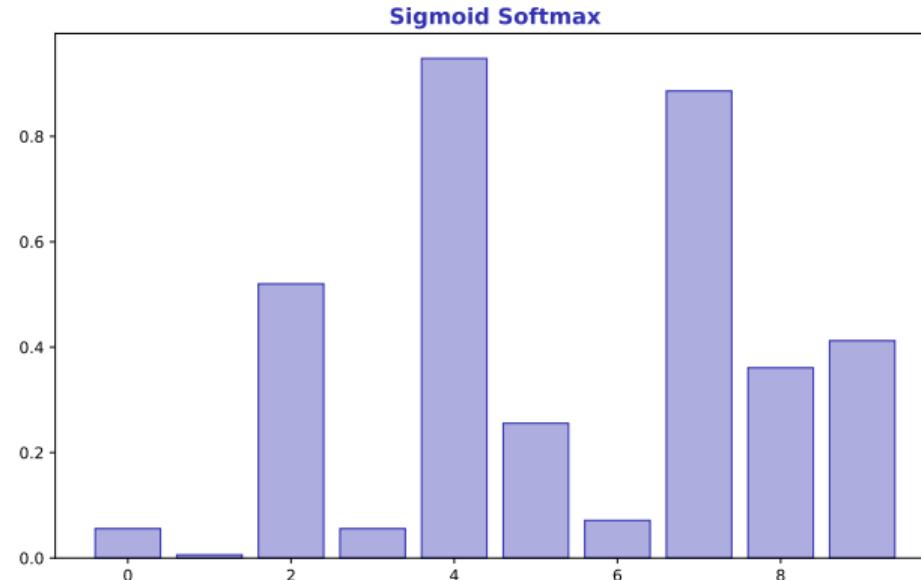
- $\text{ReLU}(x) = \max(0, x)$  – simple, fast, effective
- Solves vanishing gradient problem of sigmoid



Use ReLU for hidden layers. It's the default choice in 2024.

## Output Layer Activations

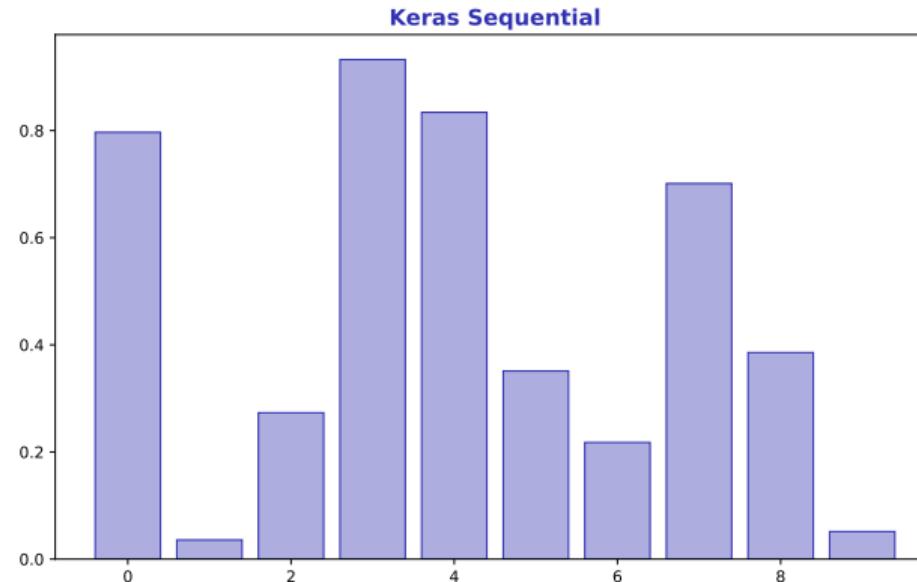
- Sigmoid: binary classification (output in [0,1])
- Softmax: multi-class (outputs sum to 1)



Rule: ReLU for hidden layers, sigmoid/softmax for output layer

## Building Networks in Python

- `model = Sequential([Dense(64, activation='relu'), Dense(1, activation='sigmoid')])`
- `model.compile(optimizer='adam', loss='binary_crossentropy')`

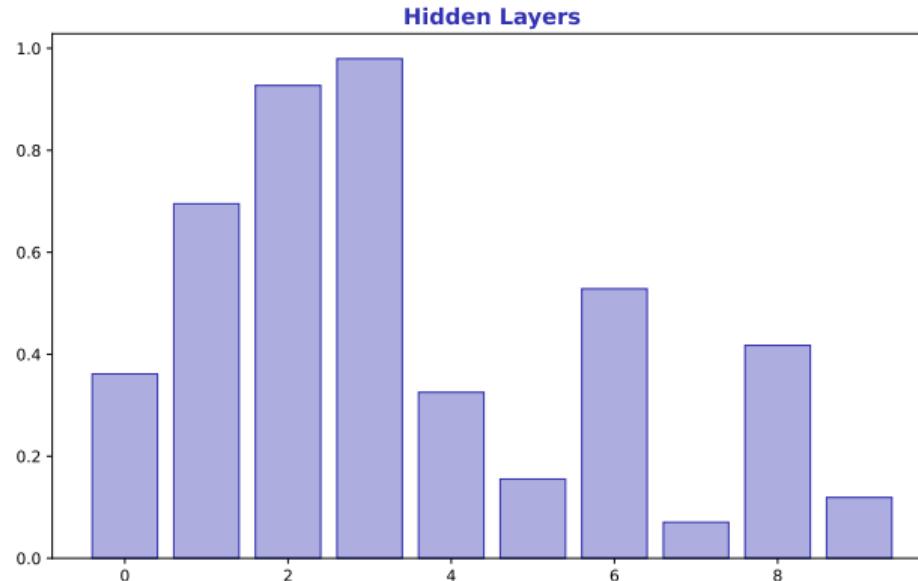


Keras pattern: add layers sequentially, compile, fit, predict

# Choosing Hidden Layers

## How Many Neurons? How Many Layers?

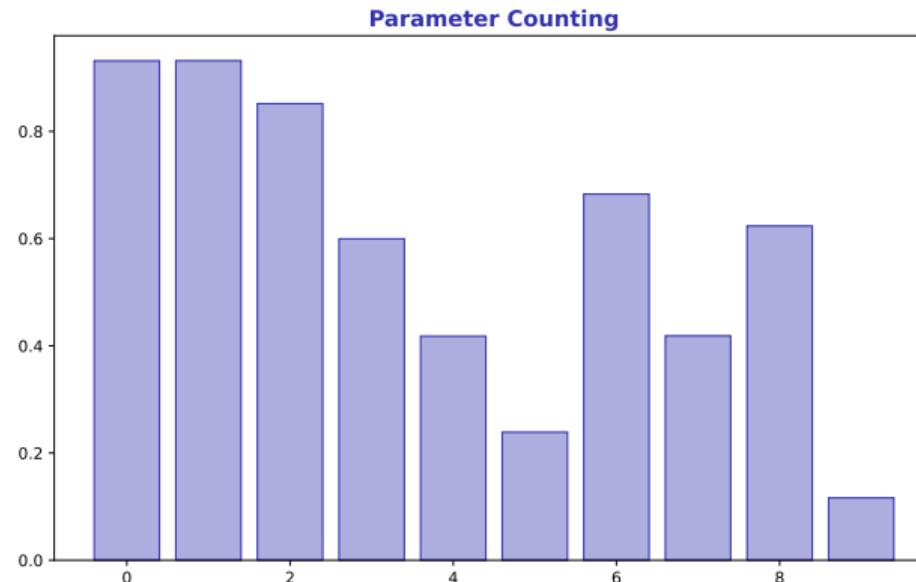
- Start simple: 1-2 hidden layers, 32-128 neurons
- More complex patterns need deeper networks



Rule of thumb: start small, increase if underfitting

## How Many Weights to Learn?

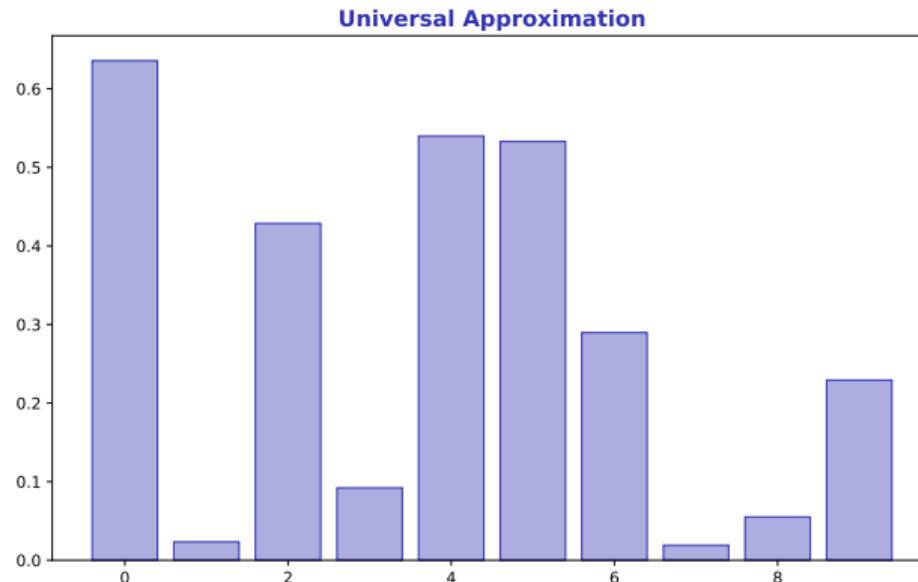
- Dense layer:  $(\text{inputs} + 1) \times \text{outputs}$  parameters
- More parameters = more expressive but slower, prone to overfit



Check `model.summary()` to see parameter count

## Why Neural Networks Are Powerful

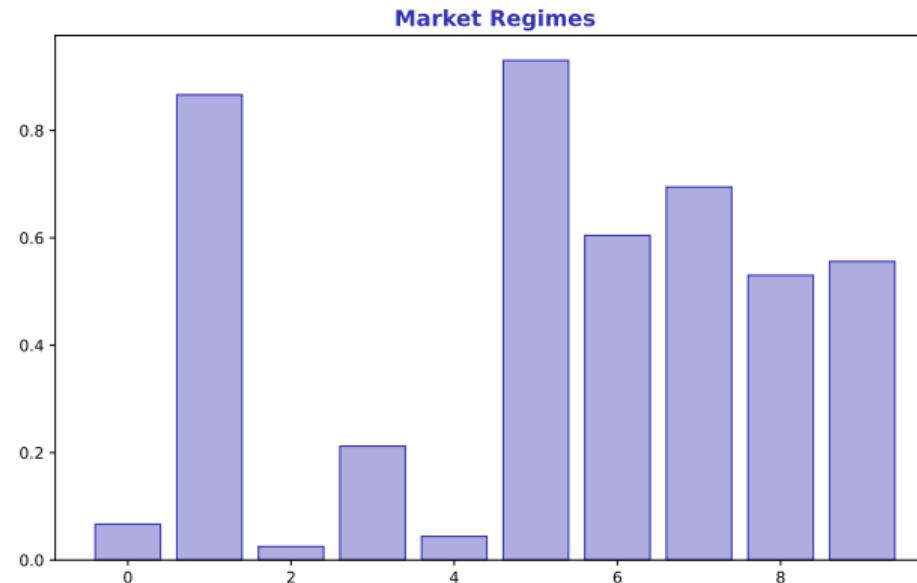
- Theorem: MLP with one hidden layer can approximate any continuous function
- Caveat: may need exponentially many neurons



Theory says possible; practice says depth often works better than width

## Finance Application

- Inputs: volatility, momentum, correlation features
- Output: regime class (bull, bear, sideways)



MLPs can capture non-linear regime boundaries that linear models miss

## Hands-On Exercise (25 min)

### Task: Build MLP for XOR and Beyond

- ① Create XOR dataset and verify perceptron fails
- ② Build MLP: 2 inputs → 4 hidden (ReLU) → 1 output (sigmoid)
- ③ Train and verify 100% accuracy on XOR
- ④ Visualize decision boundary – observe non-linearity
- ⑤ Apply to 3-class classification with softmax output

**Deliverable:** XOR decision boundary plot + 3-class accuracy.

**Extension:** Try different hidden layer sizes – how does decision boundary change?

## Lesson Summary

**Problem Solved:** Multi-layer perceptrons overcome the linear limitation and learn complex patterns.

**Key Takeaways:**

- Hidden layers enable non-linear decision boundaries
- ReLU for hidden layers, sigmoid/softmax for output
- Keras: Sequential API makes building networks easy
- Universal approximation: MLPs can learn any pattern (in theory)

**Next Lesson:** Backpropagation (L35) – how networks actually learn

**Memory:**  $\text{ReLU} = \max(0, x)$ . Hidden layers = non-linear power. Keras = easy MLPs.

## Lesson 35: Backpropagation

Data Science with Python – BSc Course

45 Minutes

## Learning Objectives

**The Problem:** Neural networks have millions of weights. How does the network figure out which weights to adjust and by how much?

**After this lesson, you will be able to:**

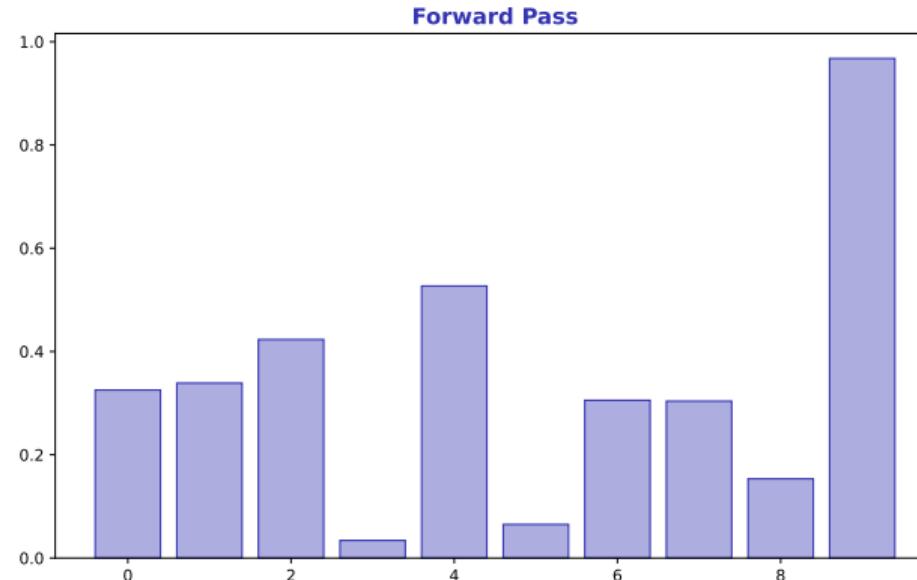
- Understand gradient descent optimization
- Interpret loss curves and diagnose training issues
- Configure learning rate and its effects
- Monitor training progress with validation metrics

**Finance Application: Training predictive models on financial data**

# Forward Pass

## Computing Predictions

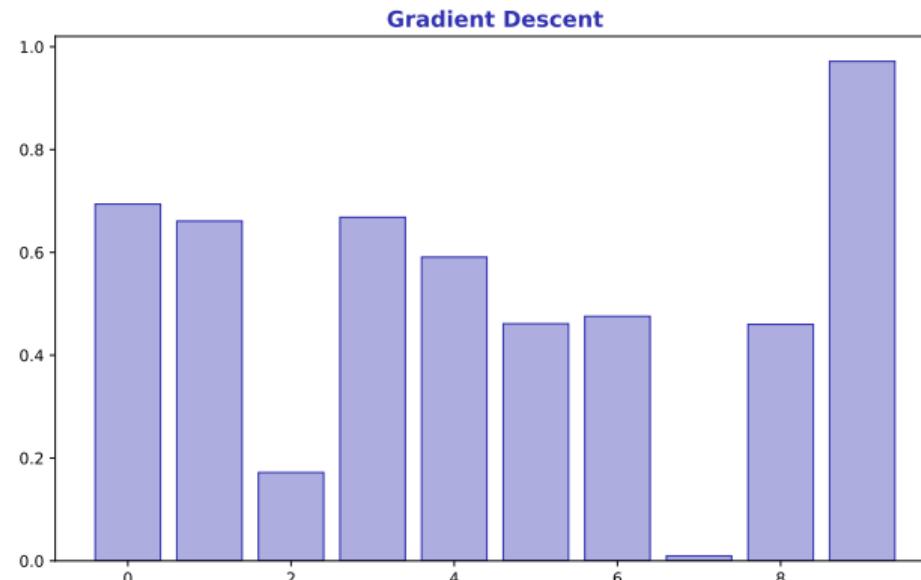
- Input flows through network layer by layer
- Each layer:  $z = Wx + b$ , then  $a = \sigma(z)$



Forward pass: input → hidden(s) → output = prediction

## Walking Downhill on the Loss Surface

- Compute gradient:  $\frac{\partial L}{\partial w}$  tells direction of steepest increase
- Update:  $w_{new} = w_{old} - \eta \cdot \frac{\partial L}{\partial w}$

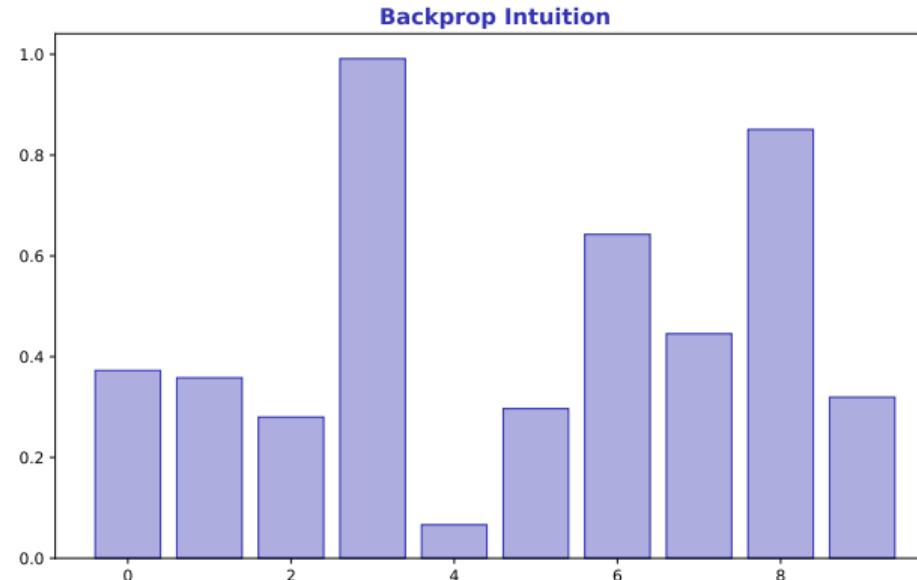


Gradient descent: repeatedly move opposite to gradient until minimum

# Backpropagation Intuition

## Efficiently Computing Gradients

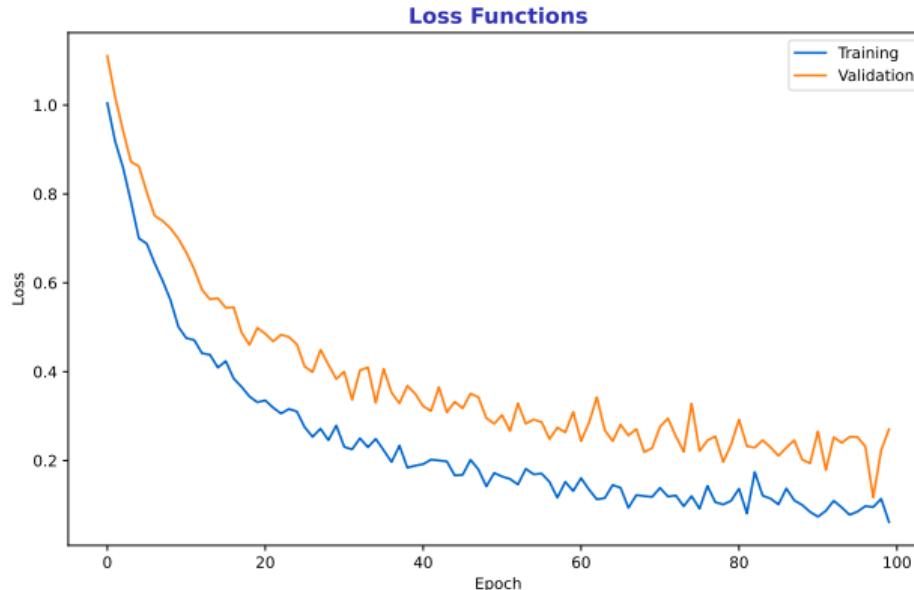
- Chain rule: propagate error backward through layers
- Each layer's gradient depends on layers after it



Backprop = efficient gradient computation via chain rule (not a learning rule)

## What Are We Minimizing?

- MSE: regression problems
- Cross-entropy: classification (binary or categorical)

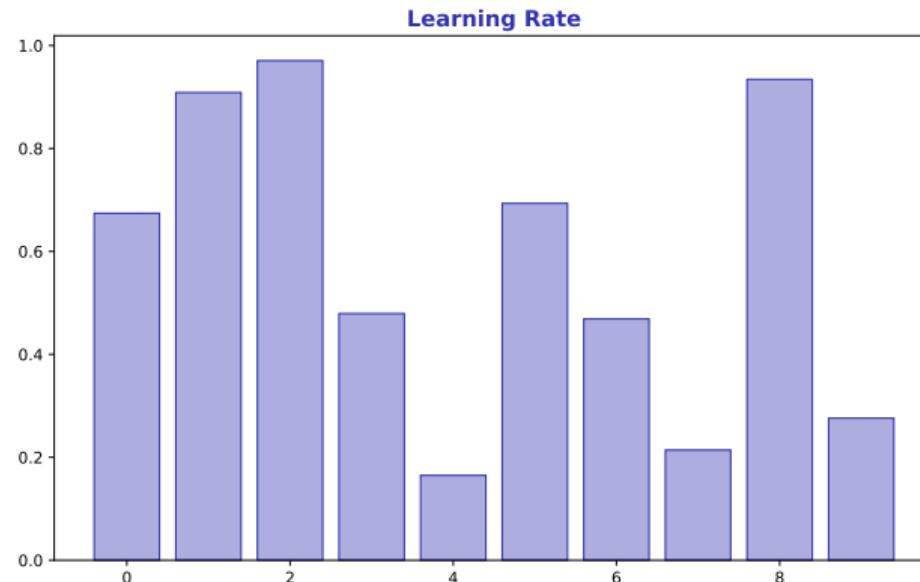


Match loss to problem: MSE for regression, cross-entropy for classification

# Learning Rate

## The Most Important Hyperparameter

- Too high: oscillates, may diverge
- Too low: converges very slowly

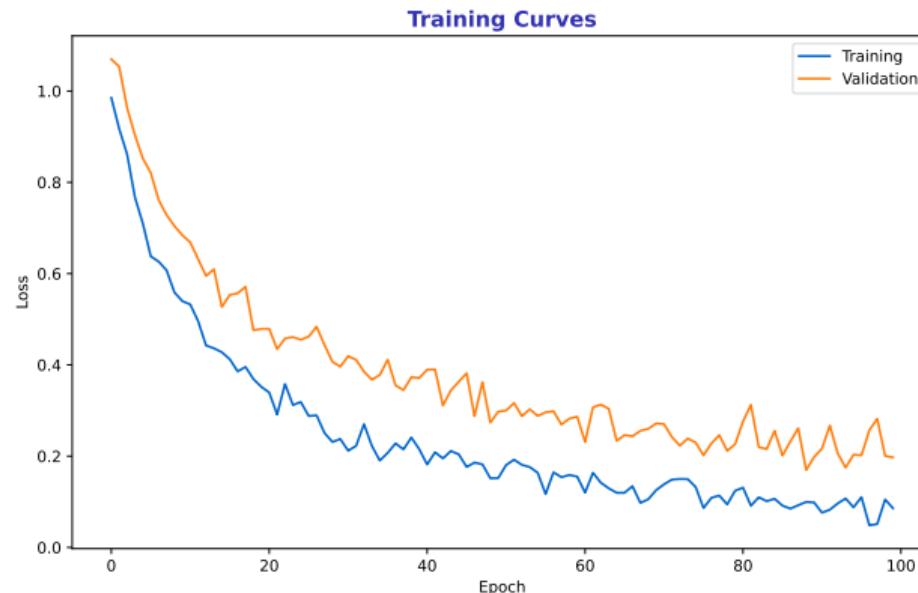


Start with 0.001 (Adam default). Reduce if loss is unstable.

# Training Curves

## Diagnosing Training

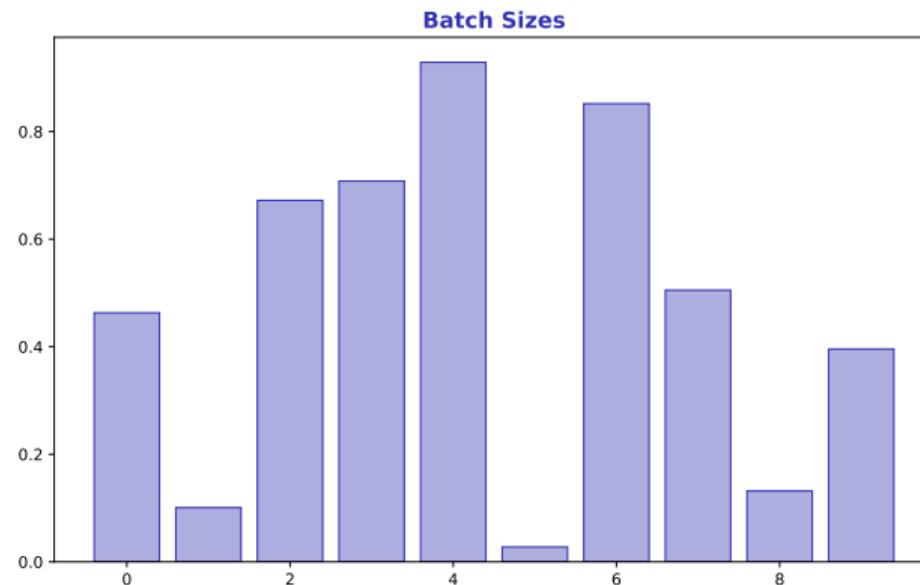
- Plot loss vs epochs for train and validation
- Train  $\downarrow$ , val  $\downarrow$ : good. Train  $\downarrow$ , val  $\uparrow$ : overfitting



Always monitor validation loss – it shows real generalization

## How Much Data Per Update?

- Batch: all data (slow, stable)
- Mini-batch: 32-256 samples (fast, noisy but works)
- Stochastic: 1 sample (very noisy)

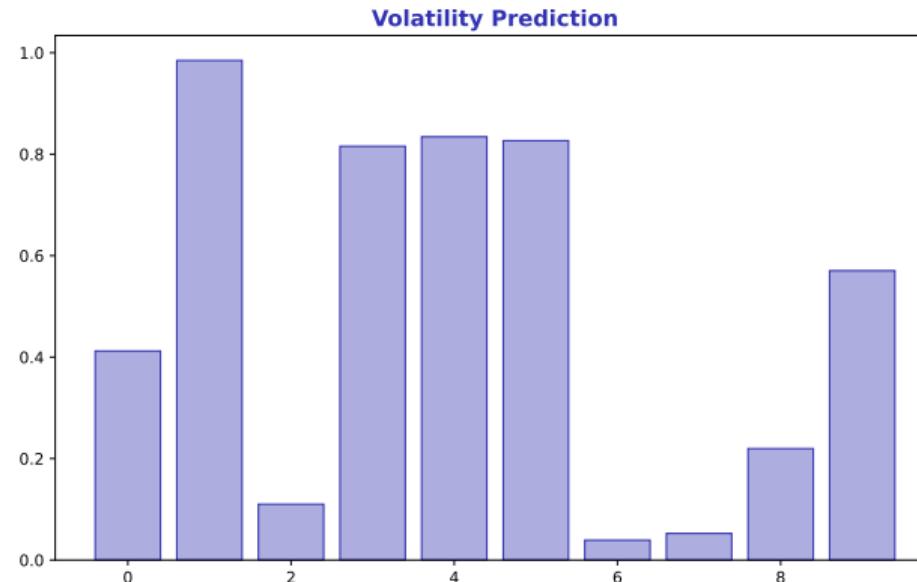


Default: mini-batch of 32. Larger batches need larger learning rates.

# Volatility Prediction

## Finance Application

- Input: lagged returns, volume, past volatility
- Output: next-day volatility (regression)



Monitor training curves carefully – financial data is noisy

## Hands-On Exercise (25 min)

### Task: Visualize Gradient Descent

- ① Create simple 2D function (e.g., parabola) and plot surface
- ② Implement gradient descent from scratch
- ③ Plot optimization path for different learning rates
- ④ Train Keras model and plot train/val loss curves
- ⑤ Try batch sizes 16, 64, 256 – compare convergence

**Deliverable:** 3D surface with optimization paths + loss curve comparison.

**Extension:** Implement momentum and compare to vanilla gradient descent

## Lesson Summary

**Problem Solved:** We understand how neural networks learn through backpropagation and gradient descent.

**Key Takeaways:**

- Forward pass computes prediction, backward pass computes gradients
- Gradient descent updates weights to minimize loss
- Learning rate controls step size – crucial hyperparameter
- Monitor train/val loss curves to diagnose training

**Next Lesson:** Overfitting Prevention (L36) – regularization for neural networks

**Memory:** Backprop = chain rule. Learning rate = step size. Watch val loss.

## Lesson 36: Overfitting Prevention

Data Science with Python – BSc Course

45 Minutes

## Learning Objectives

**The Problem:** Neural networks have millions of parameters and can memorize training data. How do we ensure they generalize to new data?

**After this lesson, you will be able to:**

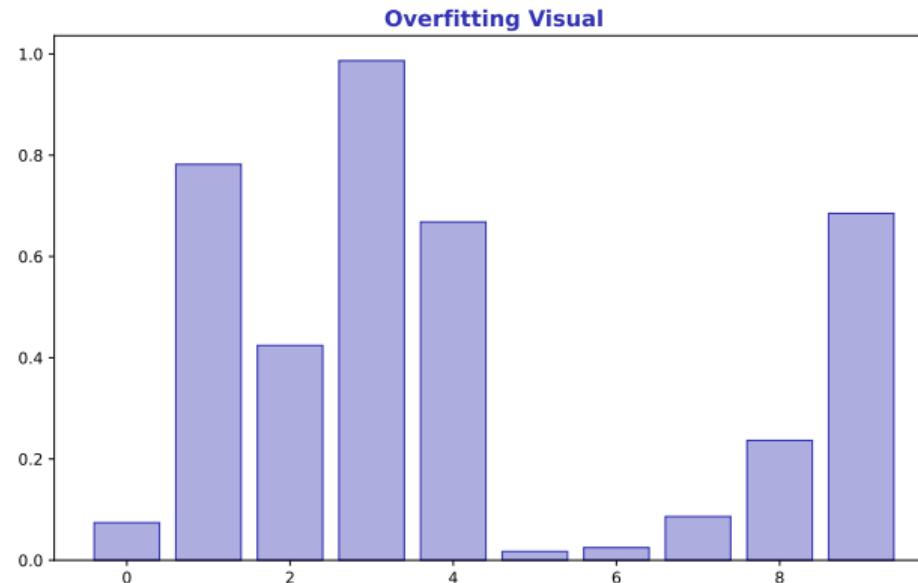
- Apply dropout regularization
- Use early stopping to prevent overfitting
- Diagnose overfitting from learning curves
- Build robust neural network models

**Finance Application:** Preventing models from fitting to noise in market data

# Recognizing Overfitting

## The Gap Between Train and Test

- Training loss keeps decreasing, validation loss increases
- Model memorizes training data instead of learning patterns

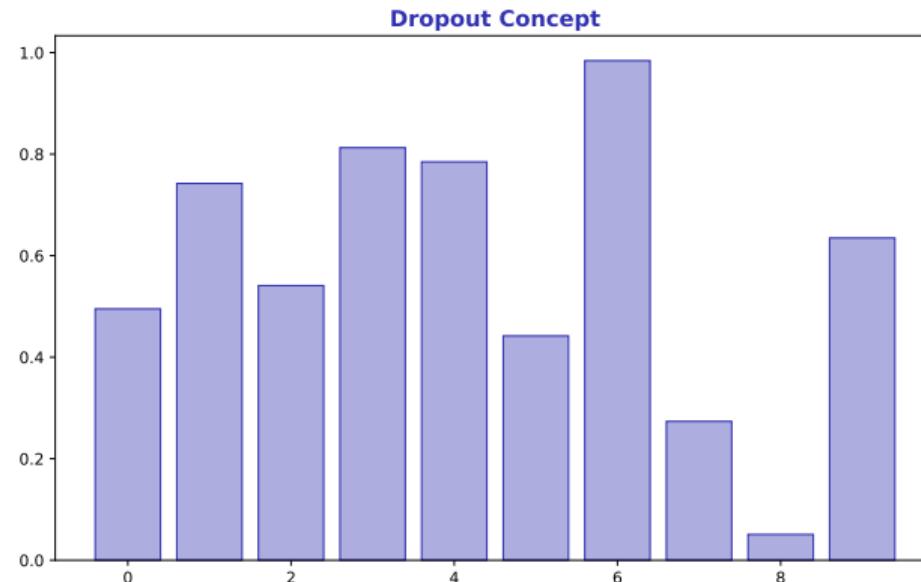


**Sign: train accuracy 99%, test accuracy 60% = severe overfitting**

# Dropout Concept

## Randomly Ignoring Neurons

- During training: randomly set fraction of neurons to zero
- Forces network to not rely on any single neuron

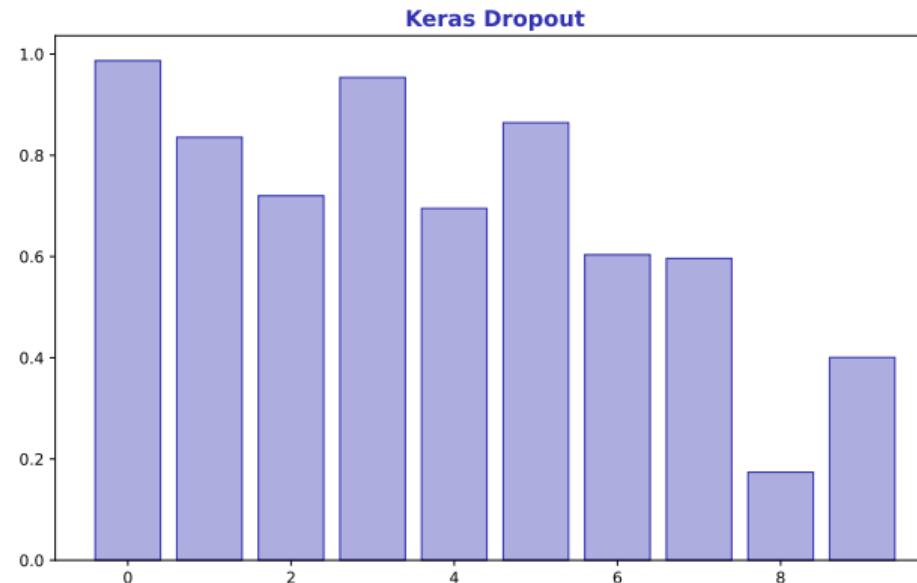


Dropout = training many thinned networks, averaging at test time

# Keras Dropout Layer

## Implementation

- `model.add(Dropout(0.5))` – drop 50% of neurons
- Place after Dense layers, typically 0.2-0.5 rate

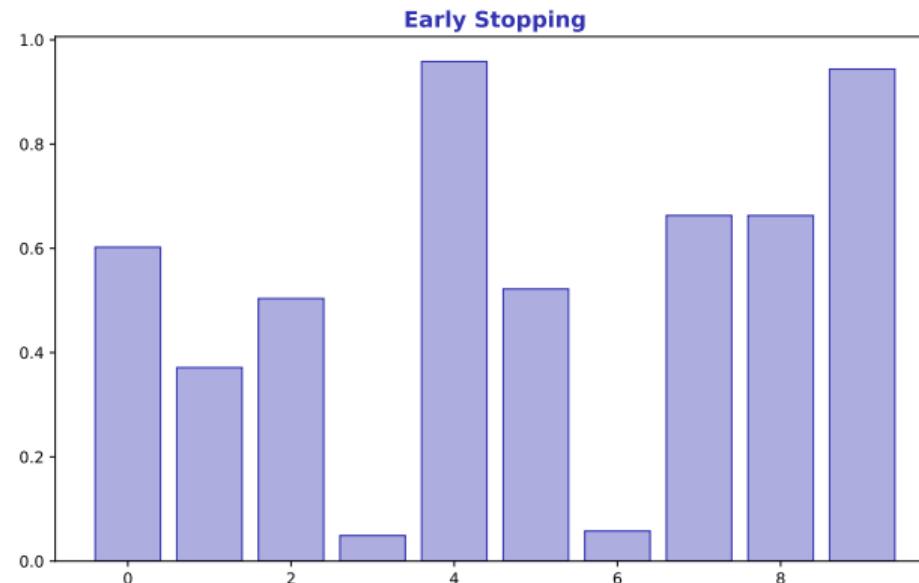


Common pattern: Dense → Dropout → Dense → Dropout

# Early Stopping

## Stop Before Overfitting

- Monitor validation loss; stop when it starts increasing
- `EarlyStopping(patience=10, restore_best_weights=True)`

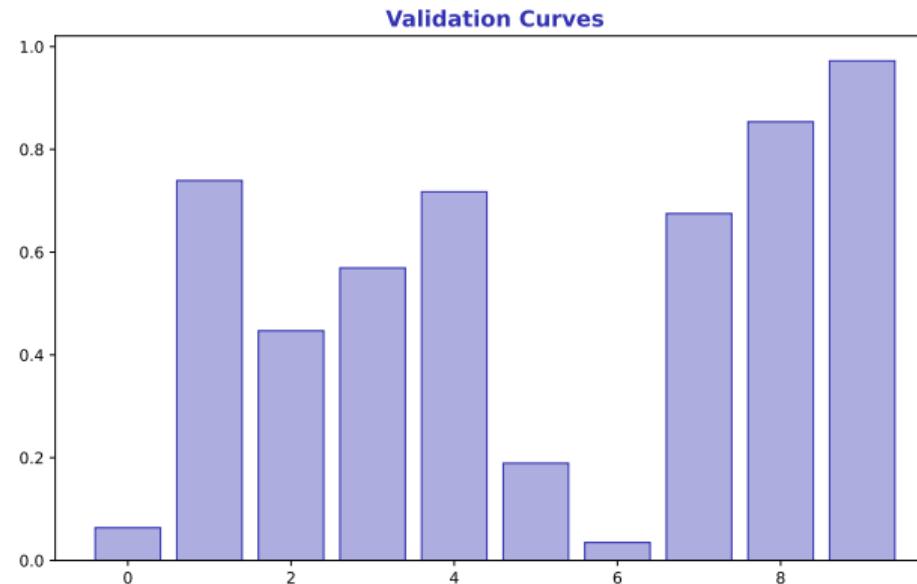


Patience = how many epochs without improvement before stopping

# Validation Curves

## Reading the Learning Curve

- Underfitting: both train and val loss high
- Overfitting: train low, val high (growing gap)
- Good fit: both low, small gap

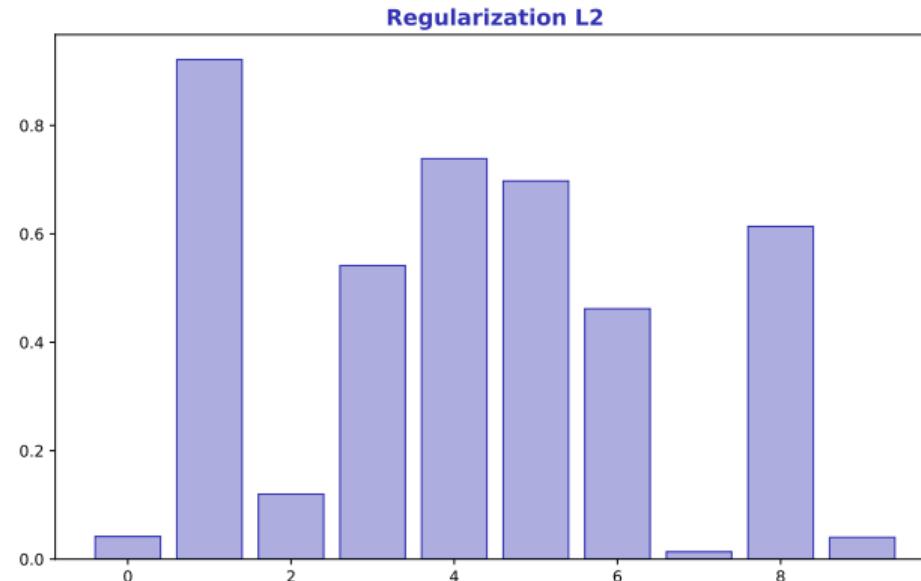


Always plot train and val loss together during development

# L2 Regularization

## Penalizing Large Weights

- Add  $\lambda \sum w^2$  to loss function
- Keras: `Dense(64, kernel_regularizer=l2(0.01))`

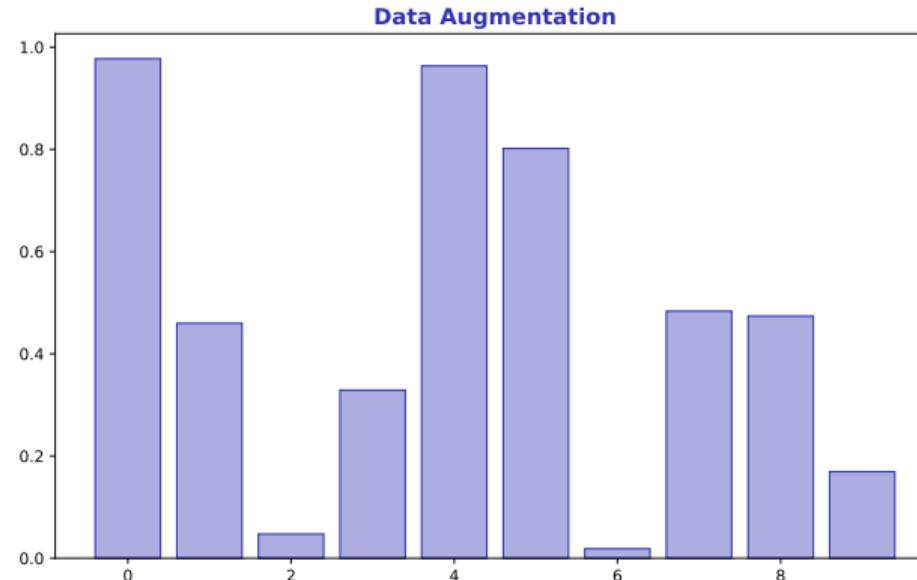


L2 regularization keeps weights small, reducing model complexity

# Data Augmentation

## Creating More Training Data

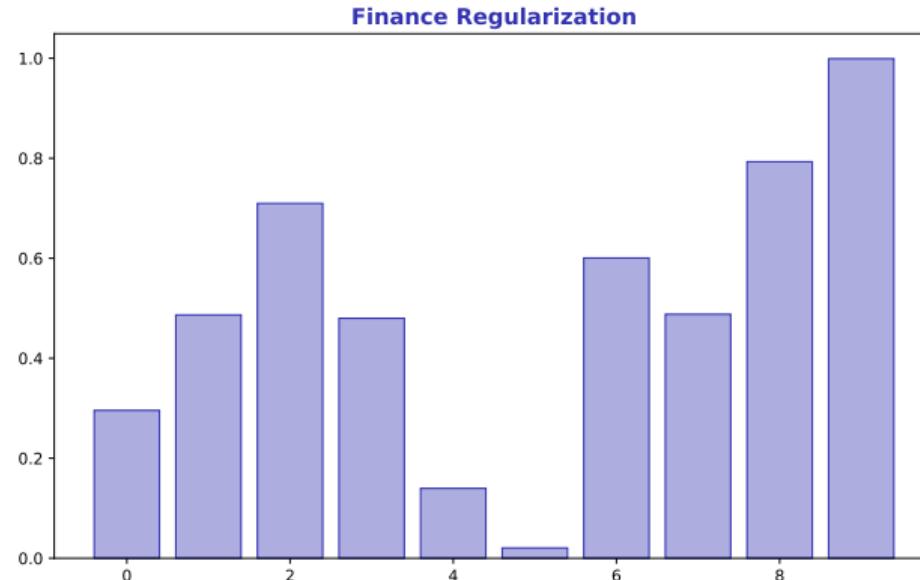
- For images: rotate, flip, crop, adjust brightness
- For time series: add noise, time warping



More diverse training data = better generalization

## Special Considerations for Financial Data

- Financial data is noisy – regularization is essential
- Use dropout + early stopping + small networks



Rule: simpler models often work better on financial data

## Hands-On Exercise (25 min)

### Task: Compare Regularization Techniques

- ① Create overfit-prone dataset (few samples, many features)
- ② Train baseline MLP – observe severe overfitting
- ③ Add Dropout(0.5) – compare train/val curves
- ④ Add EarlyStopping – when does training stop?
- ⑤ Compare final test accuracy across all variants

**Deliverable:** Side-by-side learning curves + accuracy comparison table.

**Extension:** Try combining dropout + L2 + early stopping

## Lesson Summary

**Problem Solved:** We can now prevent neural networks from memorizing data and ensure generalization.

**Key Takeaways:**

- Dropout: randomly zero neurons during training
- Early stopping: halt when validation loss stops improving
- L2 regularization: penalize large weights
- Always monitor train vs validation loss curves

**Next Lesson:** Text Preprocessing (L37) – preparing text for NLP

**Memory:** Dropout = random zeros. Early stopping = stop at best val. Watch the gap.

## Lesson 37: Text Preprocessing

Data Science with Python – BSc Course

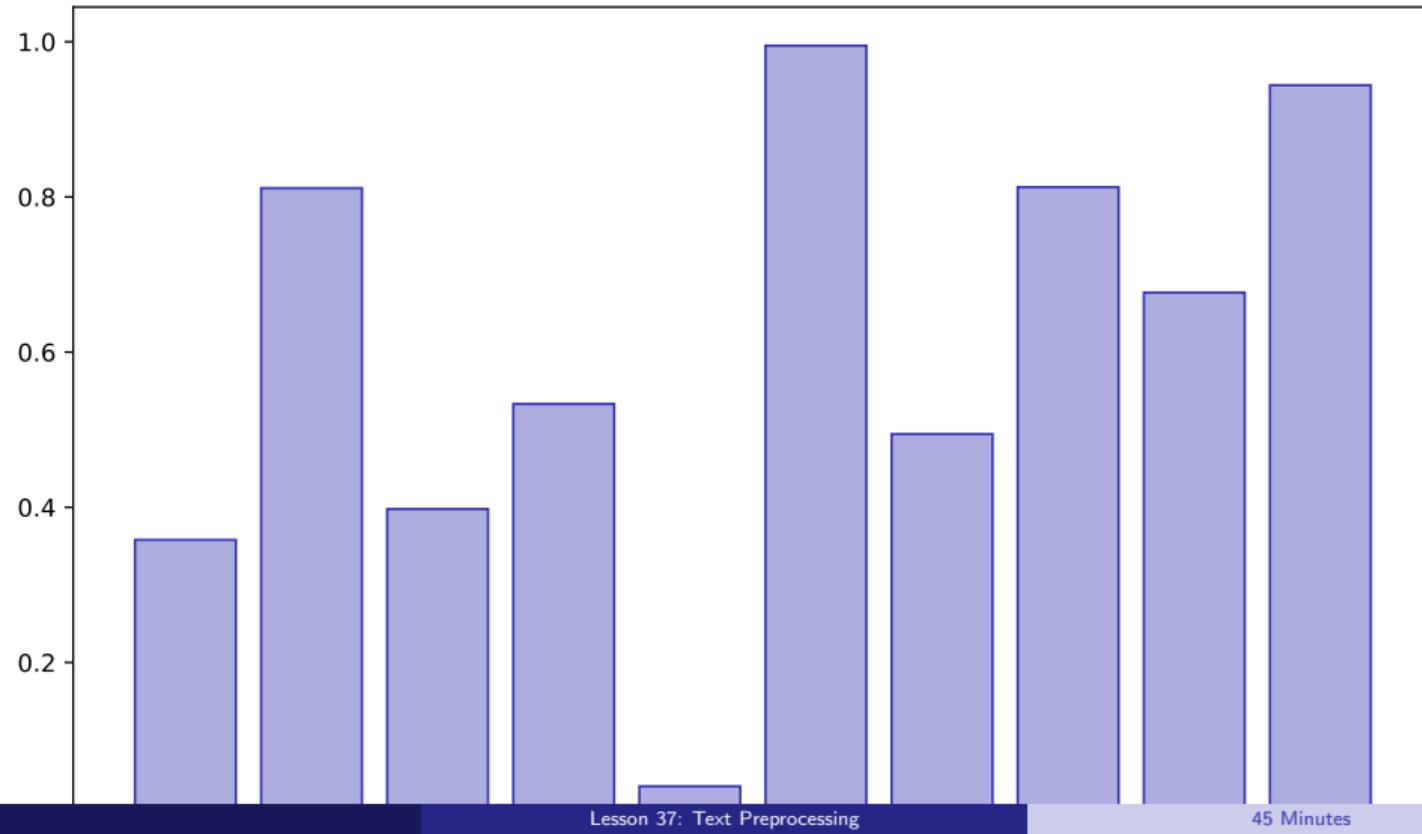
45 Minutes

**After this lesson, you will be able to:**

- Tokenize text
- Remove stopwords
- Apply stemming/lemmatization
- Clean financial text

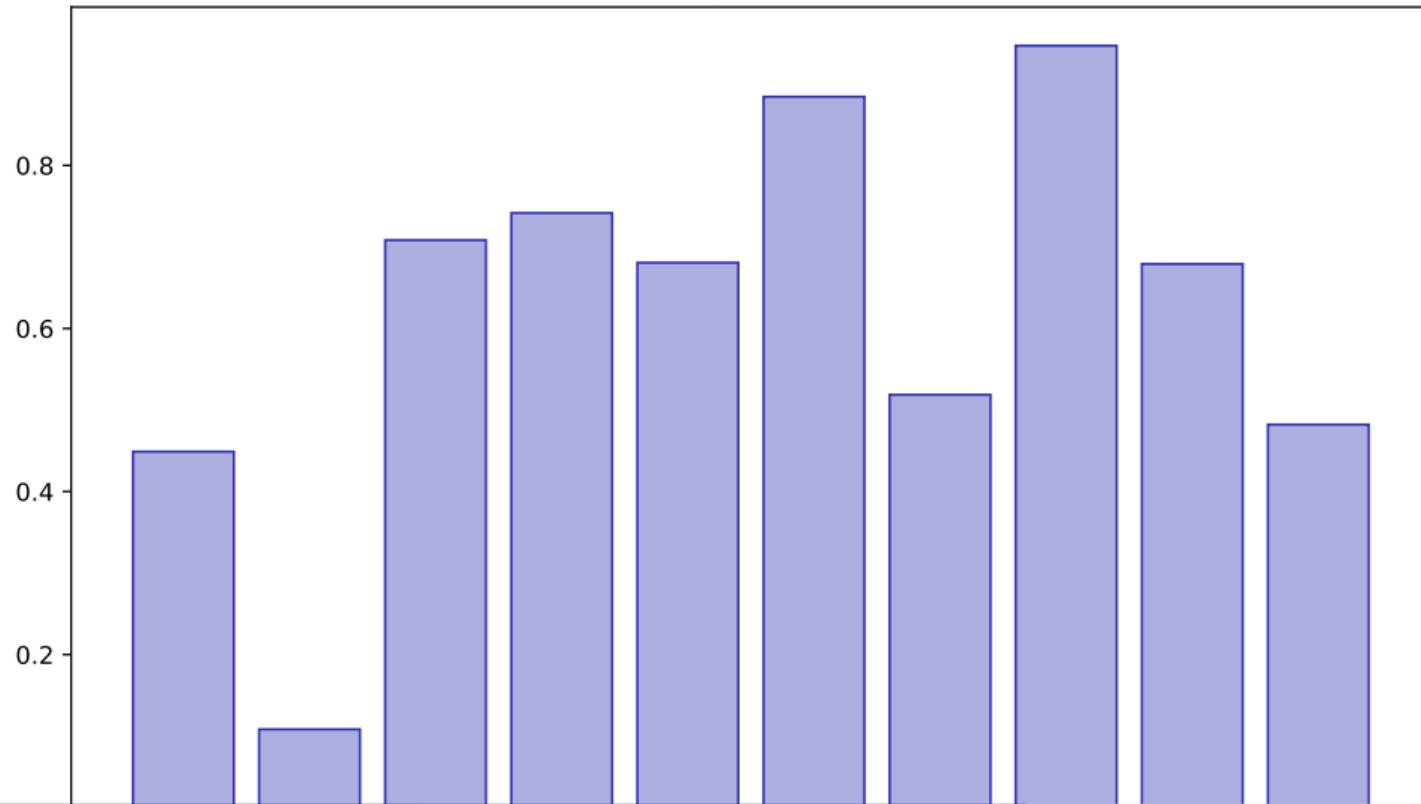
**Building towards your final project**

## Tokenization

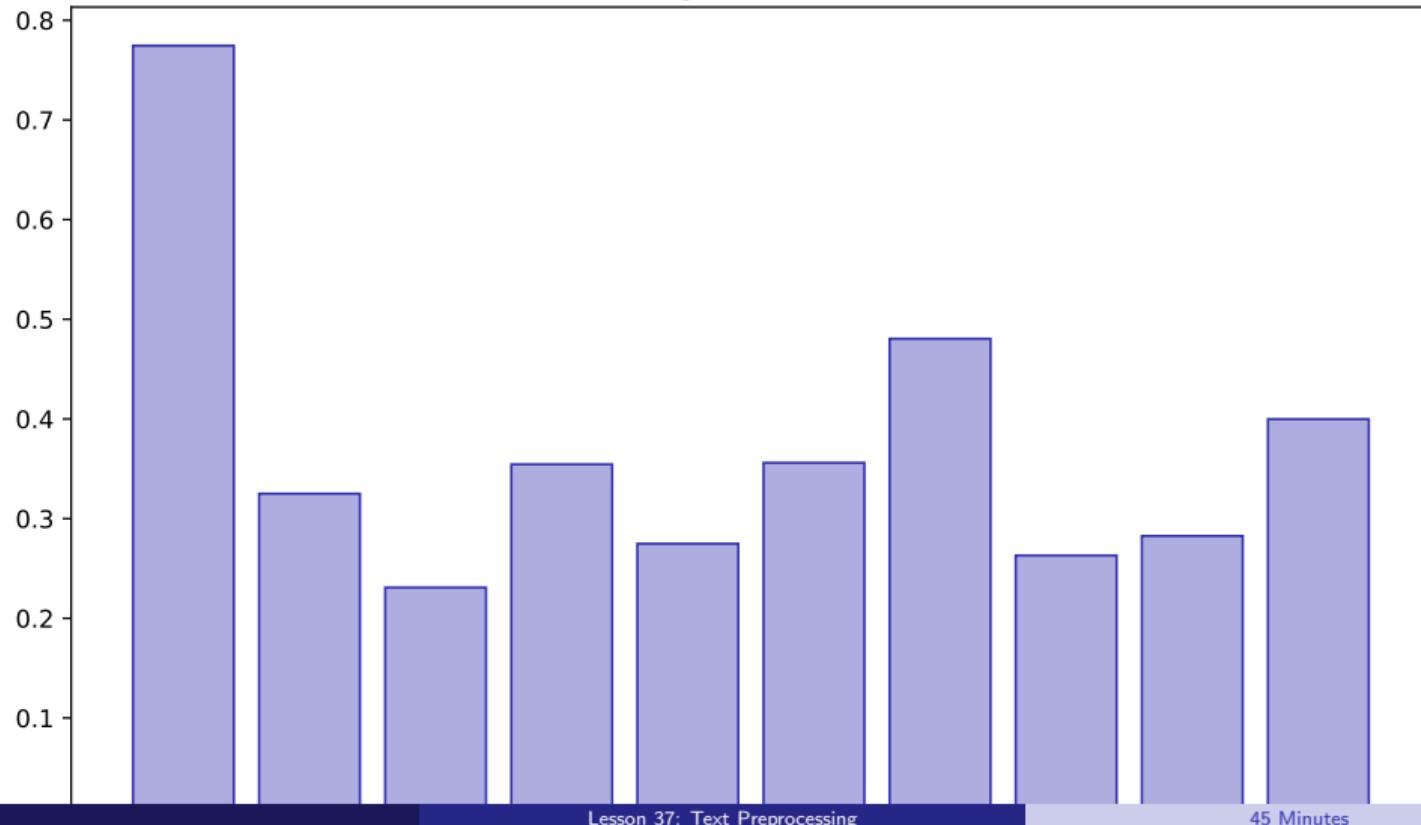


## Stopwords

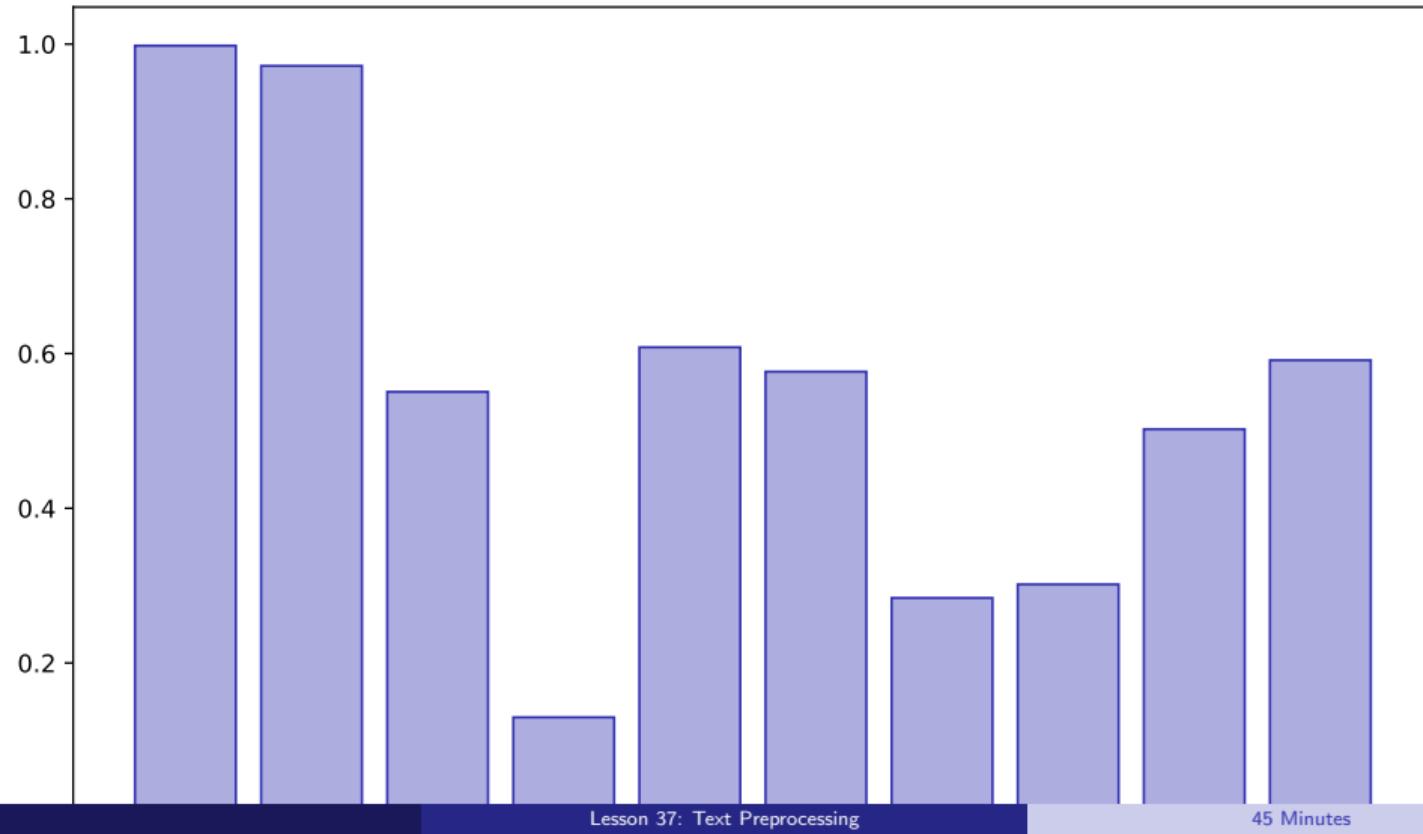
### Stopwords



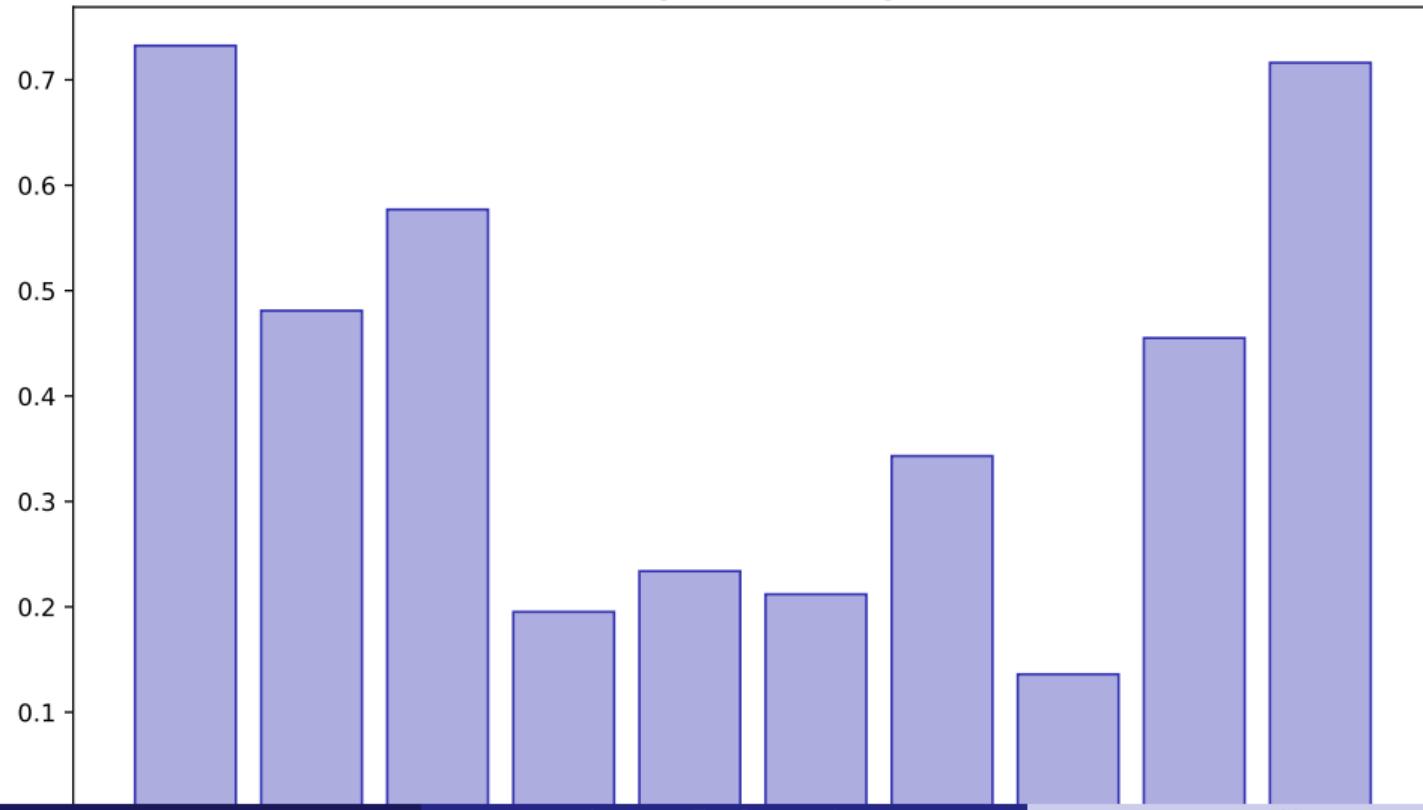
### Stemming Lemmatization



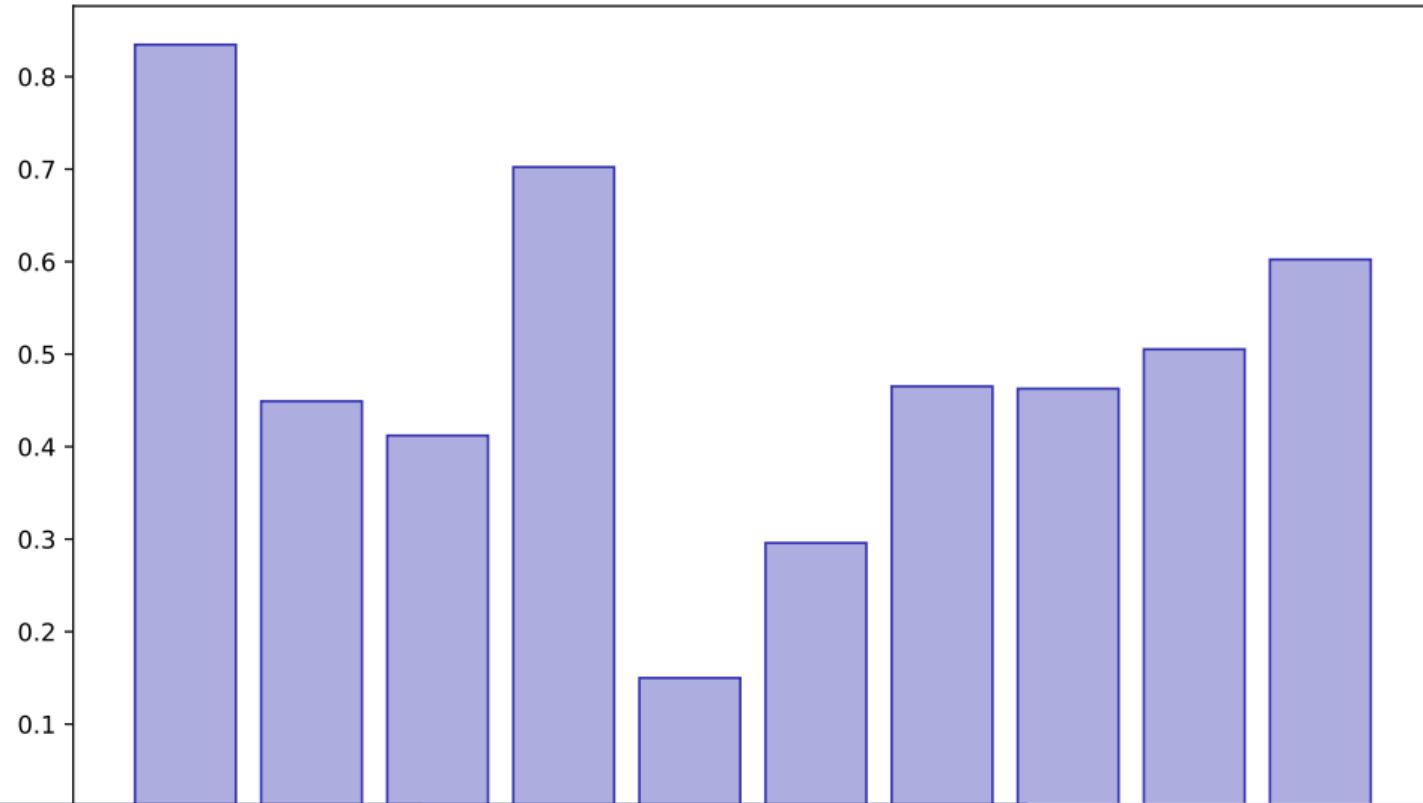
## Nltk Basics



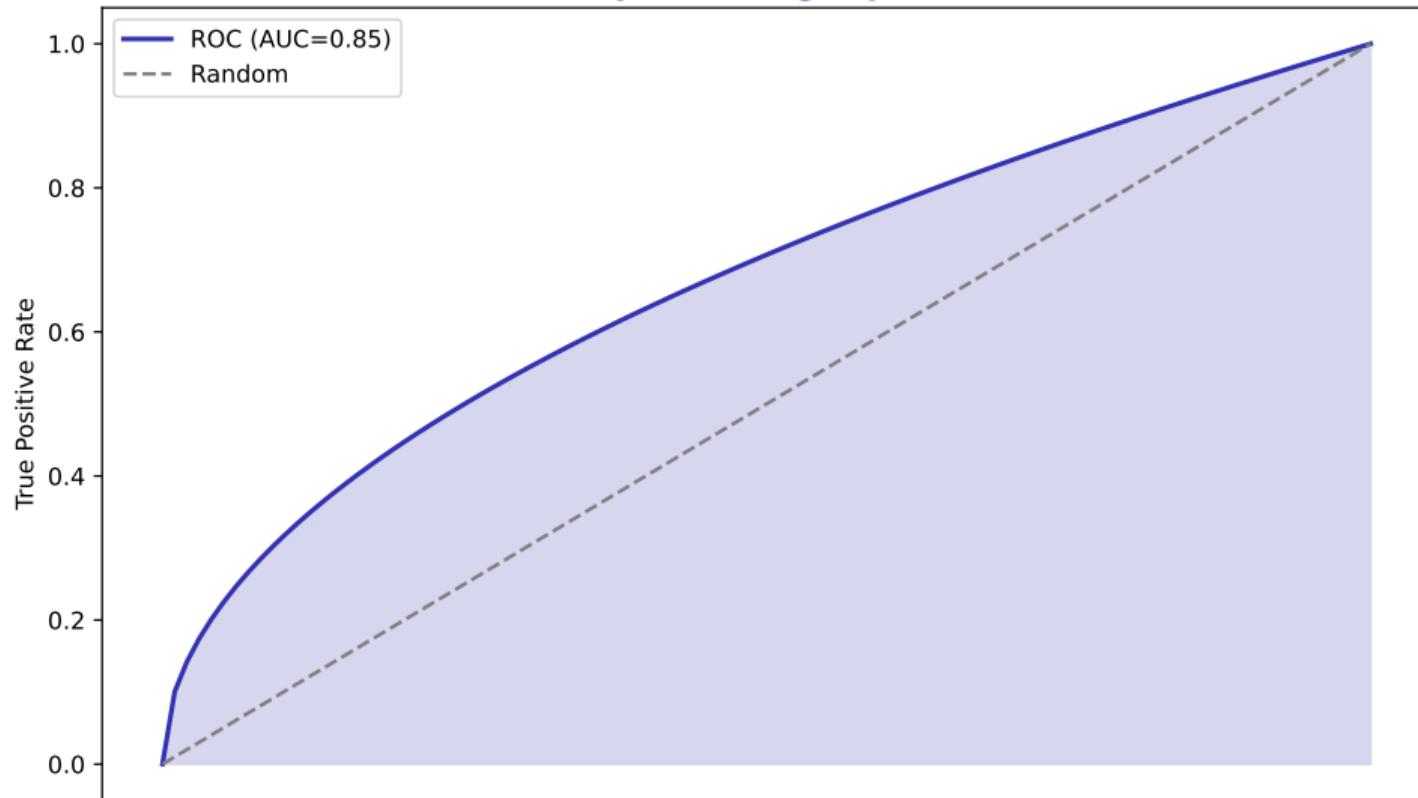
## Regex Cleaning



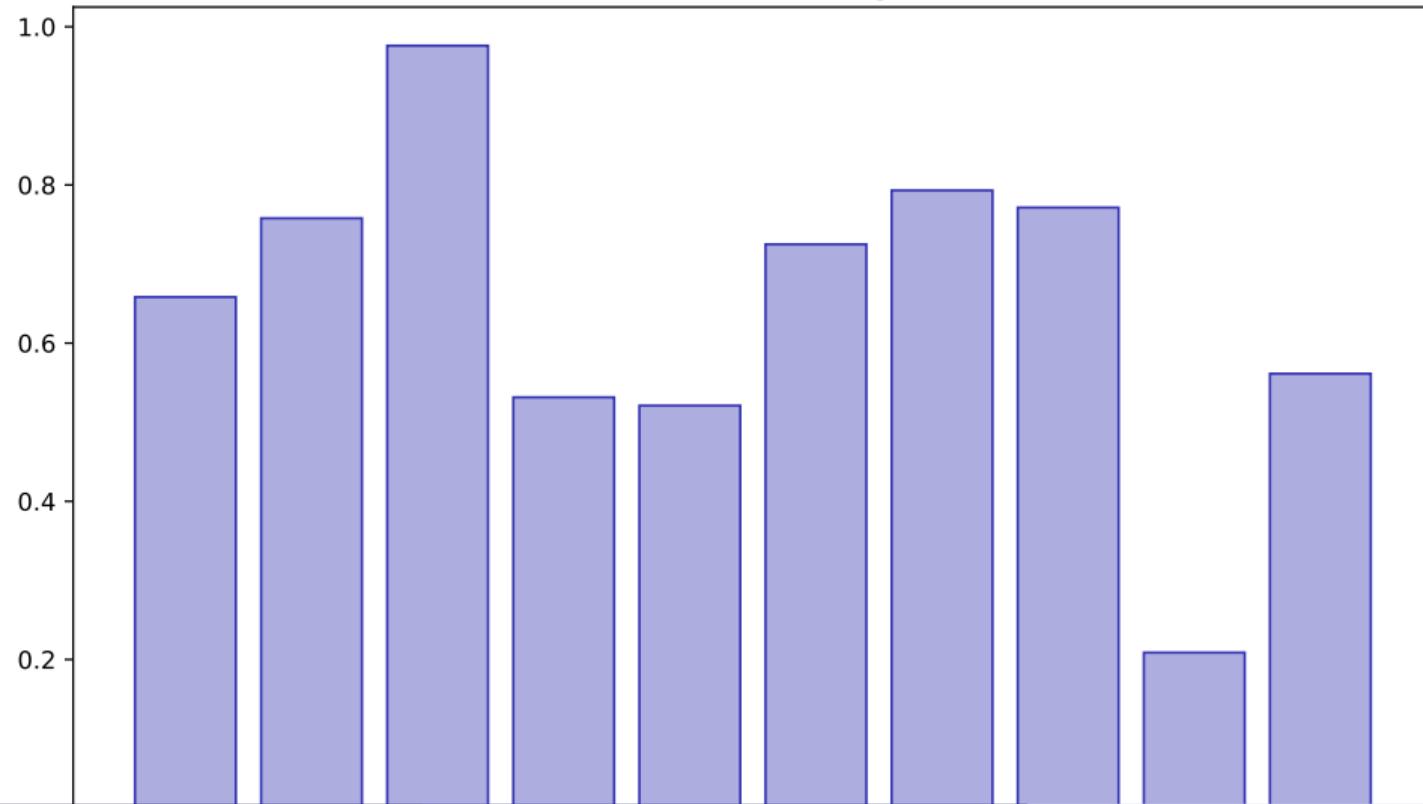
## Financial Text



### Preprocessing Pipeline



## News Cleaning



## Key Takeaways:

- Tokenize text
- Remove stopwords
- Apply stemming/lemmatization
- Clean financial text

Apply these skills in your final project

## Lesson 38: BoW and TF-IDF

Data Science with Python – BSc Course

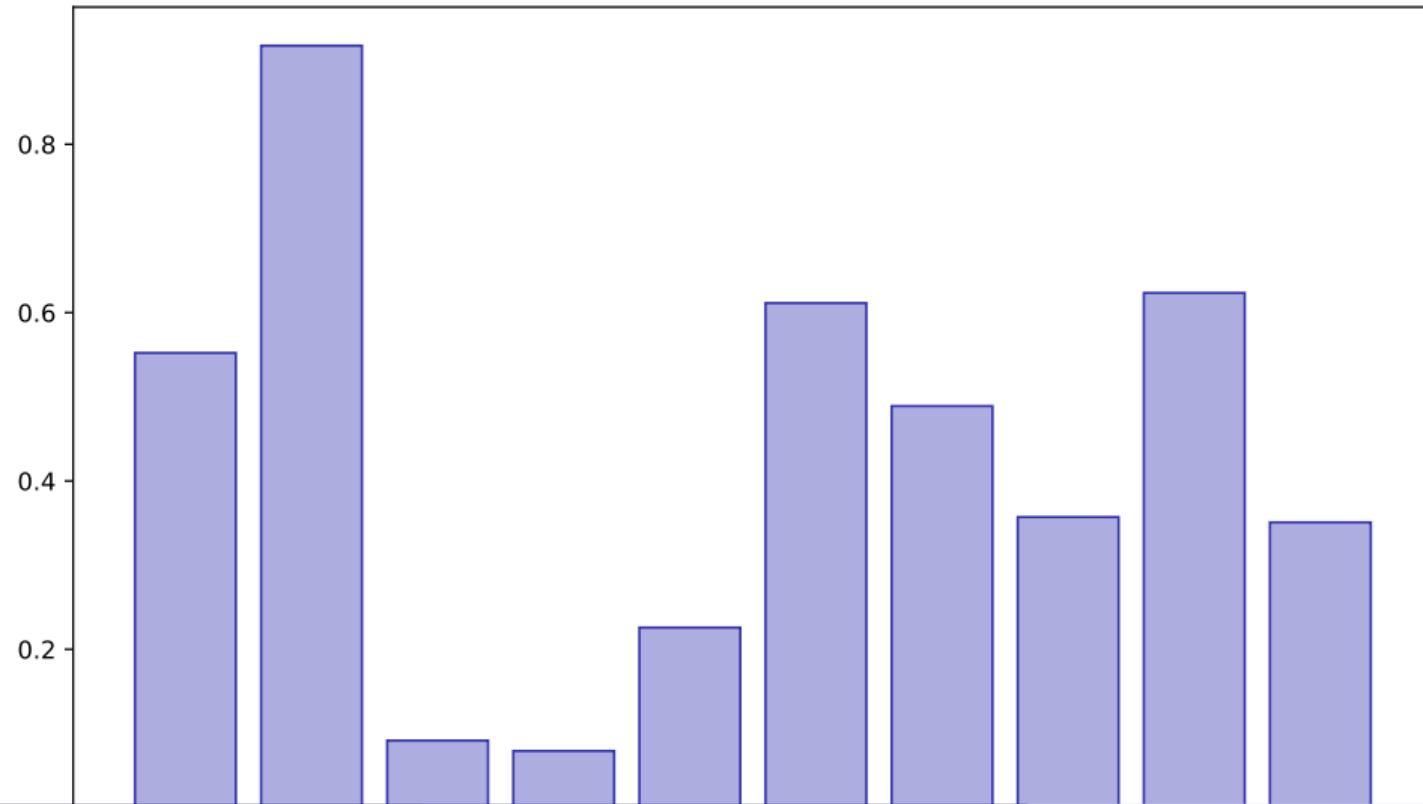
45 Minutes

**After this lesson, you will be able to:**

- Create bag of words
- Calculate TF-IDF weights
- Build document vectors
- Apply to text classification

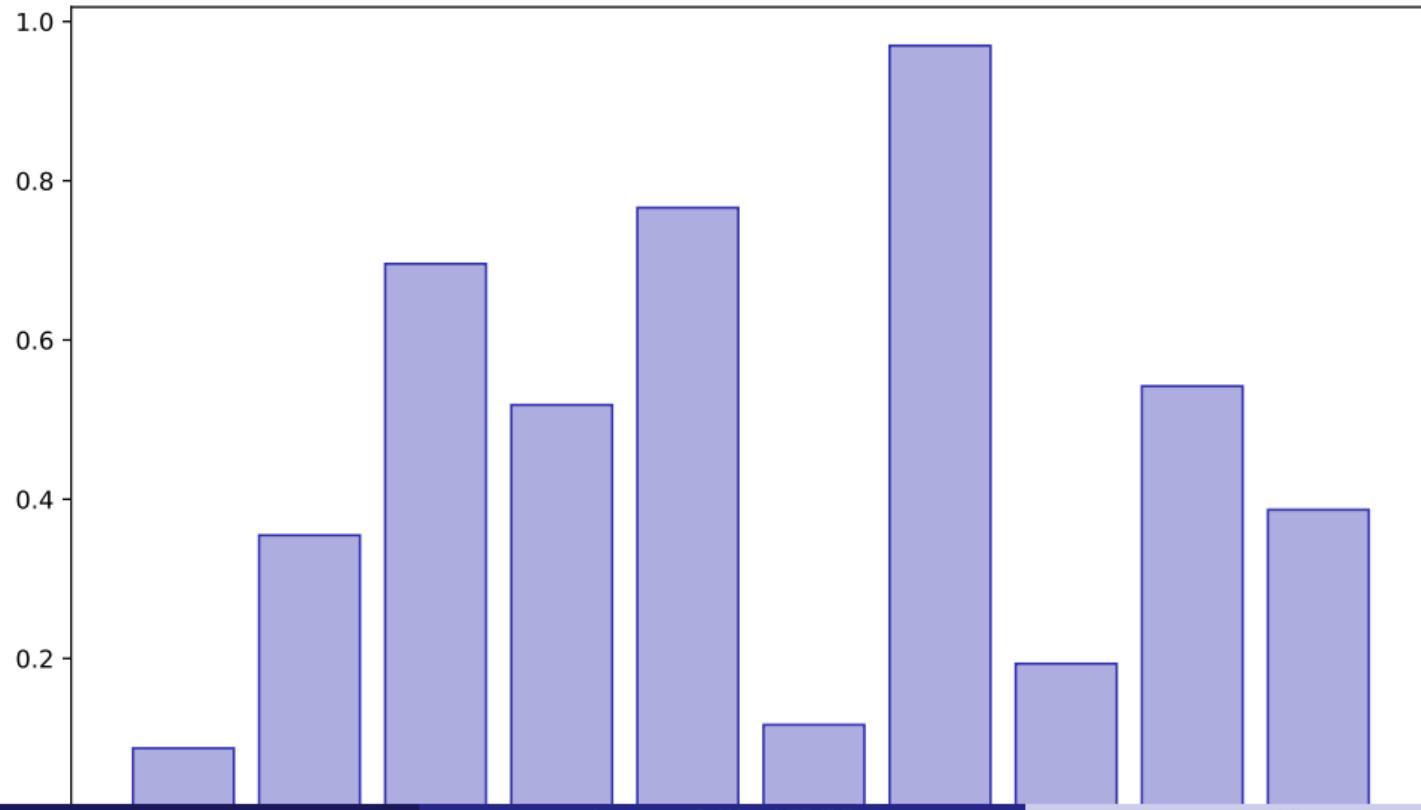
**Building towards your final project**

### Bow Concept

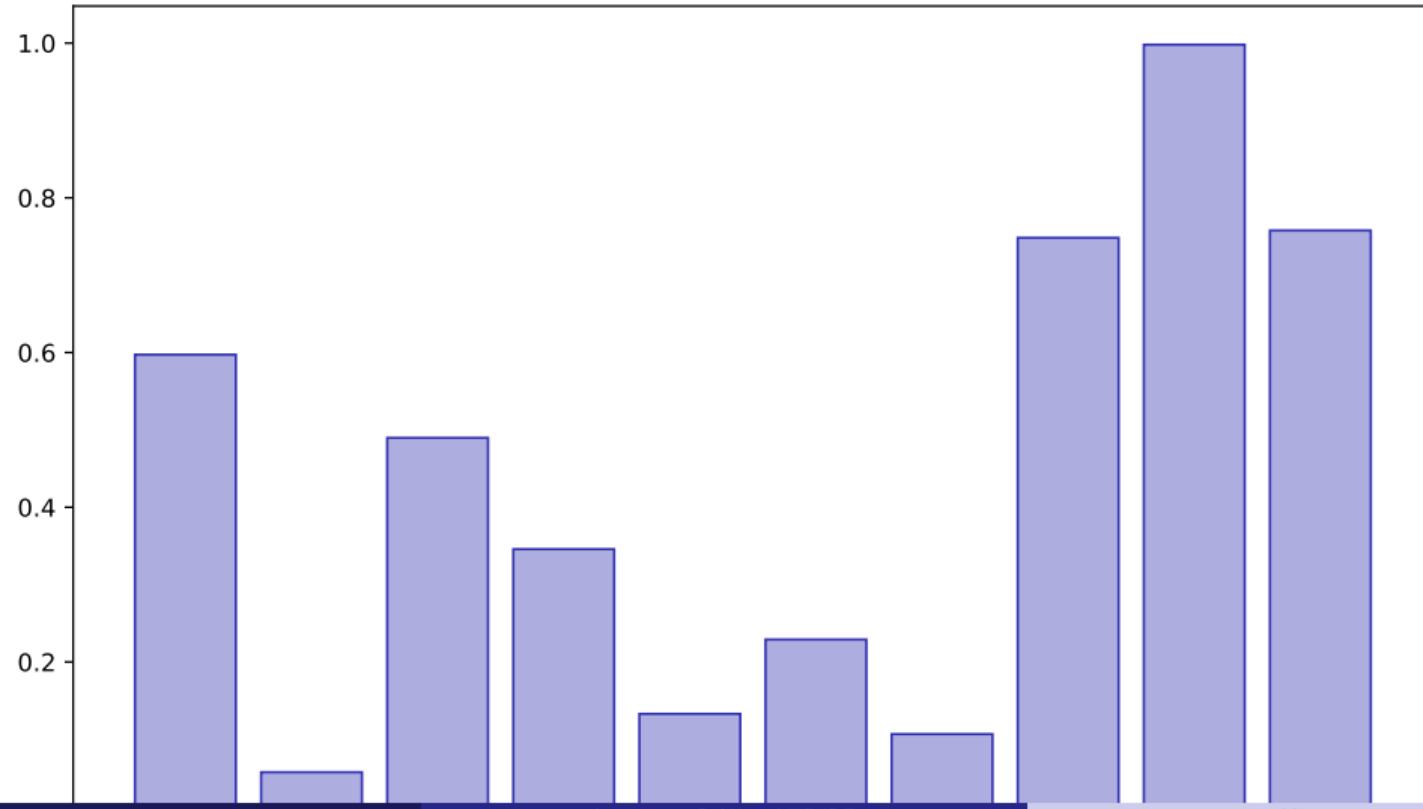


## Count Vectorizer

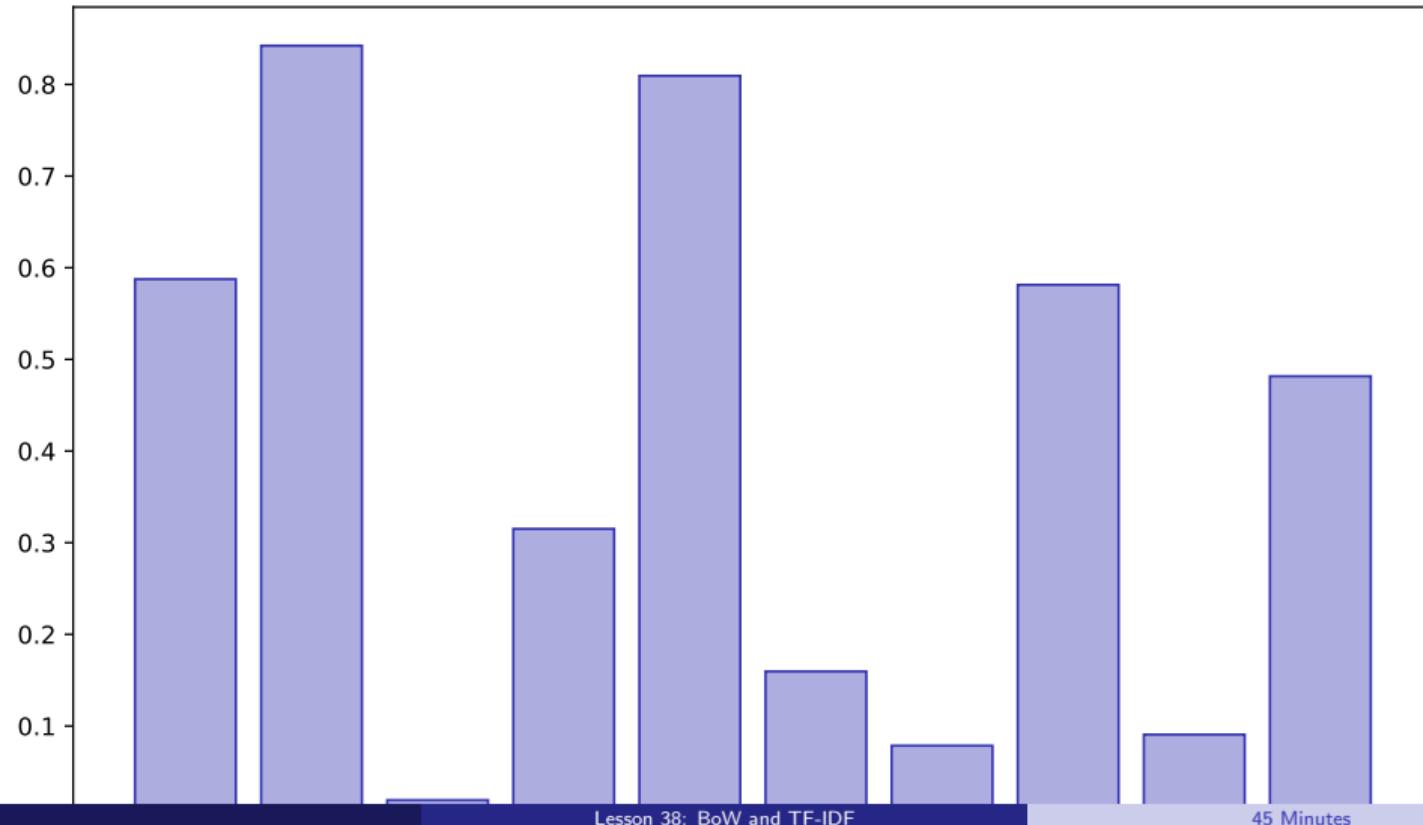
Count Vectorizer



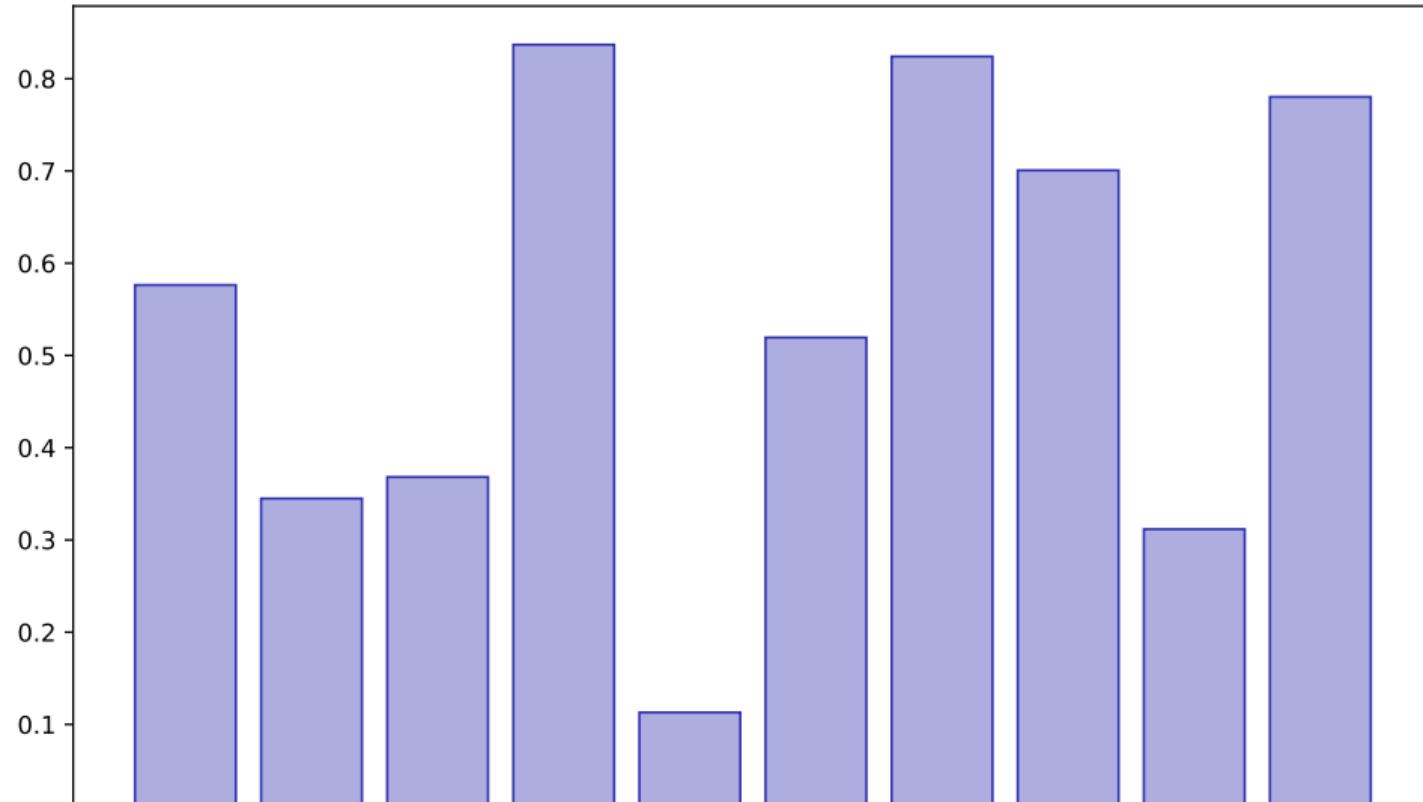
## Tfidf Formula



## Tfidf Vectorizer

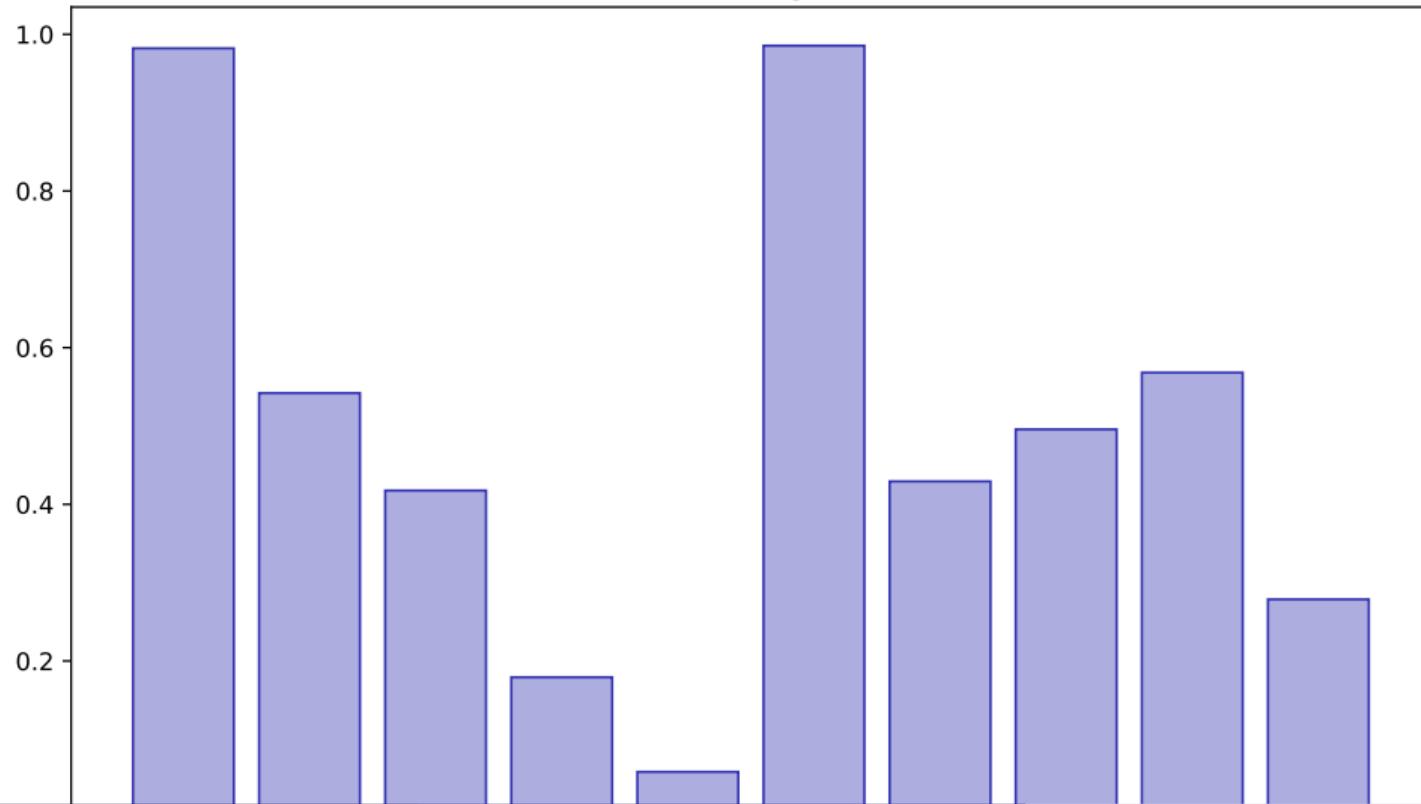


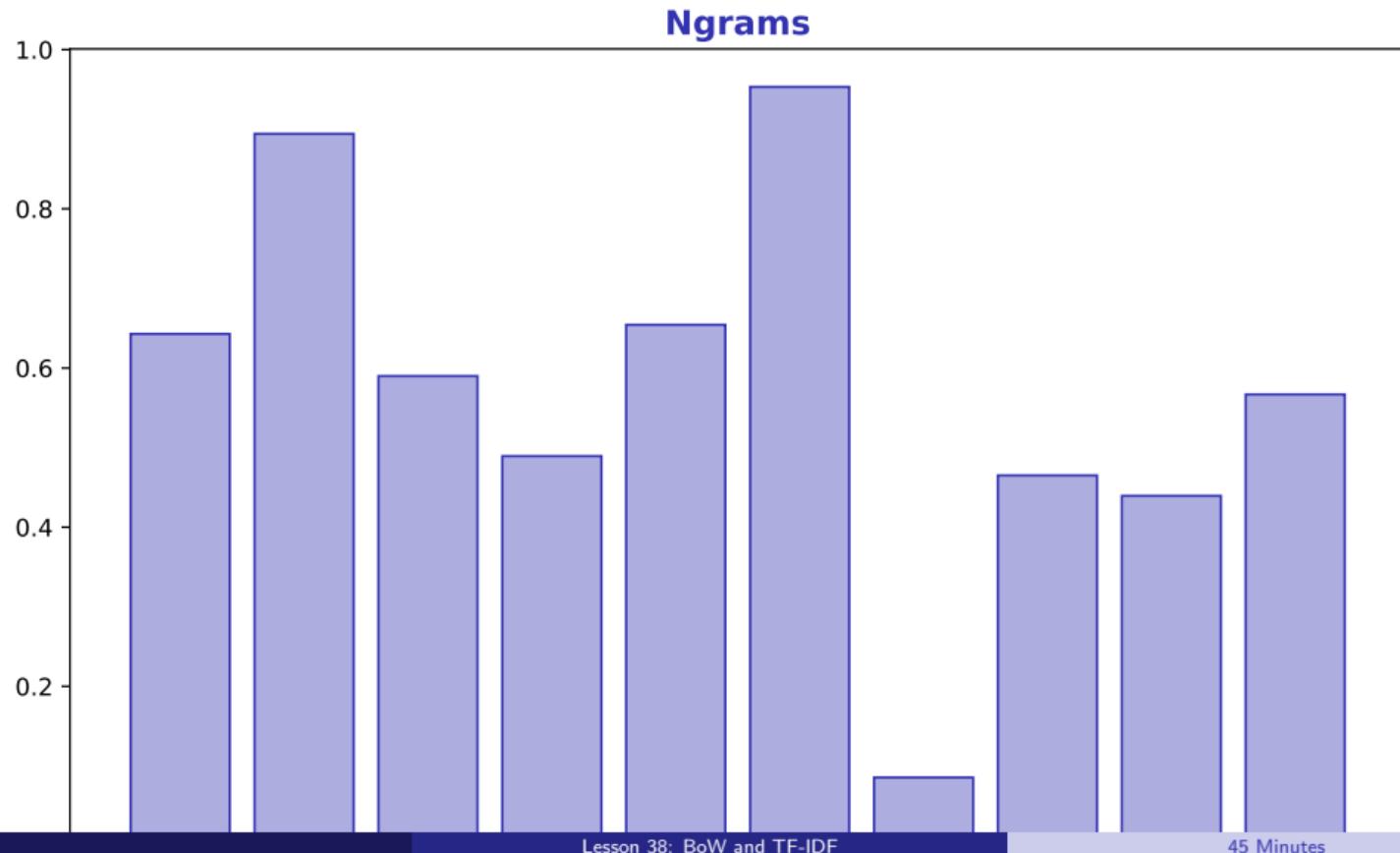
## Sparse Matrices



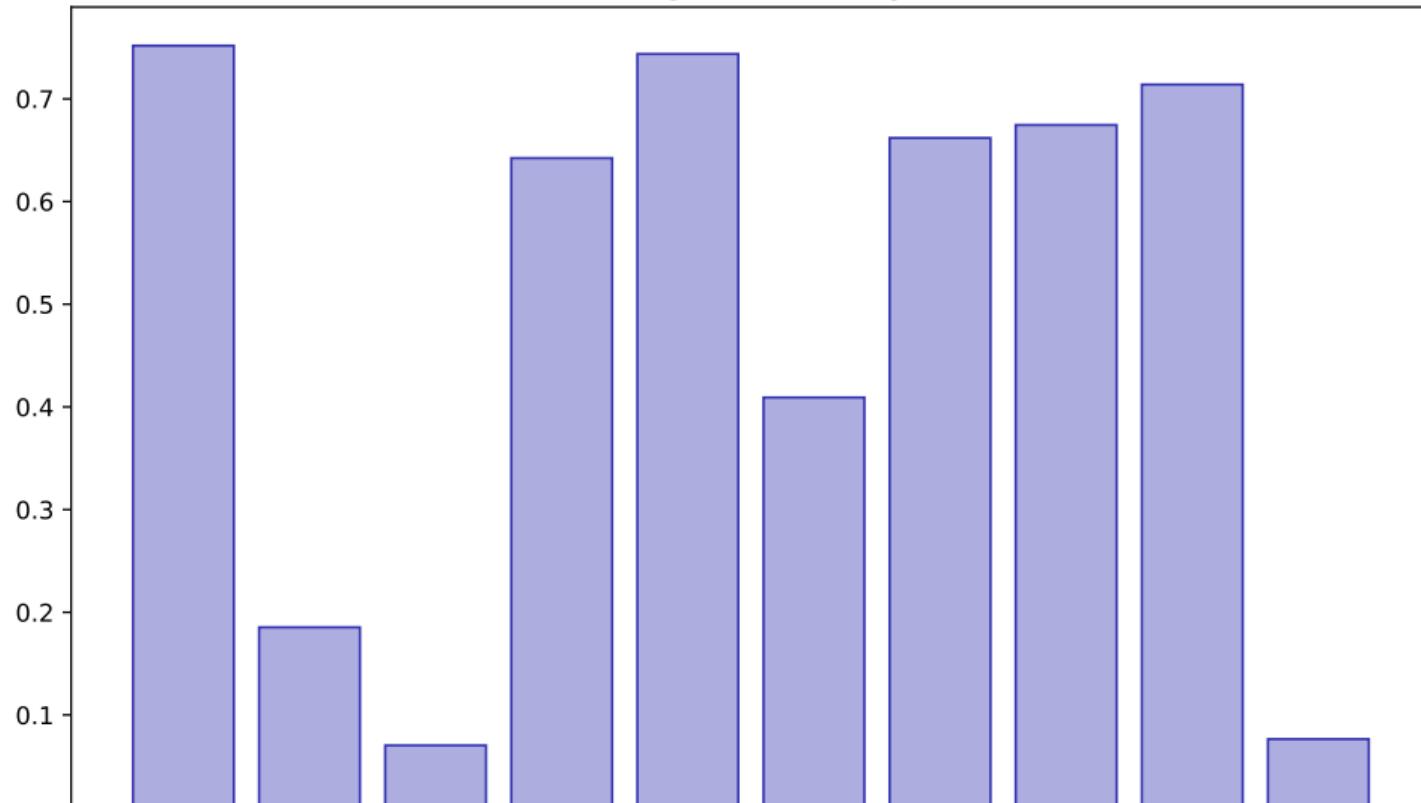
## Vocabulary Size

**Vocabulary Size**





## Earnings Call Analysis



## Lesson Summary

### Key Takeaways:

- Create bag of words
- Calculate TF-IDF weights
- Build document vectors
- Apply to text classification

Apply these skills in your final project

## Lesson 39: Word Embeddings

Data Science with Python – BSc Course

45 Minutes

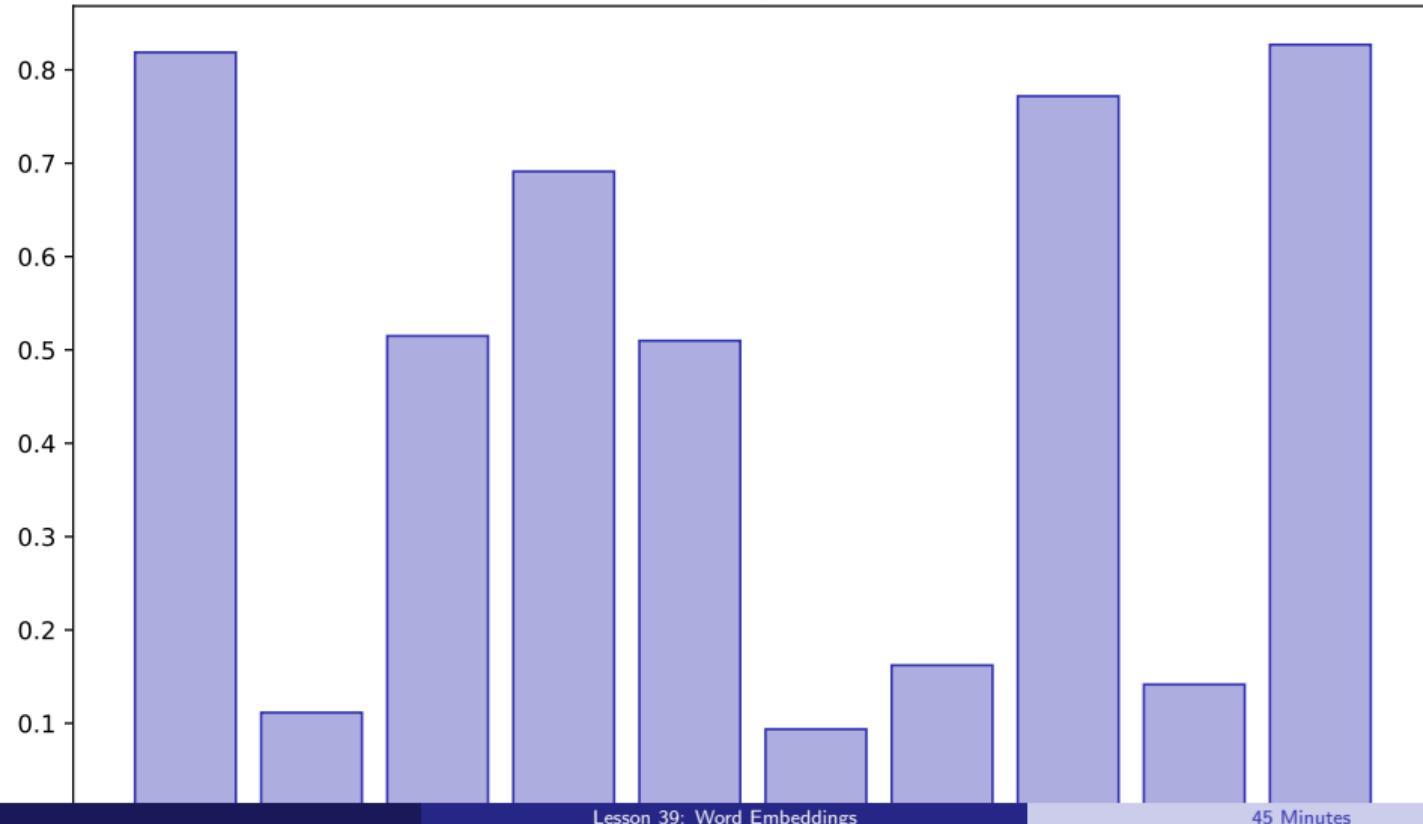
# Learning Objectives

**After this lesson, you will be able to:**

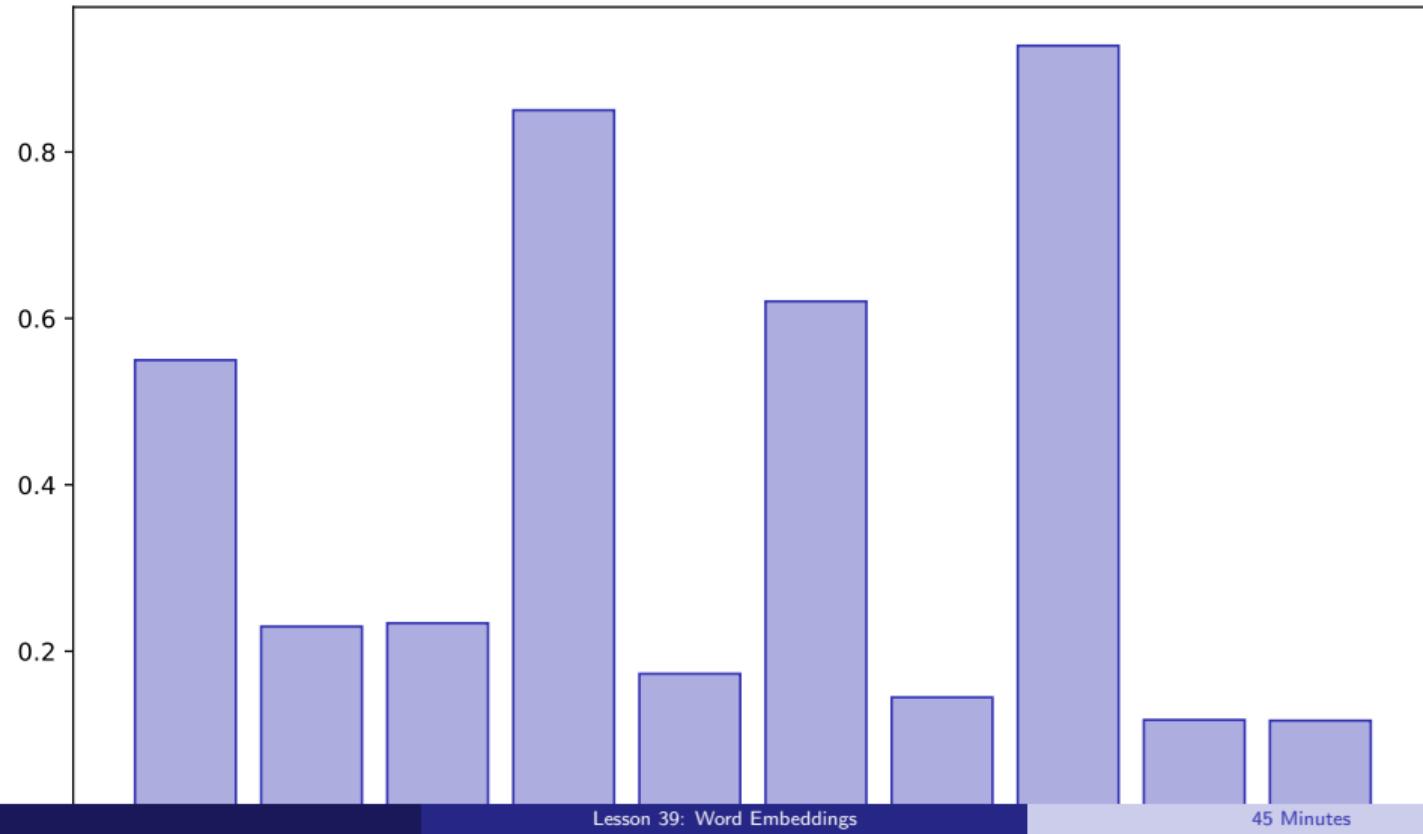
- Understand word vectors
- Use pre-trained Word2Vec
- Find similar words
- Create document embeddings

**Building towards your final project**

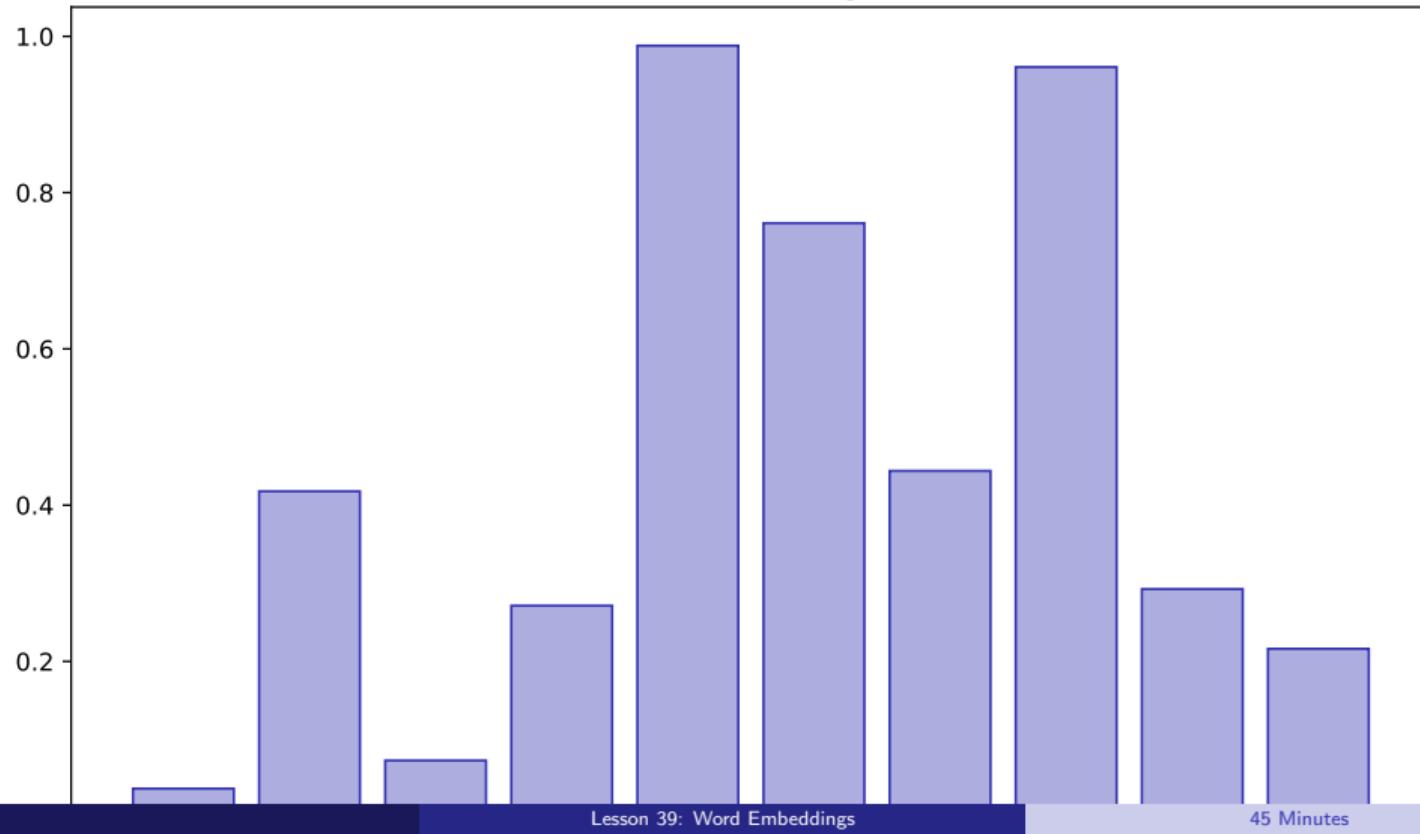
## Embedding Concept



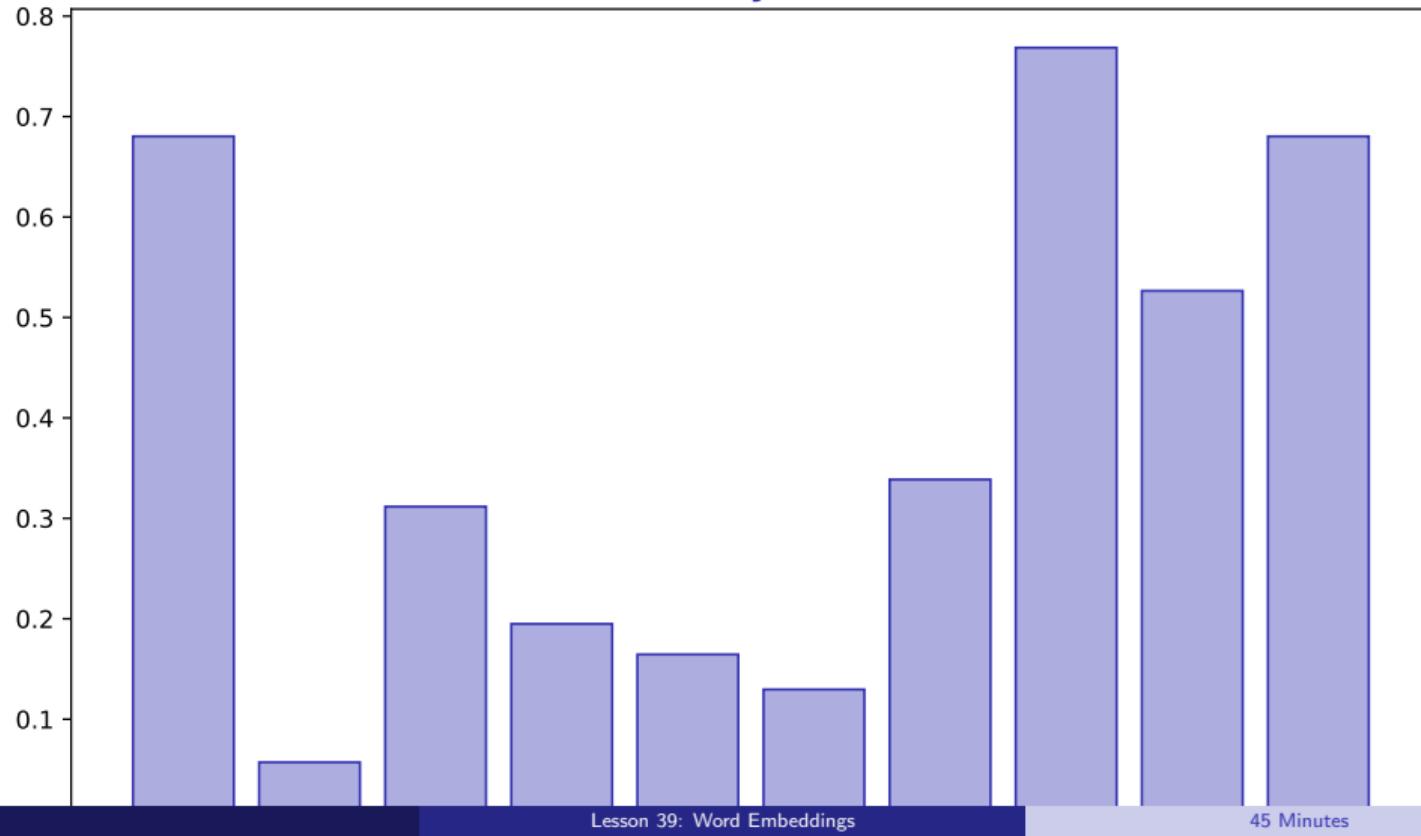
## Word2Vec



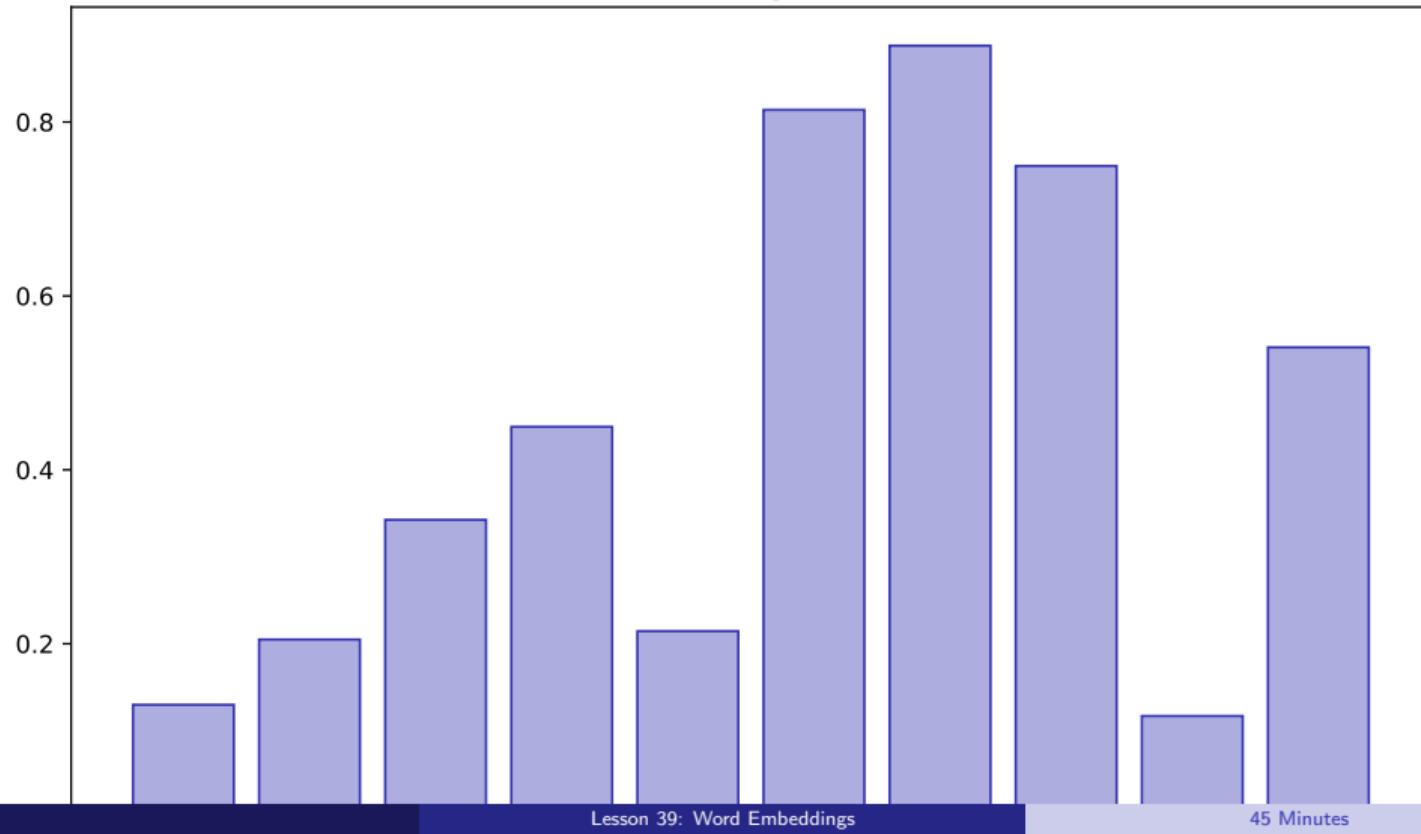
## Gensim Usage



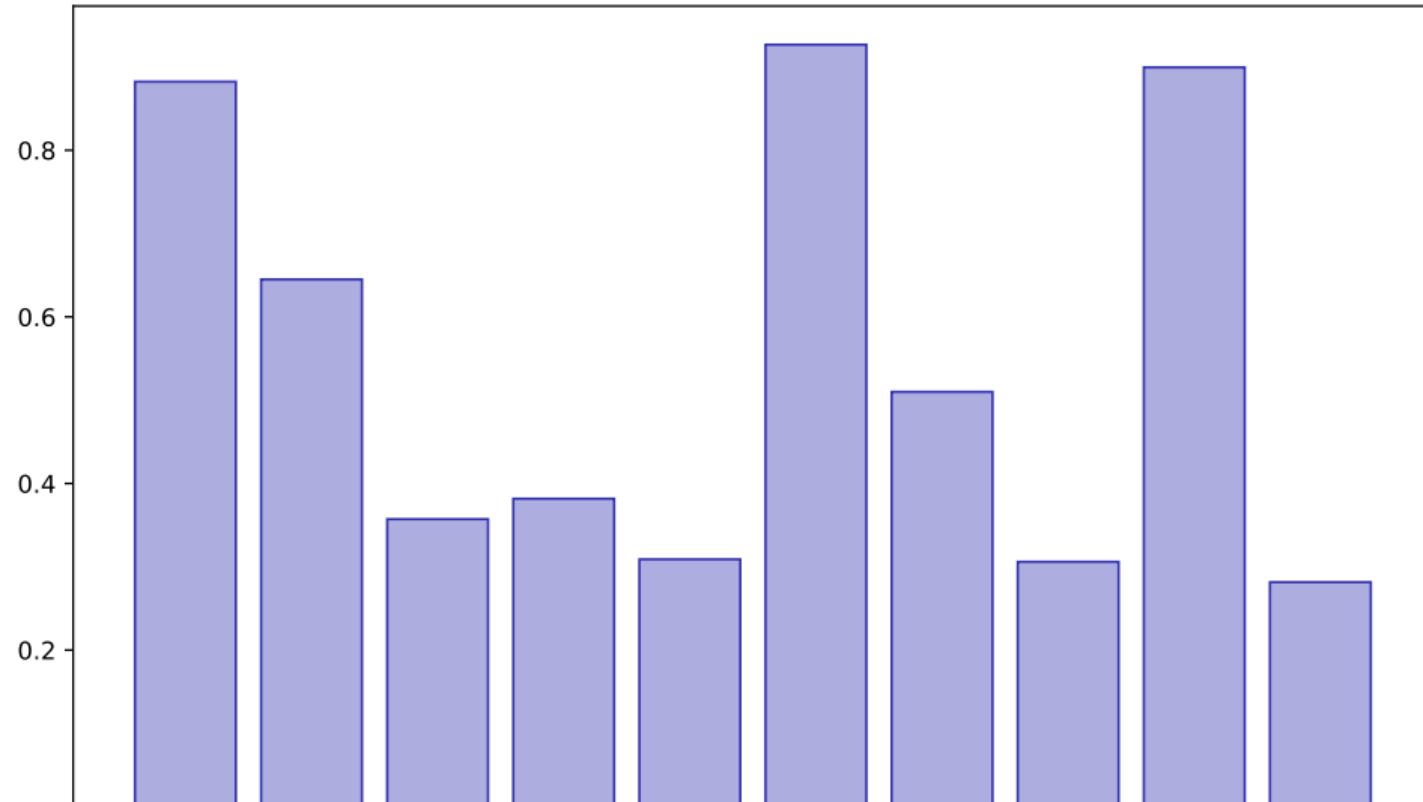
## Similarity Search



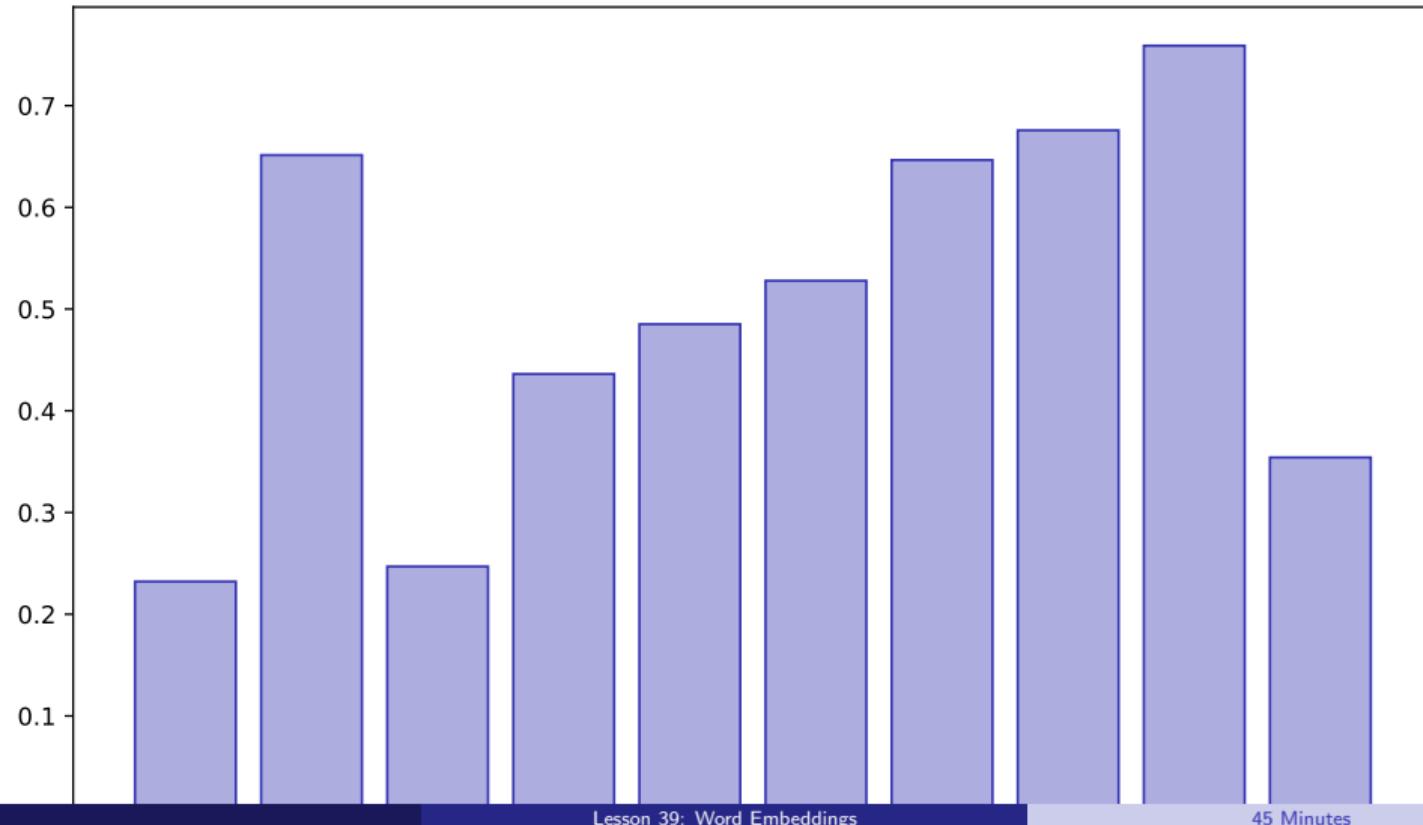
## Analogies



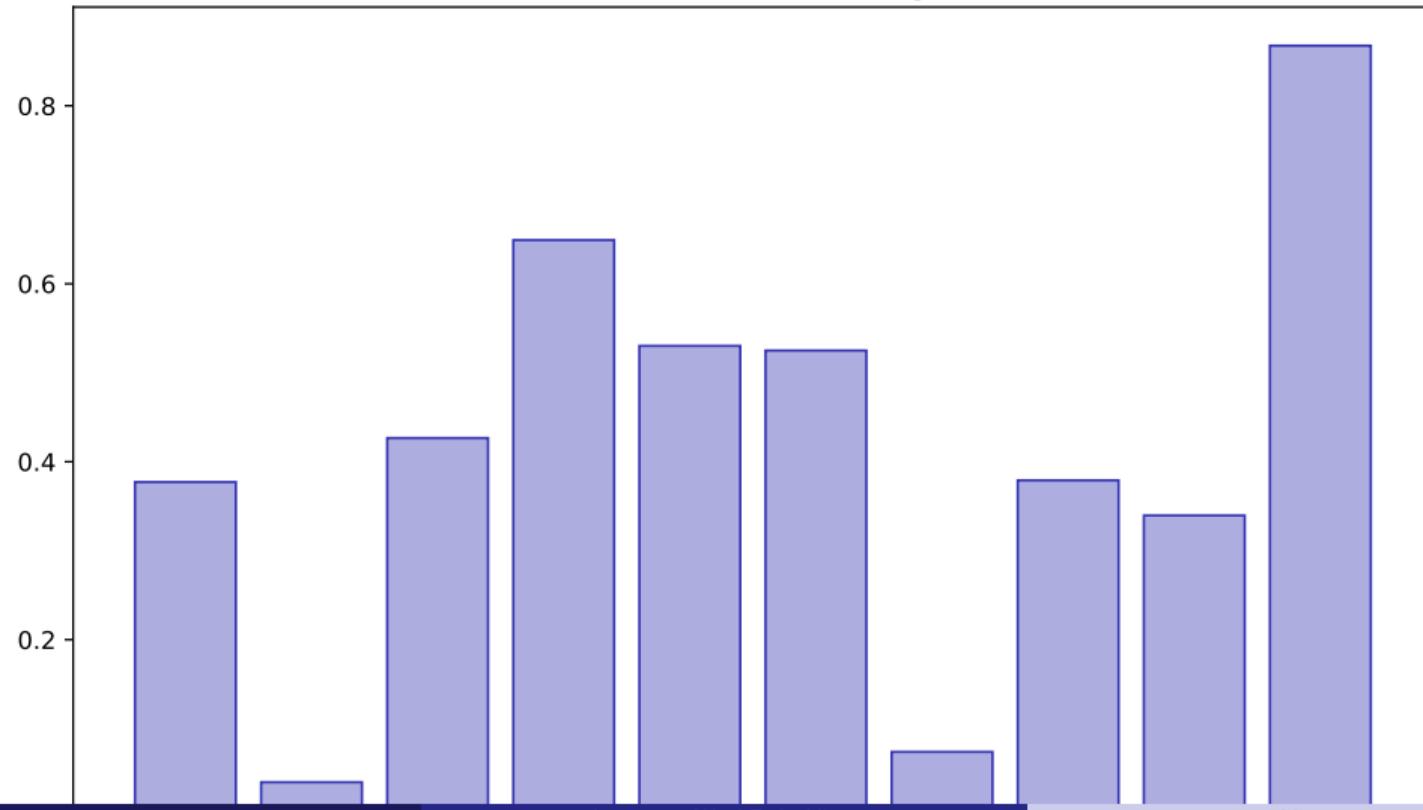
## Document Vectors



## Pretrained Models



## Finance Embeddings



## Lesson Summary

### Key Takeaways:

- Understand word vectors
- Use pre-trained Word2Vec
- Find similar words
- Create document embeddings

Apply these skills in your final project

## Lesson 40: Sentiment Analysis

Data Science with Python – BSc Course

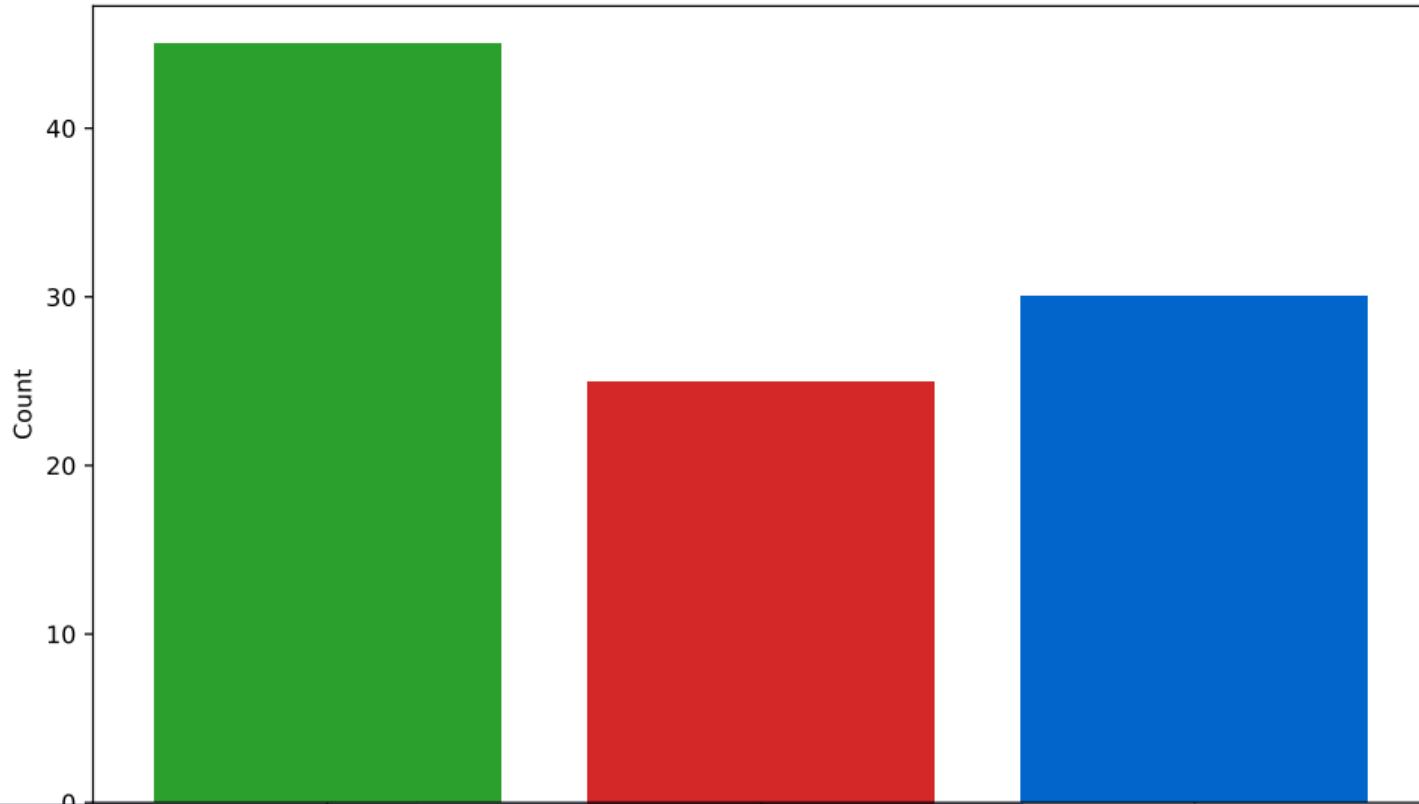
45 Minutes

**After this lesson, you will be able to:**

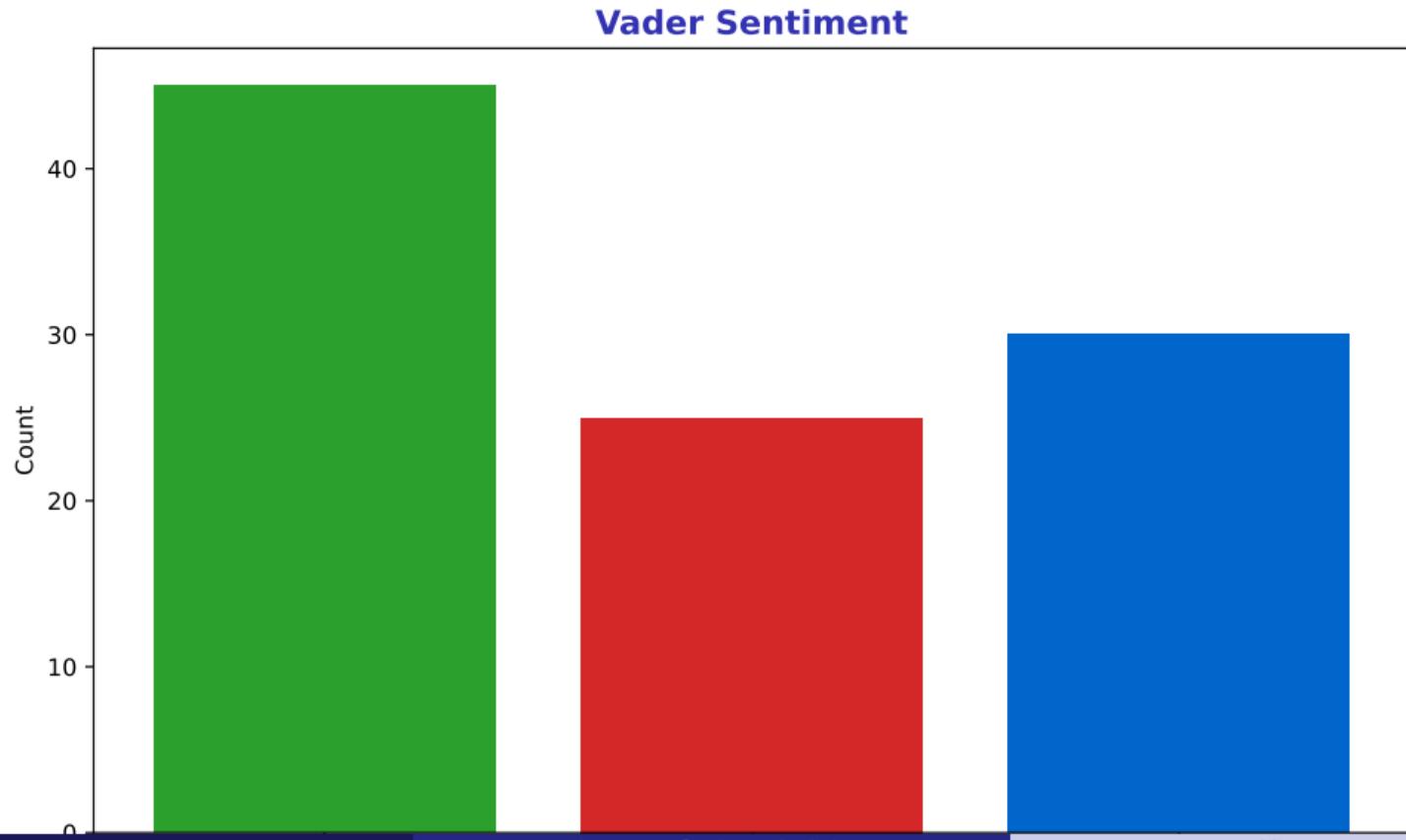
- Apply VADER sentiment
- Use FinBERT for finance
- Build sentiment classifiers
- Analyze market sentiment

**Building towards your final project**

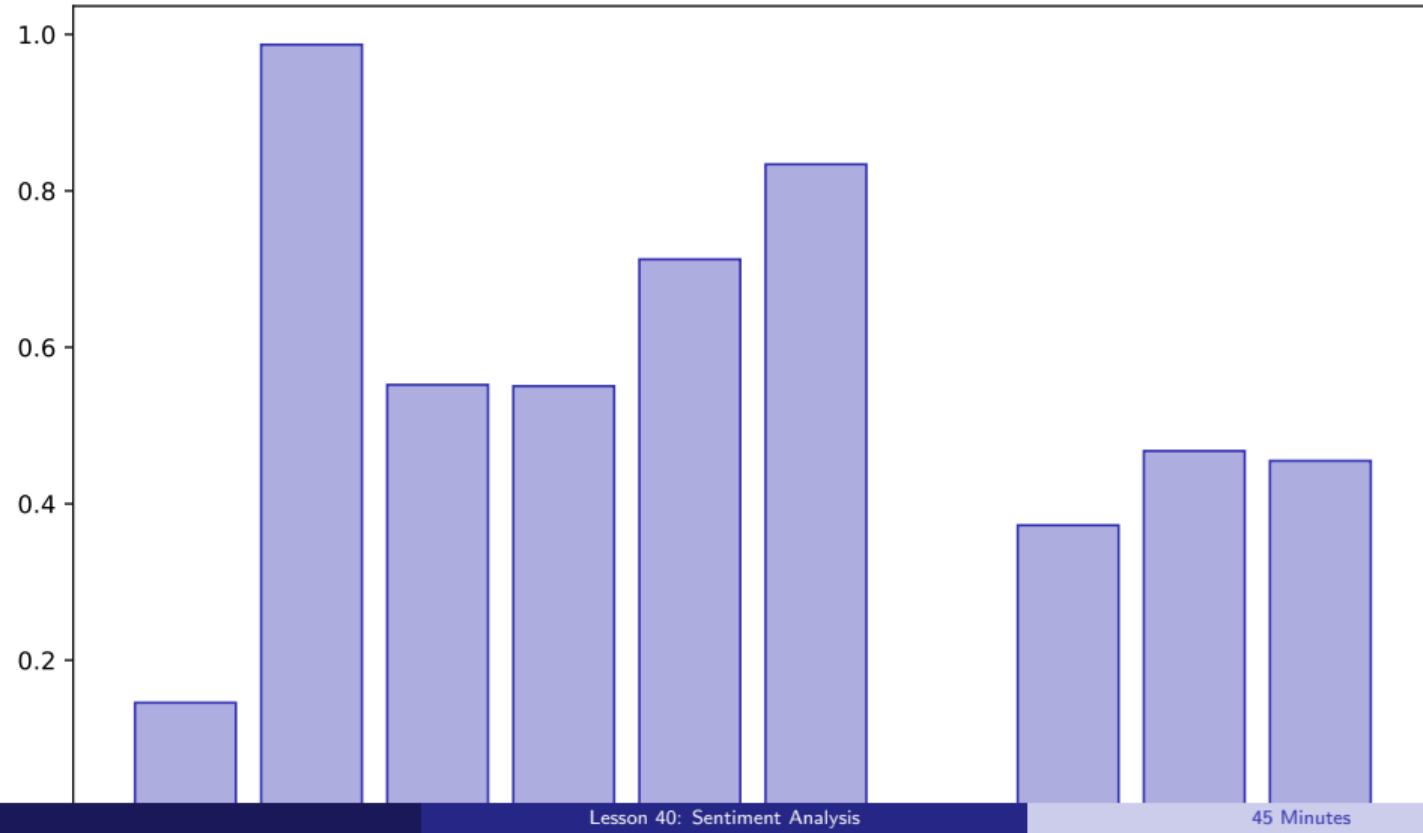
### Sentiment Concept



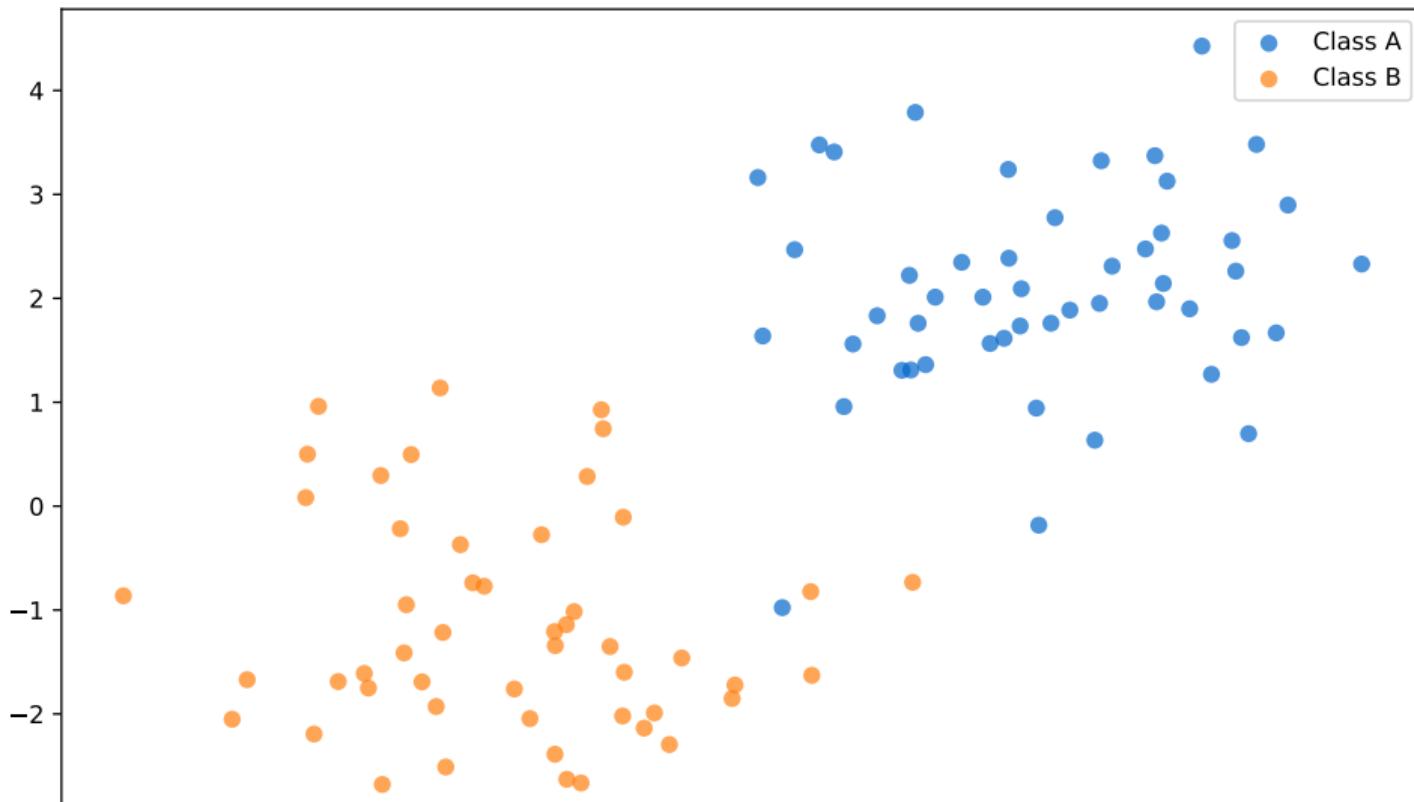
## Vader Sentiment



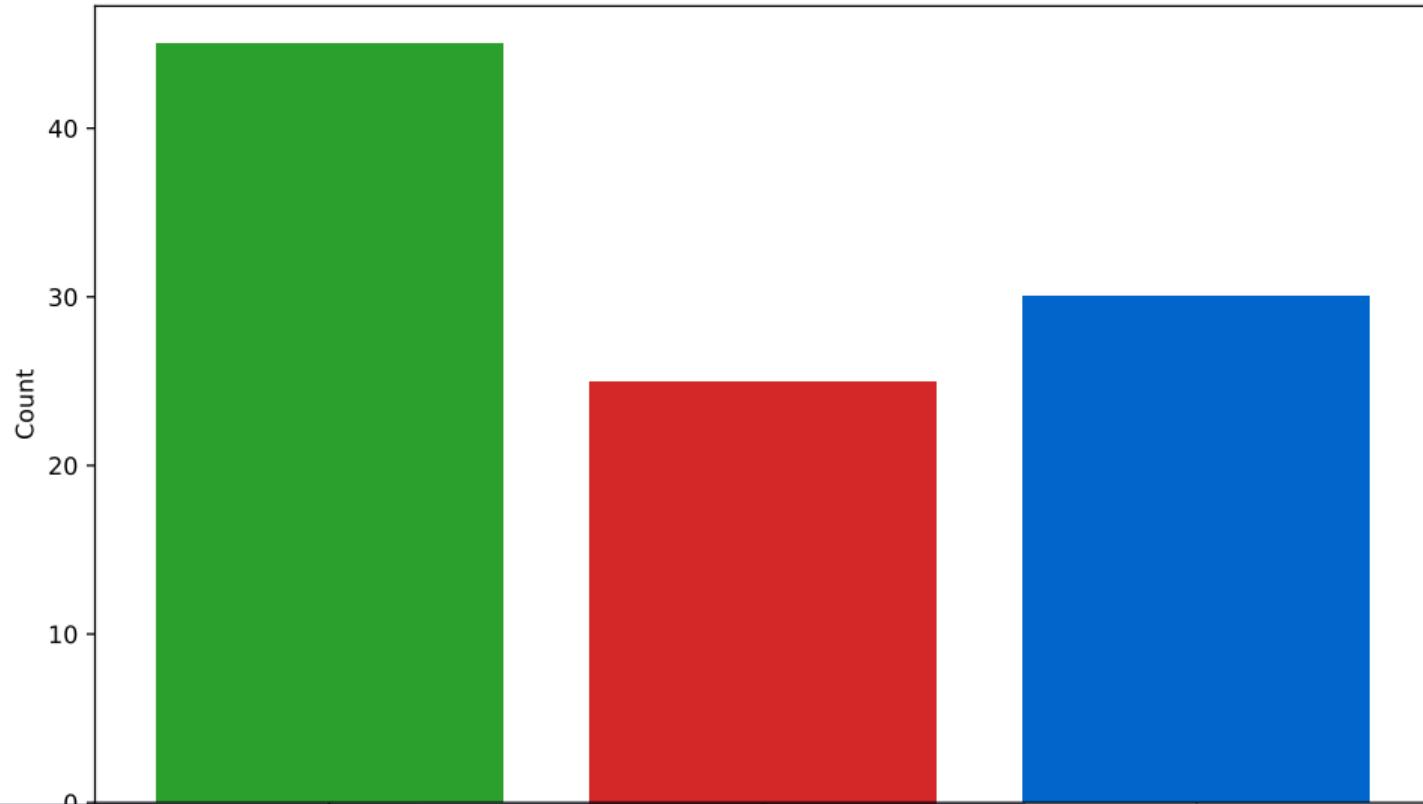
## Finbert Intro



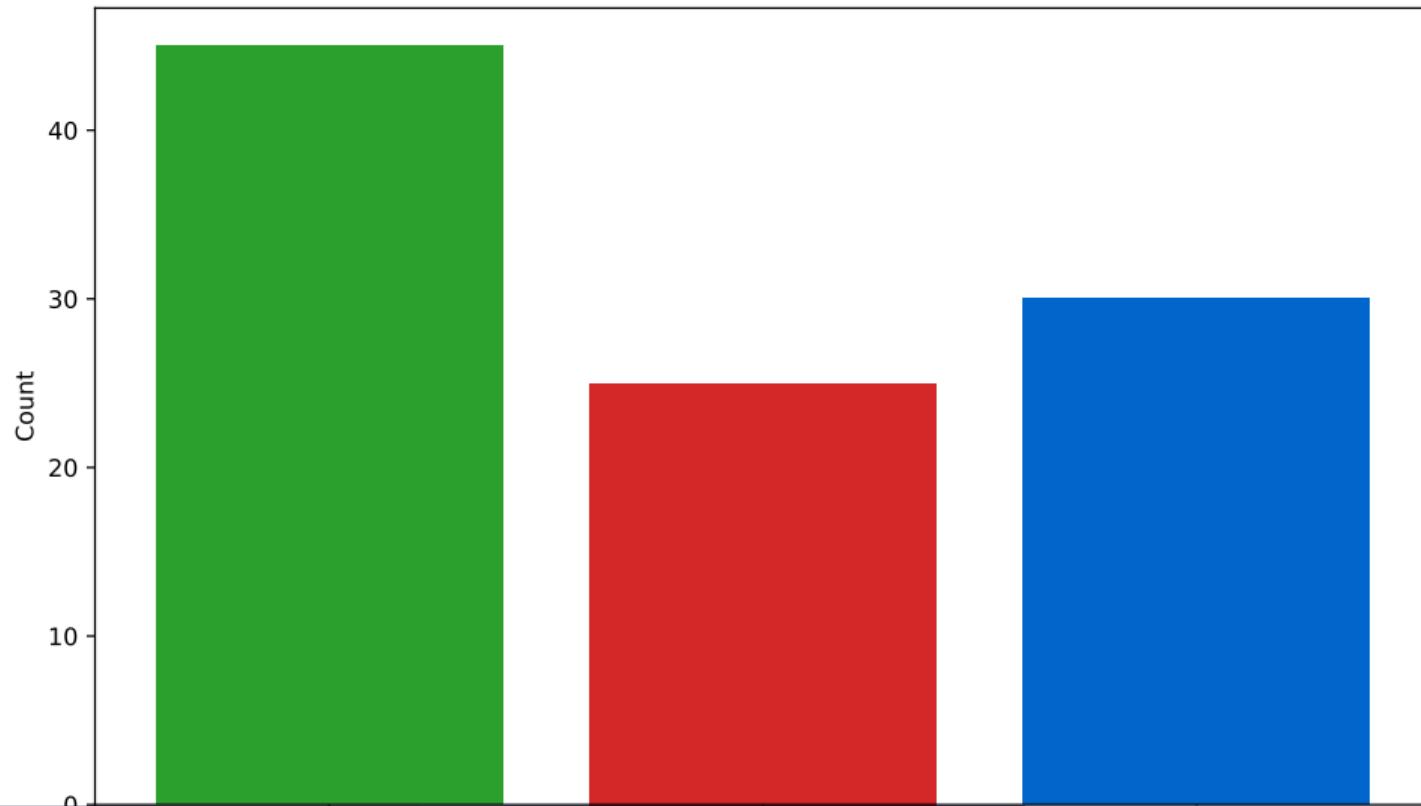
## Sentiment Classification



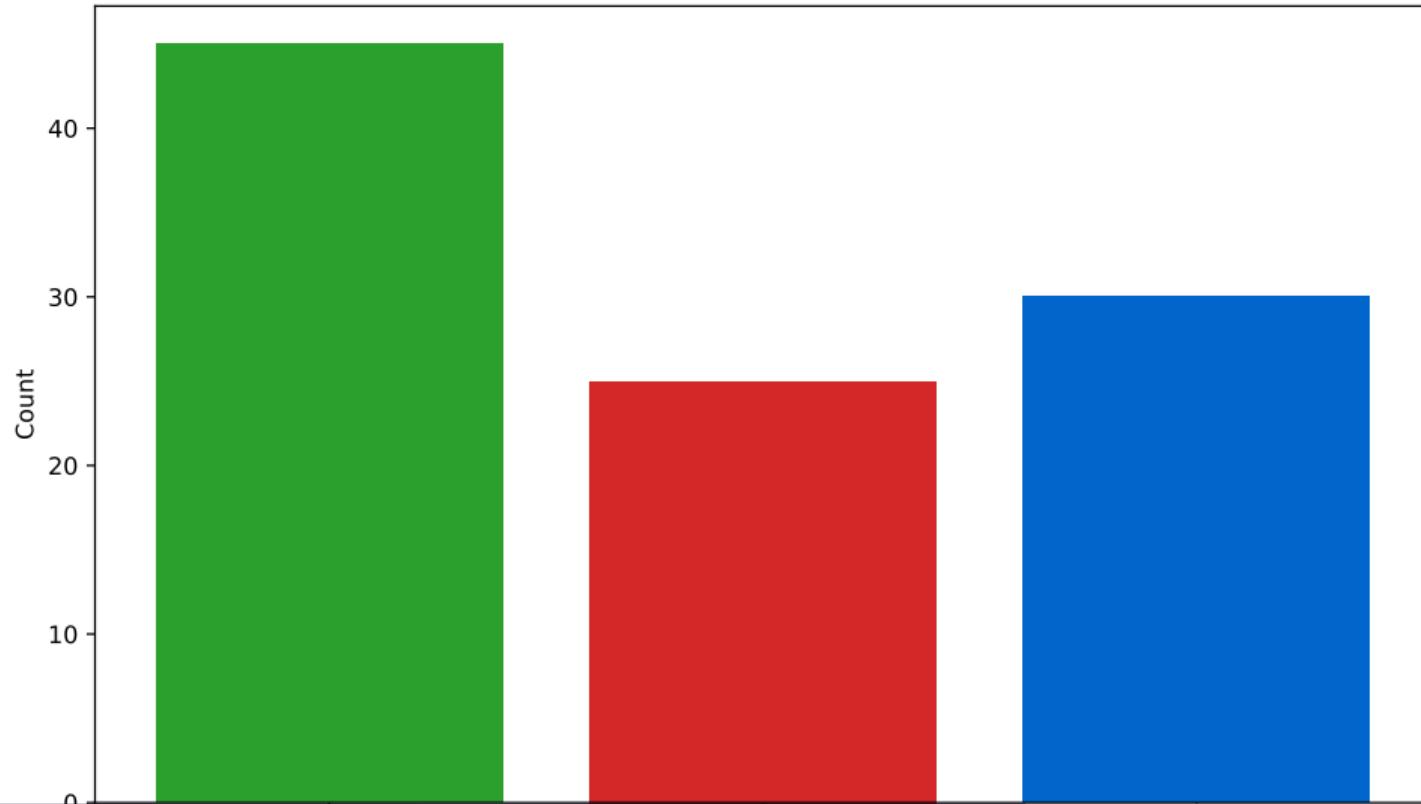
Aspect Sentiment



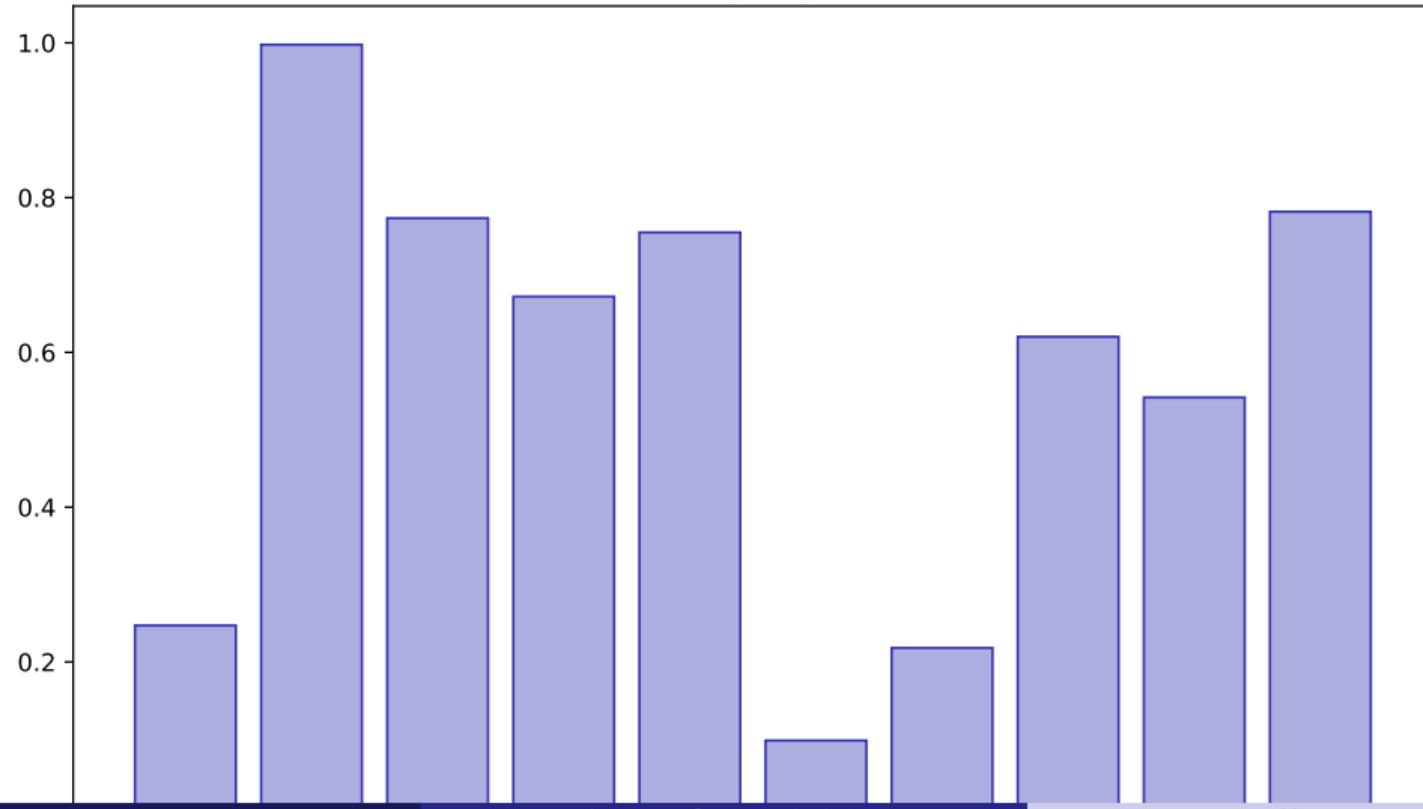
### Time Series Sentiment



## News Sentiment



## Trading Signals



## Lesson Summary

### Key Takeaways:

- Apply VADER sentiment
- Use FinBERT for finance
- Build sentiment classifiers
- Analyze market sentiment

Apply these skills in your final project

## Lesson 41: Model Serialization

Data Science with Python – BSc Course

45 Minutes

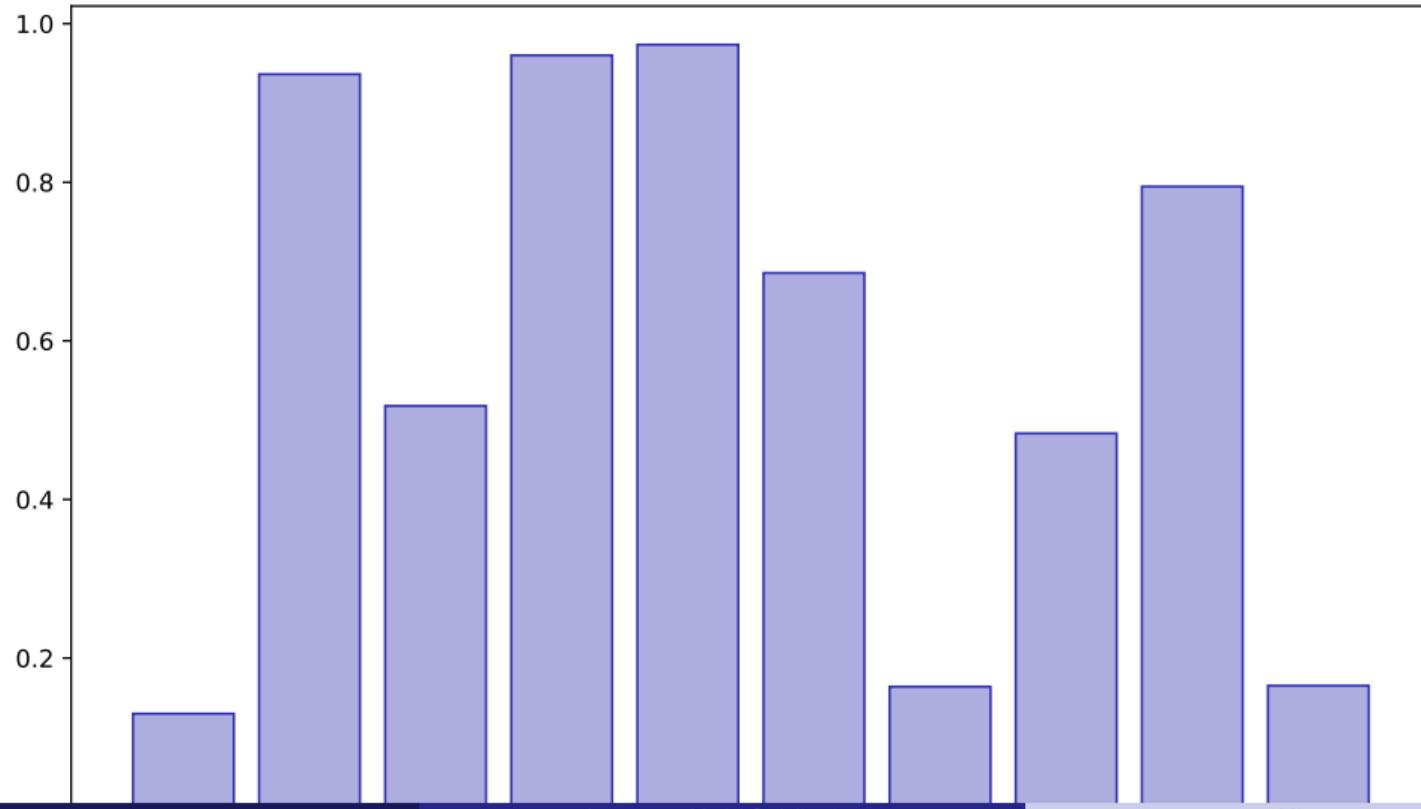
# Learning Objectives

**After this lesson, you will be able to:**

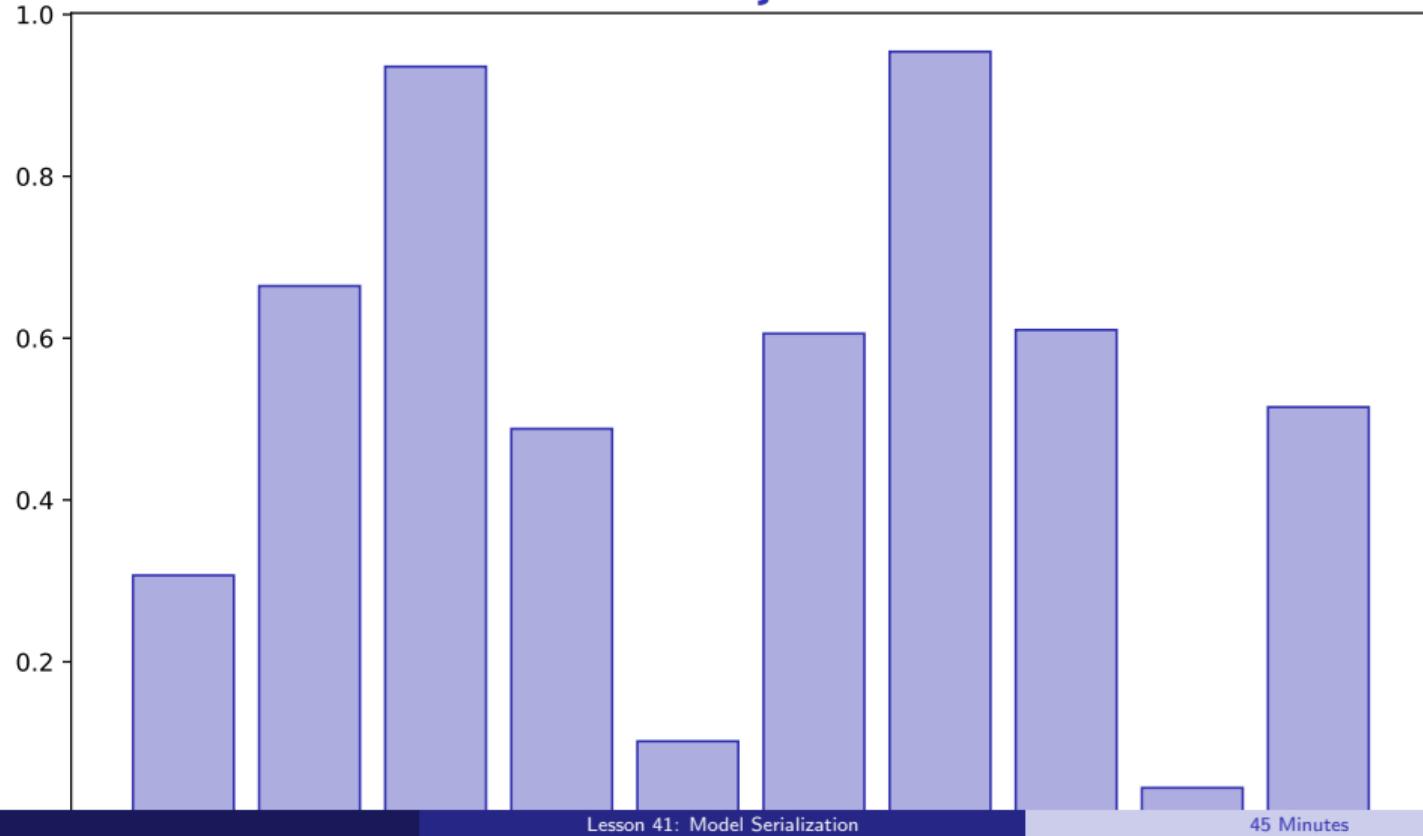
- Save models with joblib
- Load and predict
- Version models
- Prepare for deployment

**Building towards your final project**

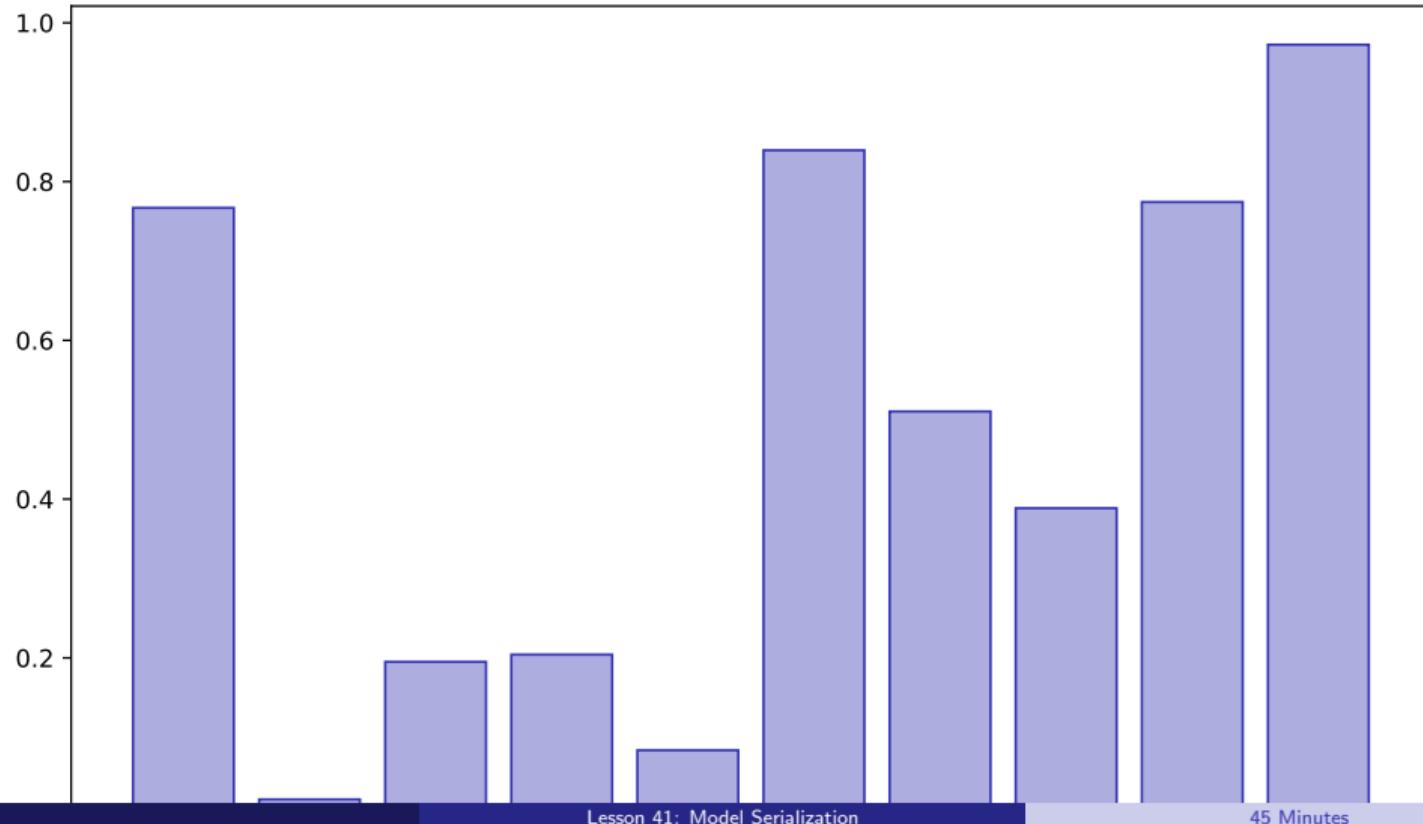
## Serialization Concept



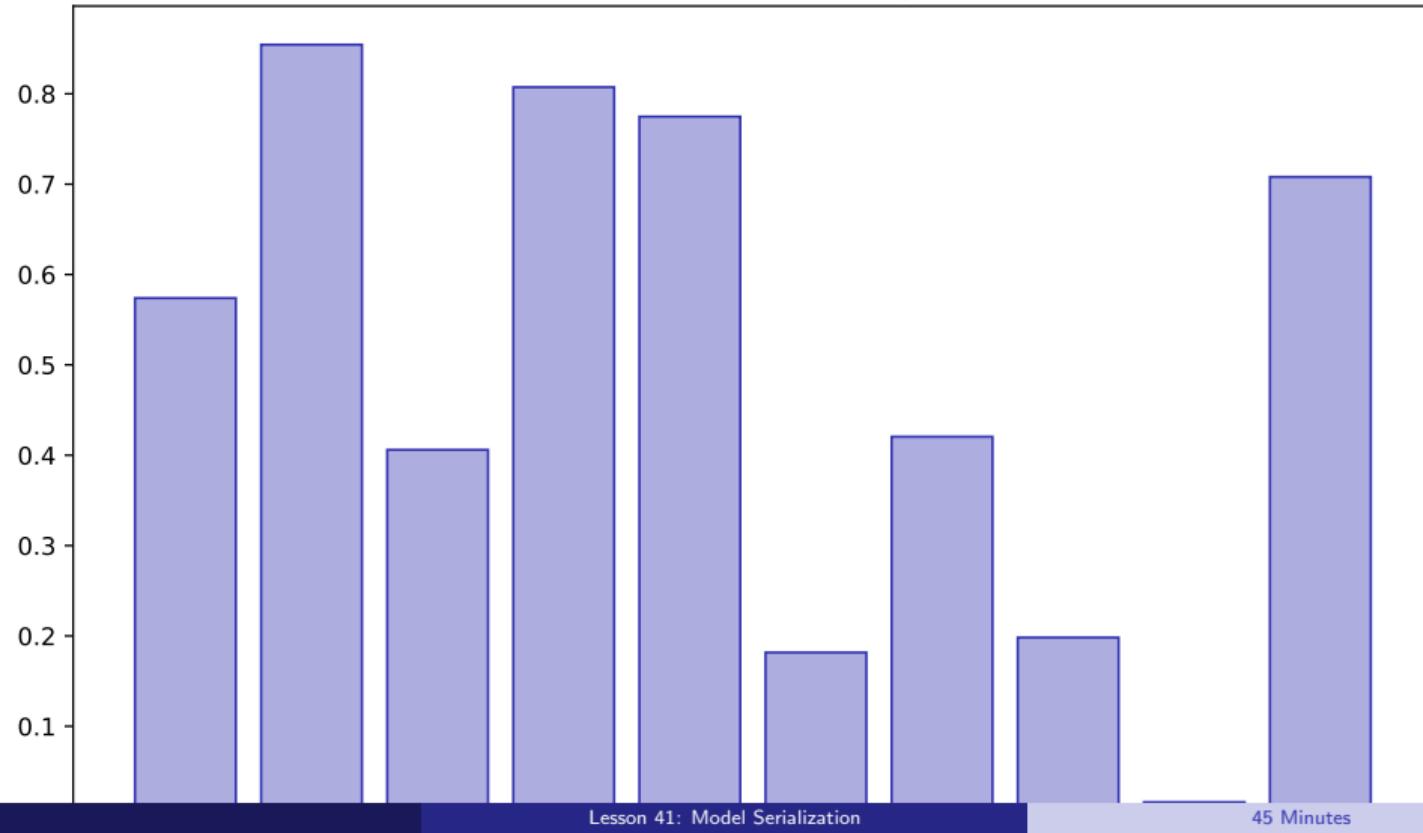
## Pickle Joblib



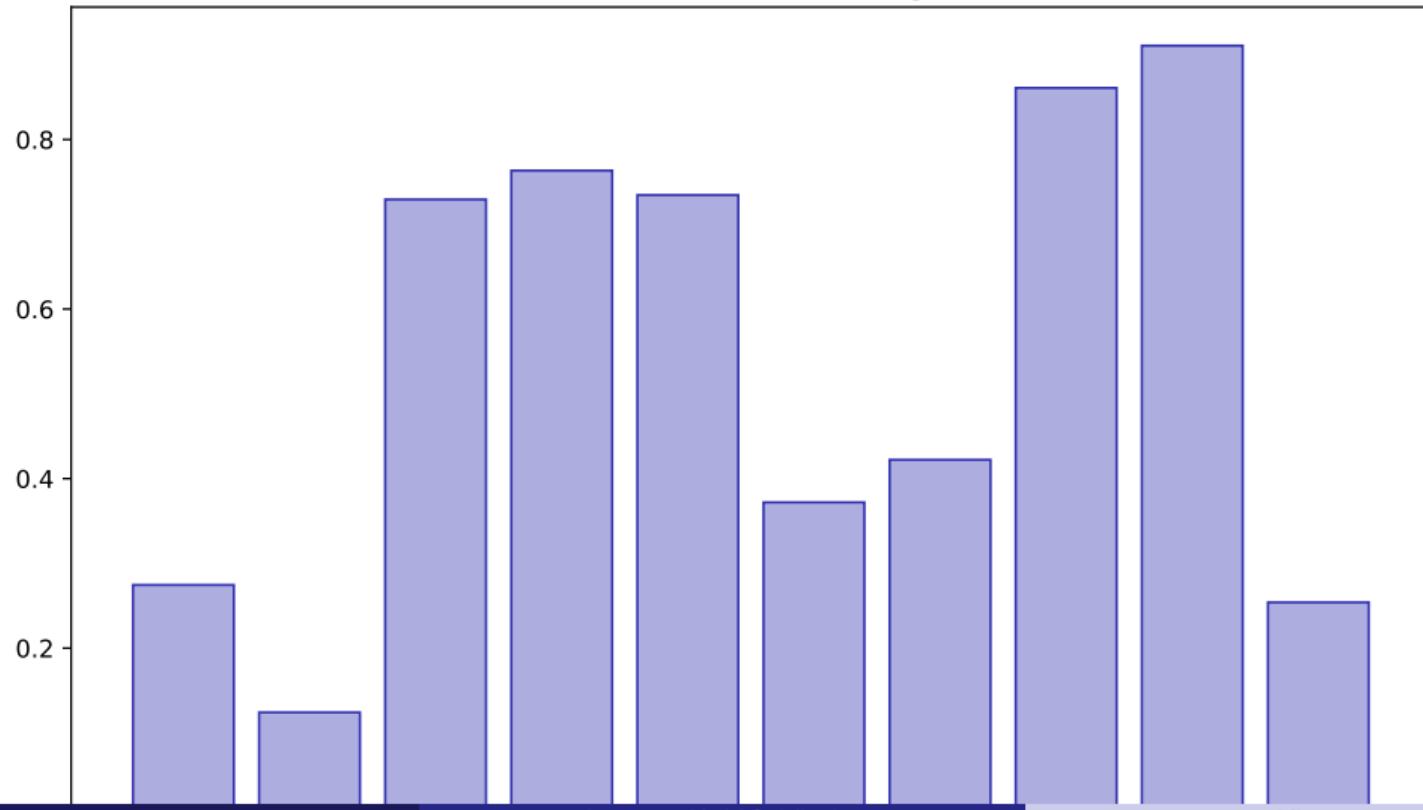
## Save Load Workflow



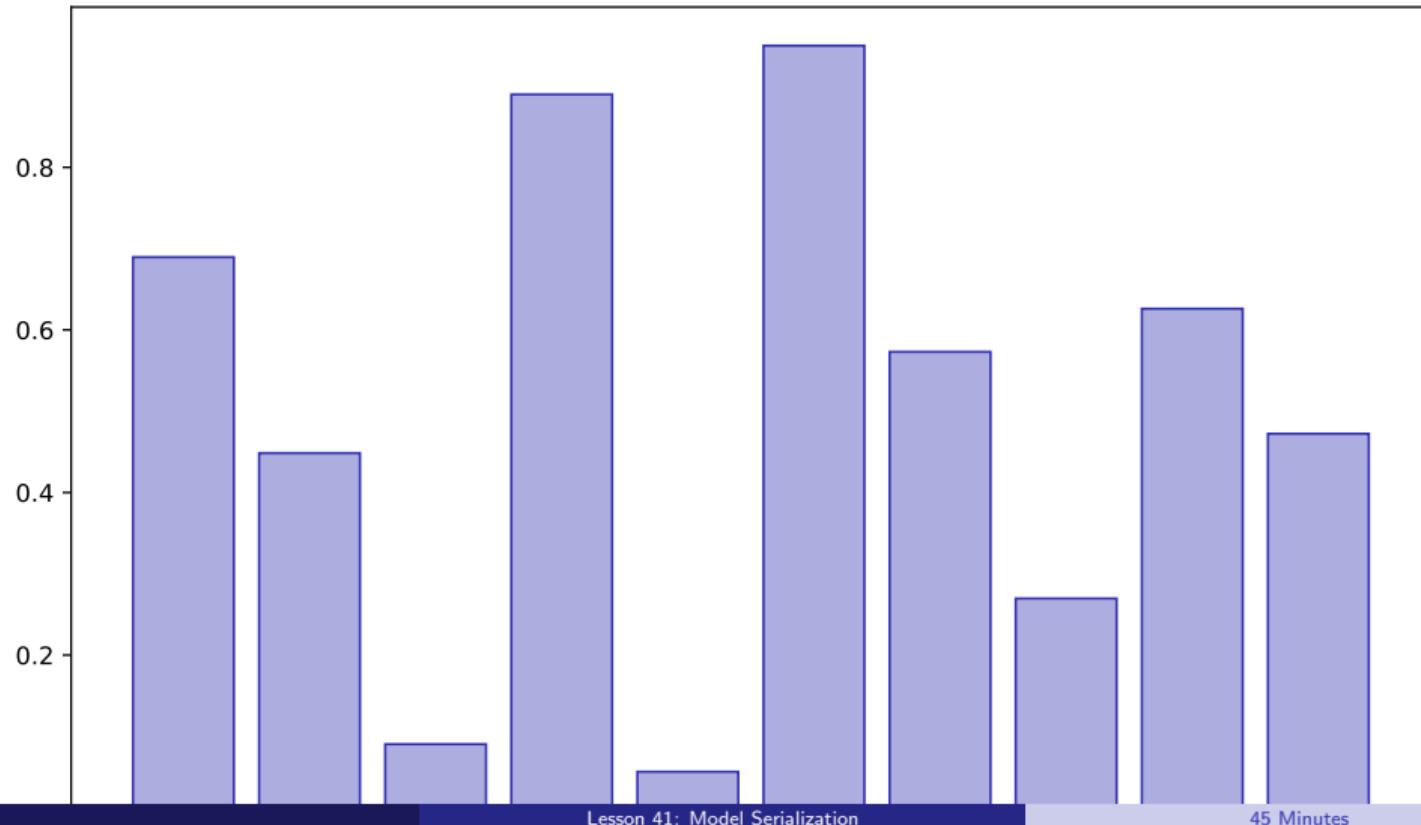
## Model Versioning



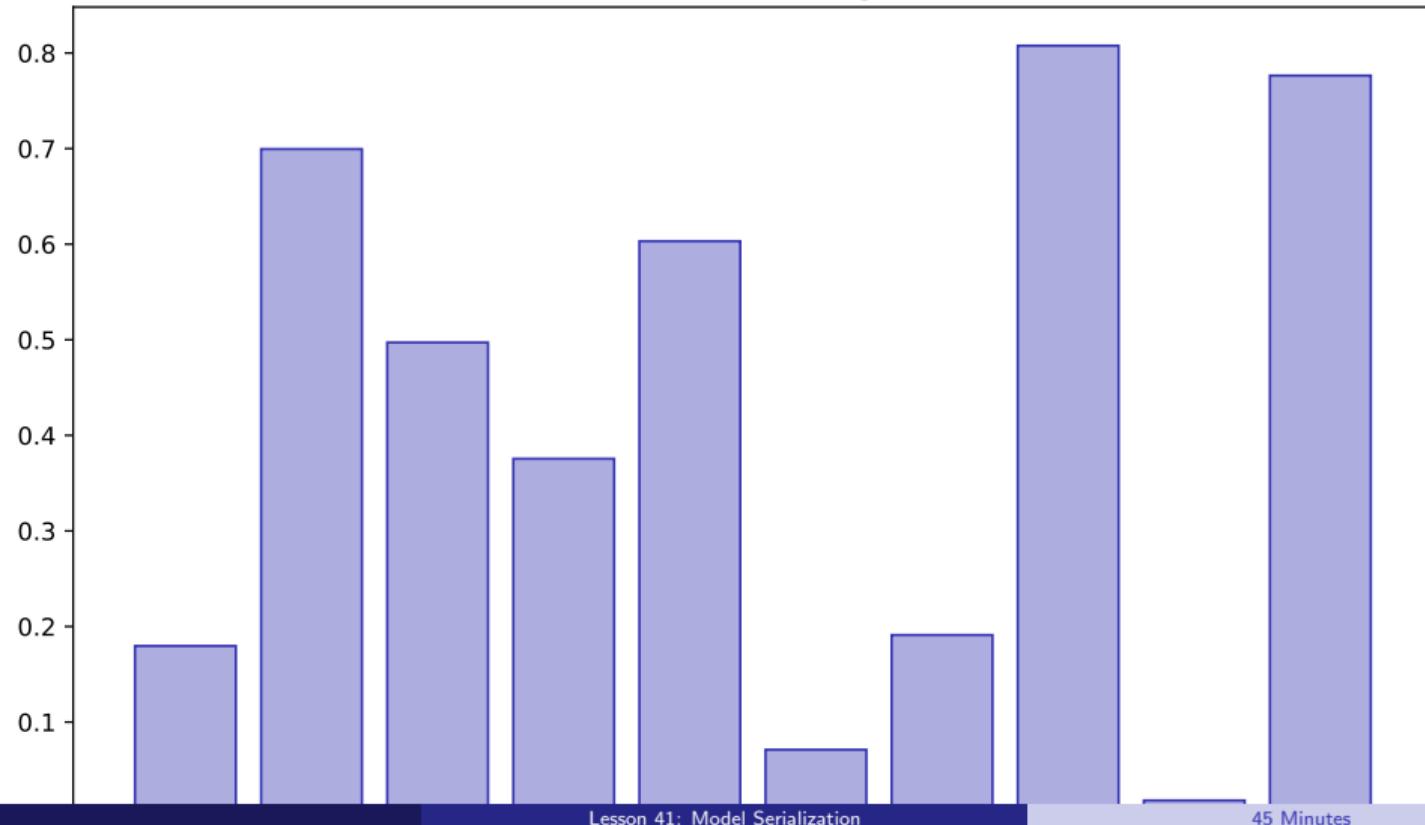
## Metadata Tracking



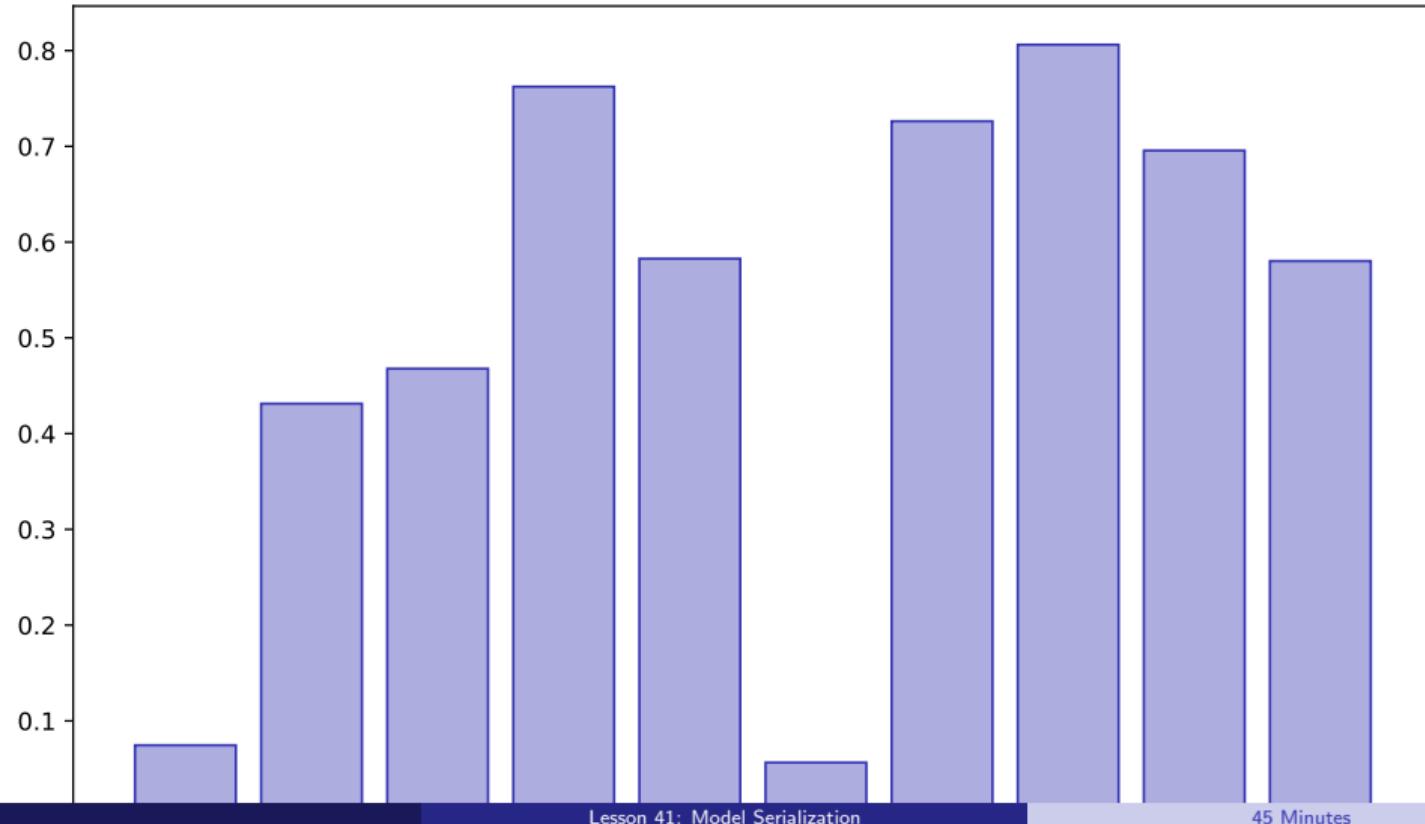
## Sklearn Persistence



## Keras Saving



### Production Models



## Lesson Summary

### Key Takeaways:

- Save models with joblib
- Load and predict
- Version models
- Prepare for deployment

Apply these skills in your final project

## Lesson 42: REST APIs with FastAPI

Data Science with Python – BSc Course

45 Minutes

# Learning Objectives

**After this lesson, you will be able to:**

- Create API endpoints
- Design input schemas
- Handle predictions
- Document with Swagger

**Building towards your final project**

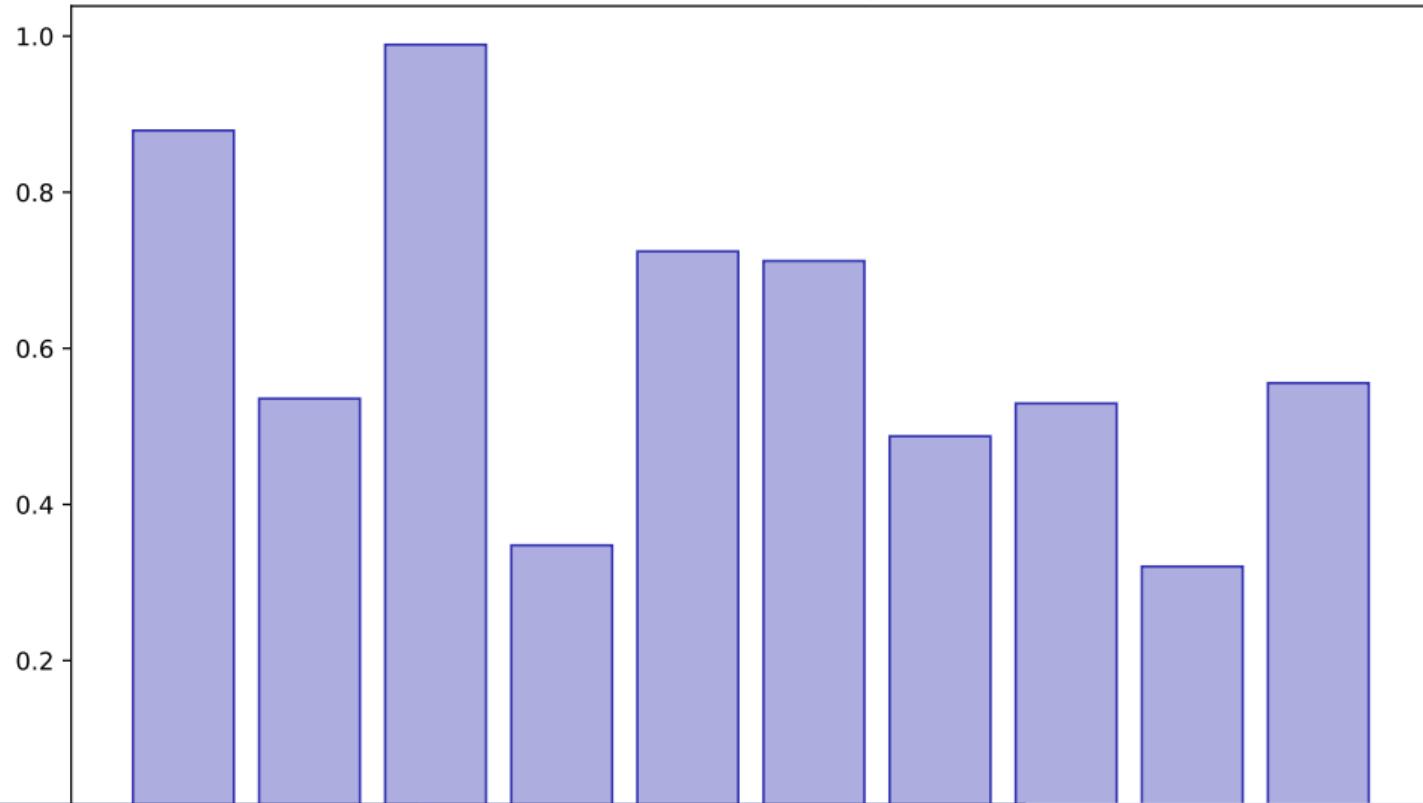
## Api Concept



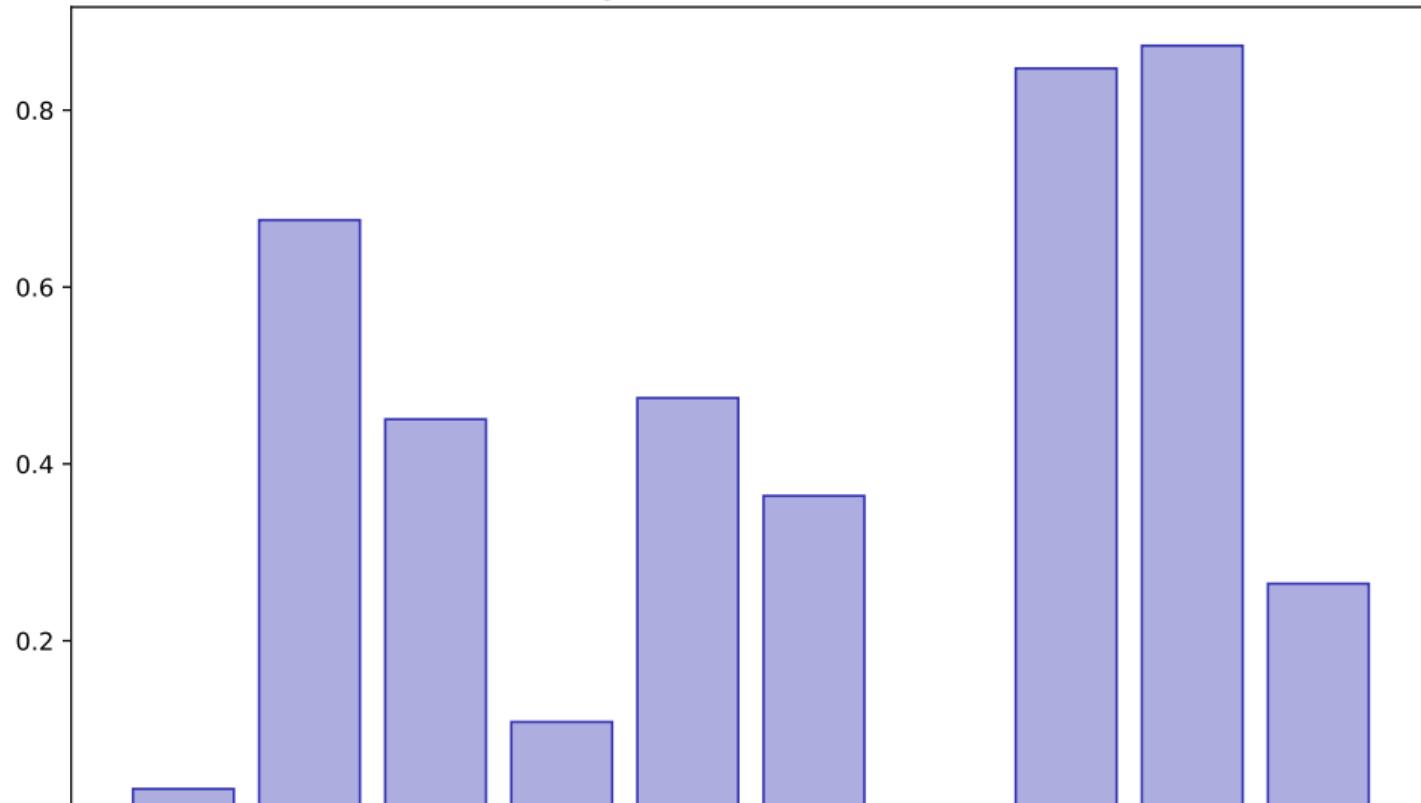
## Fastapi Basics



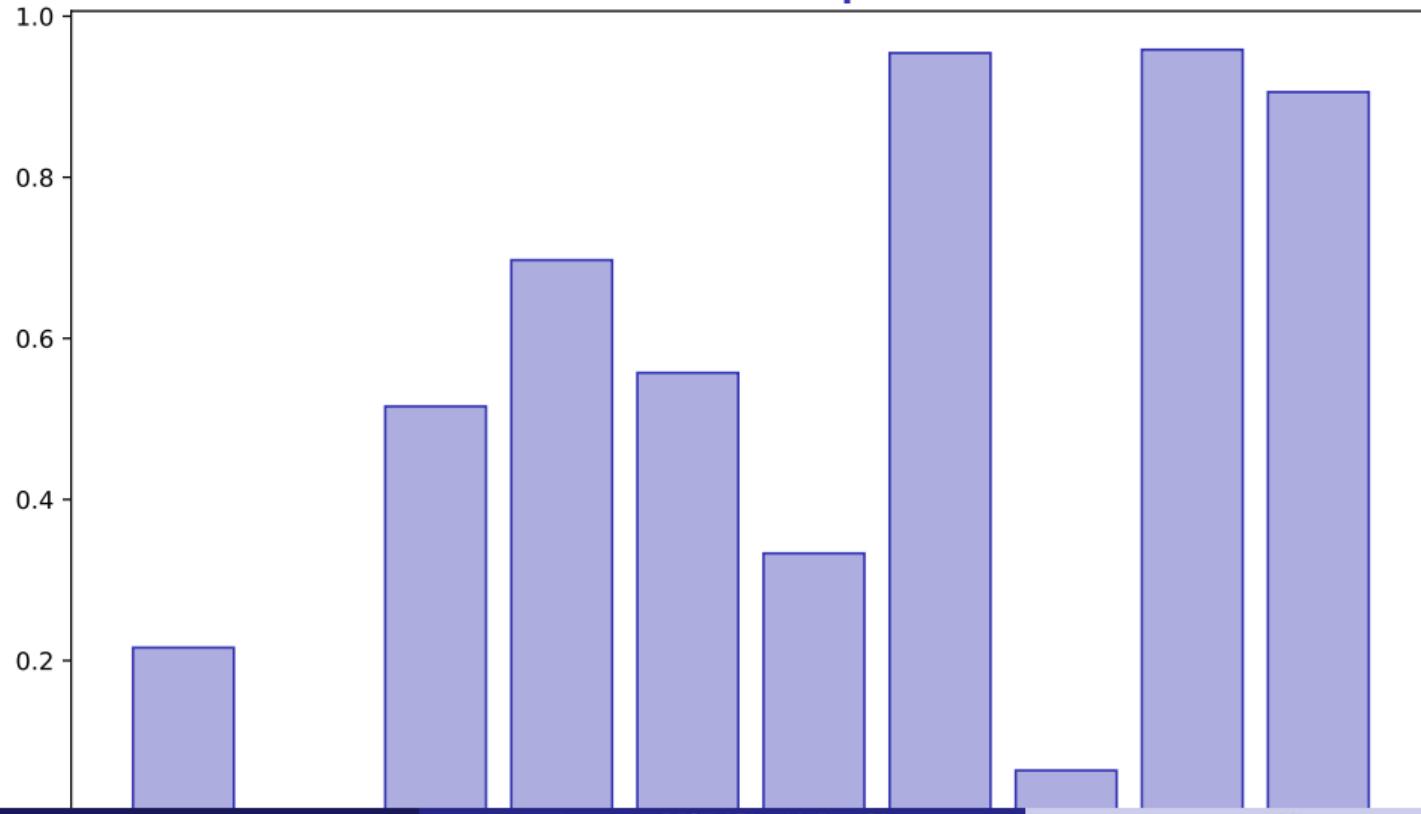
## Endpoint Design



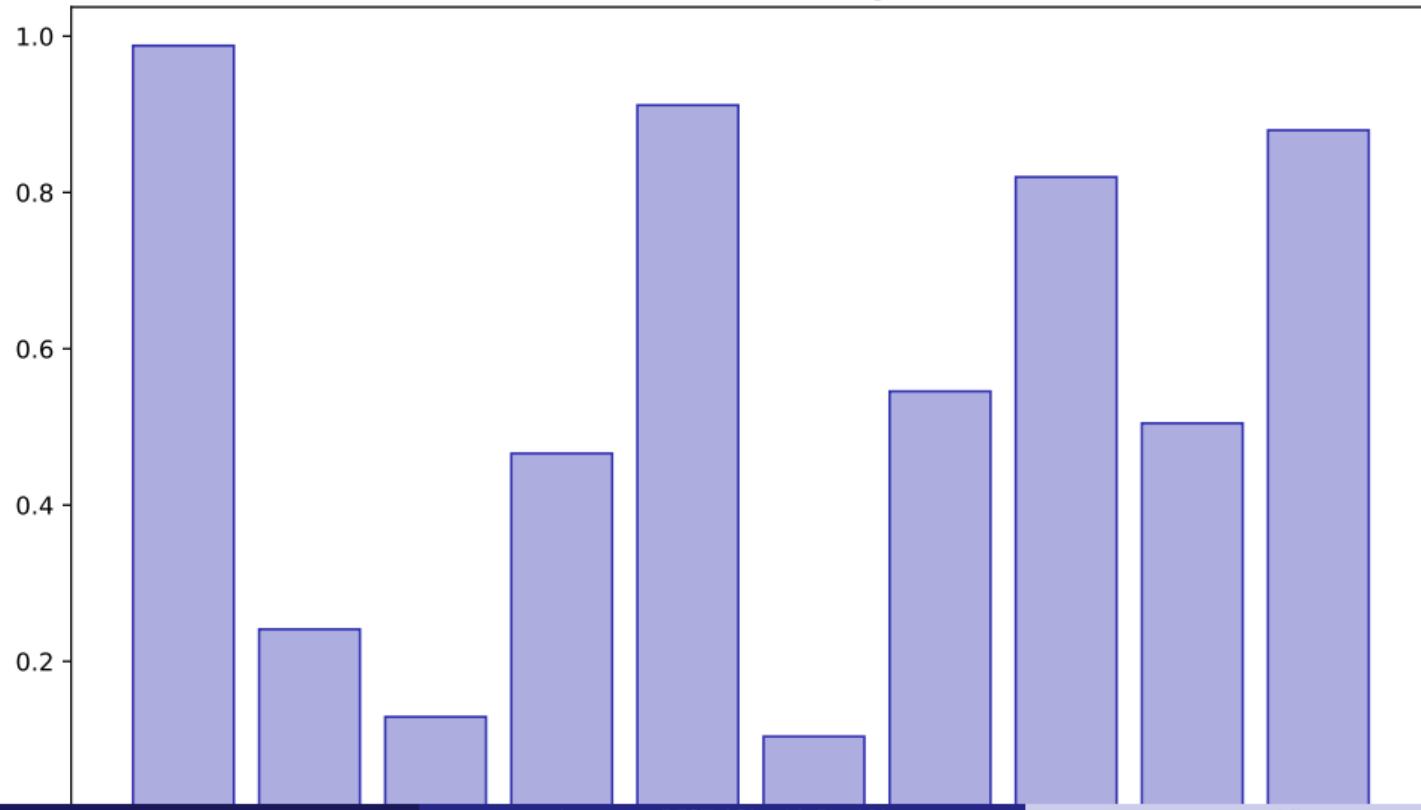
## Pydantic Schemas



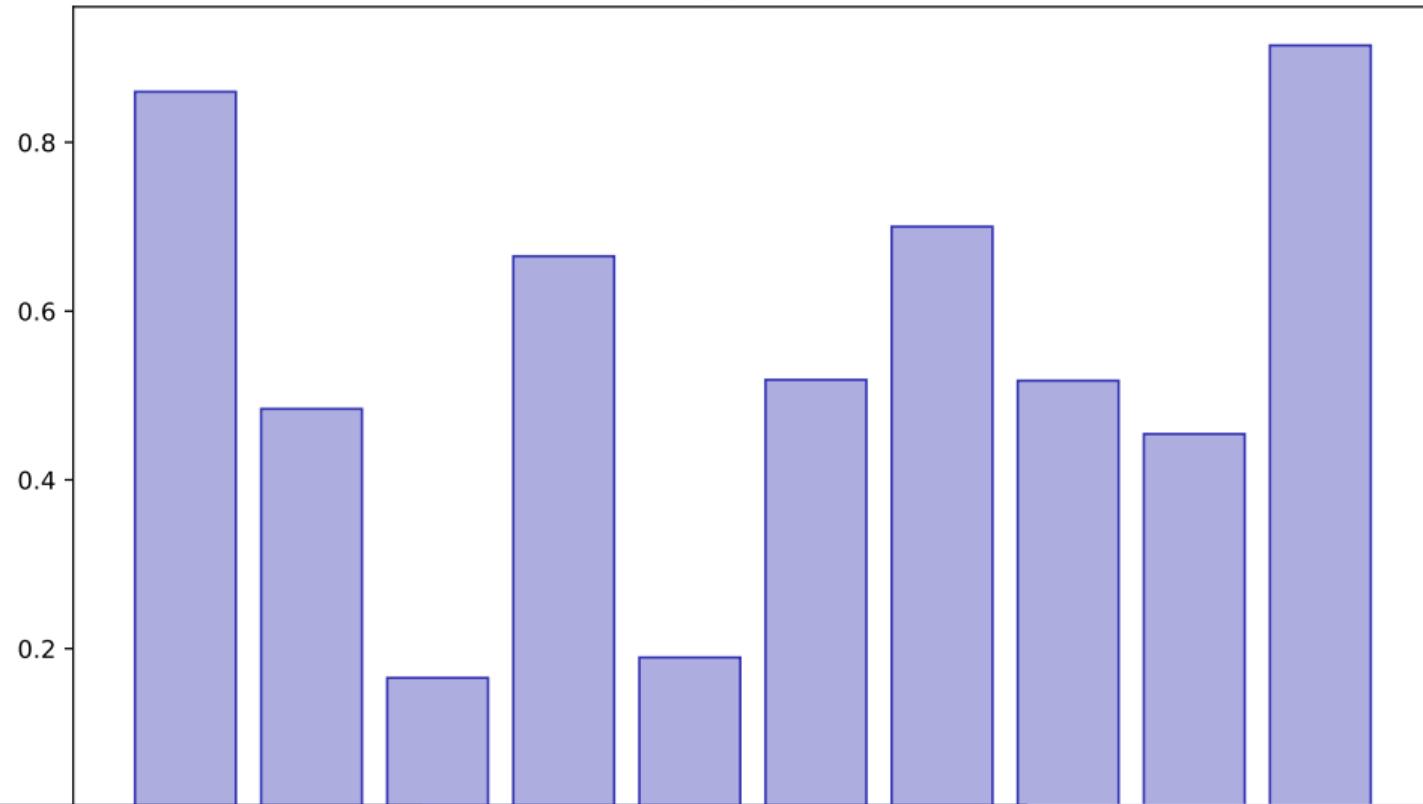
### Prediction Endpoint



## Error Handling



## Swagger Docs



## Finance Api



# Lesson Summary

## Key Takeaways:

- Create API endpoints
- Design input schemas
- Handle predictions
- Document with Swagger

Apply these skills in your final project

## Lesson 43: Streamlit Dashboards

Data Science with Python – BSc Course

45 Minutes

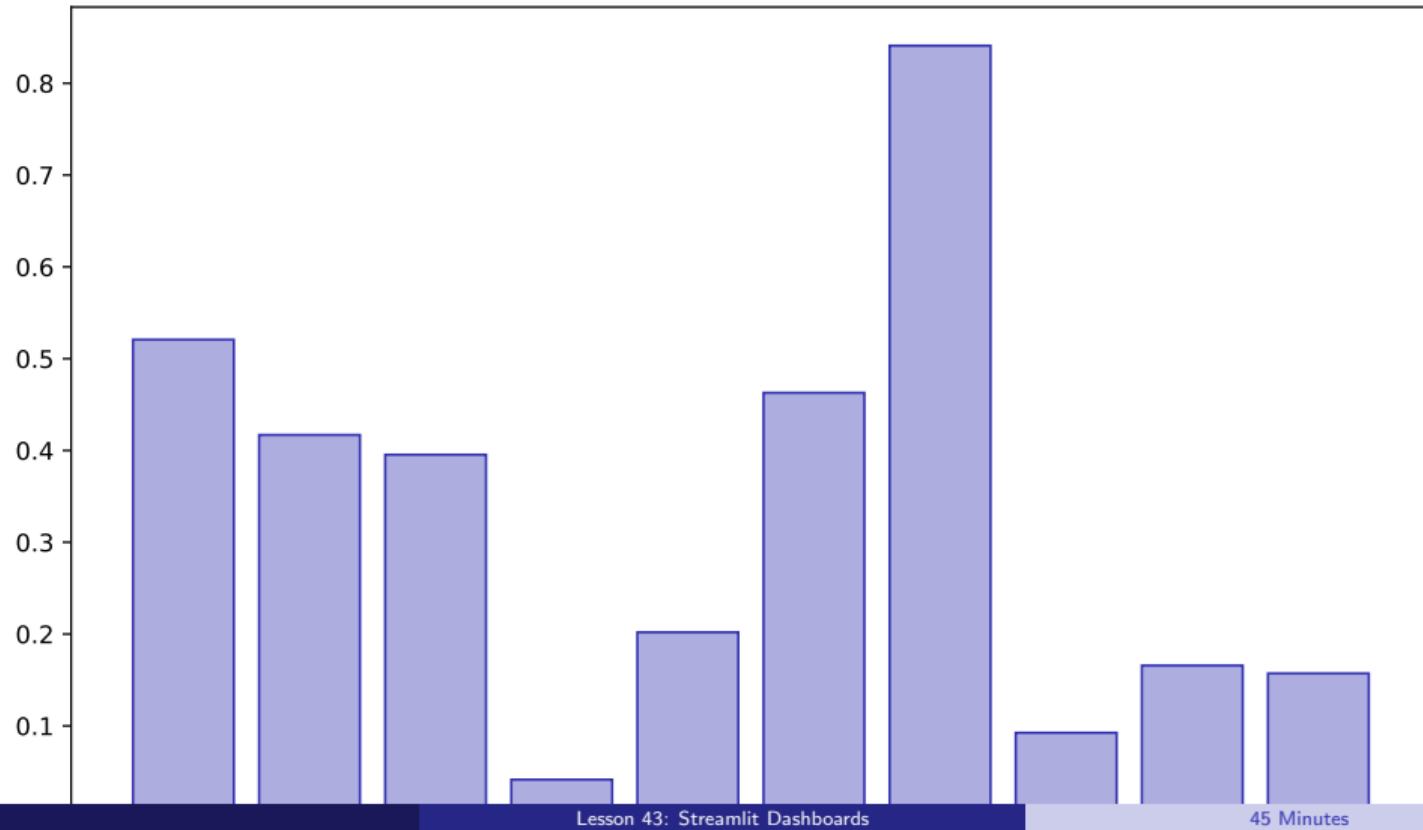
# Learning Objectives

**After this lesson, you will be able to:**

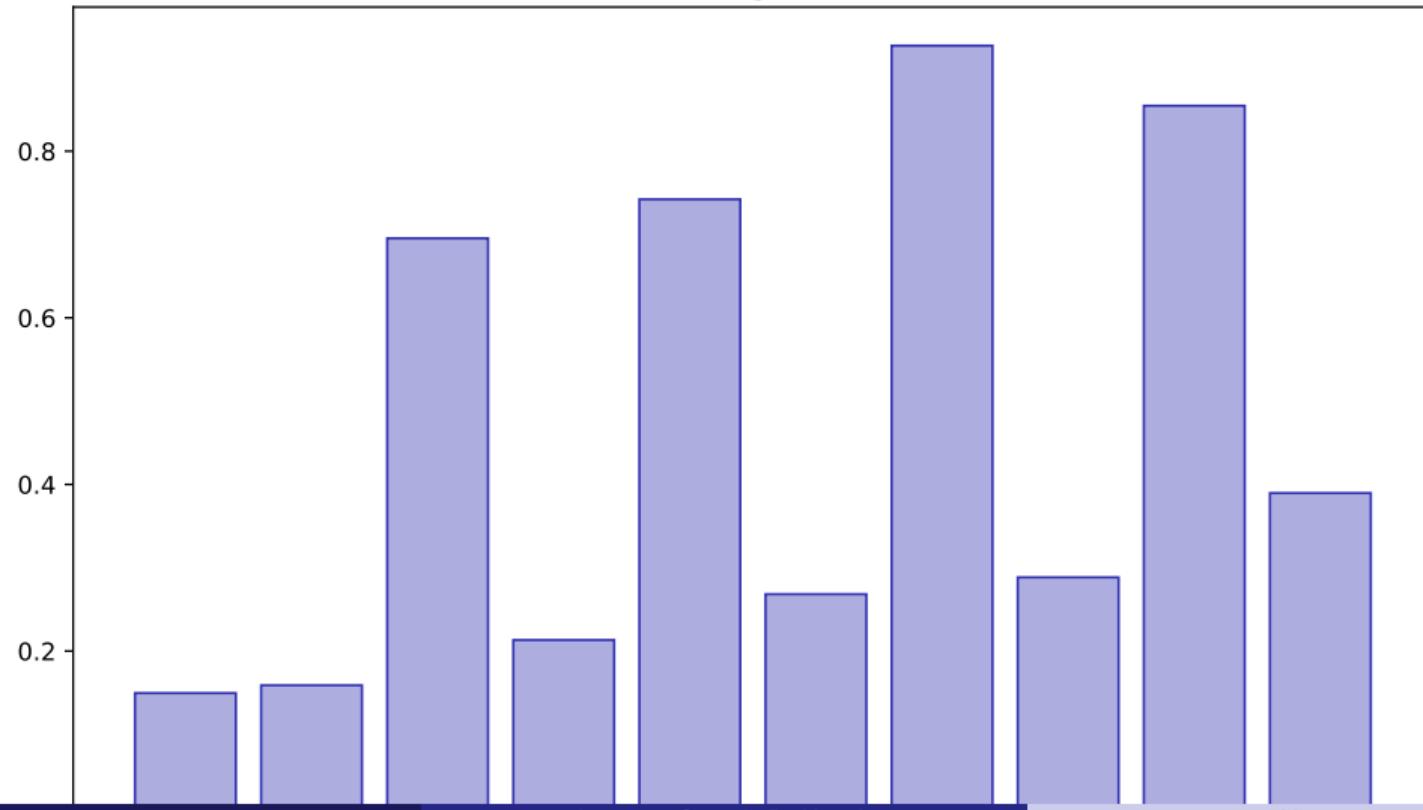
- Build interactive apps
- Add widgets and inputs
- Display predictions
- Create financial dashboards

**Building towards your final project**

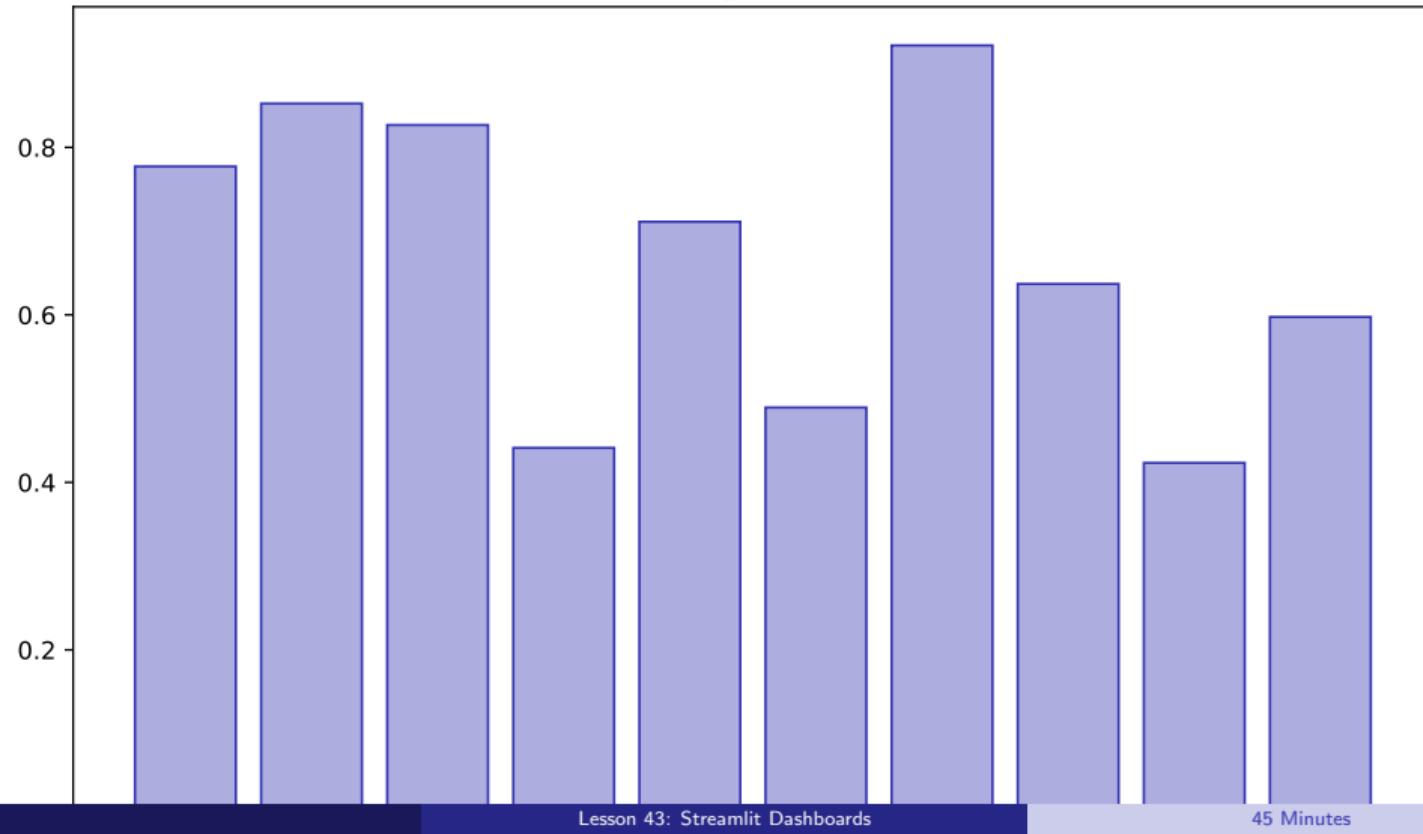
## Streamlit Intro



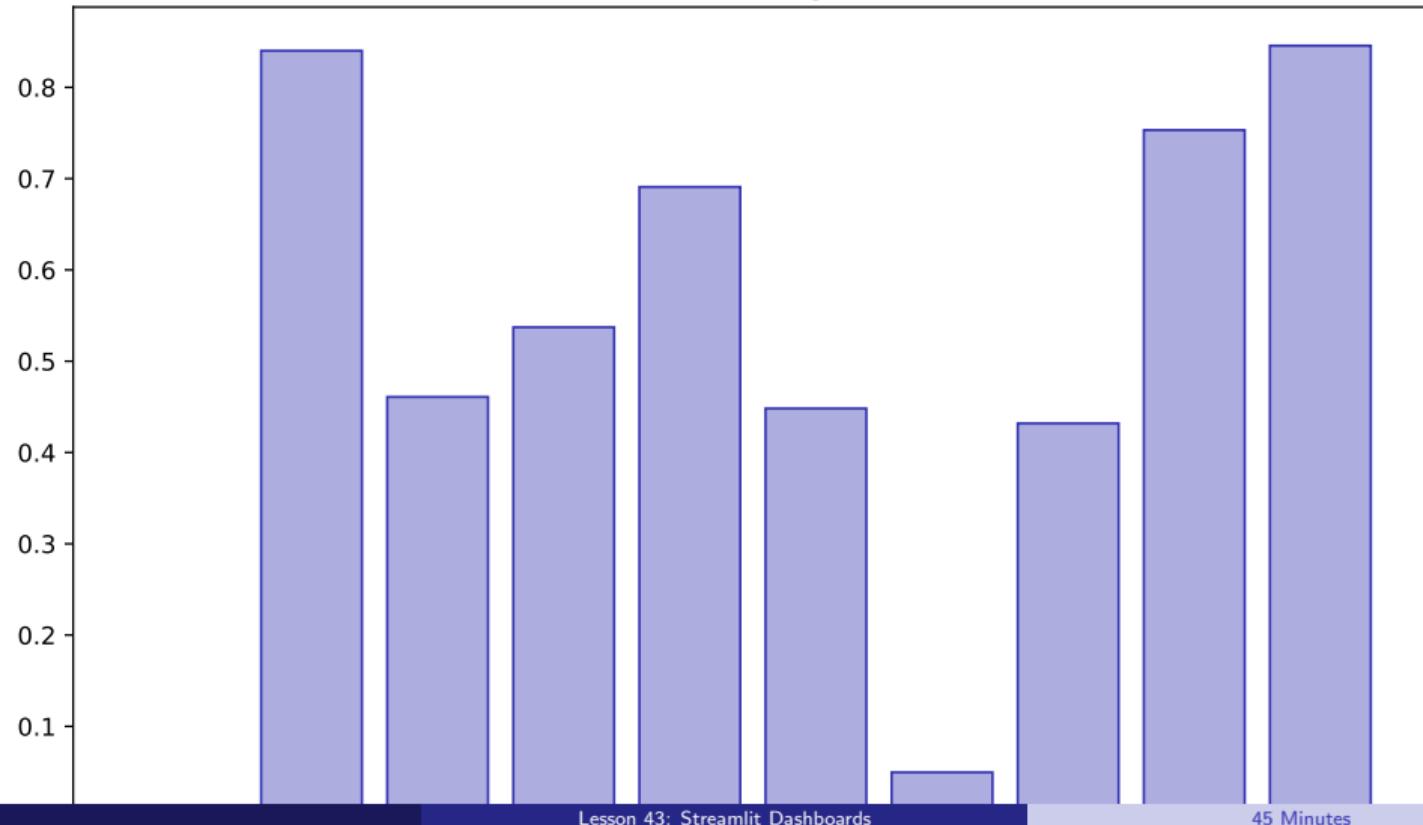
## Widgets



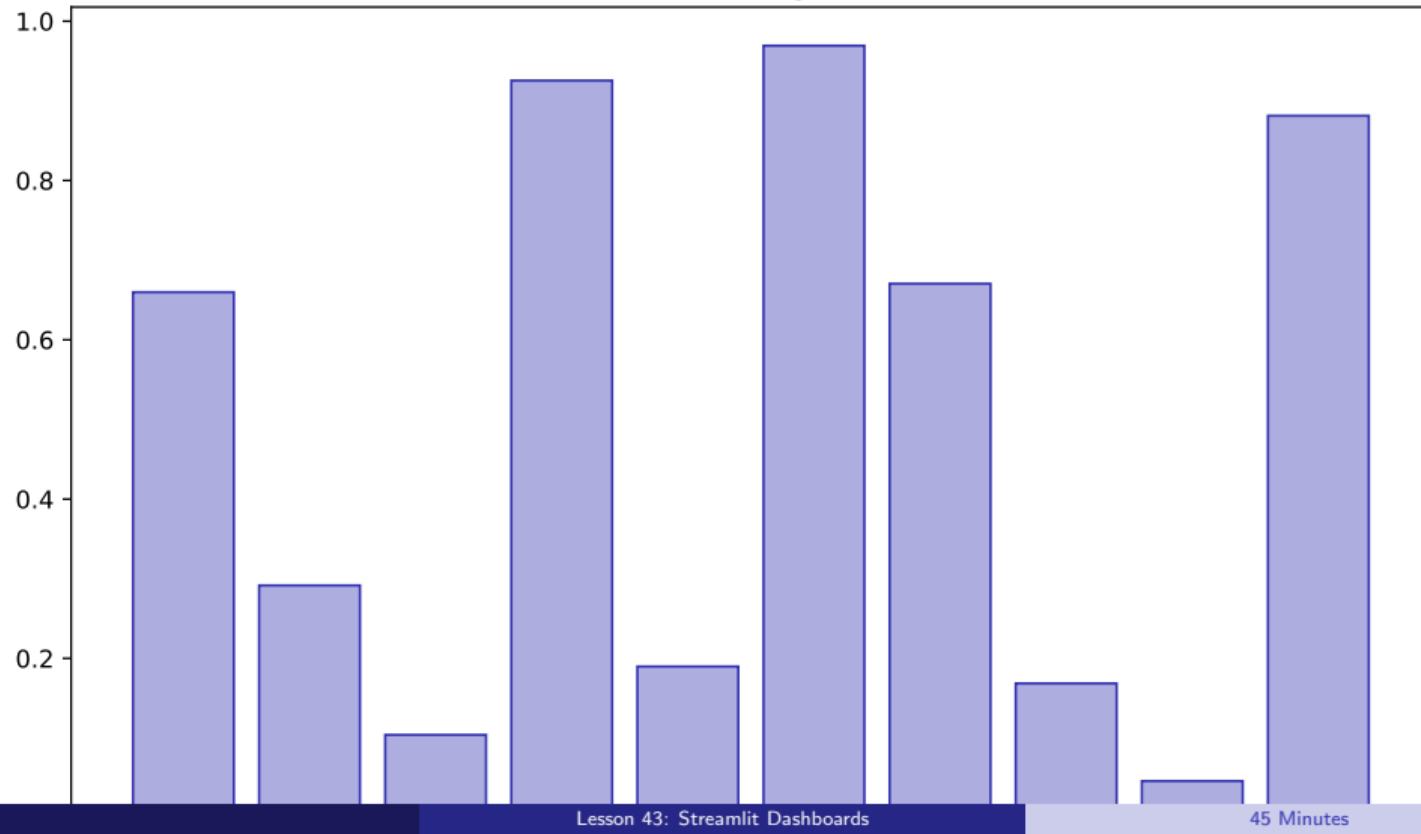
## Layouts



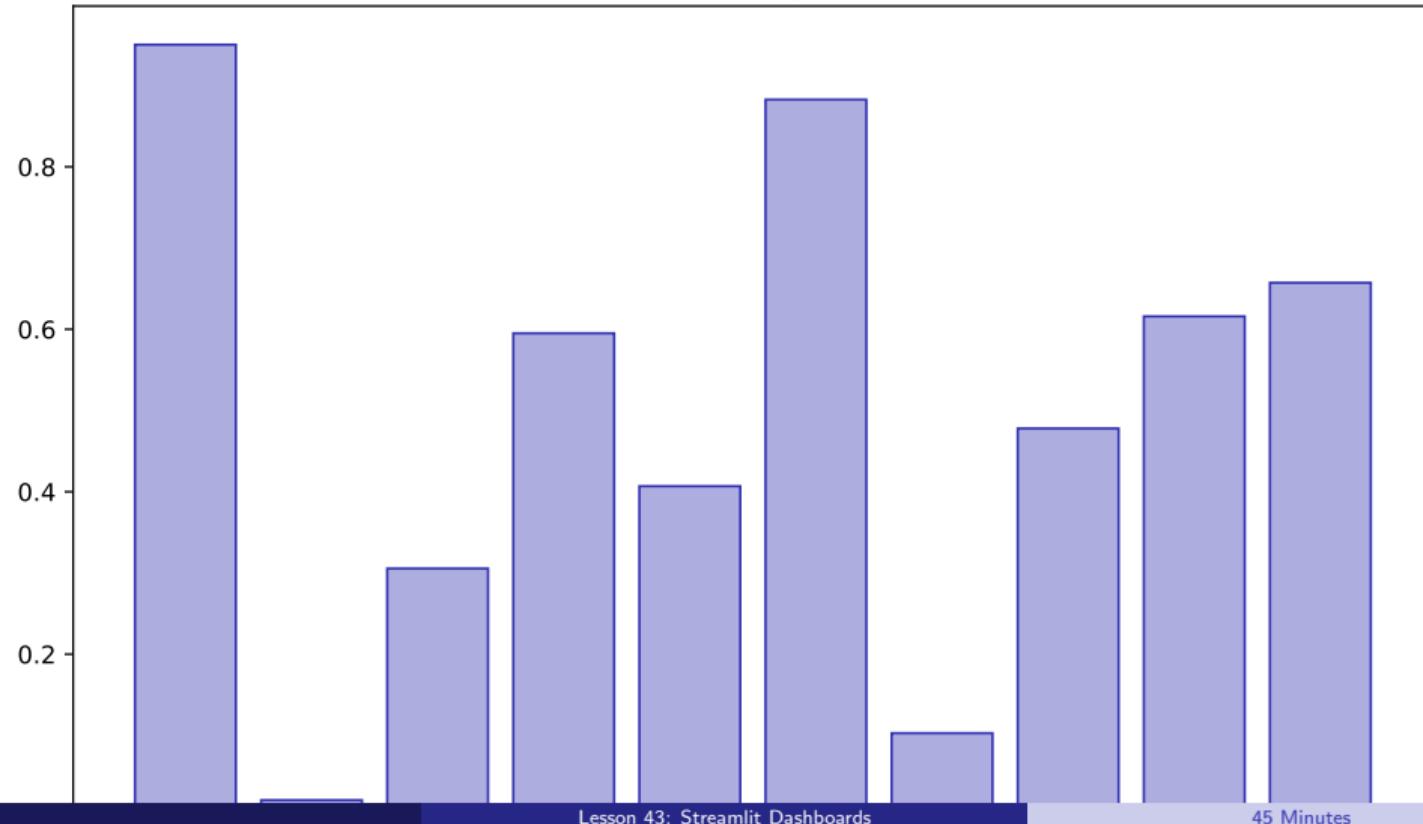
## Caching



## Charts Integration



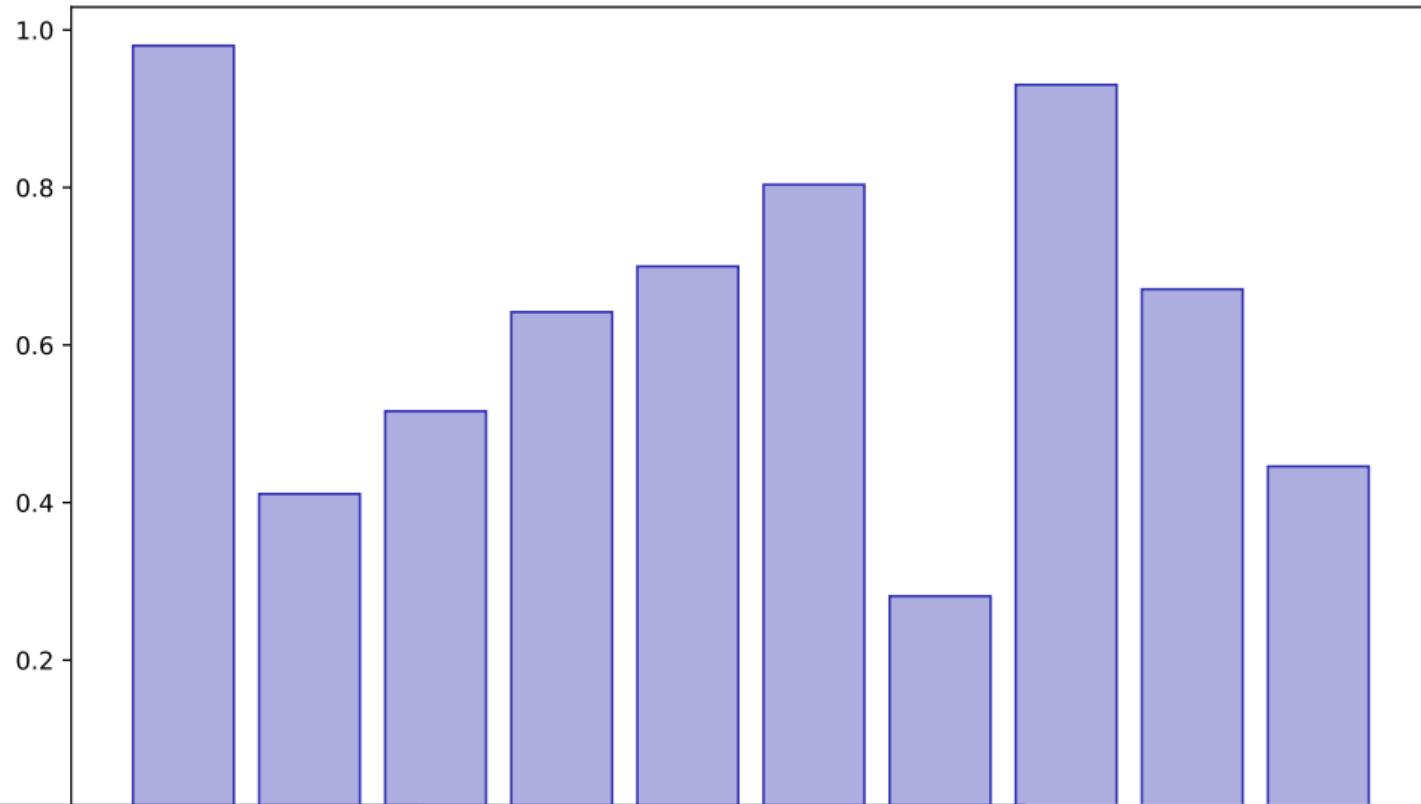
## Model Integration



## Deployment Prep



## Stock Dashboard



## Key Takeaways:

- Build interactive apps
- Add widgets and inputs
- Display predictions
- Create financial dashboards

Apply these skills in your final project

## Lesson 44: Cloud Deployment

Data Science with Python – BSc Course

45 Minutes

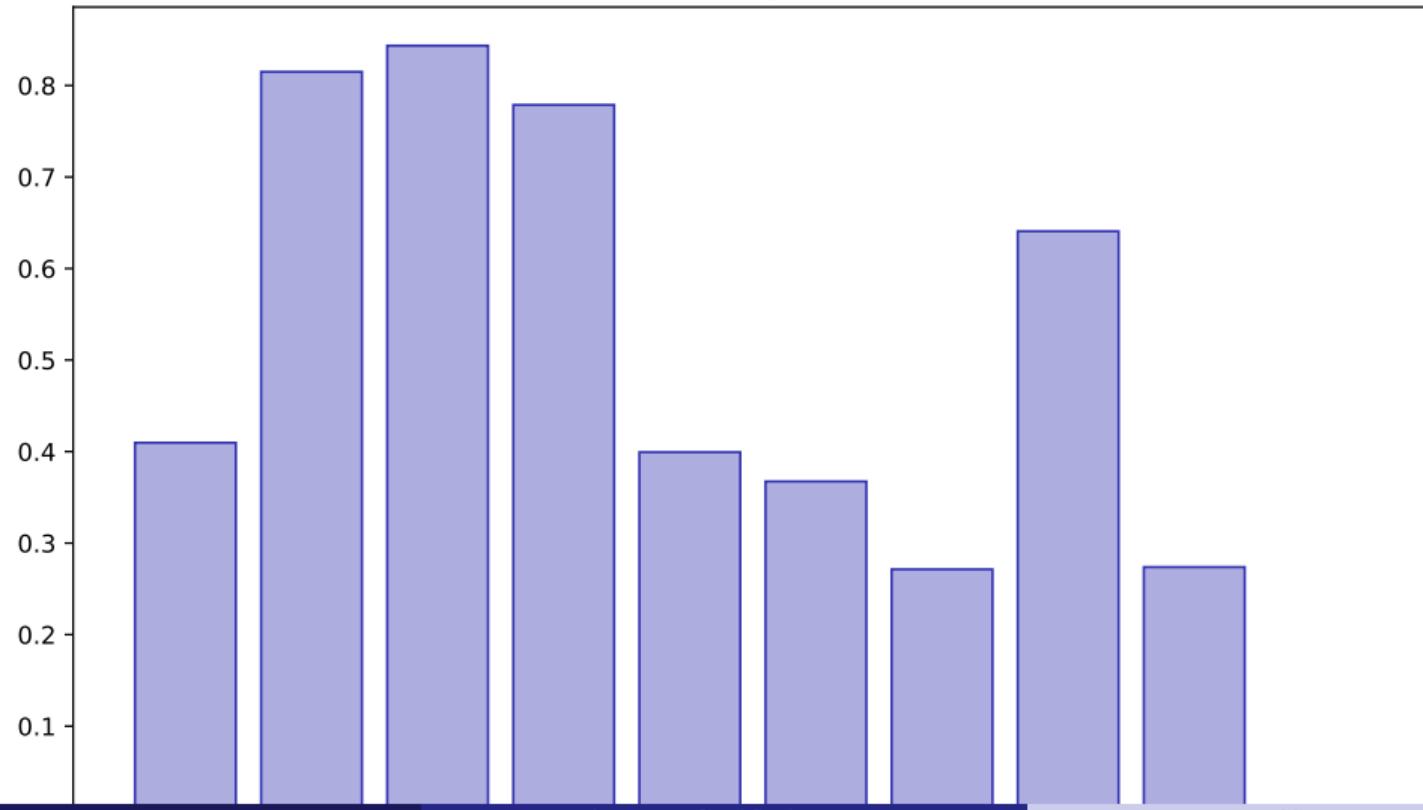
# Learning Objectives

**After this lesson, you will be able to:**

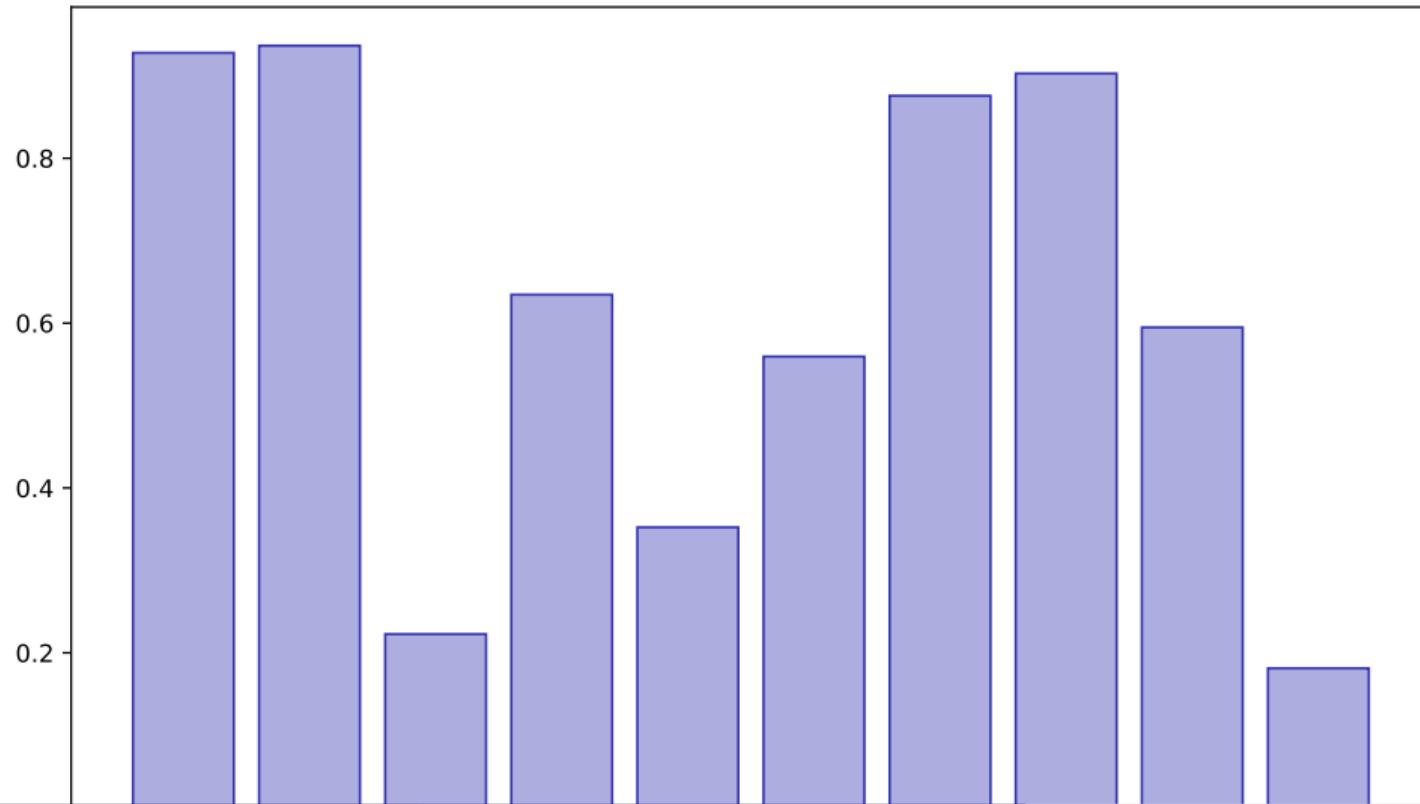
- Deploy to Streamlit Cloud
- Configure requirements
- Manage secrets
- Monitor applications

**Building towards your final project**

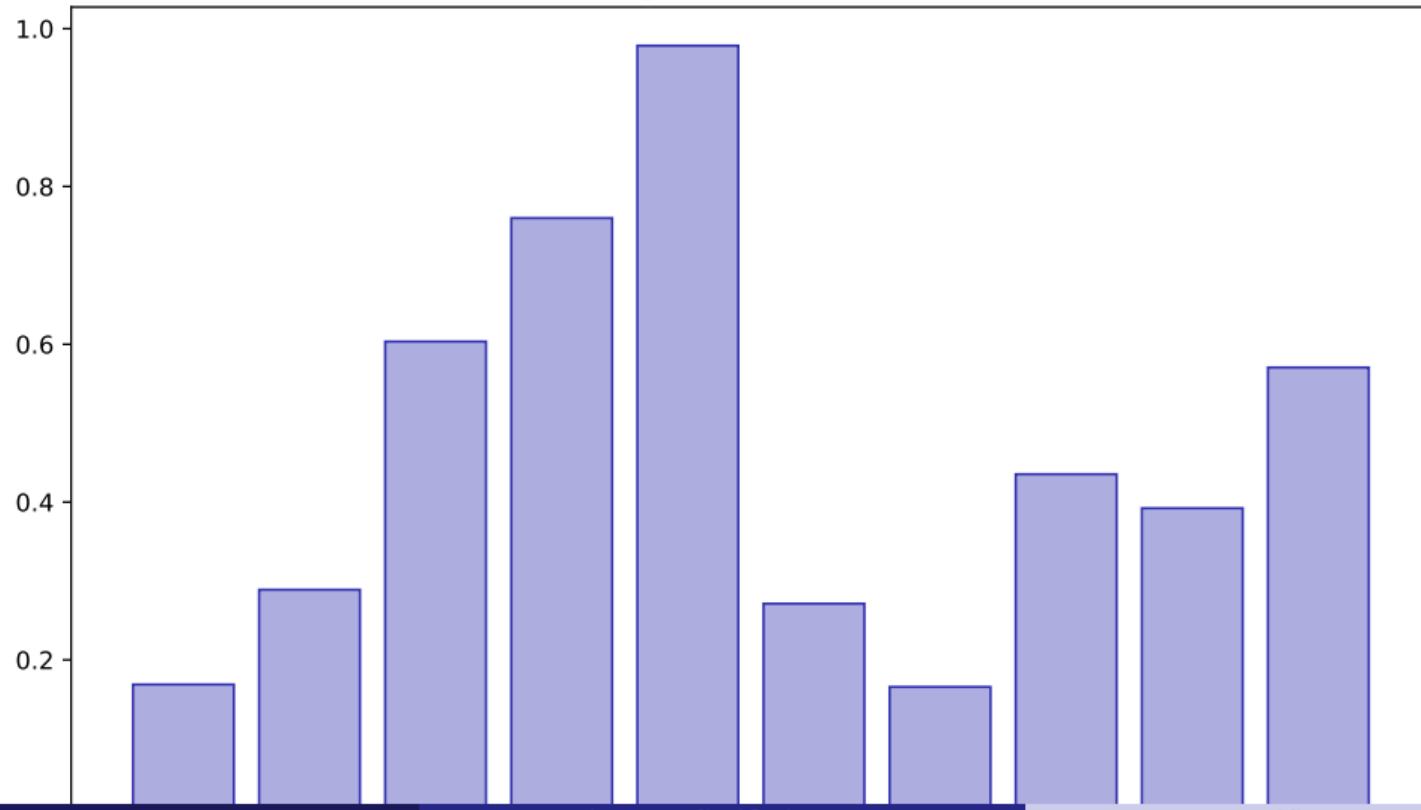
## Cloud Options



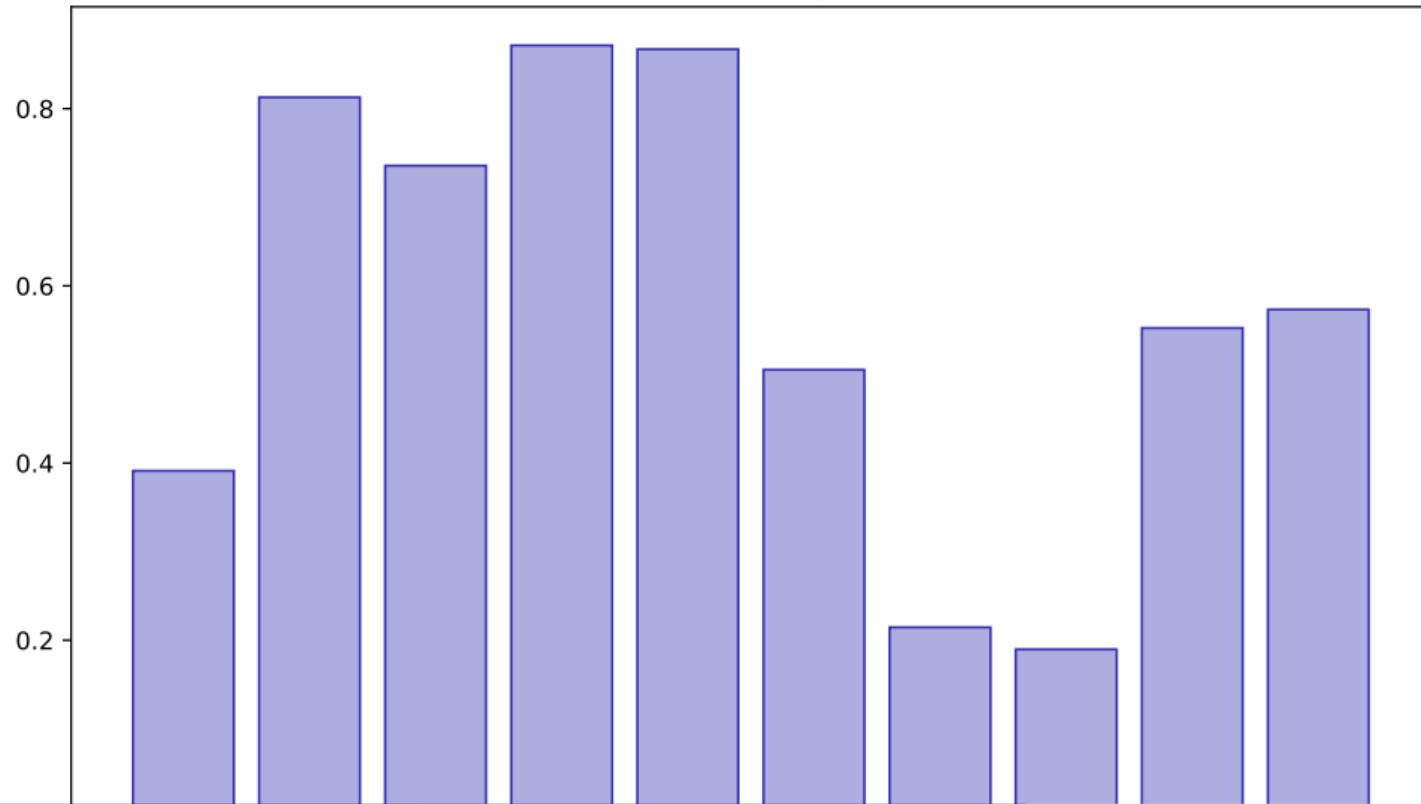
## Streamlit Cloud



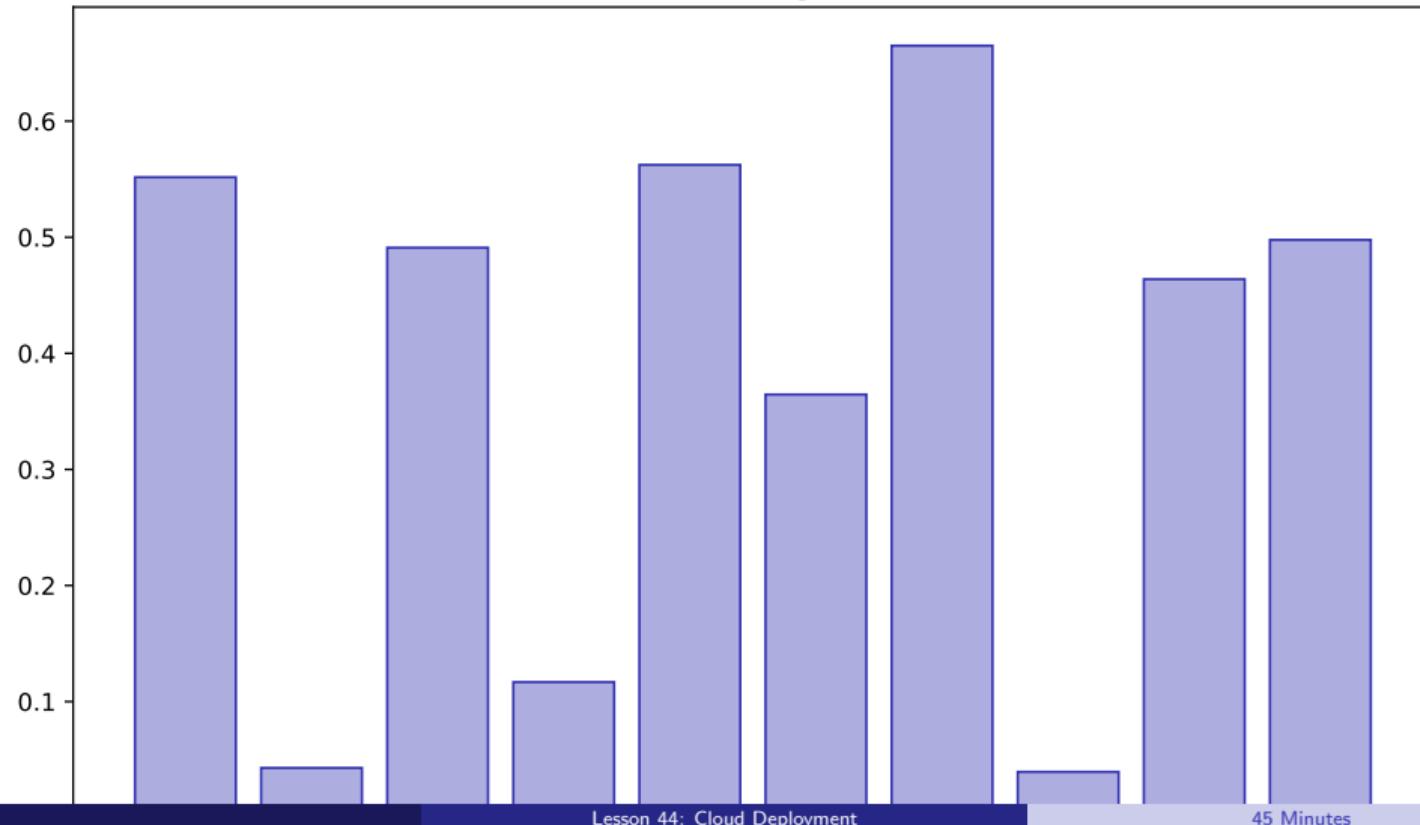
### Requirements Txt



### Secrets Management



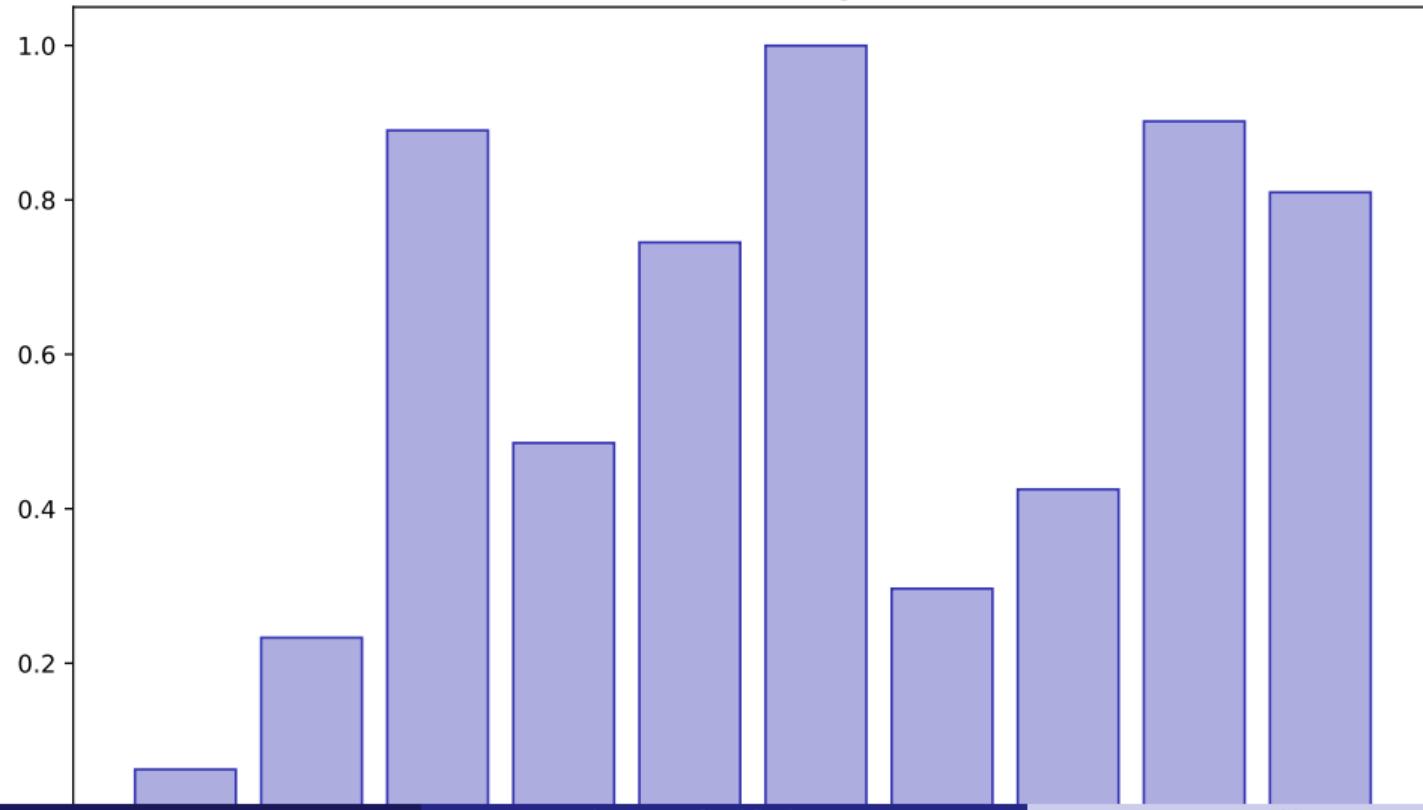
## Github Integration



## Deployment Workflow



## Monitoring



## Live Deployment



## Lesson Summary

### Key Takeaways:

- Deploy to Streamlit Cloud
- Configure requirements
- Manage secrets
- Monitor applications

Apply these skills in your final project

## Lesson 45: Project Work Session 1

Data Science with Python – BSc Course

45 Minutes

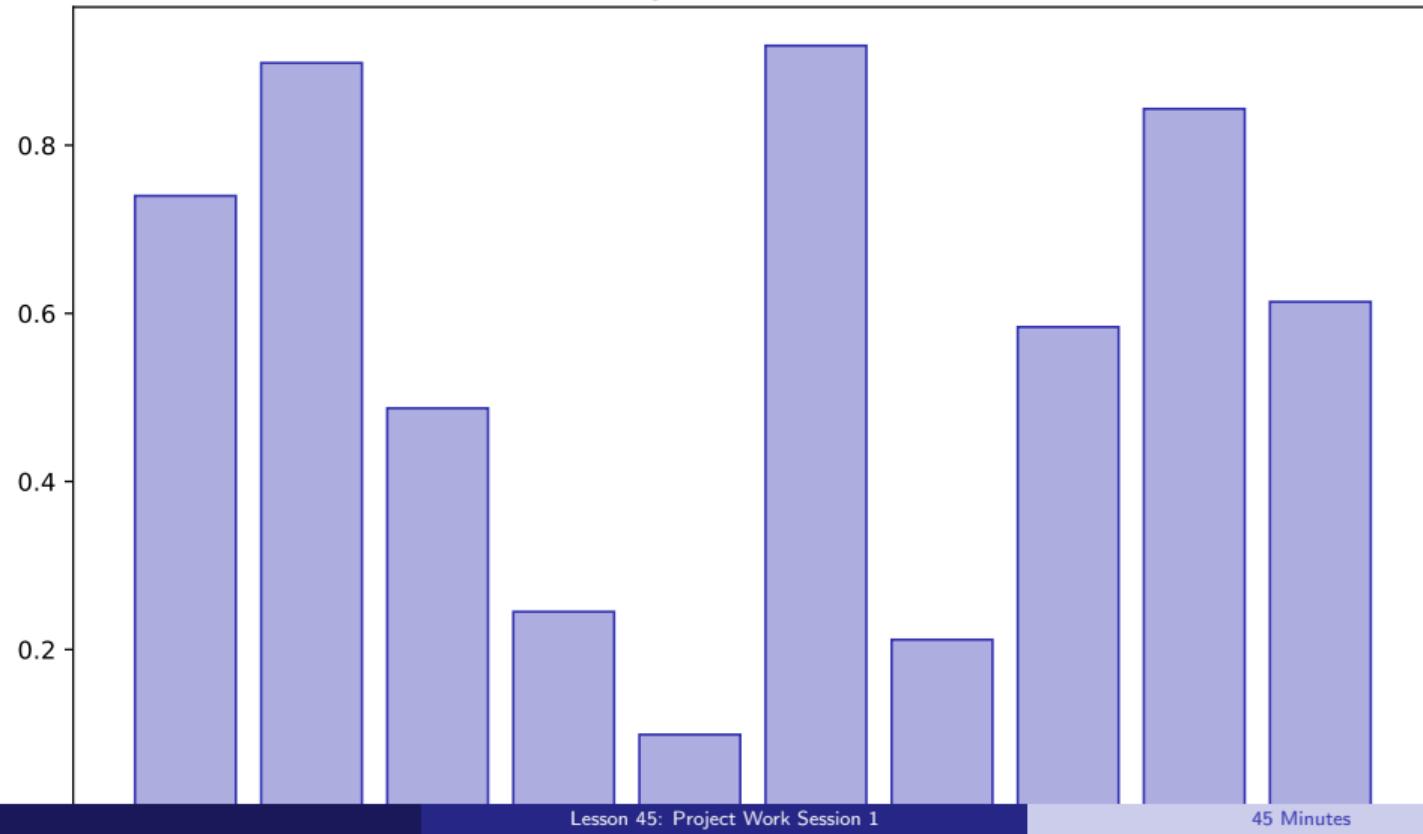
# Learning Objectives

**After this lesson, you will be able to:**

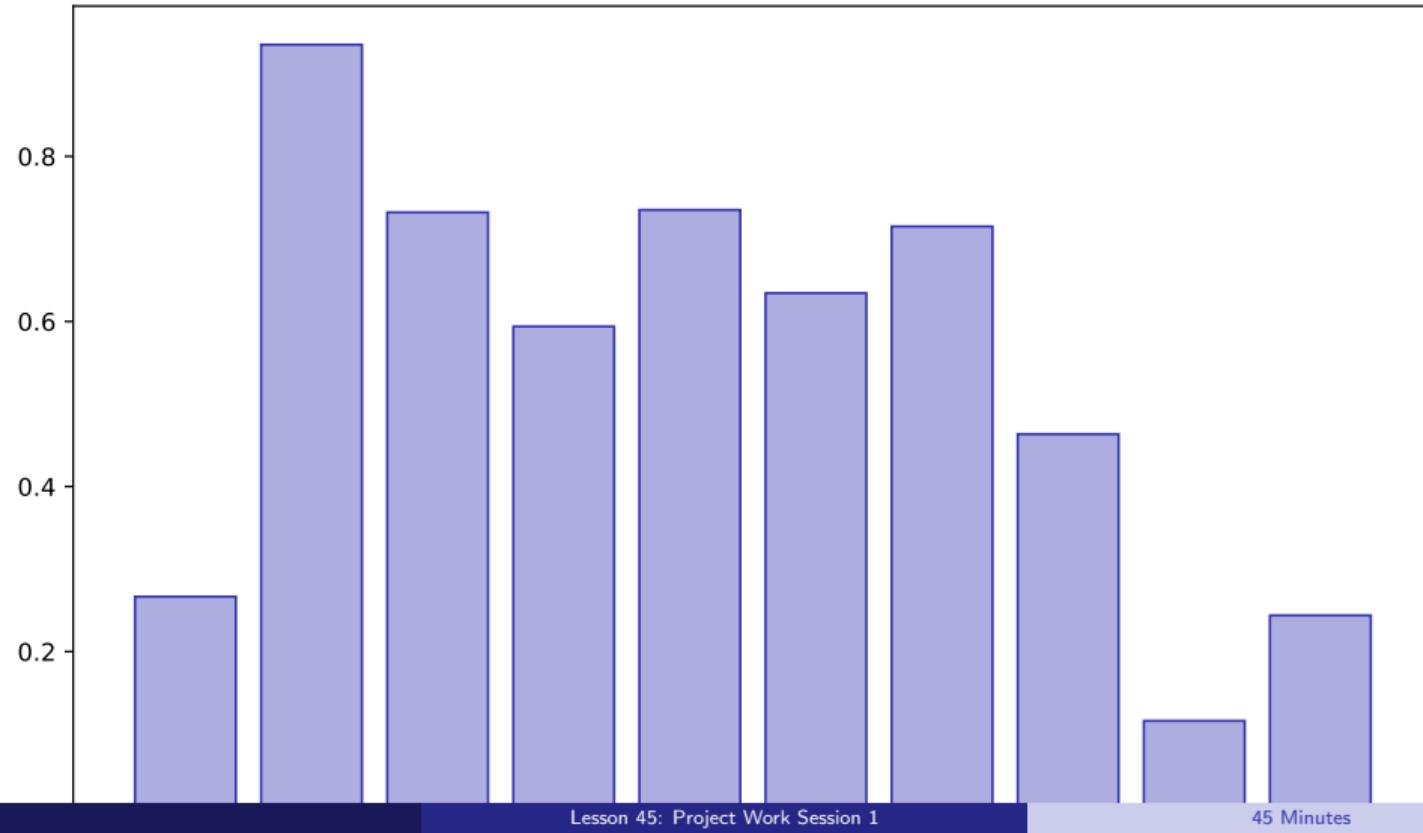
- Review project requirements
- Plan implementation
- Start coding
- Get feedback

**Building towards your final project**

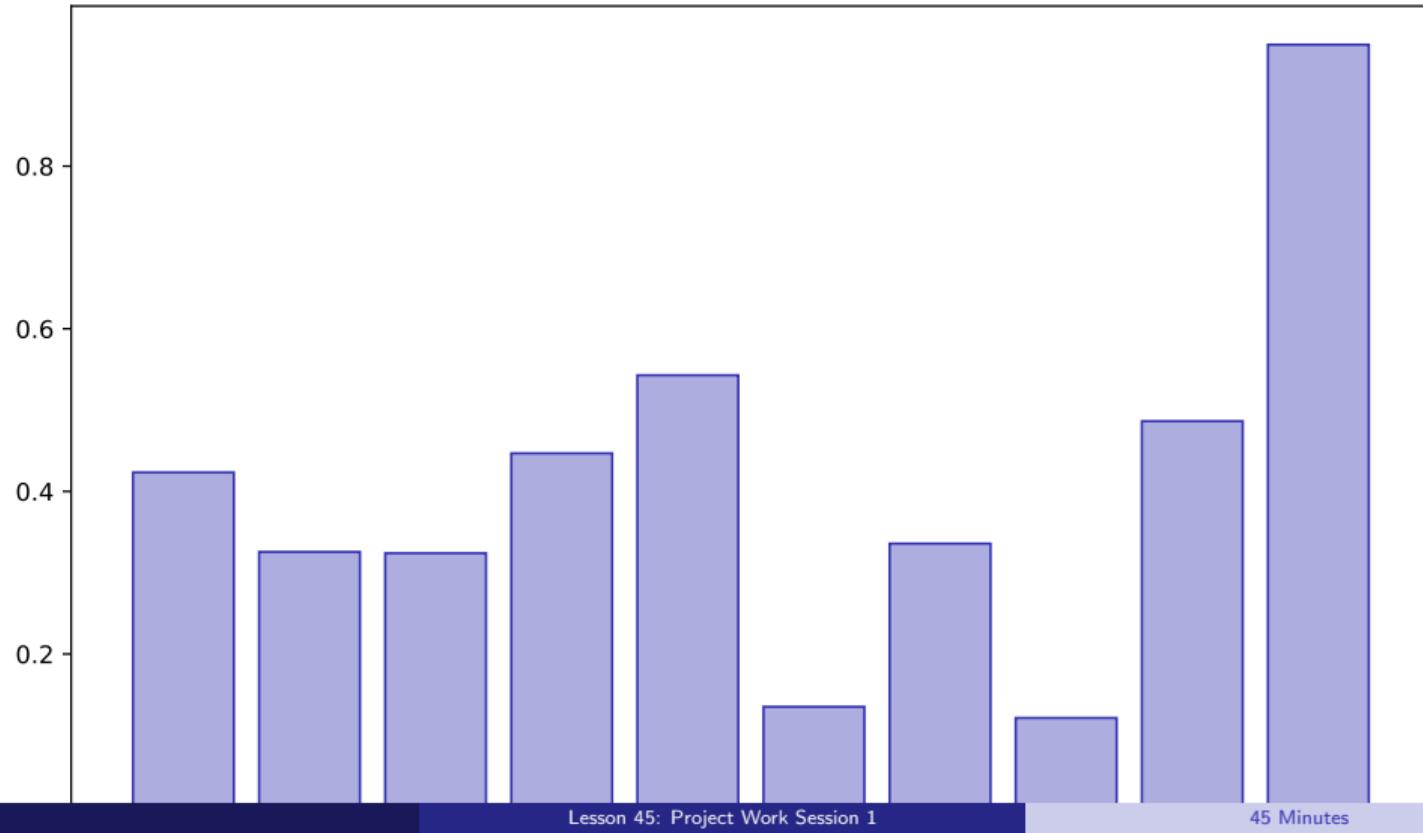
## Project Overview



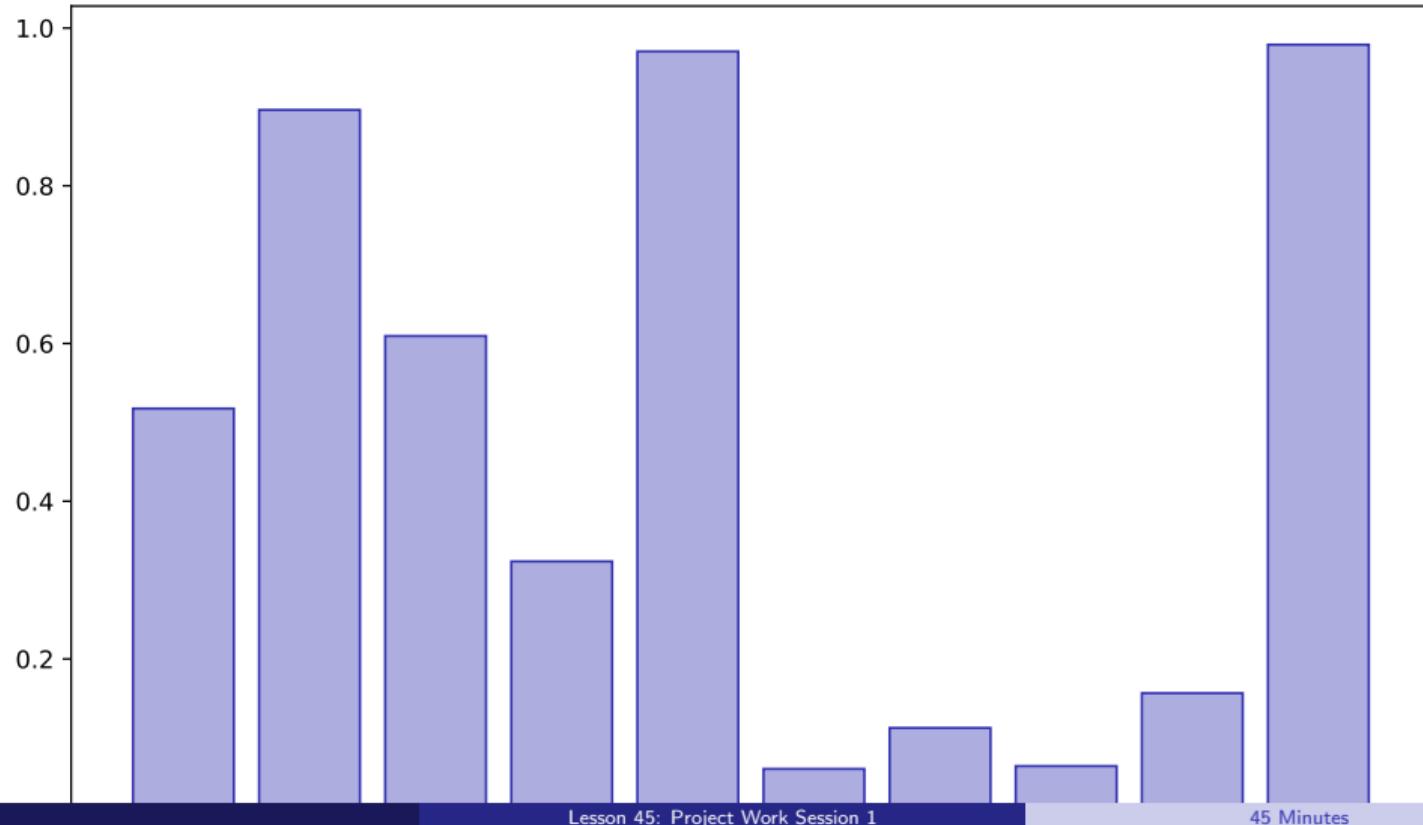
### Rubric Review



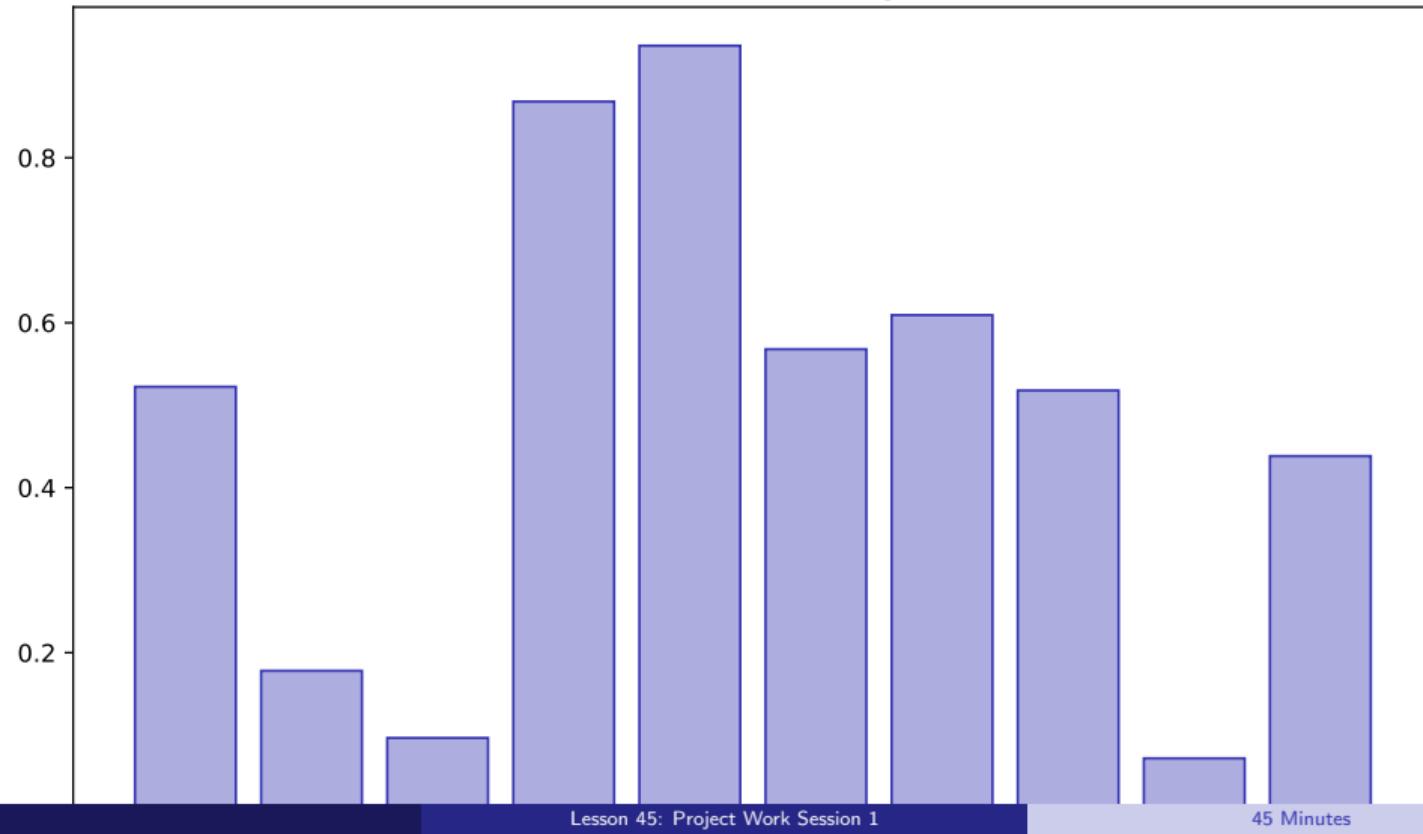
### Topic Selection



## Data Requirements

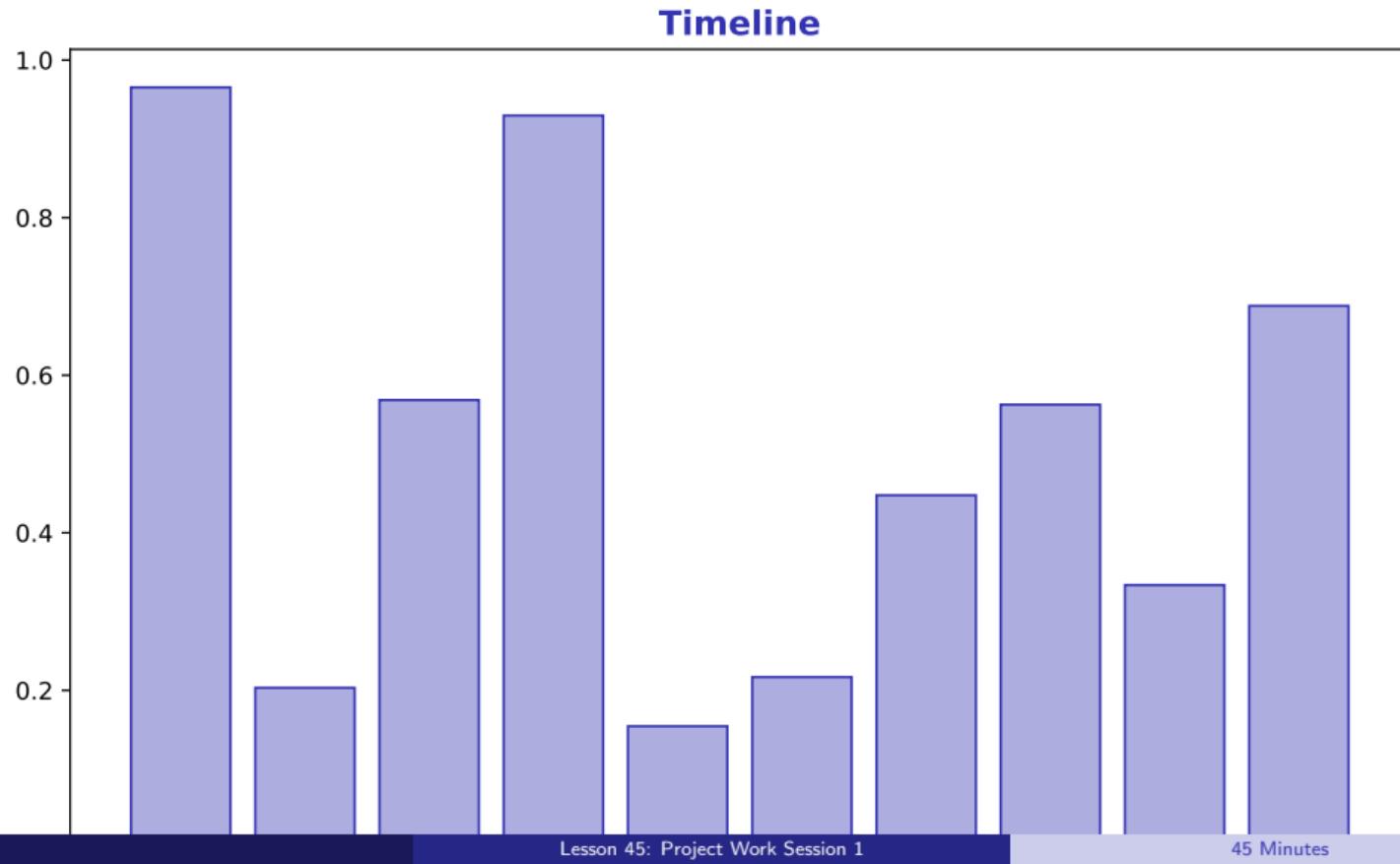


## Model Planning



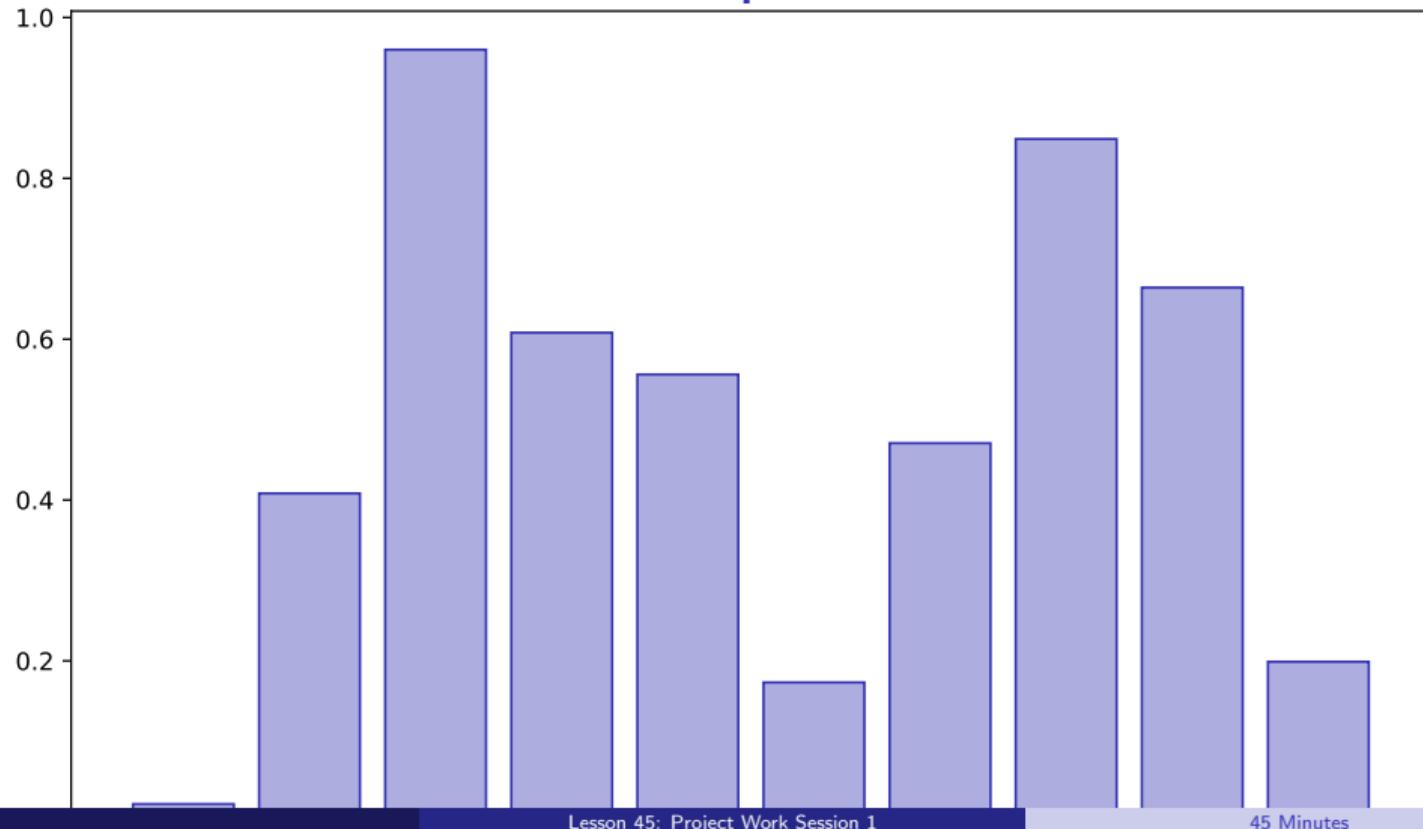
## Deployment Planning





## Checkpoint 1

### Checkpoint 1



## Lesson Summary

### Key Takeaways:

- Review project requirements
- Plan implementation
- Start coding
- Get feedback

Apply these skills in your final project

## Lesson 46: Project Work Session 2

Data Science with Python – BSc Course

45 Minutes

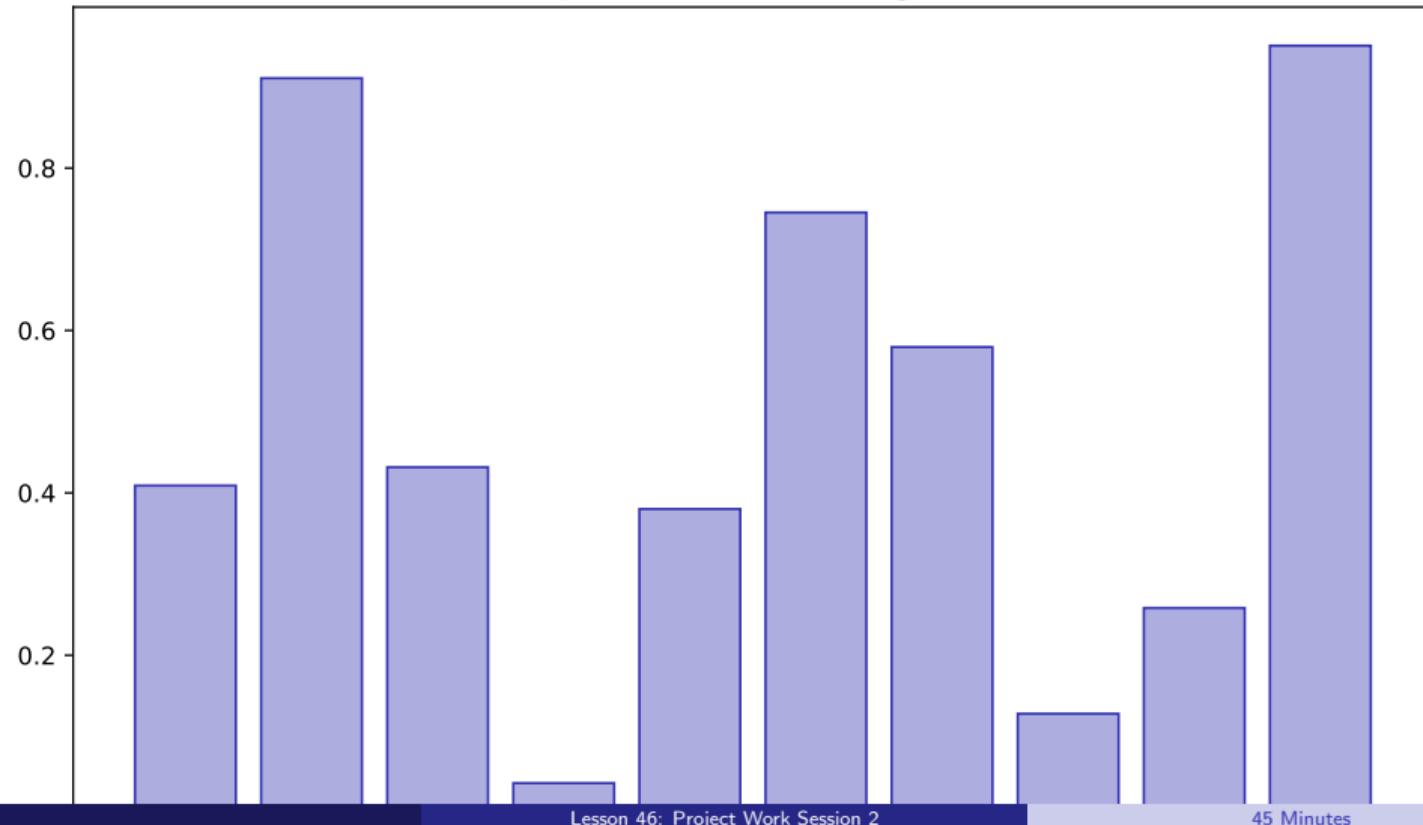
## Learning Objectives

**After this lesson, you will be able to:**

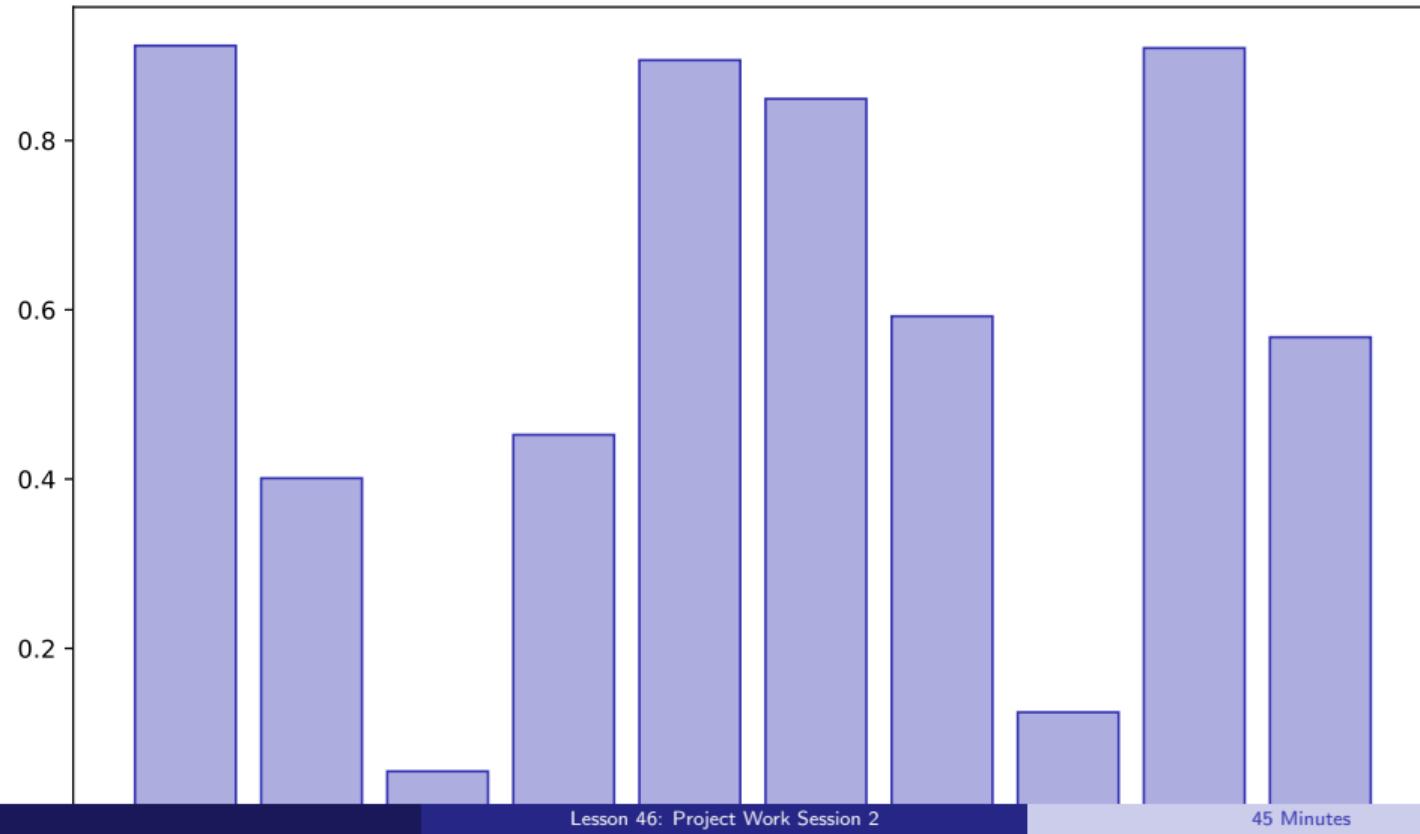
- Continue implementation
- Prepare presentation
- Practice demo
- Finalize code

**Building towards your final project**

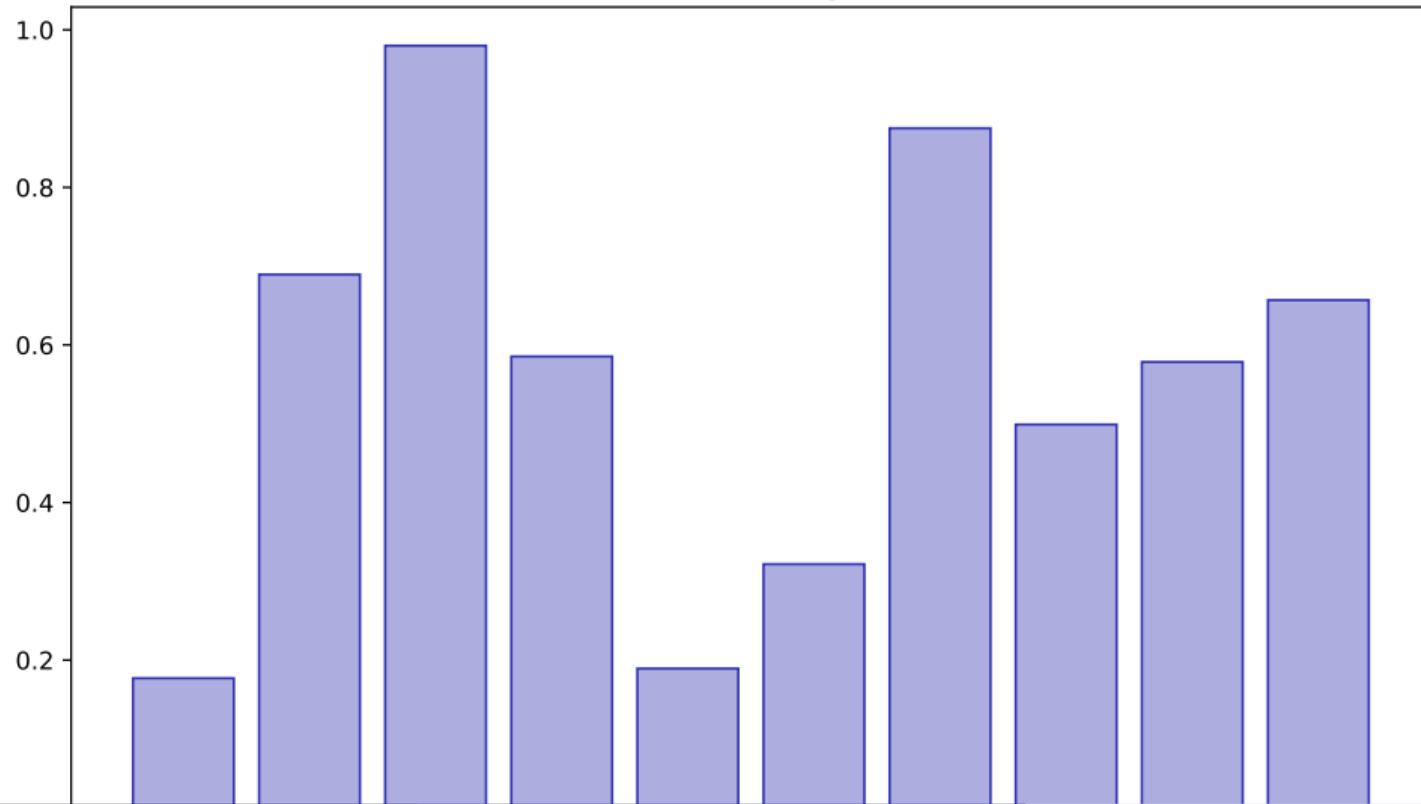
## Implementation Progress



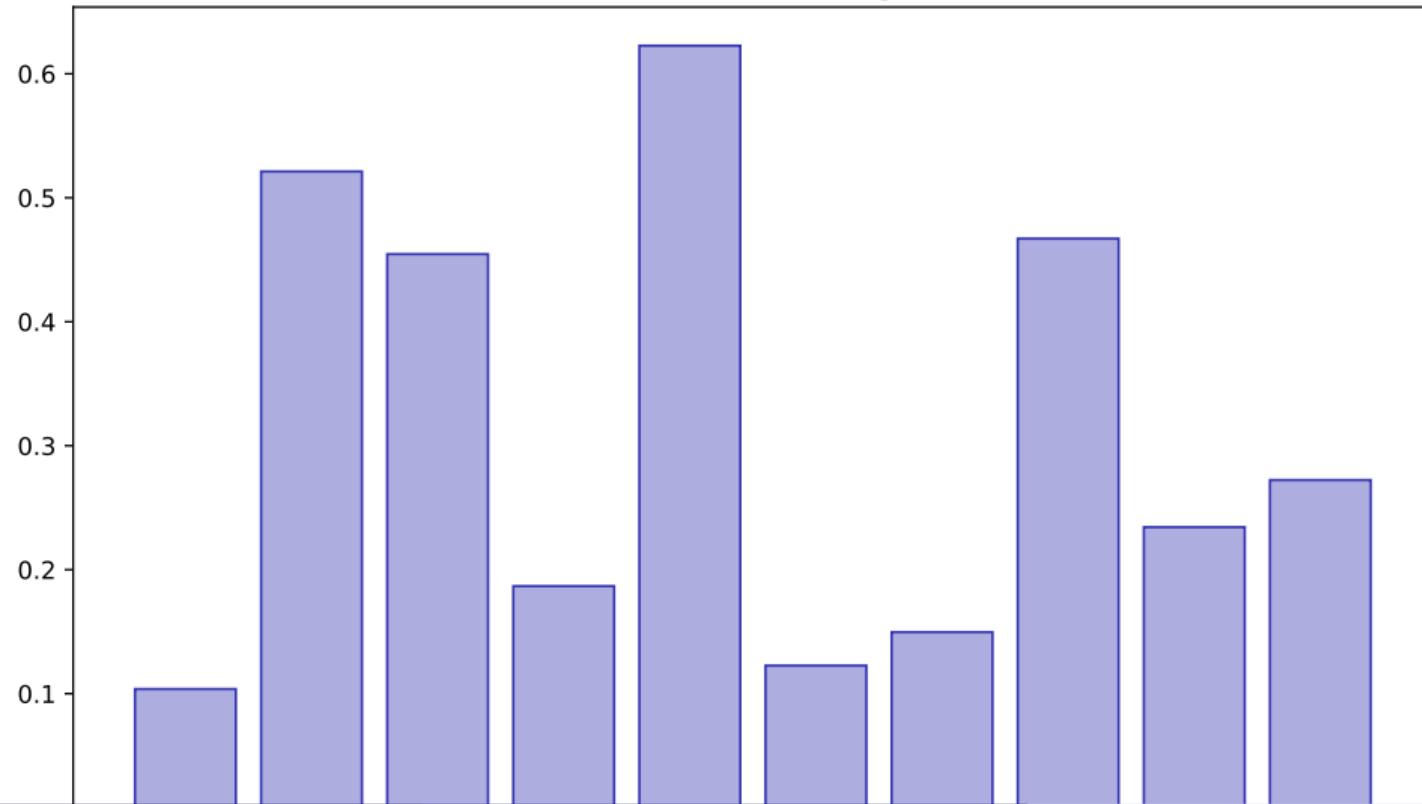
## Presentation Structure



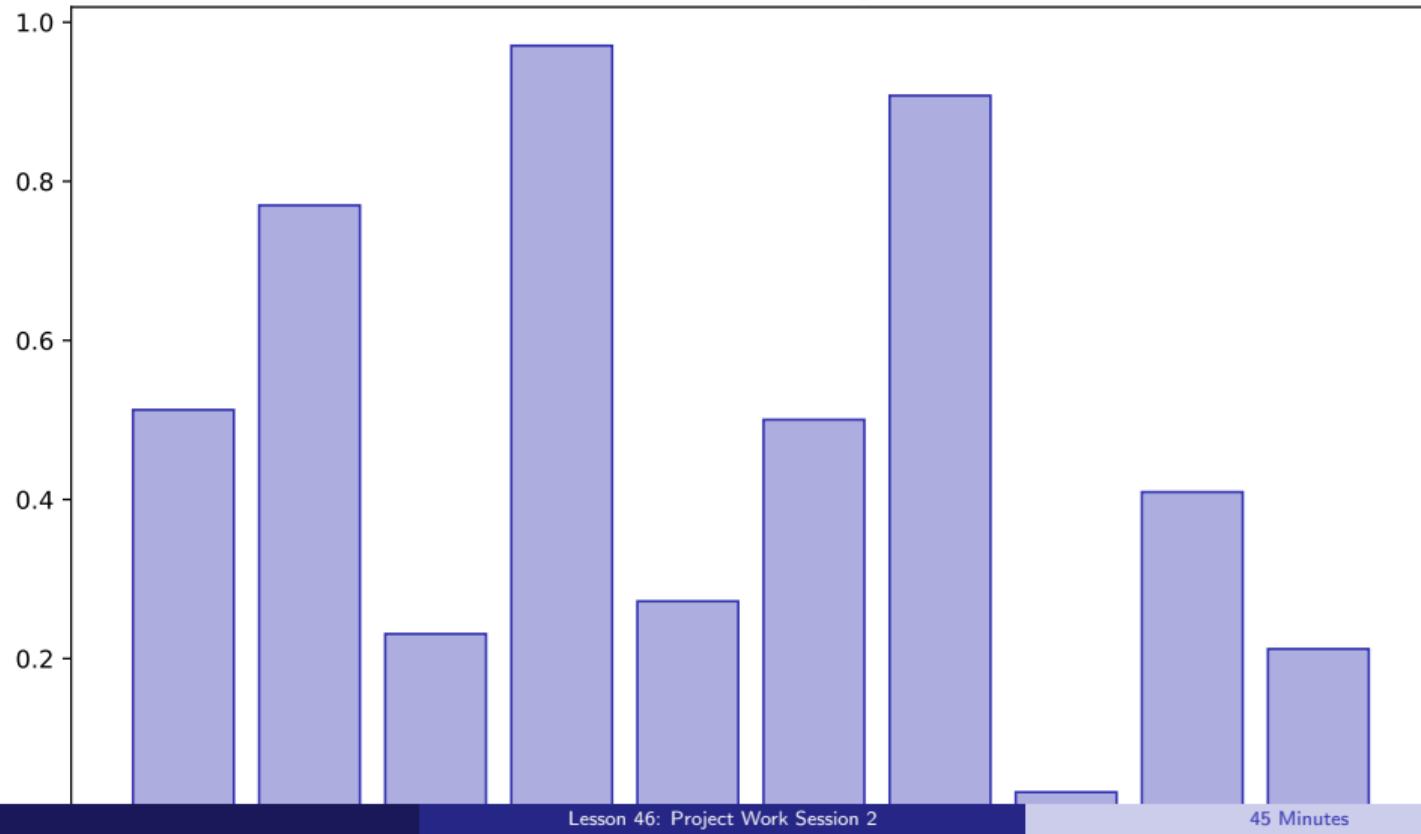
## Slide Design



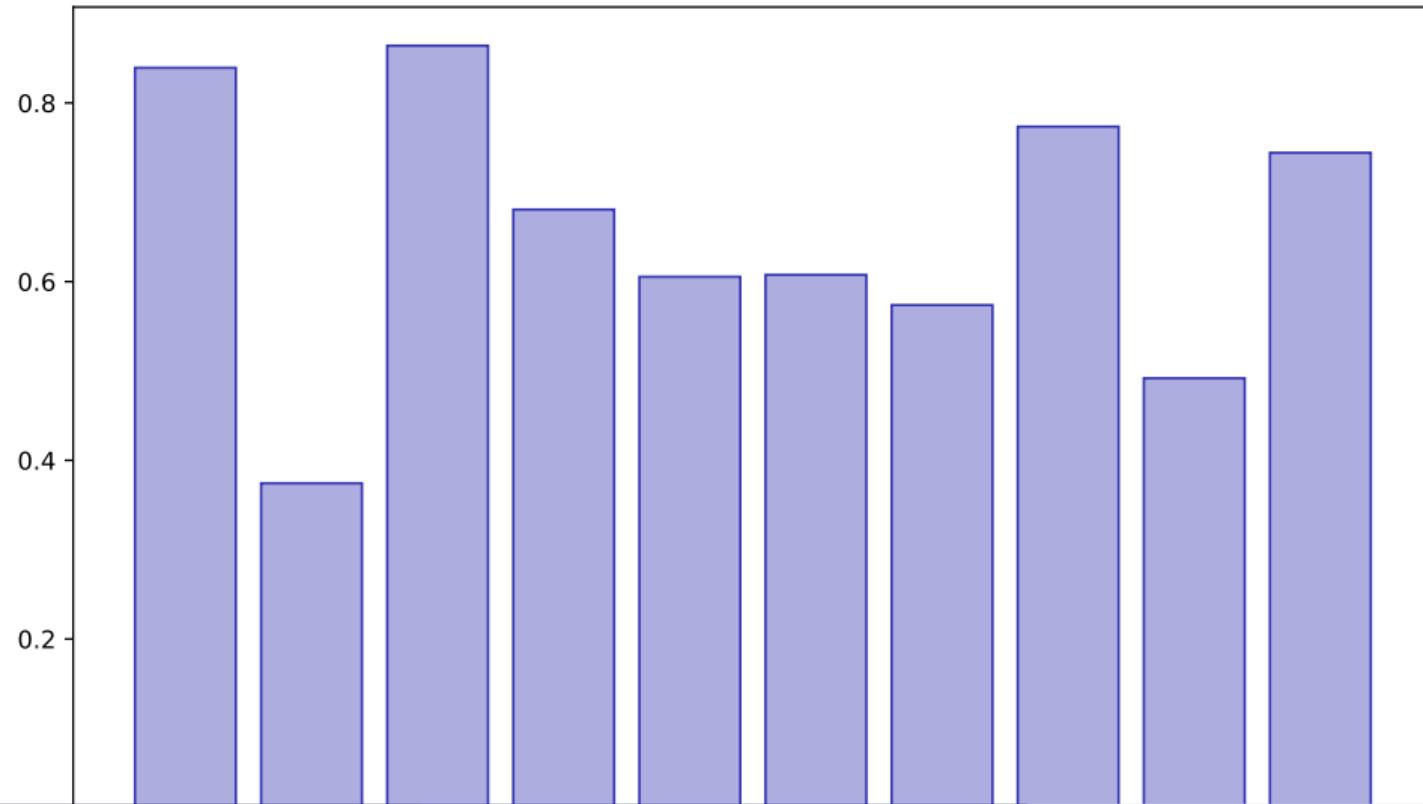
## Demo Planning



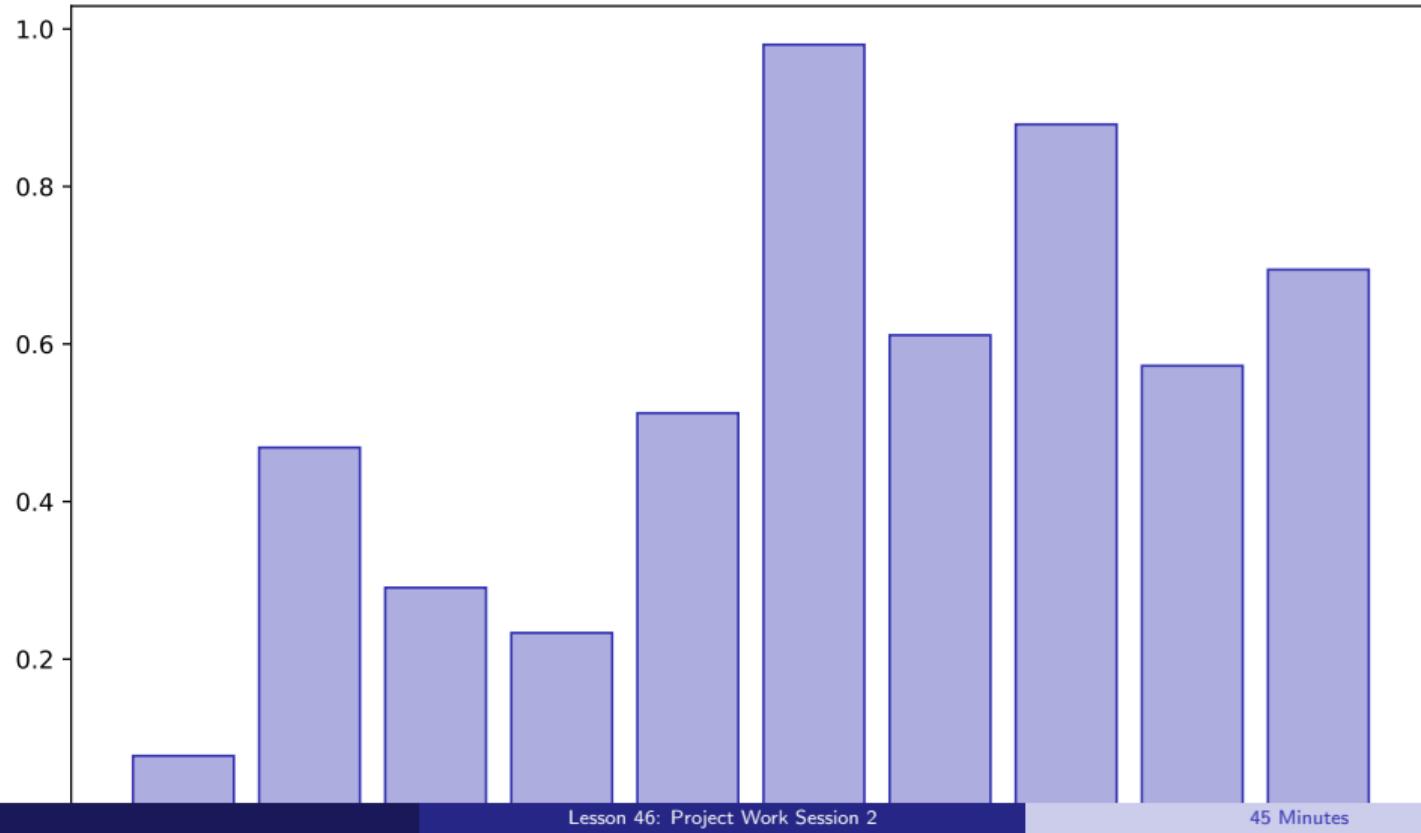
## Code Cleanup



## Documentation

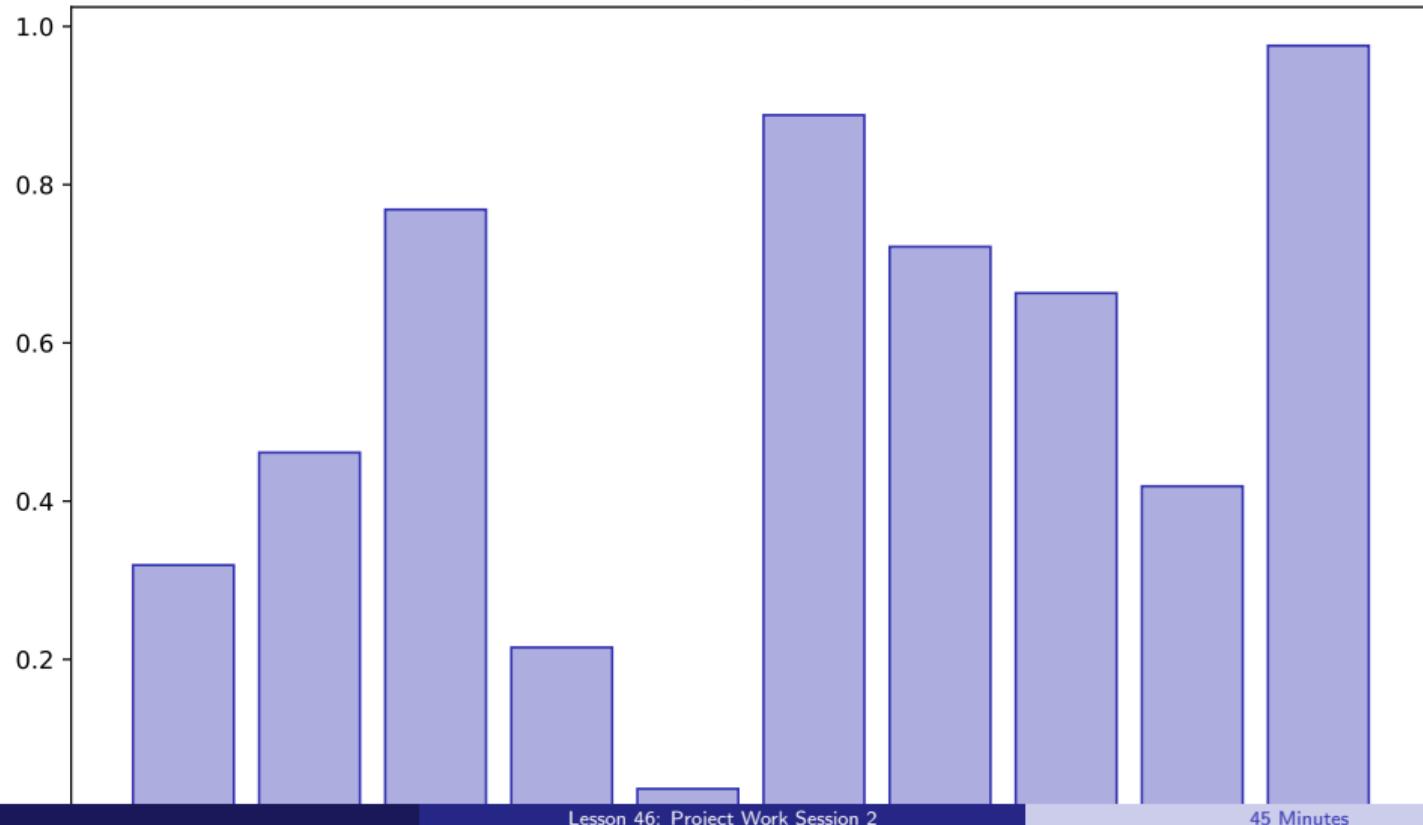


### Peer Feedback



## Checkpoint 2

### Checkpoint 2



### Key Takeaways:

- Continue implementation
- Prepare presentation
- Practice demo
- Finalize code

Apply these skills in your final project

## Lesson 47: ML Ethics in Finance

Data Science with Python – BSc Course

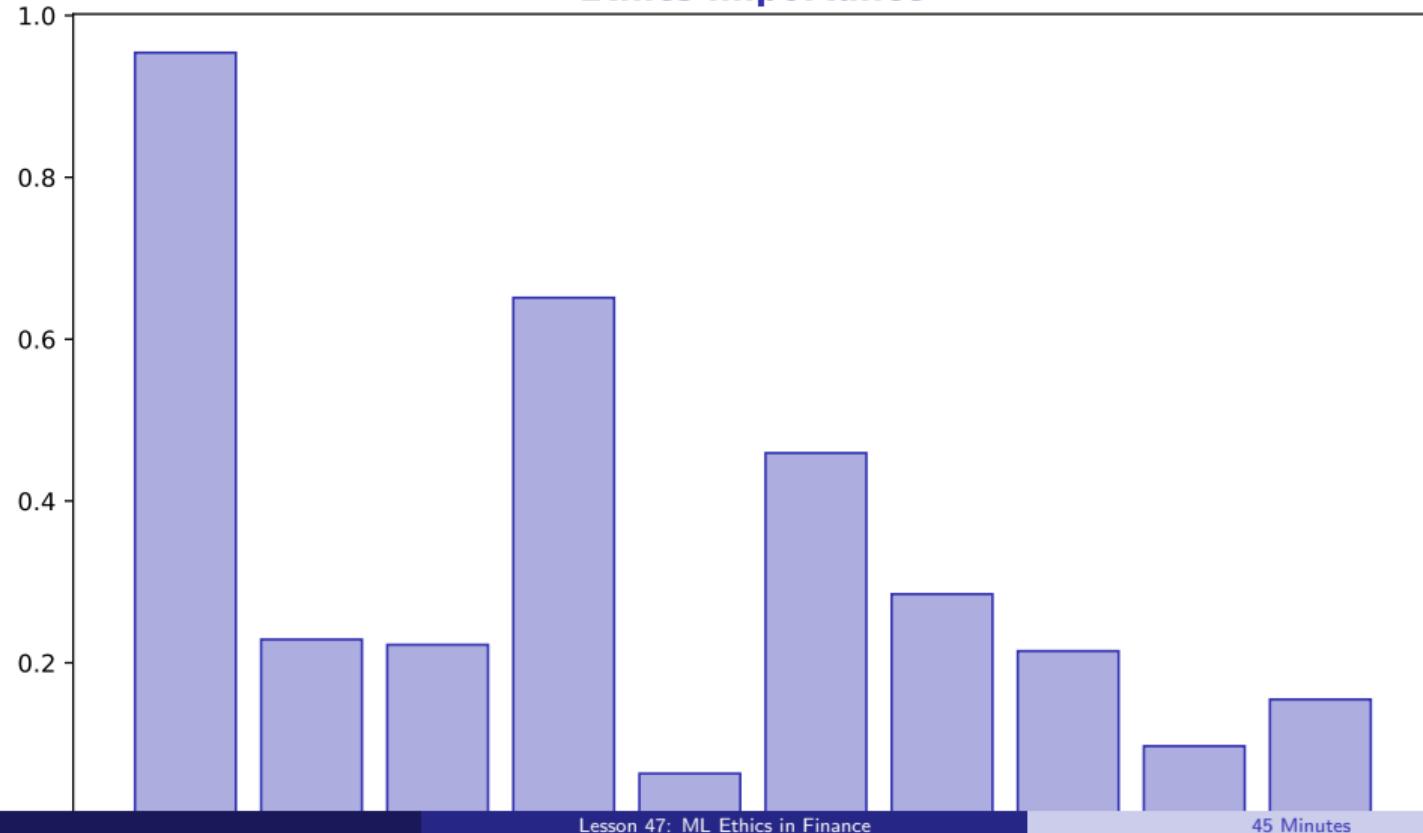
45 Minutes

**After this lesson, you will be able to:**

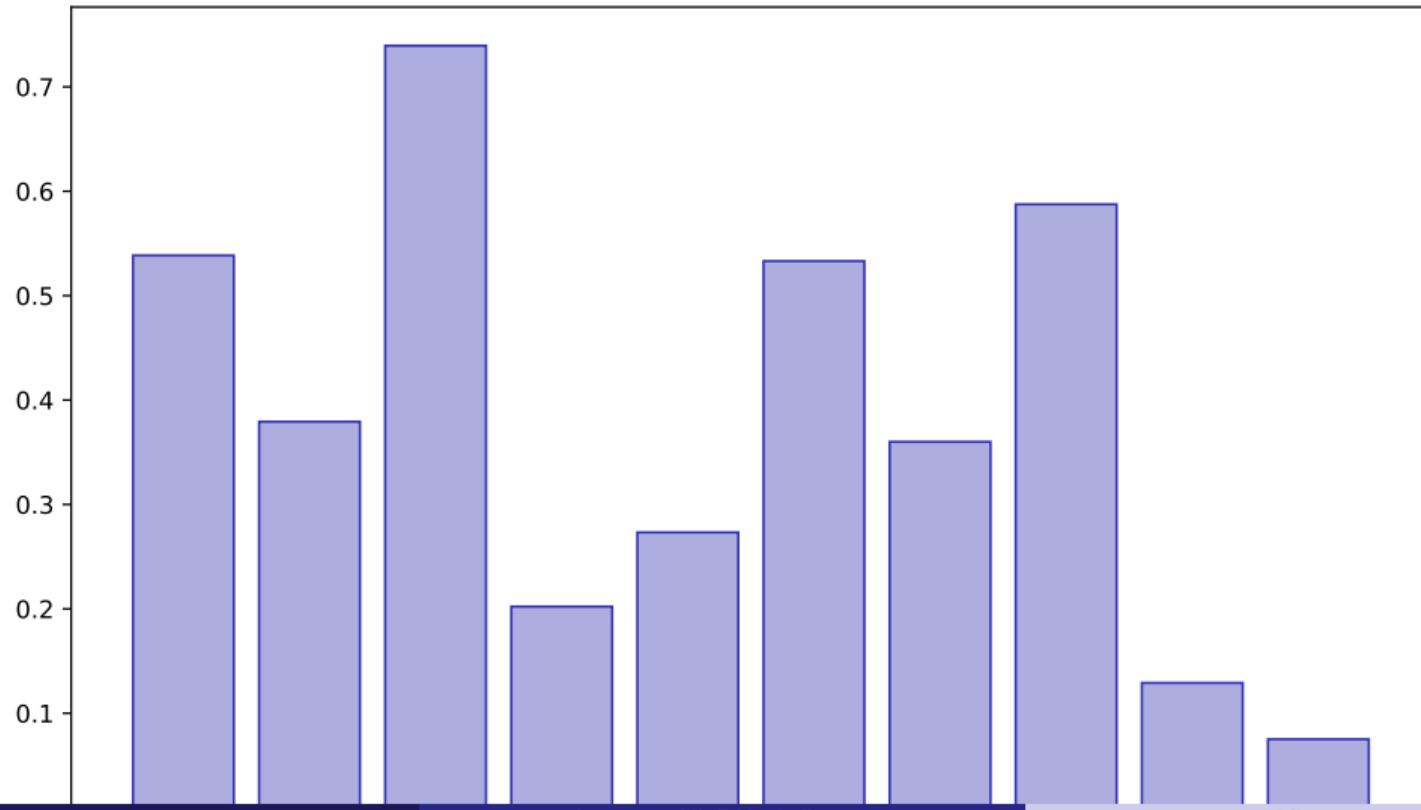
- Identify bias in models
- Ensure explainability
- Consider fairness
- Follow regulations

**Building towards your final project**

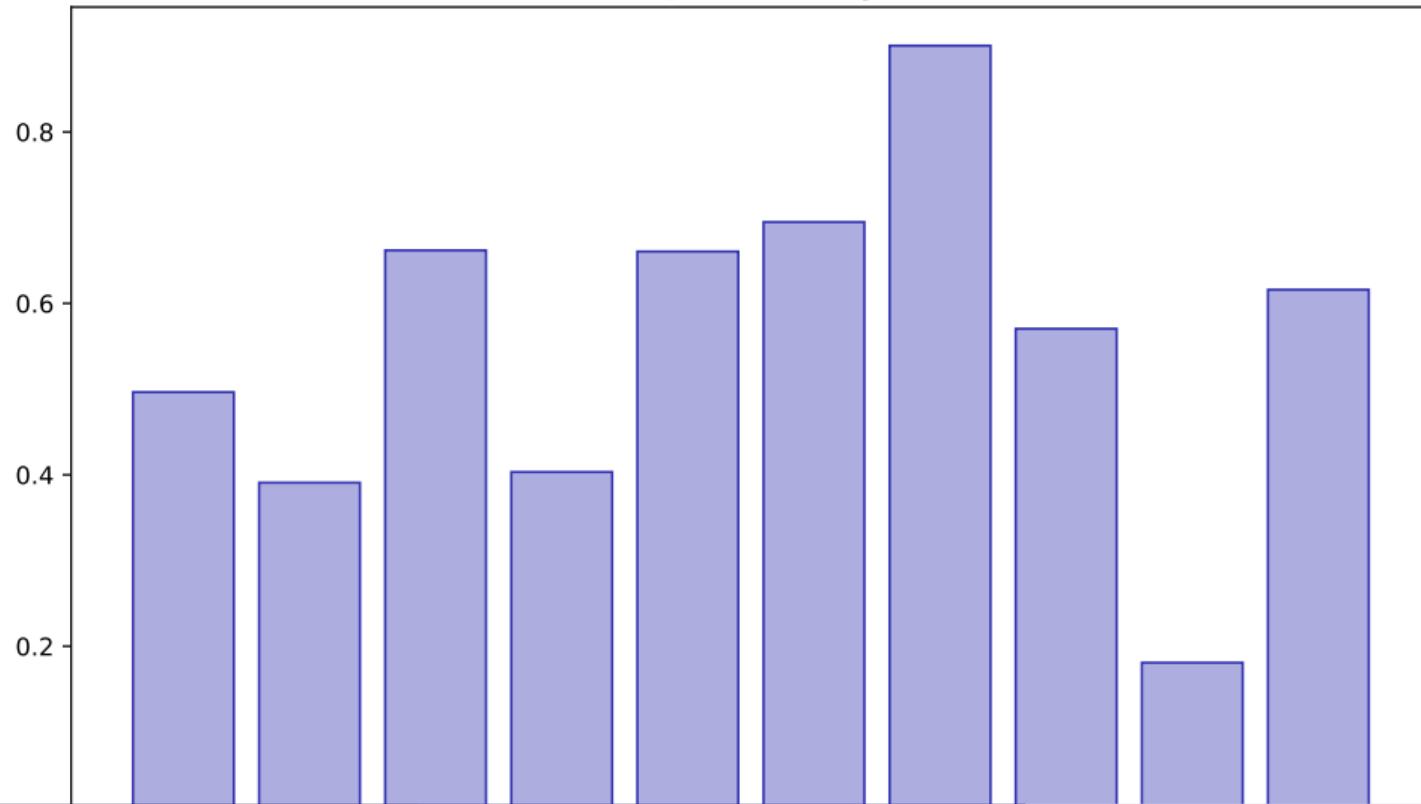
## Ethics Importance



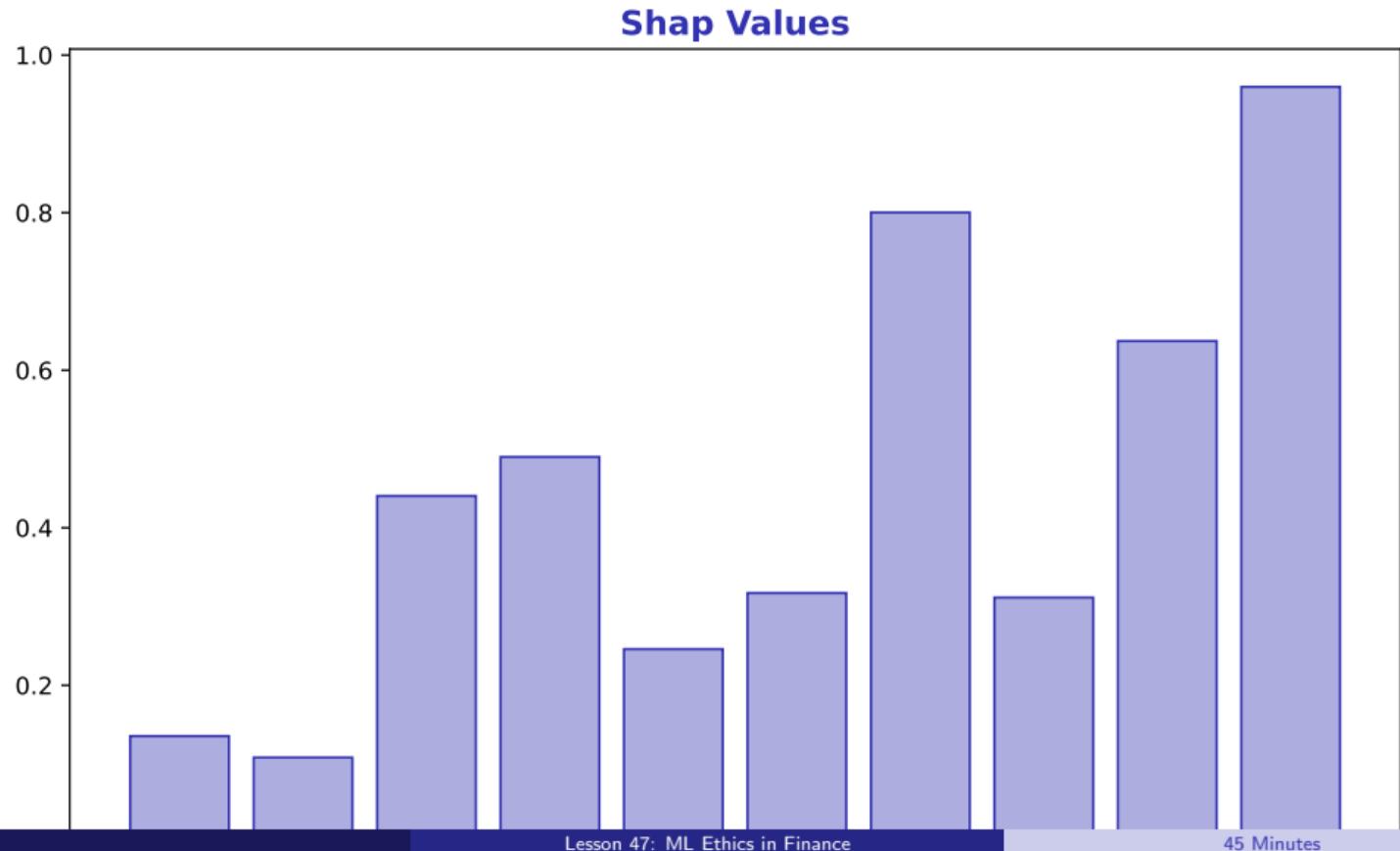
## Bias Sources



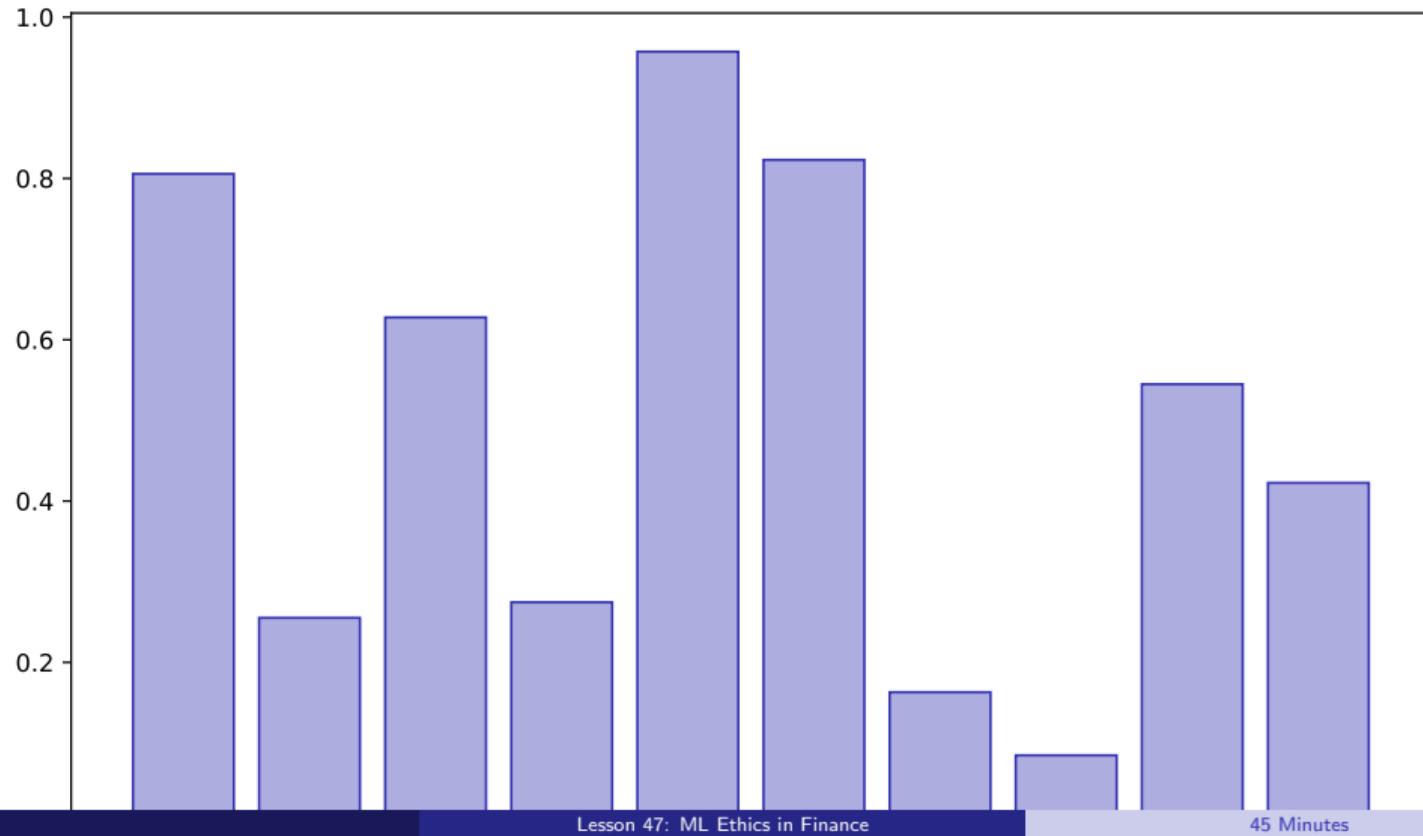
## Explainability



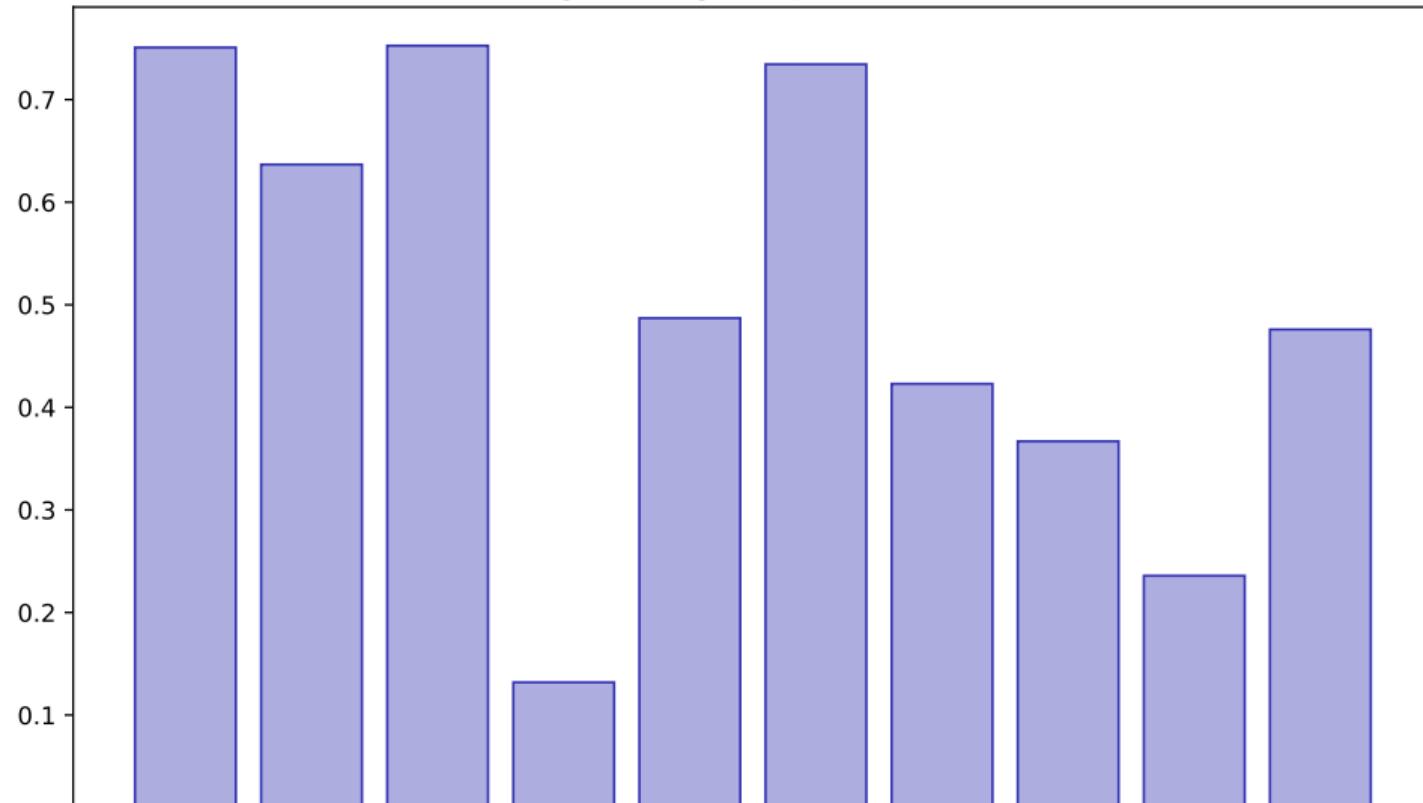
# Shap Values



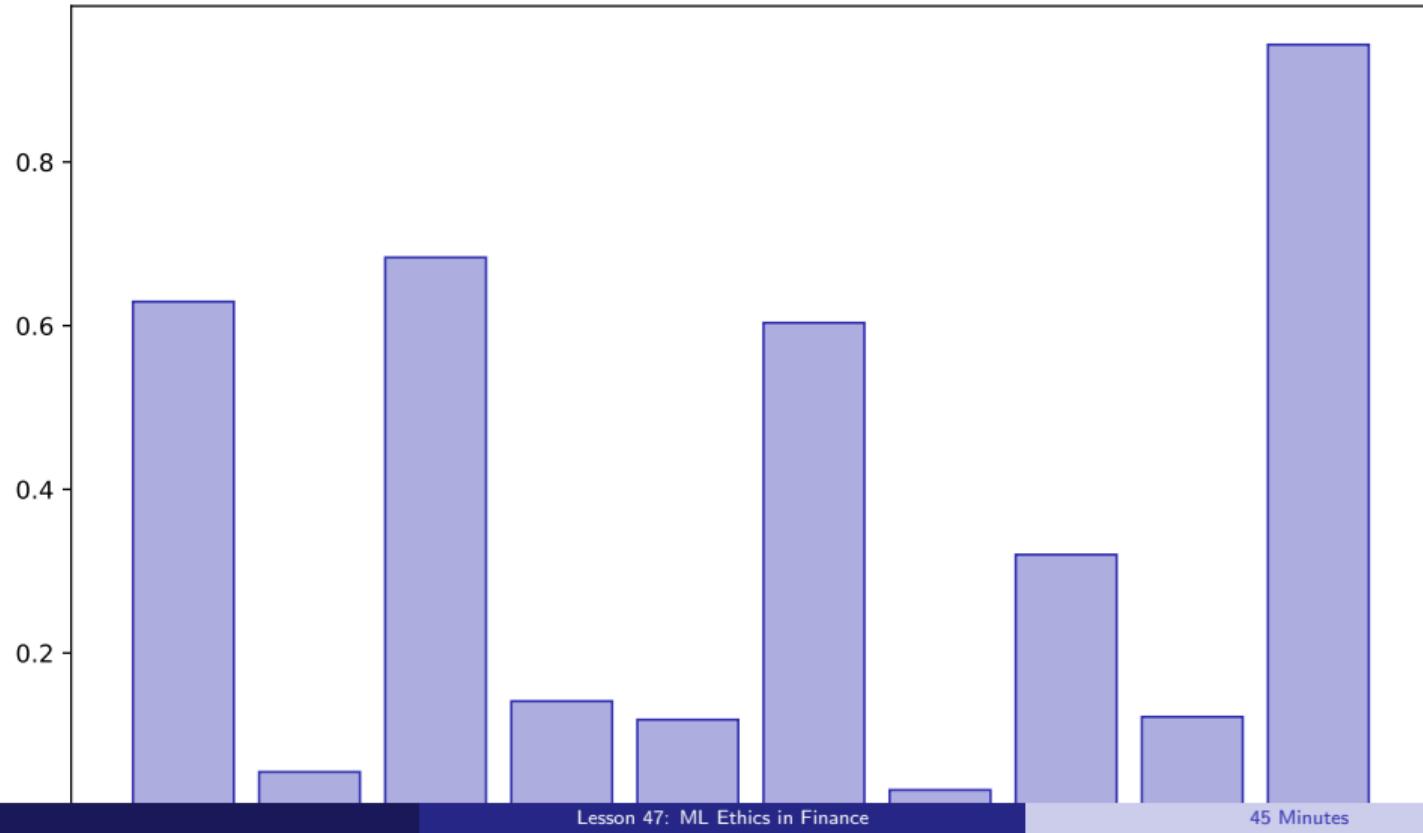
## Fairness Metrics



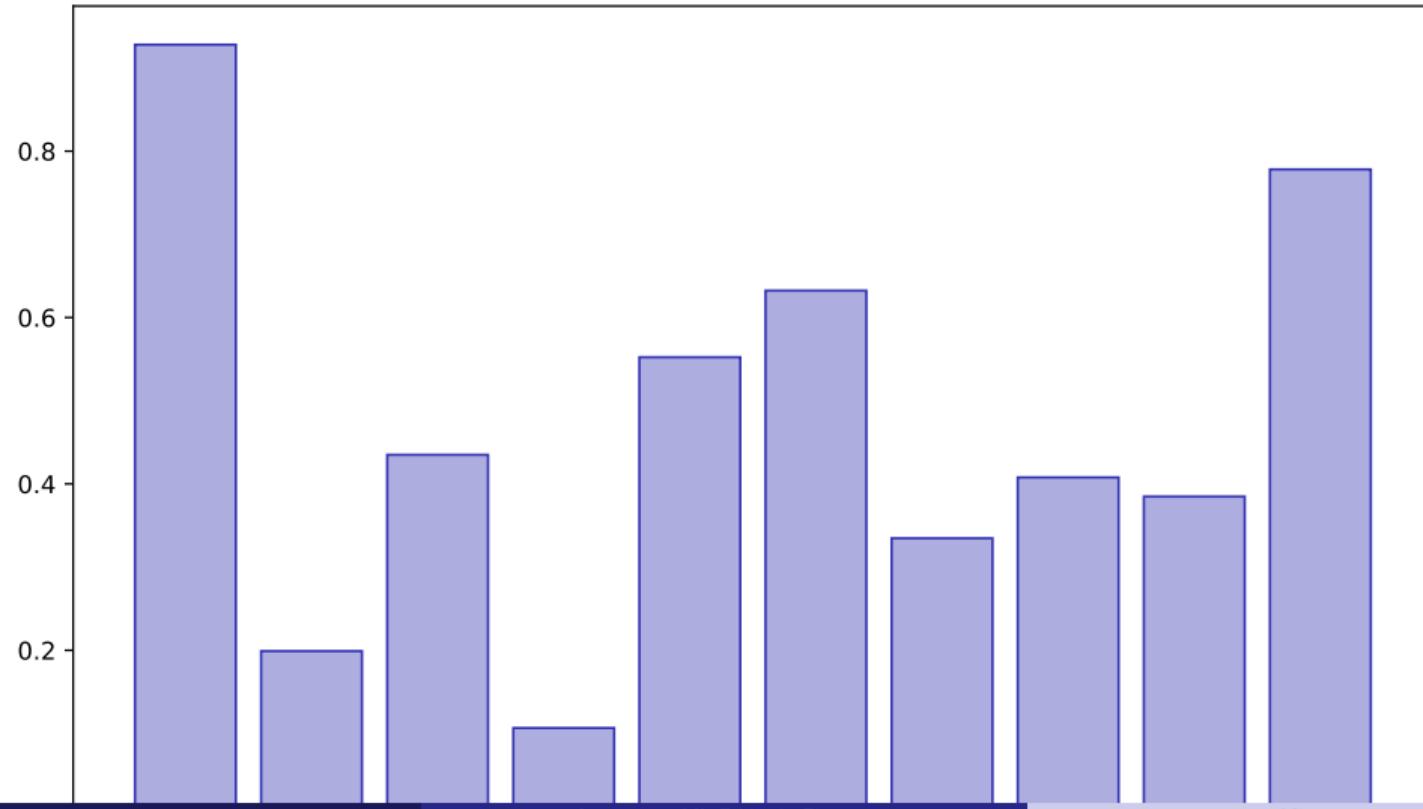
## Regulatory Requirements



## Responsible Ai



## Finance Ethics



## Lesson Summary

### Key Takeaways:

- Identify bias in models
- Ensure explainability
- Consider fairness
- Follow regulations

Apply these skills in your final project

## Lesson 48: Final Presentations

Data Science with Python – BSc Course

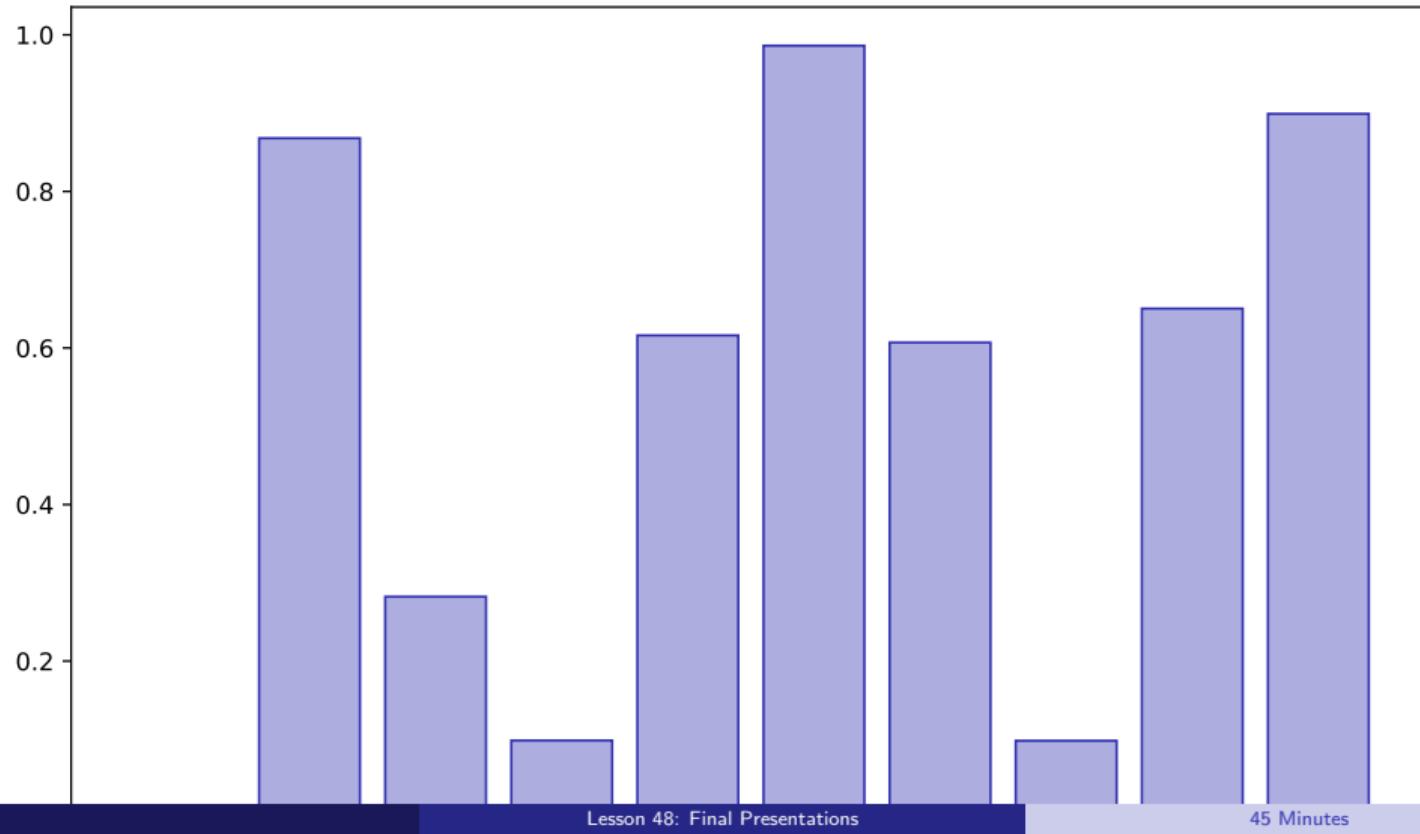
45 Minutes

**After this lesson, you will be able to:**

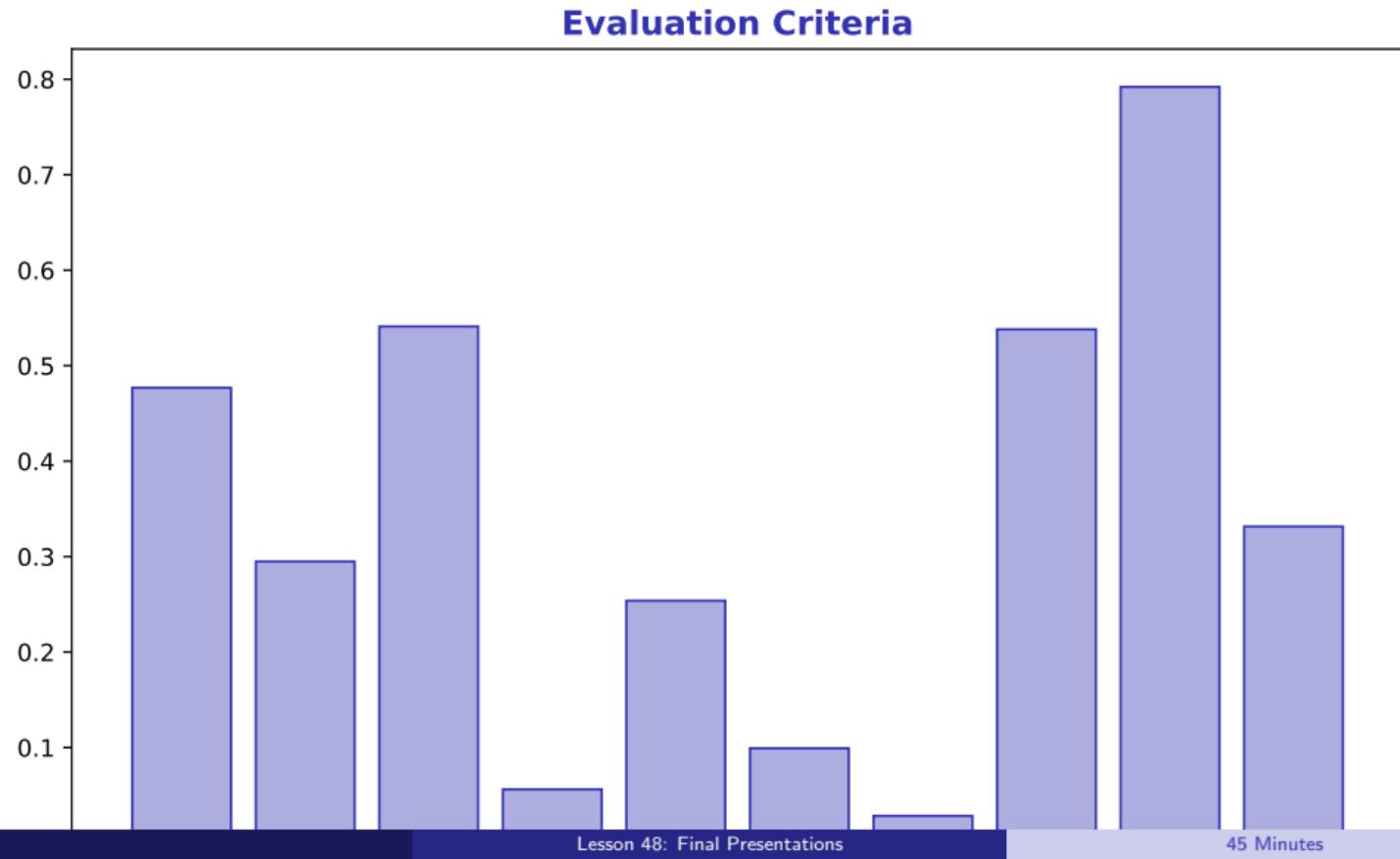
- Present projects
- Demonstrate models
- Answer questions
- Receive feedback

**Building towards your final project**

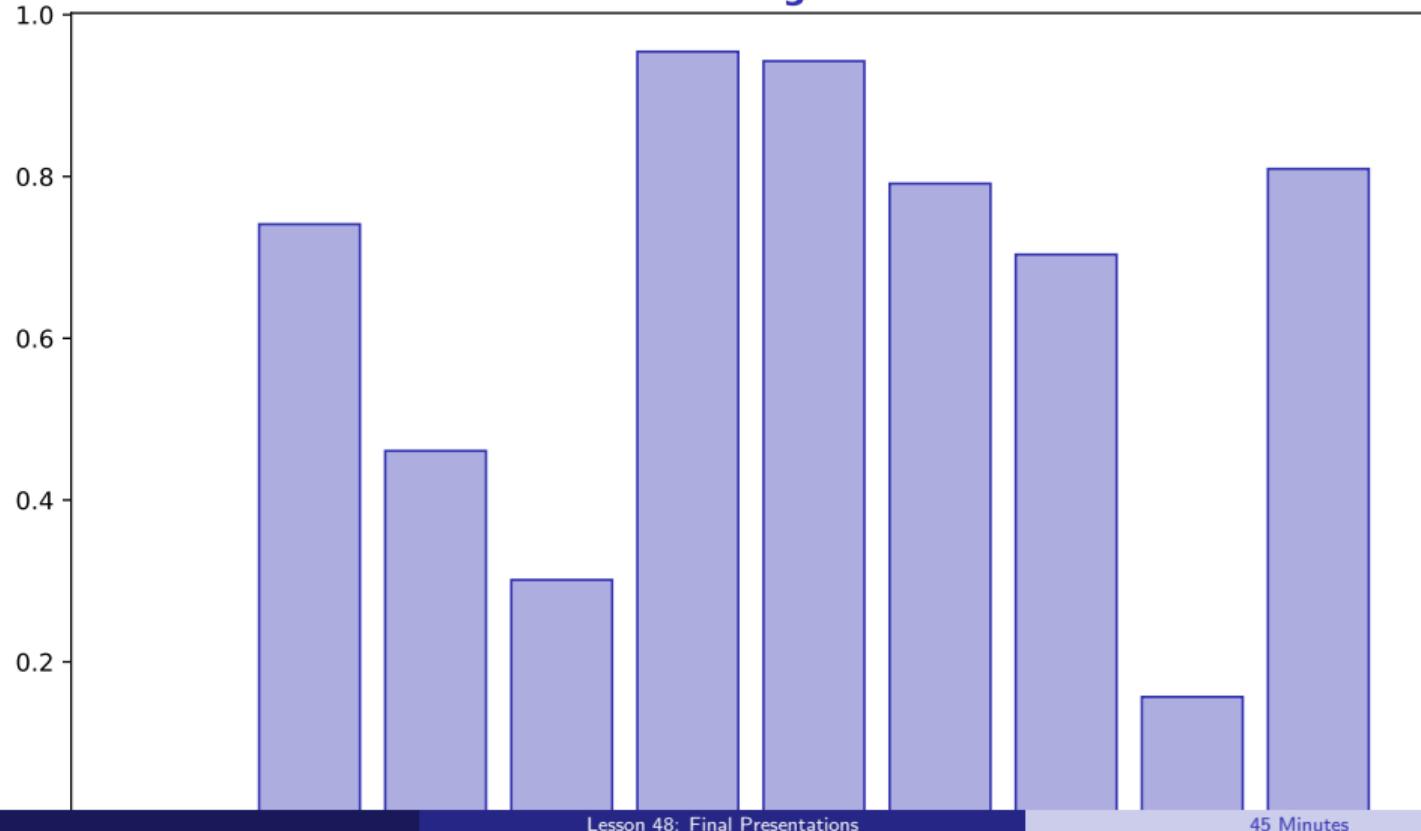
## Presentation Guidelines



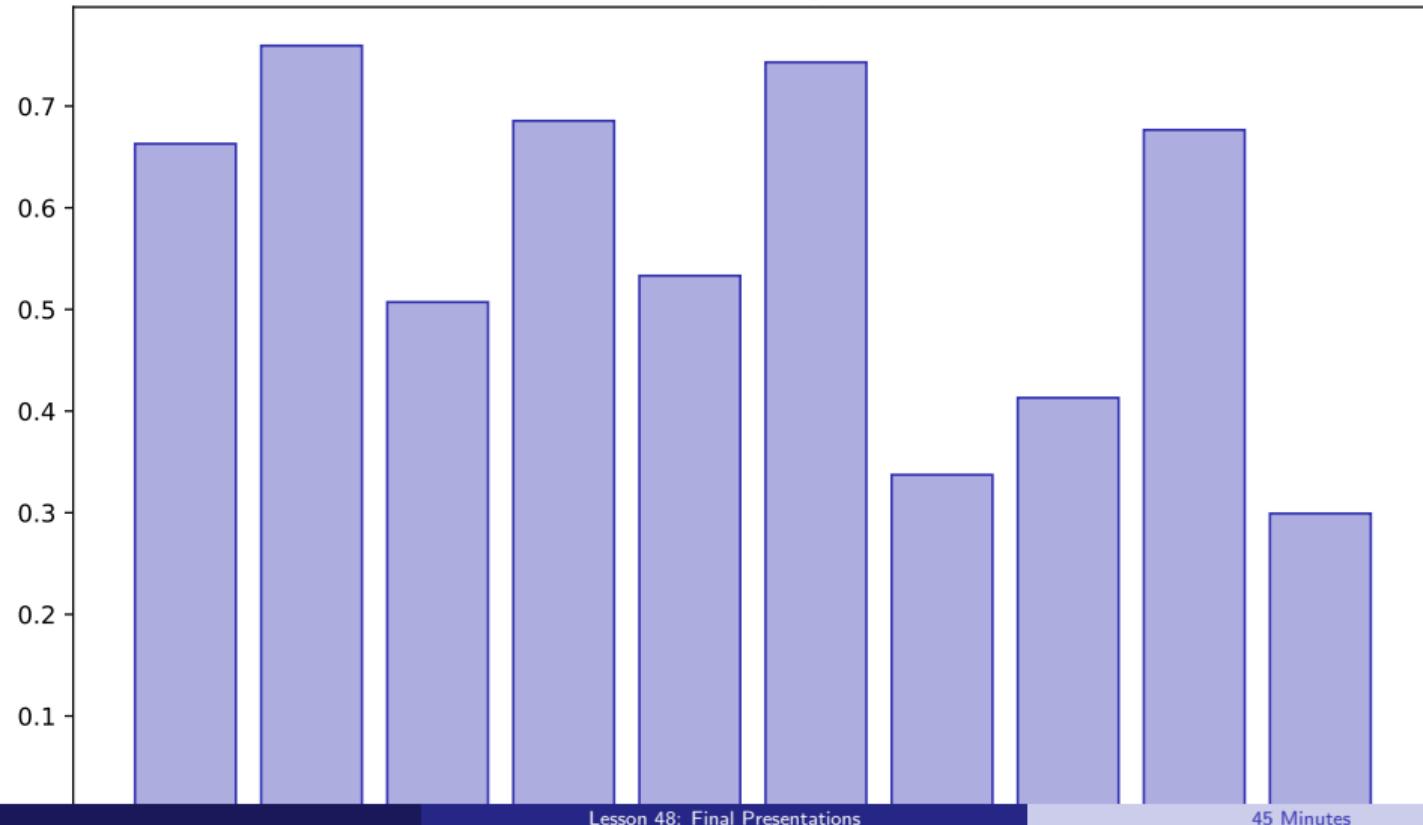
## Evaluation Criteria



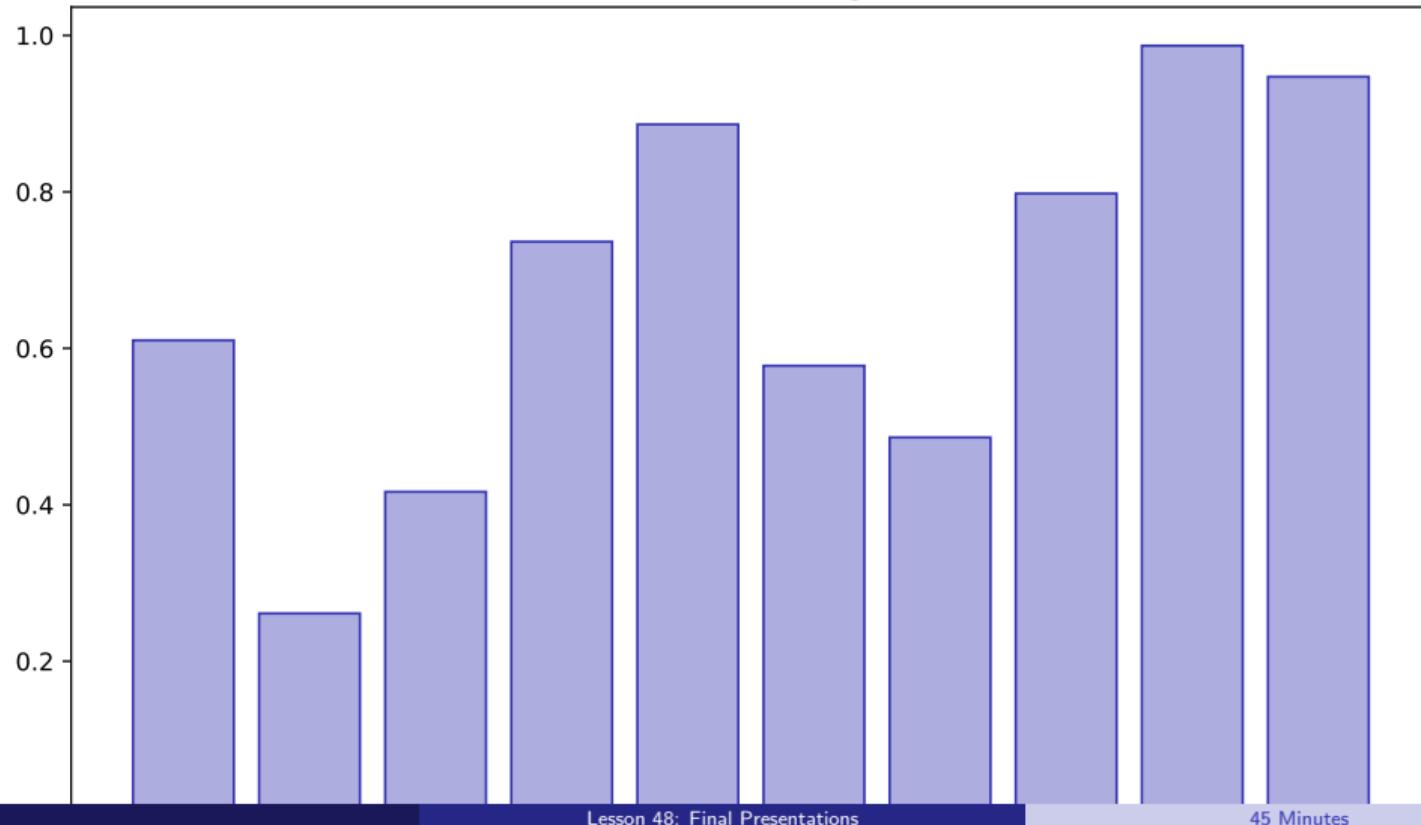
## Time Management



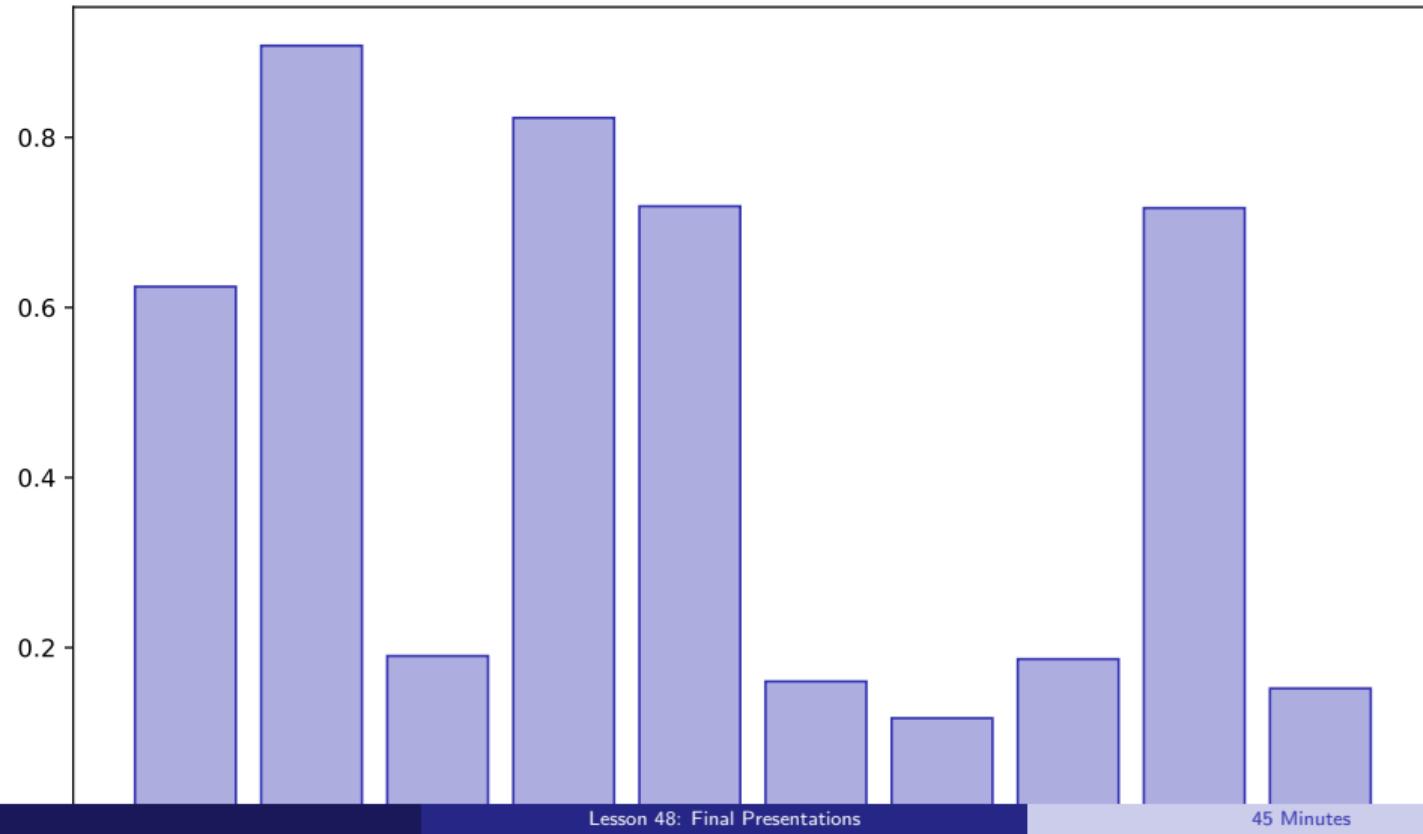
### Demo Tips



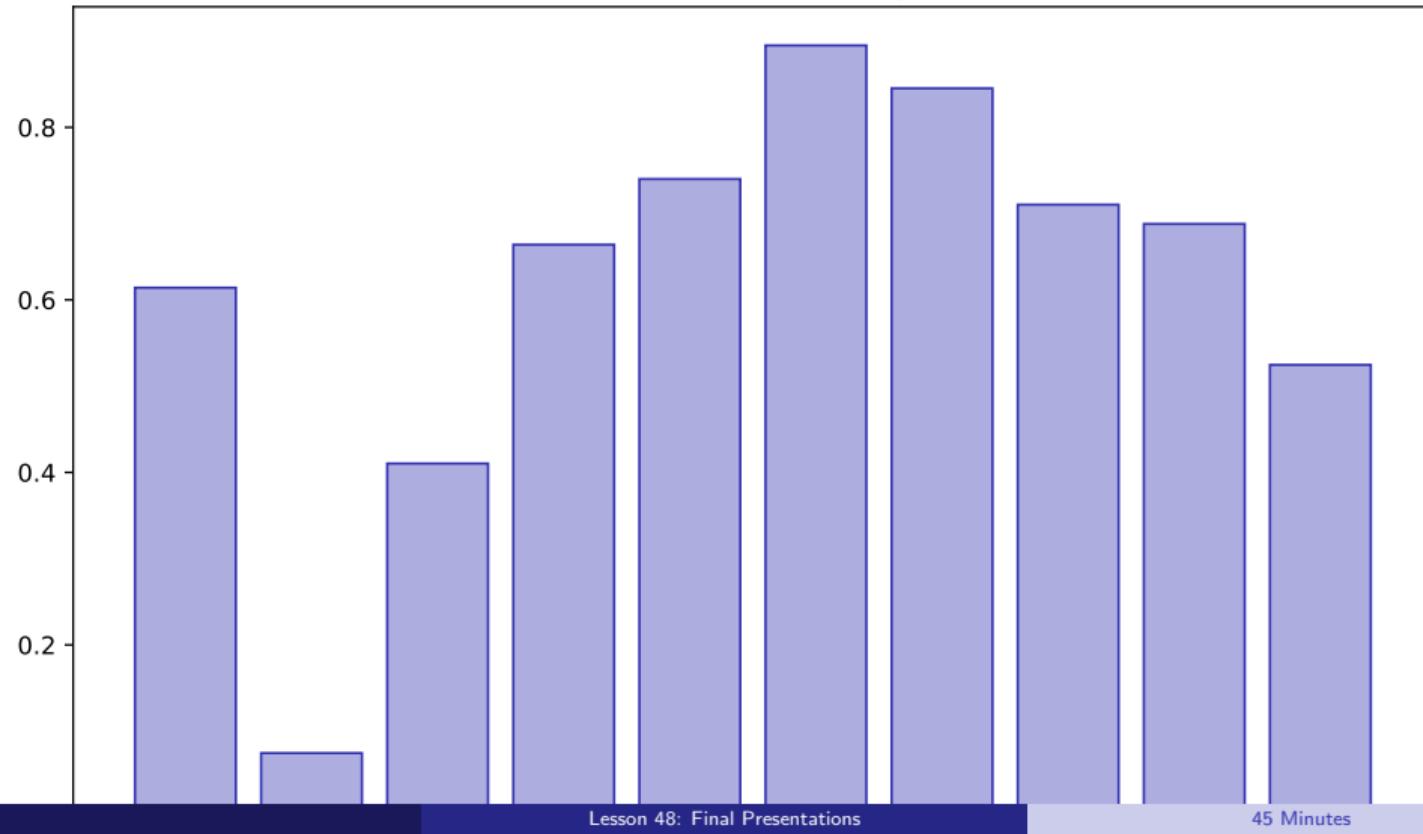
## Qa Handling



### Peer Evaluation

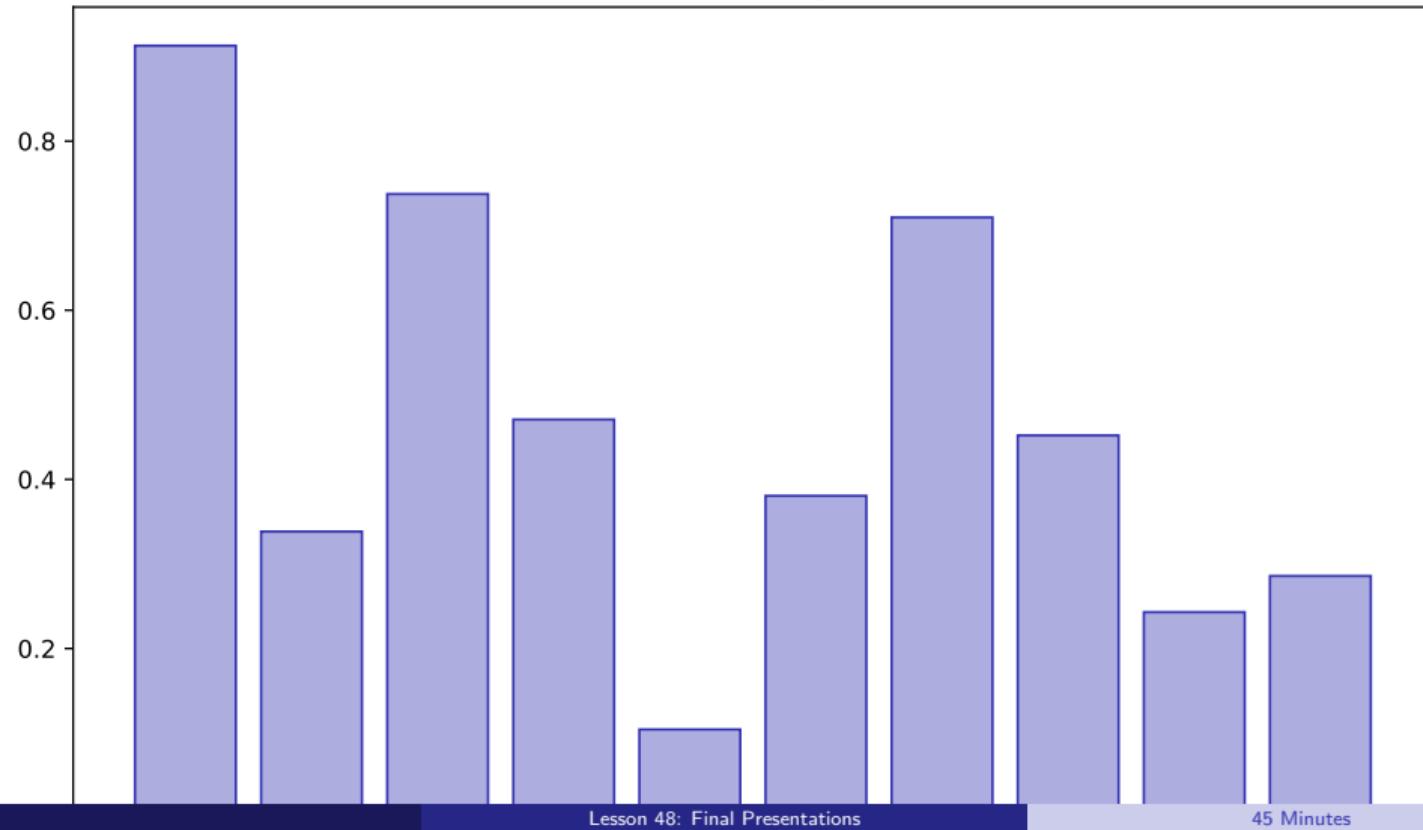


## Course Summary



## Next Steps

### Next Steps



## Lesson Summary

### Key Takeaways:

- Present projects
- Demonstrate models
- Answer questions
- Receive feedback

Apply these skills in your final project