

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

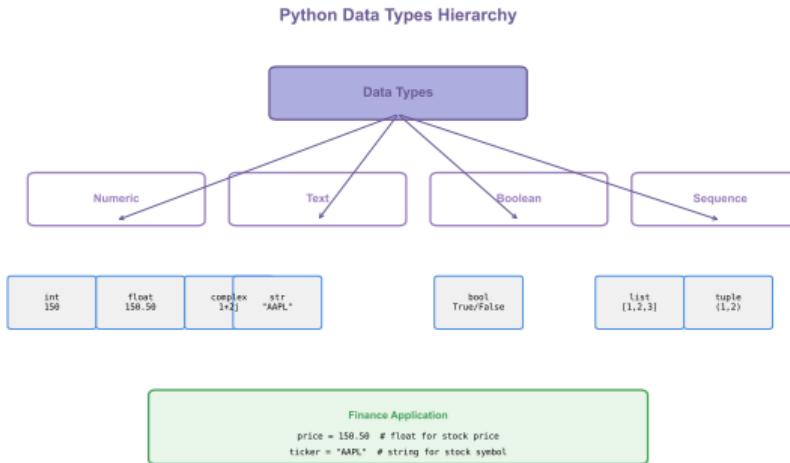
Getting Started:

- 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
```

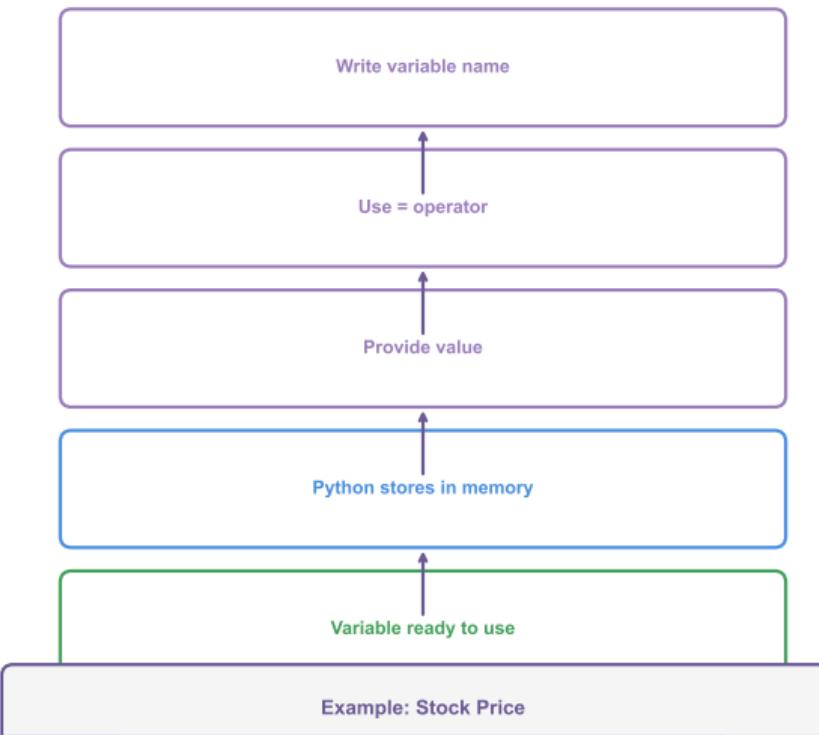
`str` – Text strings

```
ticker = "AAPL"
```

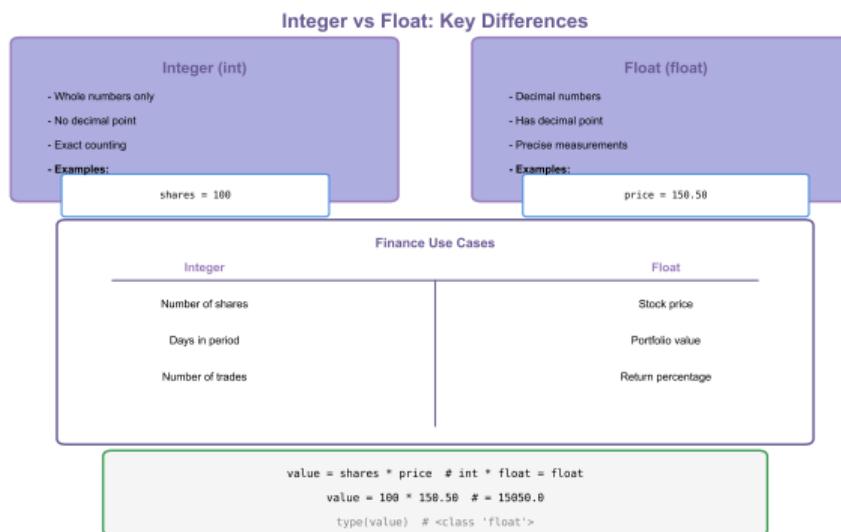
`bool` – True/False

```
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

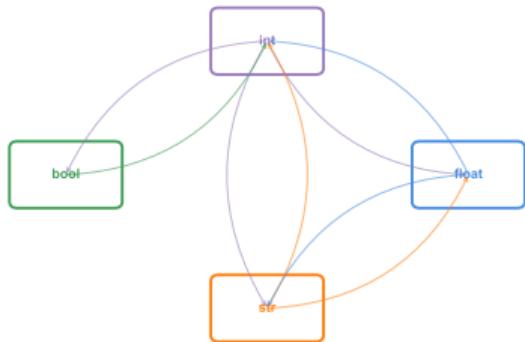
Example:

`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

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)

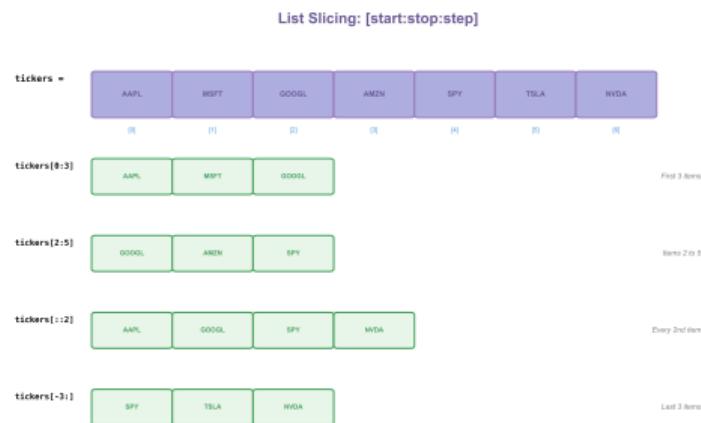
prices[1] → 190 (second)

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

Dictionary: Key-Value Pairs

| KEY | : | VALUE |
|---------|---|--------|
| "AAPL" | : | 150.5 |
| "MSFT" | : | 340.0 |
| "GOOGL" | : | 325.75 |
| "AMZN" | : | 165.0 |

Dictionary Operations

```
portfolio["AAPL"]           → 150.5      # Access value by key  
portfolio["AAPL"] = 155.00          # Update value  
portfolio["TSLA"] = 250.00          # Add new key-value pair  
"MSFT" in portfolio           → True       # Check if key exists
```

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:

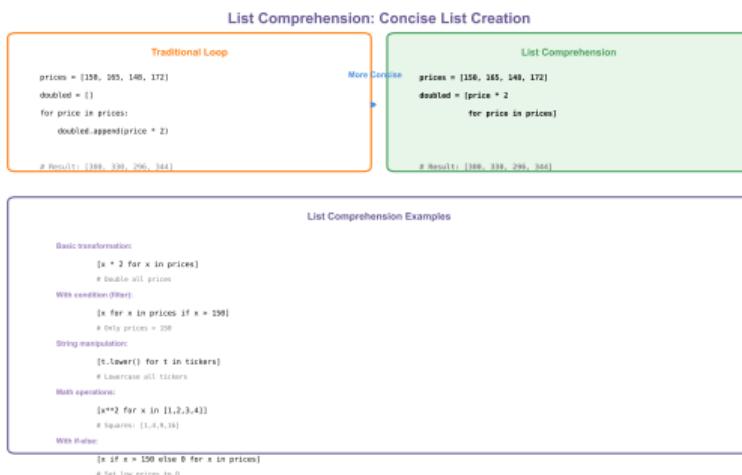
```
total = sum(prices.values())
```

Check Existence:

```
"AAPL" in prices → True
```

Dictionaries are ideal for ticker-to-data mappings

List Comprehension



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

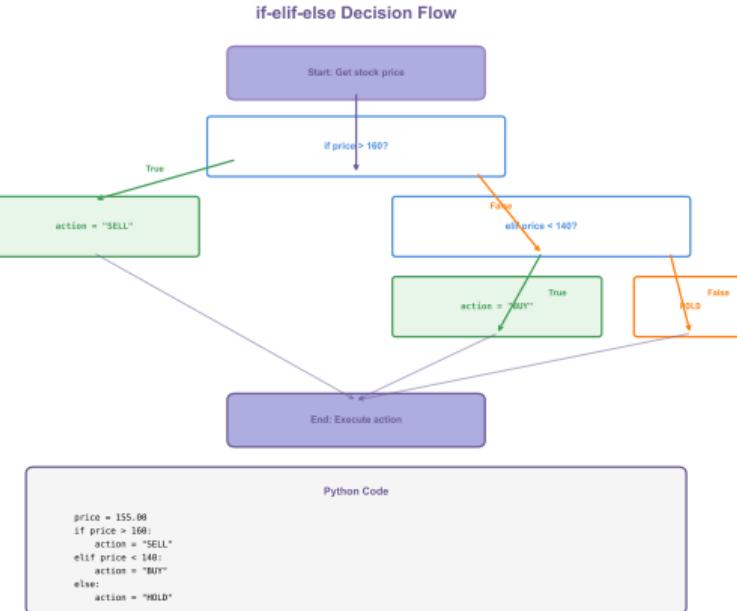
After this lesson, you will be able to:

- 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"
```

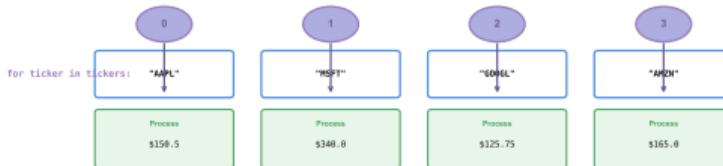
Key Points:

- 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)

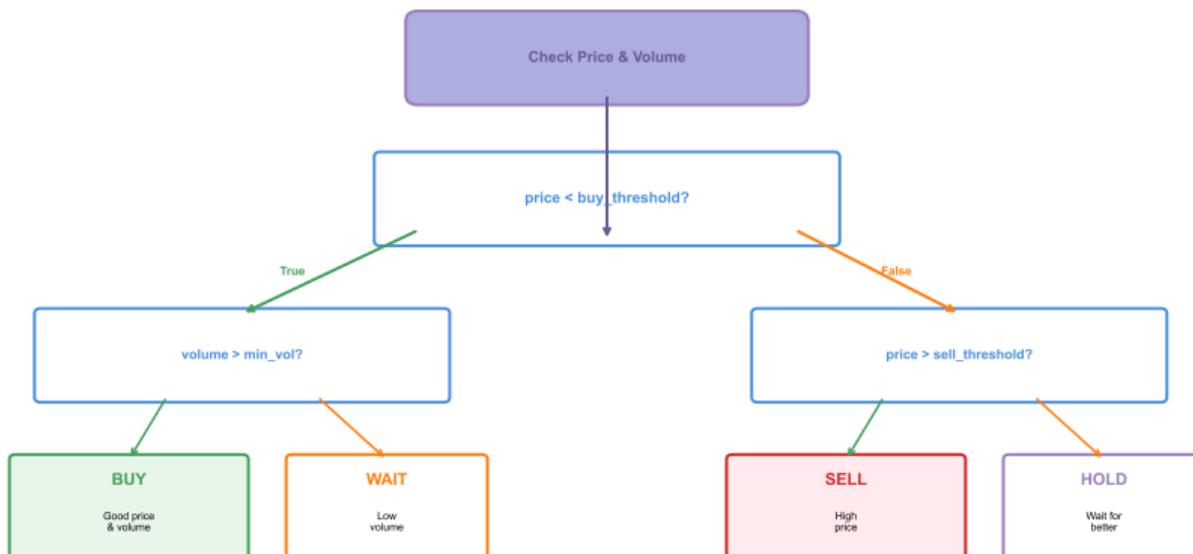
Key Differences

| | | |
|-----------------|-------------------------|-------------------------|
| break | Exits loop completely | Used when condition met |
| continue | Skips current iteration | Used to filter items |
| break | No more iterations | Loop terminates |
| continue | 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 < 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
```

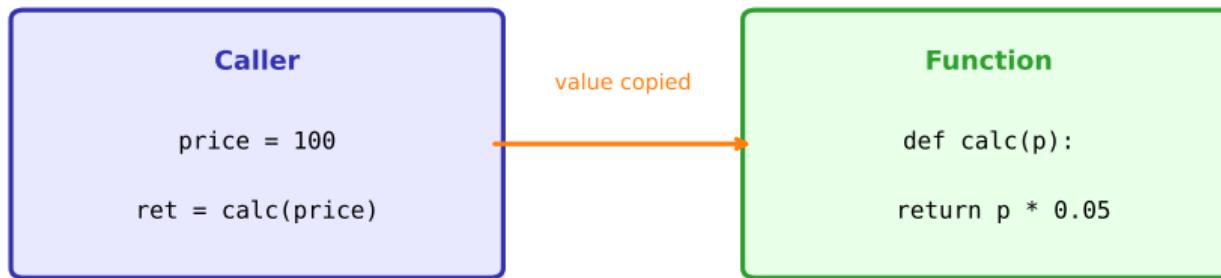
The diagram shows a Python function definition within a blue-bordered box. Annotations with arrows point to specific parts of the code:

- 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



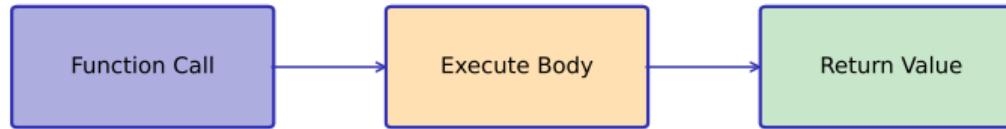
Positional: func(a, b)

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

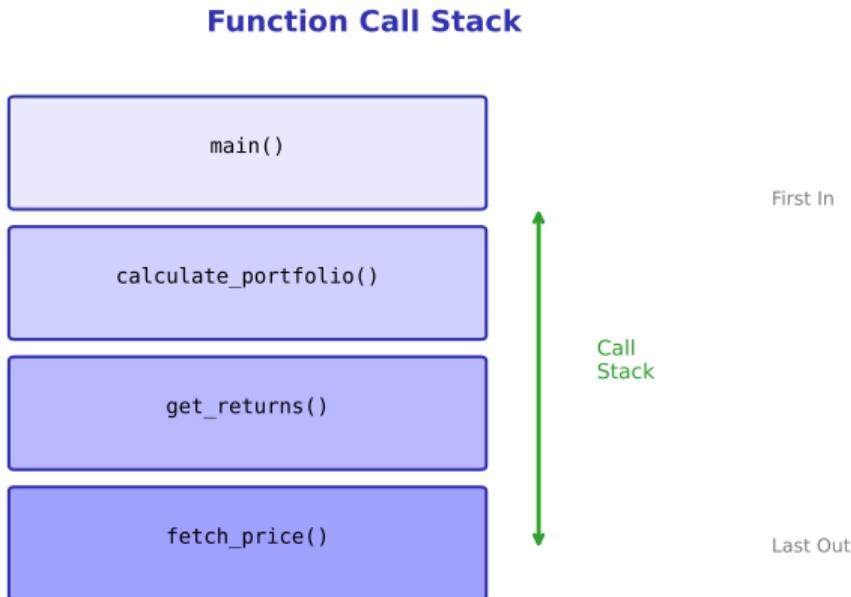
Docstring Best Practices

```
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 |

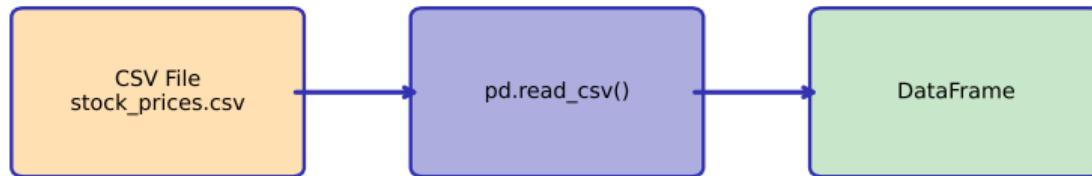
DataFrame (2D)

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):

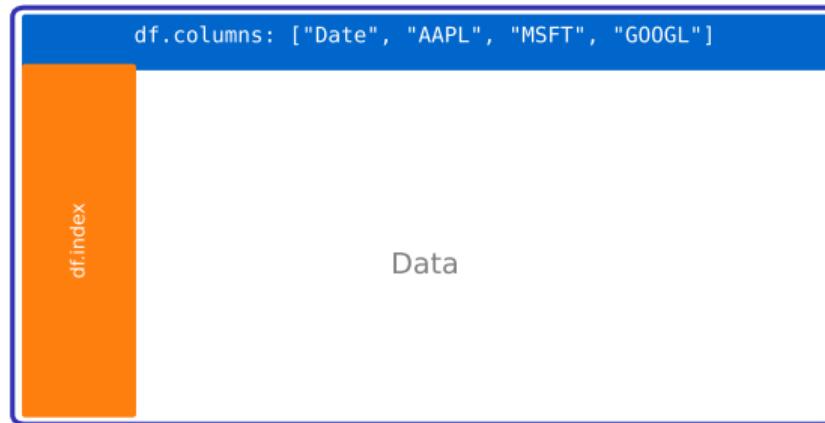
 Date      252 non-null datetime64
 AAPL      252 non-null float64
 MSFT      252 non-null float64
 GOOGL     250 non-null float64 (2 missing)

memory usage: 10.0 KB
```

Summary Statistics: df.describe()

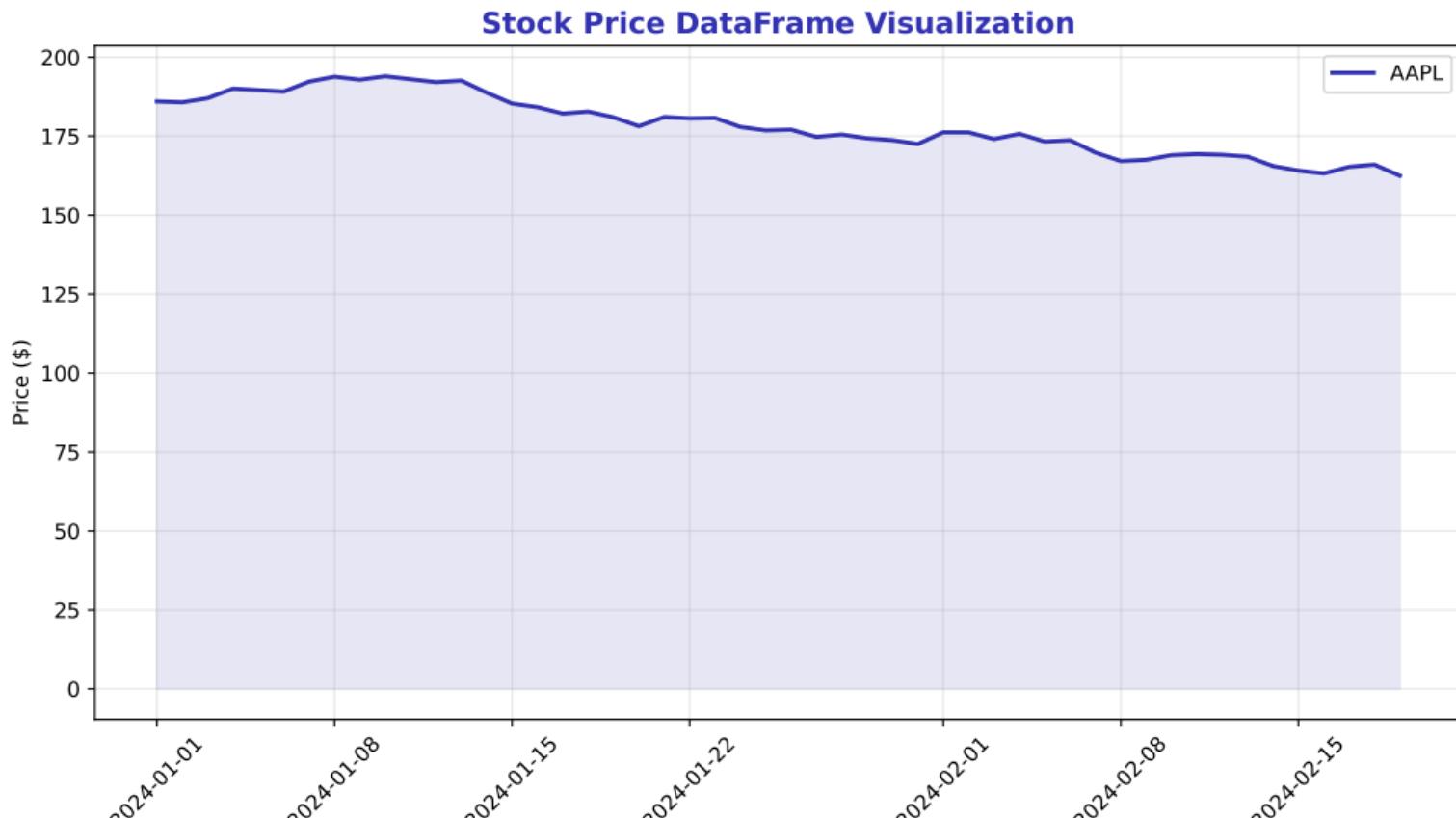
| Stat | AAPL | MSFT |
|-------|-------|-------|
| count | 252 | 252 |
| mean | 189.5 | 385.2 |
| std | 8.2 | 12.5 |
| min | 175.1 | 355.8 |
| 25% | 183.4 | 375.6 |
| 50% | 188.9 | 384.1 |
| 75% | 195.2 | 394.8 |
| max | 210.3 | 420.5 |

Index and Columns



df.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

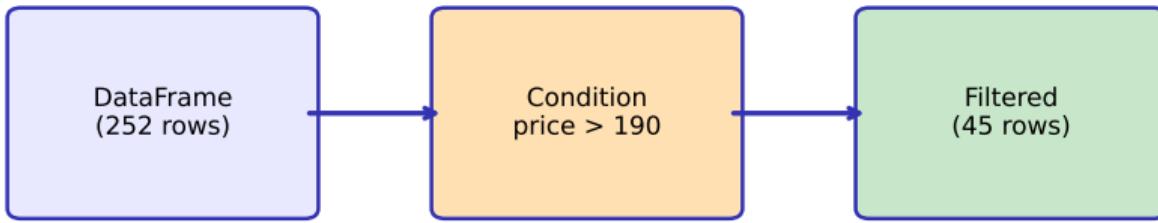
iloc: exclusive end | loc: inclusive end

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")
```

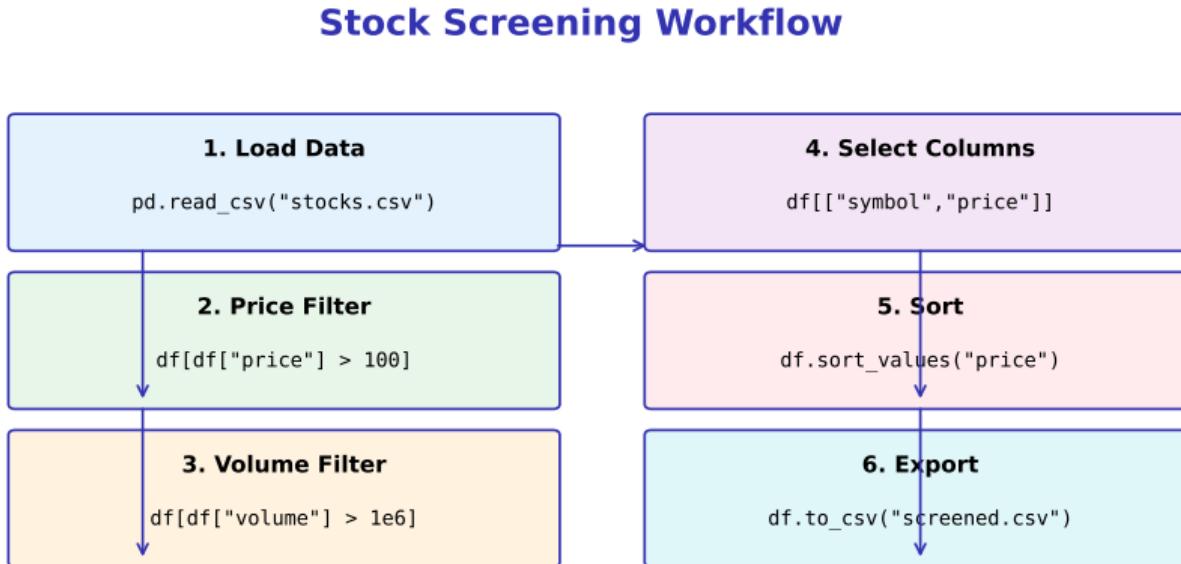
query() is more readable for complex filters

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 > 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

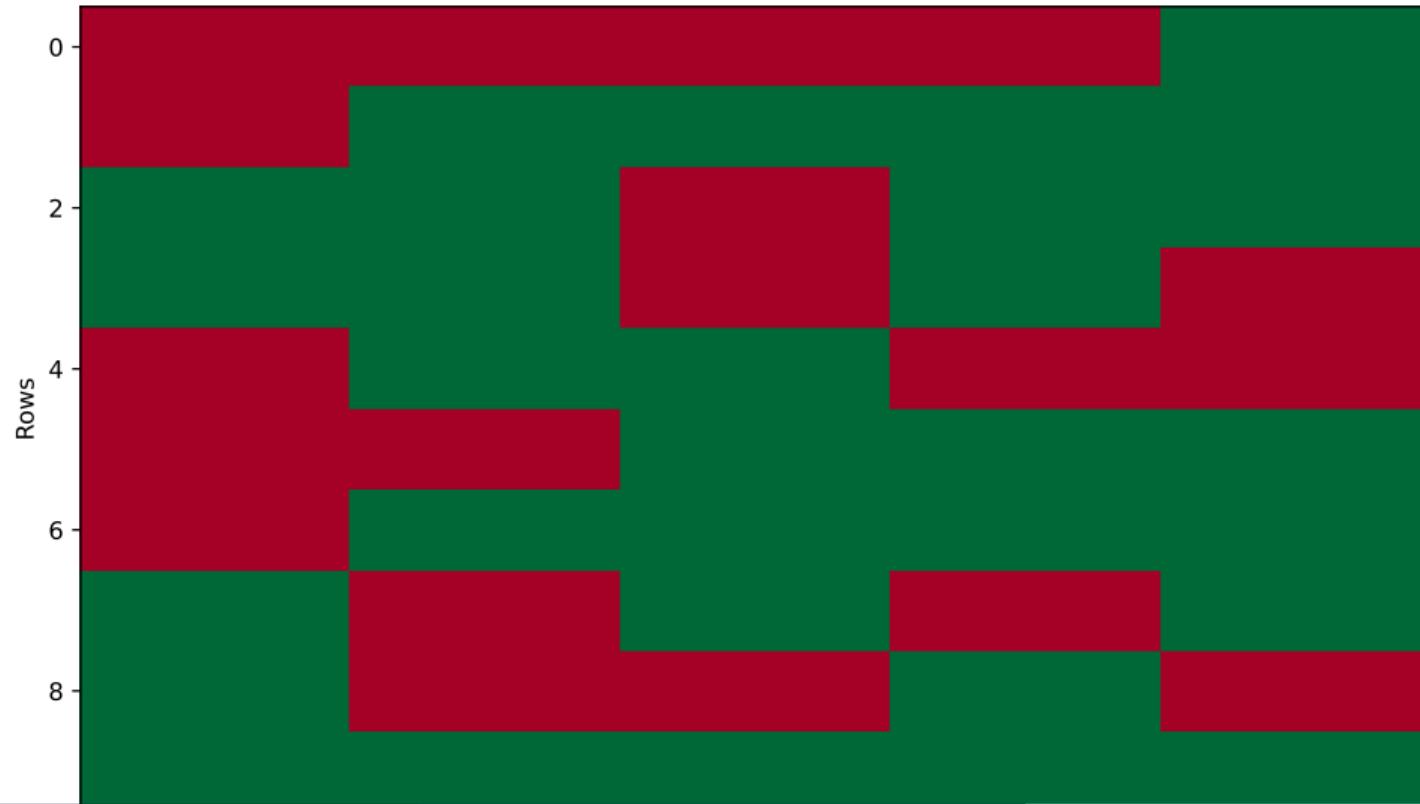
Learning Objectives

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)

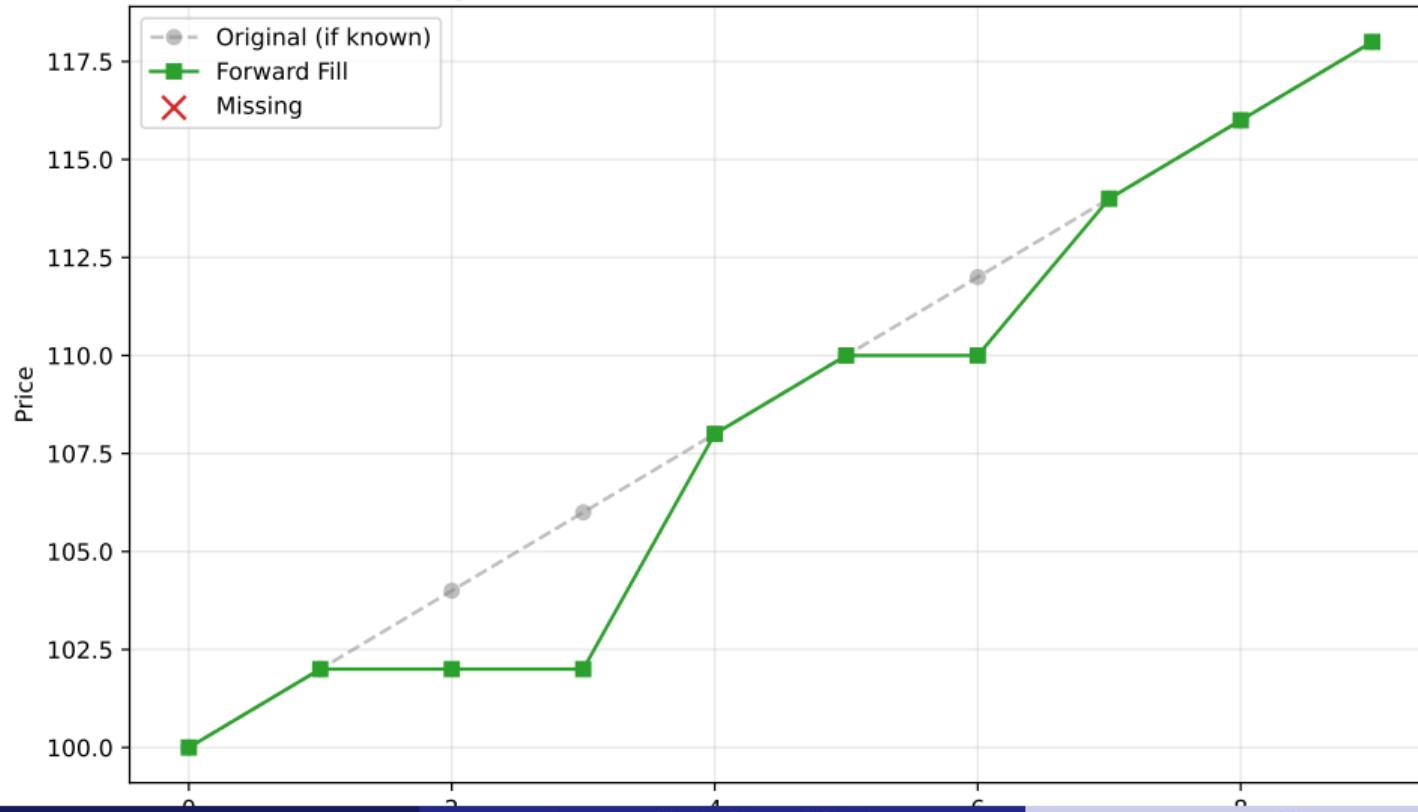
`fillna(df.mean())`

Fill with column mean

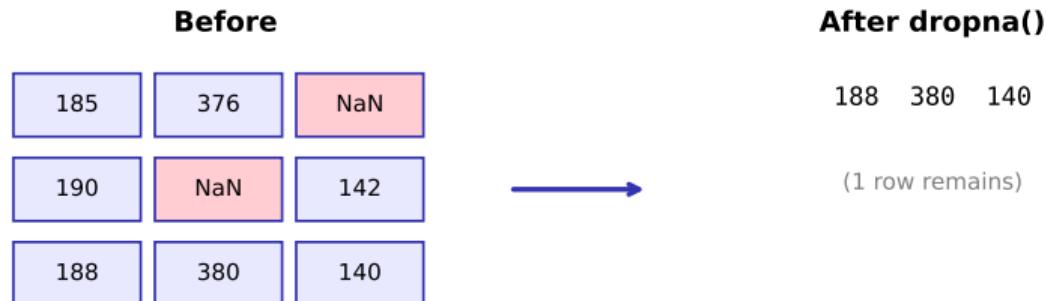
Data Quality Checklist

- [] 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()

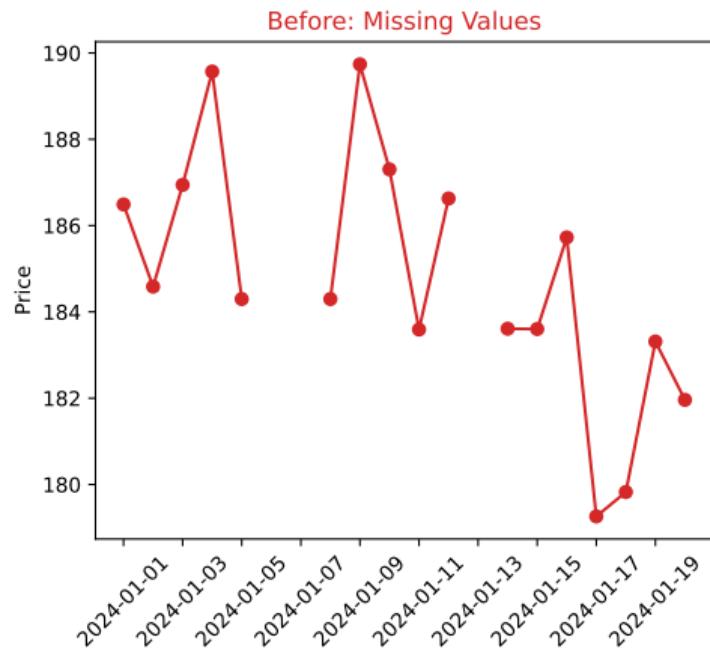
3. Handle missing values

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)
```

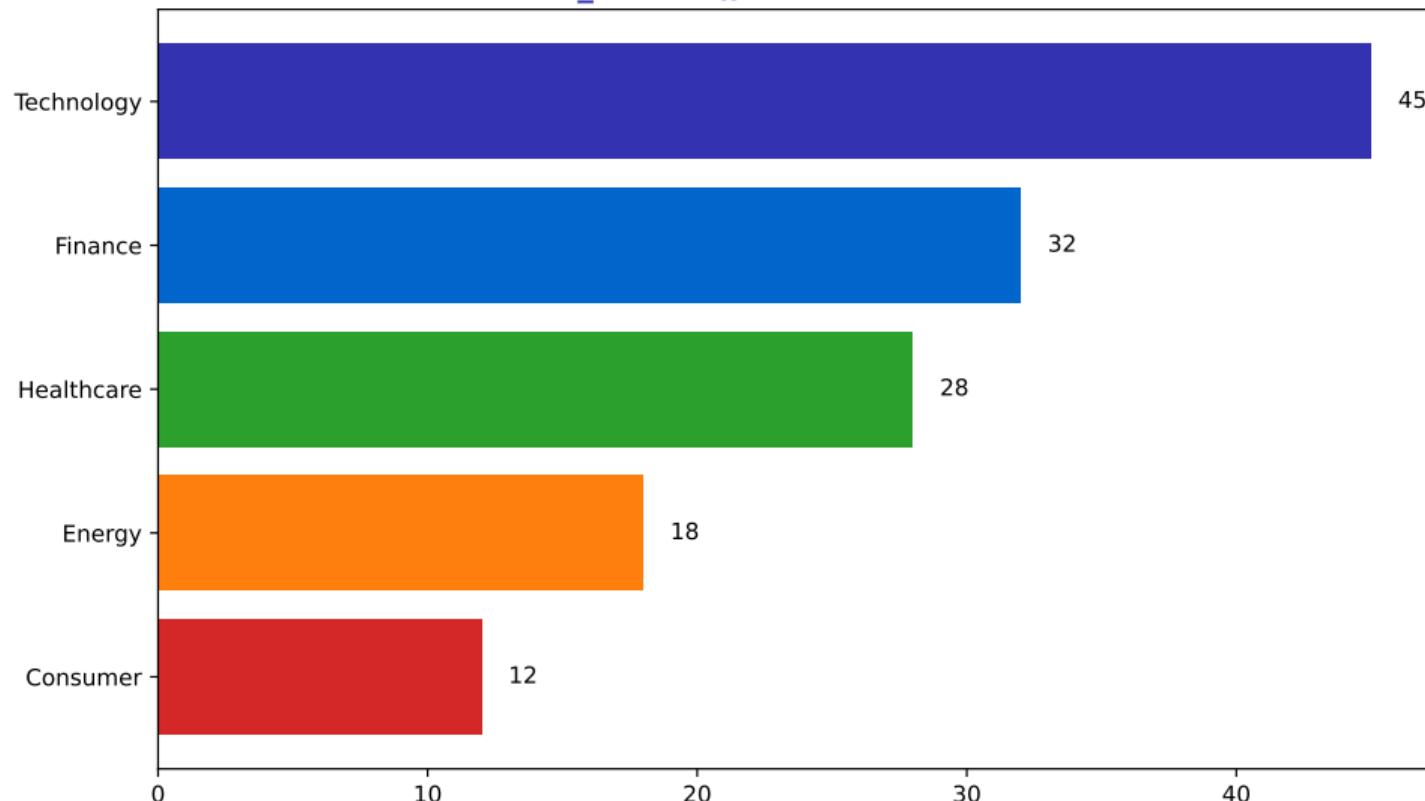
Sort descending

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

Sort by multiple columns

05 Value Counts

`value_counts(): Sector Distribution`



Calculating Returns

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

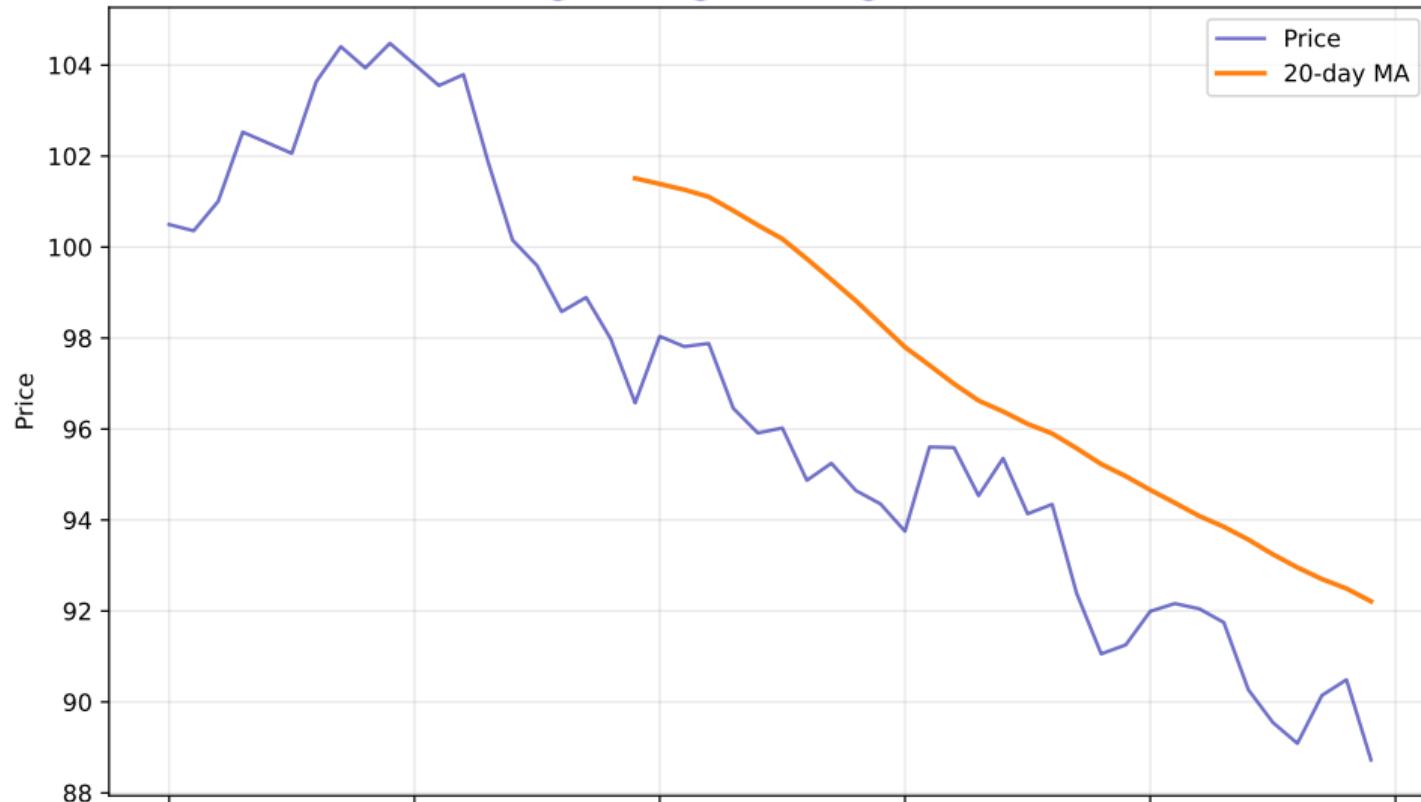
```
df["Return"] = df["Price"].pct_change()
```

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

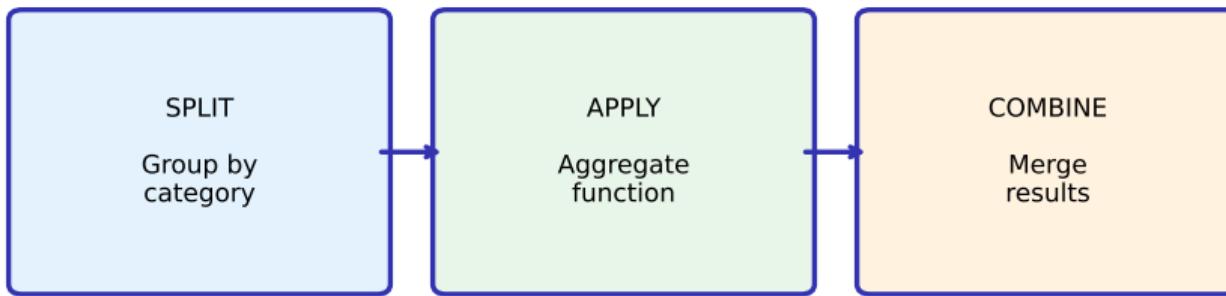
45 Minutes

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()
```

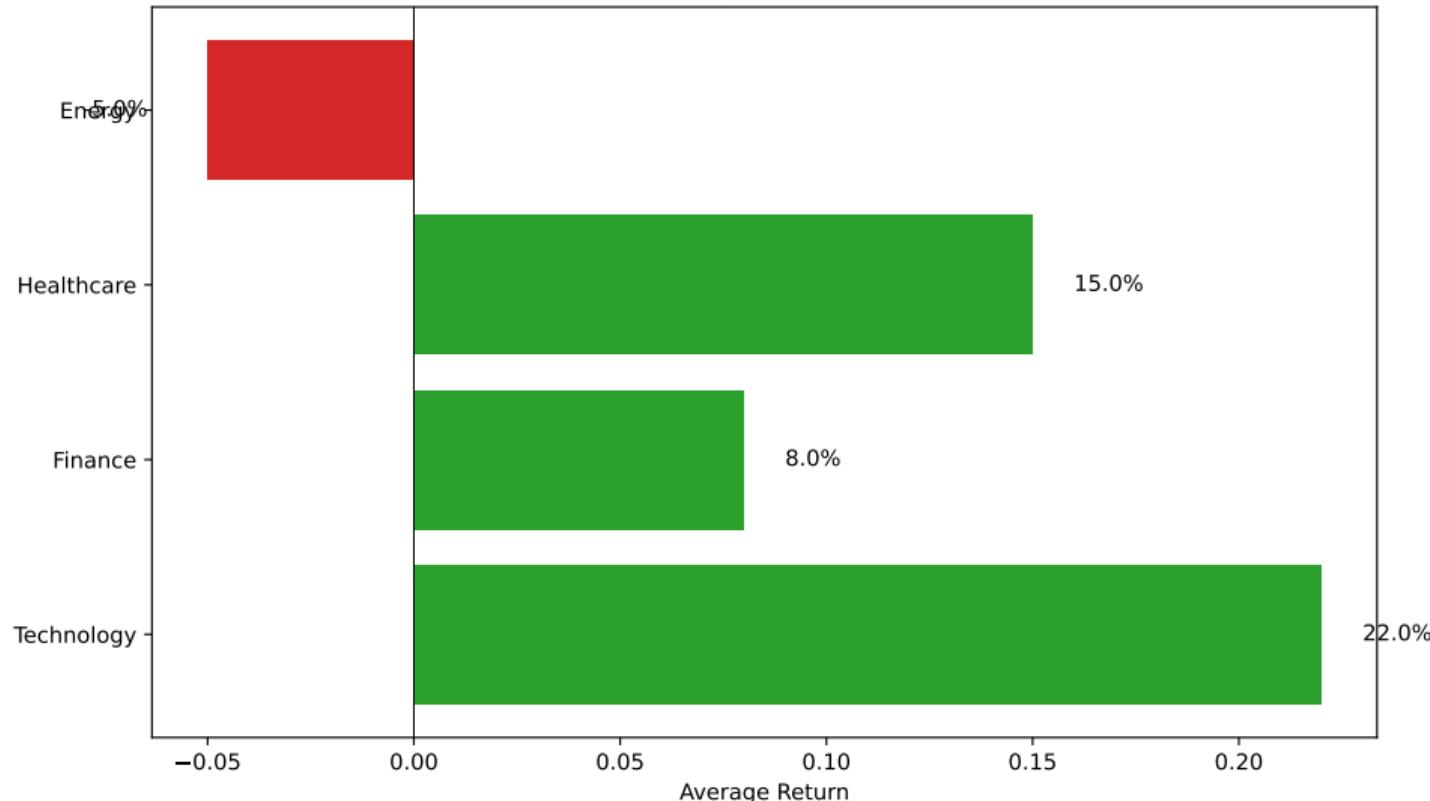
Creates hierarchical grouping:

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()
```

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

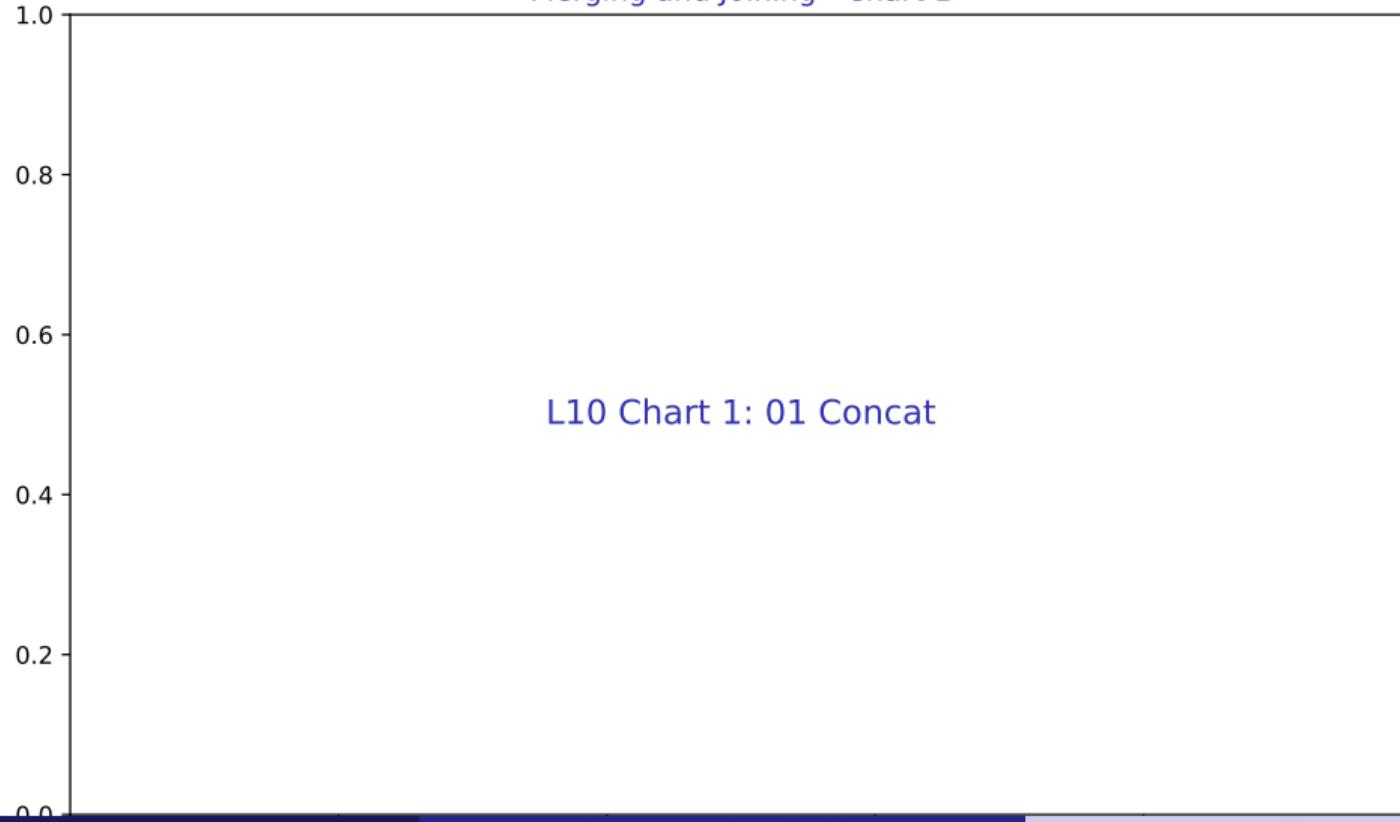
Learning Objectives

After this lesson, you will be able to:

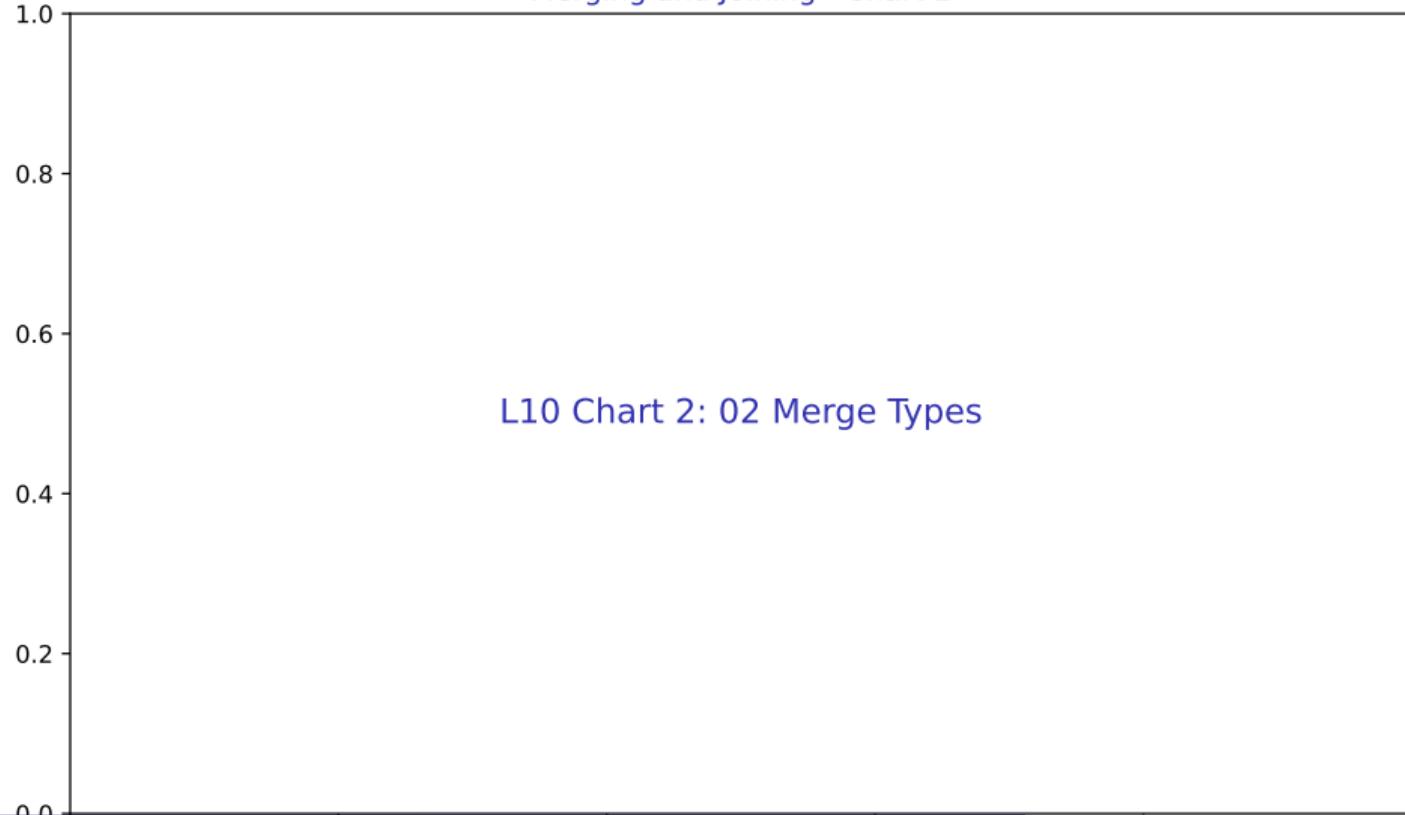
- pd.concat() for stacking
- pd.merge() for SQL-style joins
- Join types: inner, outer, left, right
- Handling key columns

Finance application: Stock data processing and analysis

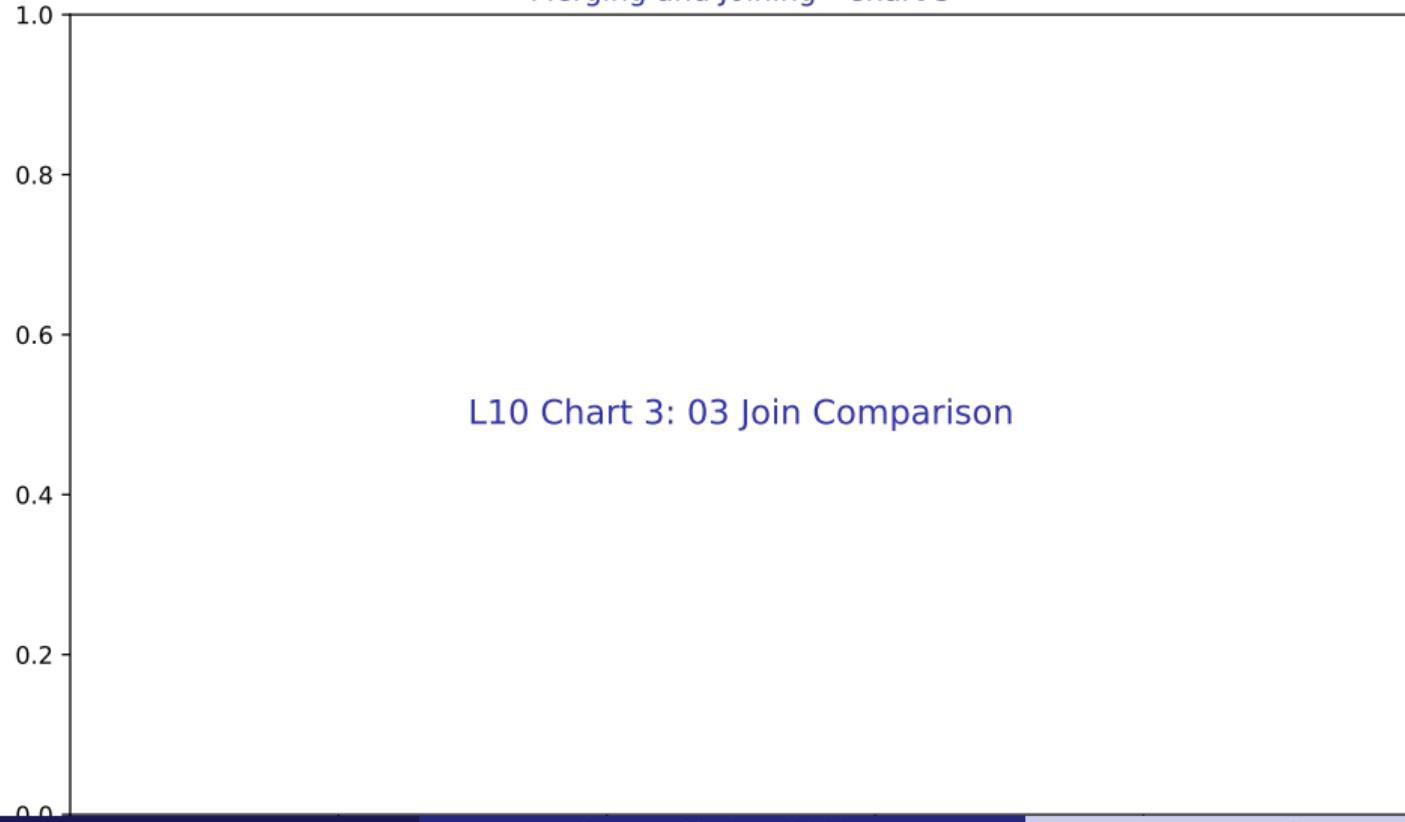
Merging and Joining - Chart 1



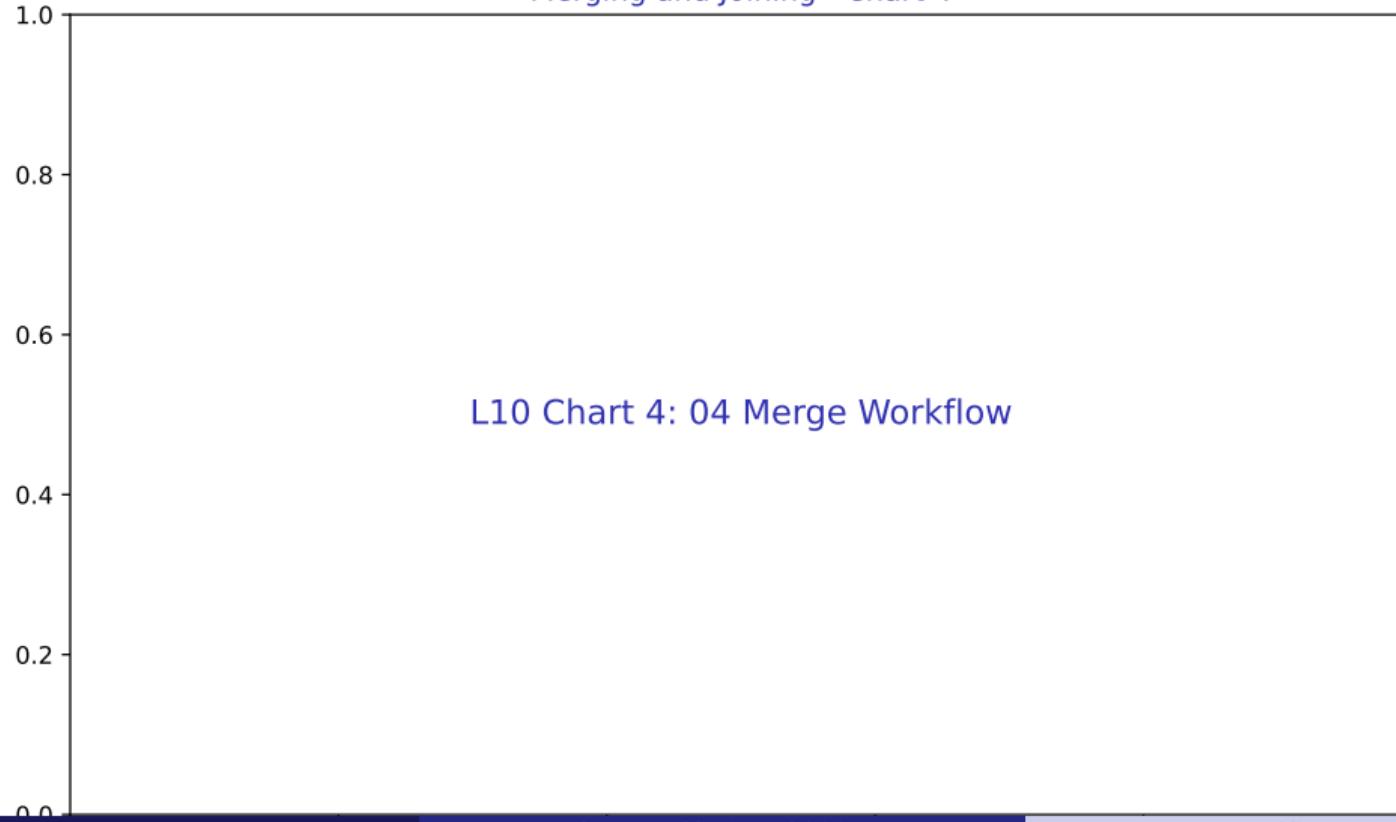
Merging and Joining - Chart 2



Merging and Joining - Chart 3

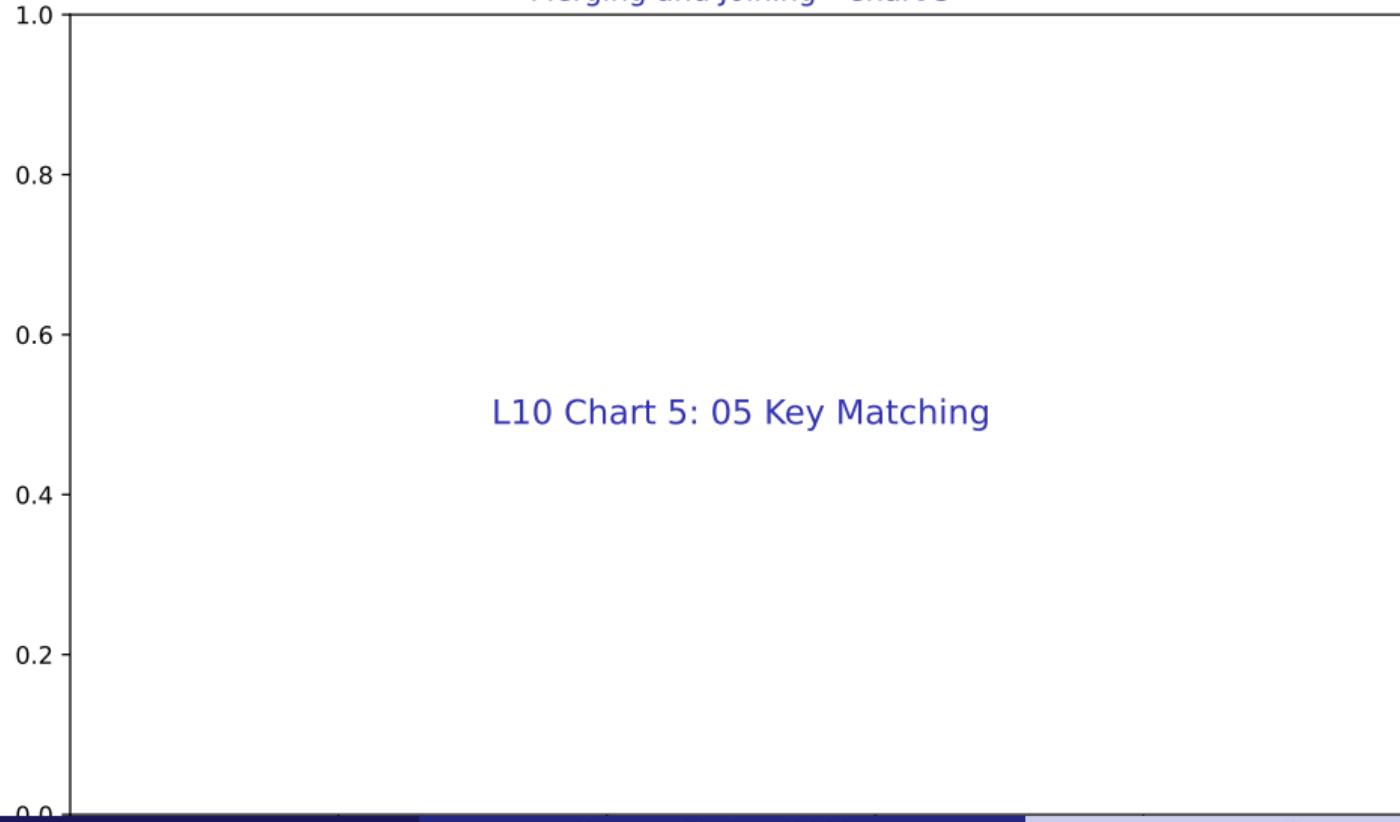


Merging and Joining - Chart 4

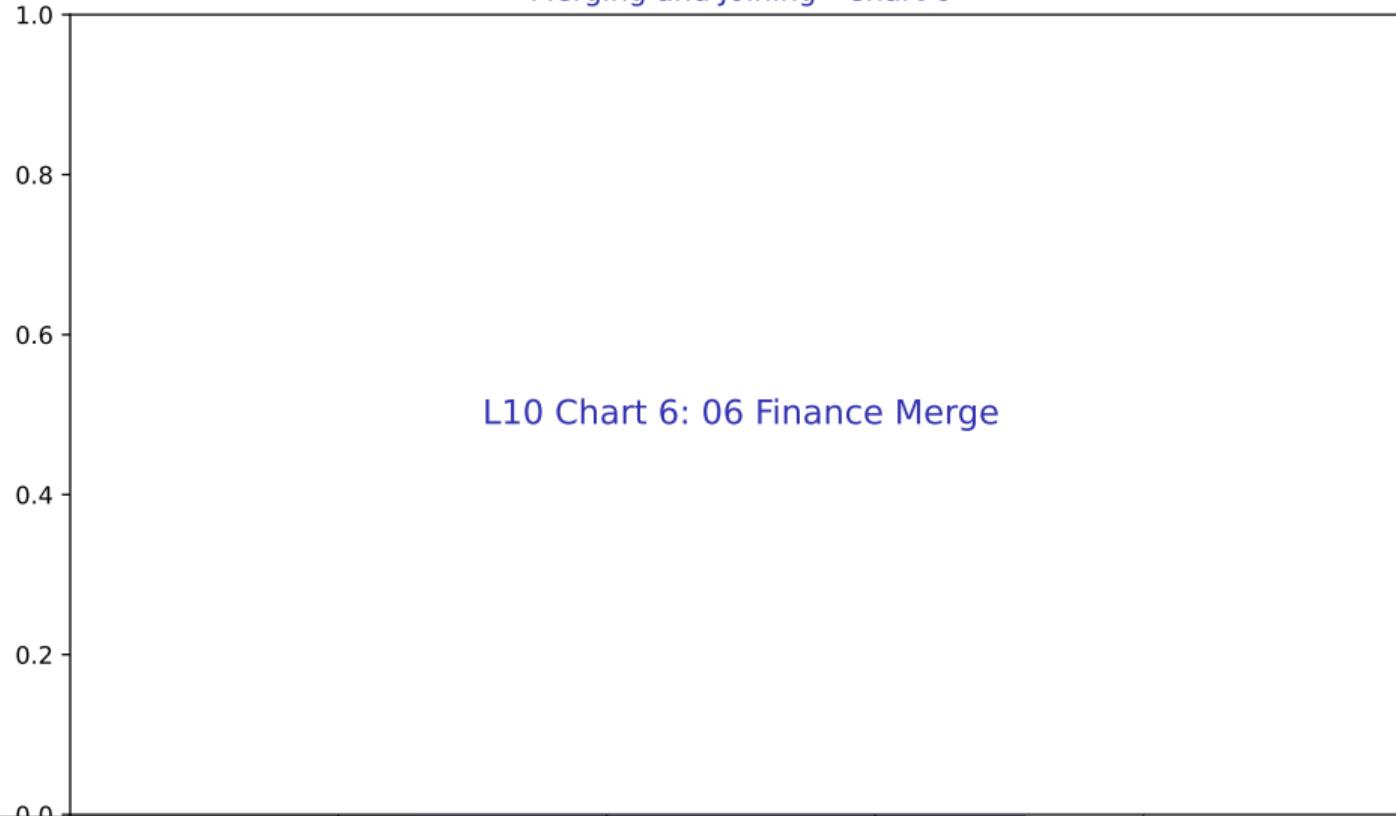


05 Key Matching

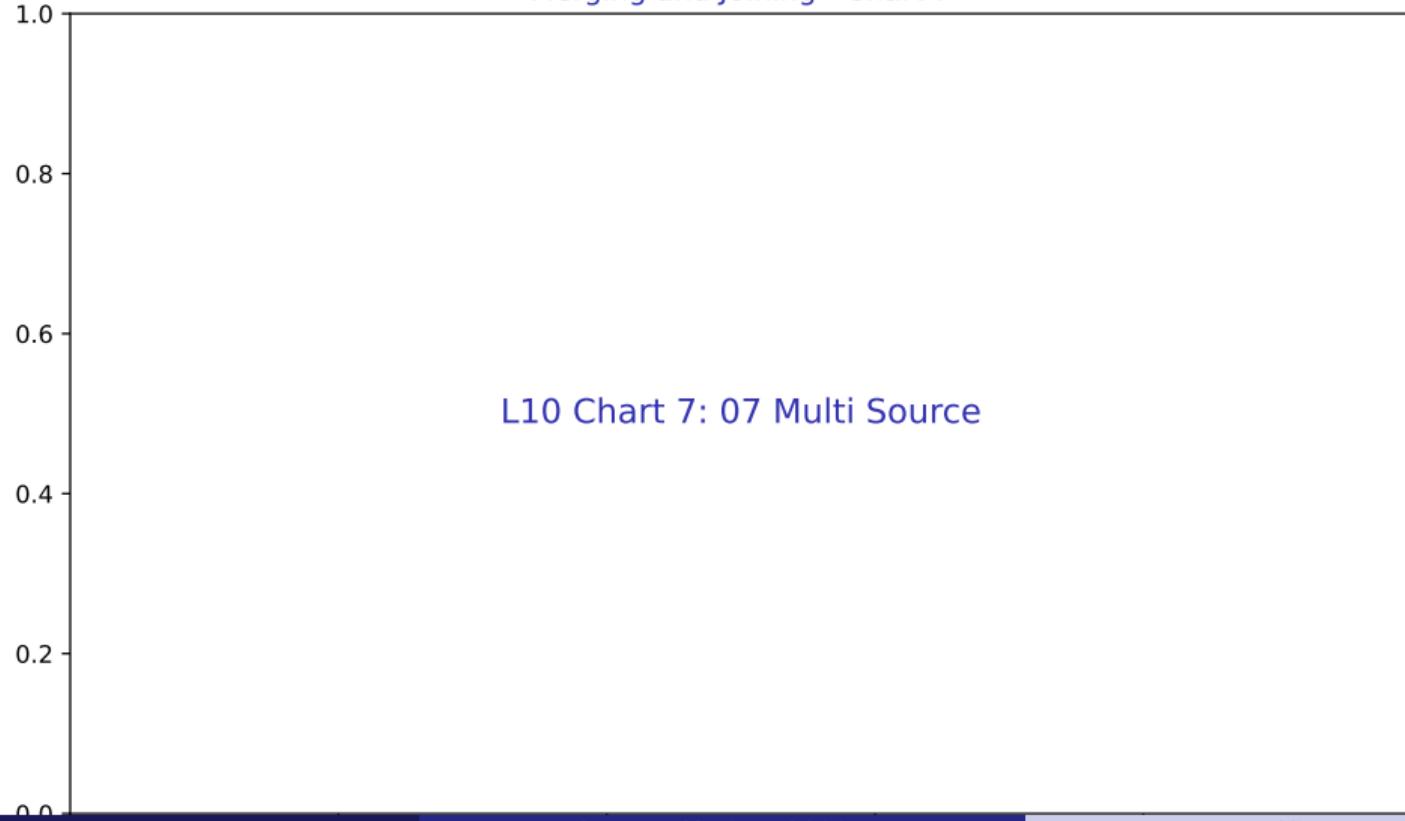
Merging and Joining - Chart 5



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:

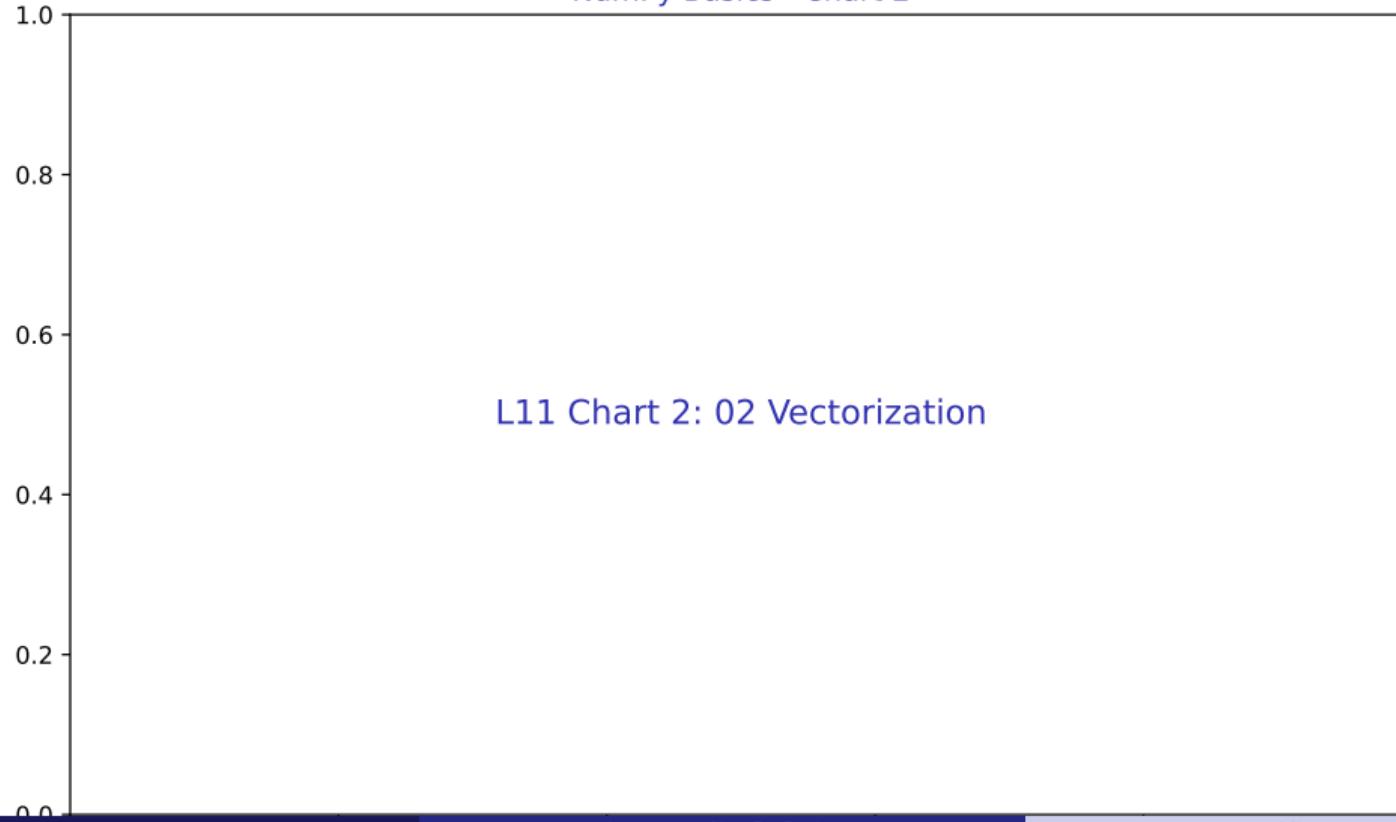
- Arrays vs lists
- Vectorized operations
- Broadcasting
- Mathematical functions
- Portfolio calculations

Finance application: Stock data processing and analysis

NumPy Basics - Chart 1



NumPy Basics - Chart 2



03 Broadcasting

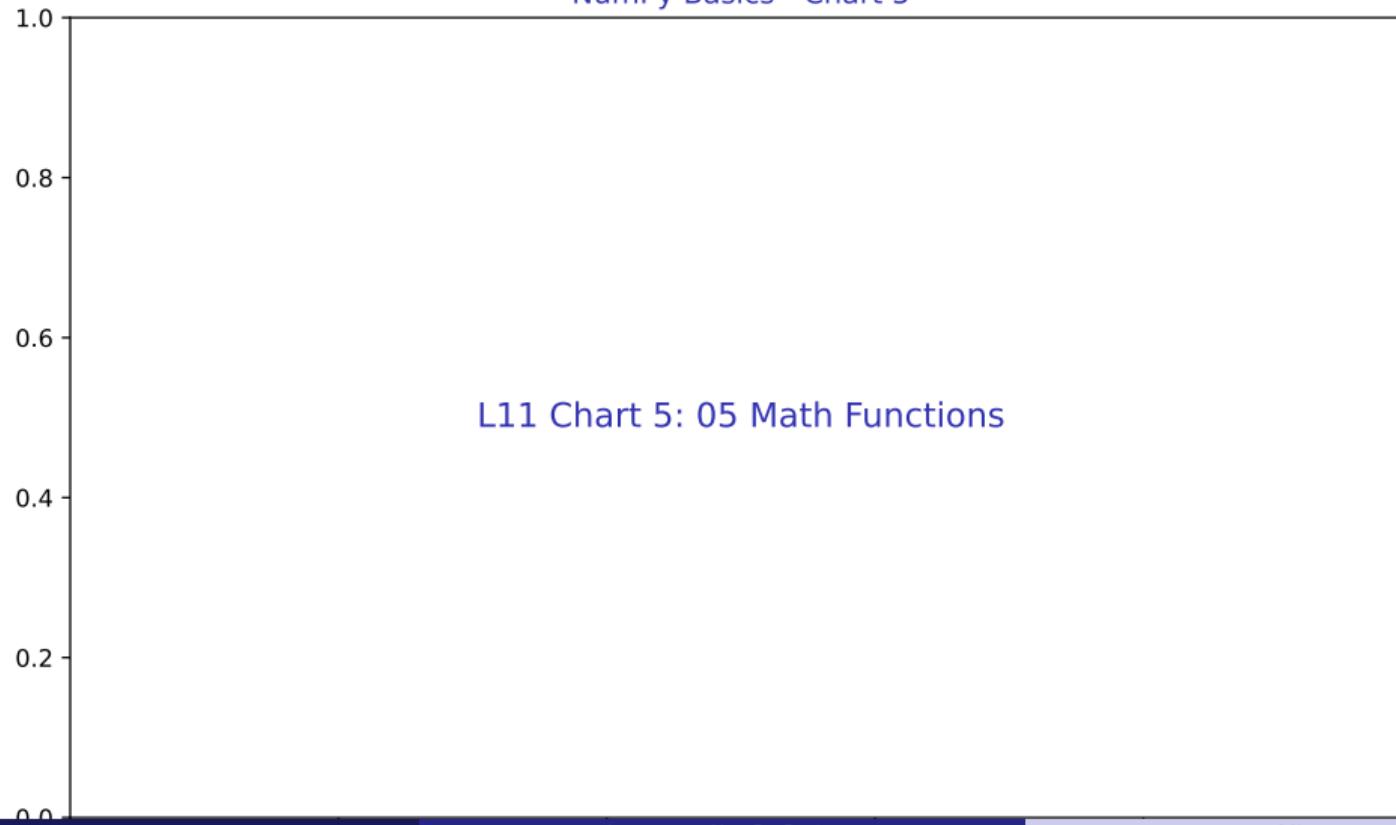
NumPy Basics - Chart 3



NumPy Basics - Chart 4

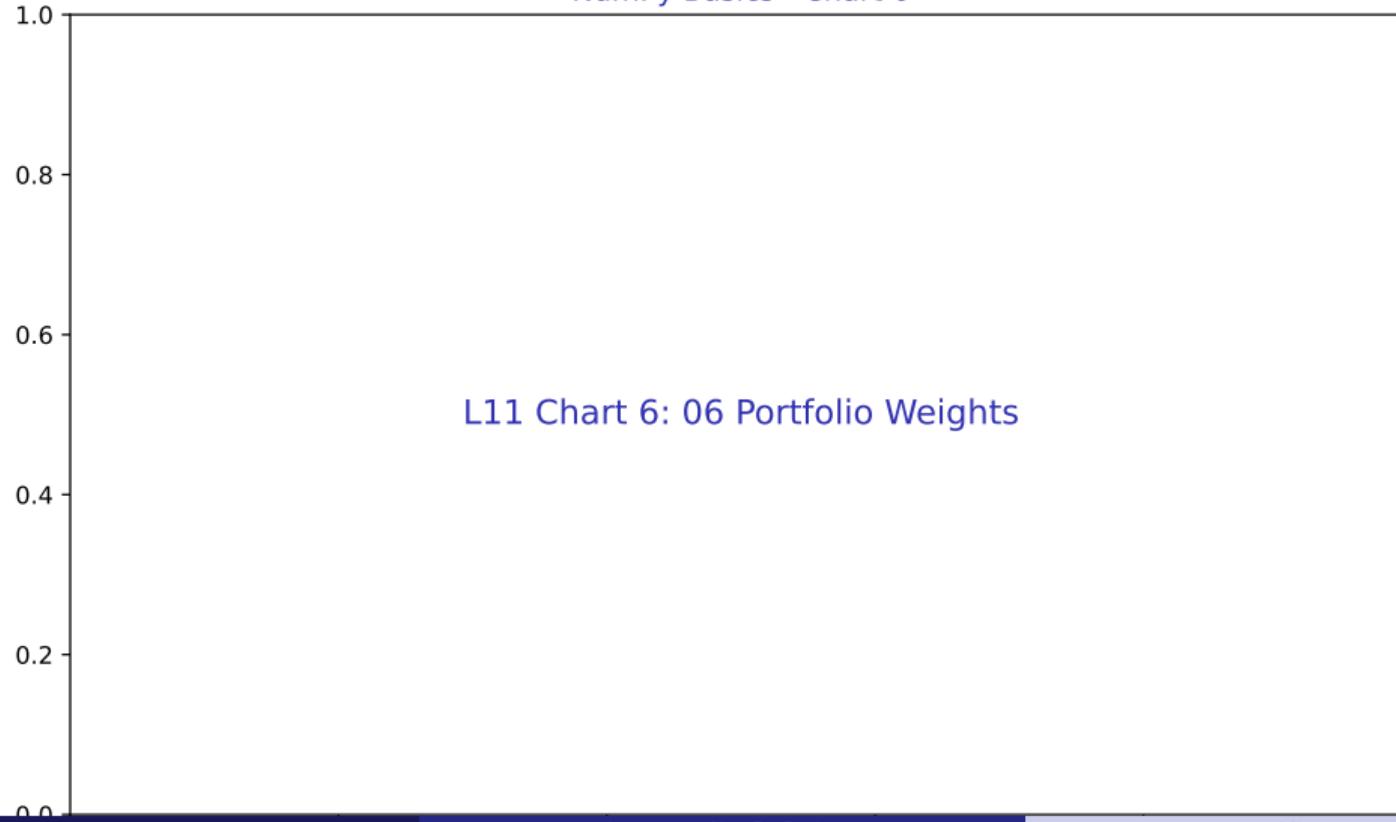


NumPy Basics - Chart 5

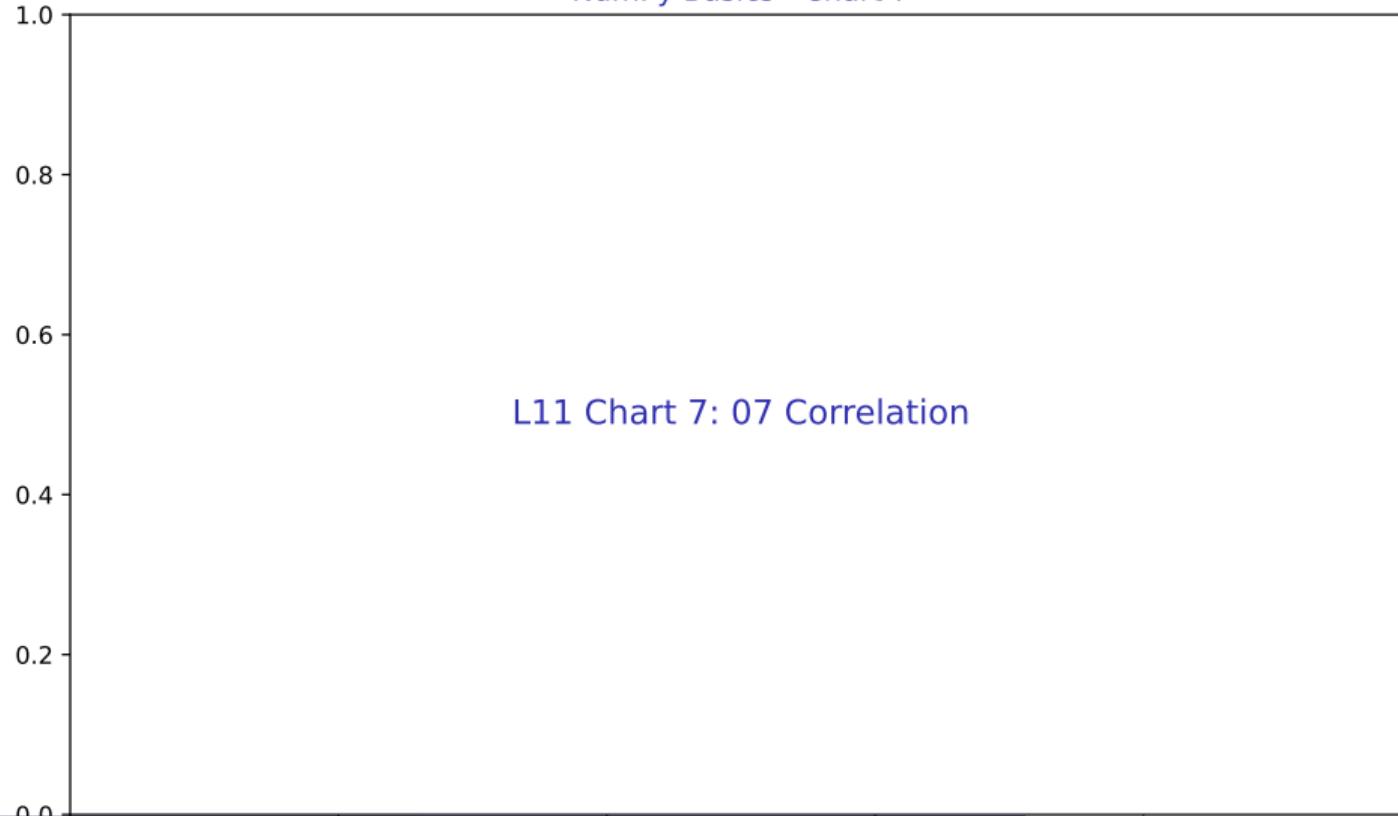


06 Portfolio Weights

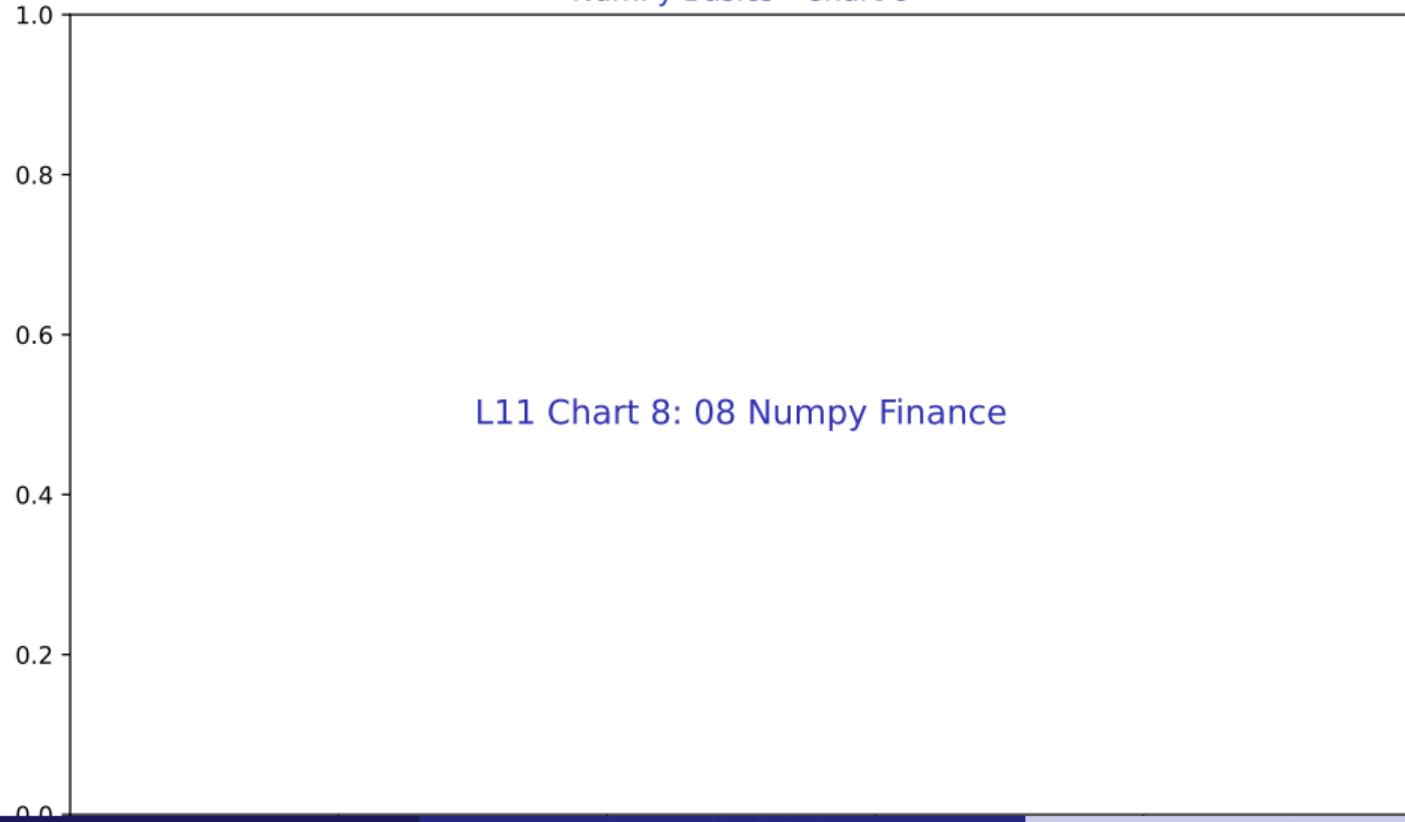
NumPy Basics - Chart 6



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

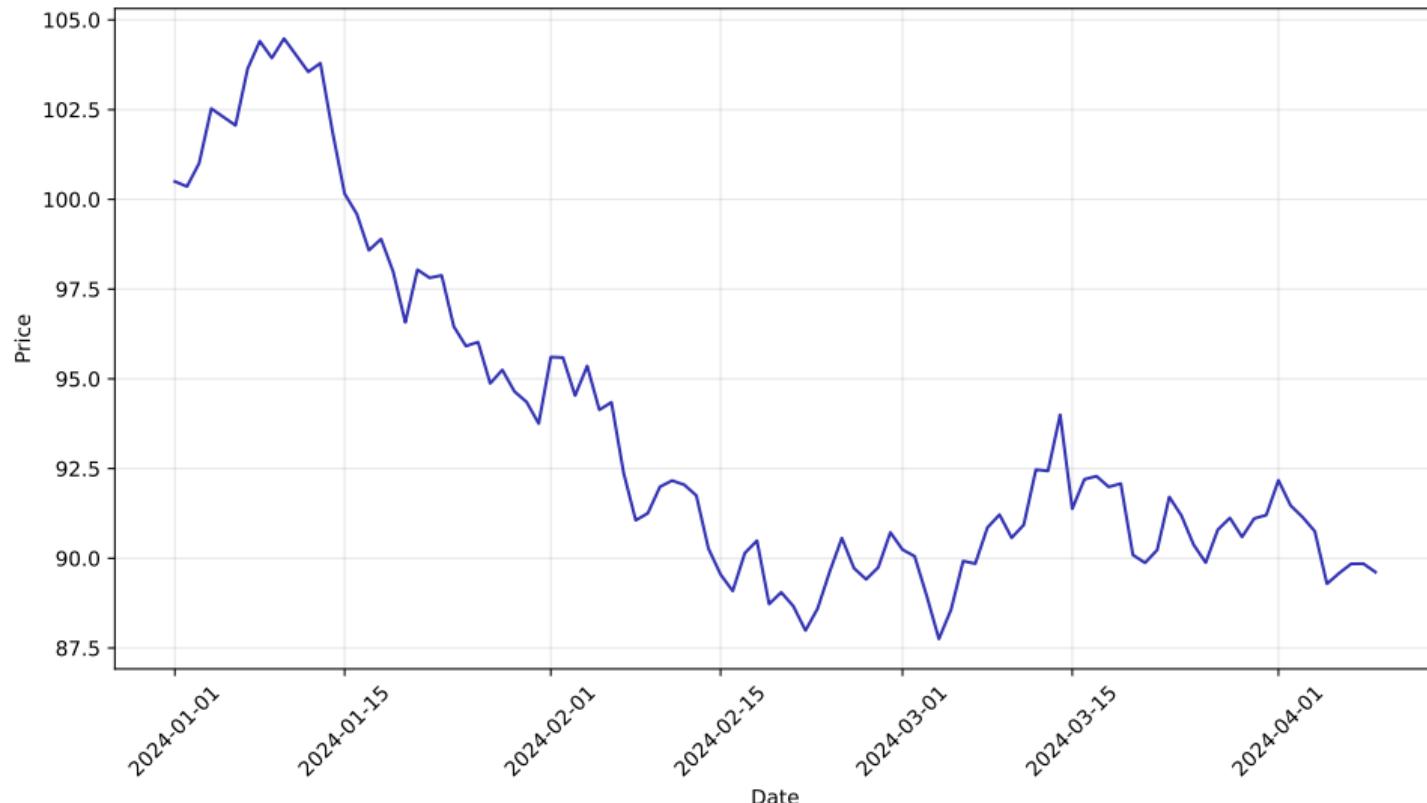
After this lesson, you will be able to:

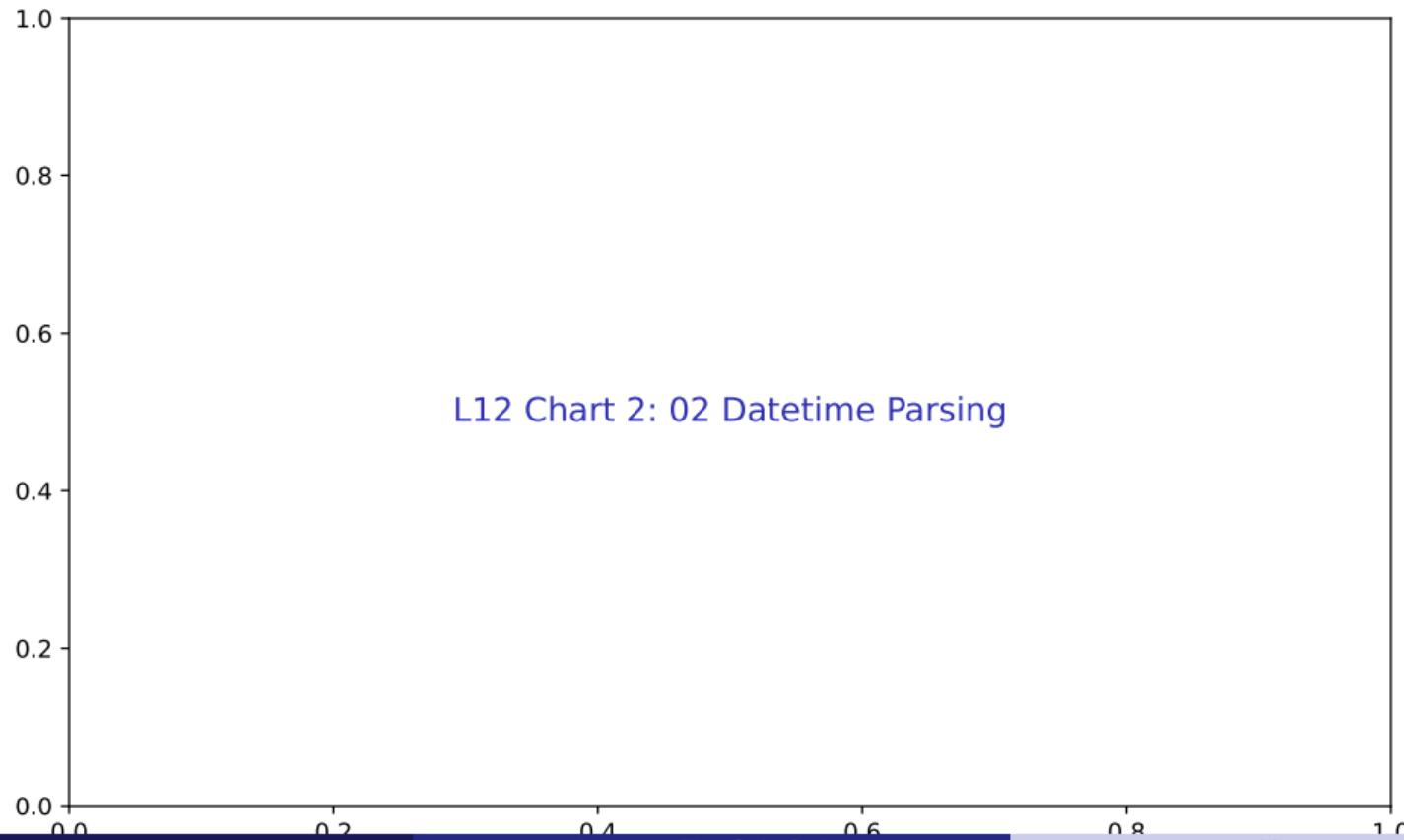
- DateTime index
- Resampling (daily to monthly)
- Rolling windows
- `shift()` and `pct_change()`

- Time series patterns in finance

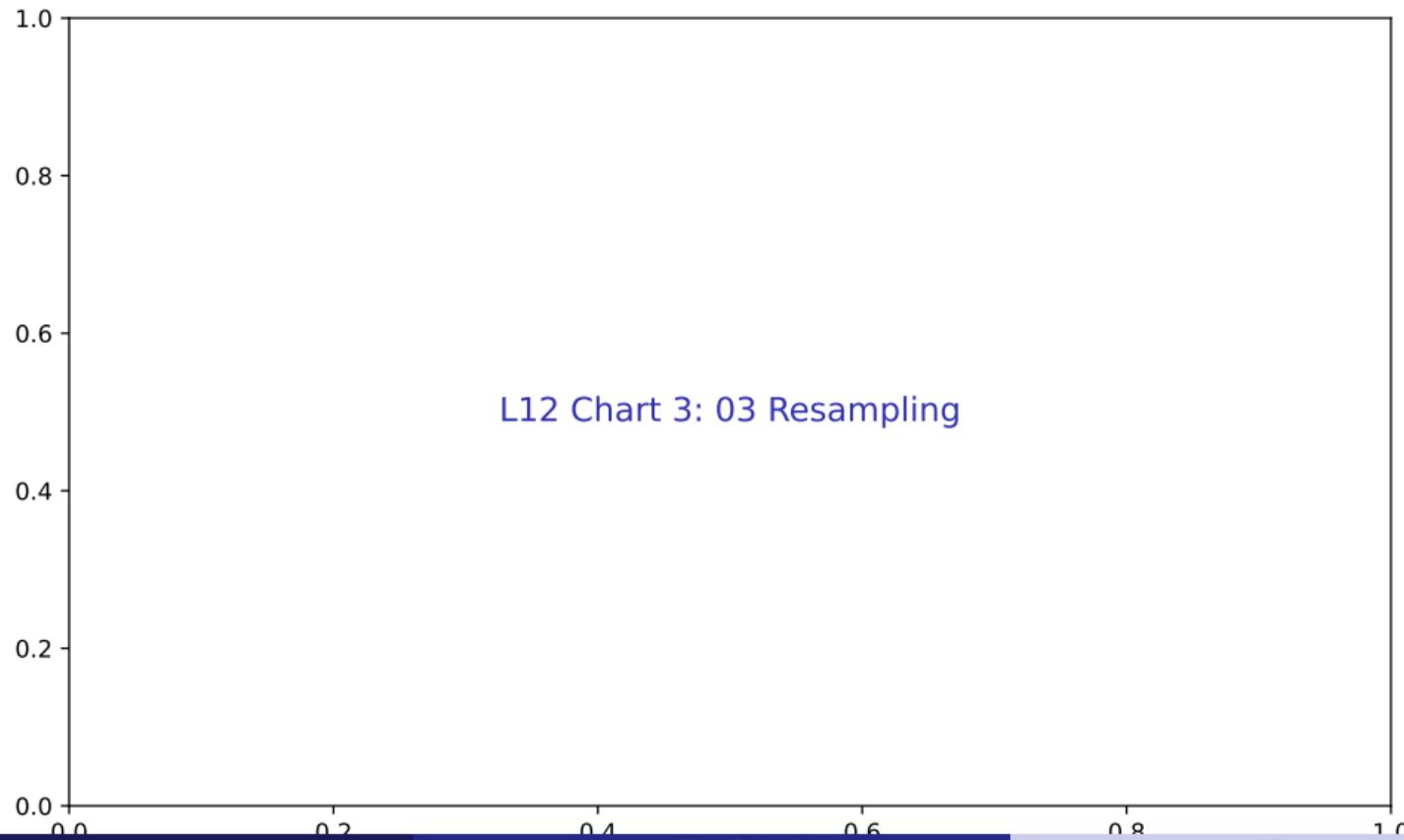
Finance application: Stock data processing and analysis

Financial Time Series

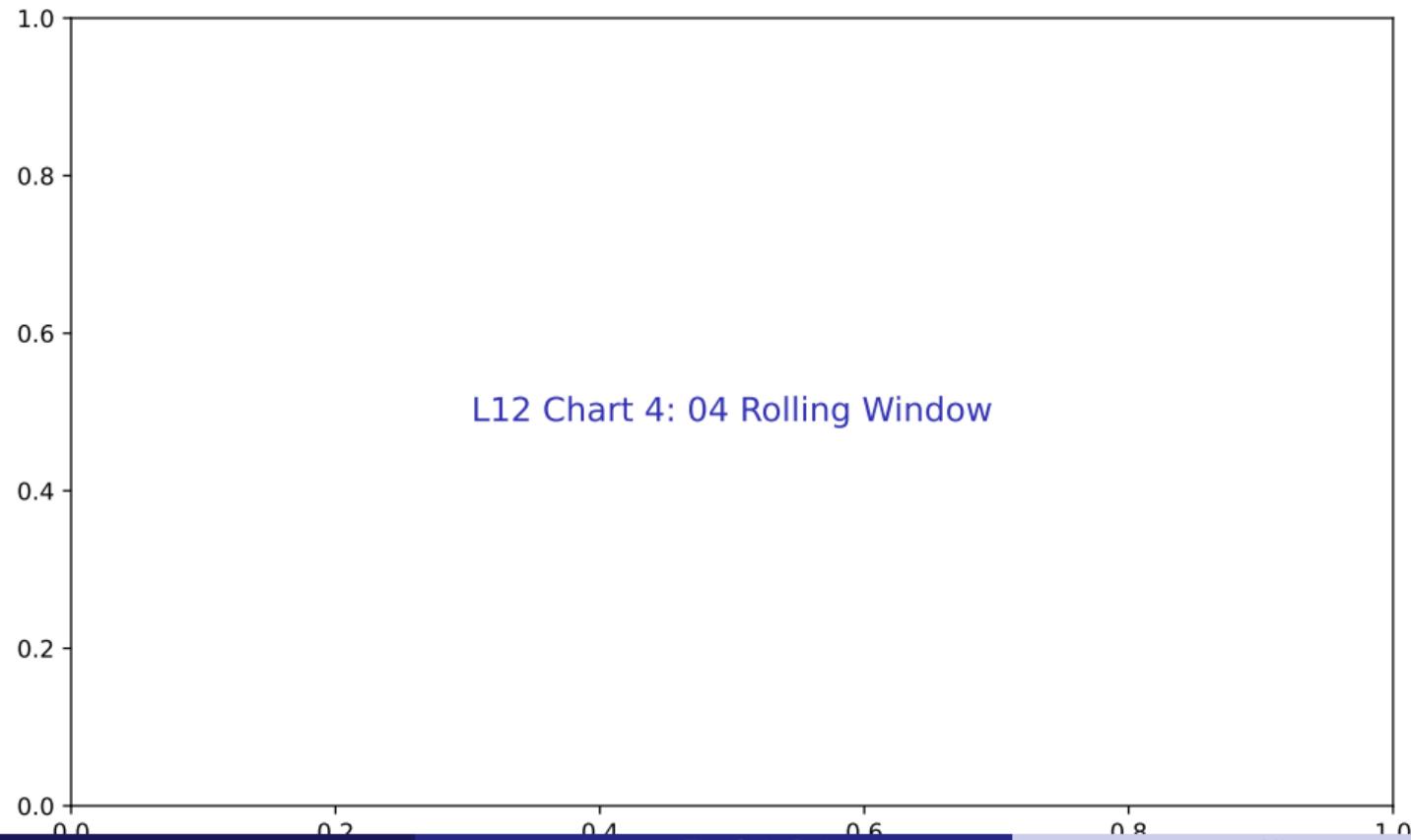




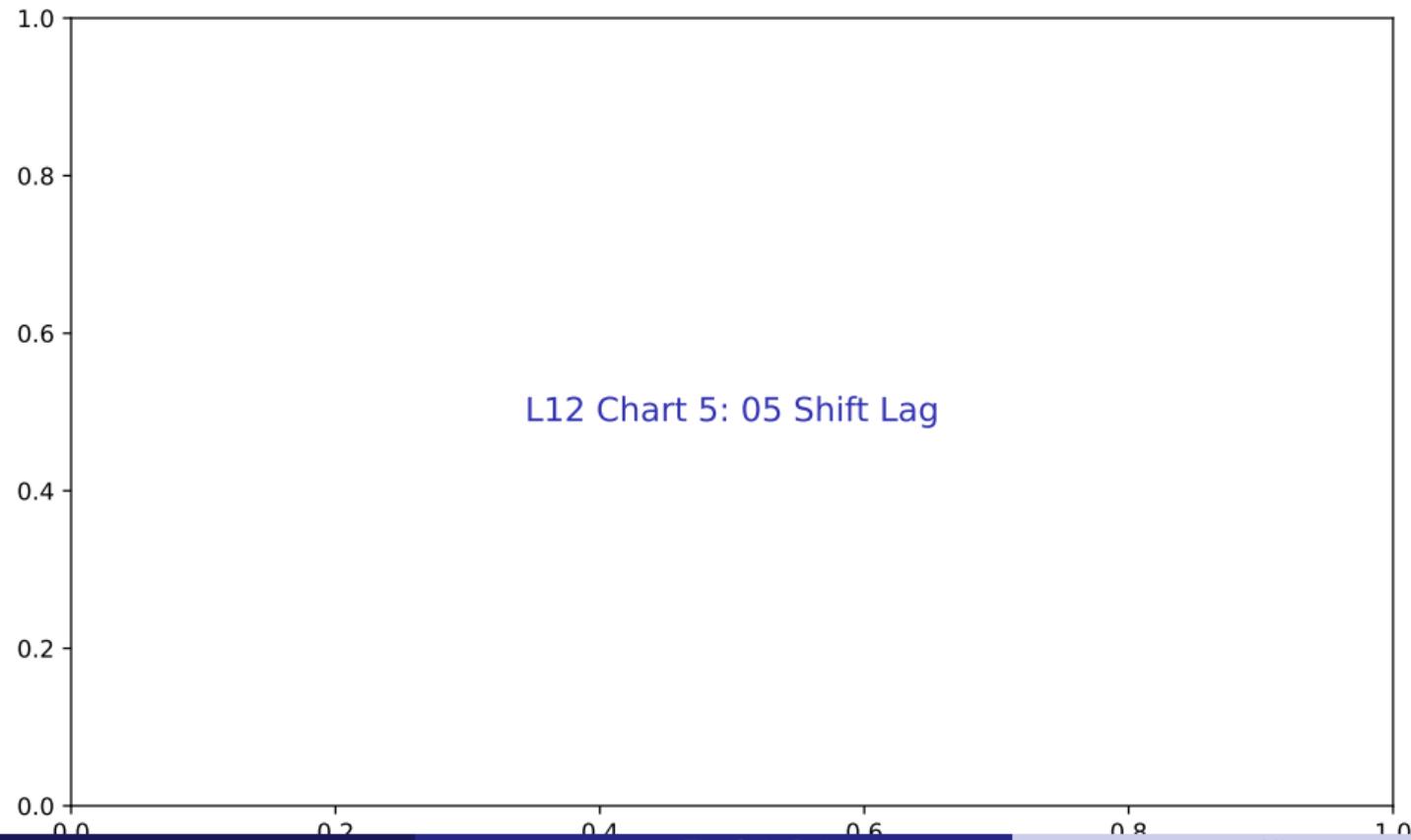
03 Resampling



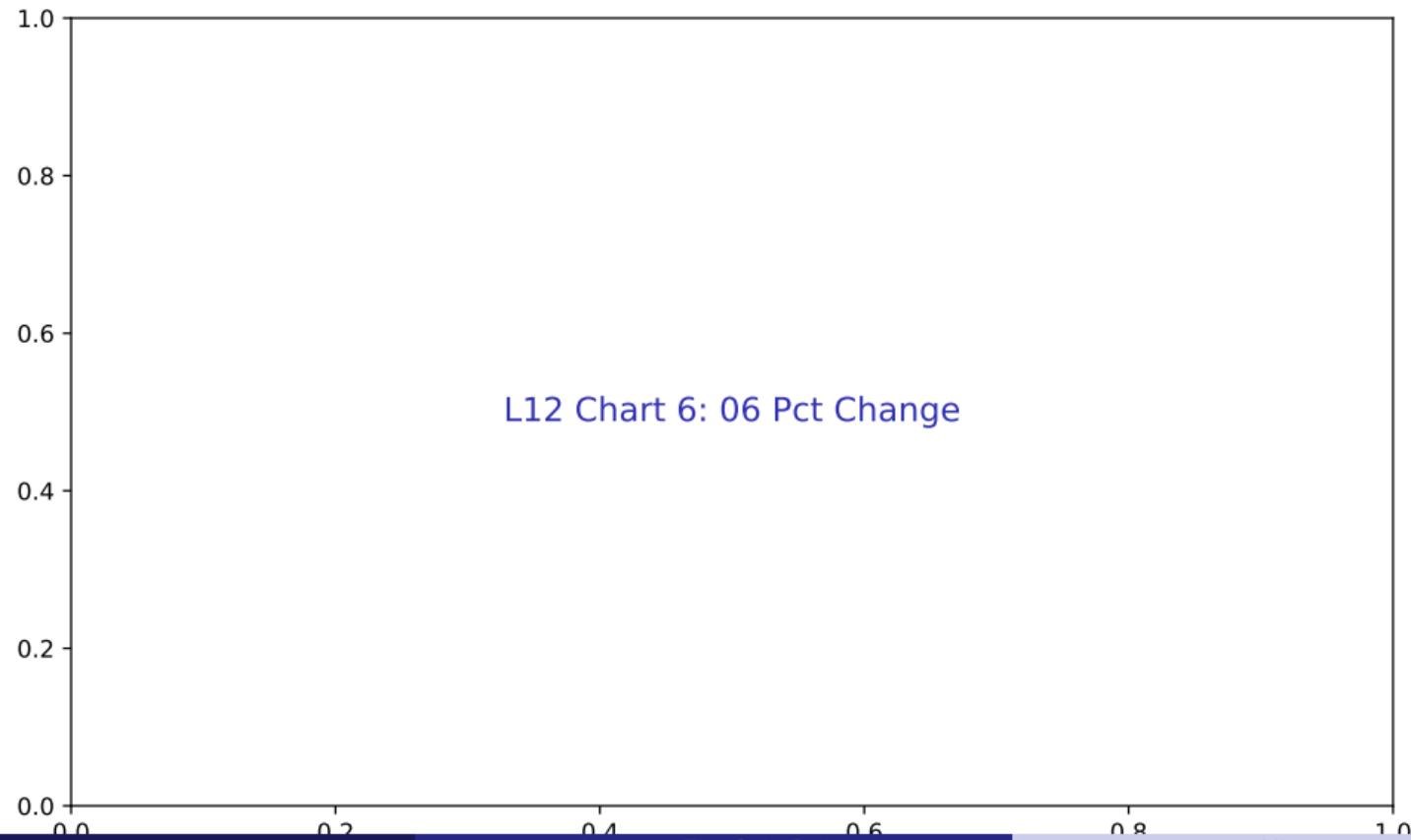
04 Rolling Window



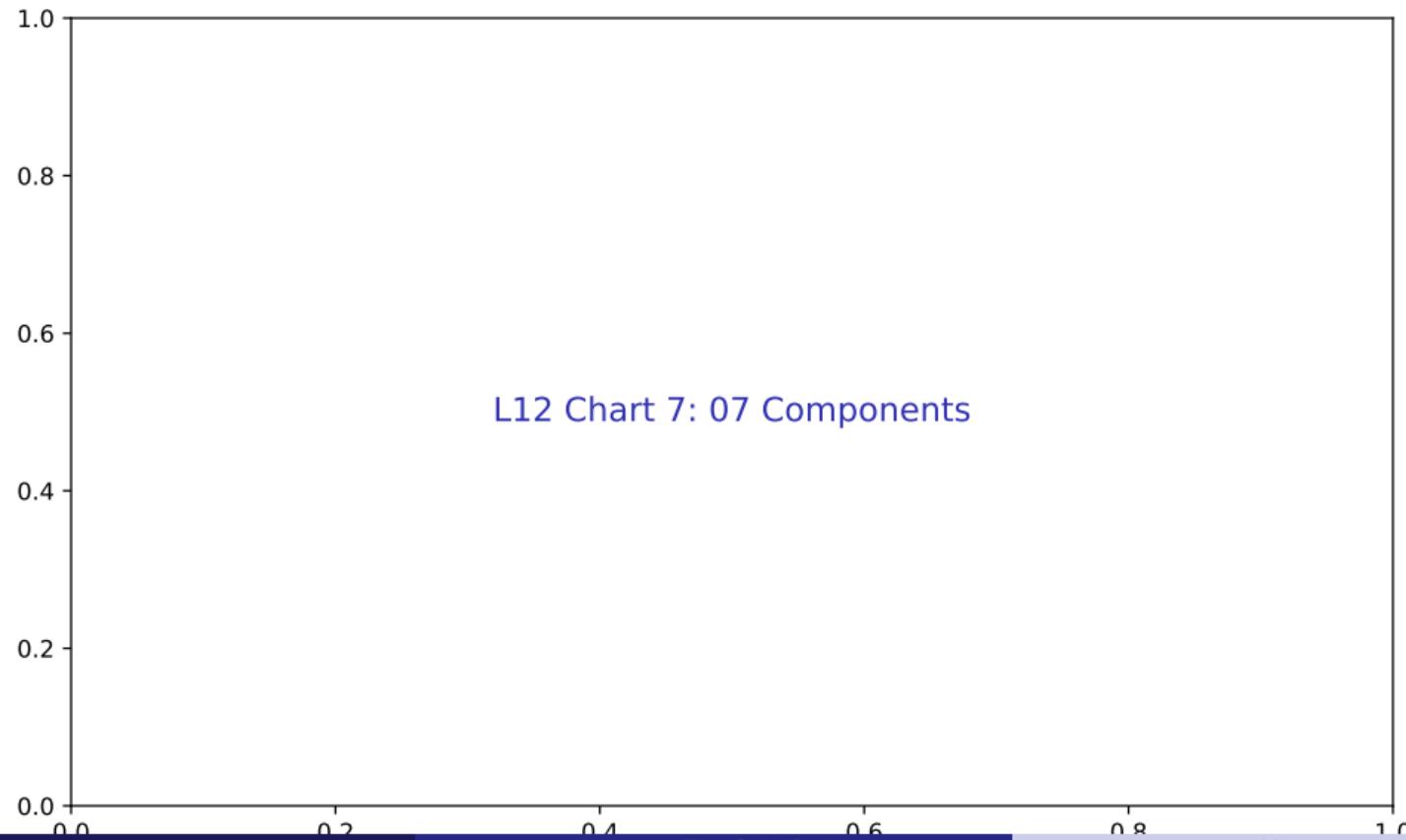
05 Shift Lag



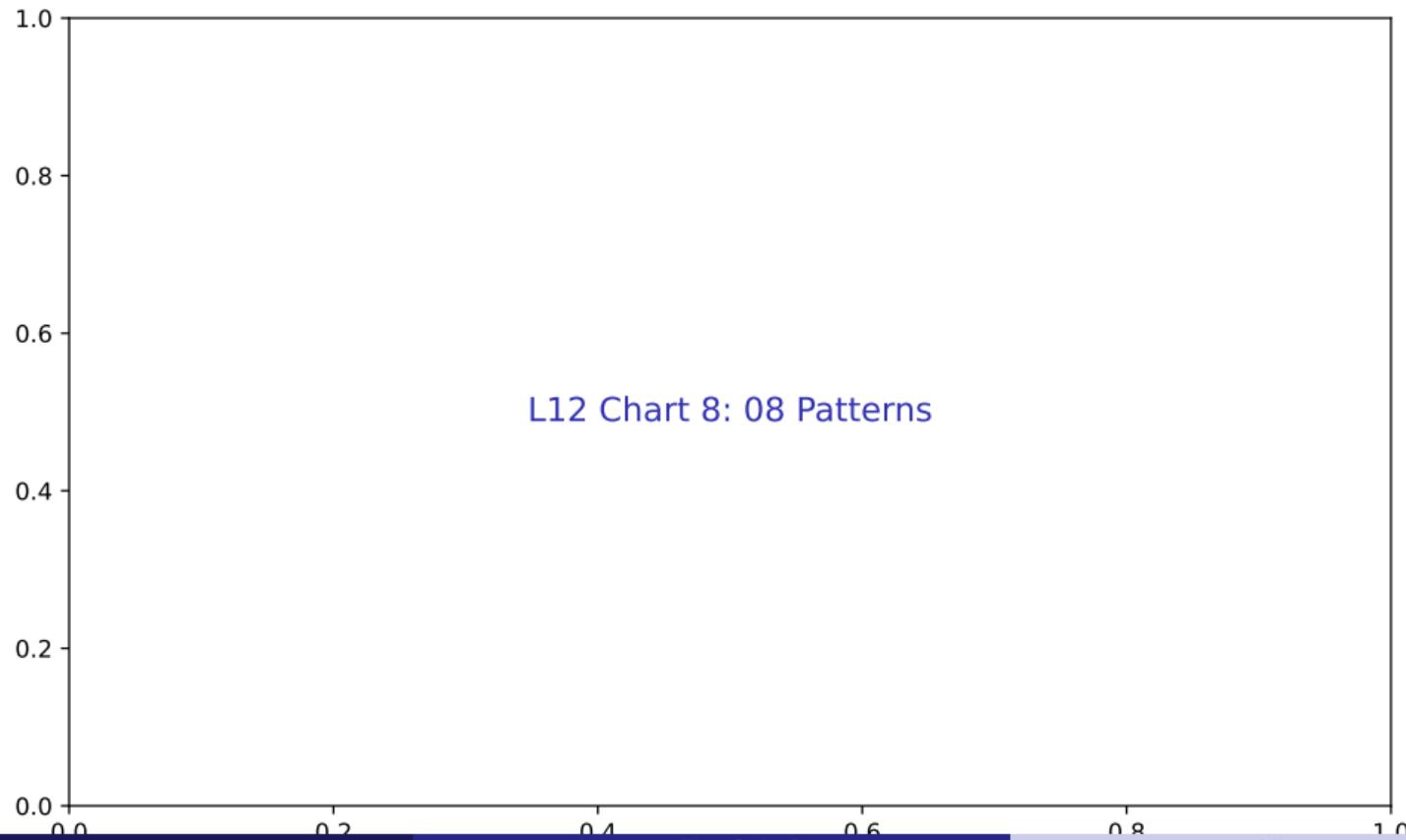
06 Pct Change



07 Components



08 Patterns



Key Takeaways:

- DateTime index
- Resampling (daily to monthly)
- Rolling windows
- `shift()` and `pct_change()`

- Time series patterns in finance

Practice: Apply these concepts to the stock price dataset.

Lesson 13: Descriptive Statistics

Data Science with Python – BSc Course

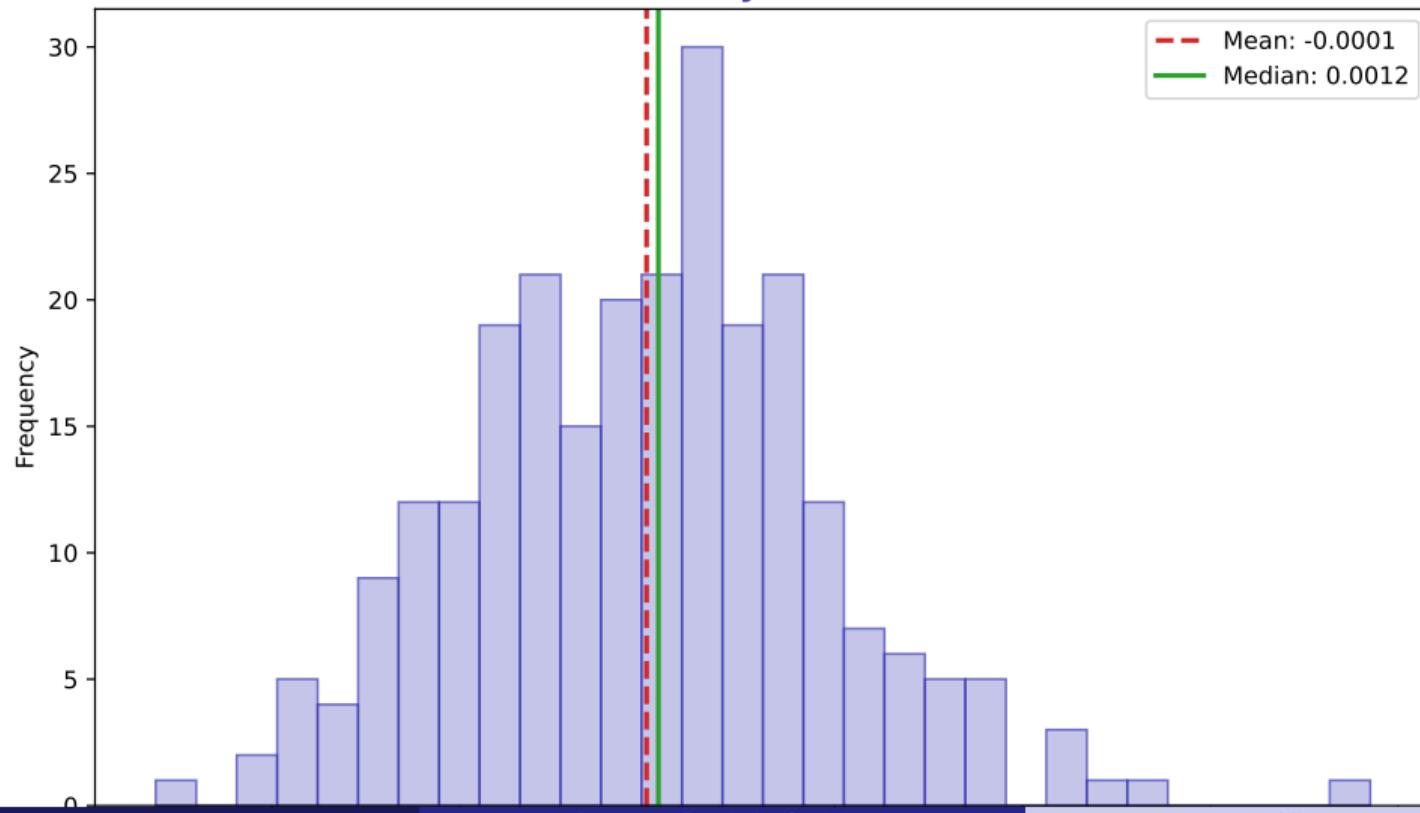
45 Minutes

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

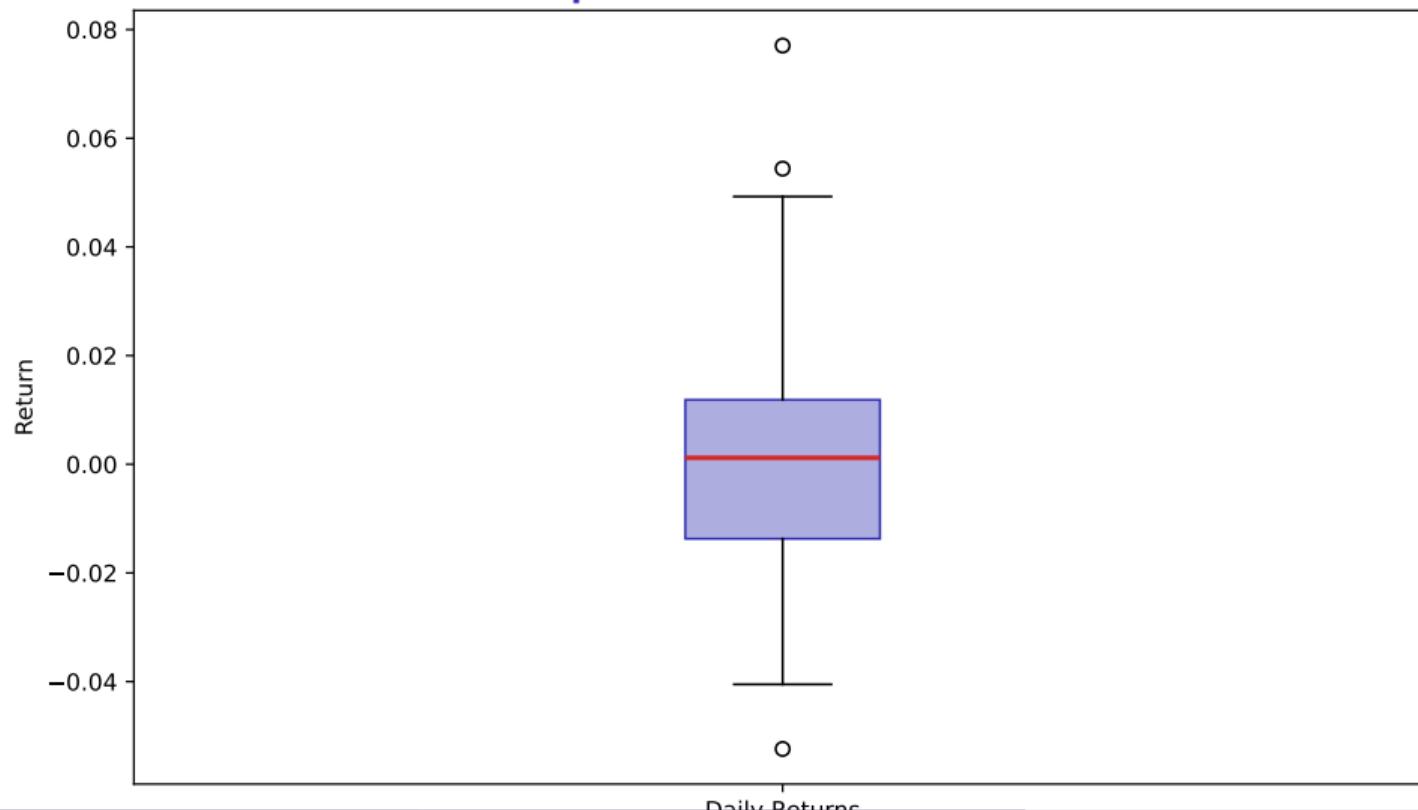
Finance application: Statistical analysis of market data

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

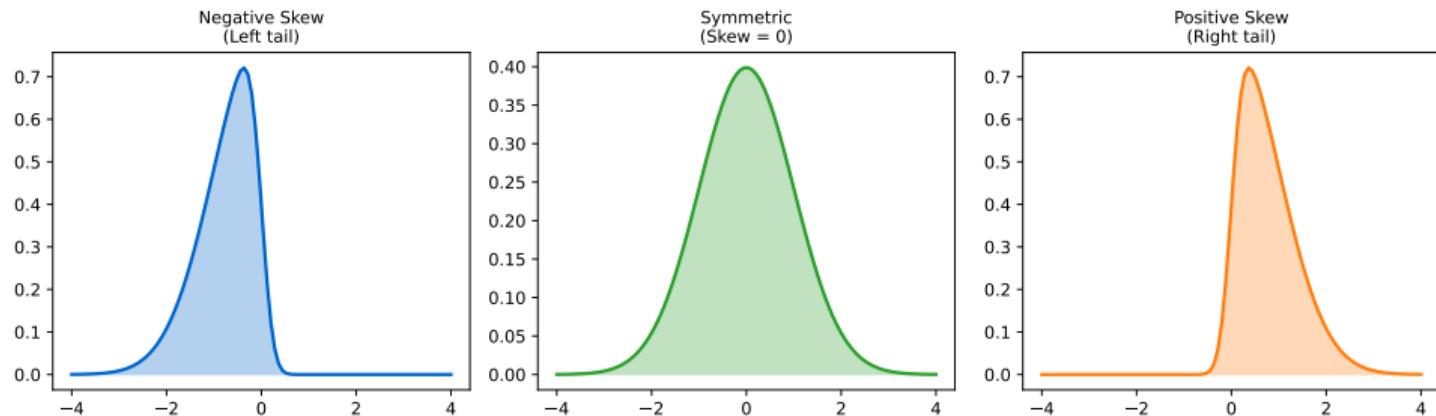
Q3 (75th): 0.0119

Maximum: 0.0771

IQR: 0.0256

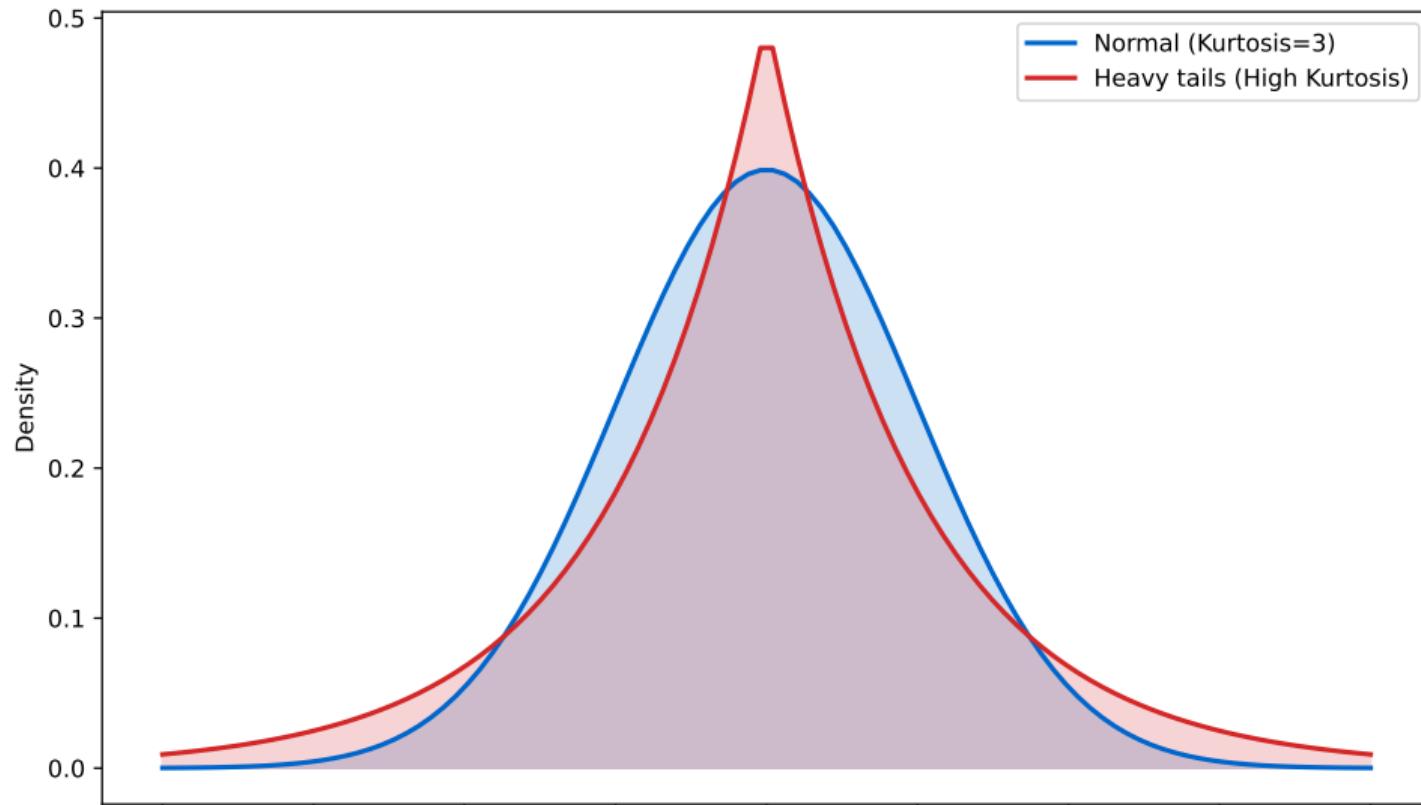
Skewness

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**

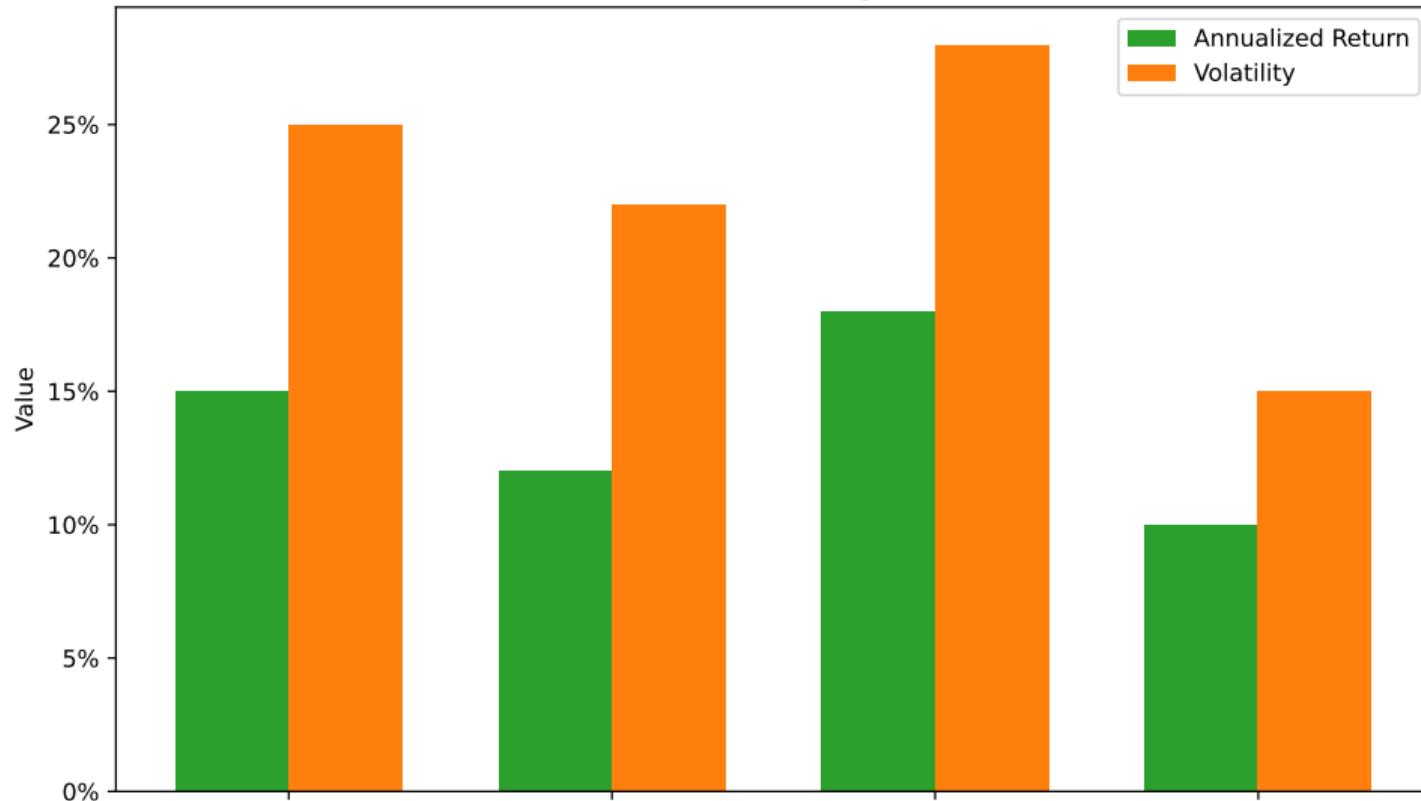
(ret - rf) / vol

-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

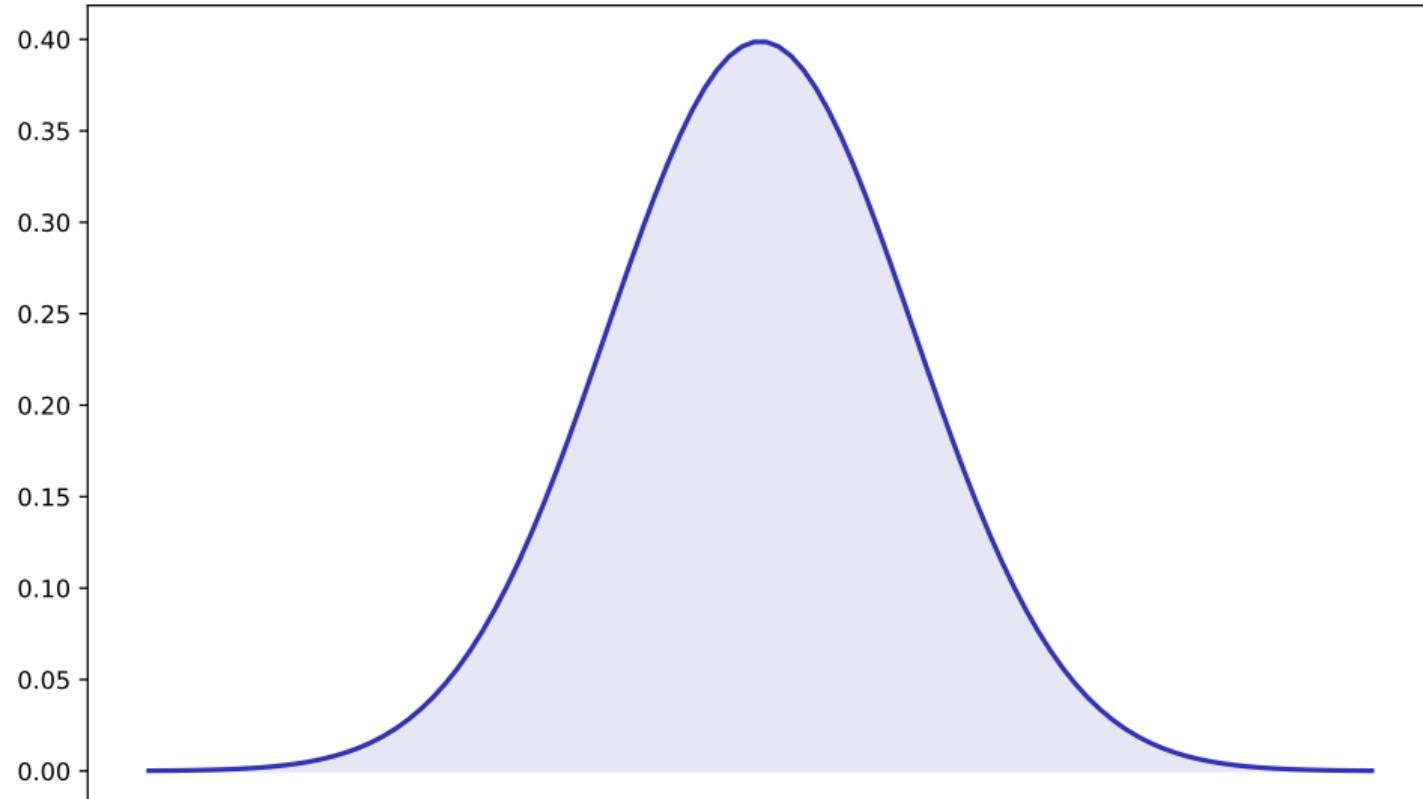
45 Minutes

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

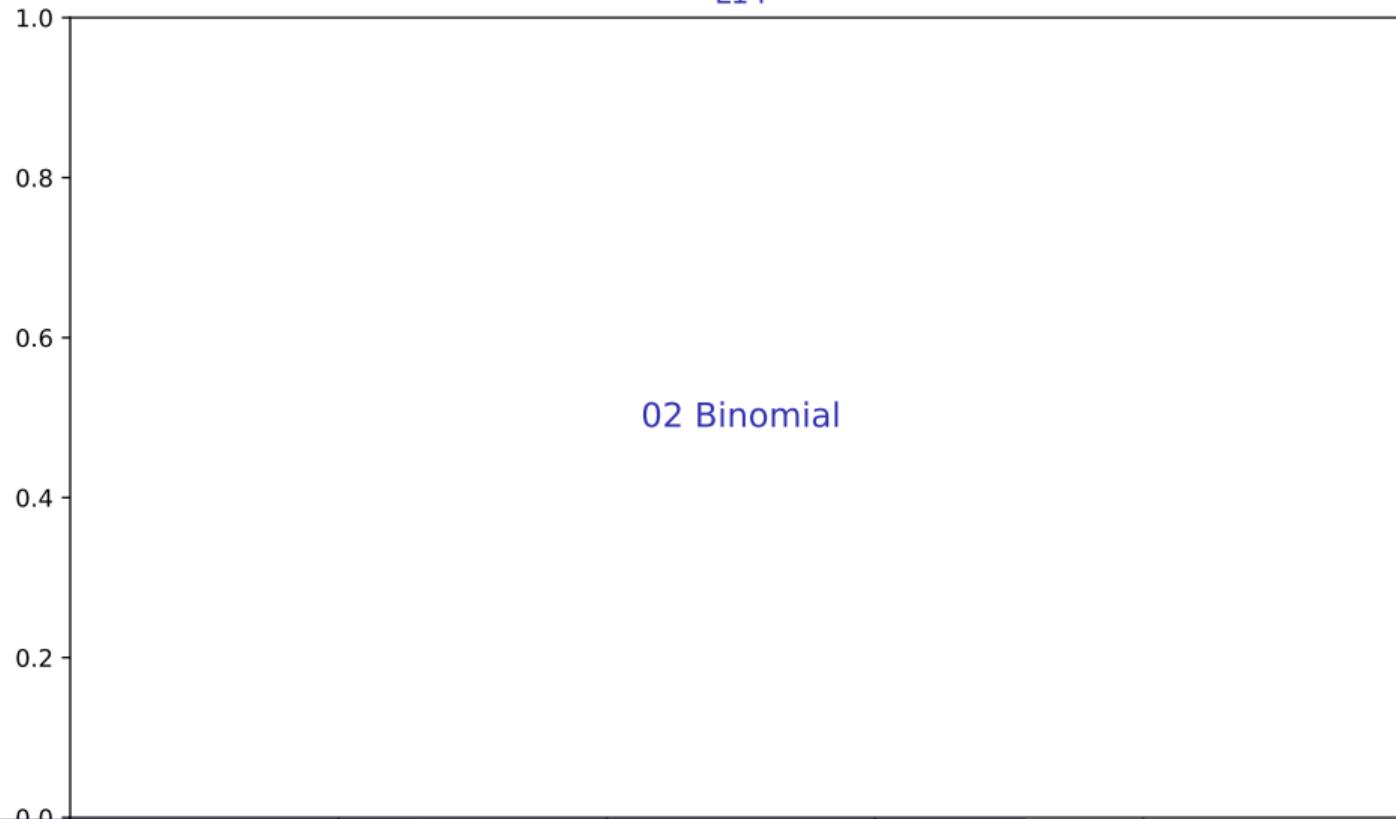
Finance application: Statistical analysis of market data

01 Normal Distribution



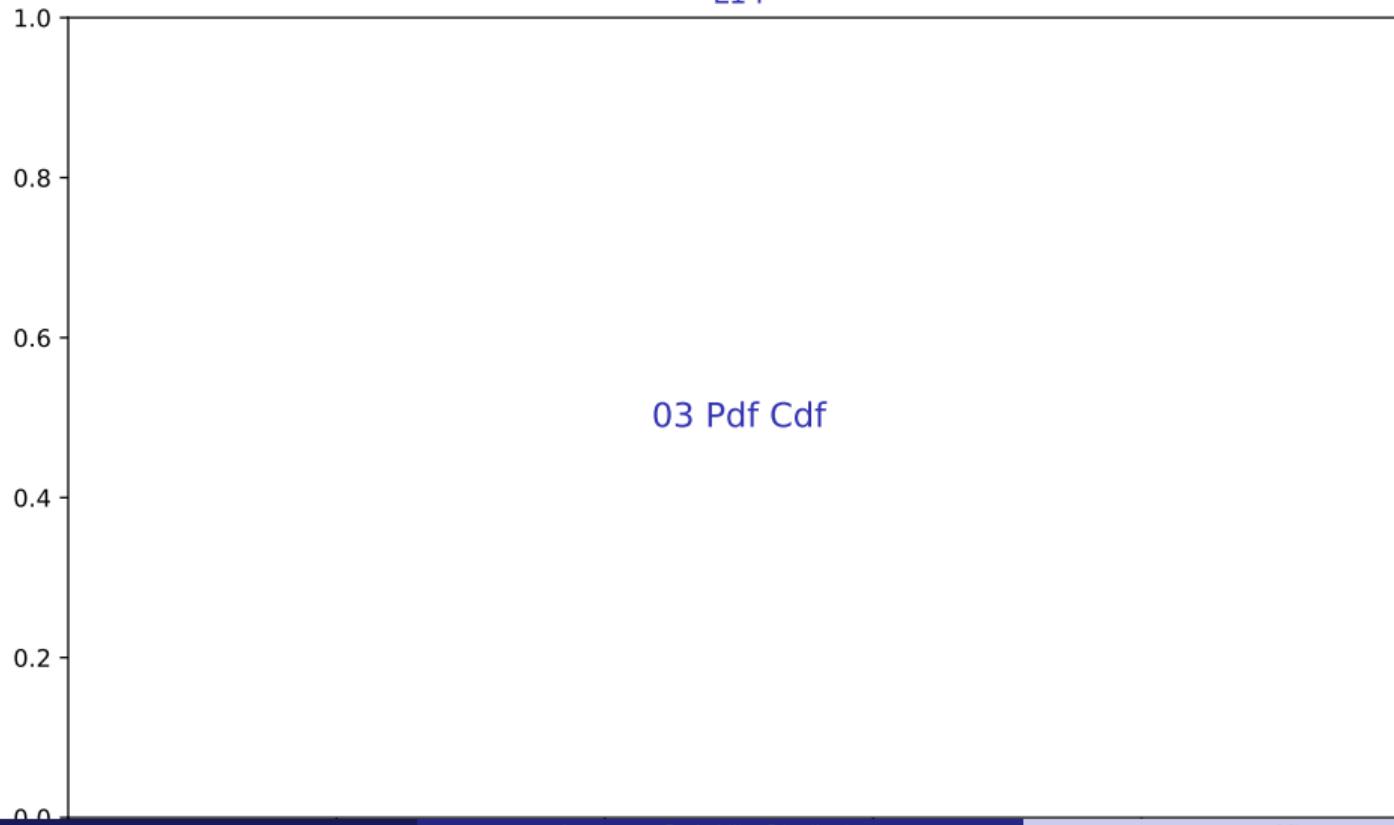
L14

02 Binomial



L14

03 Pdf Cdf



L14

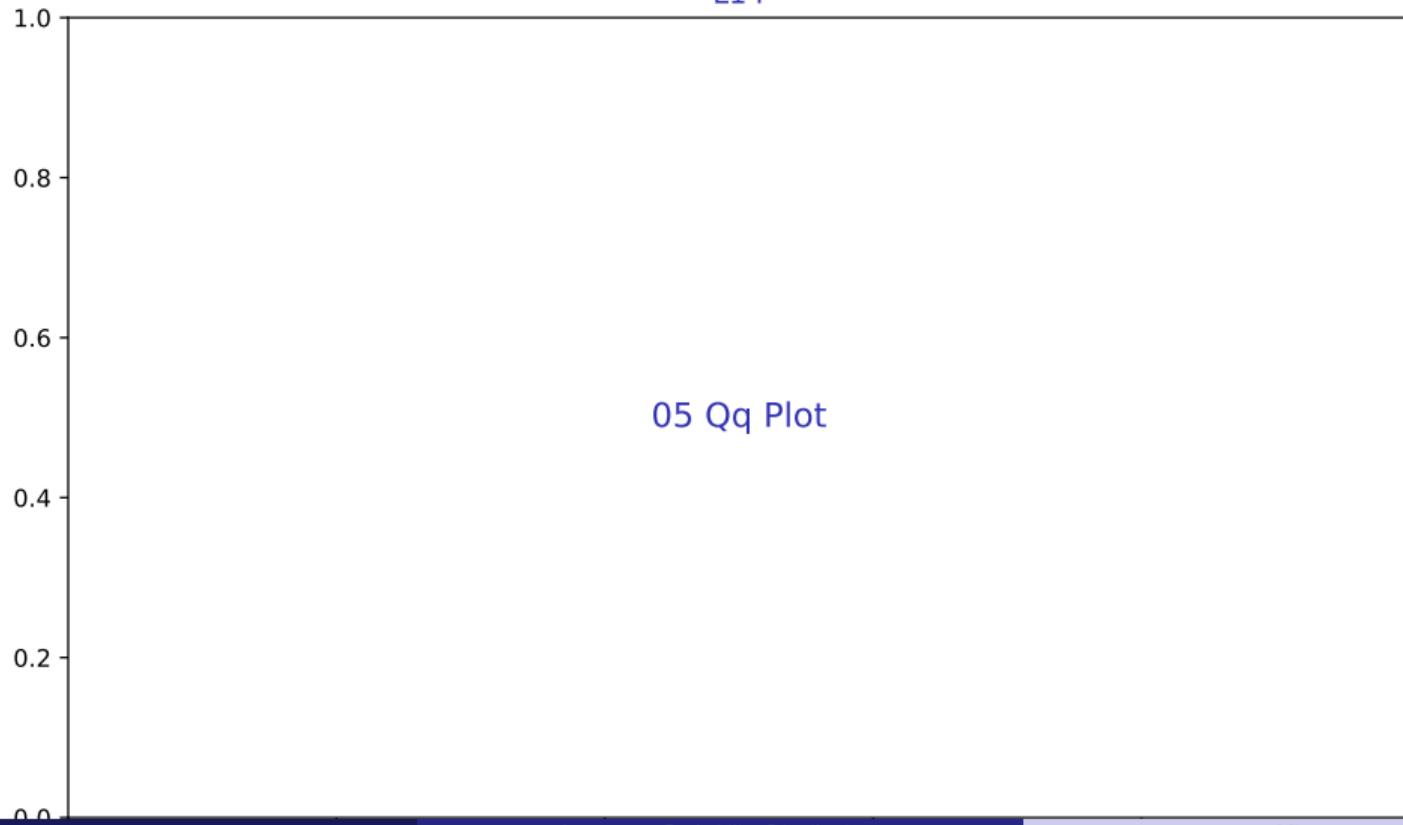
04 Stock Returns



Qq Plot

L14

05 Qq Plot

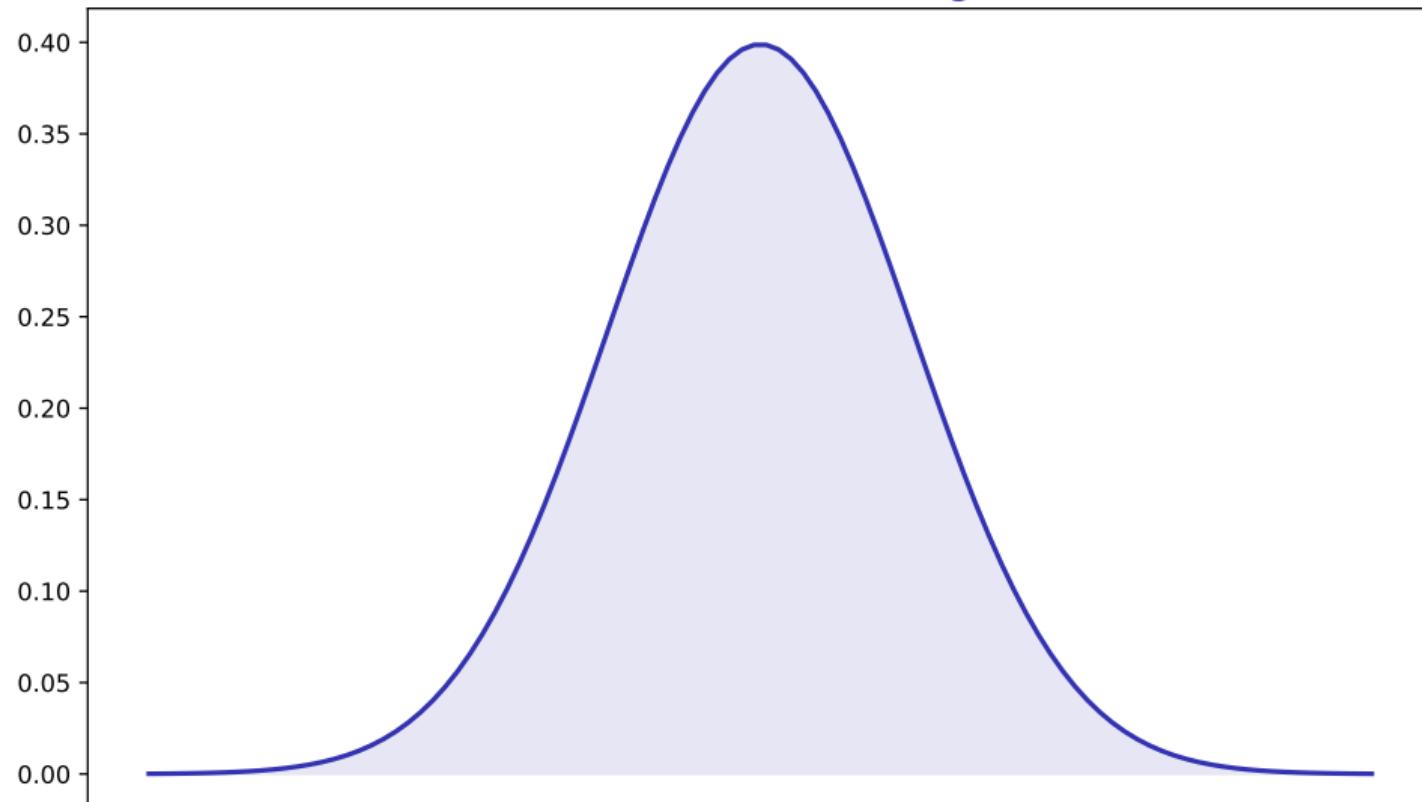


L14

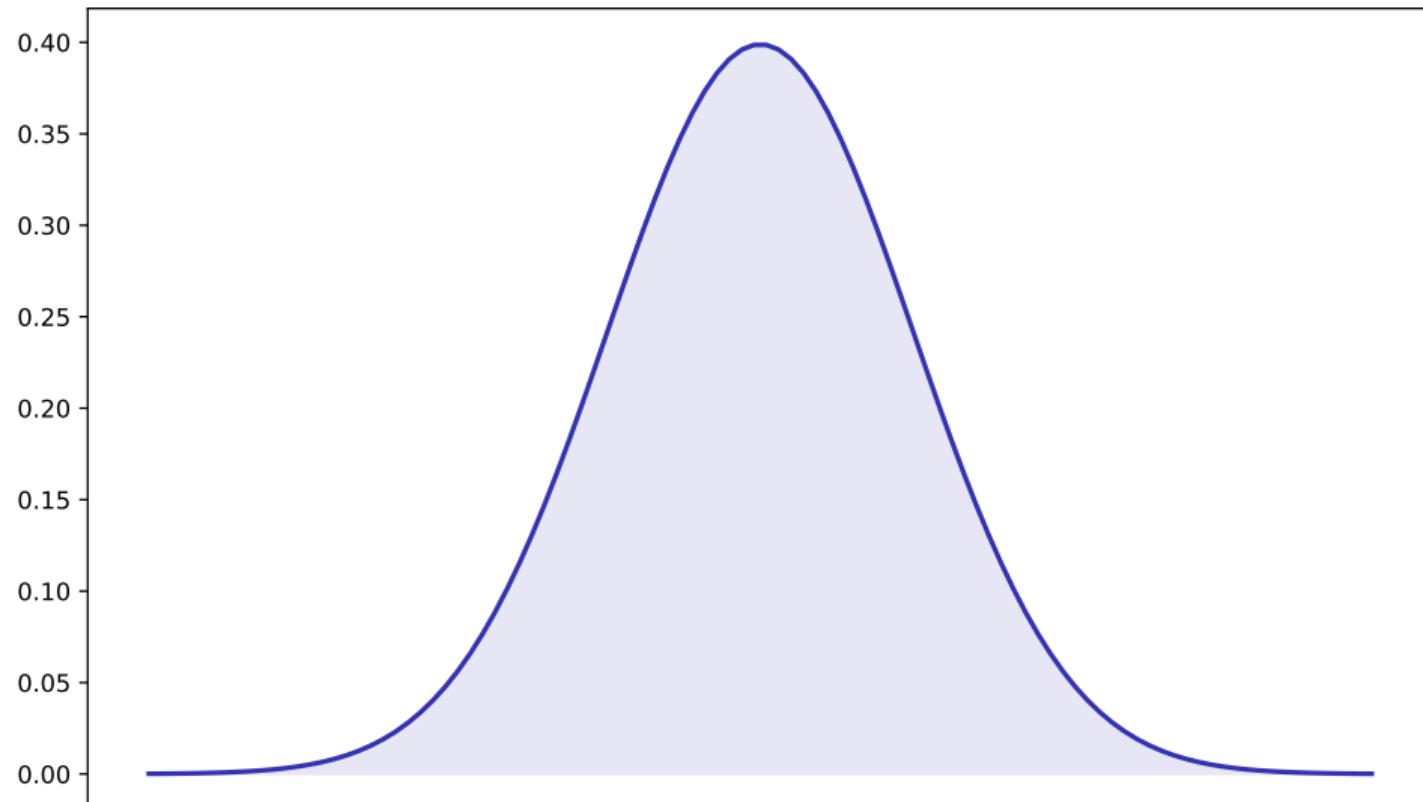
06 Fat Tails



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

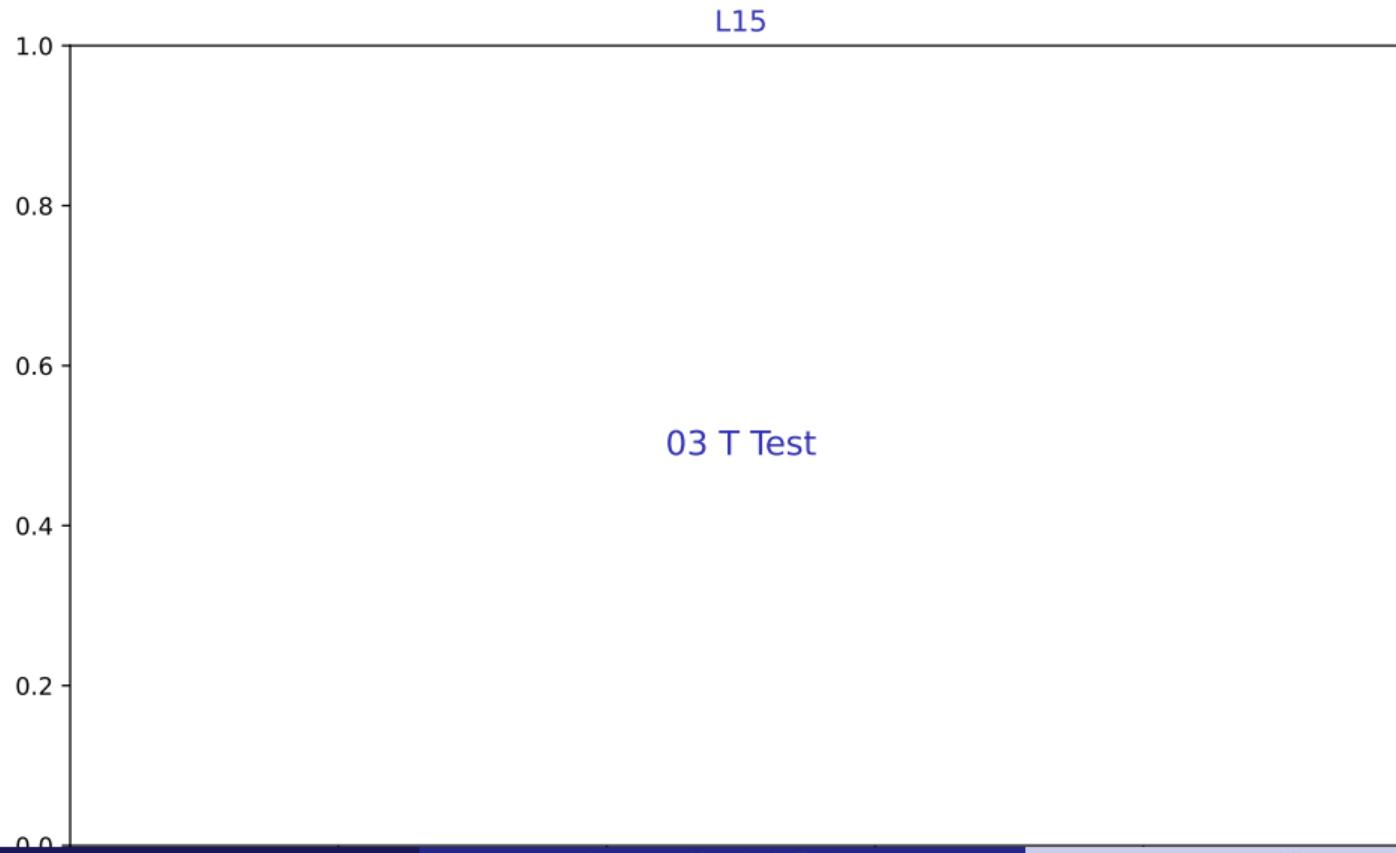
L15

01 Hypothesis Concept

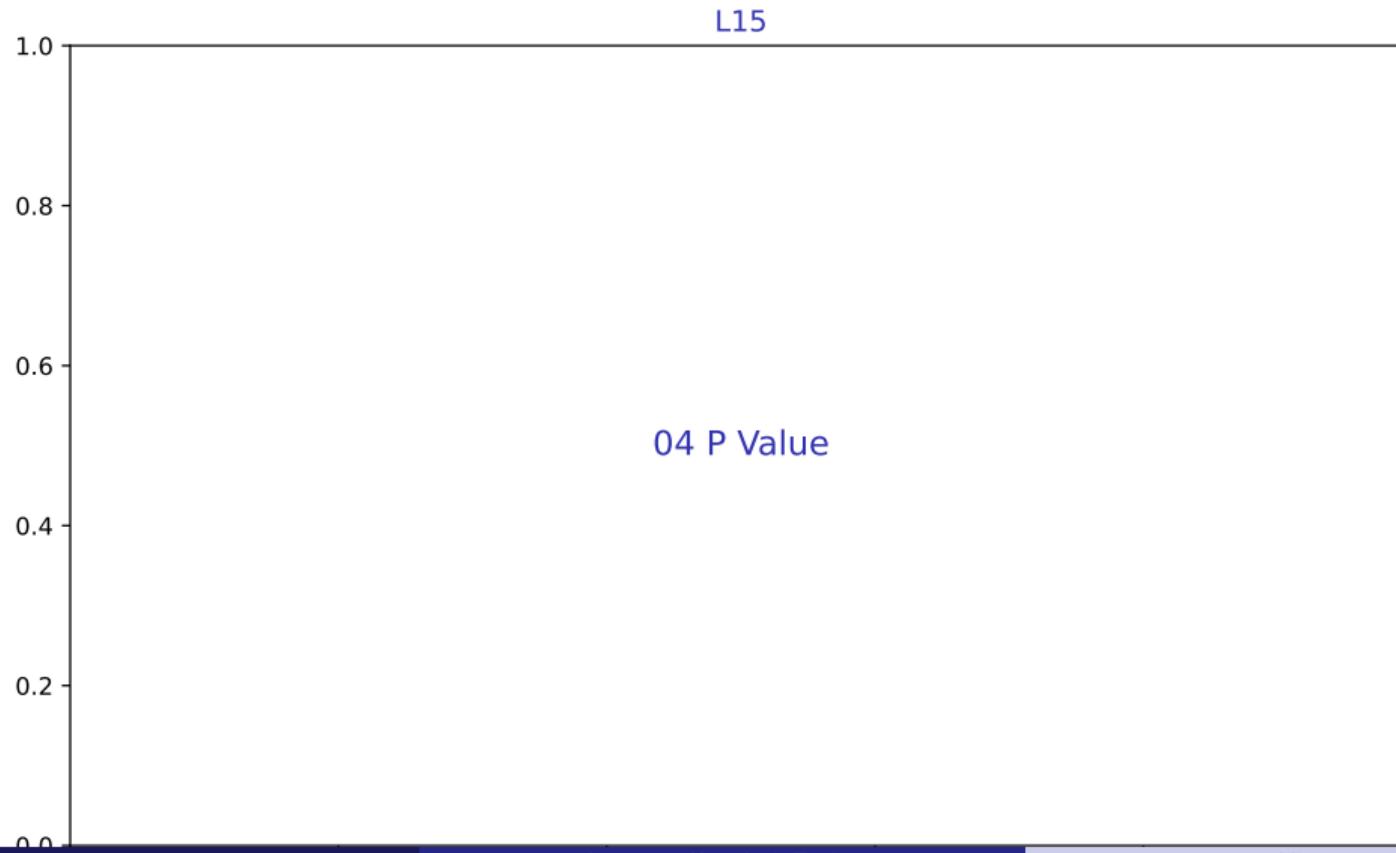
Null Alternative



T Test

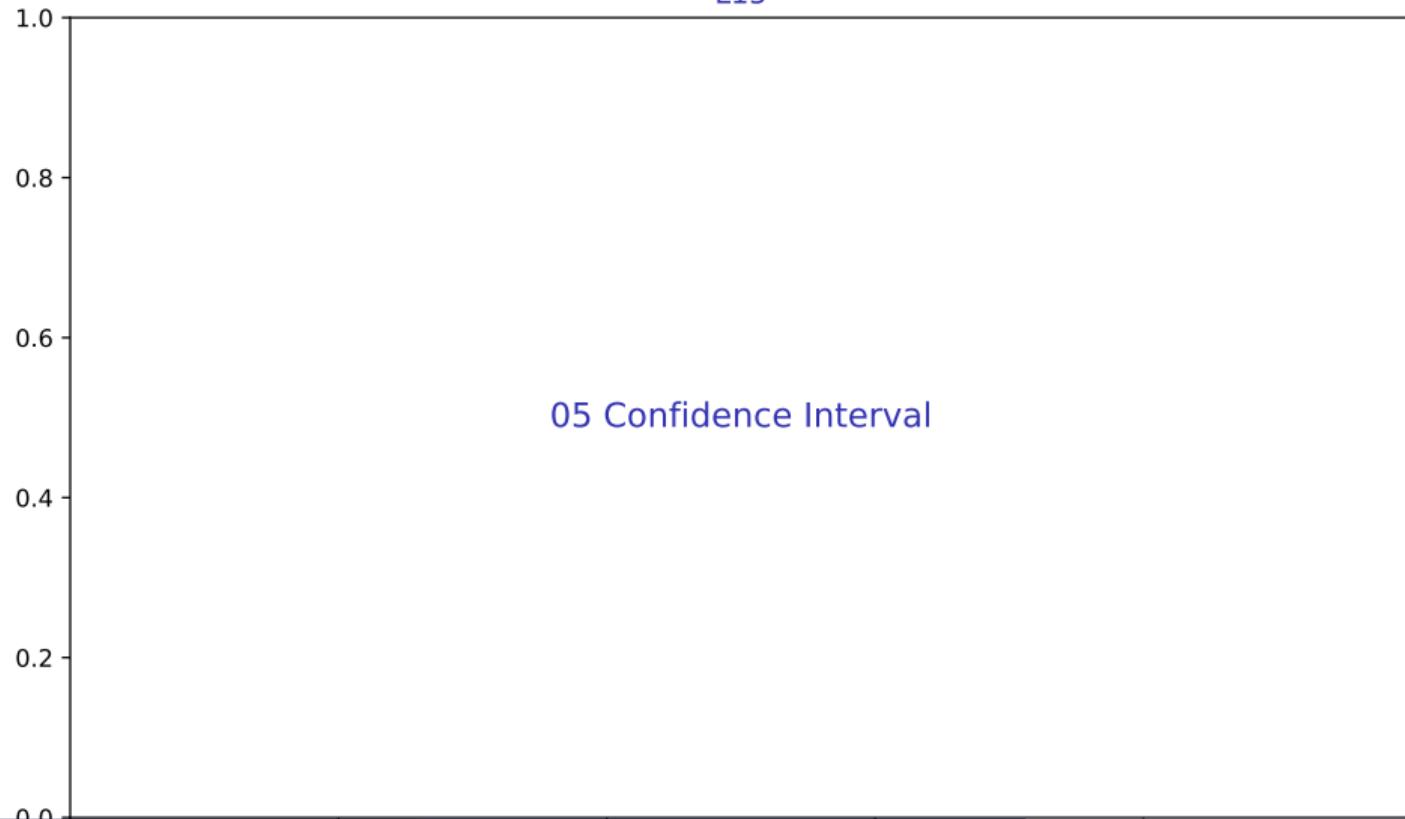


P Value

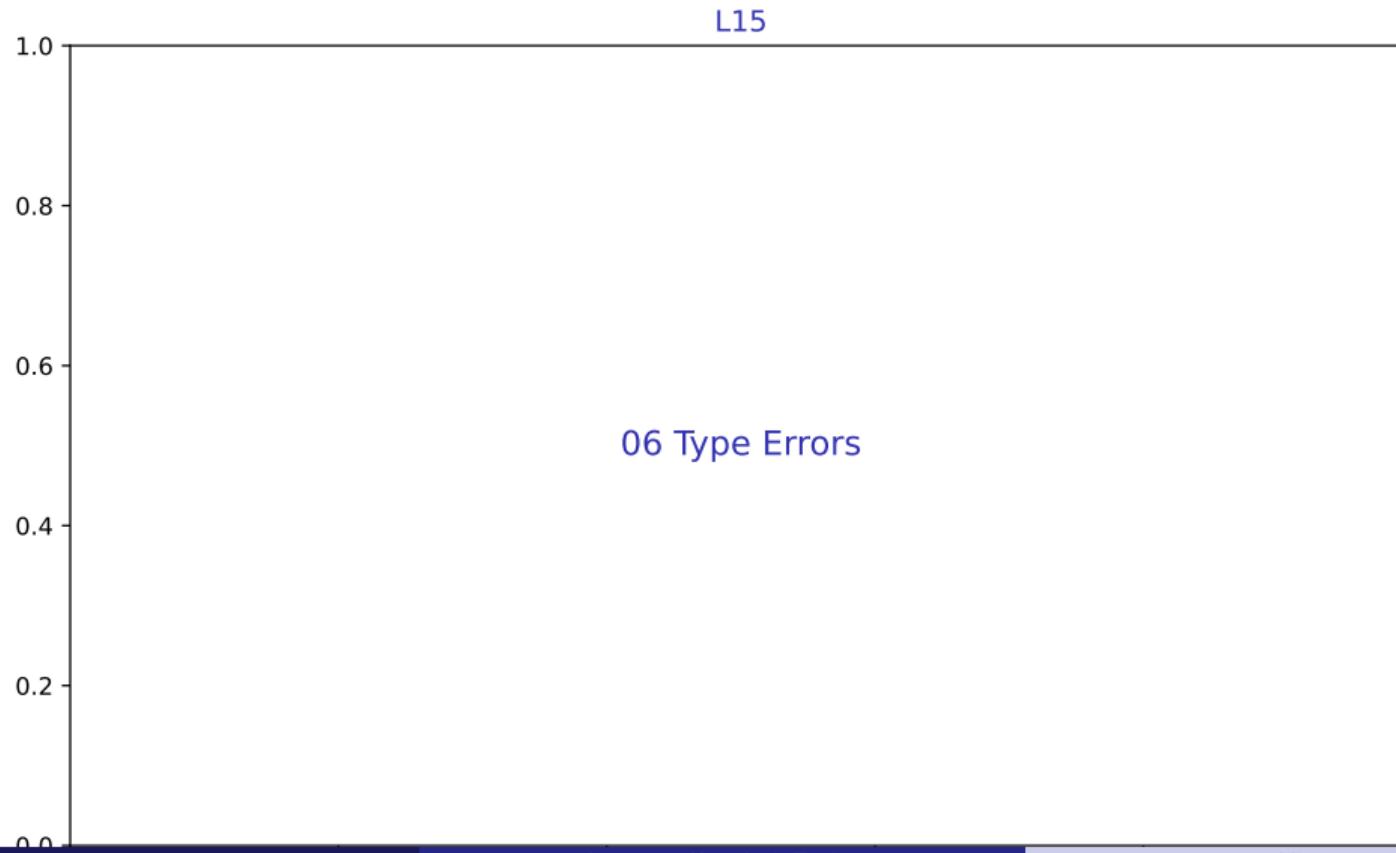


Confidence Interval

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

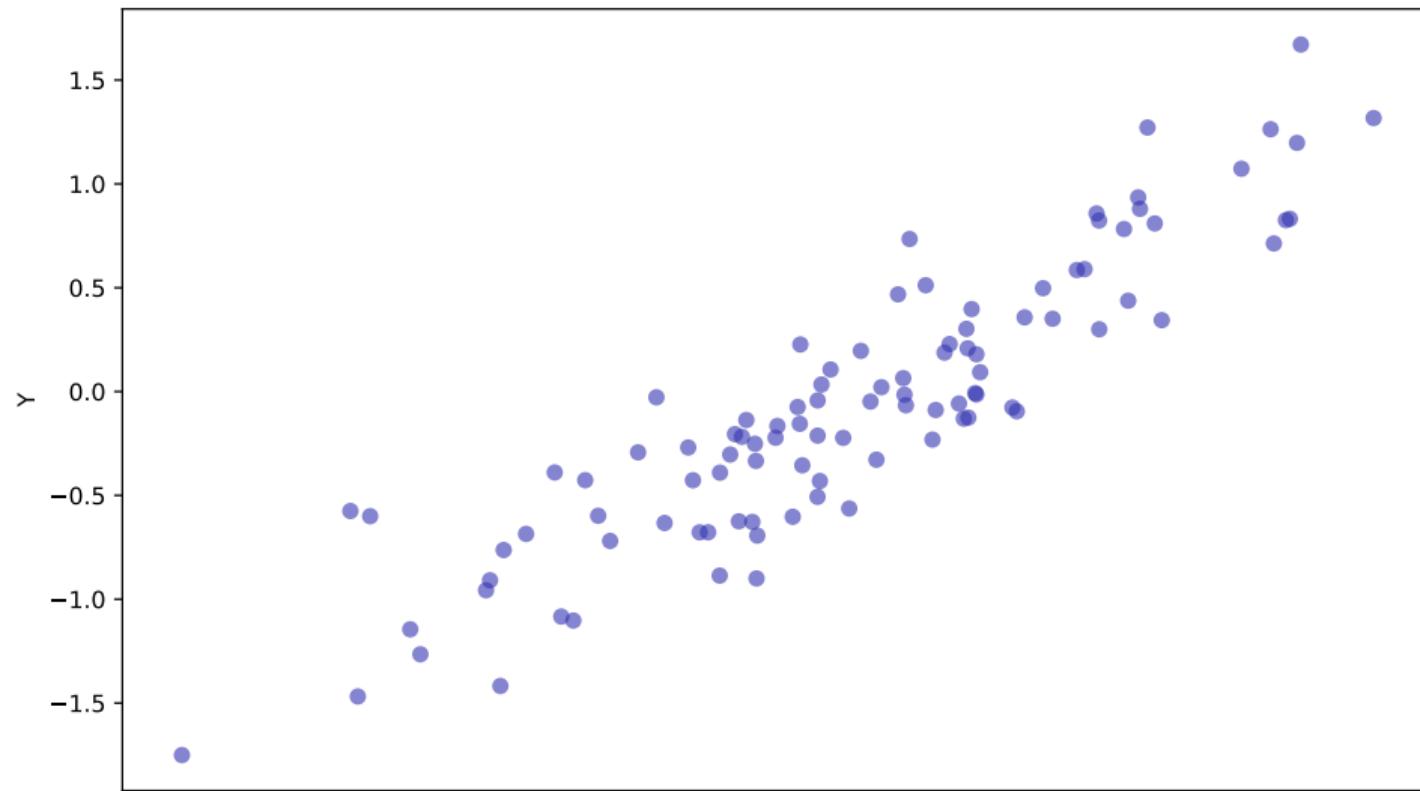
45 Minutes

After this lesson, you will be able to:

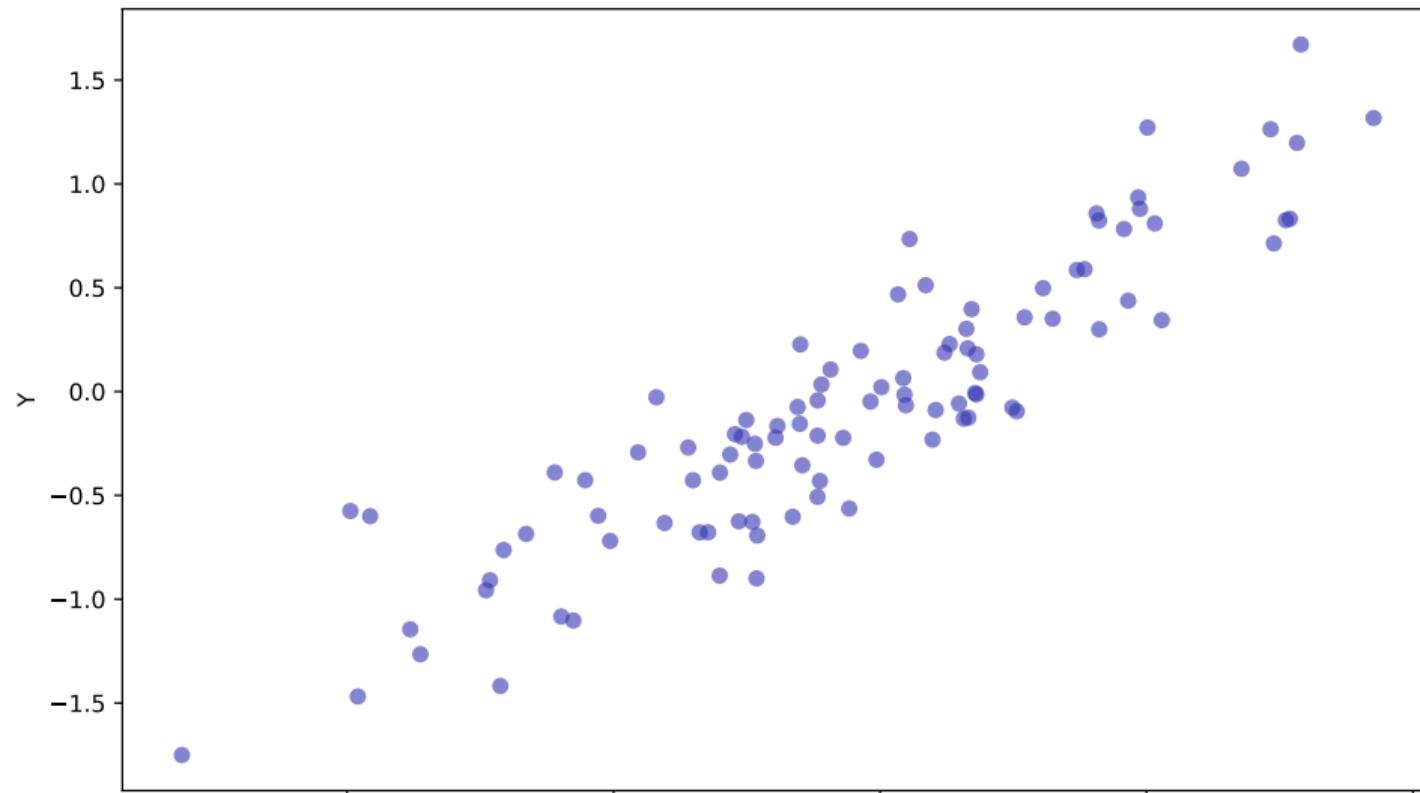
- Calculate Pearson and Spearman correlation
- Create correlation heatmaps
- Distinguish correlation from causation
- Apply to portfolio analysis

Finance application: Statistical analysis of market data

01 Correlation Scatter



02 Correlation Values



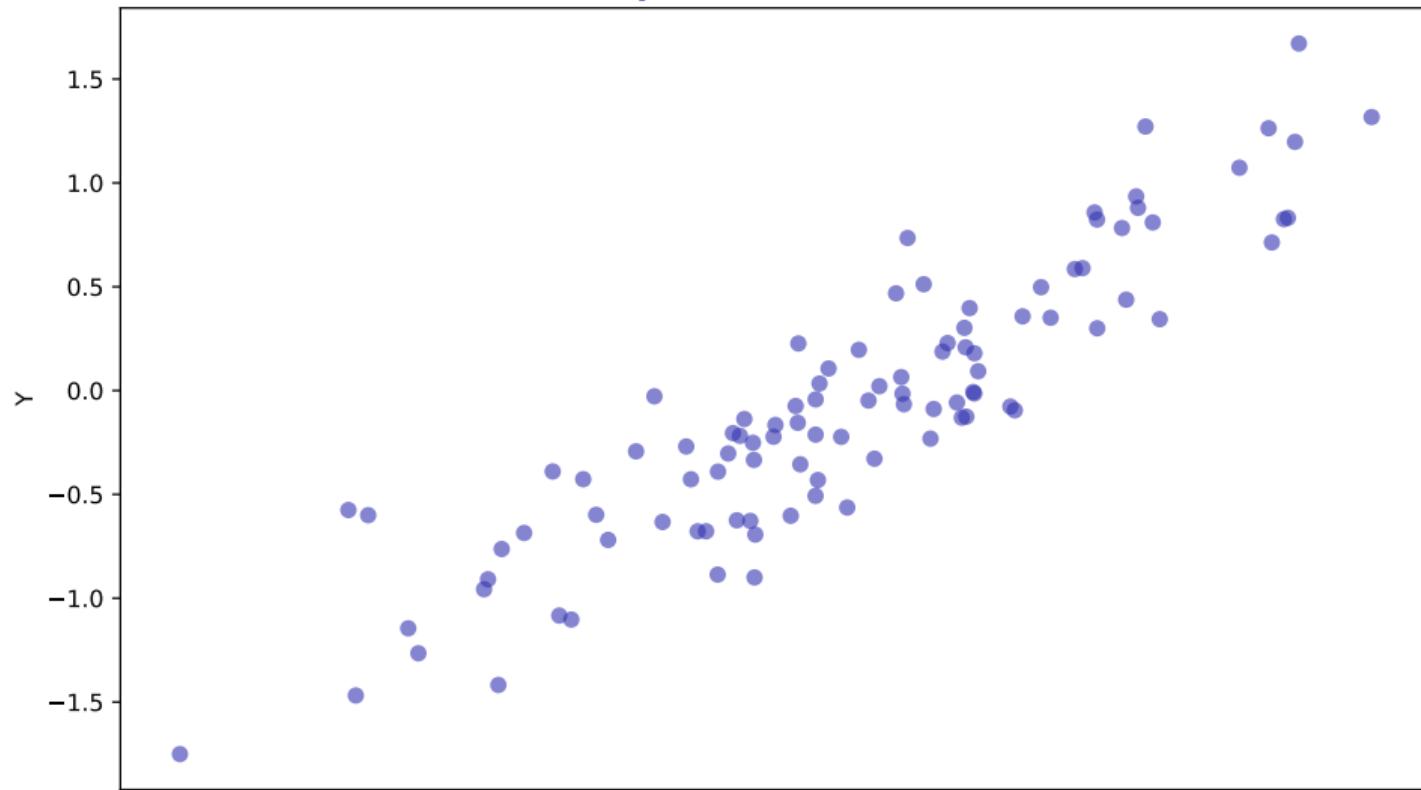
L16



04 Heatmap



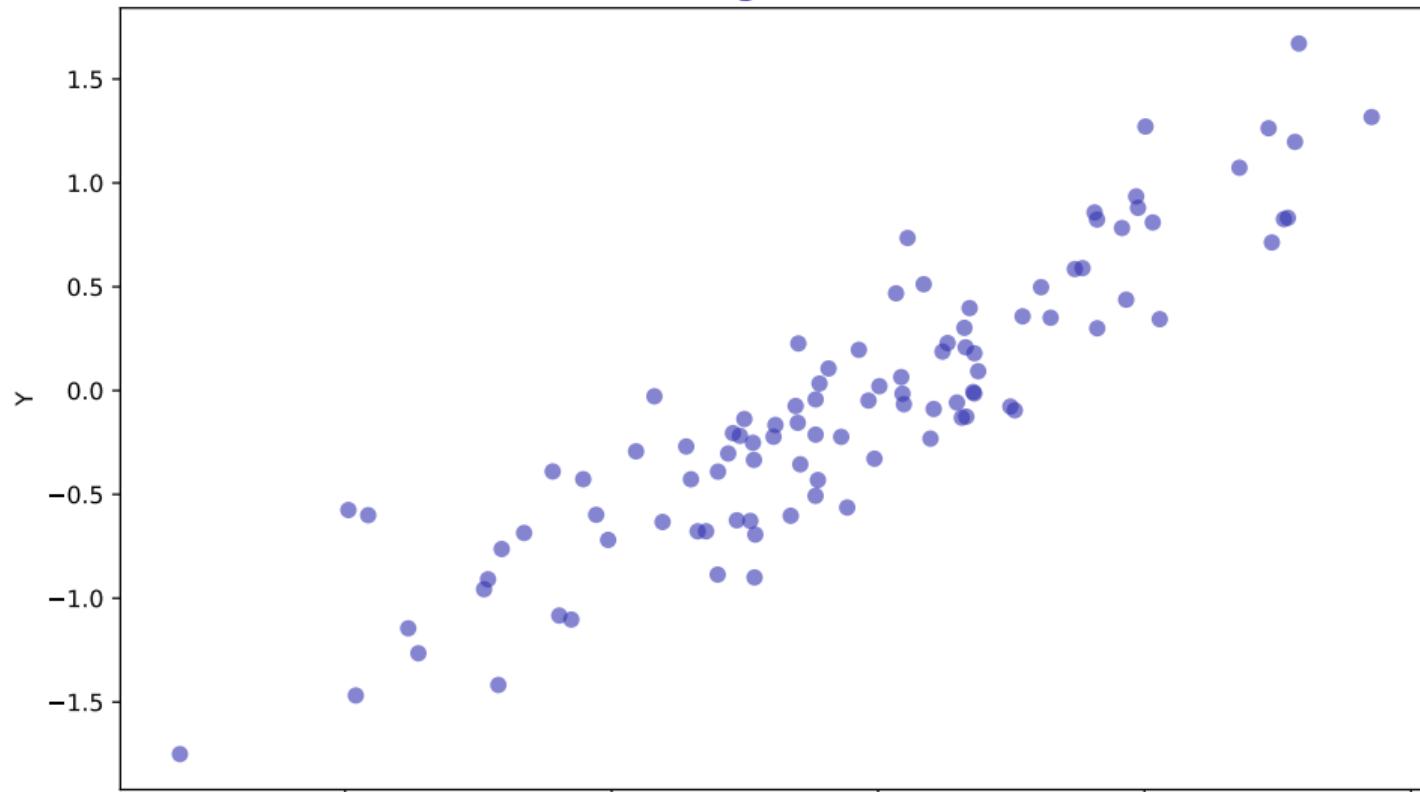
05 Spurious Correlation



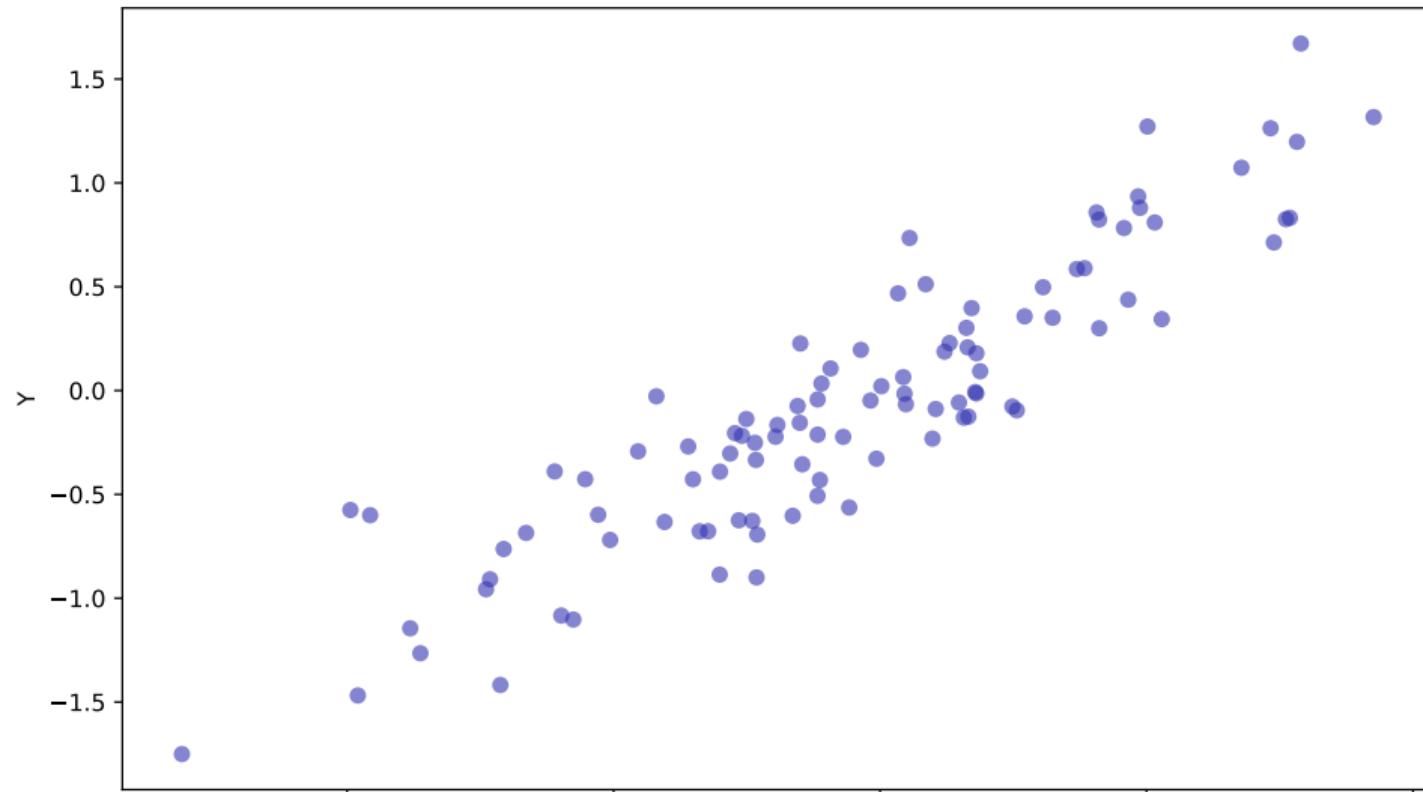
L16

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

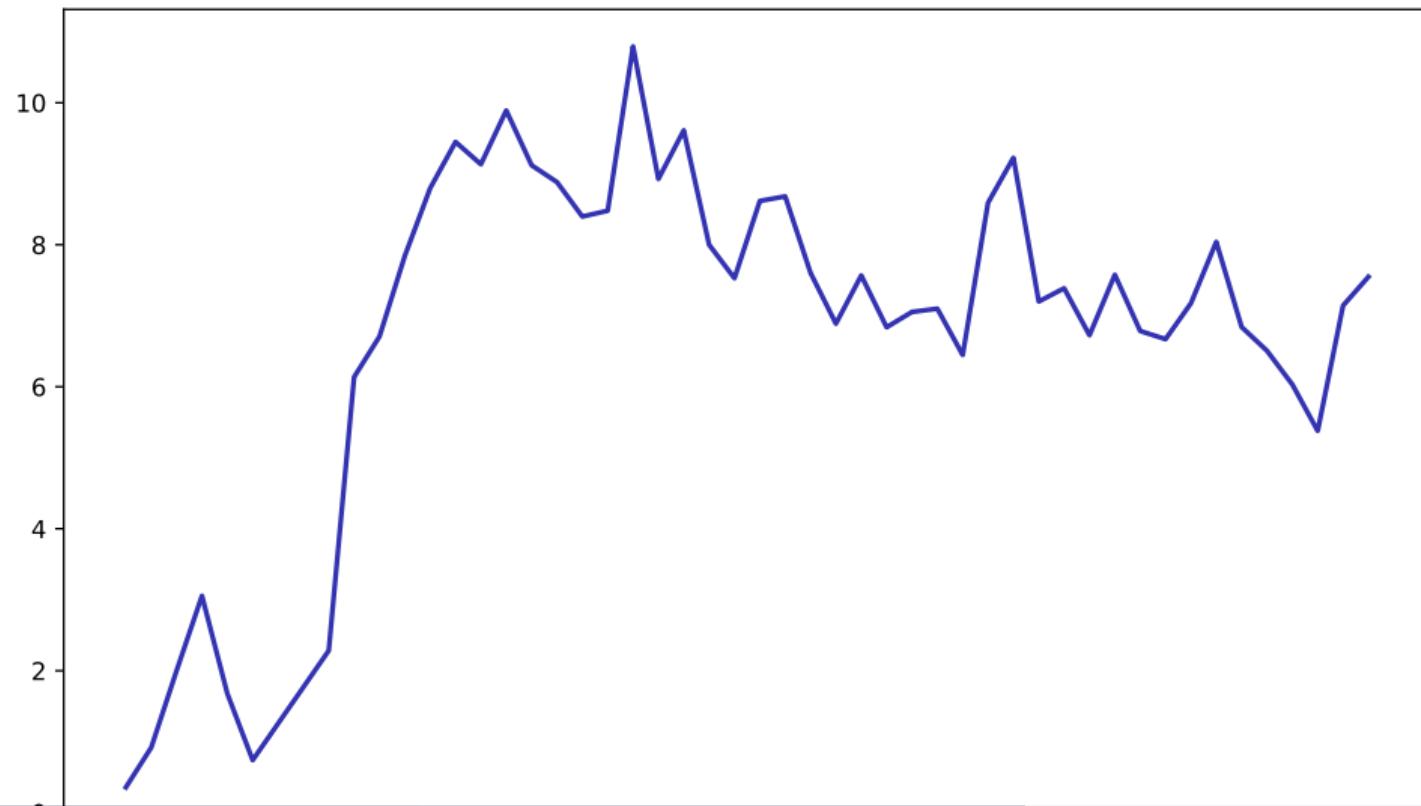
After this lesson, you will be able to:

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

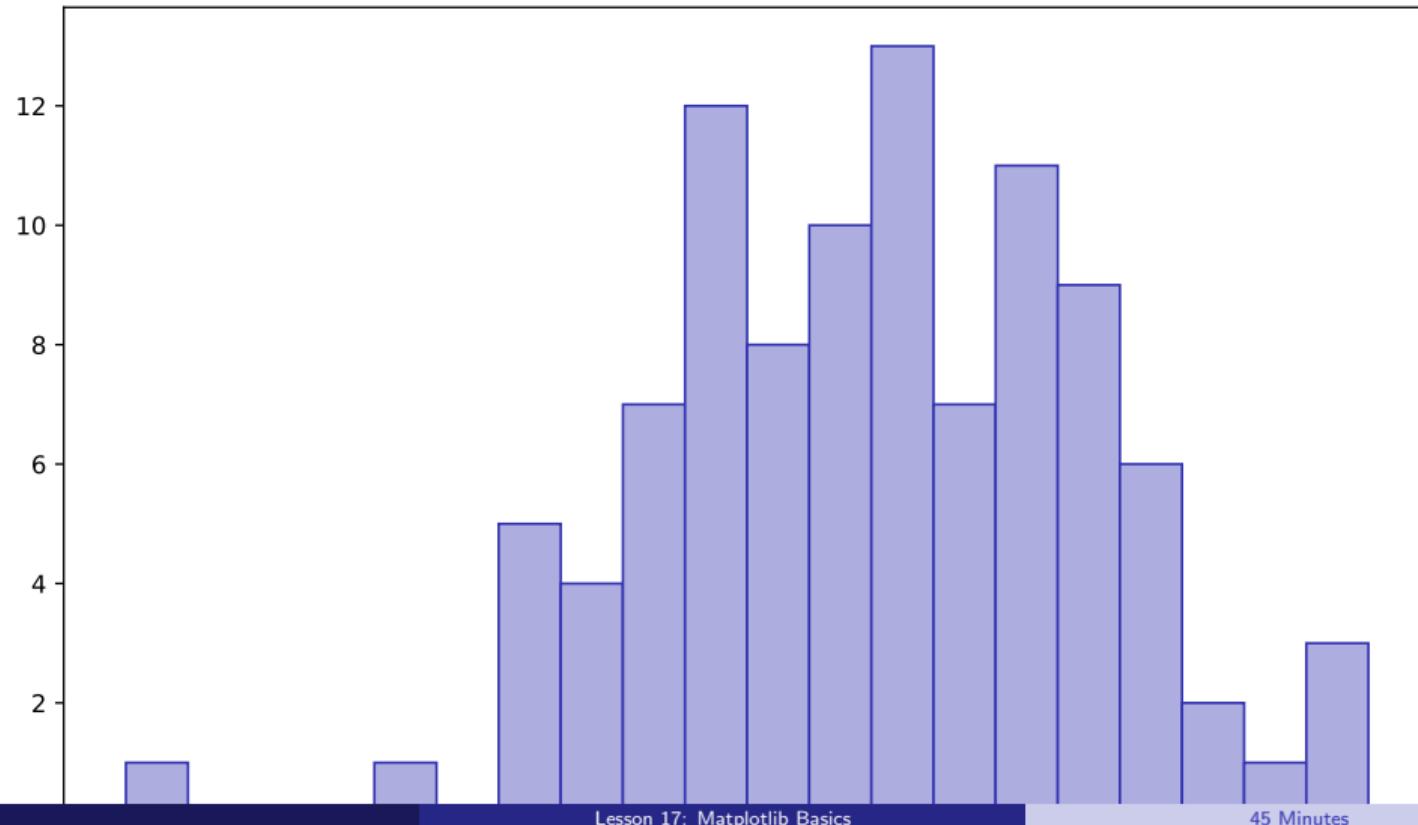
Finance application: Statistical analysis of market data

Line Plot

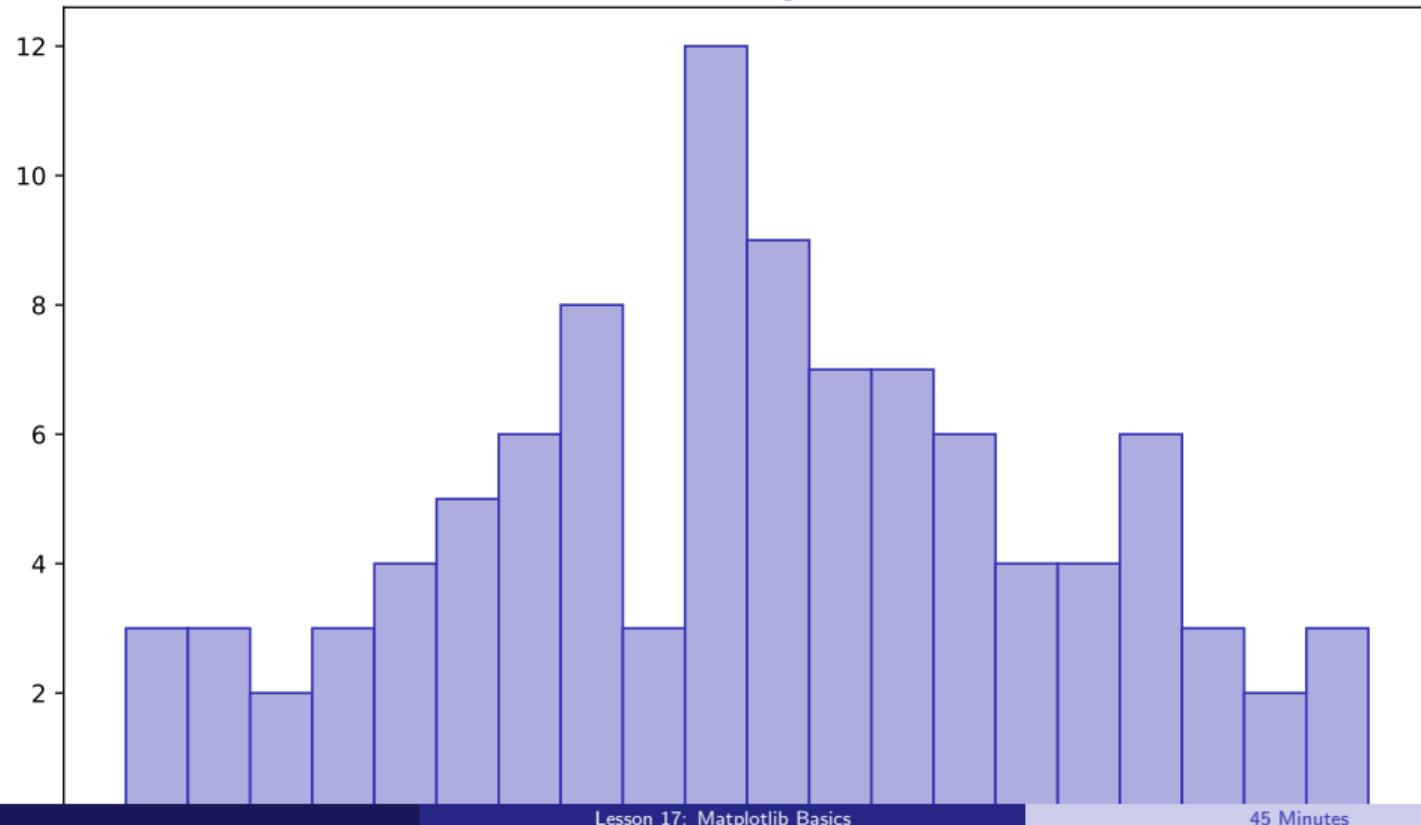
01 Line Plot



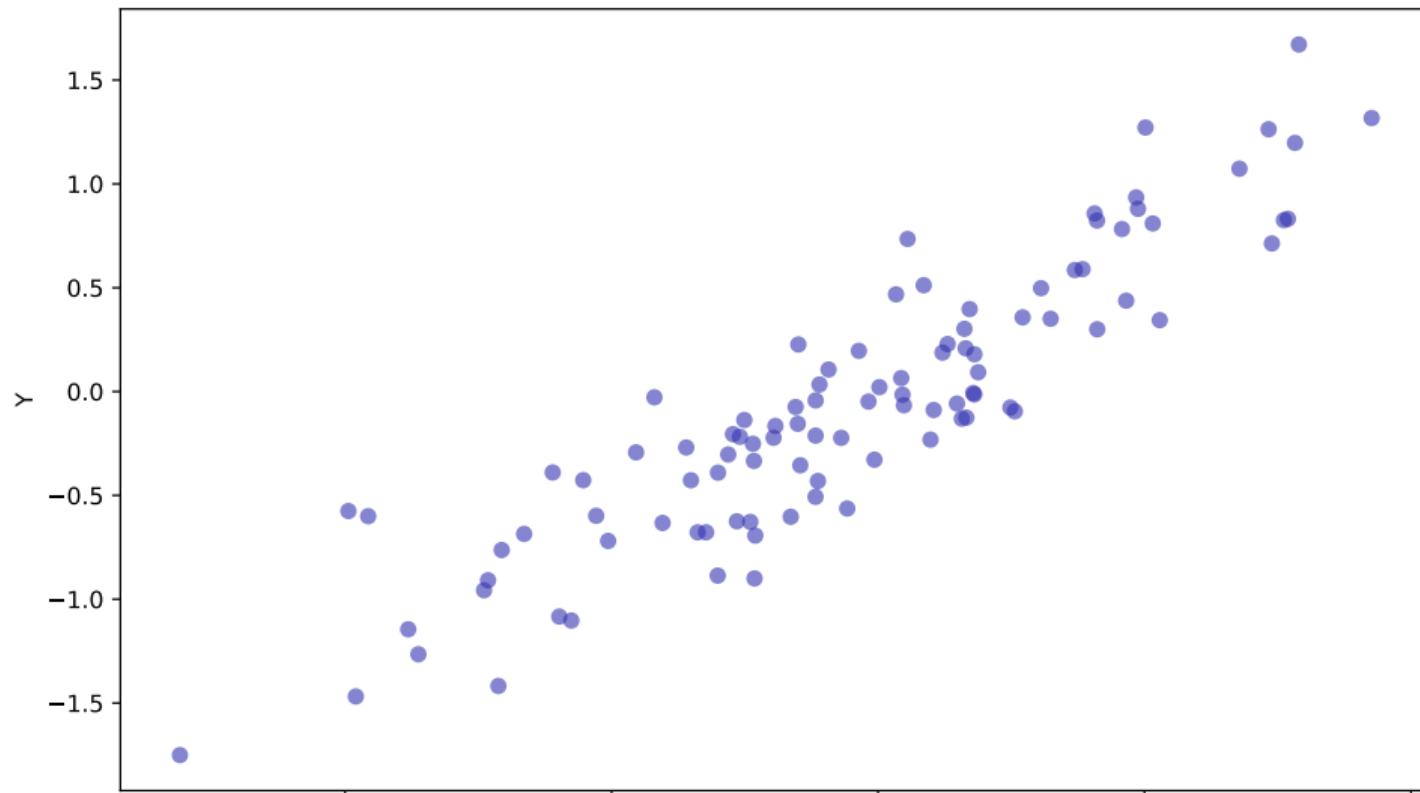
02 Bar Chart



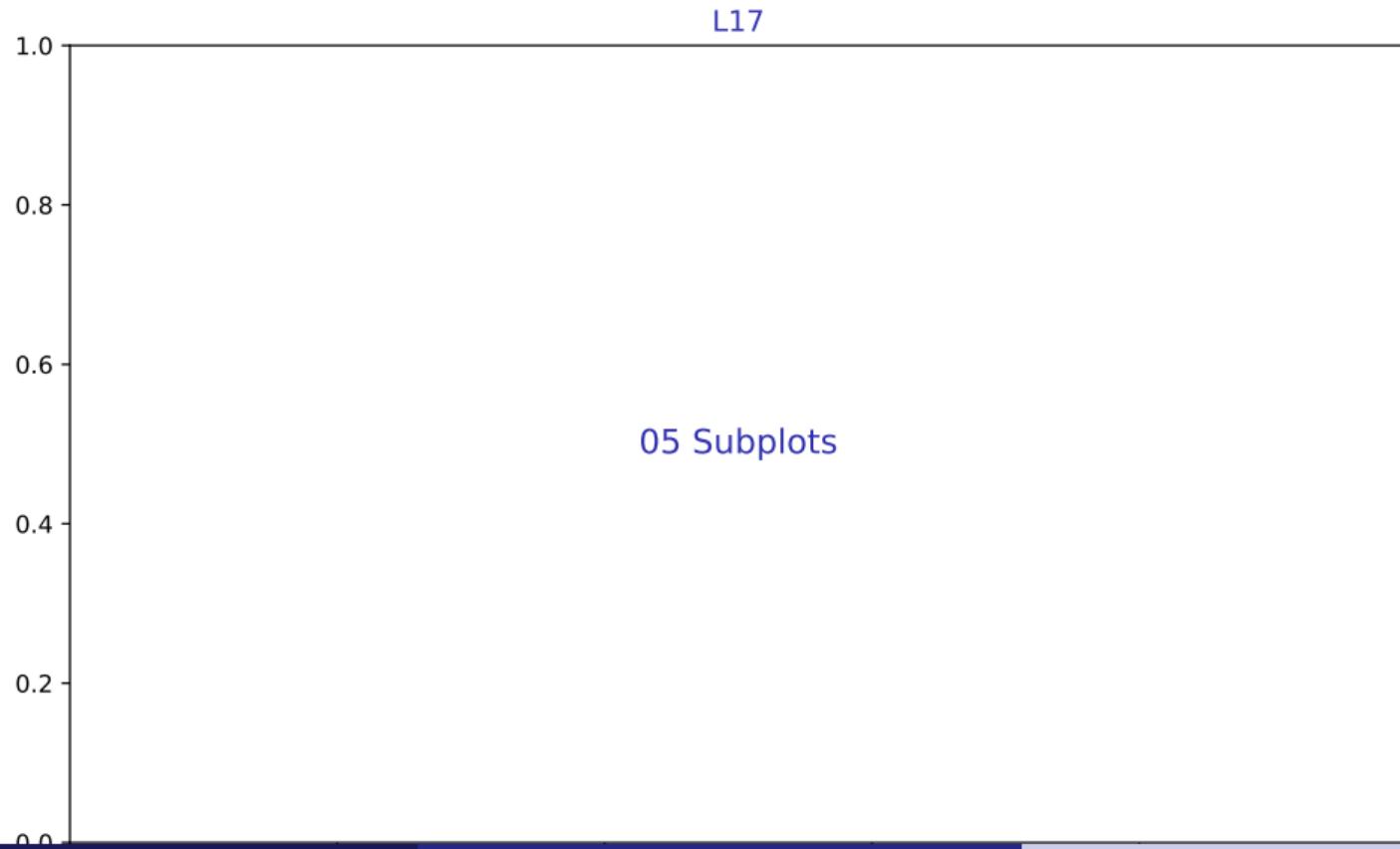
03 Histogram



04 Scatter Plot



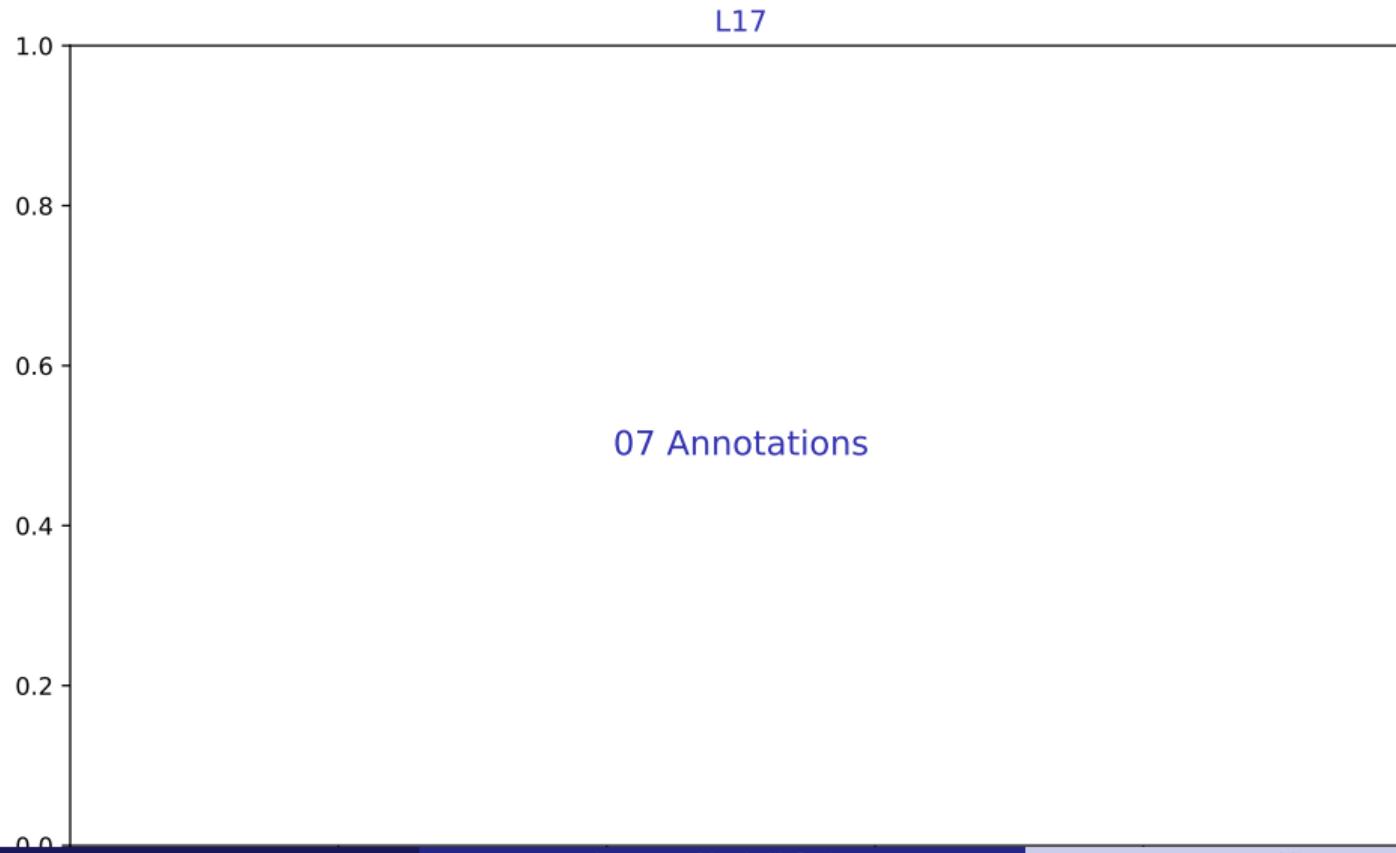
Subplots



L17

06 Customization

Annotations



L17

08 Finance Charts

Lesson Summary

Key Takeaways:

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

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:

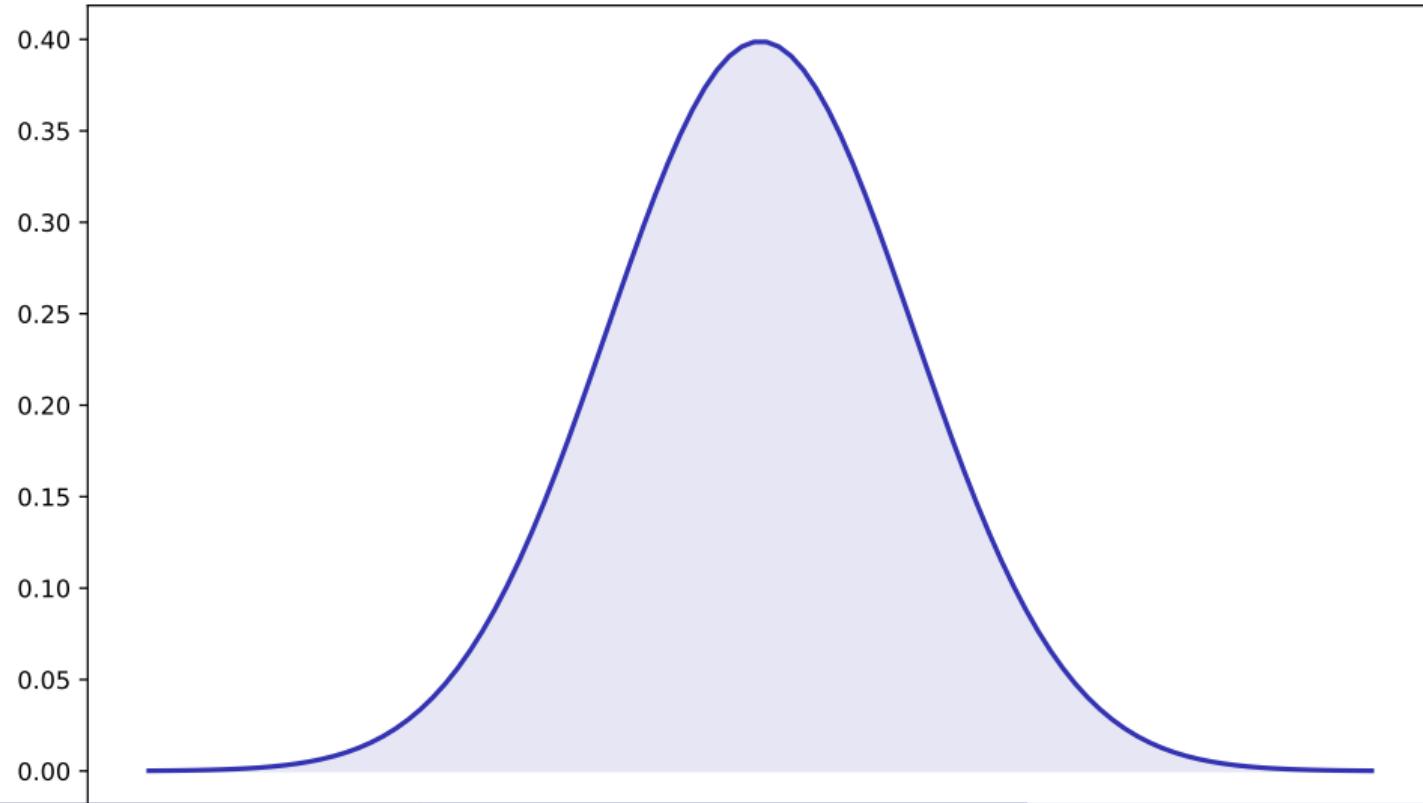
- Use seaborn for statistical visualization
- Create distribution and categorical plots
- Build correlation heatmaps
- Apply professional styling

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

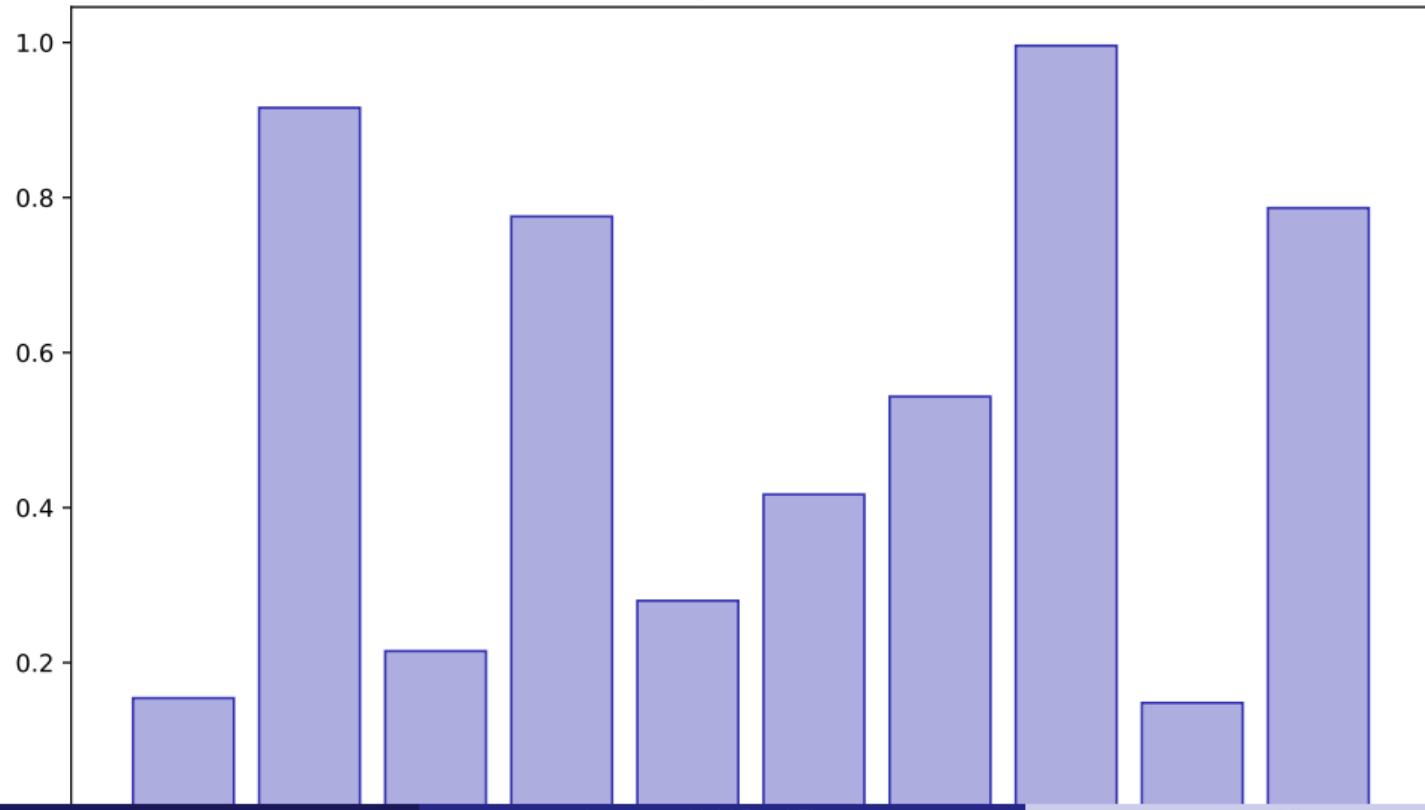
After this lesson, you will be able to:

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

Building towards your final project

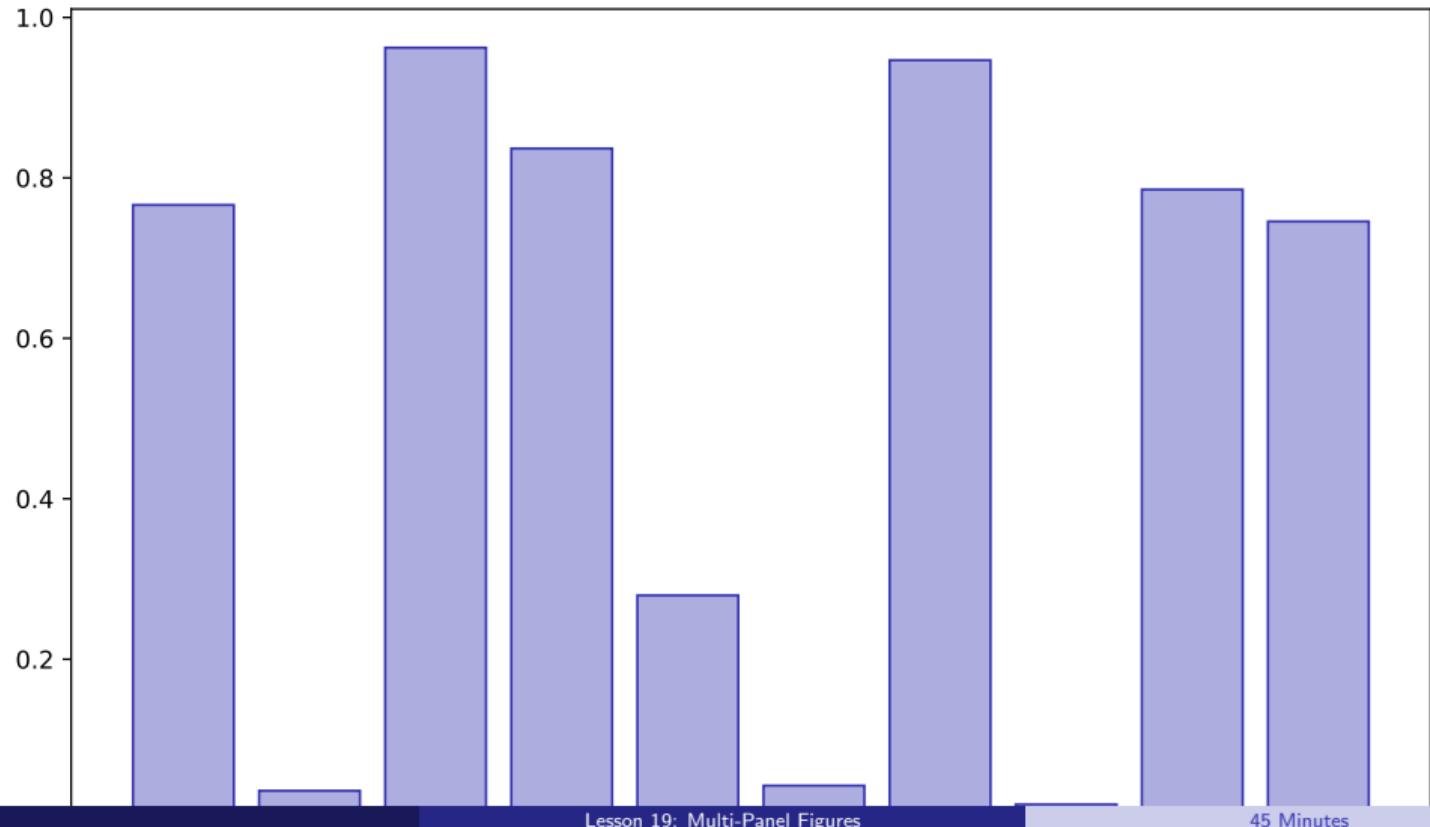
Subplots Grid

Subplots Grid



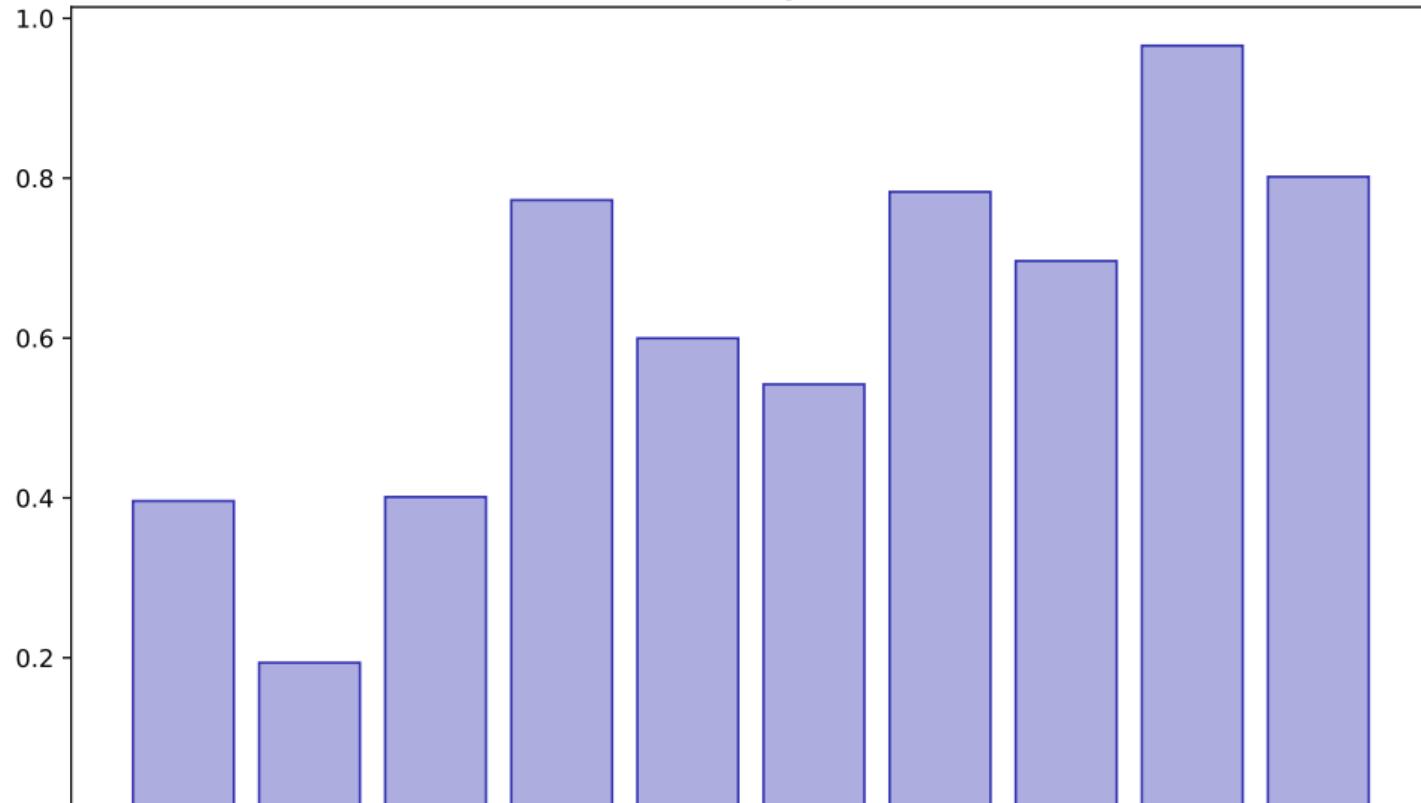
Shared Axes

Shared Axes

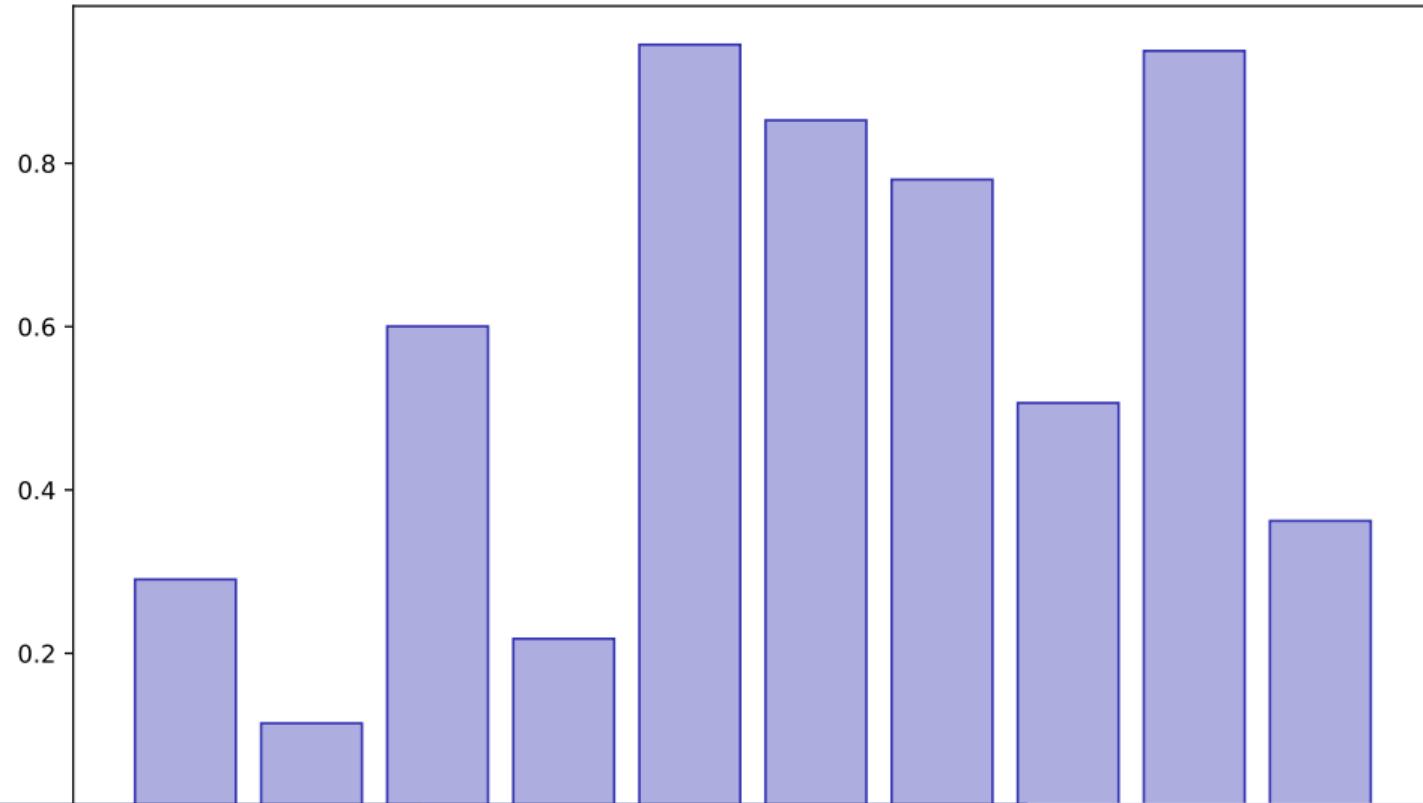


Mixed Layouts

Mixed Layouts



Gridspec



Nested Plots

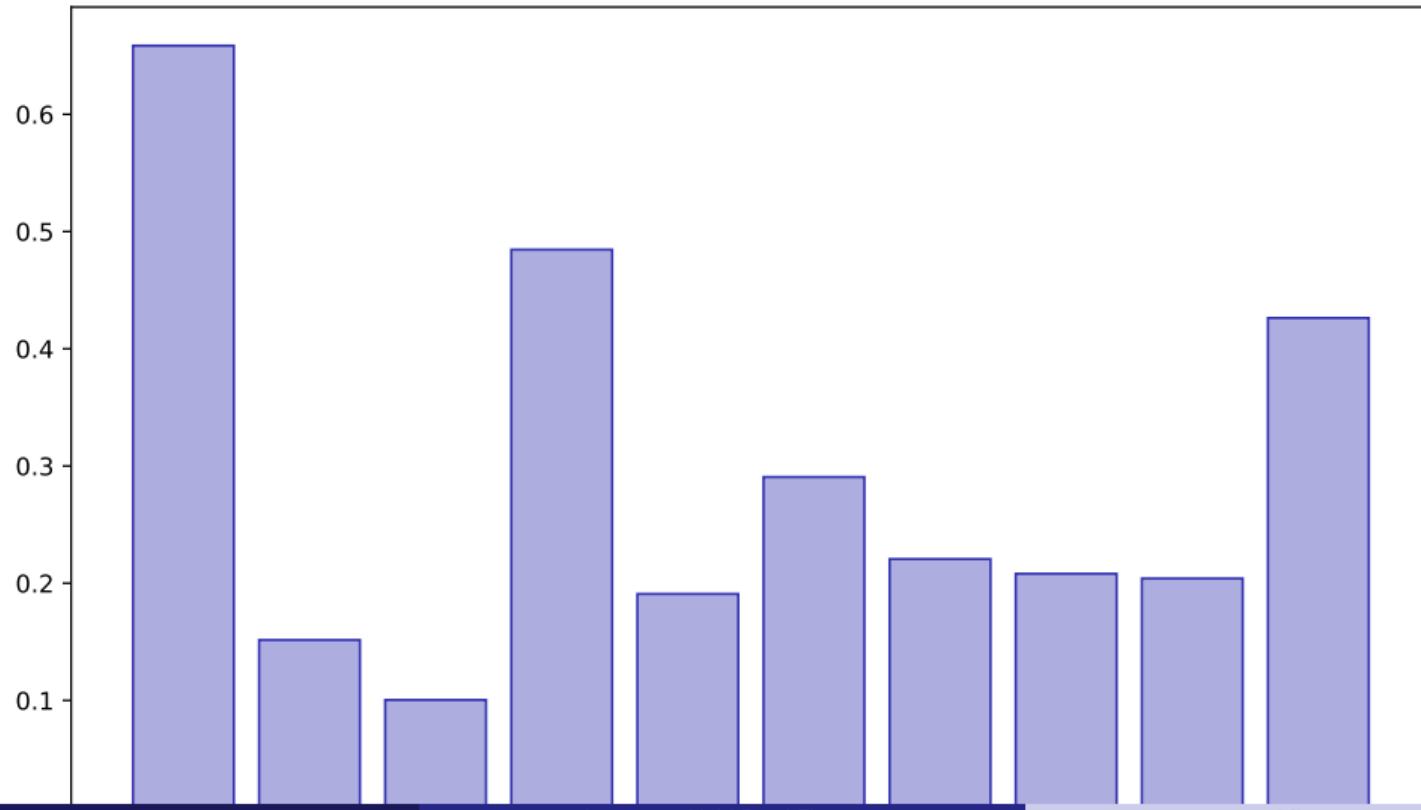
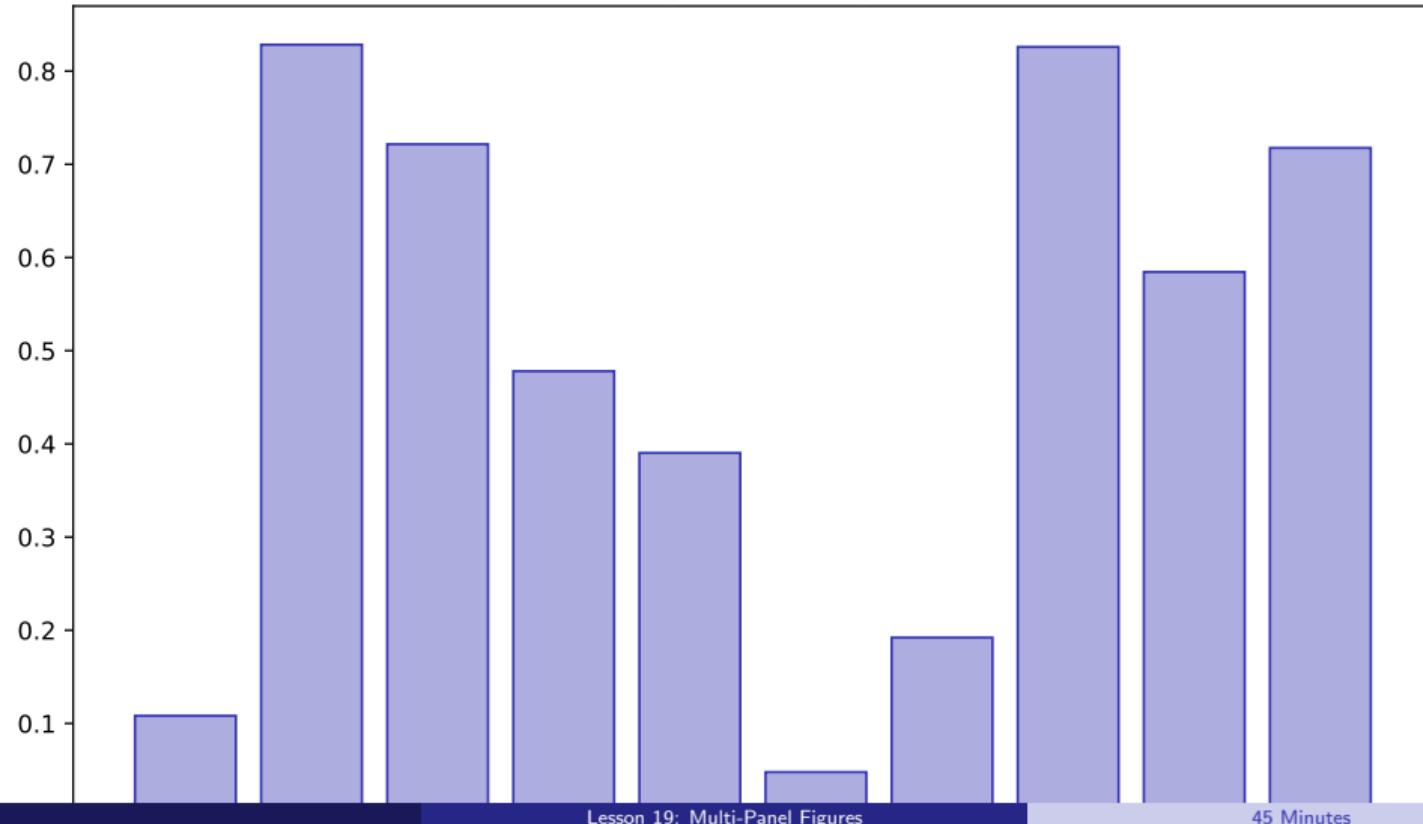
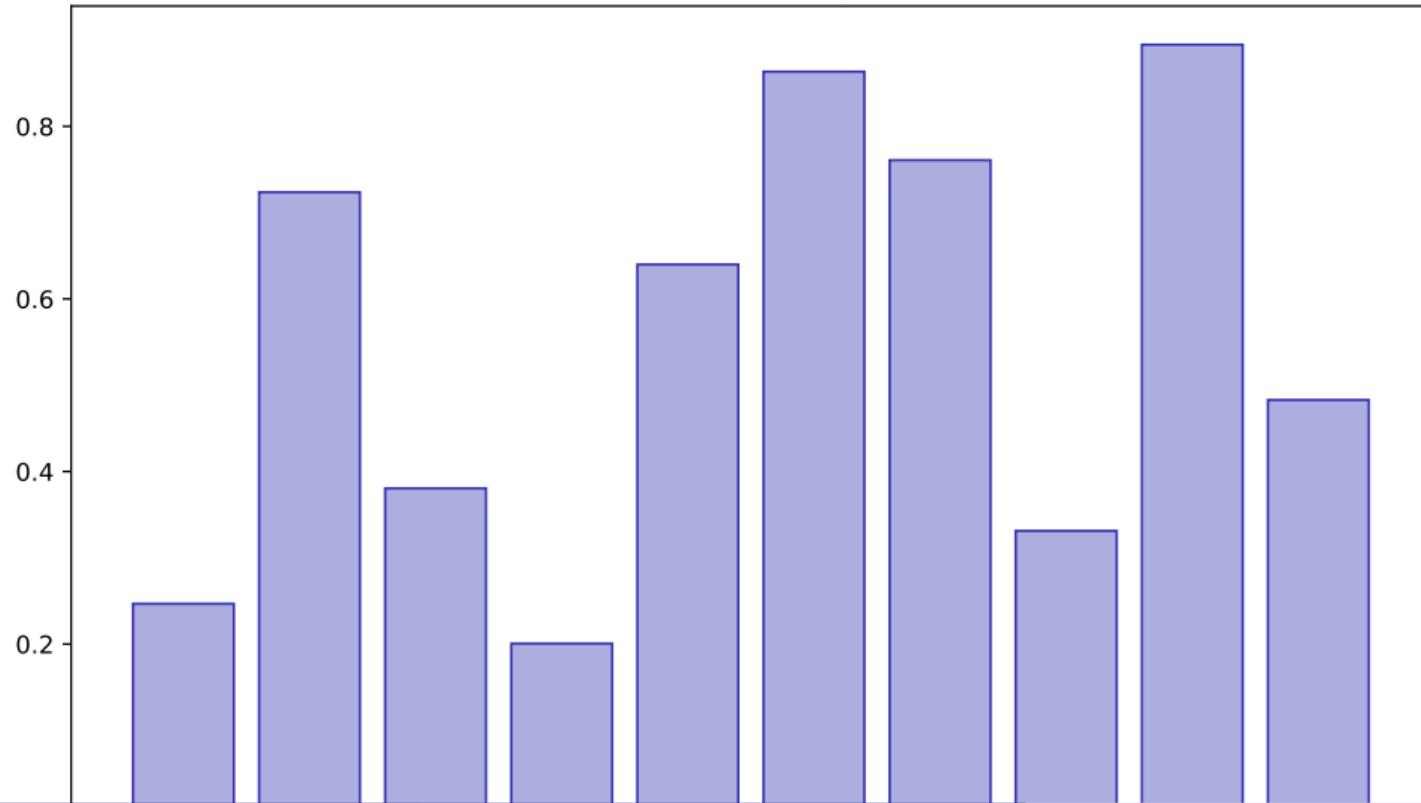


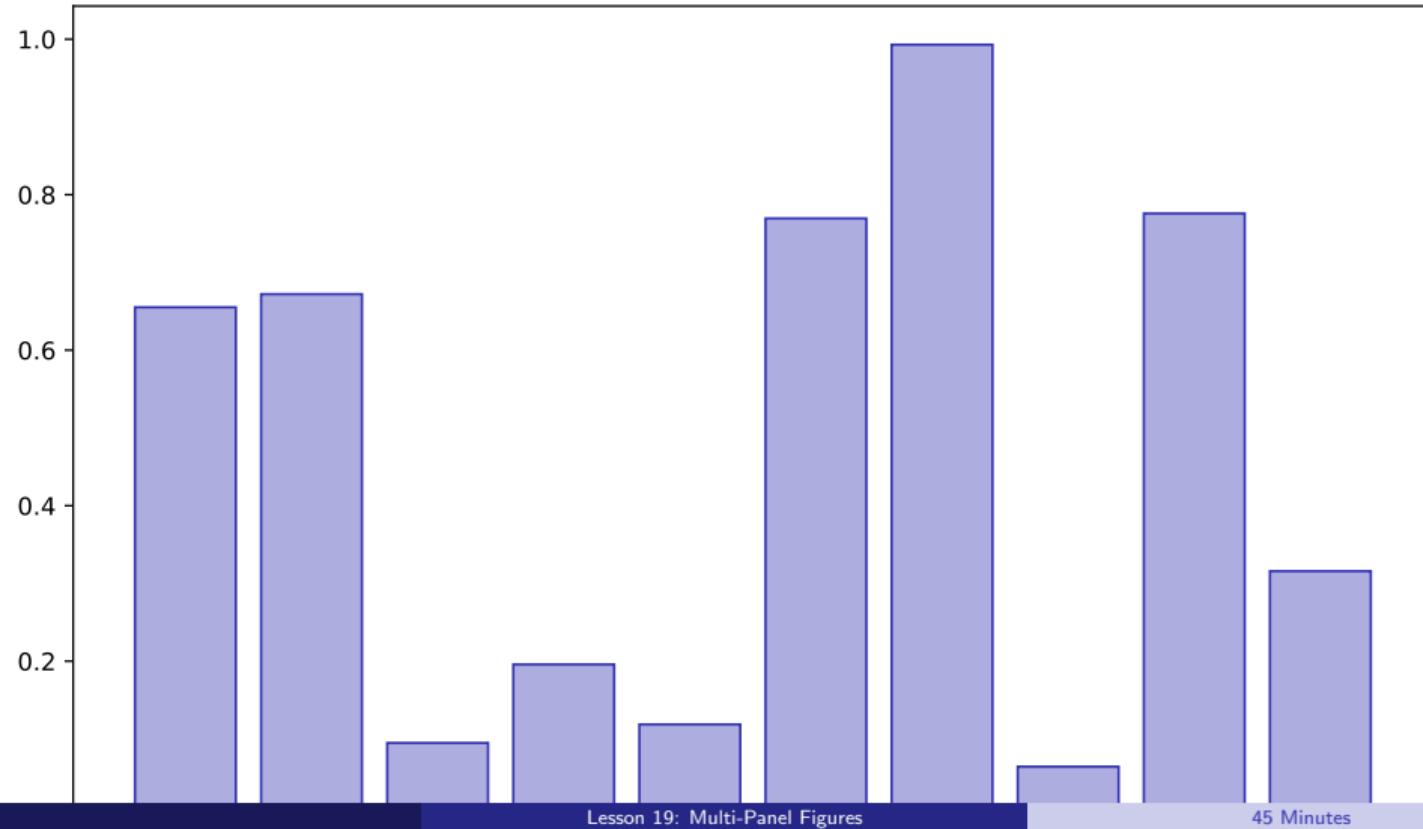
Figure Sizing



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

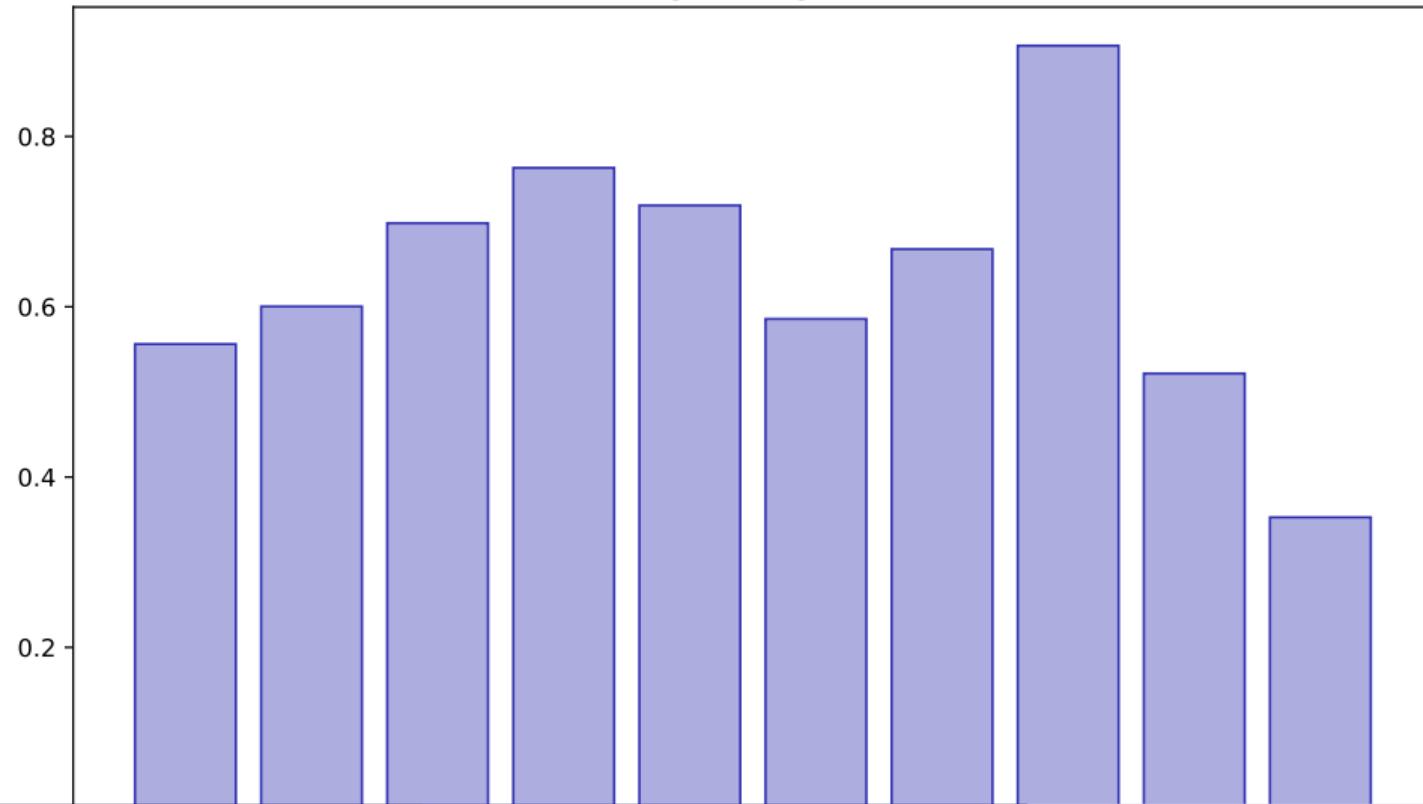
Learning Objectives

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

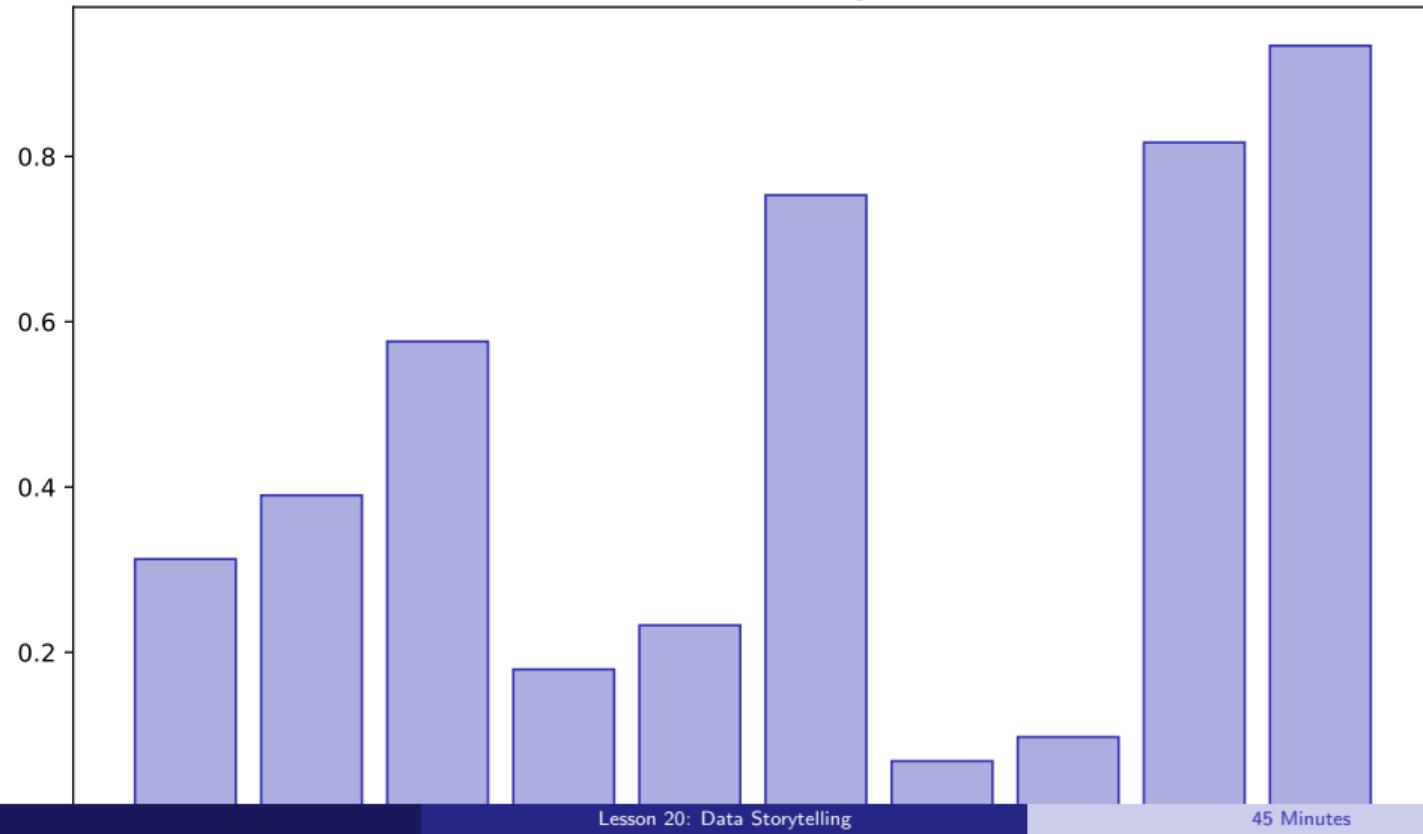
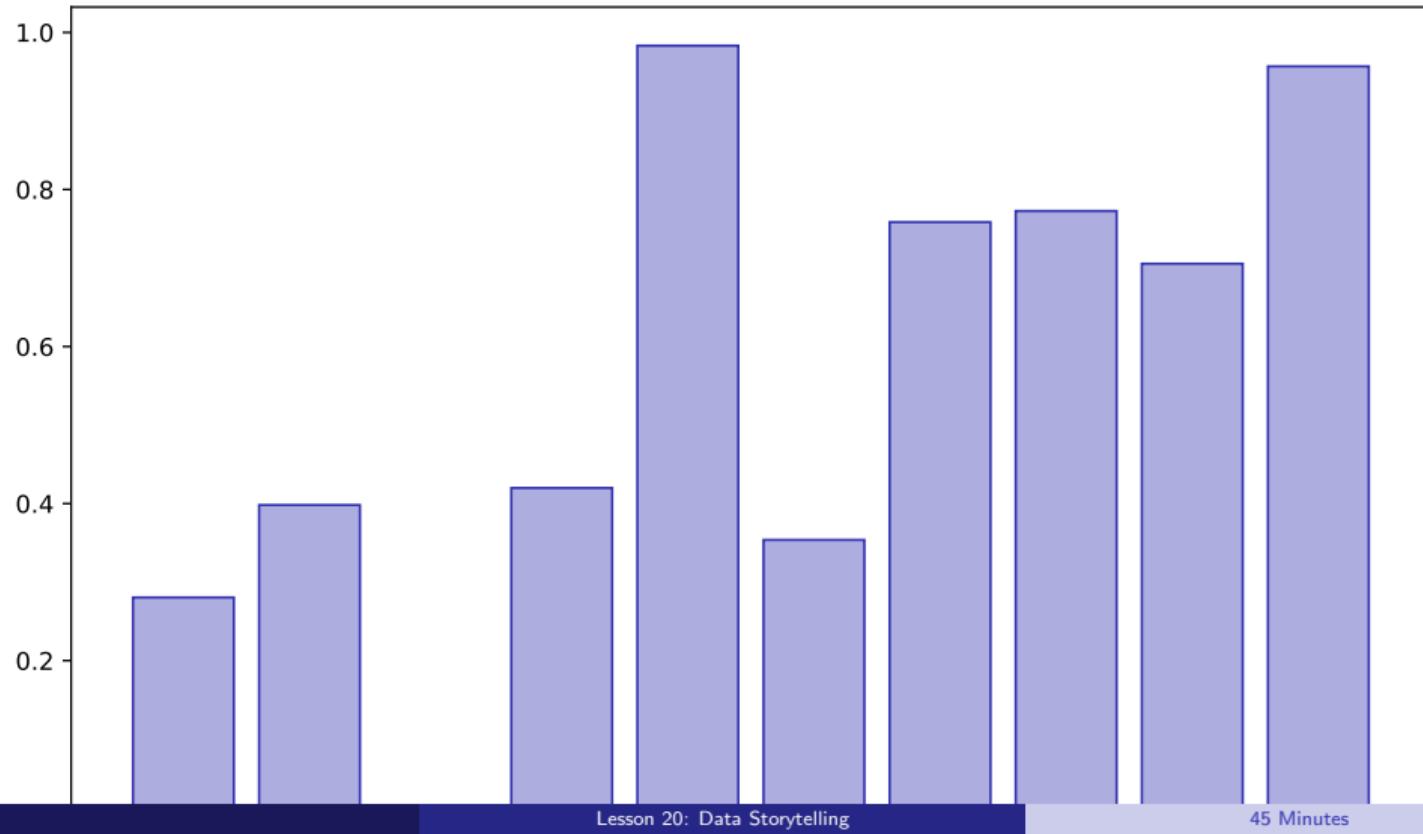
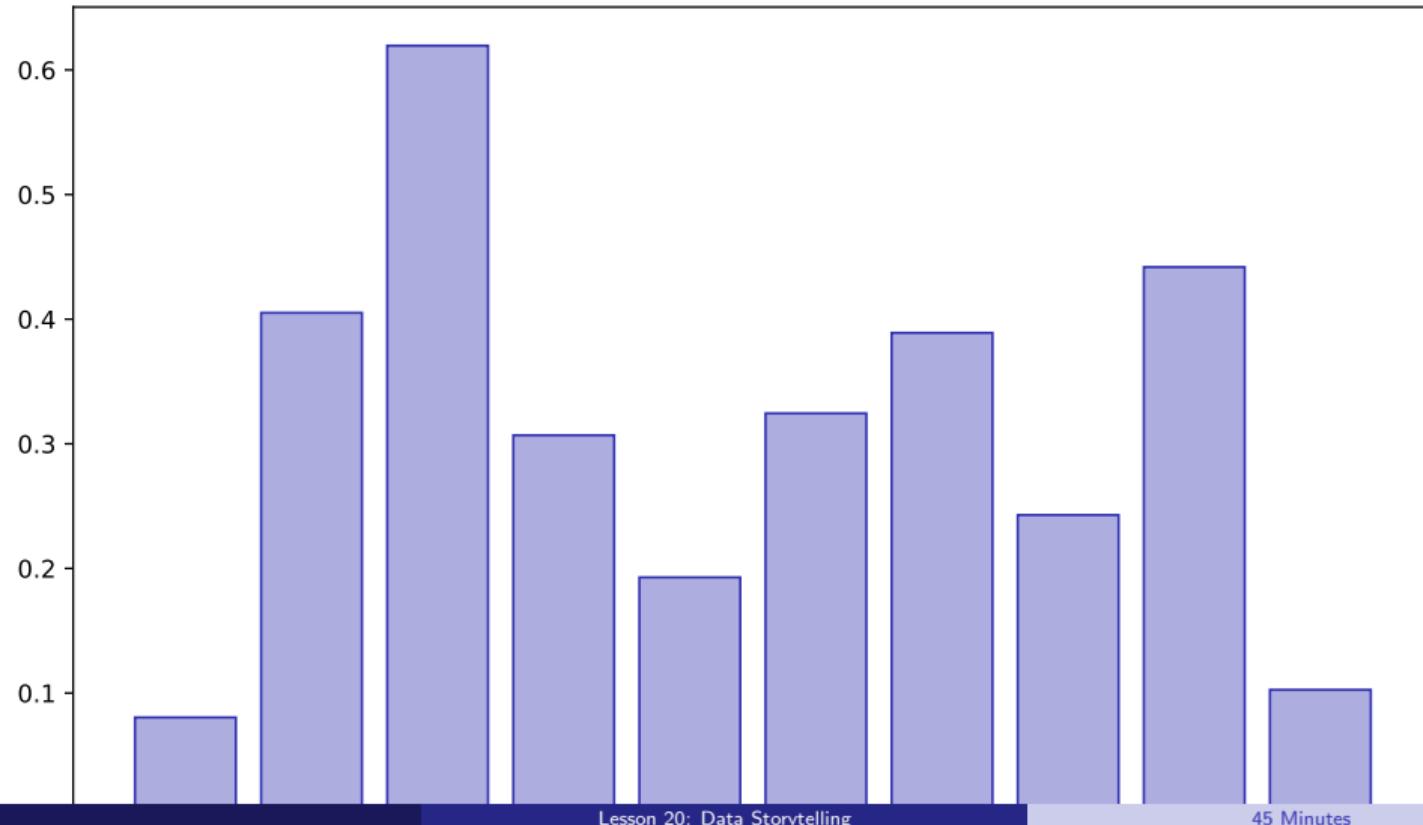


Chart Selection

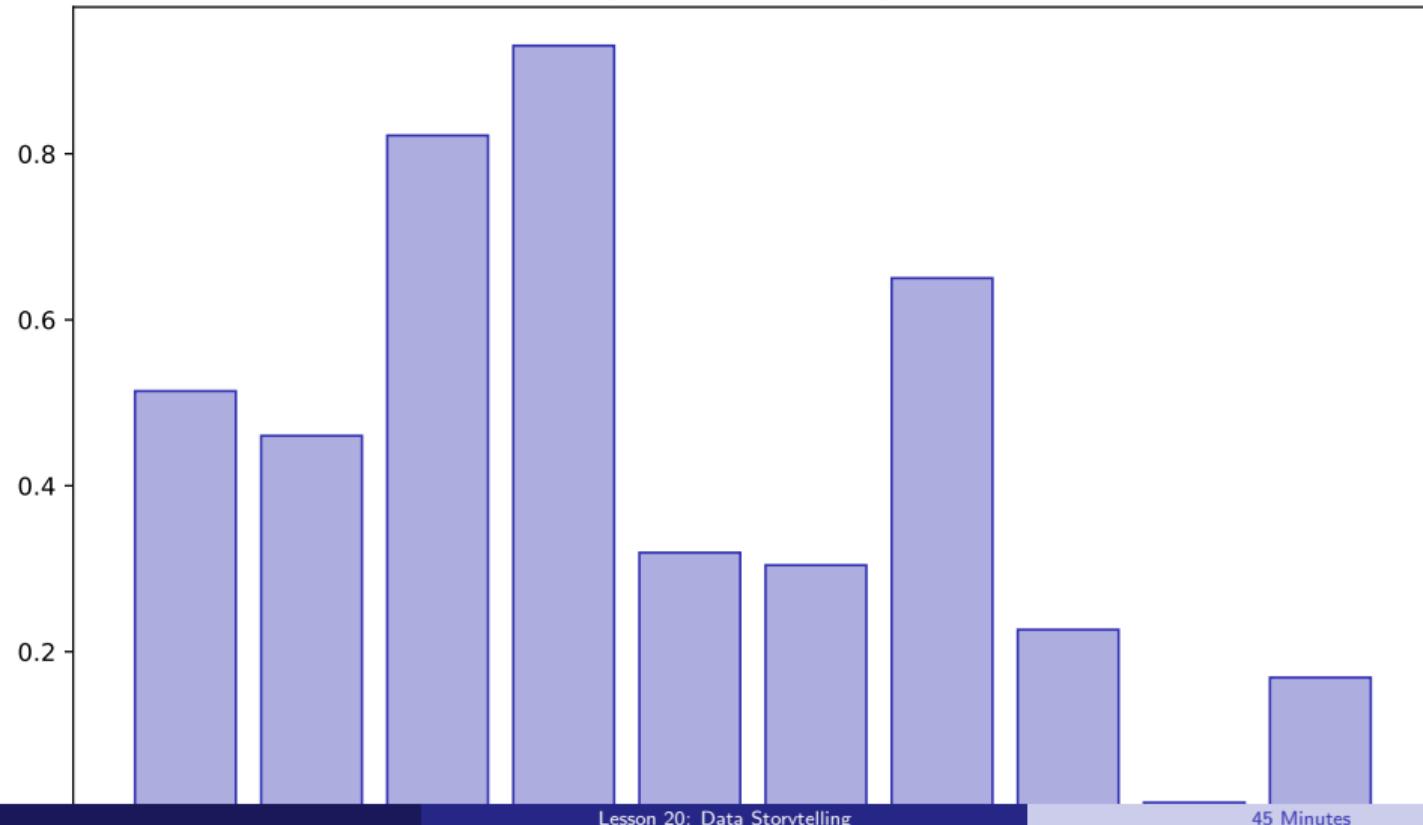


Annotations

Annotations

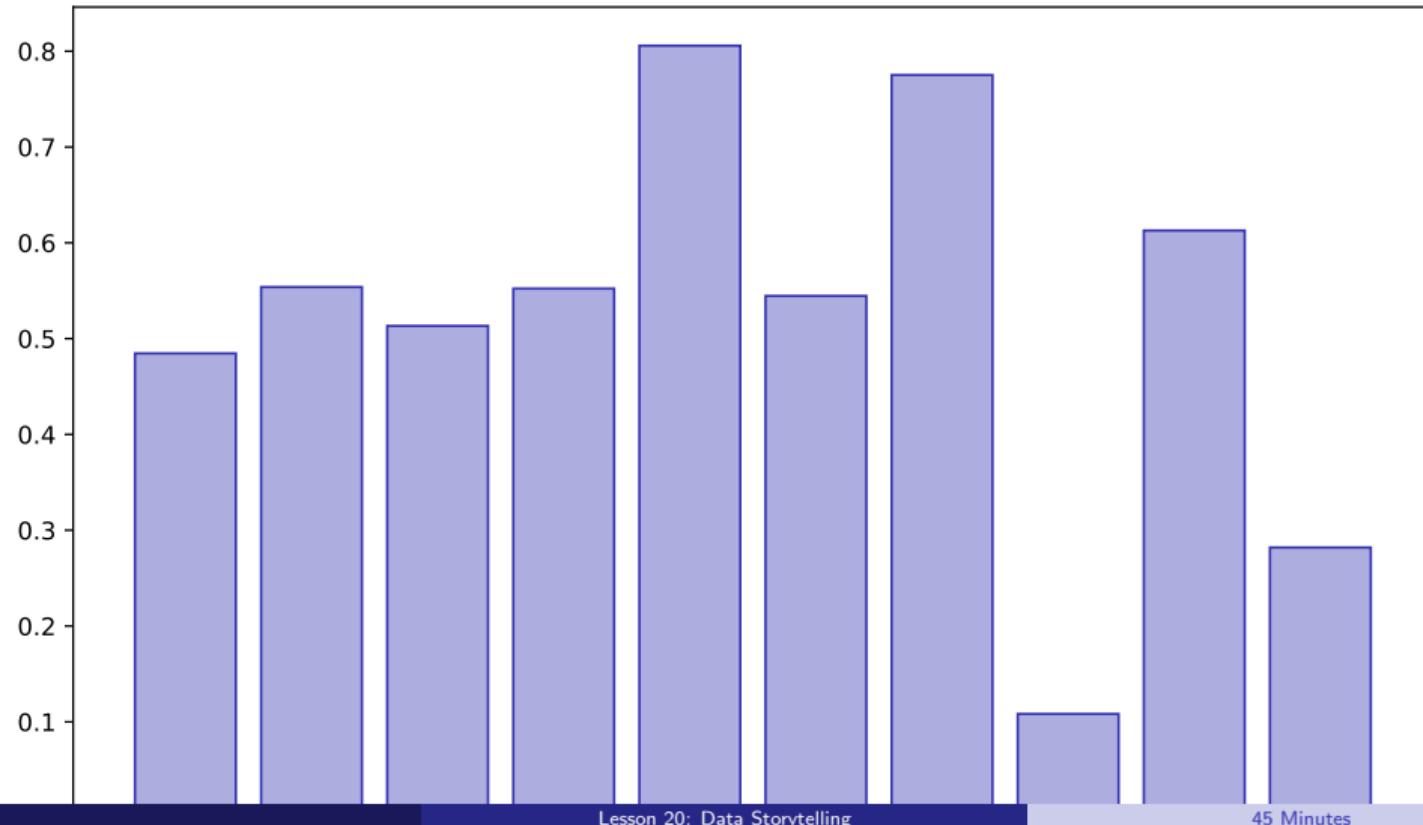


Emphasis Techniques

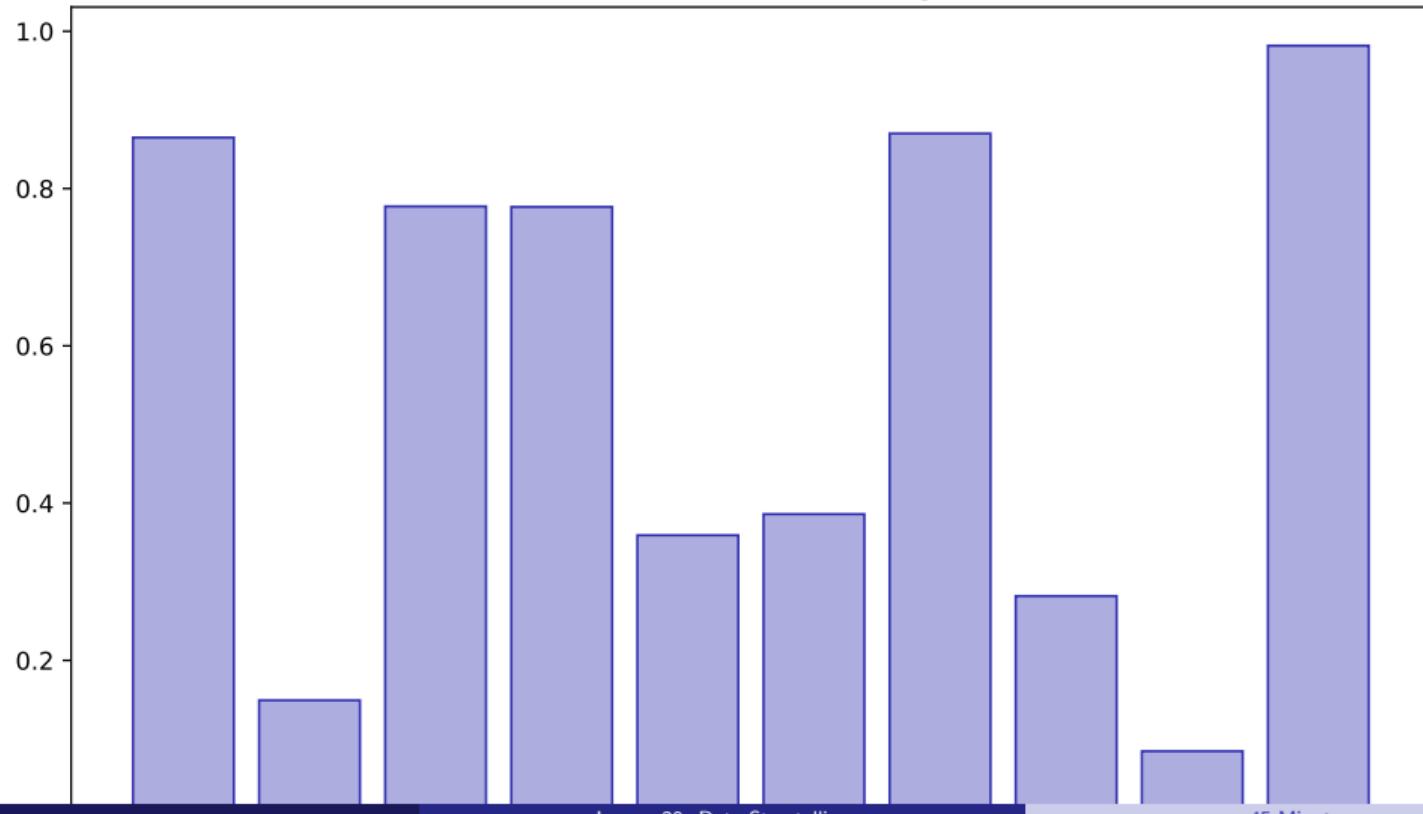


Before After

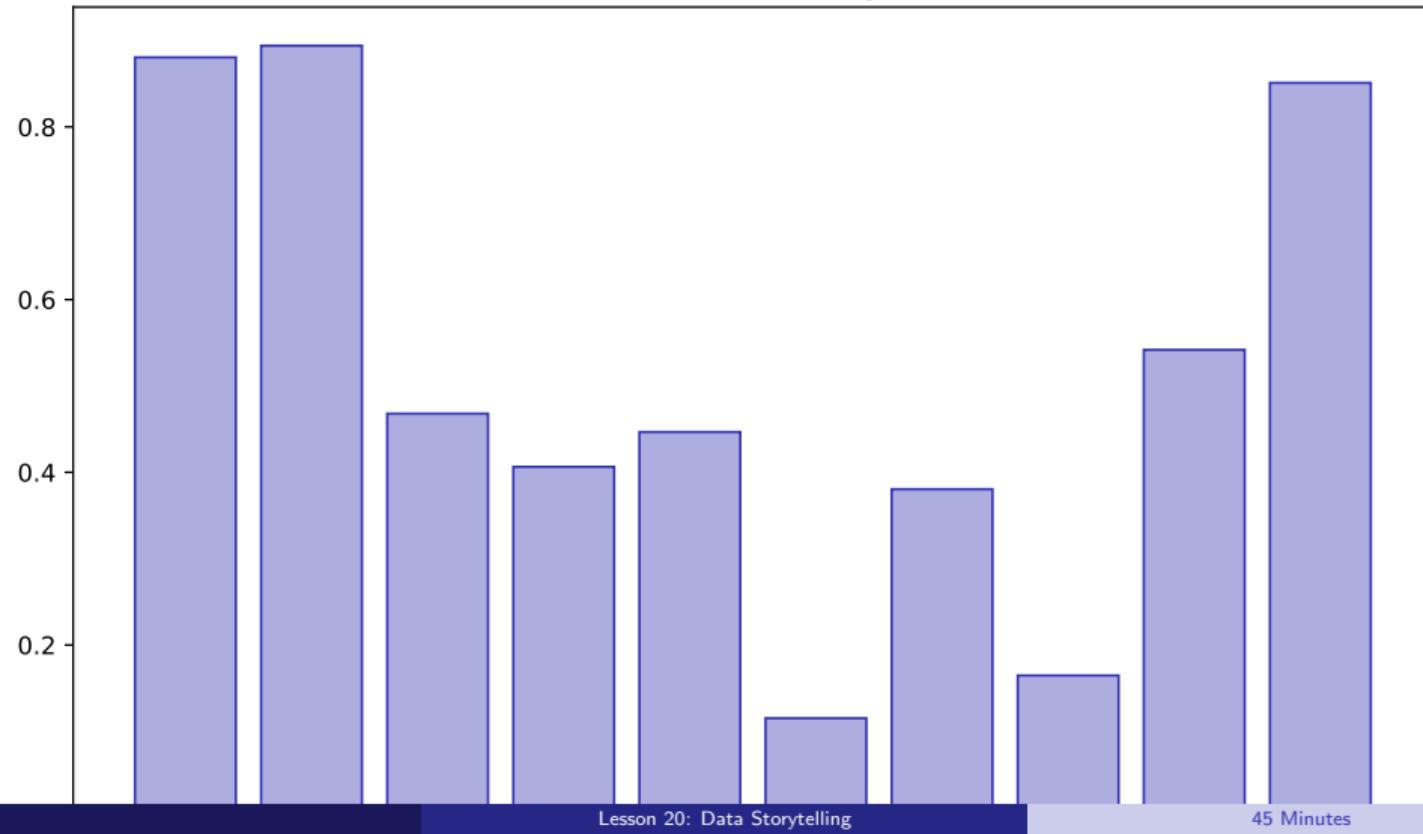
Before After



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

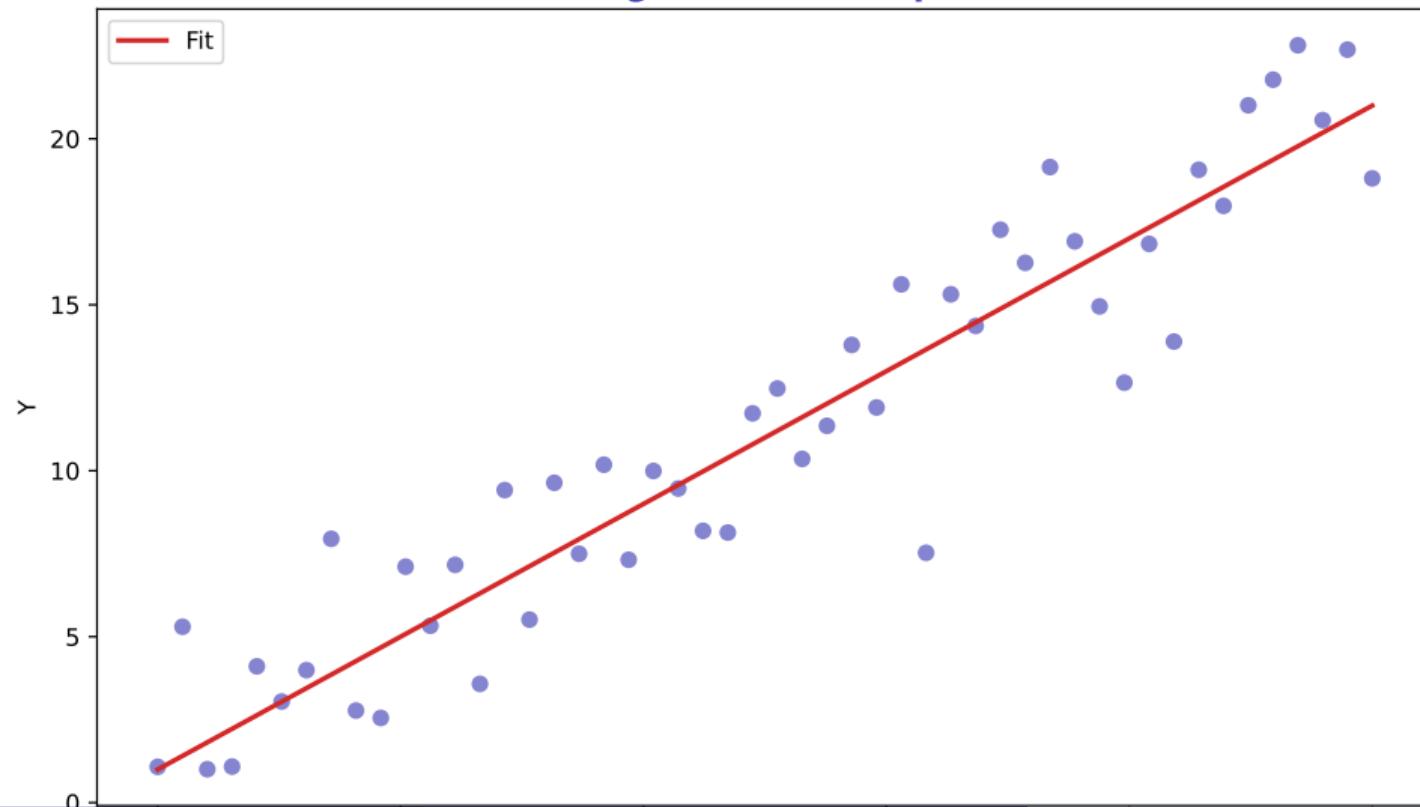
Learning Objectives

After this lesson, you will be able to:

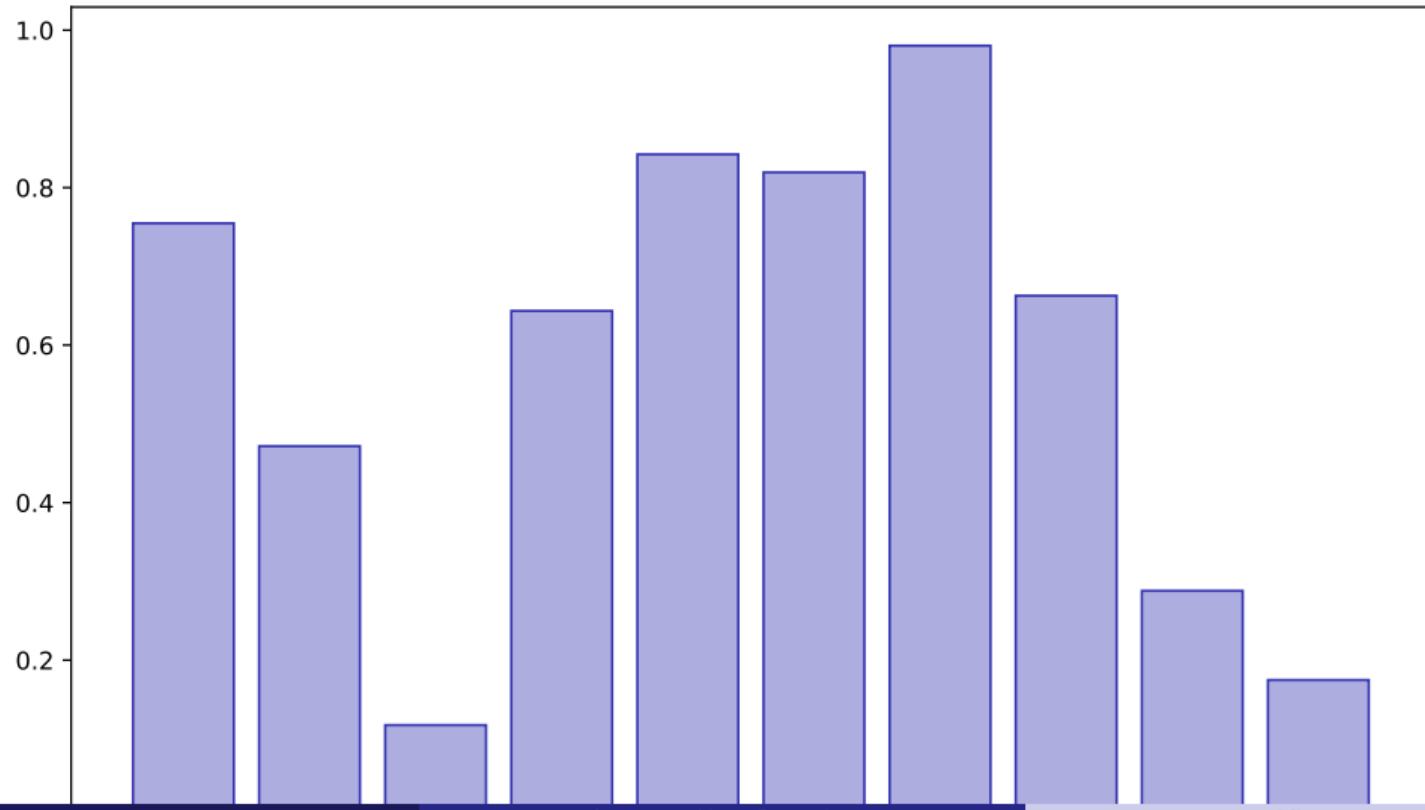
- Understand OLS estimation
- Fit linear models with sklearn
- Interpret coefficients
- Apply to stock price prediction

Building towards your final project

Regression Concept



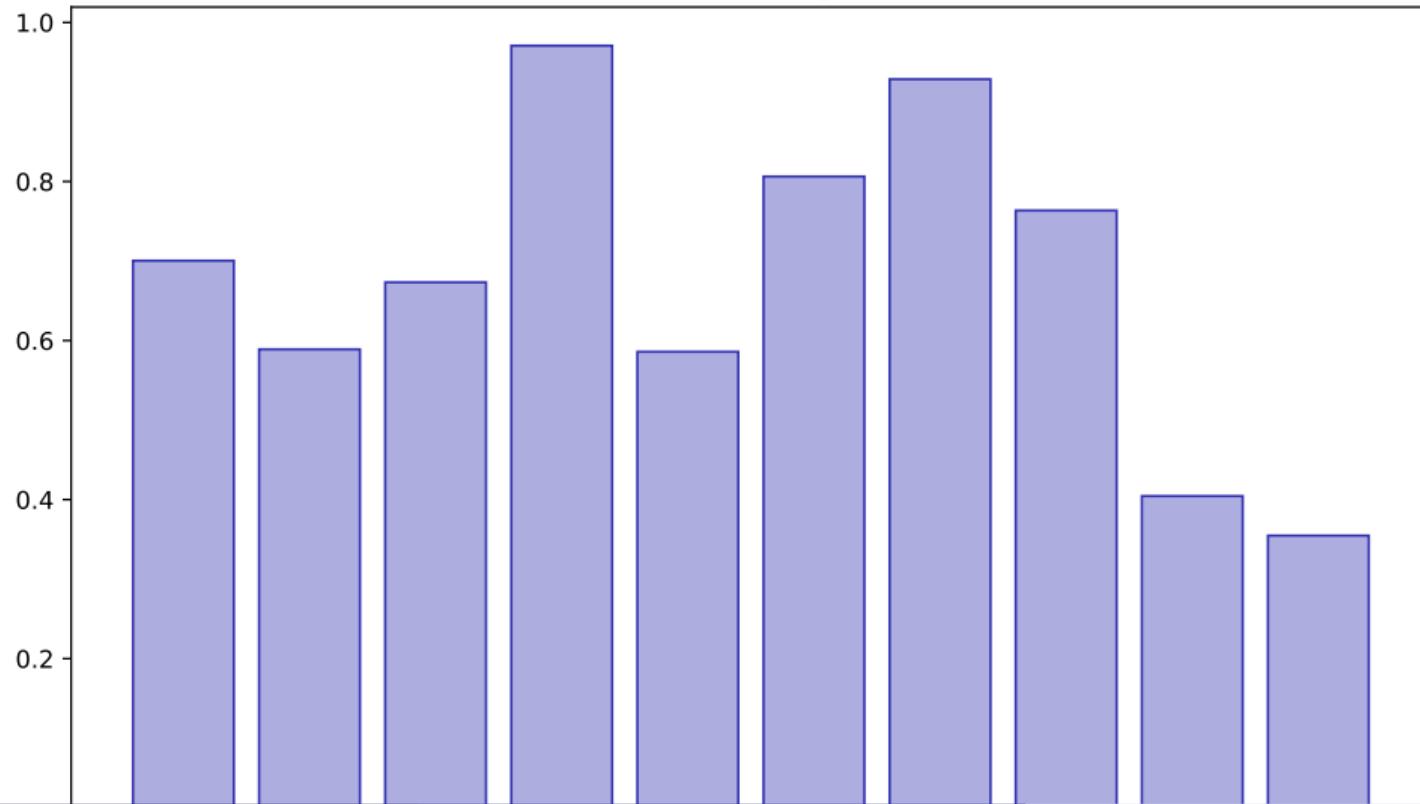
Ols Formula



Sklearn Api

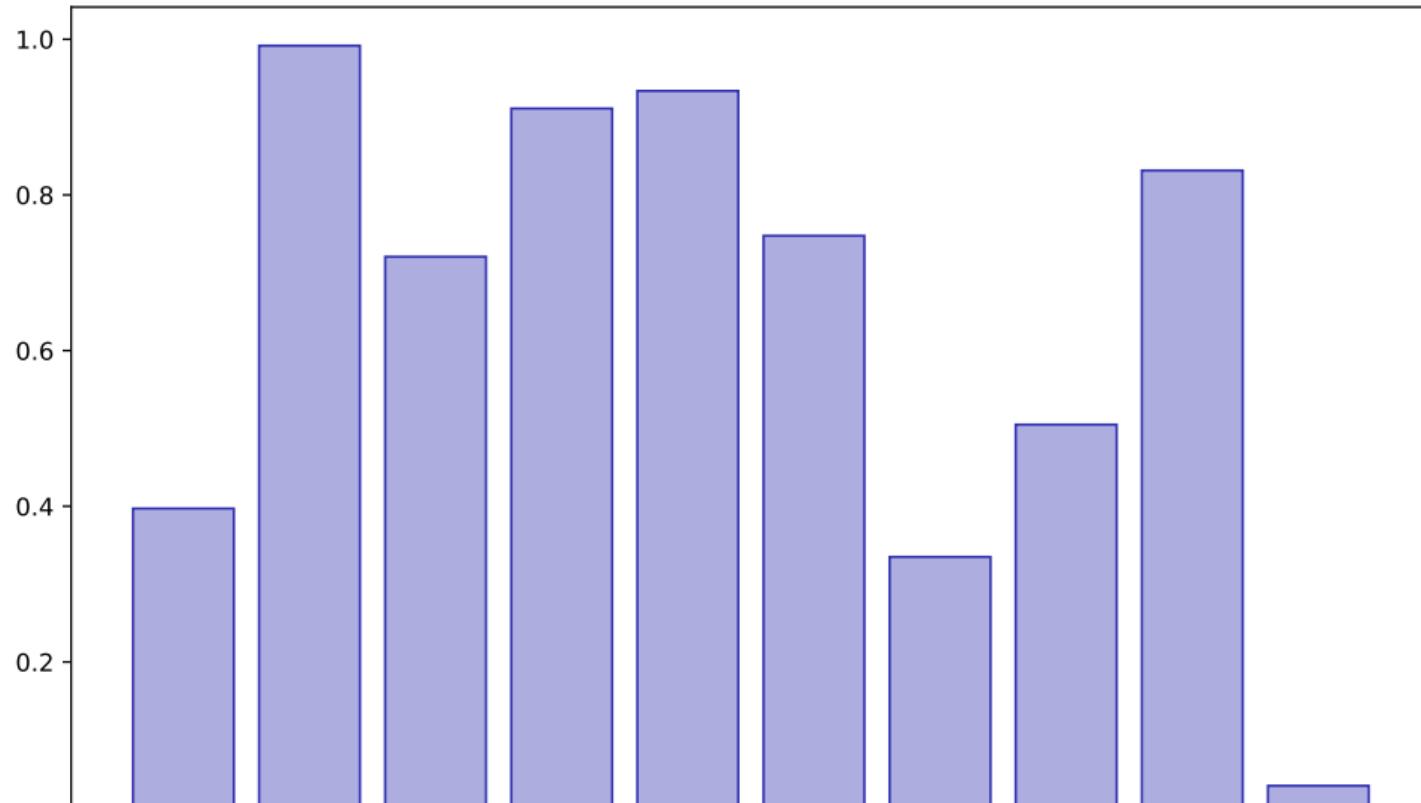


Coefficient Interpretation

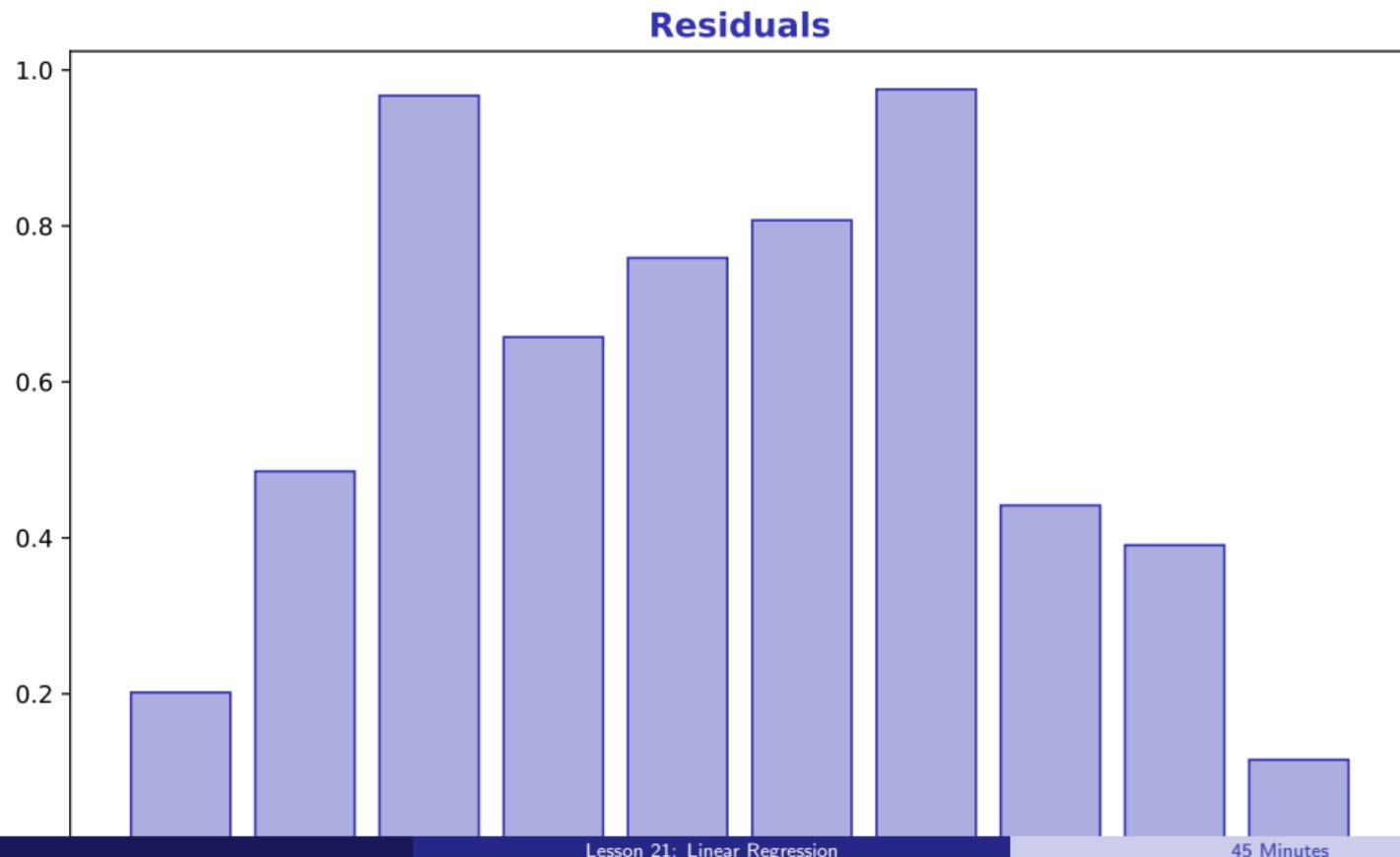


Prediction Line

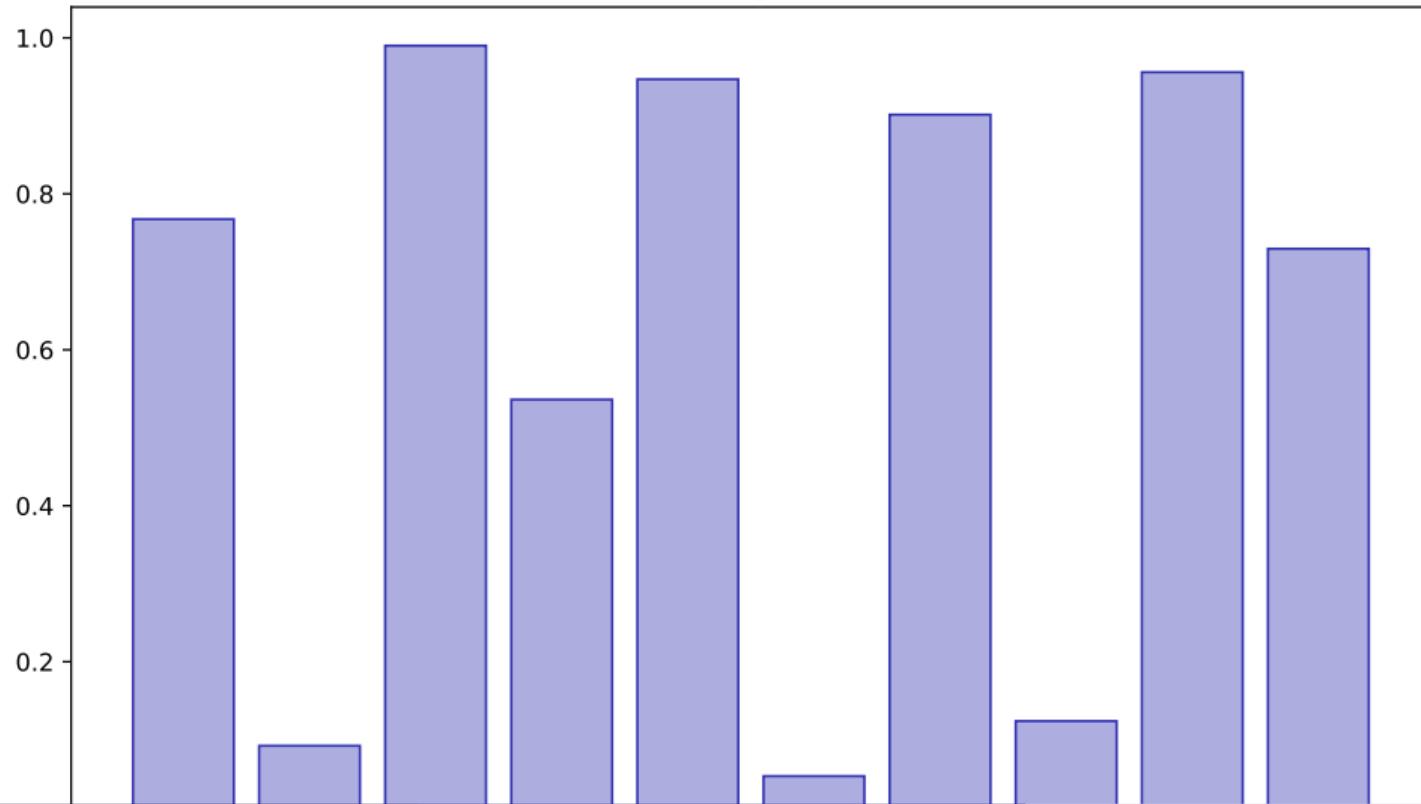
Prediction Line



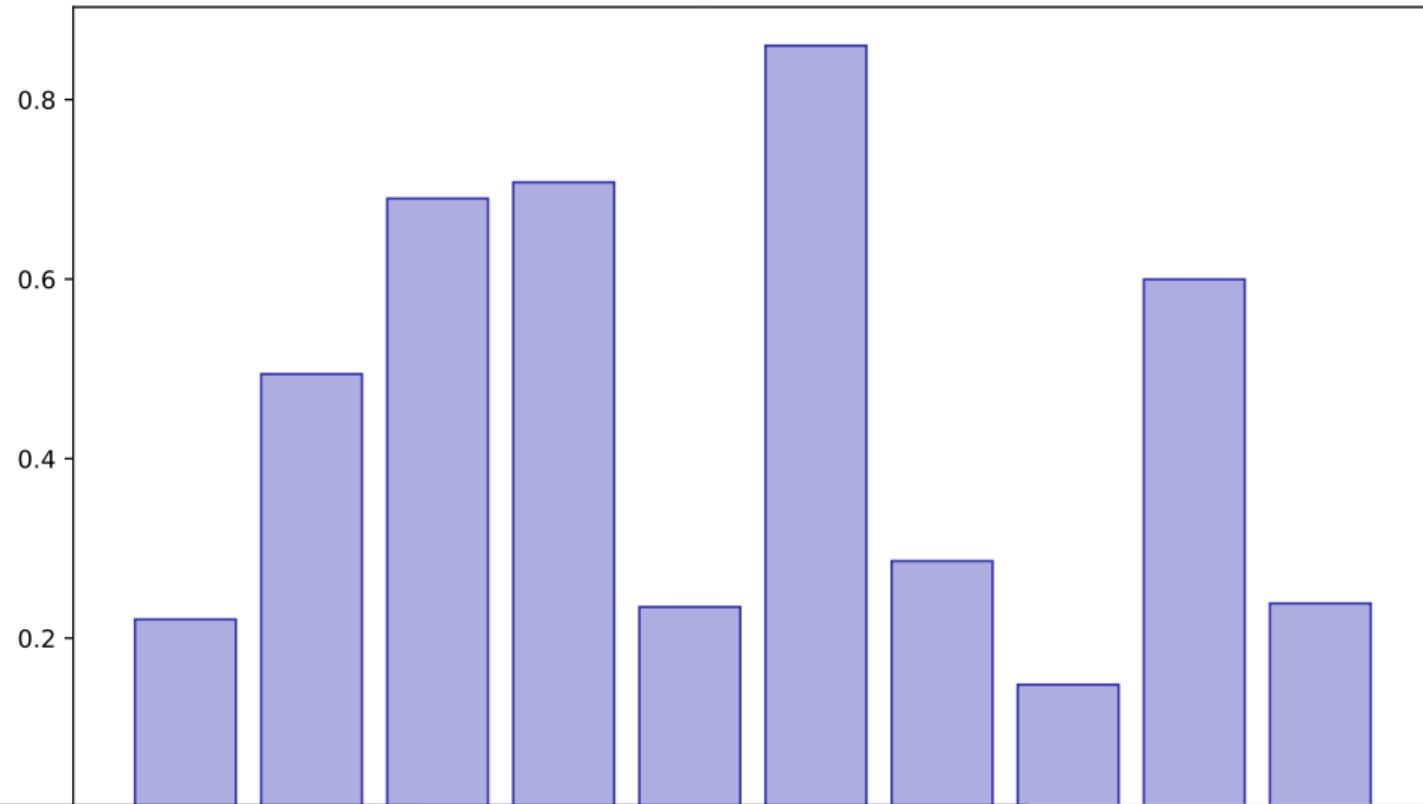
Residuals



Assumptions



Capm Beta



Lesson Summary

Key Takeaways:

- Understand OLS estimation
- Fit linear models with sklearn
- Interpret coefficients
- Apply to stock price prediction

Apply these skills in your final project

Lesson 22: Regularization

Data Science with Python – BSc Course

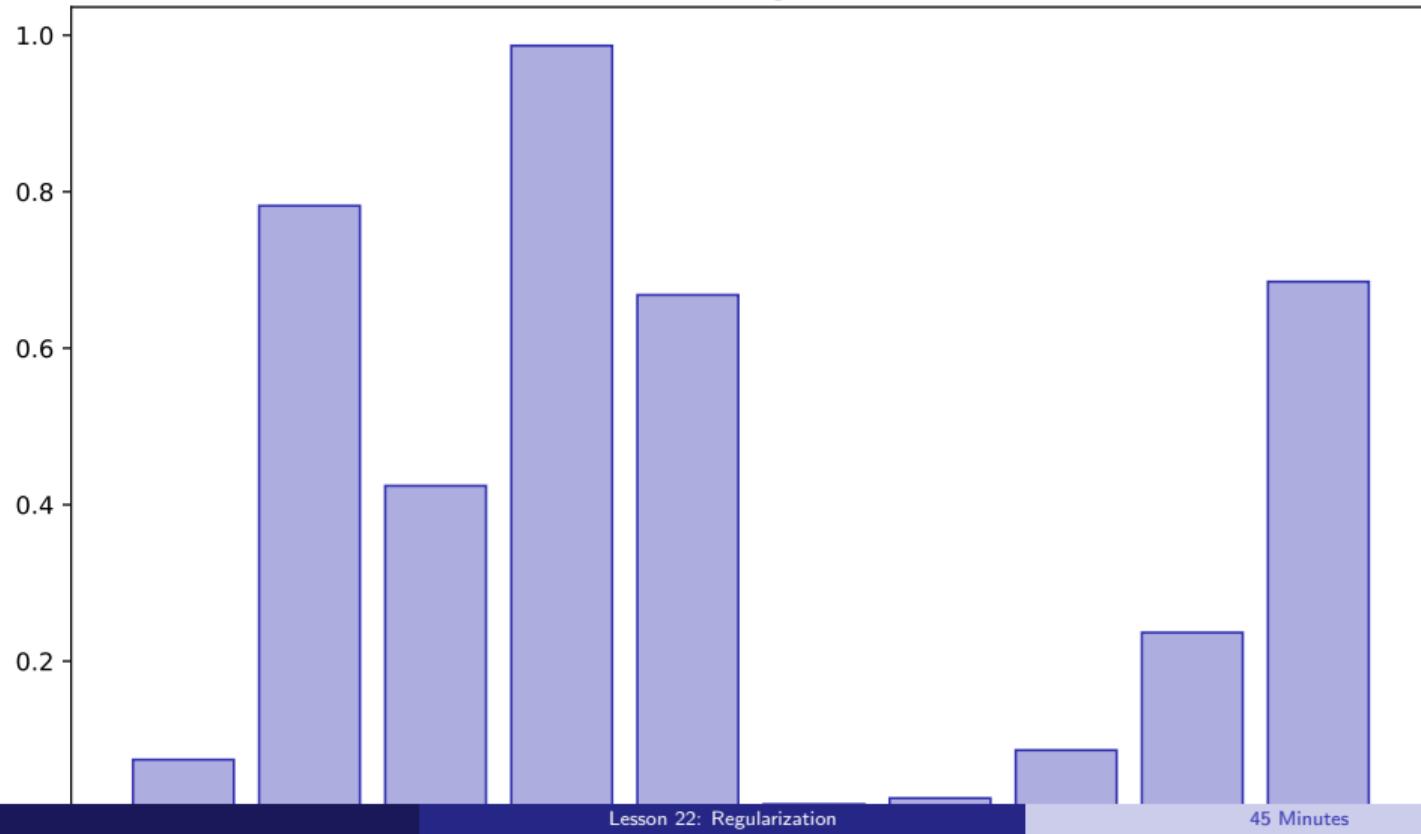
45 Minutes

After this lesson, you will be able to:

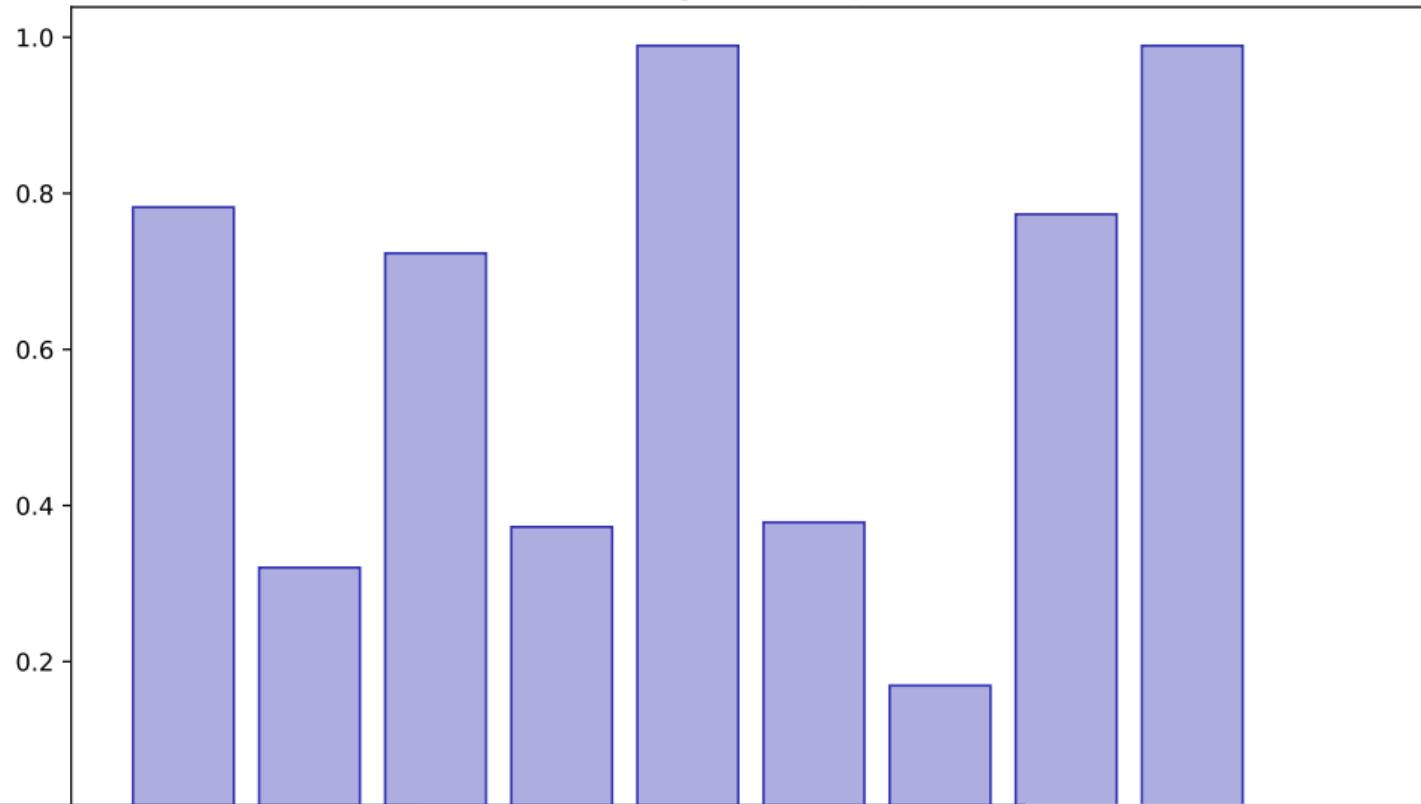
- Apply Ridge (L2) regularization
- Apply Lasso (L1) for feature selection
- Tune lambda with cross-validation
- Handle multicollinearity

Building towards your final project

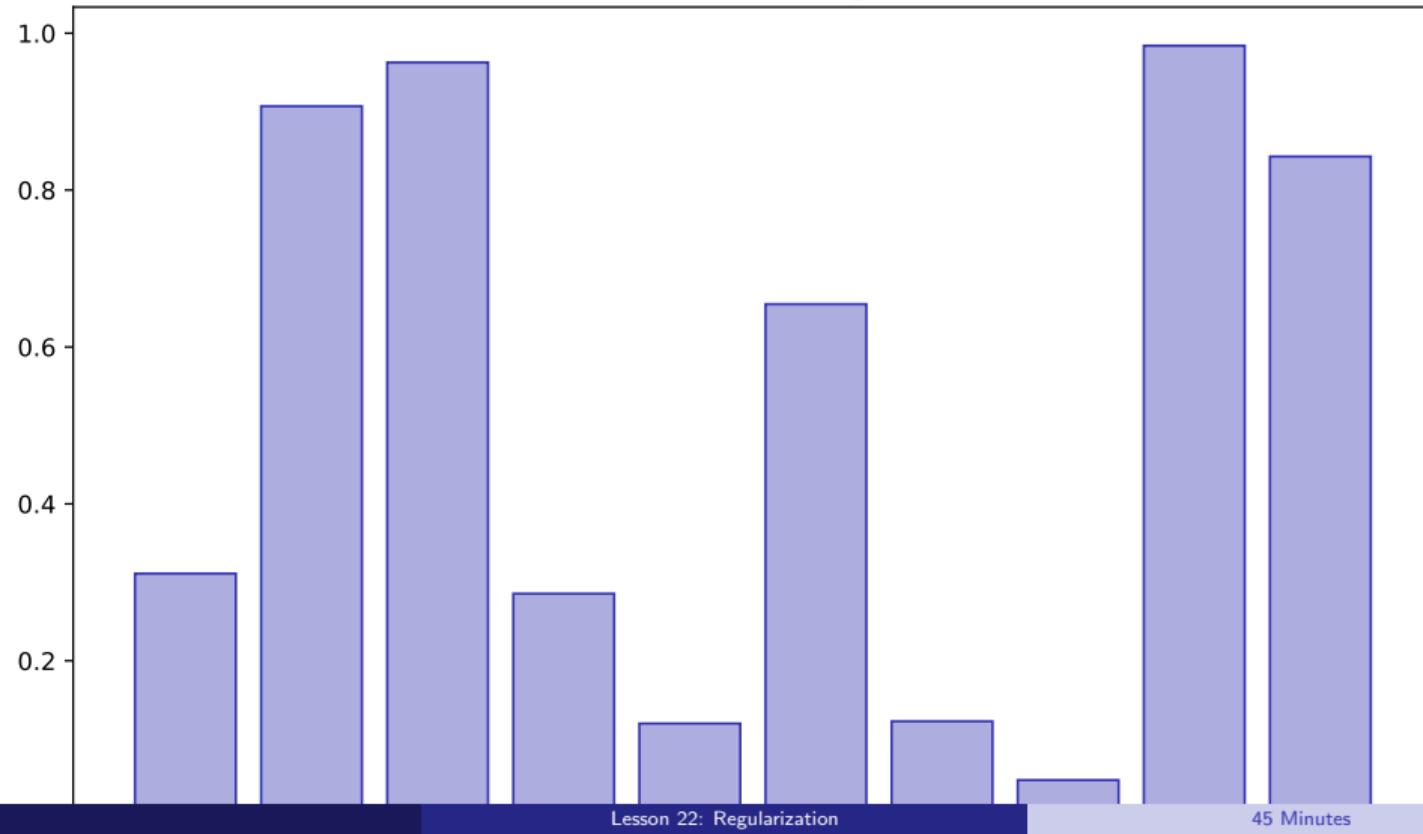
Overfitting Visual



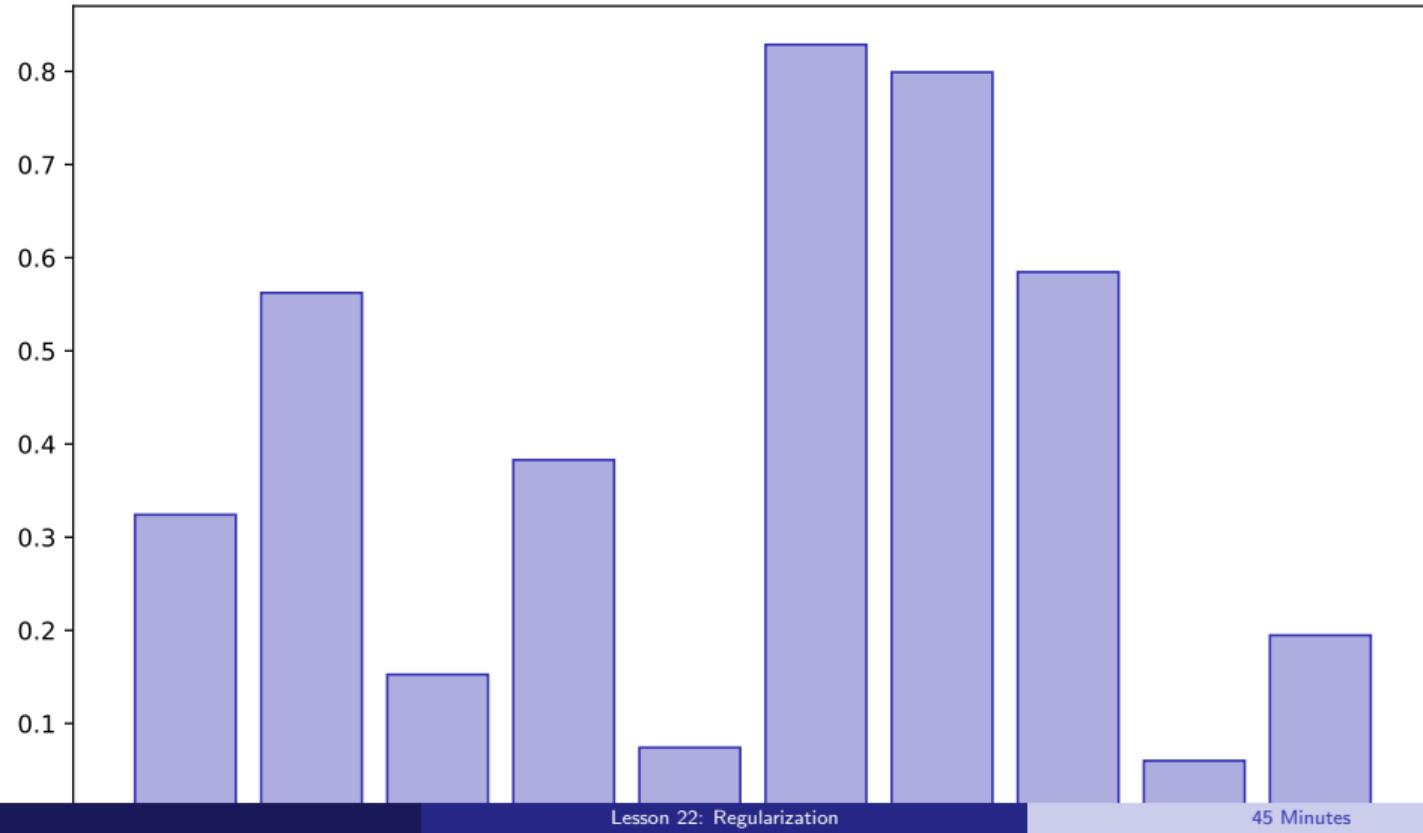
Ridge Concept



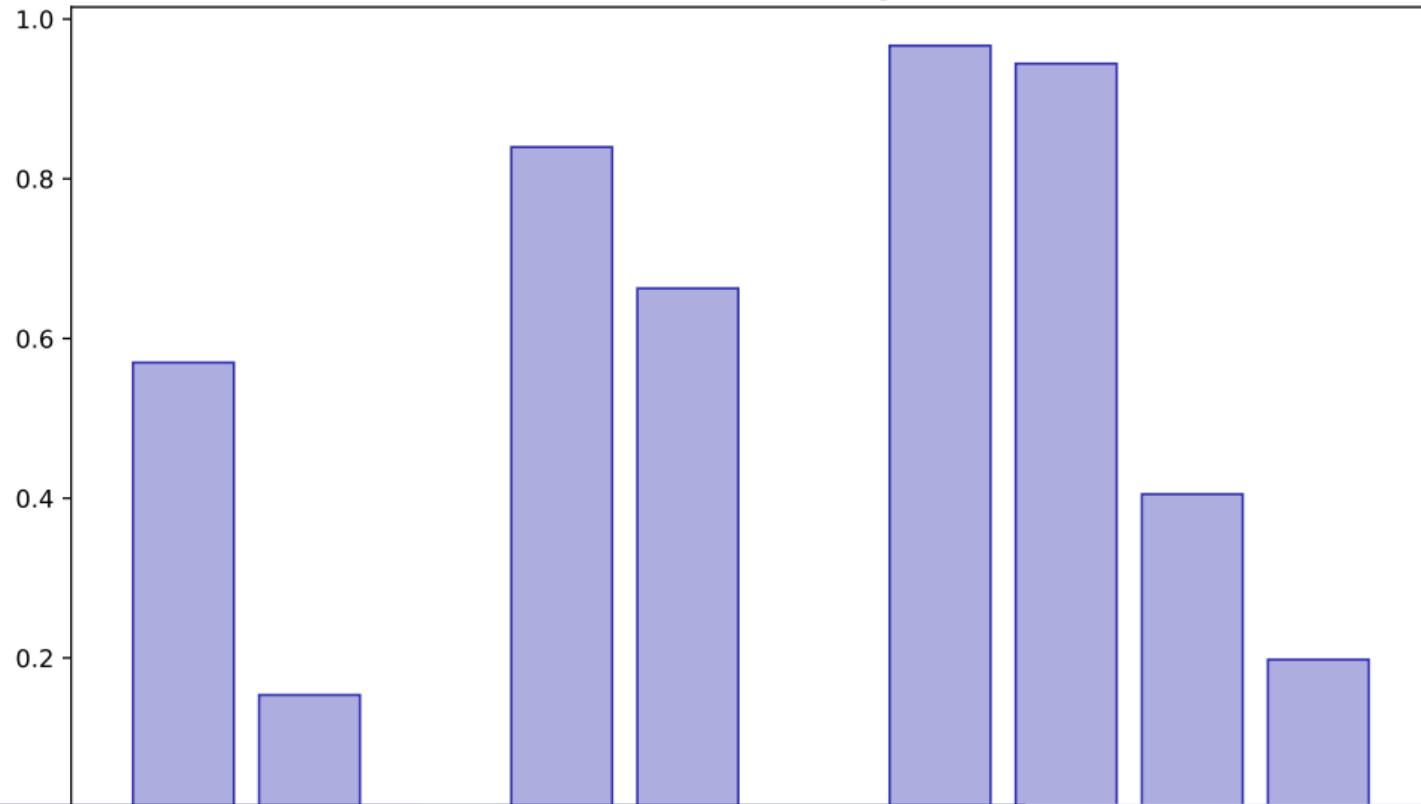
Lasso Concept



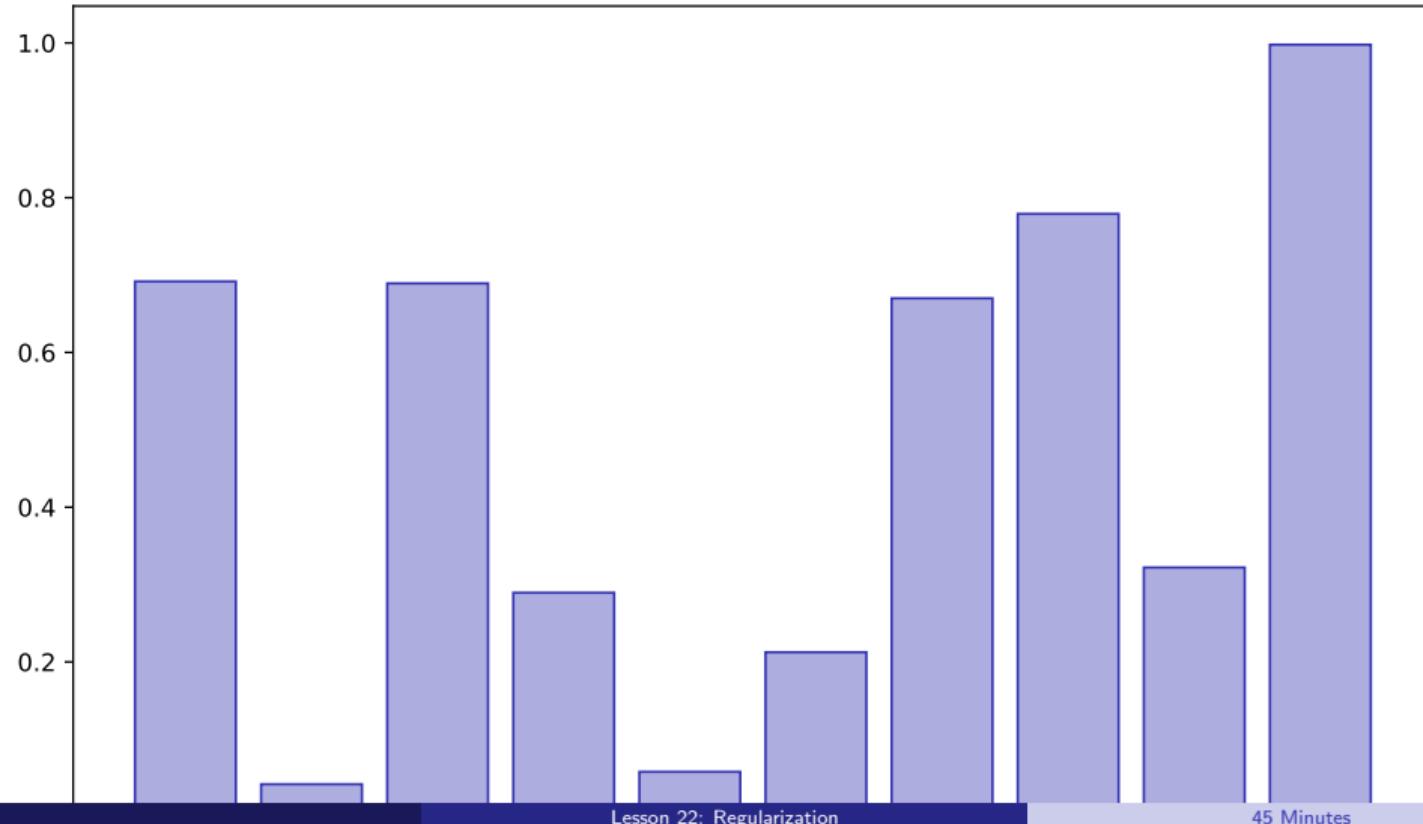
Coefficient Paths



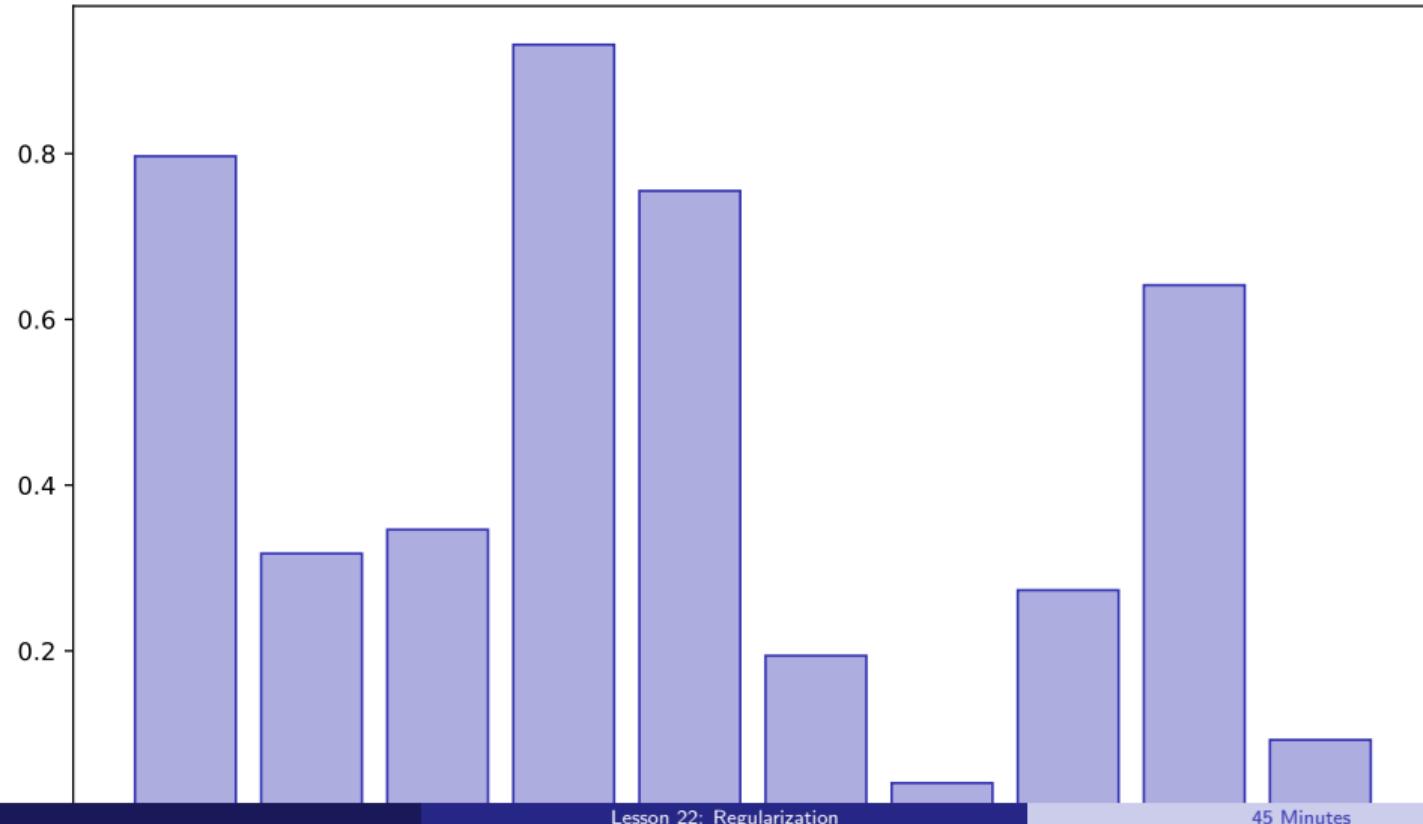
Lambda Tuning



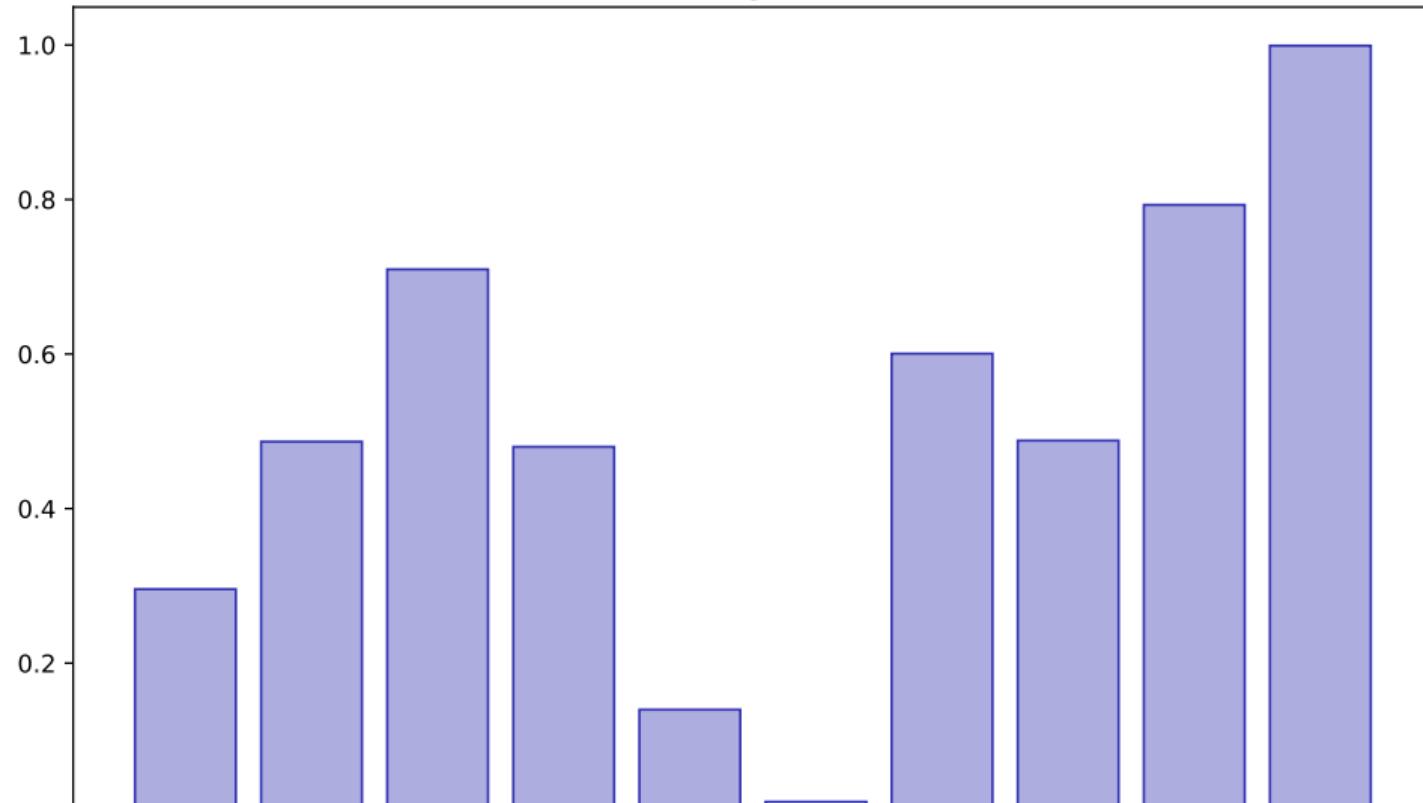
Cross Validation



Feature Selection



Finance Regularization



Key Takeaways:

- Apply Ridge (L2) regularization
- Apply Lasso (L1) for feature selection
- Tune lambda with cross-validation
- Handle multicollinearity

Apply these skills in your final project

Lesson 23: Regression Metrics

Data Science with Python – BSc Course

45 Minutes

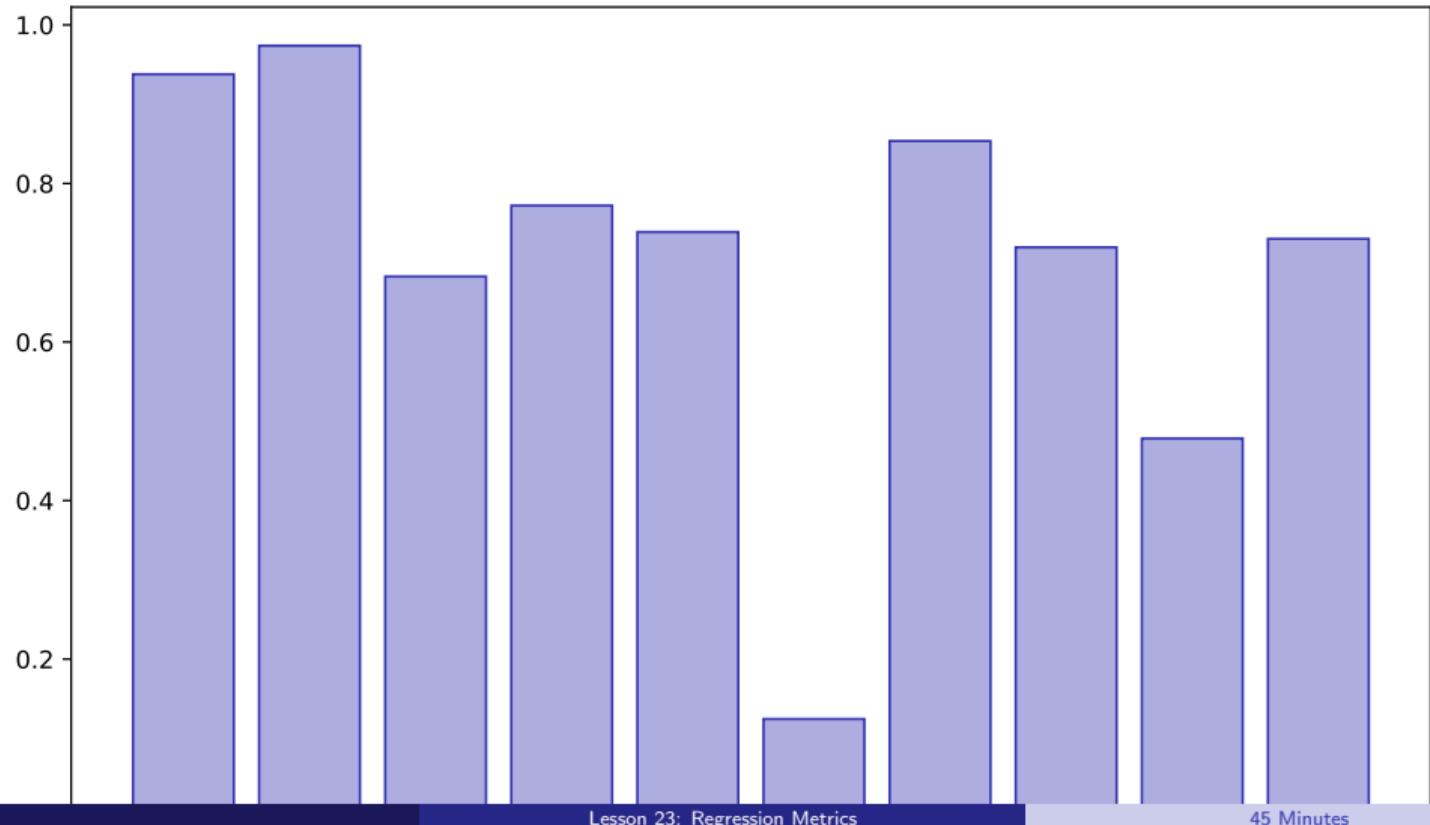
Learning Objectives

After this lesson, you will be able to:

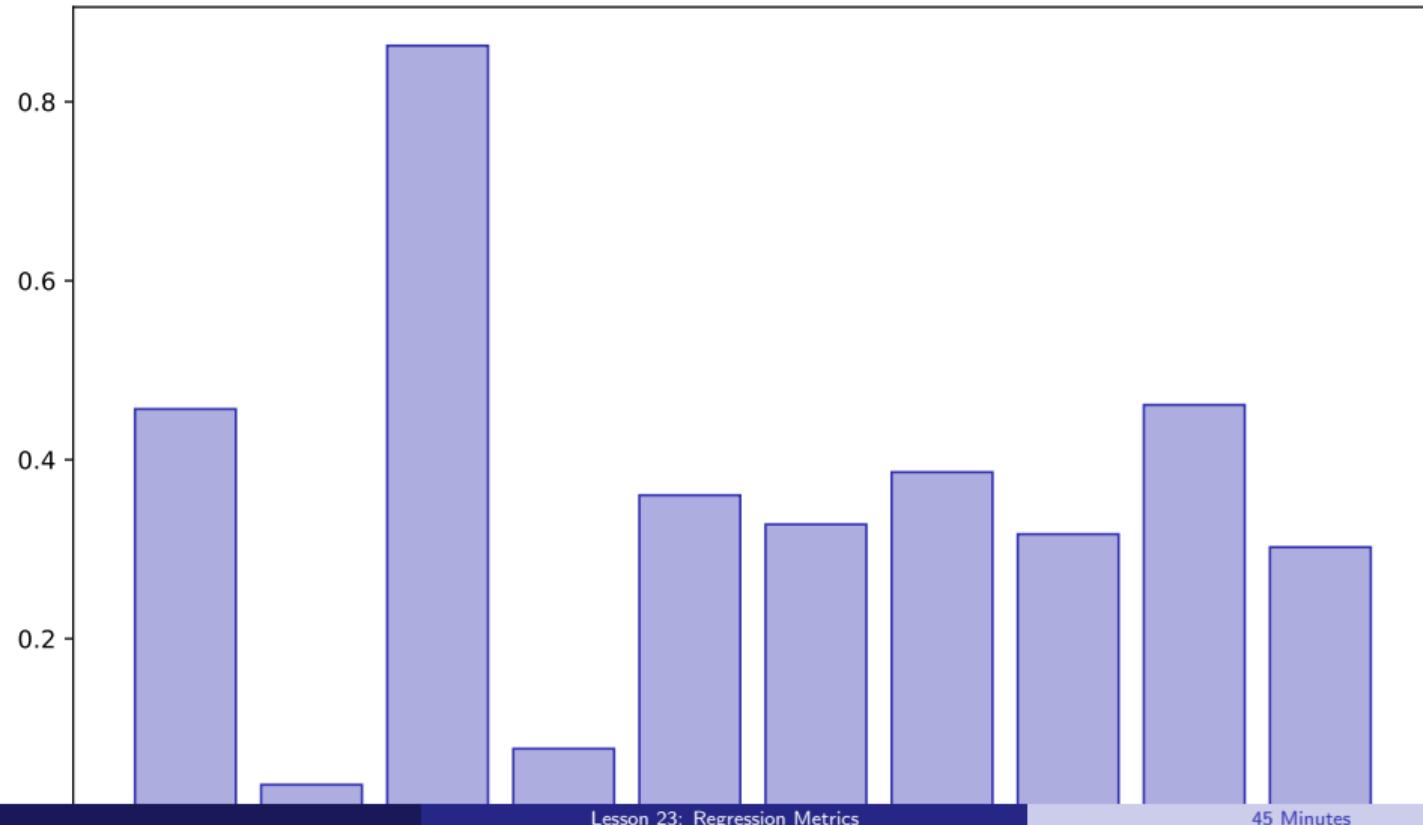
- Calculate MSE, RMSE, MAE
- Interpret R-squared
- Compare models fairly
- Handle time series validation

Building towards your final project

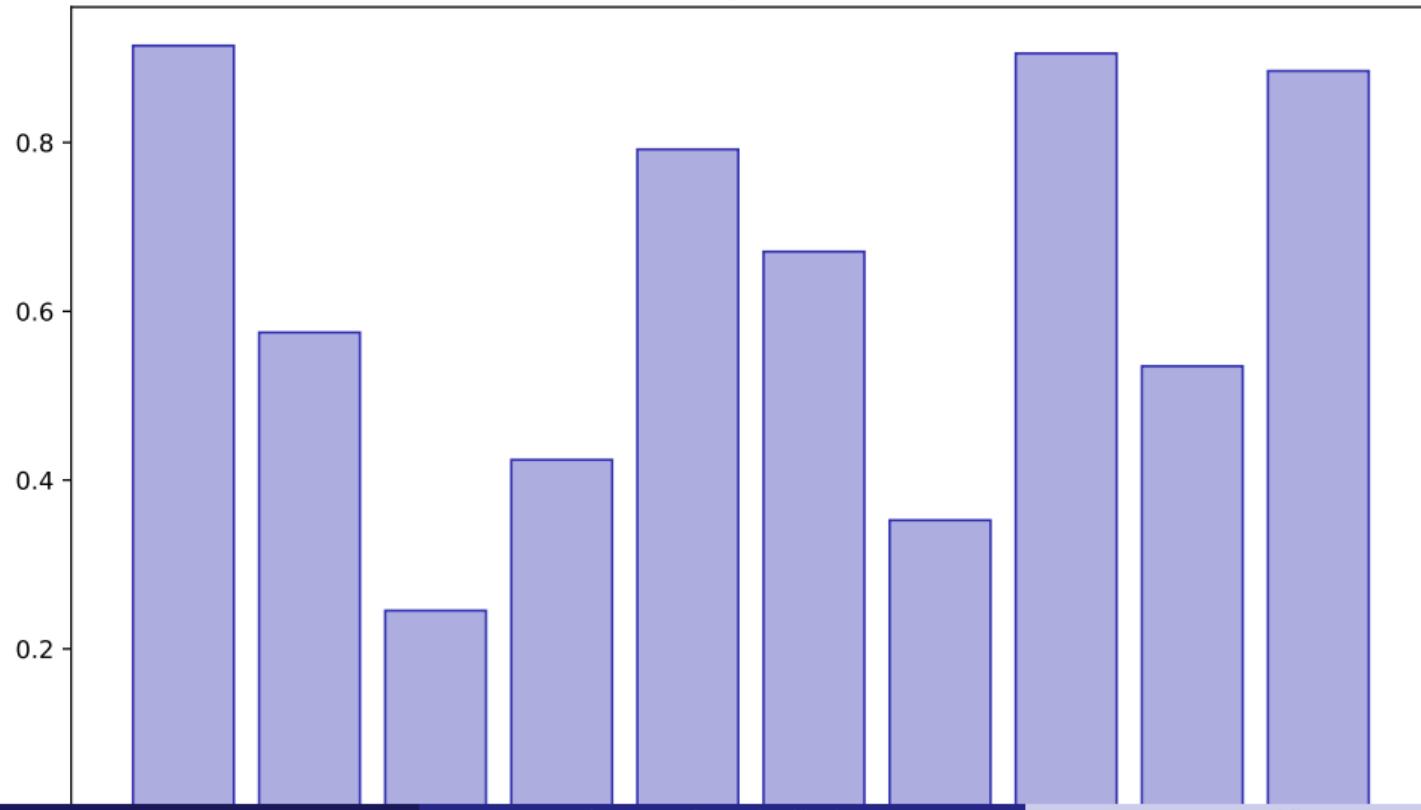
Mse Formula



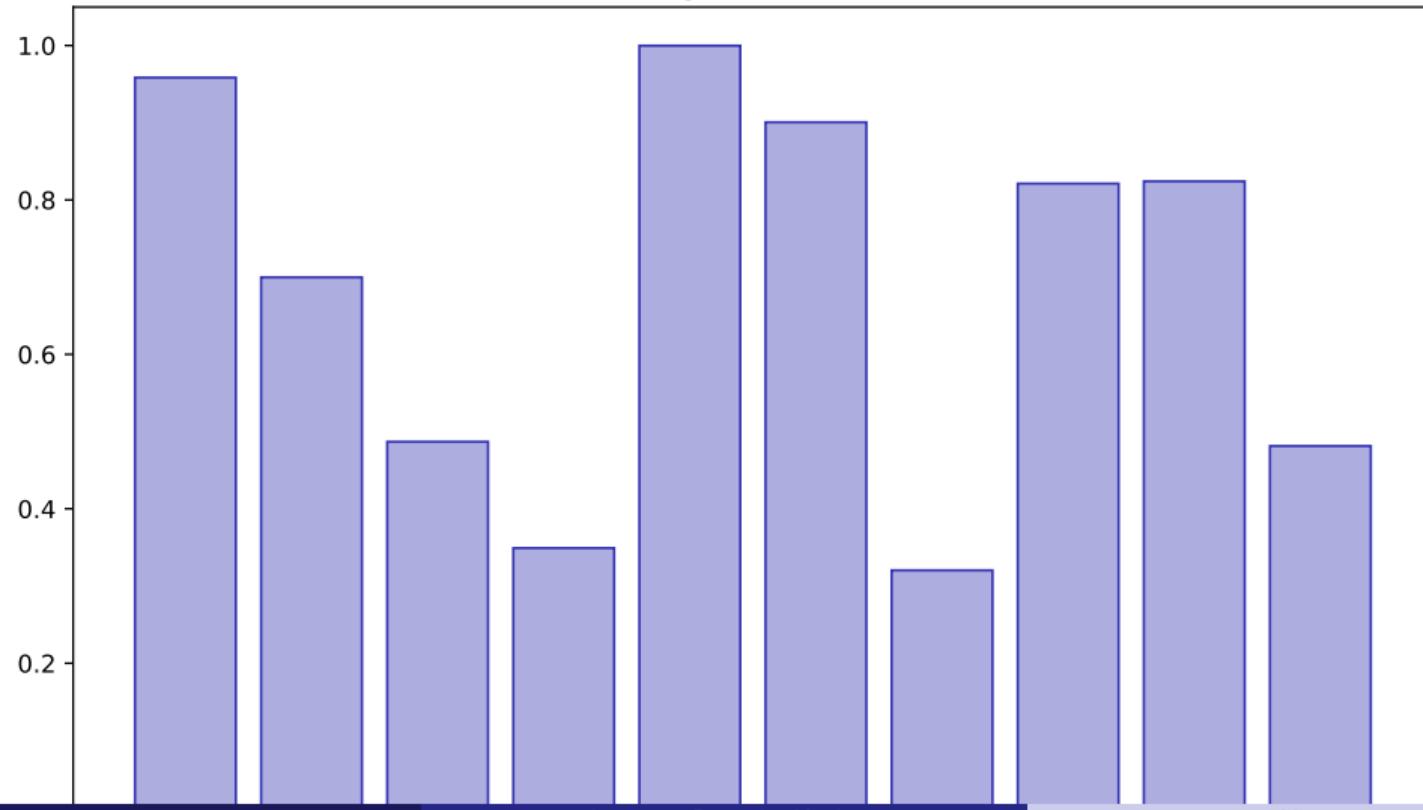
Rmse Mae



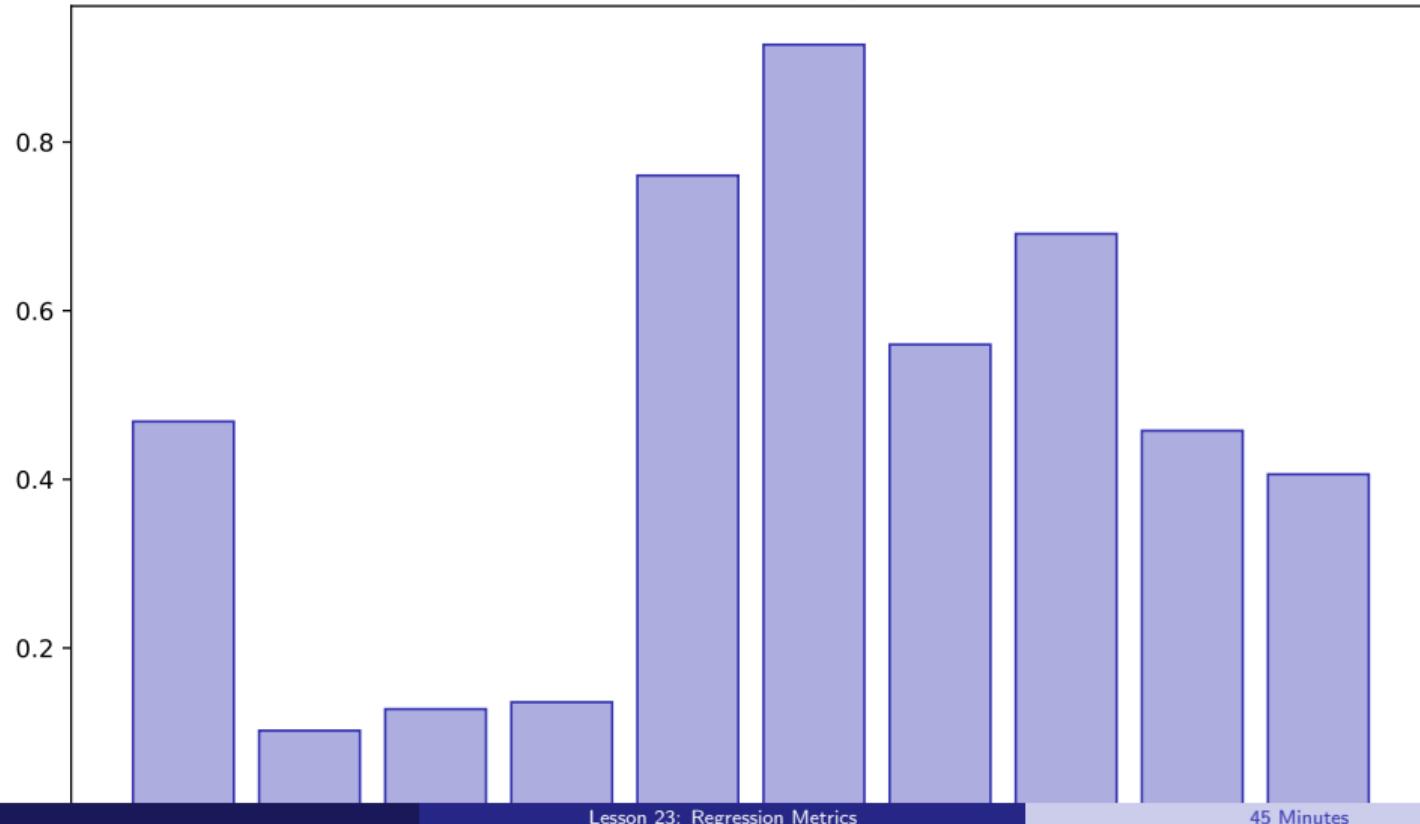
R Squared



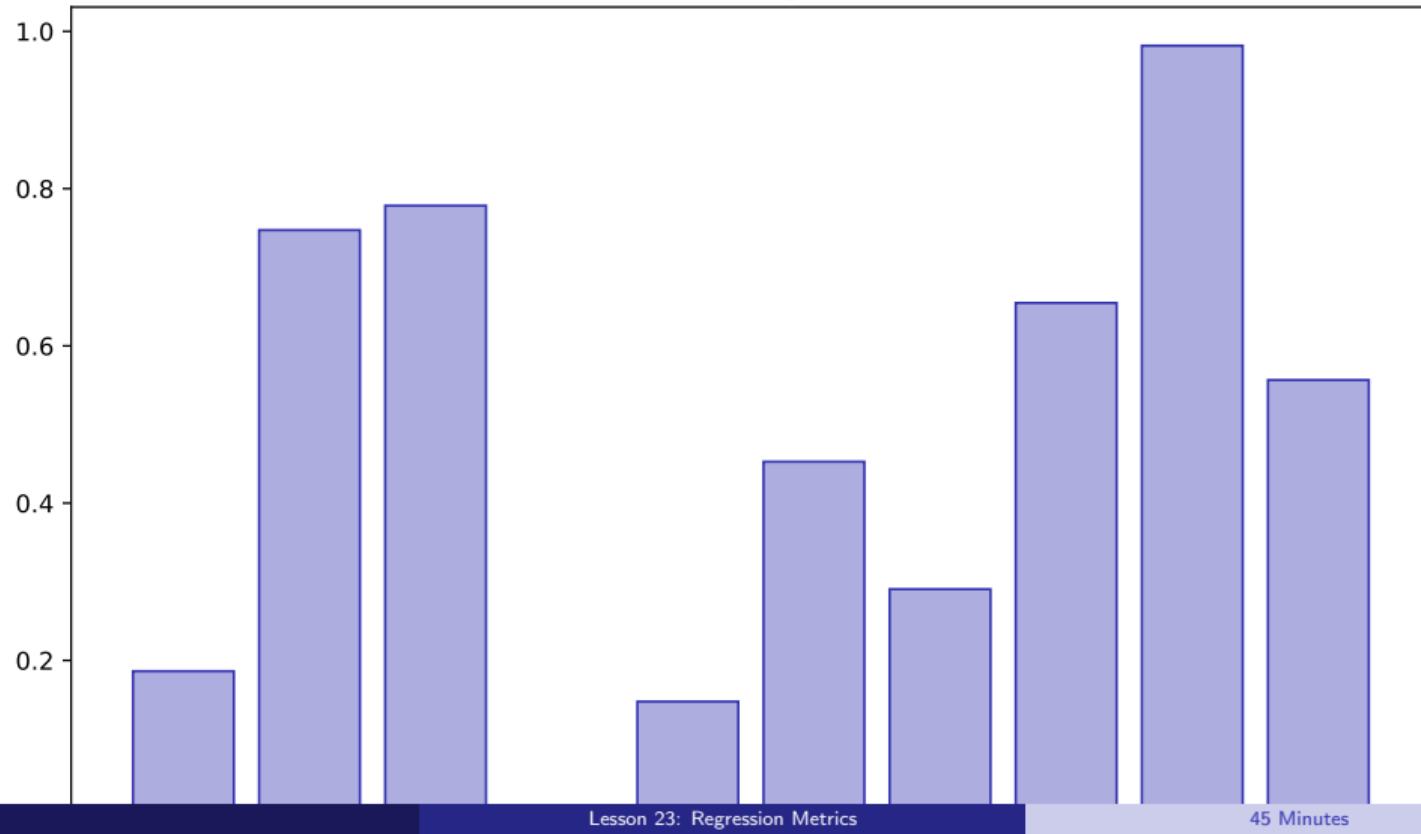
Adjusted R2



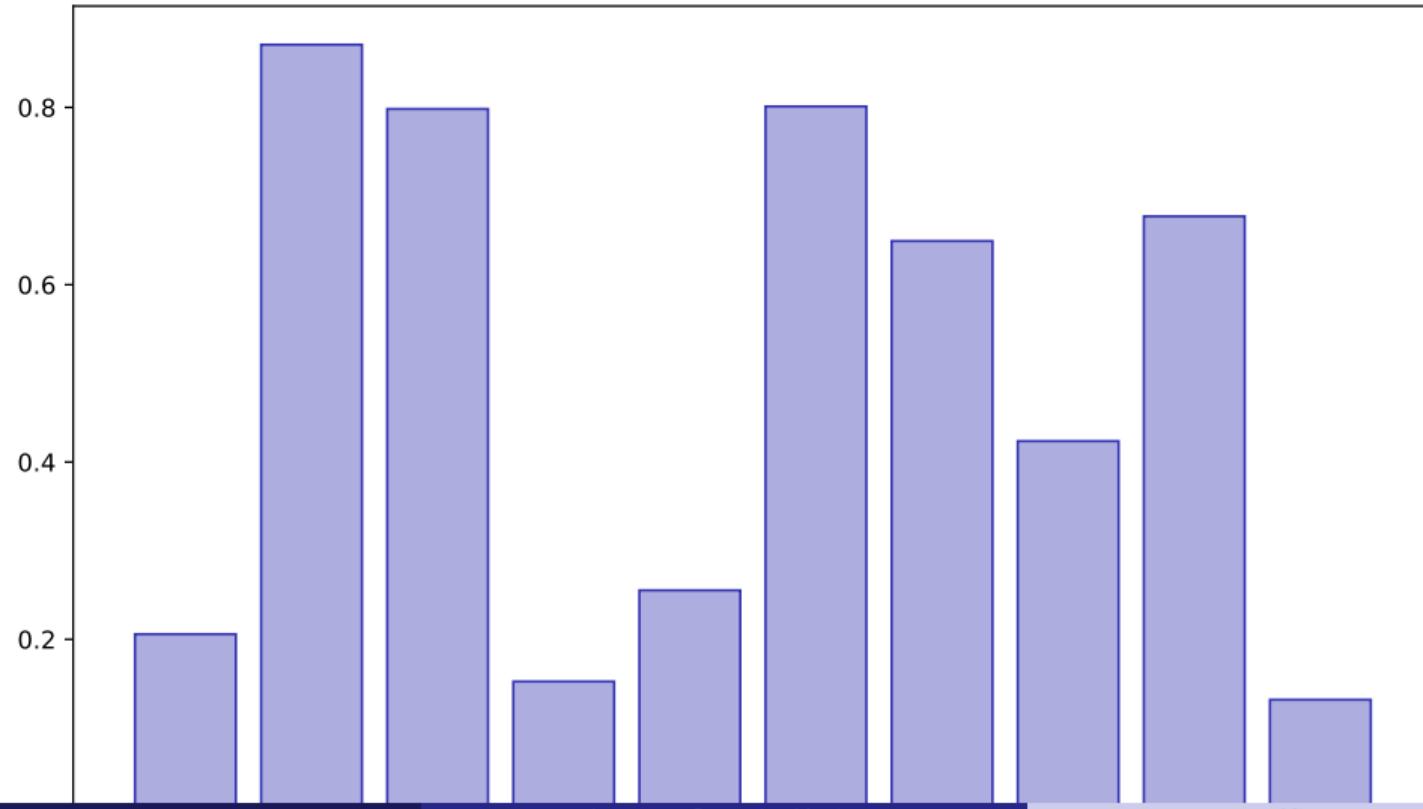
Residual Analysis



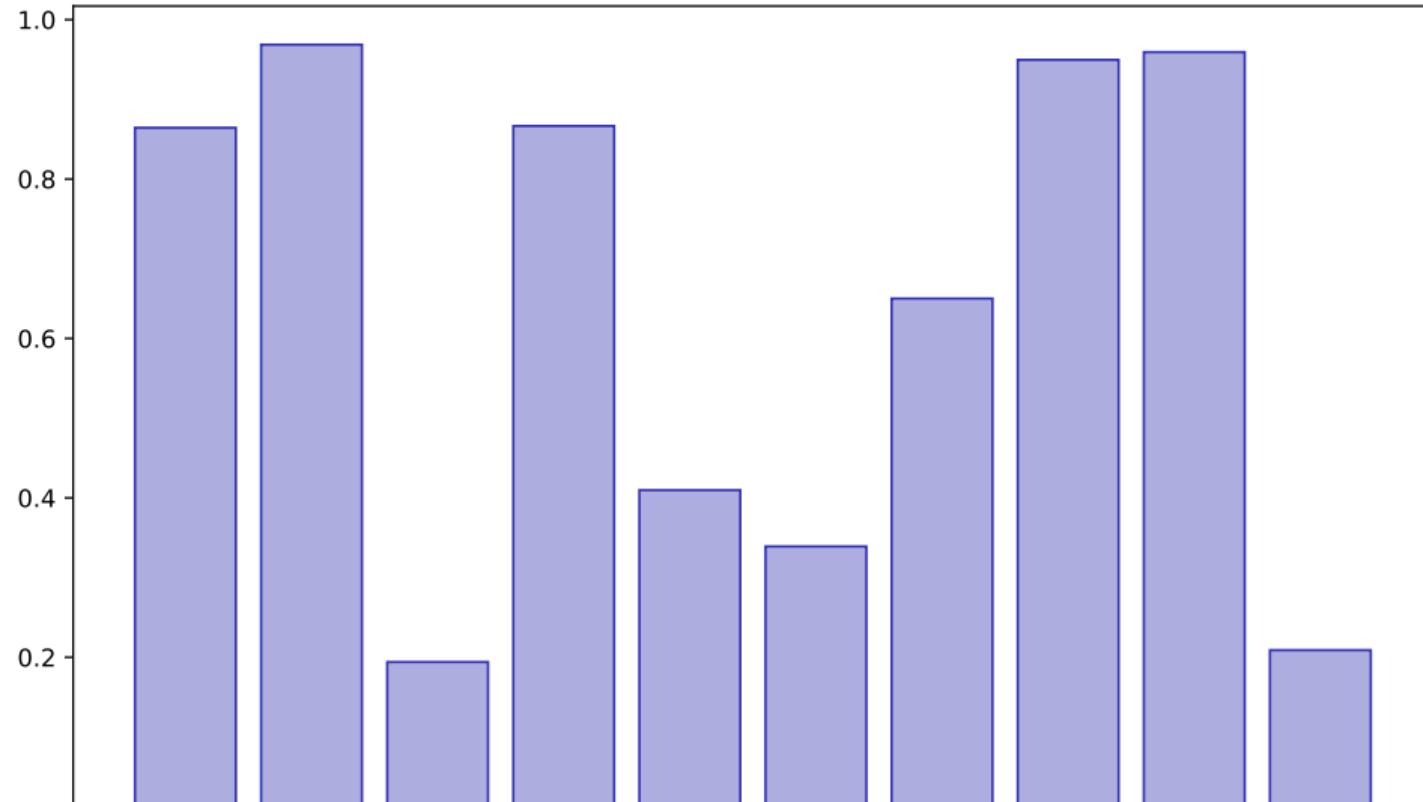
Model Comparison



Time Series Cv



Finance Metrics



Key Takeaways:

- Calculate MSE, RMSE, MAE
- Interpret R-squared
- Compare models fairly
- Handle time series validation

Apply these skills in your final project

Lesson 24: Factor Models

Data Science with Python – BSc Course

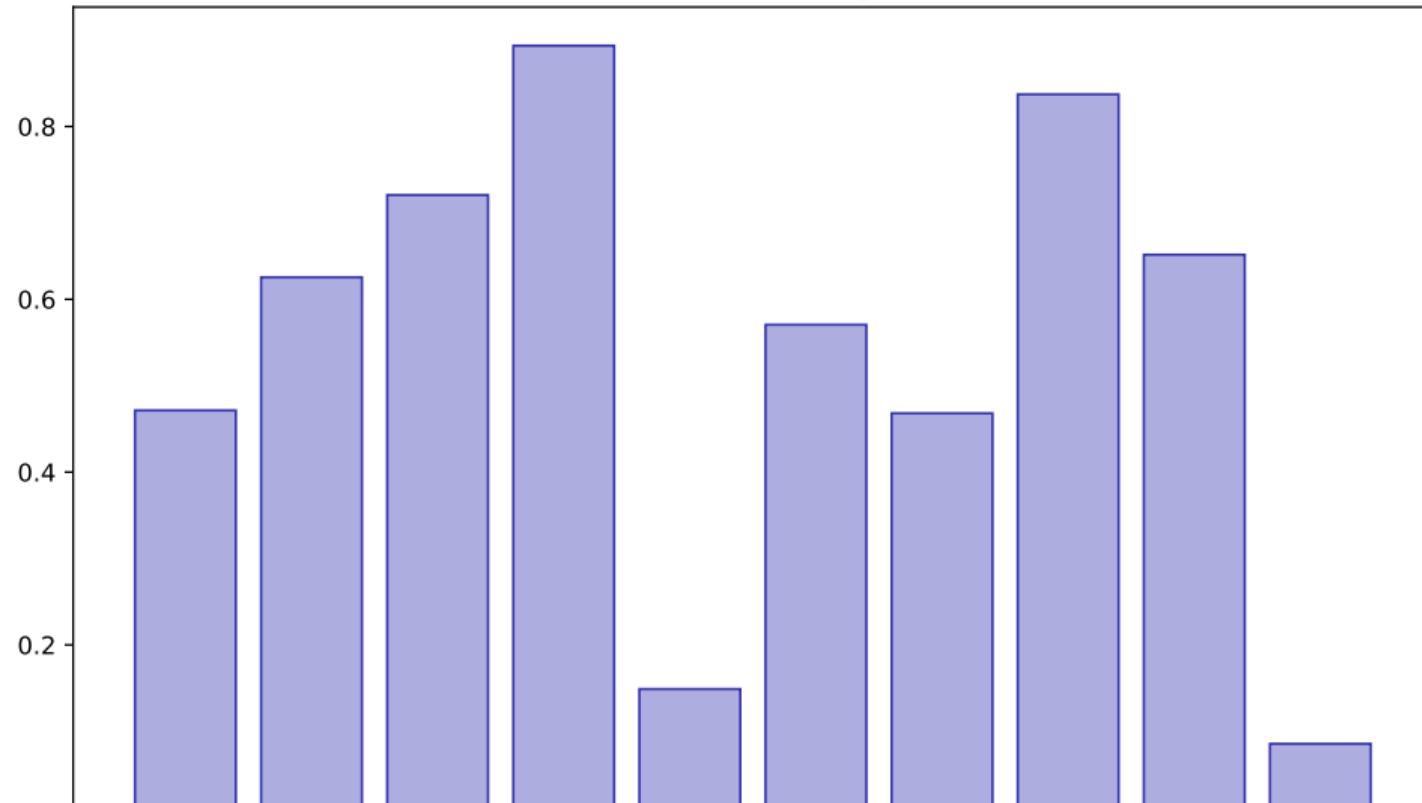
45 Minutes

After this lesson, you will be able to:

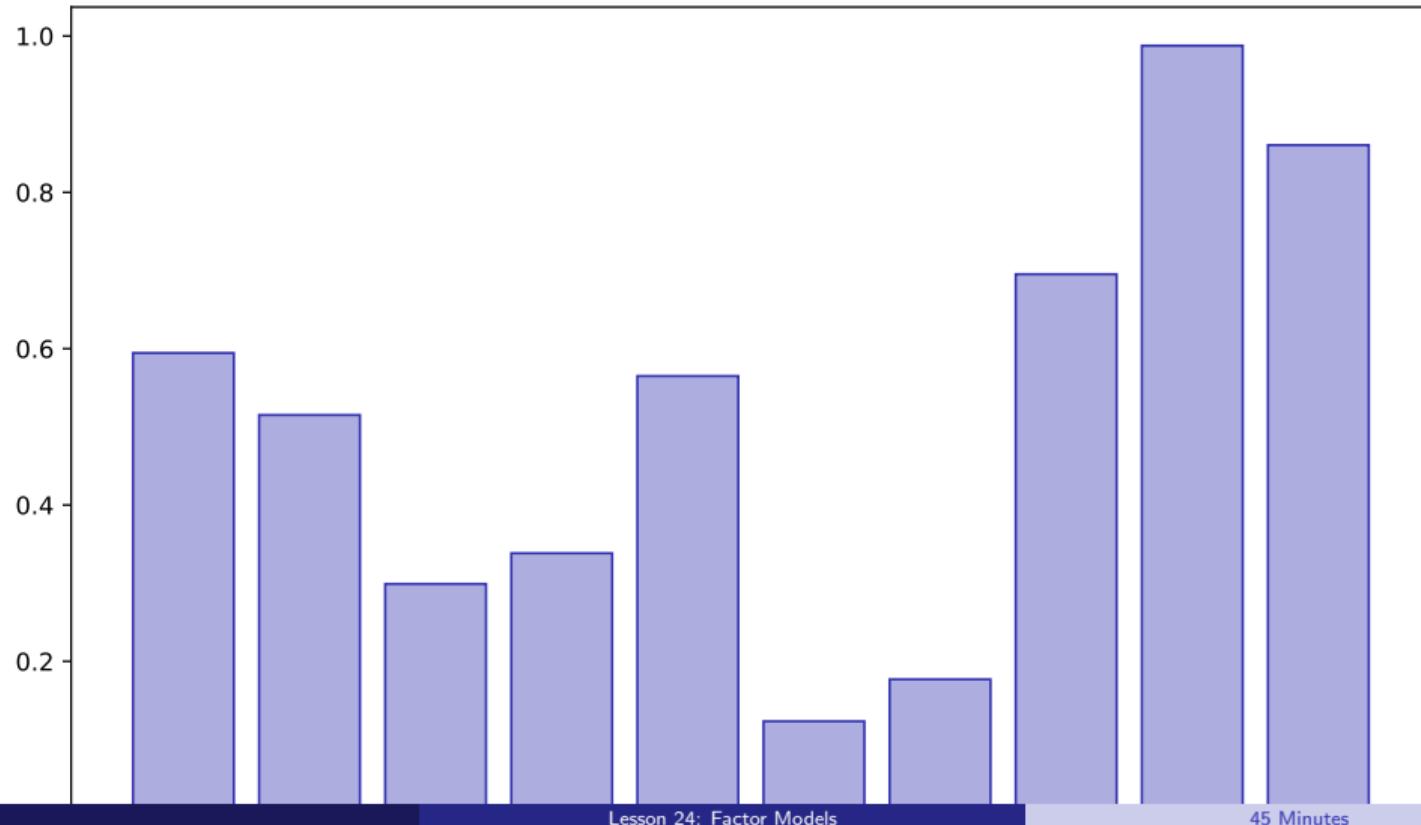
- Build multi-factor regression
- Understand Fama-French factors
- Interpret factor loadings
- Create complete ML pipelines

Building towards your final project

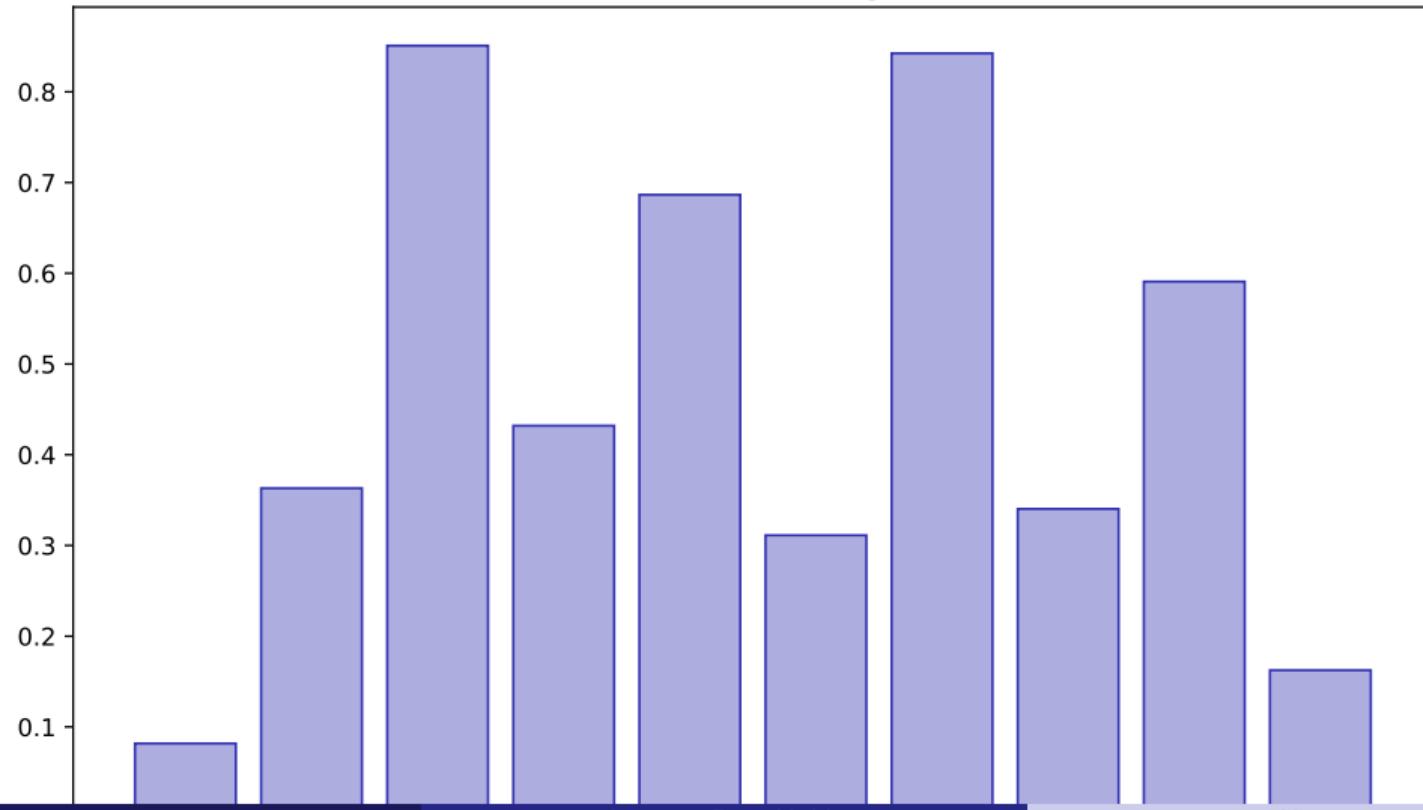
Factor Concept



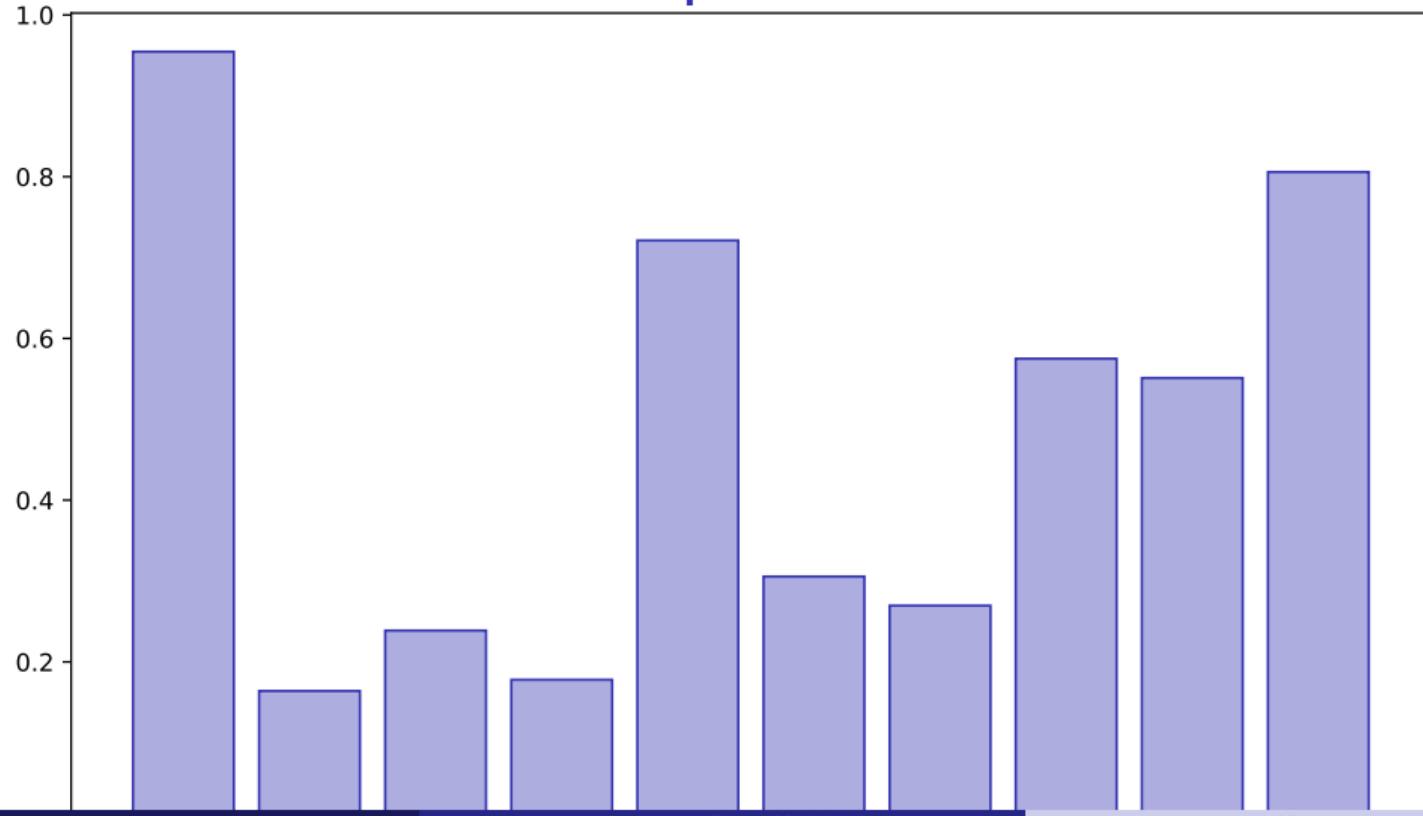
Fama French



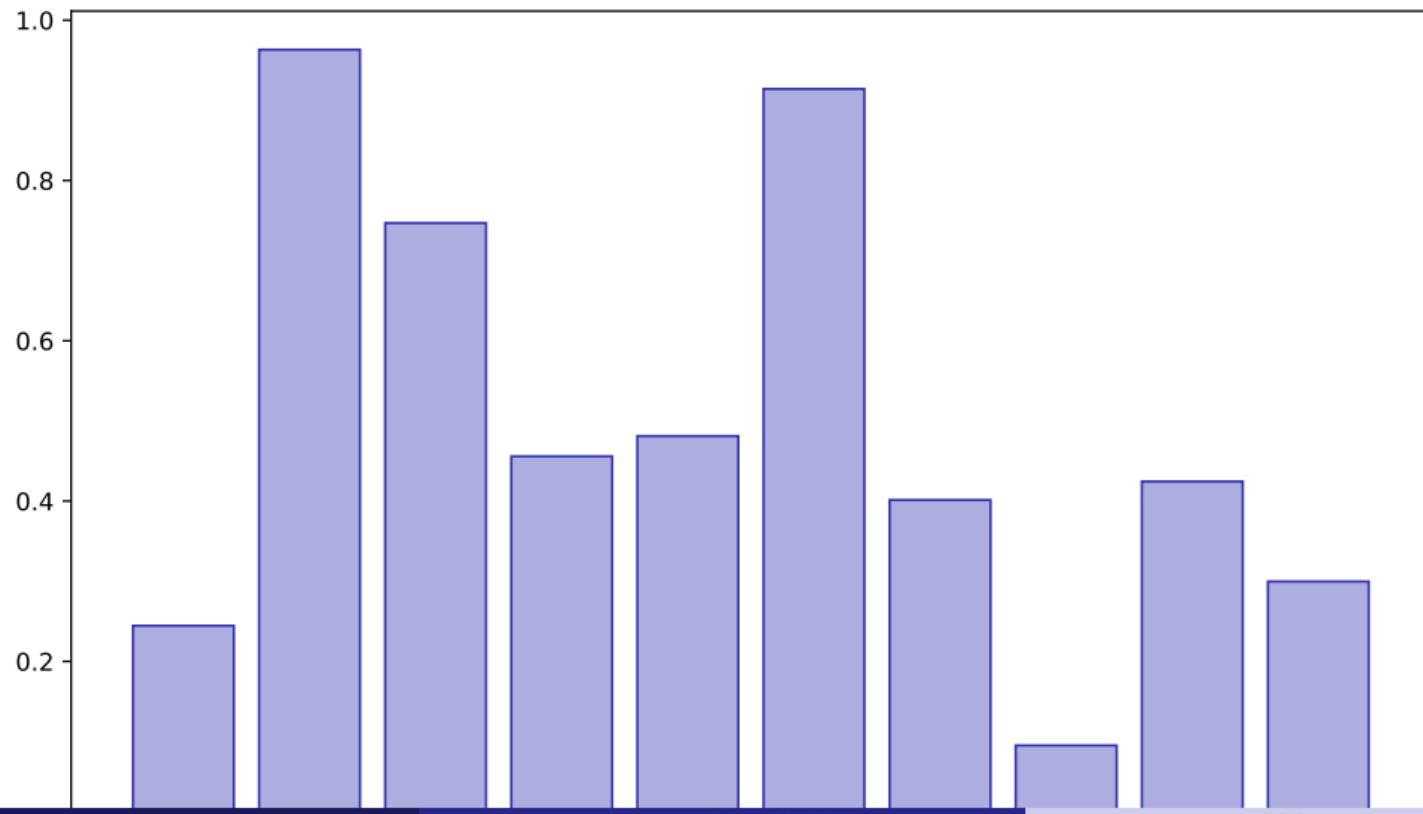
Factor Loadings



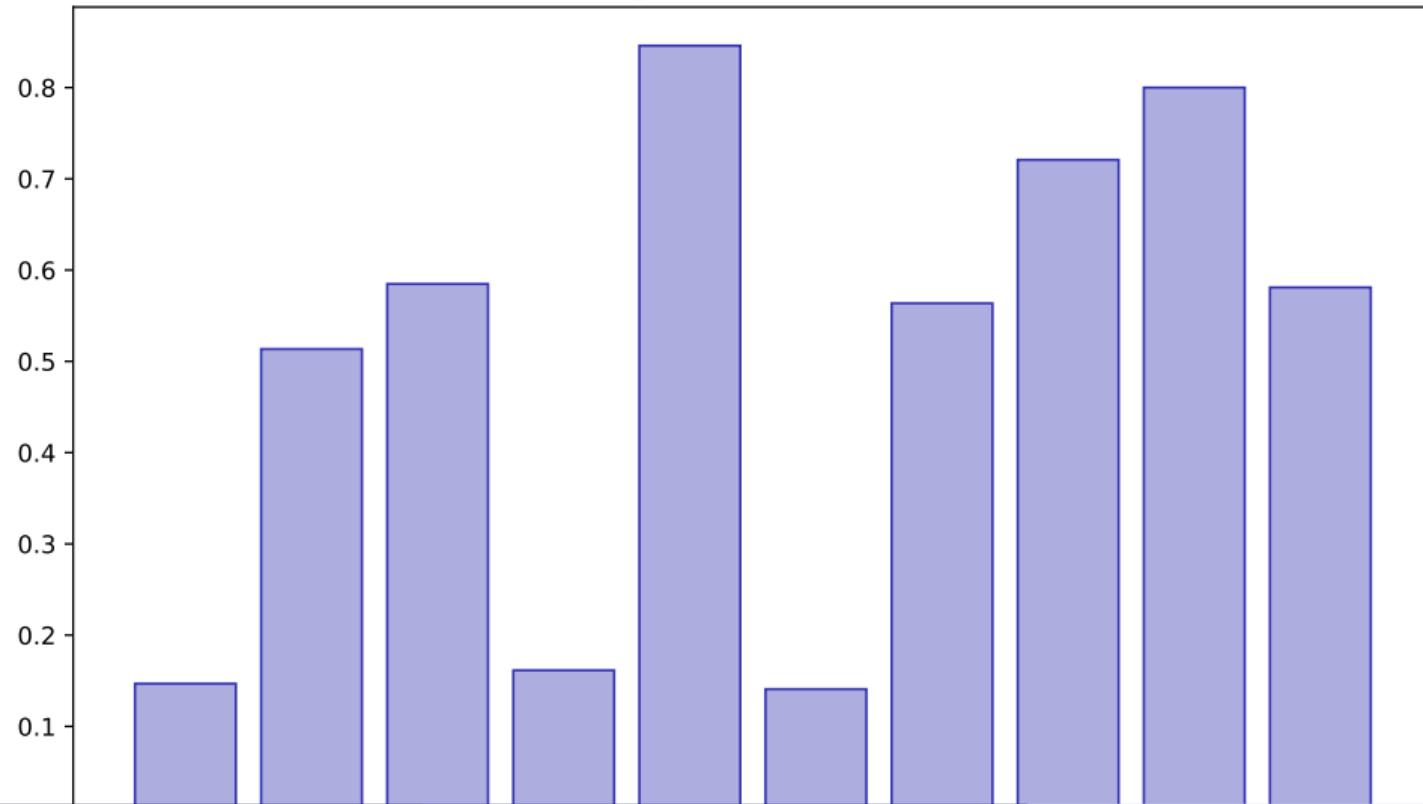
Alpha Beta



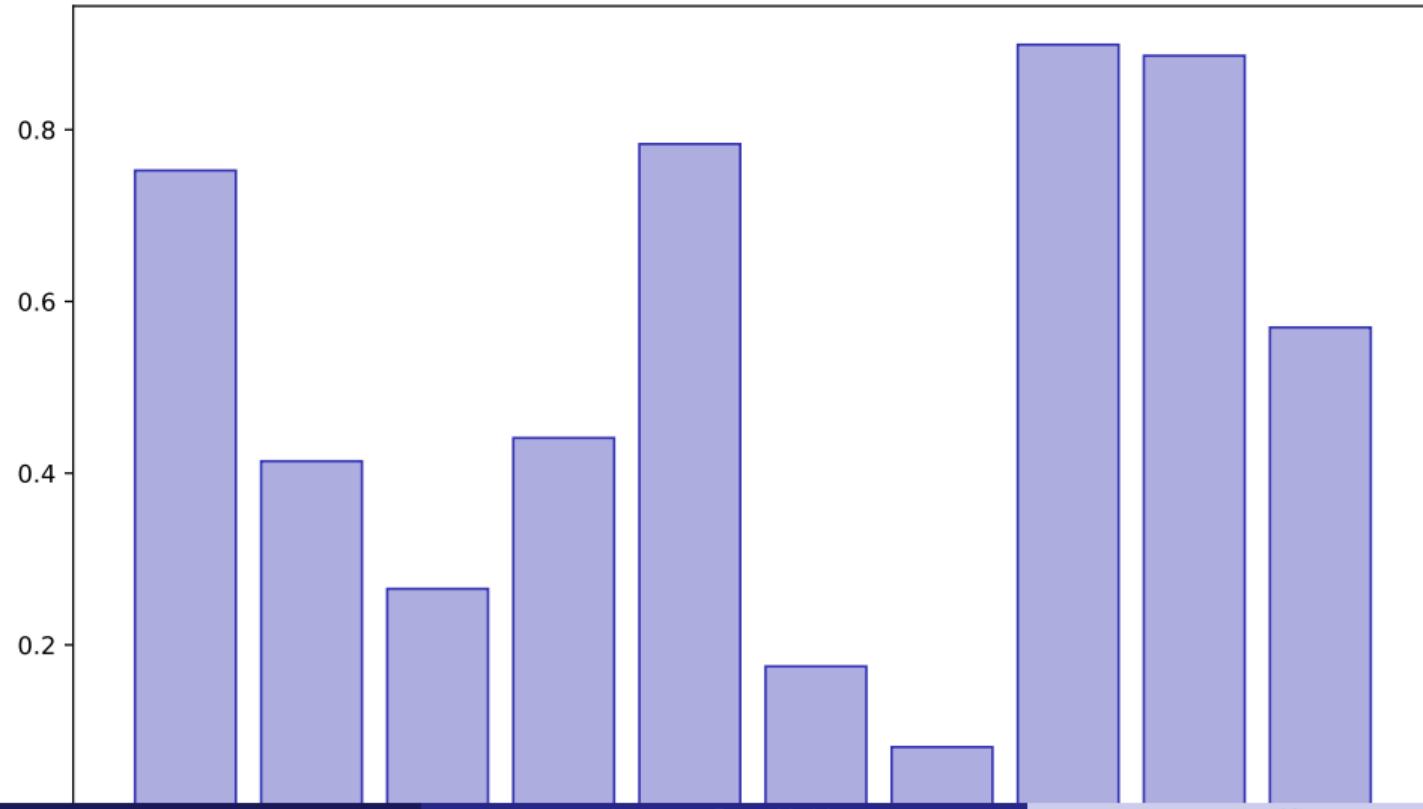
Multi Factor



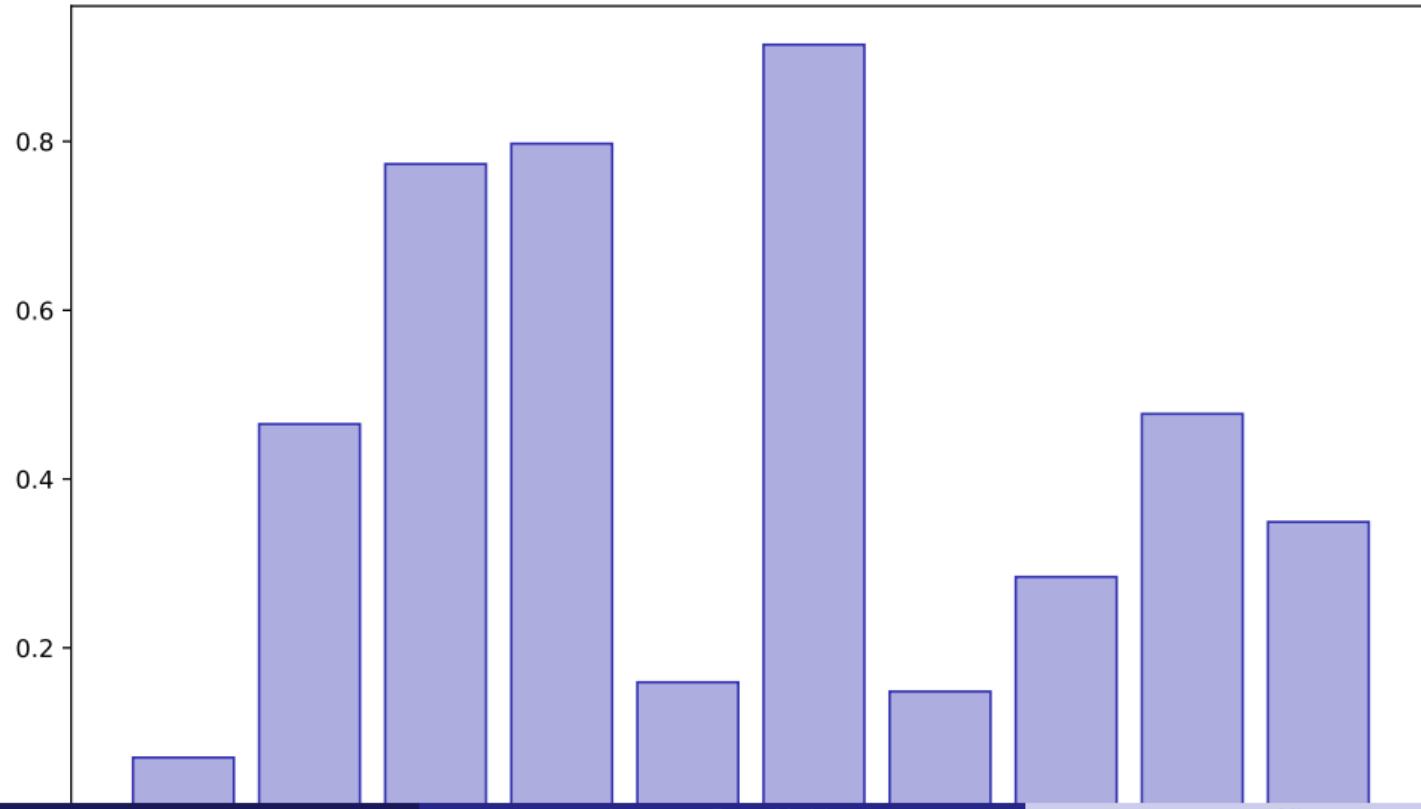
Pipeline Sklearn



Model Persistence



Portfolio Factors



Lesson Summary

Key Takeaways:

- Build multi-factor regression
- Understand Fama-French factors
- Interpret factor loadings
- Create complete ML pipelines

Apply these skills in your final project

Lesson 25: Logistic Regression

Data Science with Python – BSc Course

45 Minutes

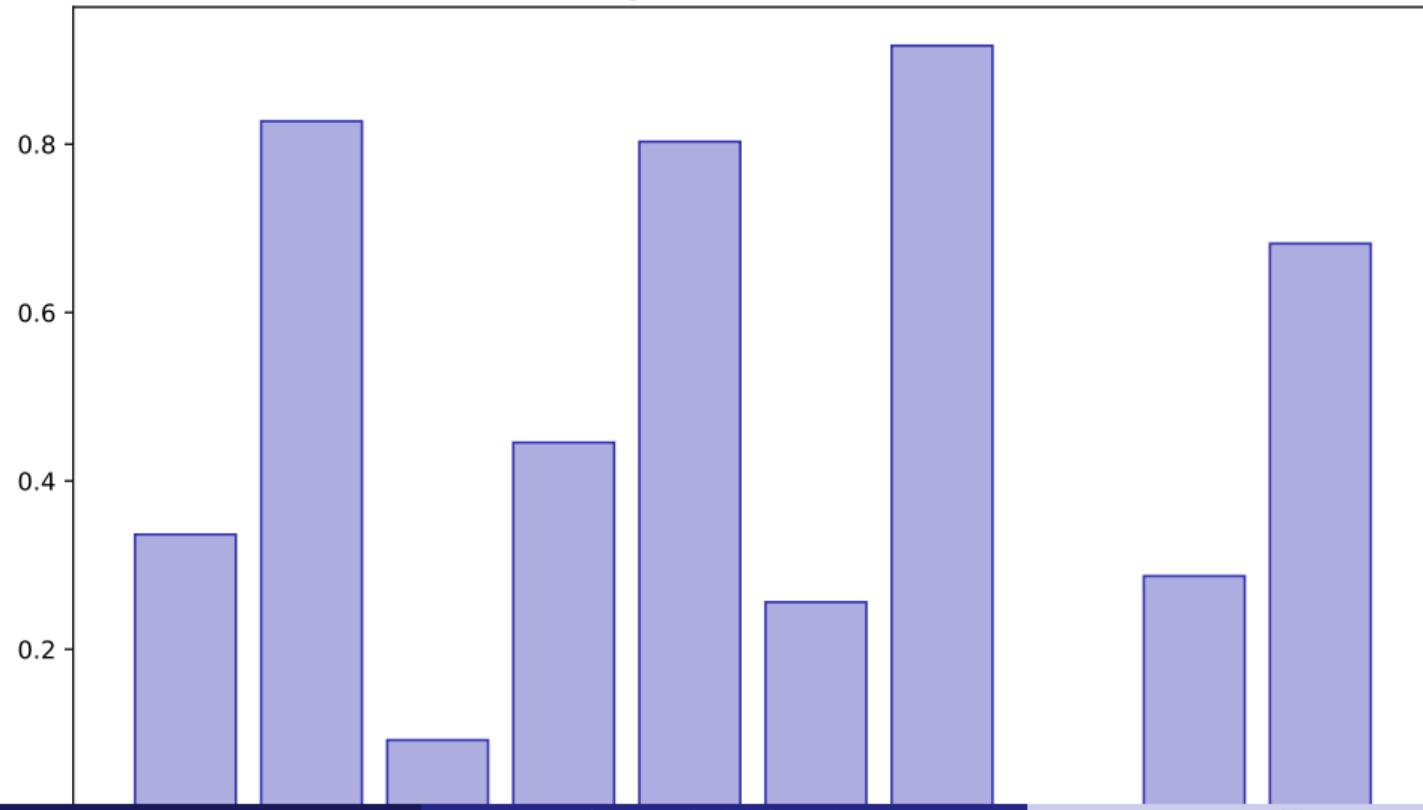
After this lesson, you will be able to:

- Understand sigmoid function
- Build binary classifiers
- Interpret odds ratios
- Predict market direction

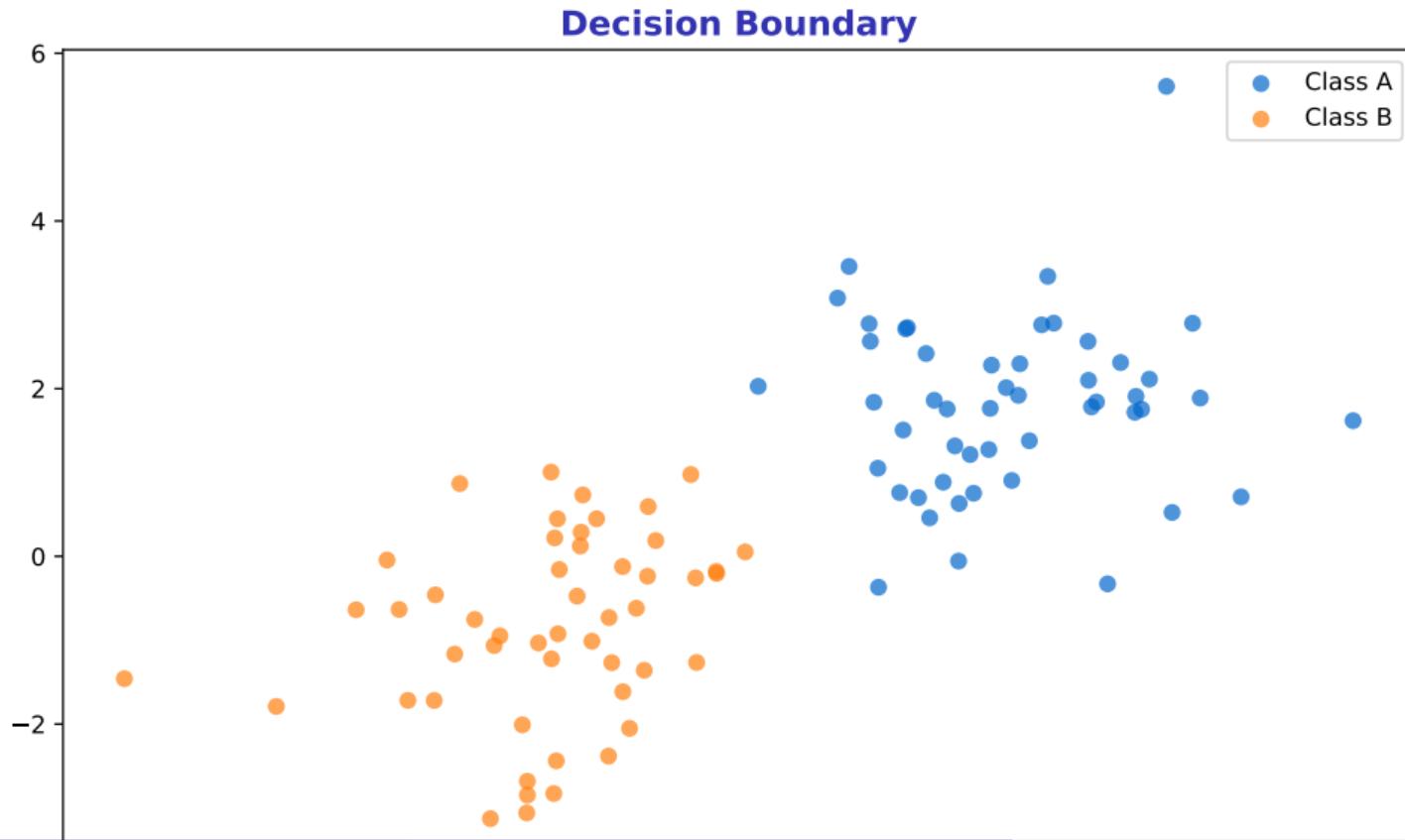
Building towards your final project

Sigmoid Function

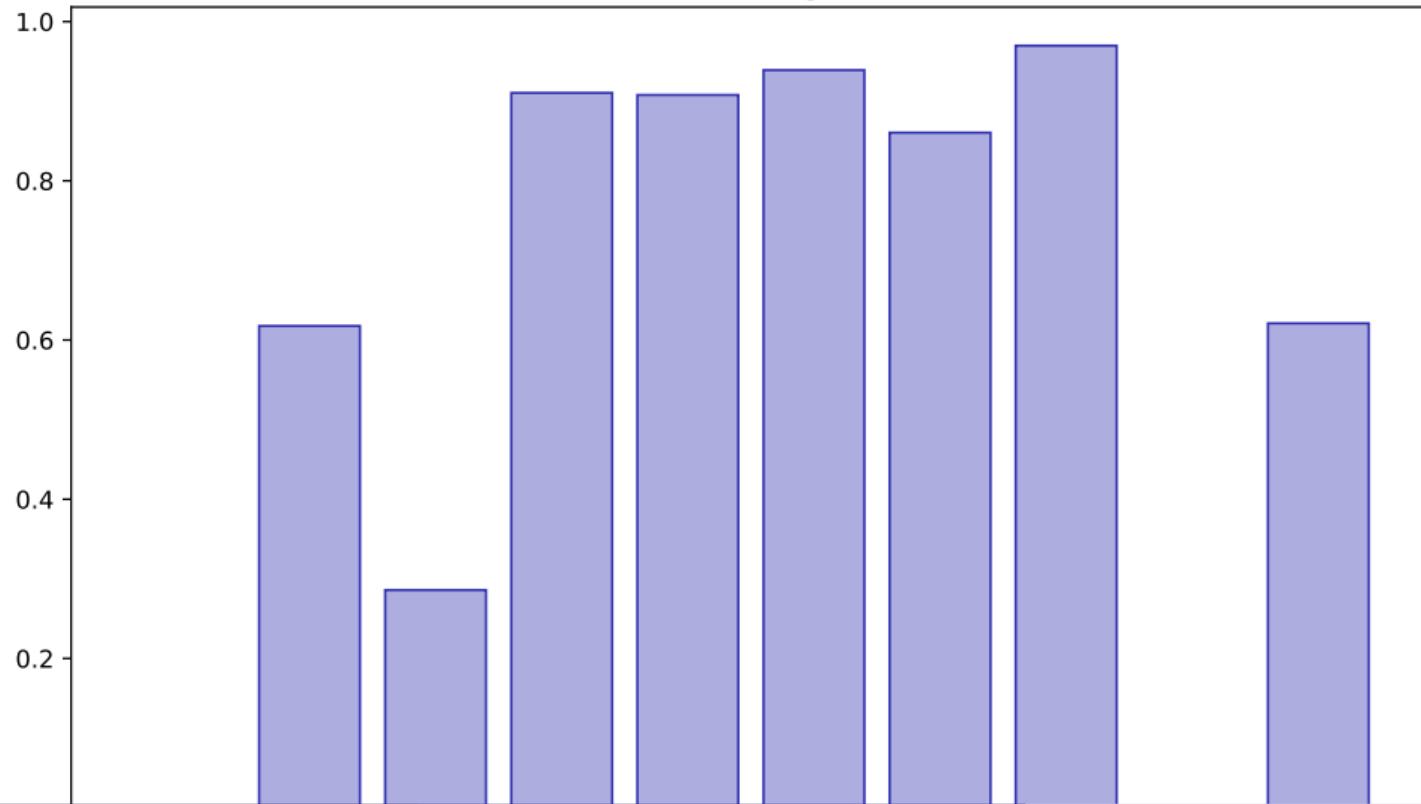
Sigmoid Function



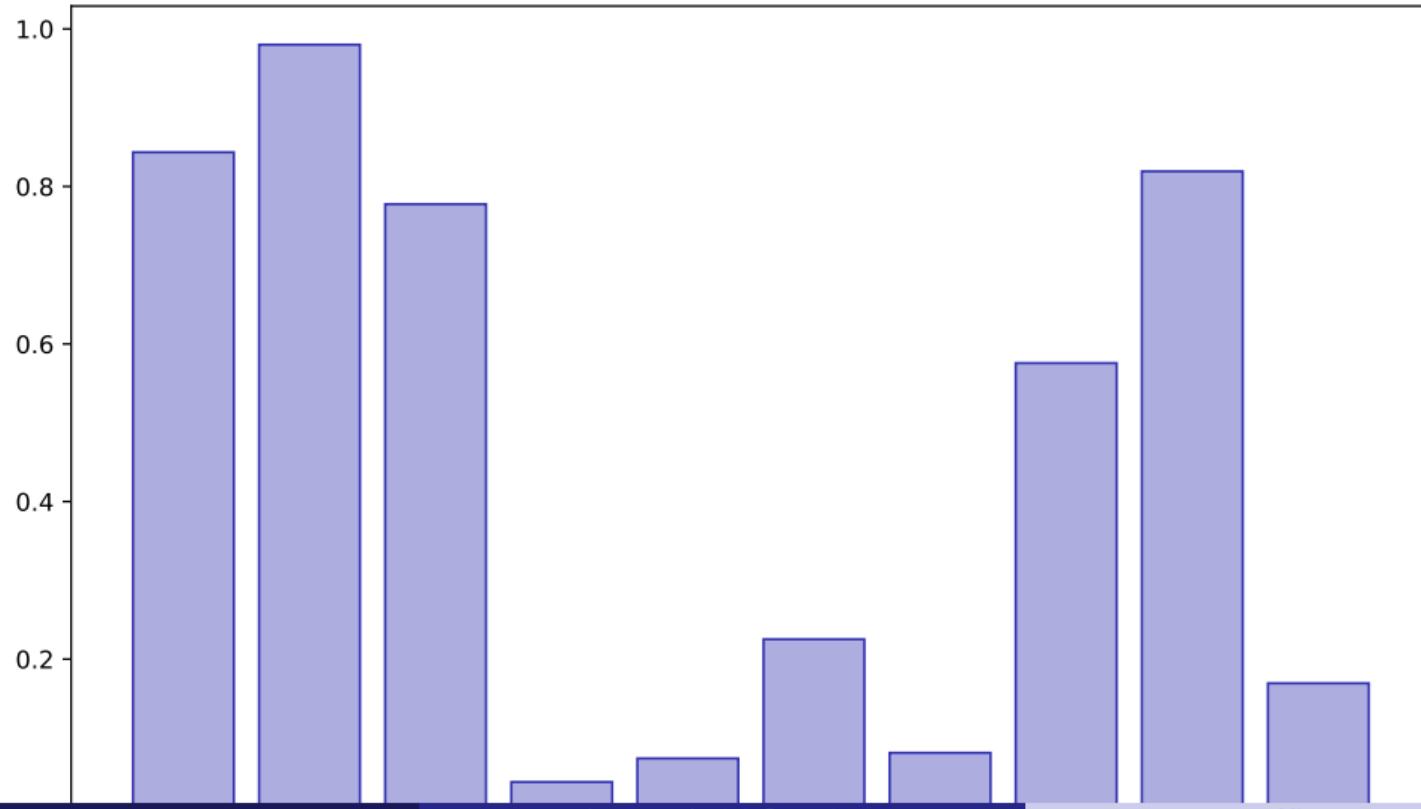
Decision Boundary



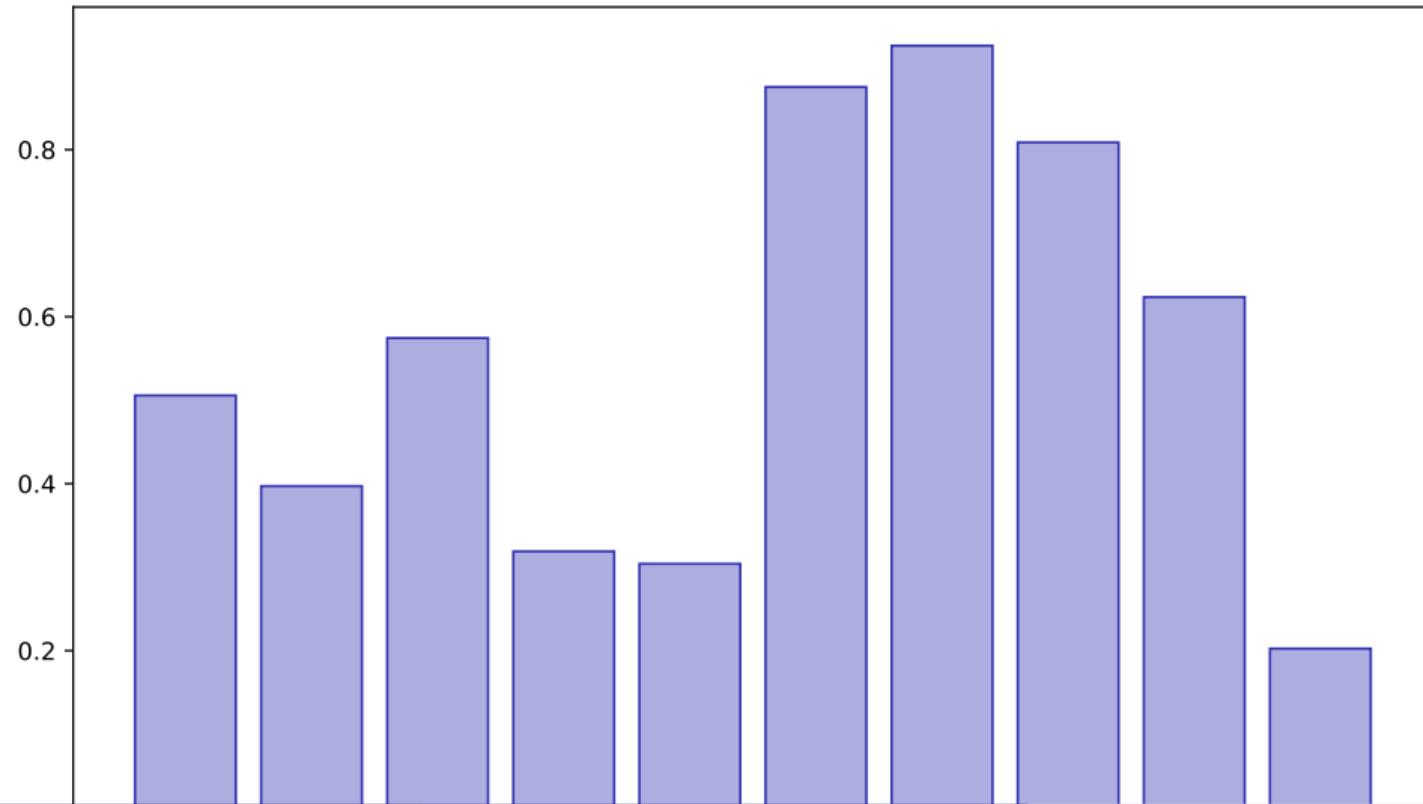
Sklearn Logistic



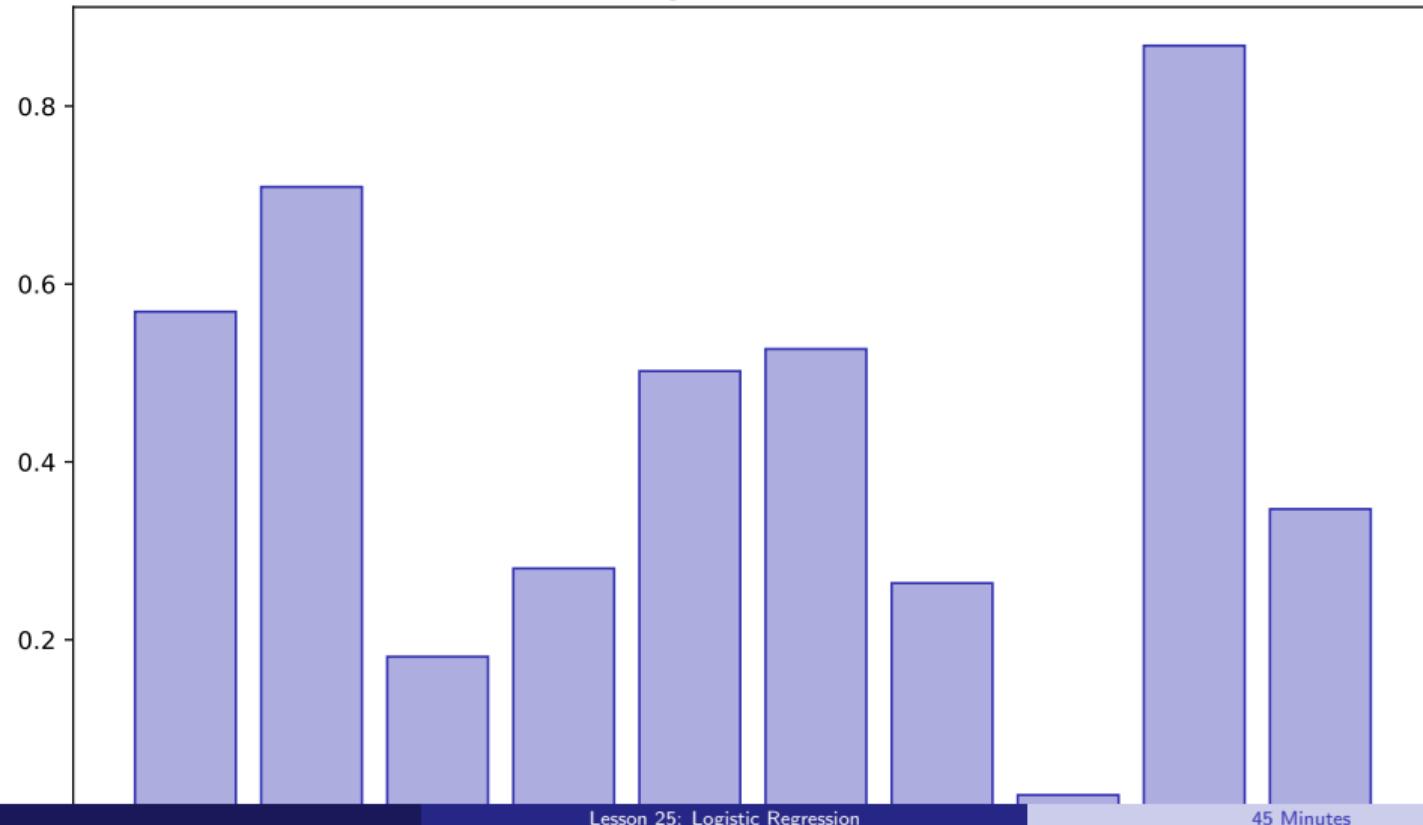
Odds Ratio



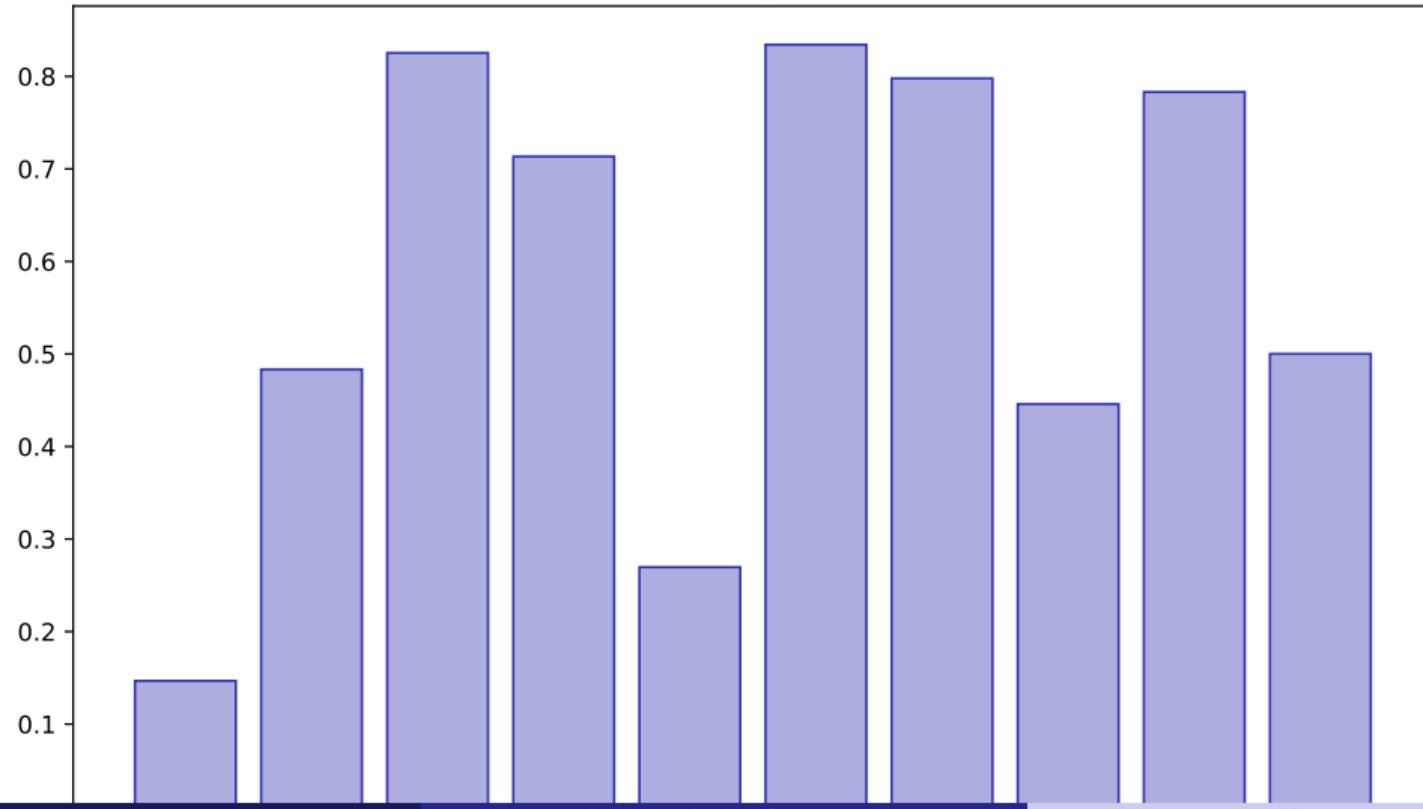
Multiclass



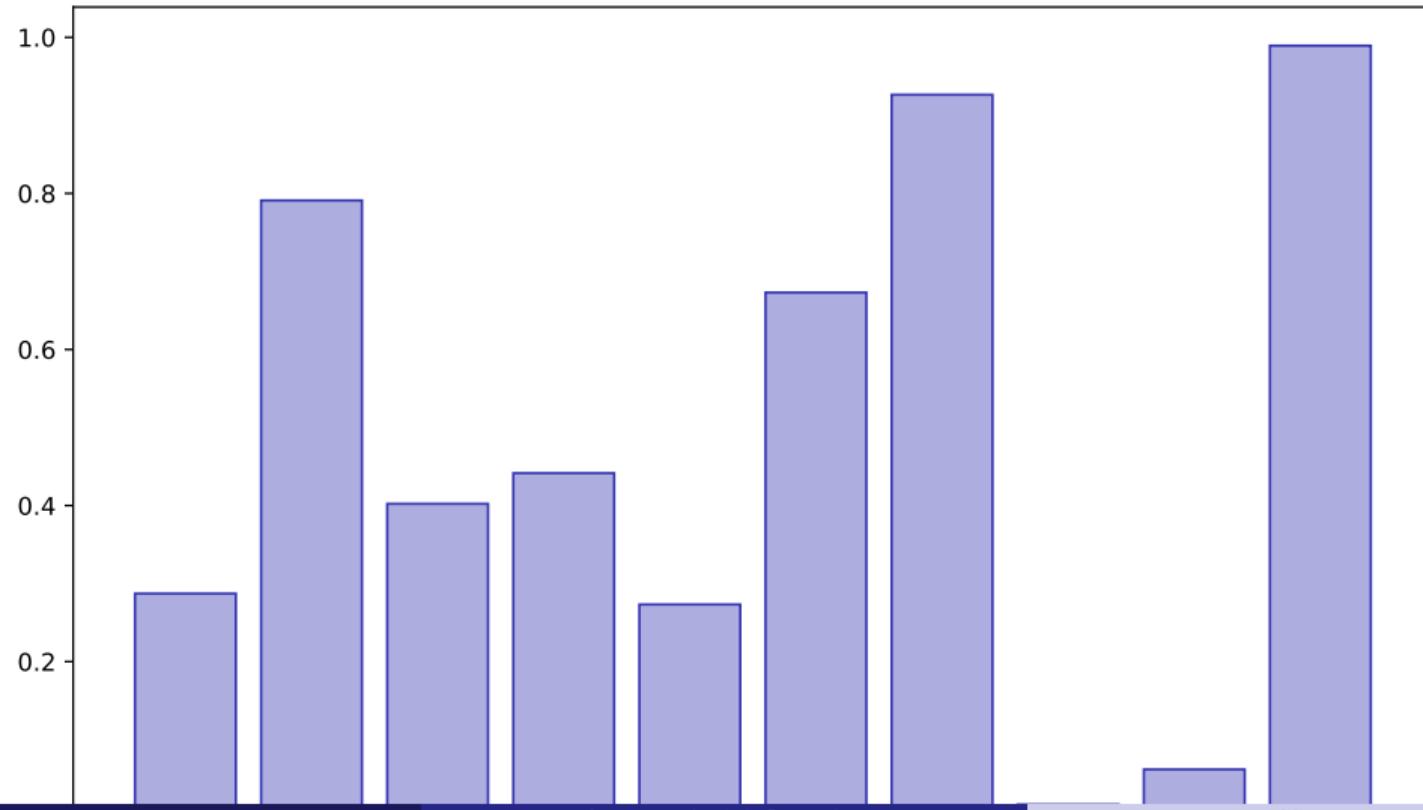
Regularization



Probability Output



Market Direction



Key Takeaways:

- Understand sigmoid function
- Build binary classifiers
- Interpret odds ratios
- Predict market direction

Apply these skills in your final project

Lesson 26: Decision Trees

Data Science with Python – BSc Course

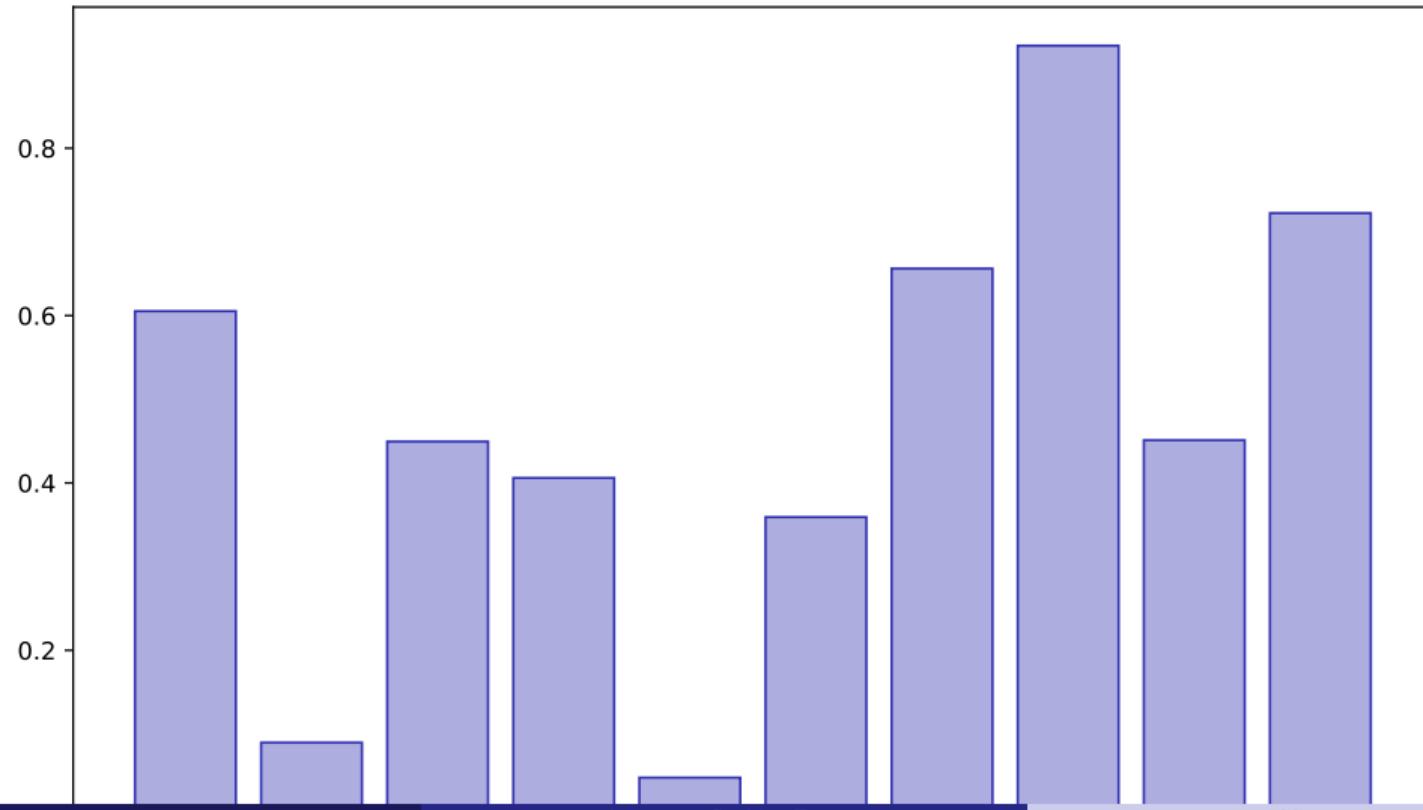
45 Minutes

After this lesson, you will be able to:

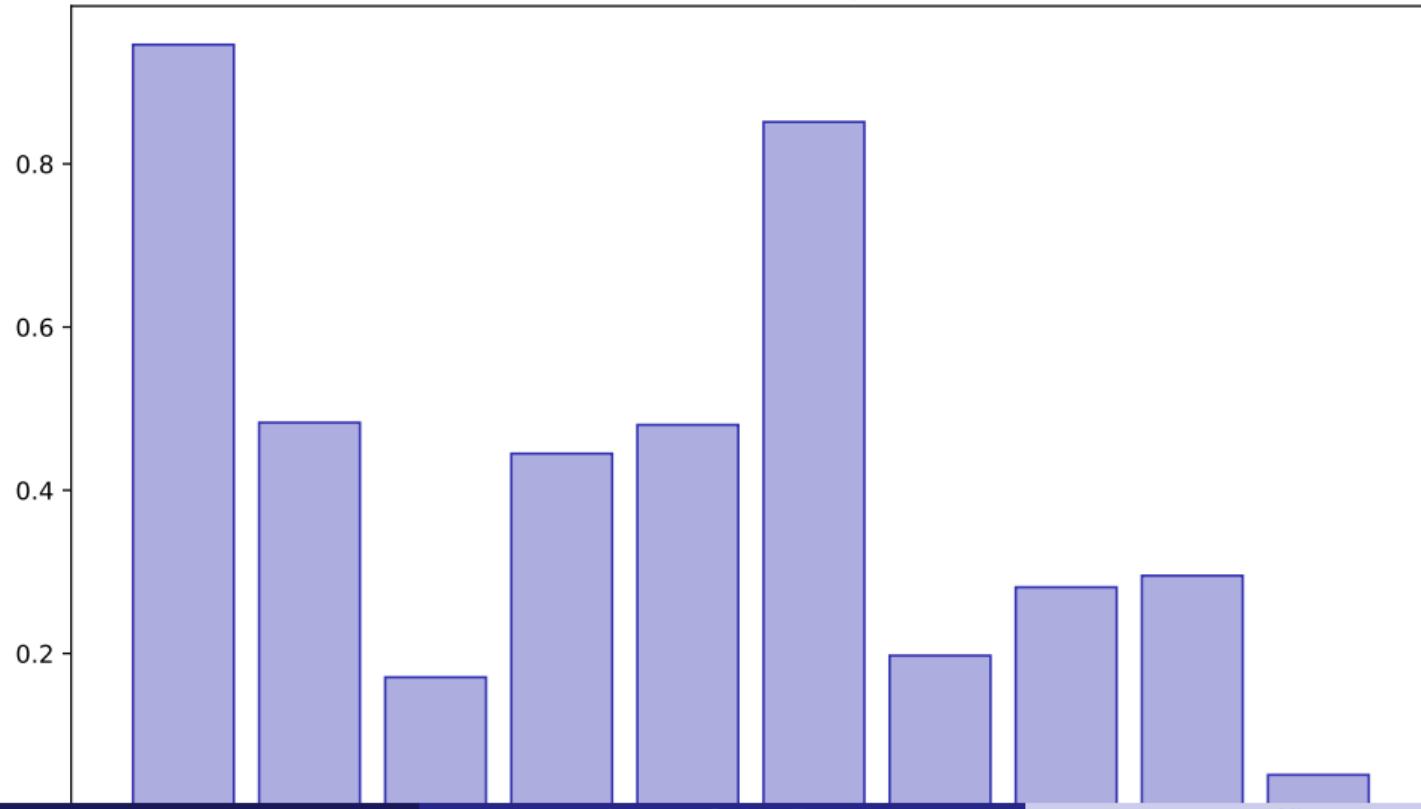
- Build decision tree classifiers
- Understand splitting criteria
- Apply Random Forest ensemble
- Interpret feature importance

Building towards your final project

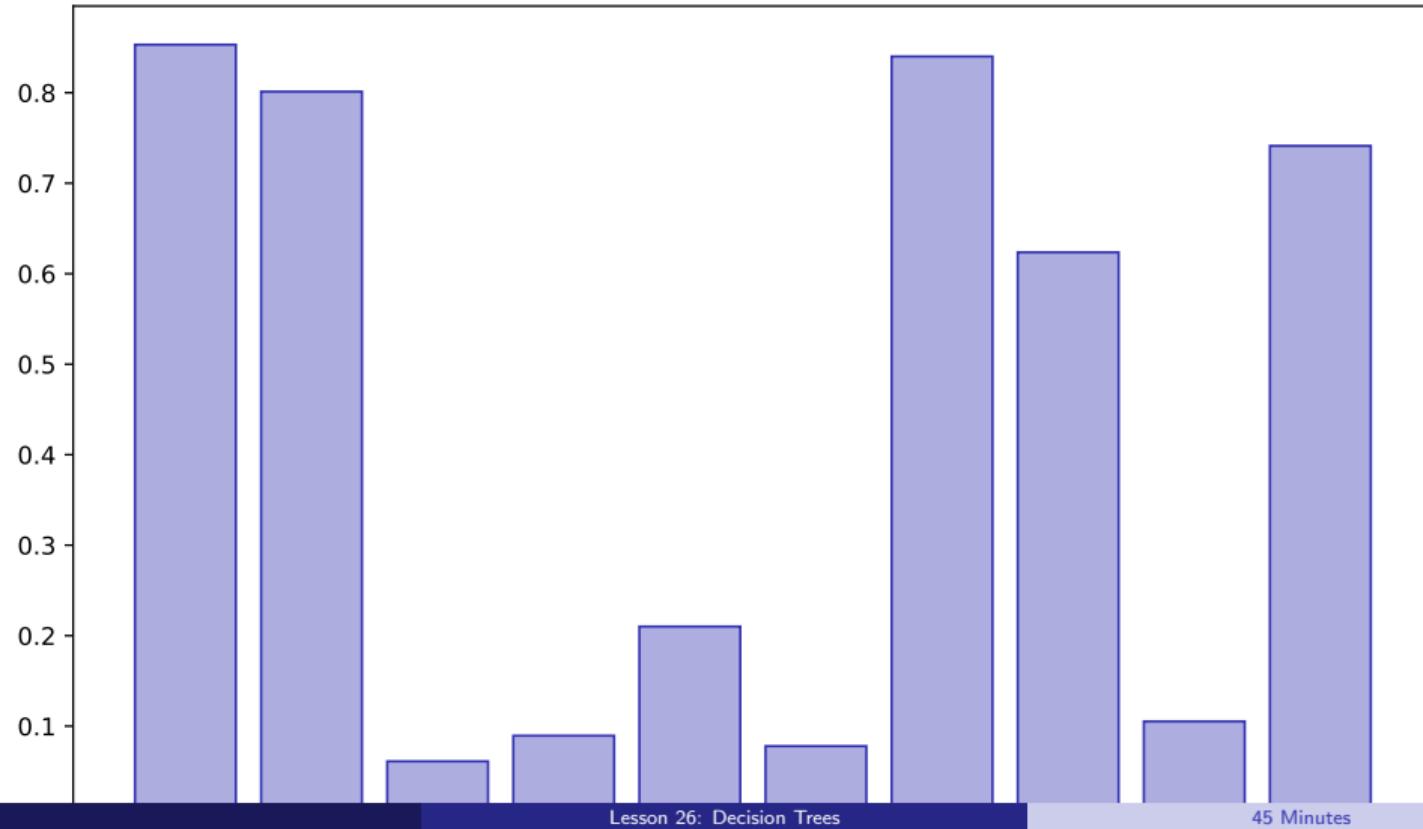
Tree Structure



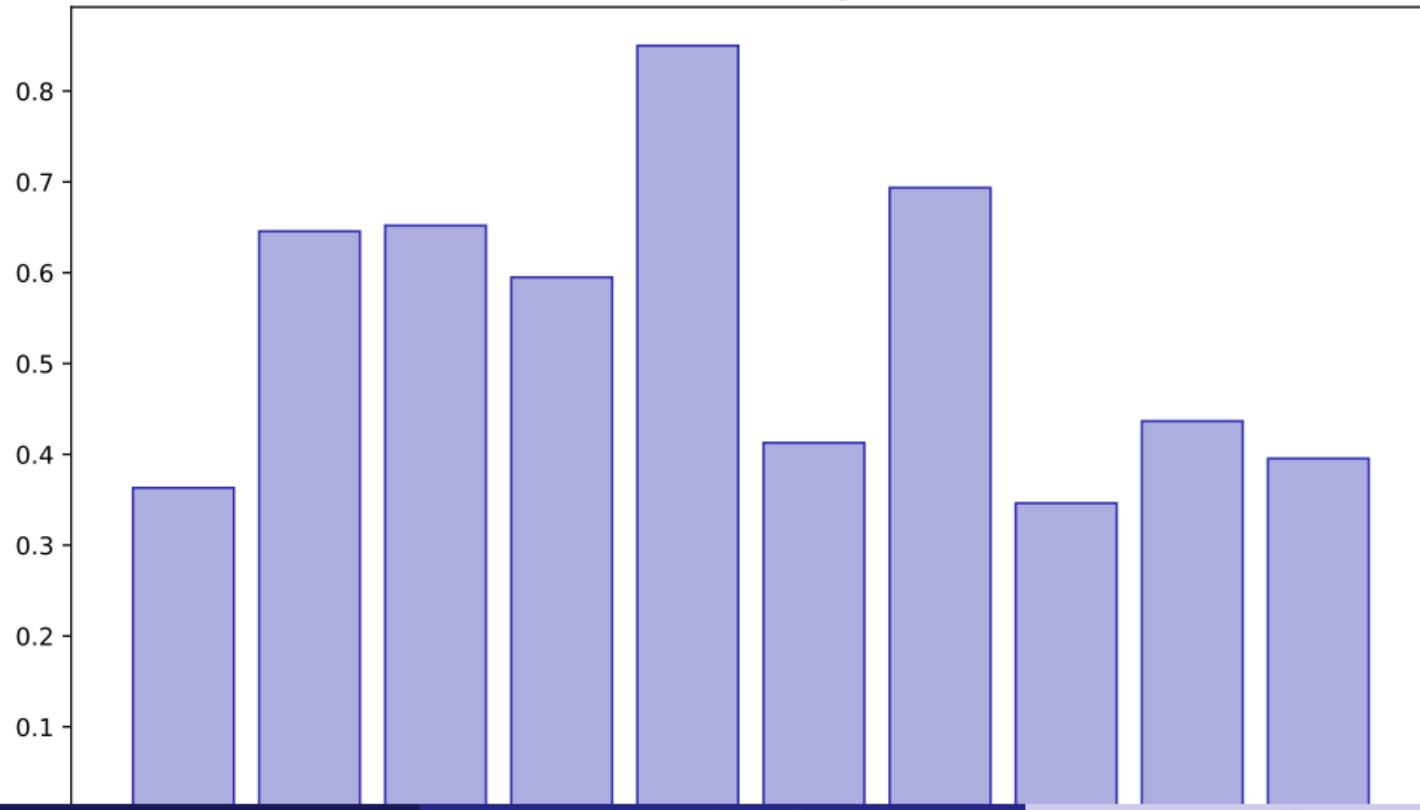
Gini Entropy



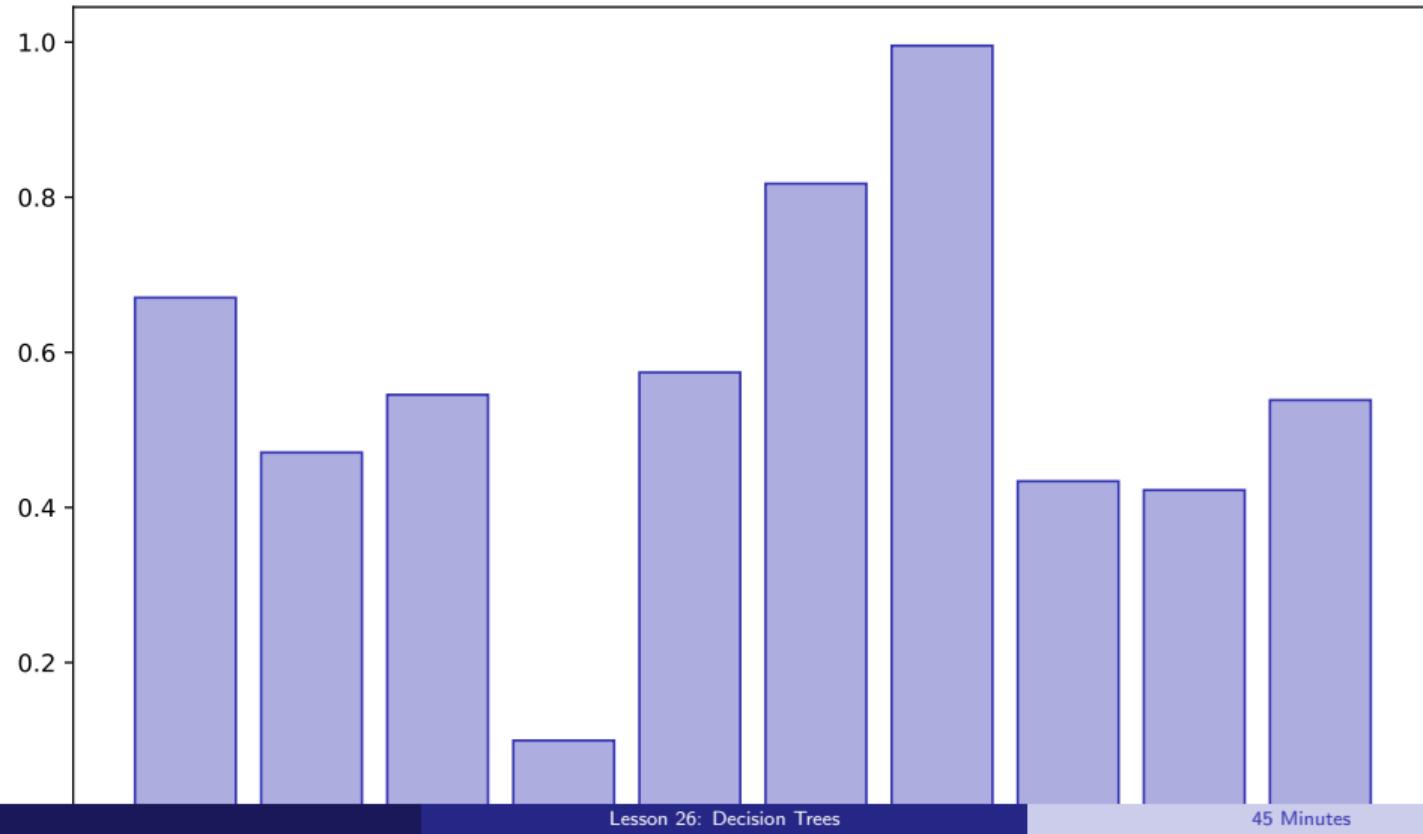
Tree Sklearn



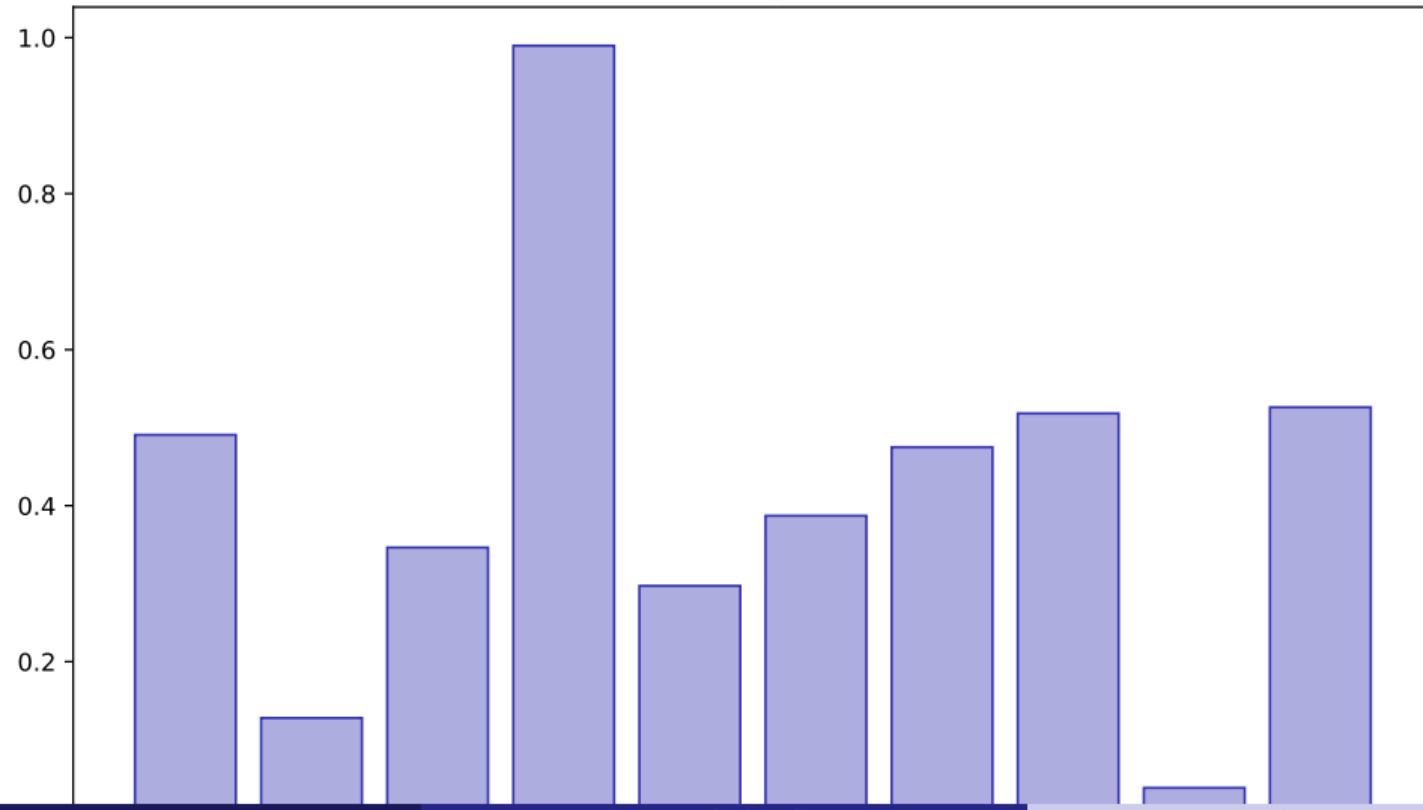
Overfitting



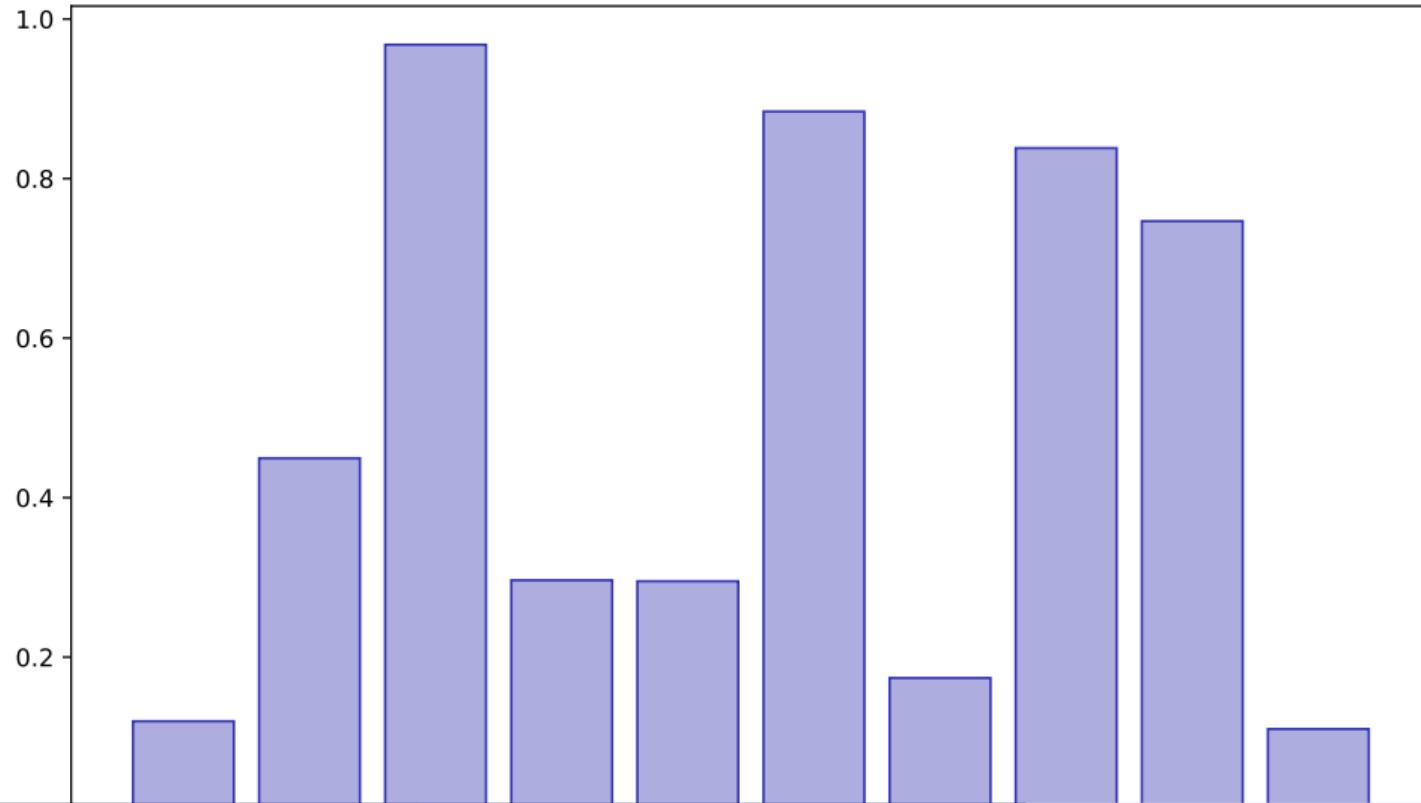
Random Forest



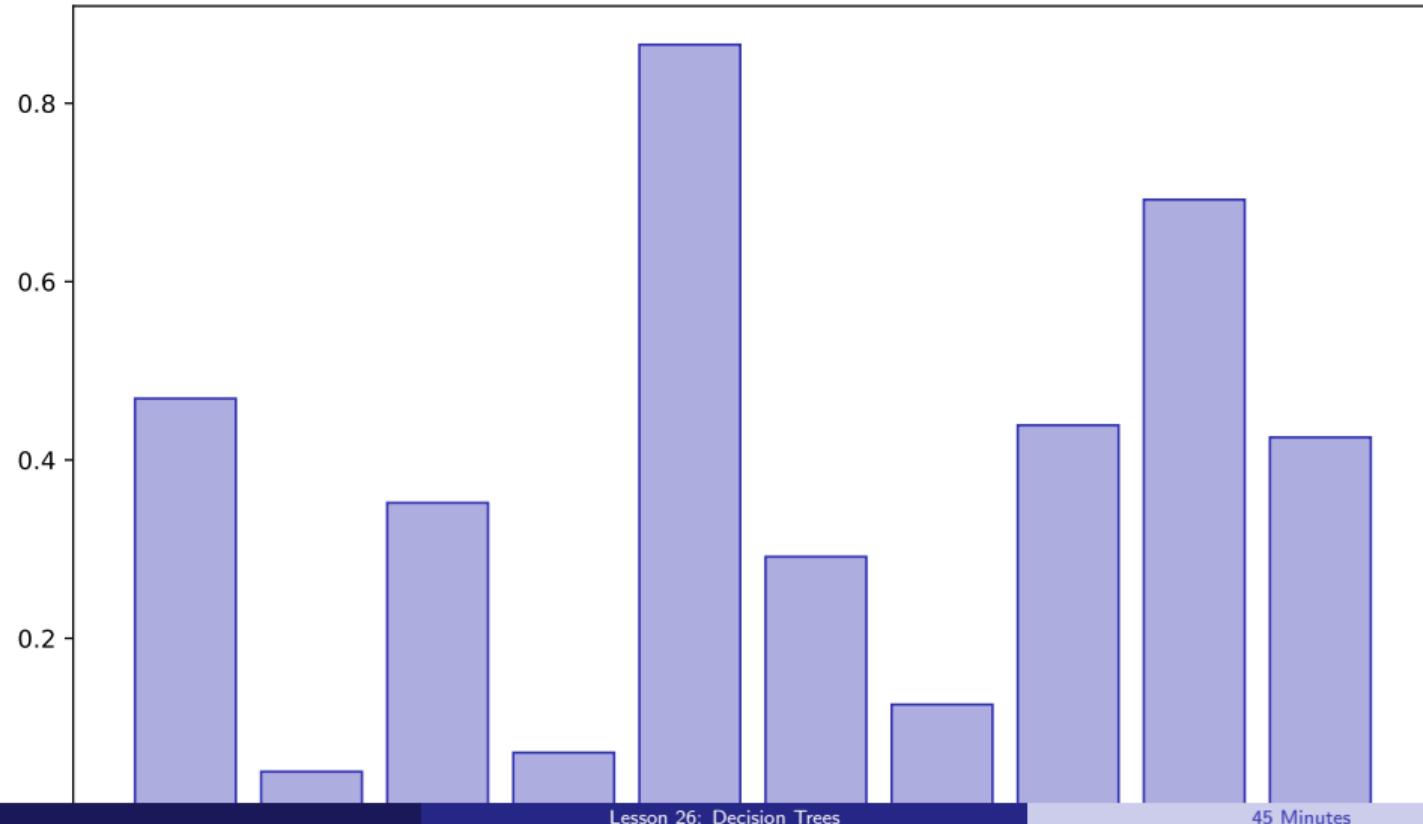
Feature Importance



Tree Visualization



Finance Trees



Lesson Summary

Key Takeaways:

- Build decision tree classifiers
- Understand splitting criteria
- Apply Random Forest ensemble
- Interpret feature importance

Apply these skills in your final project

Lesson 27: Classification Metrics

Data Science with Python – BSc Course

45 Minutes

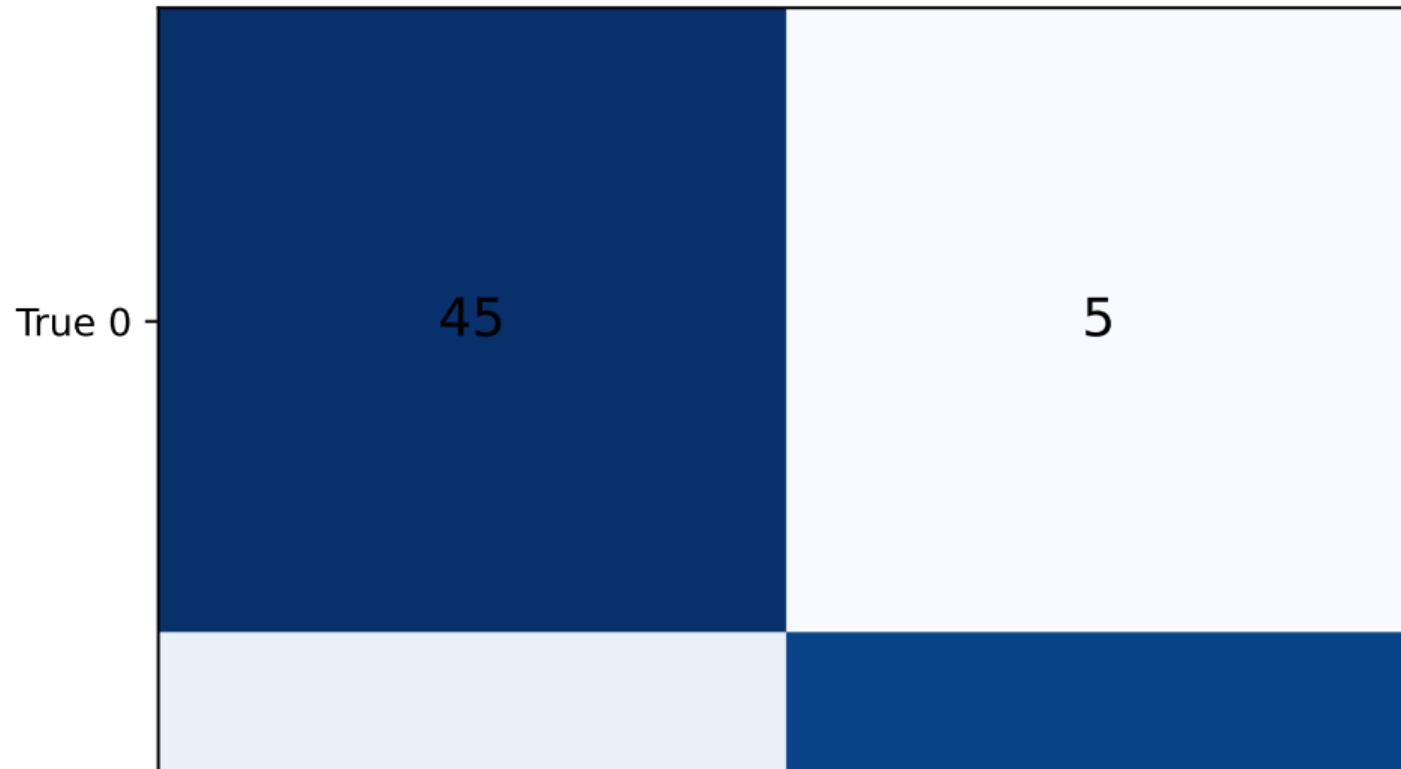
Learning Objectives

After this lesson, you will be able to:

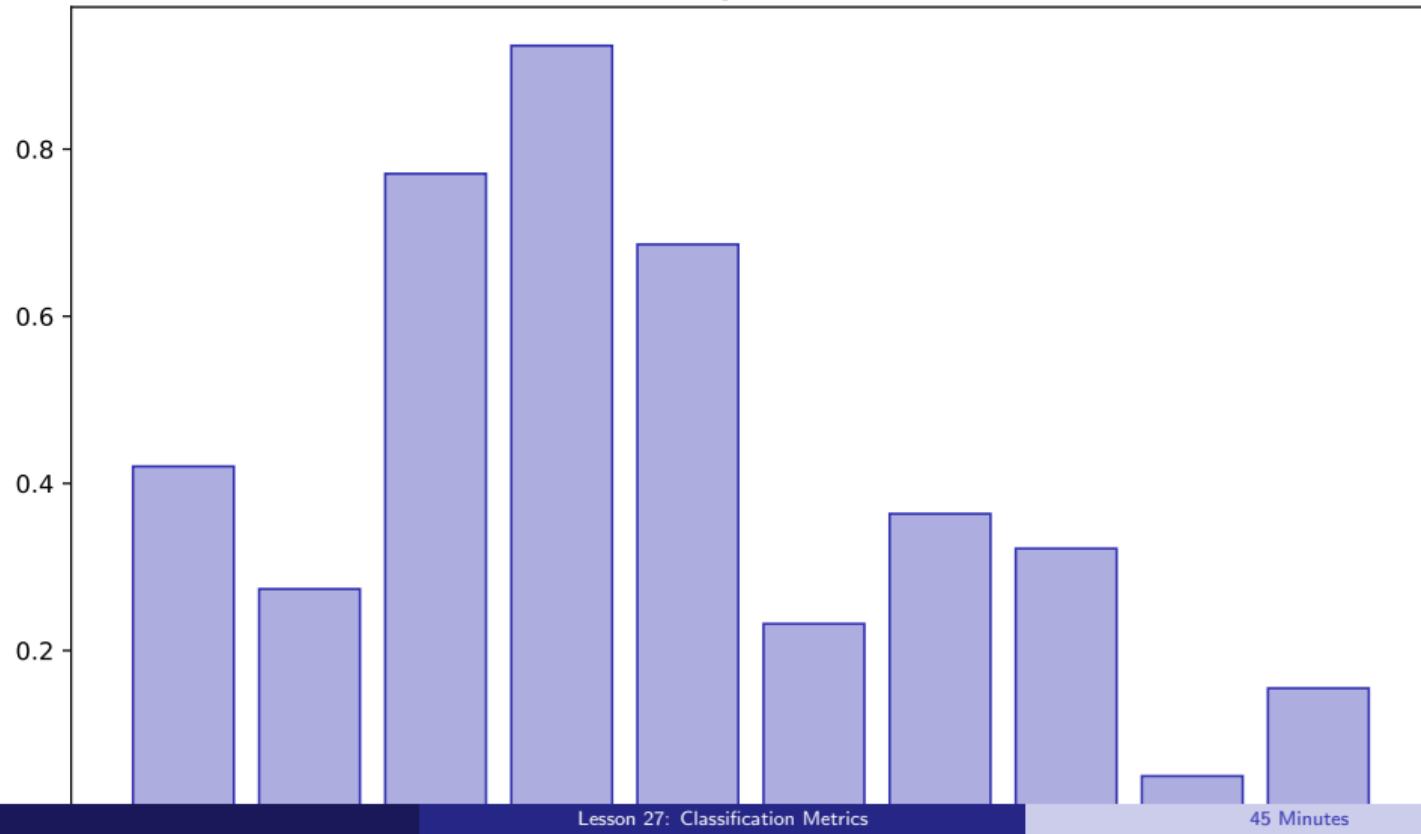
- Build confusion matrices
- Calculate precision and recall
- Plot ROC curves
- Handle class imbalance

Building towards your final project

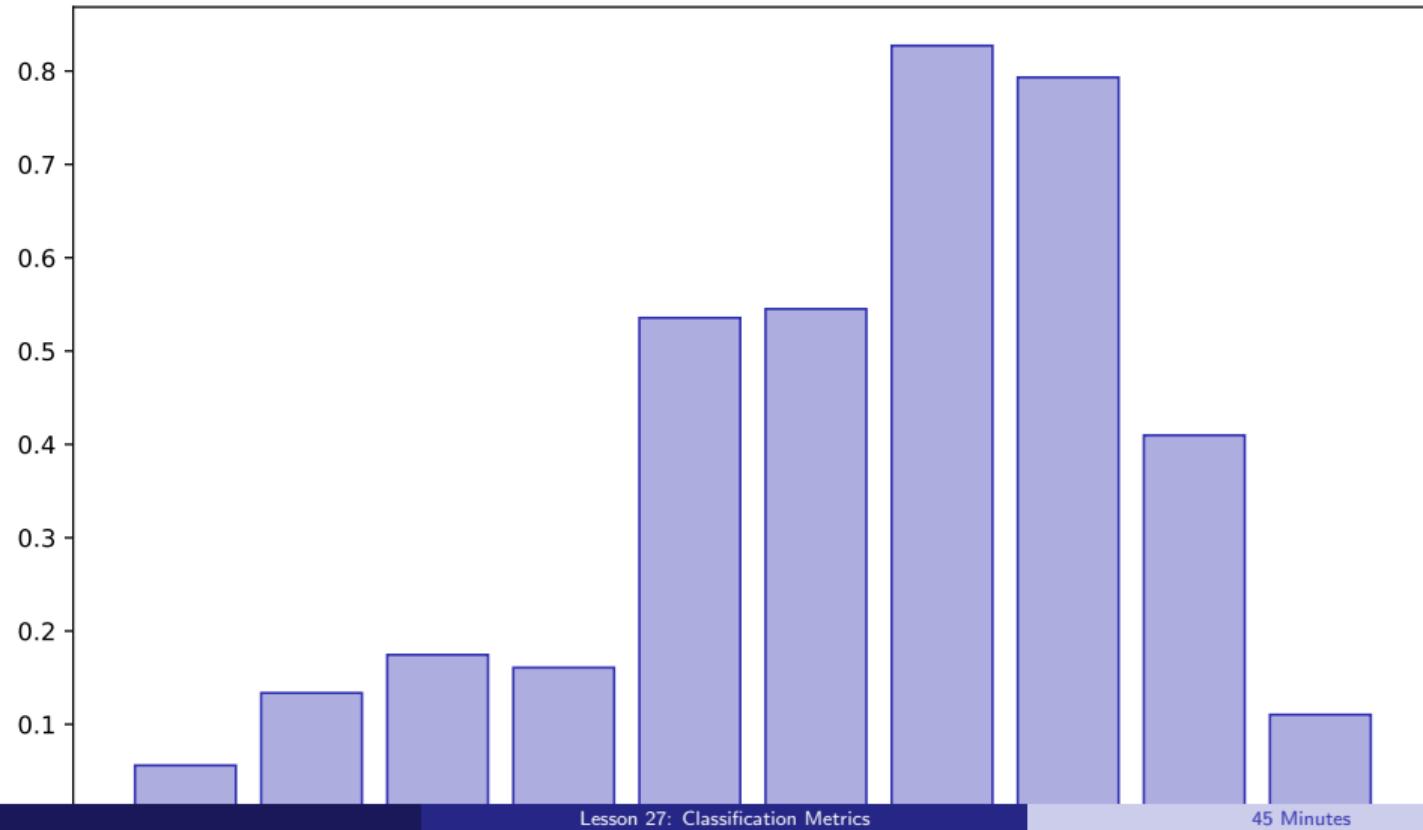
Confusion Matrix



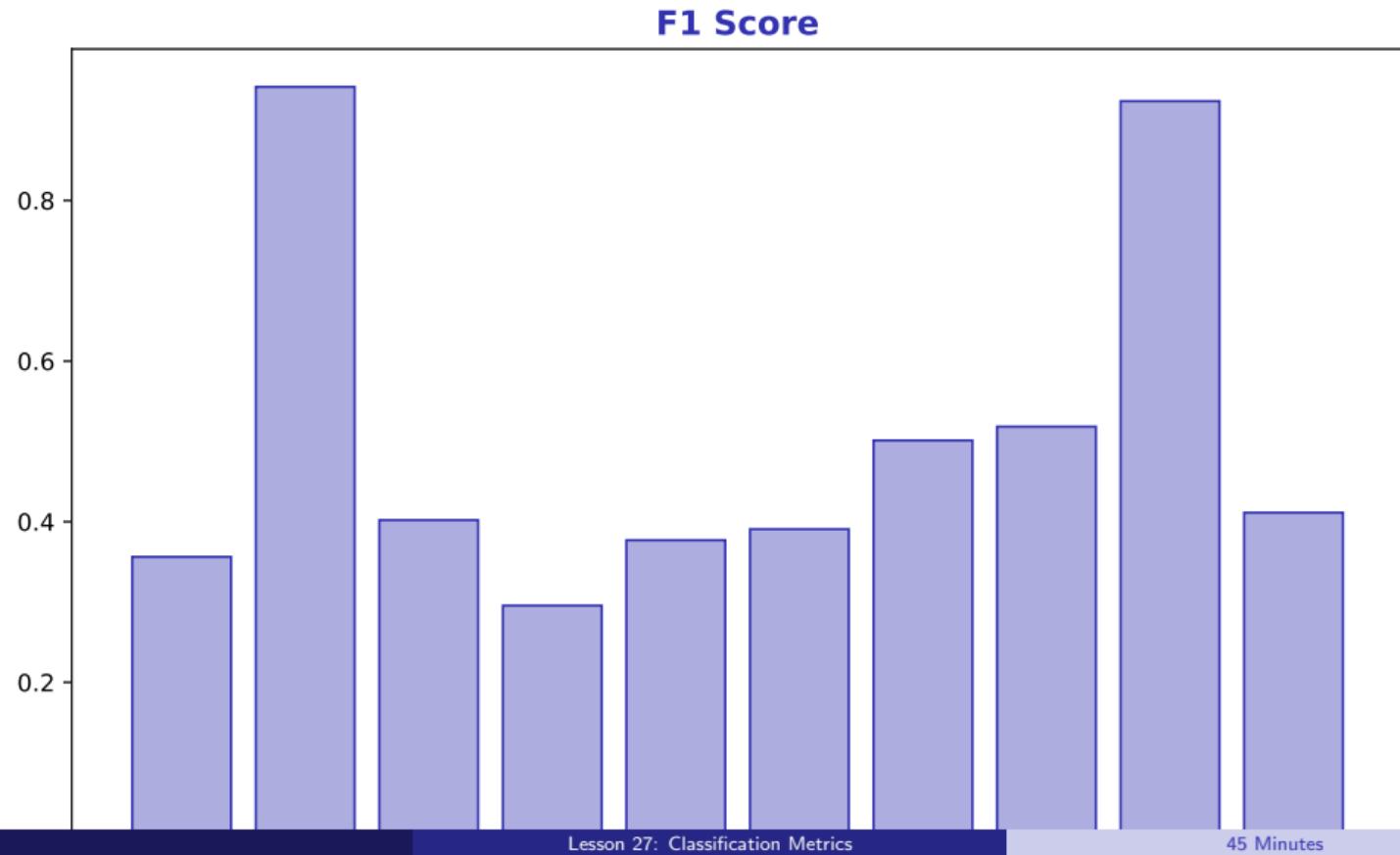
Accuracy Problems



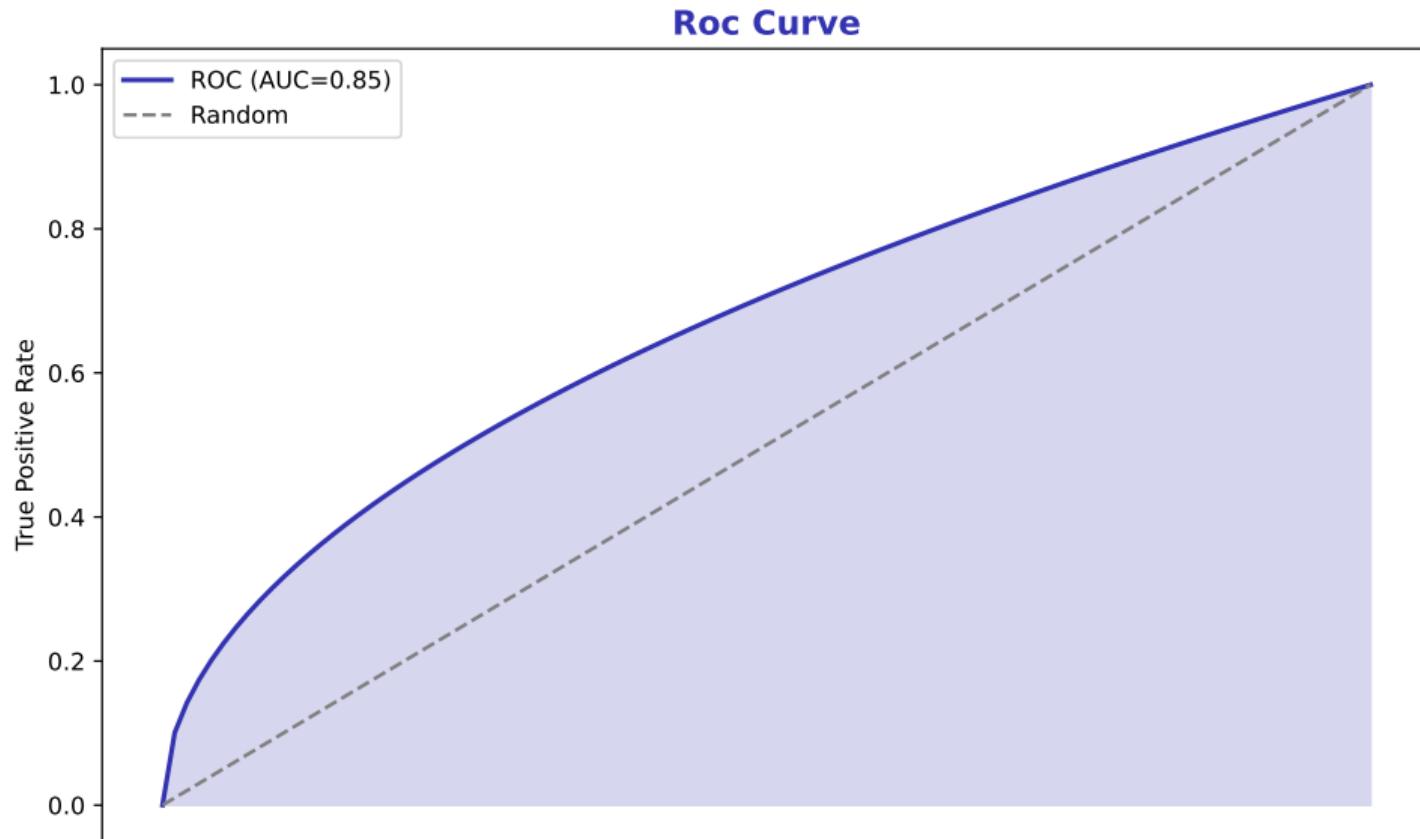
Precision Recall



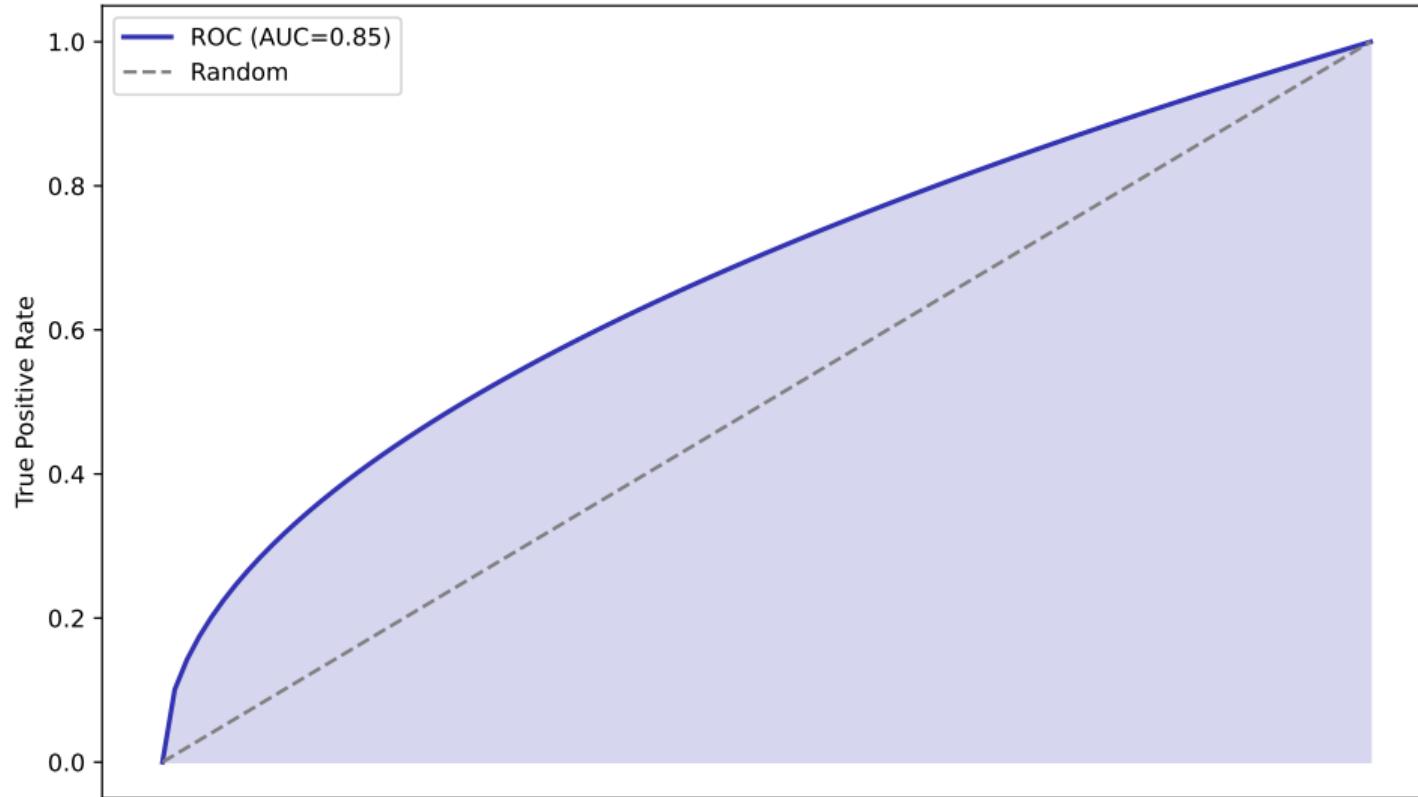
F1 Score



Roc Curve

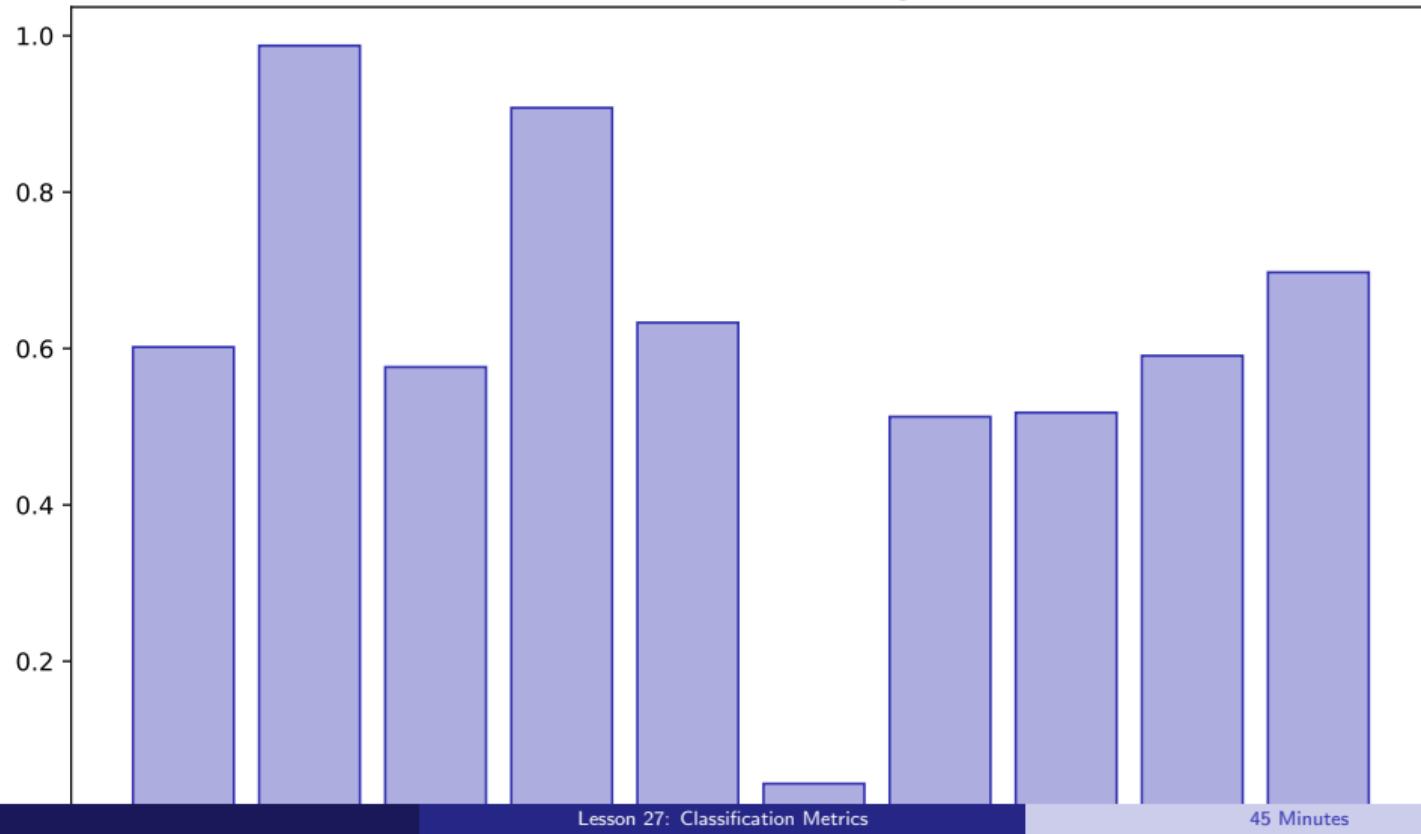


Auc Interpretation

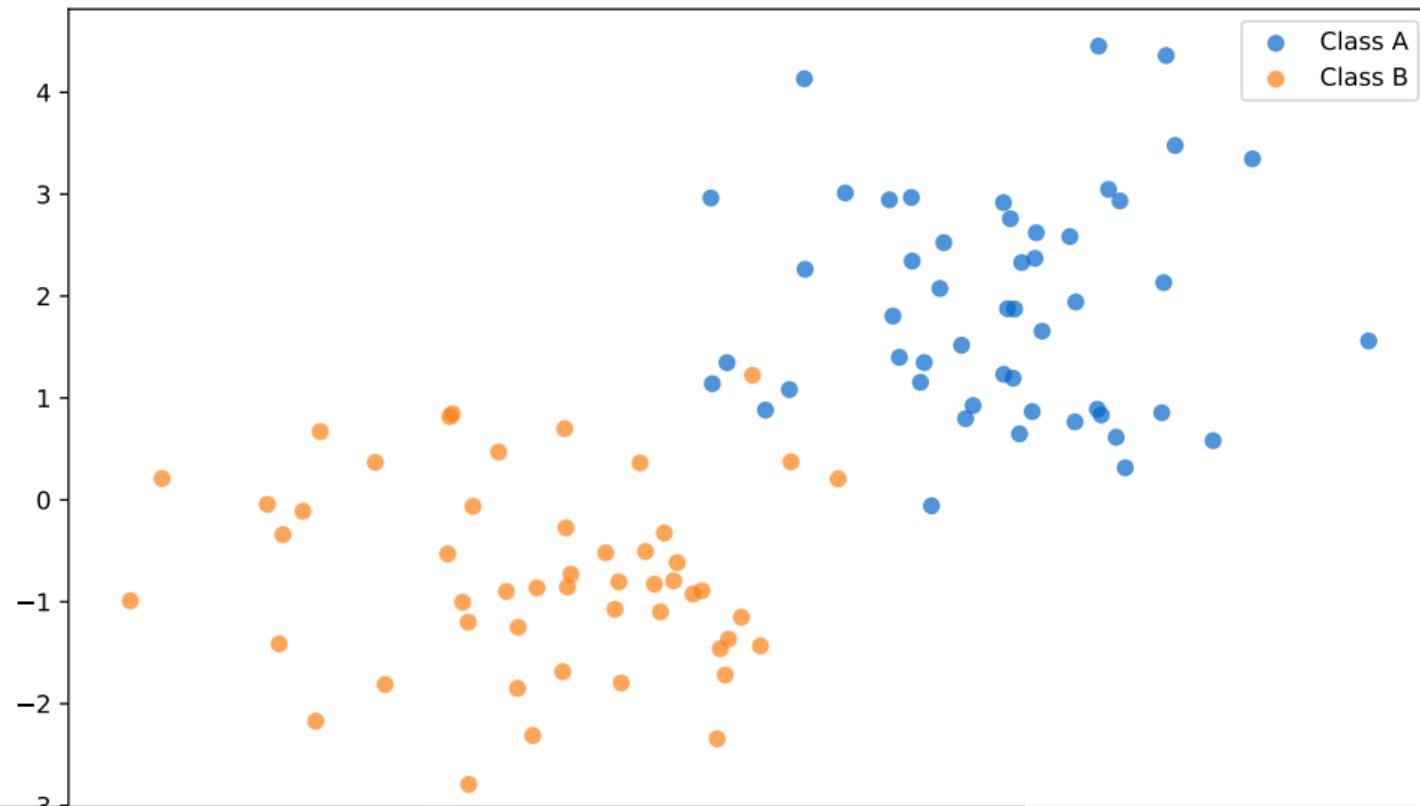


Threshold Tuning

Threshold Tuning



Finance Classification



Key Takeaways:

- Build confusion matrices
- Calculate precision and recall
- Plot ROC curves
- Handle class imbalance

Apply these skills in your final project

Lesson 28: Class Imbalance

Data Science with Python – BSc Course

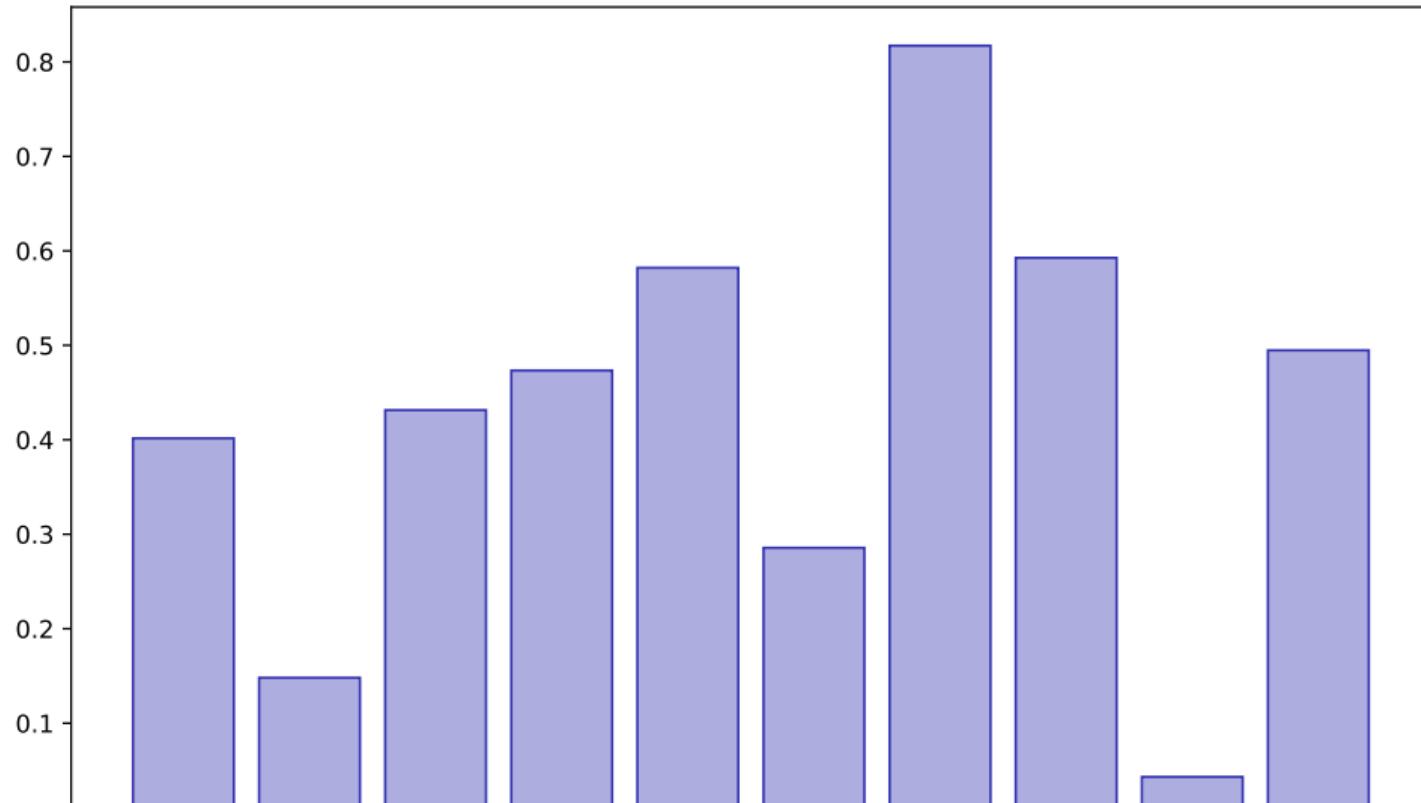
45 Minutes

After this lesson, you will be able to:

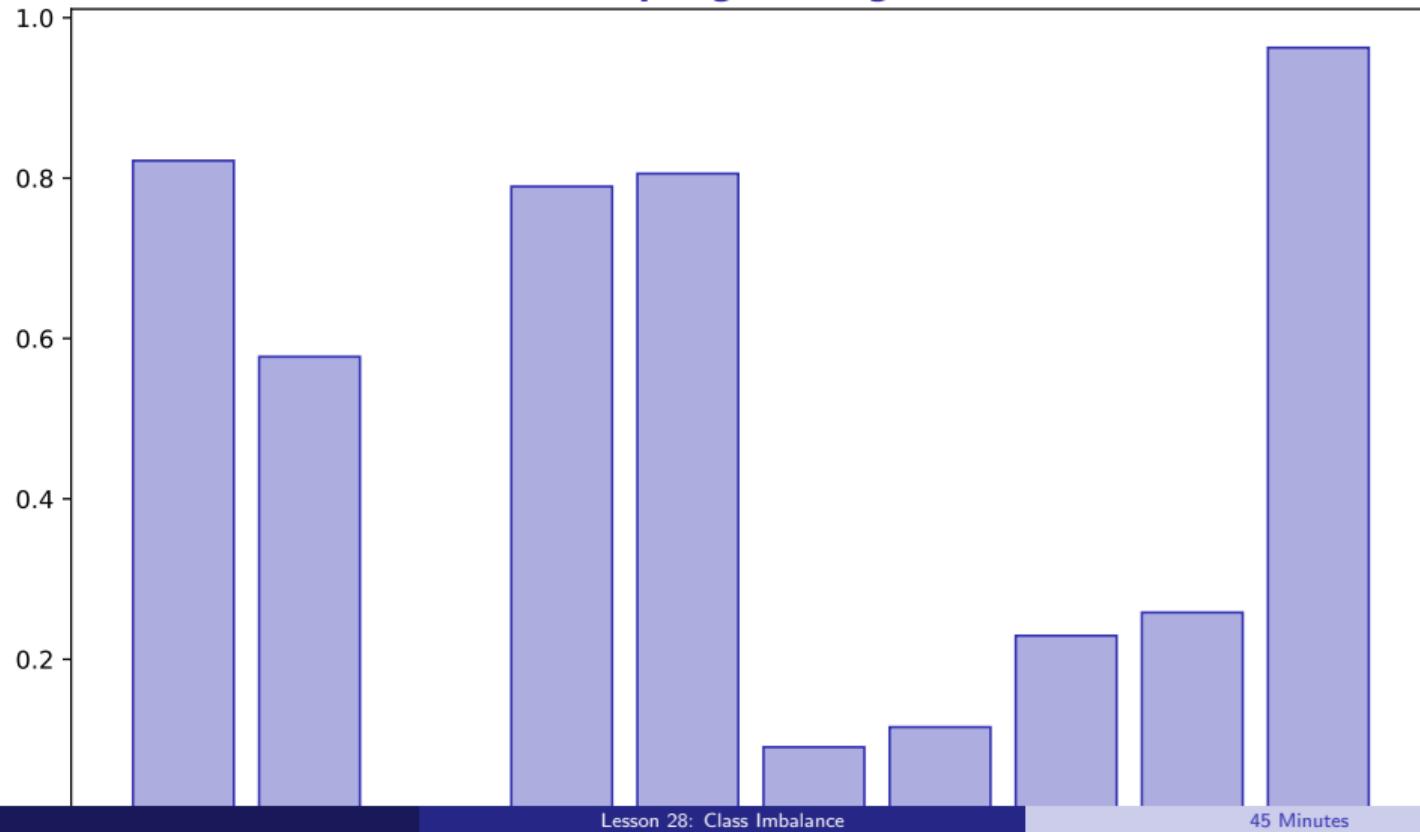
- Identify imbalanced datasets
- Apply SMOTE oversampling
- Use class weights
- Evaluate fairly

Building towards your final project

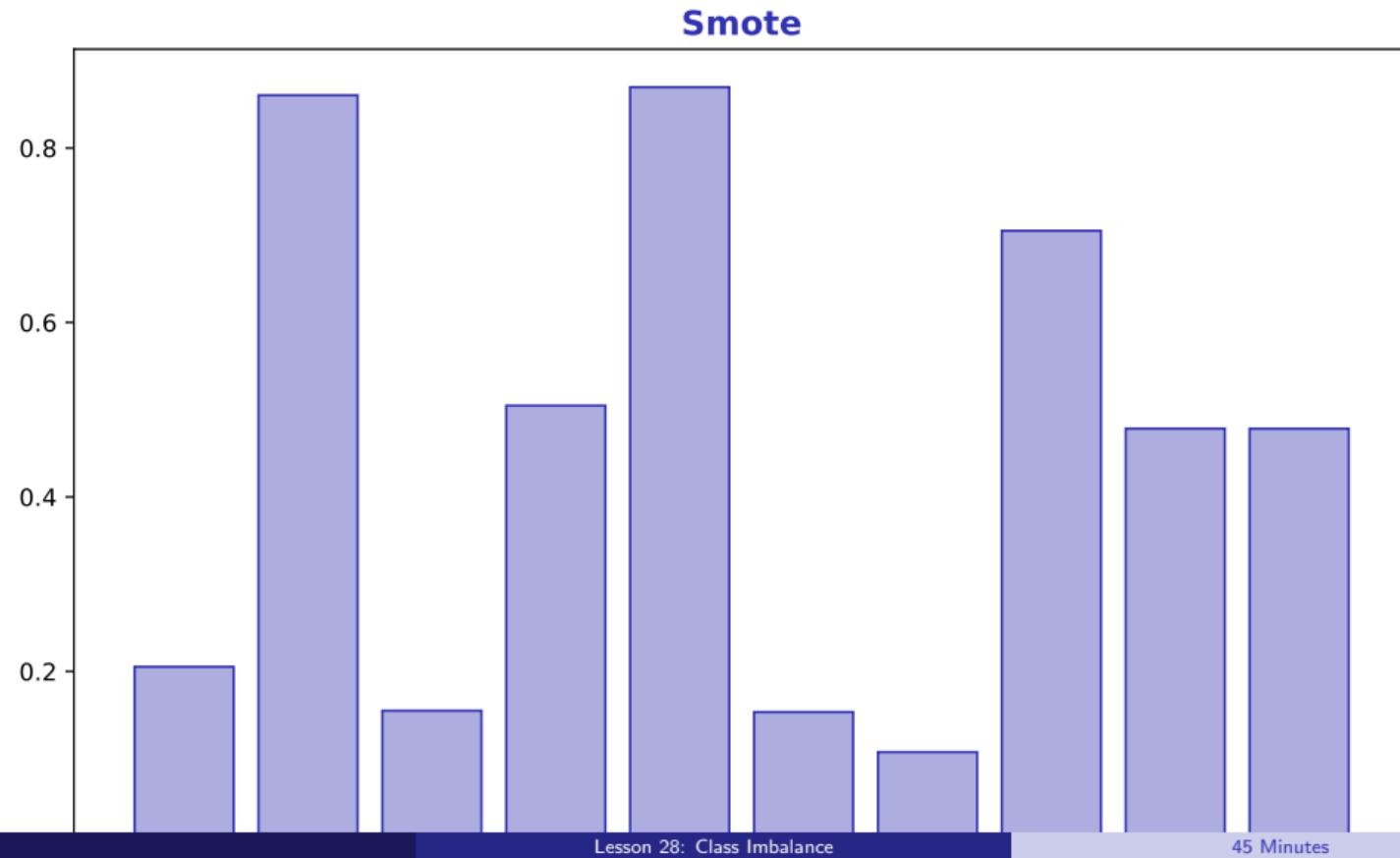
Imbalance Problem



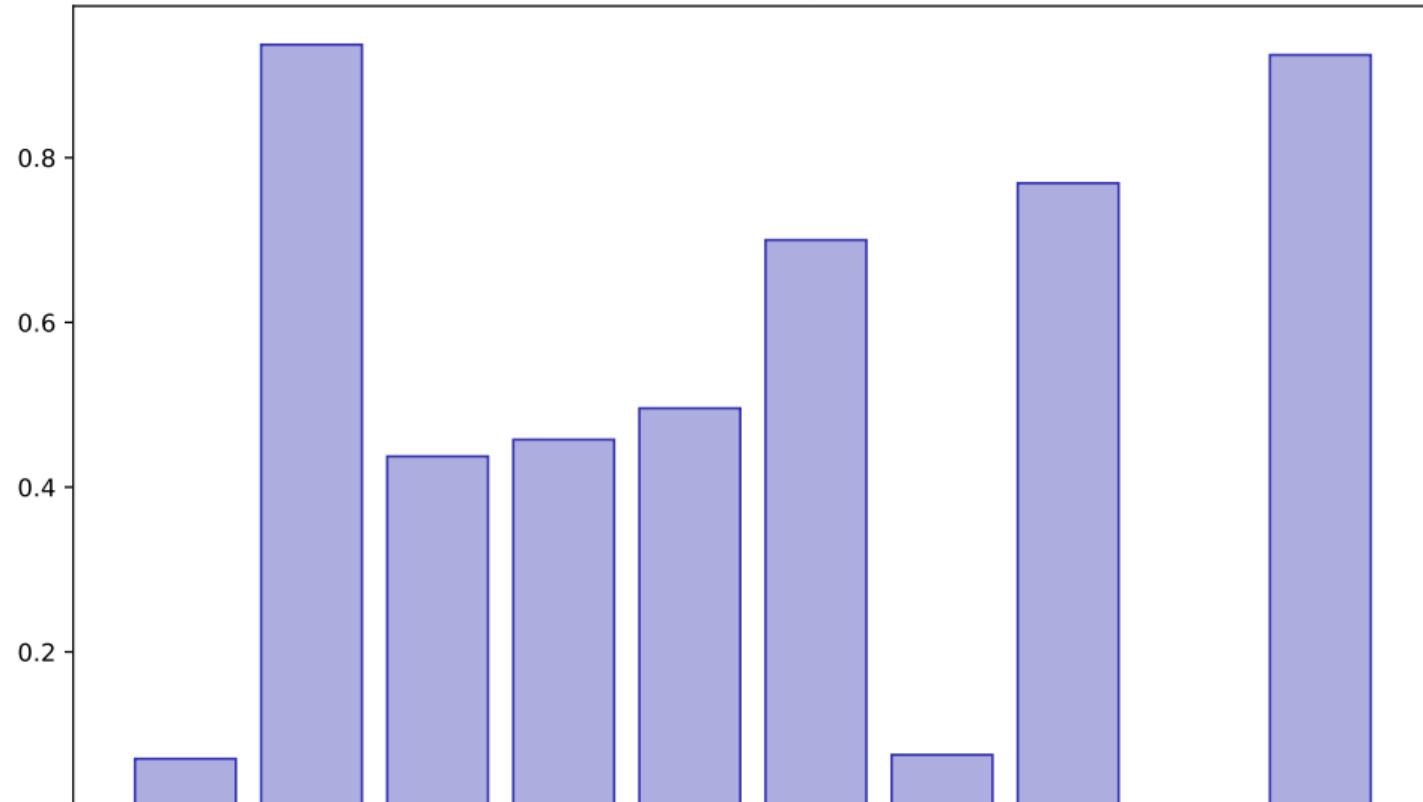
Sampling Strategies



Smote

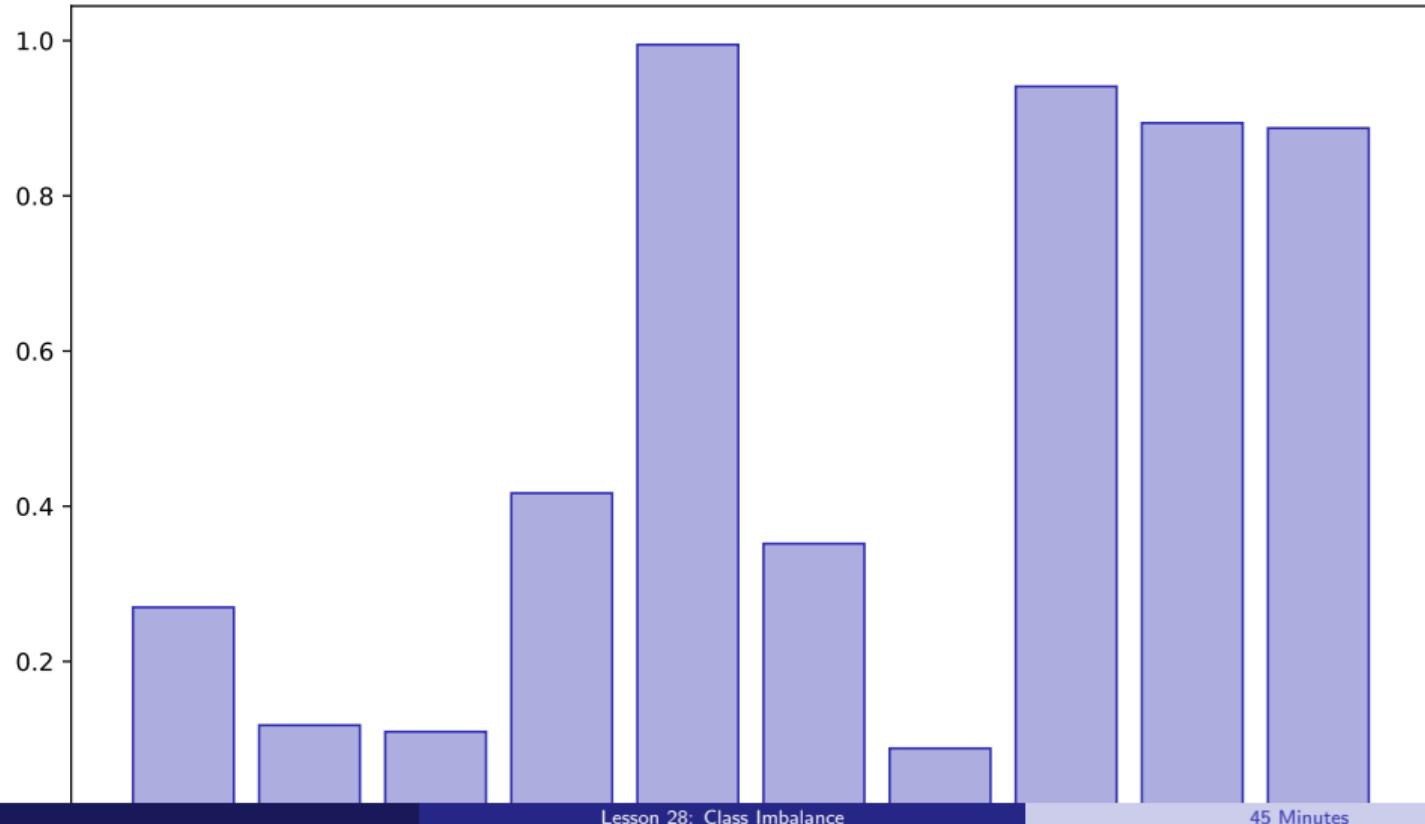


Class Weights

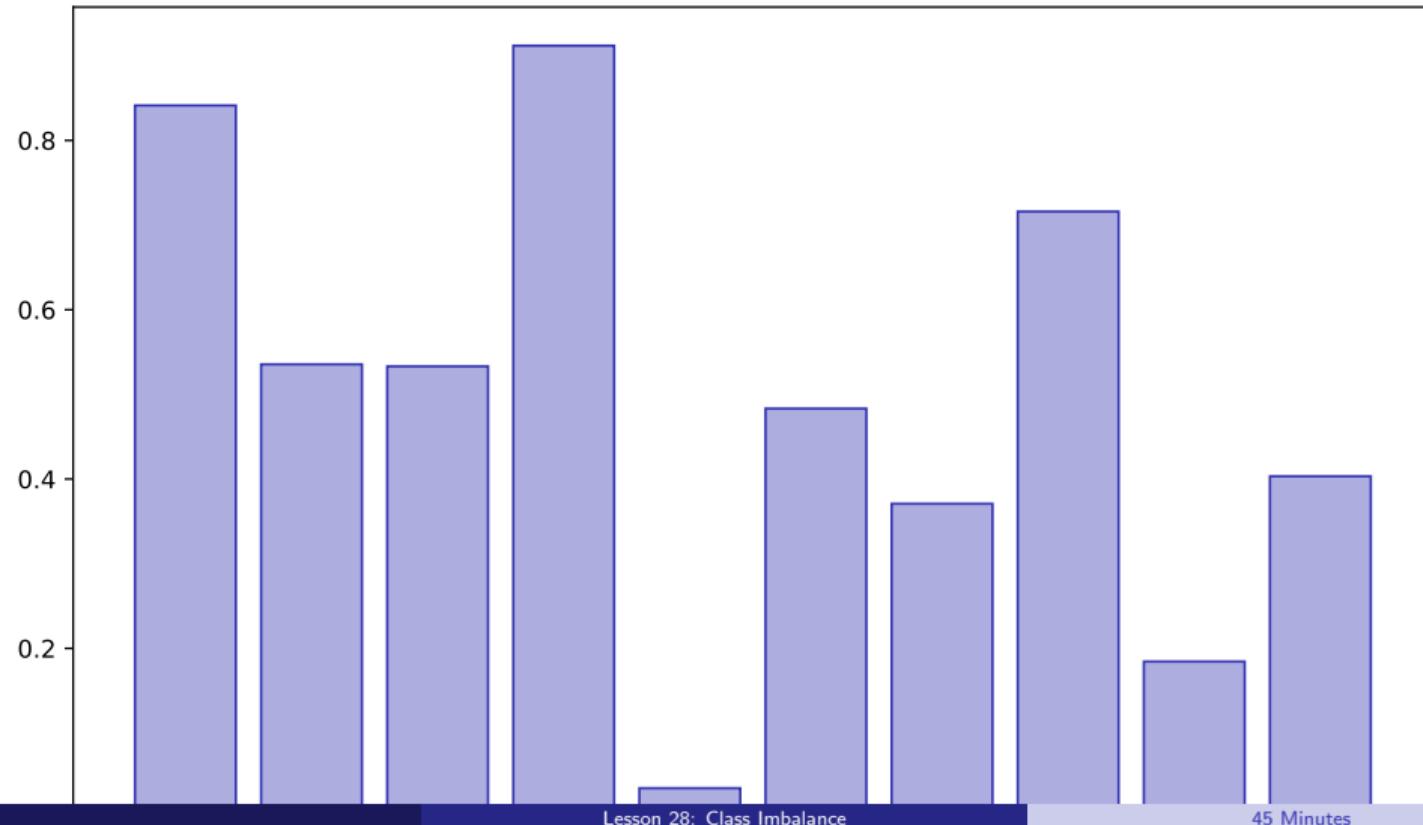


Stratified Cv

Stratified Cv

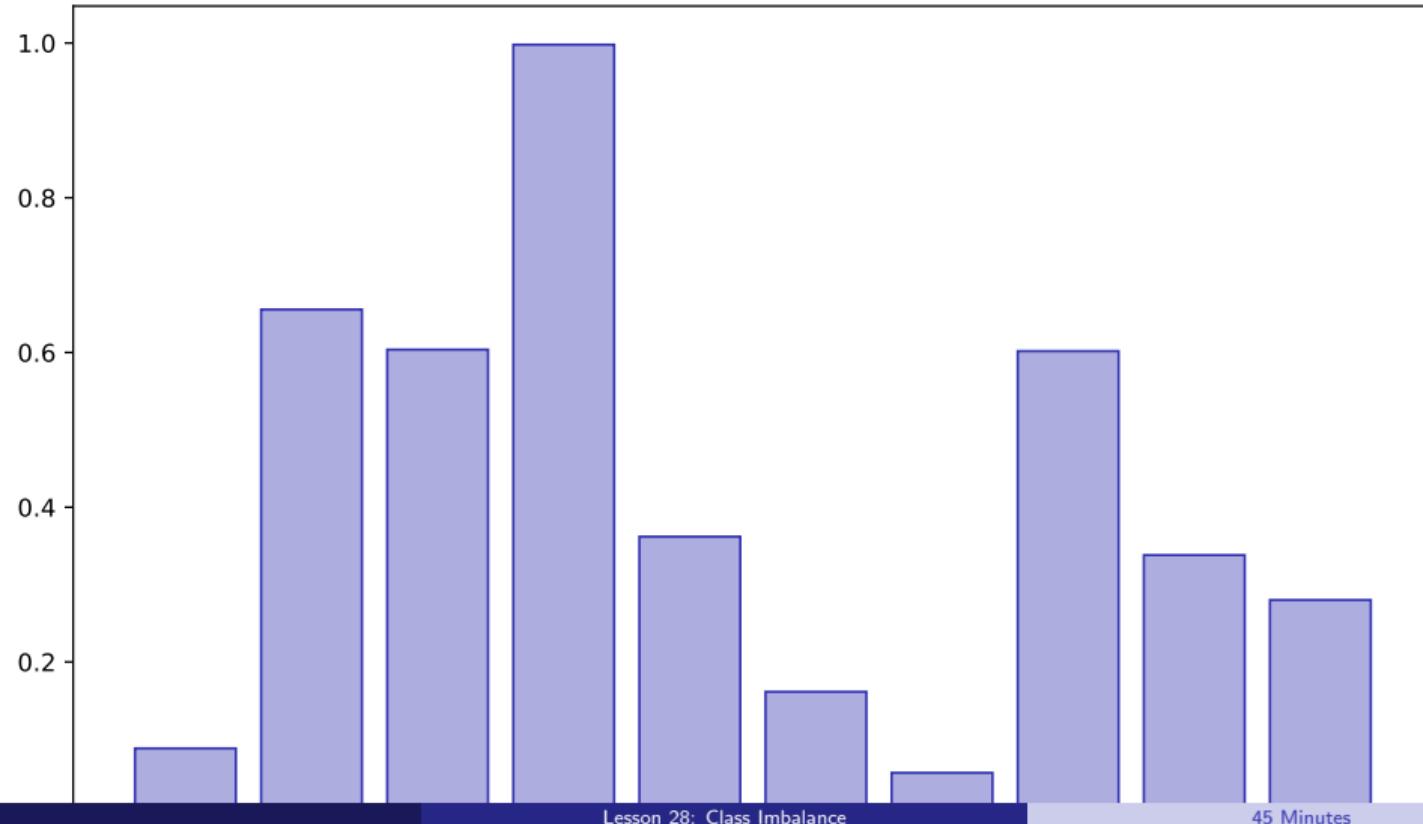


Precision Recall Tradeoff

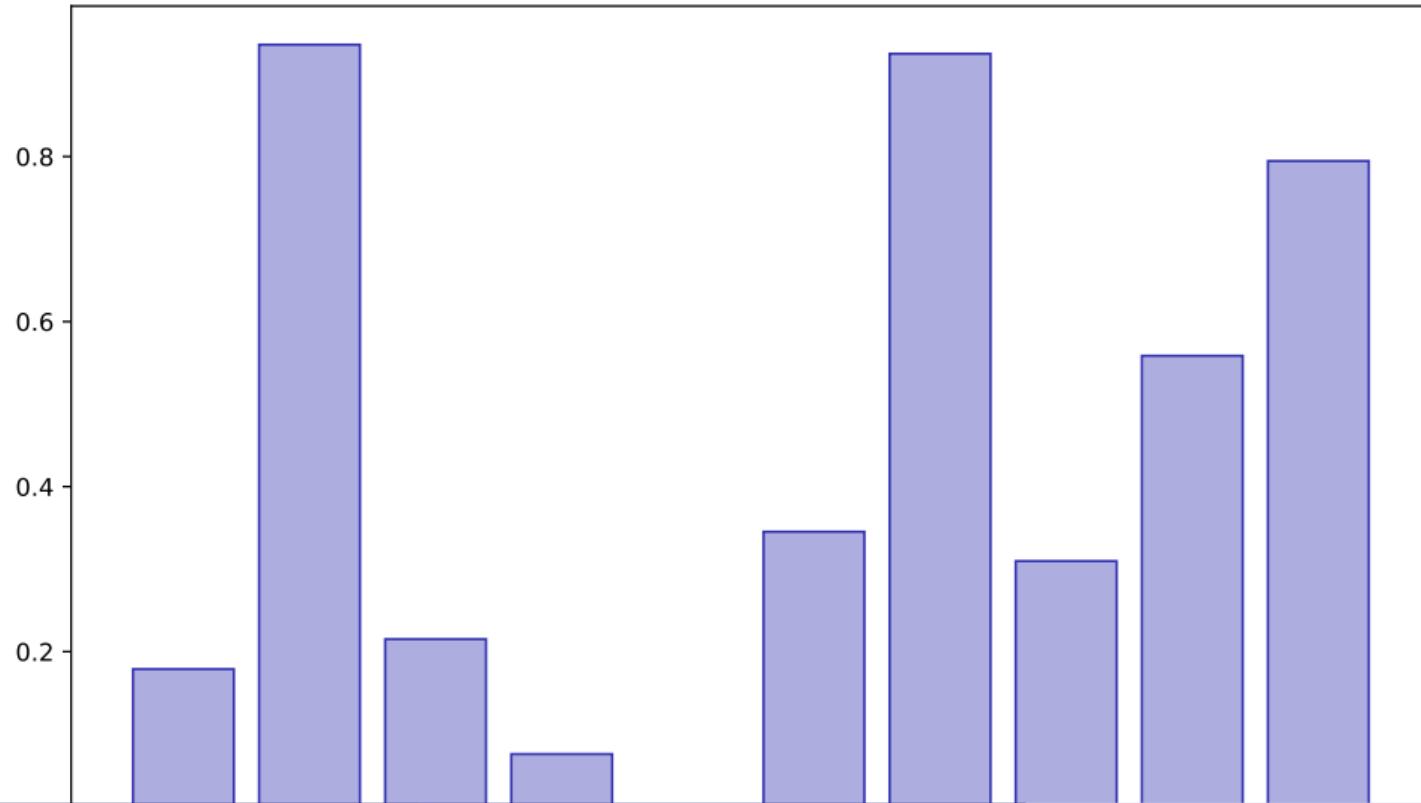


Cost Sensitive

Cost Sensitive



Fraud Detection



Lesson Summary

Key Takeaways:

- Identify imbalanced datasets
- Apply SMOTE oversampling
- Use class weights
- Evaluate fairly

Apply these skills in your final project

Lesson 29: K-Means Clustering

Data Science with Python – BSc Course

45 Minutes

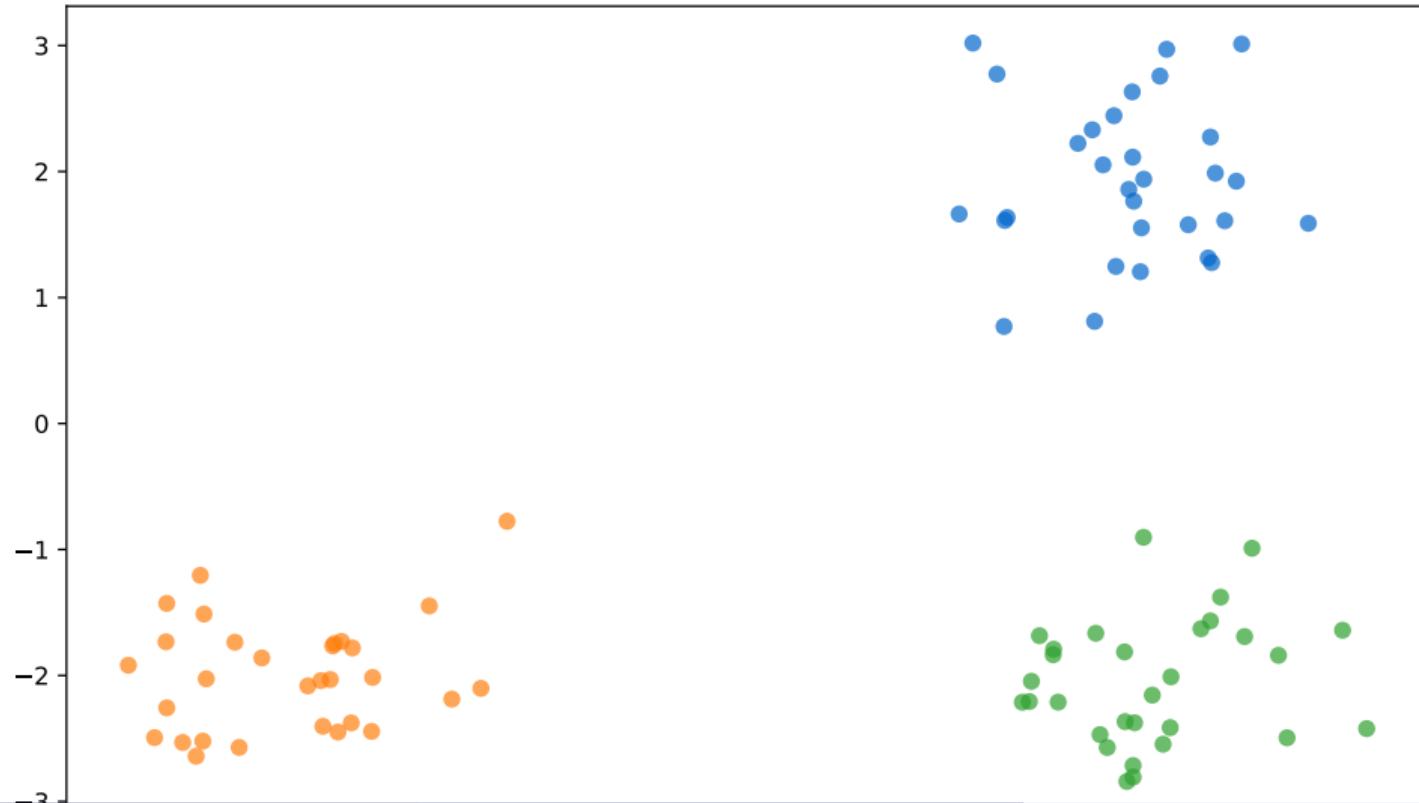
After this lesson, you will be able to:

- Apply K-Means algorithm
- Choose optimal K (elbow method)
- Interpret cluster centers
- Segment financial assets

Building towards your final project

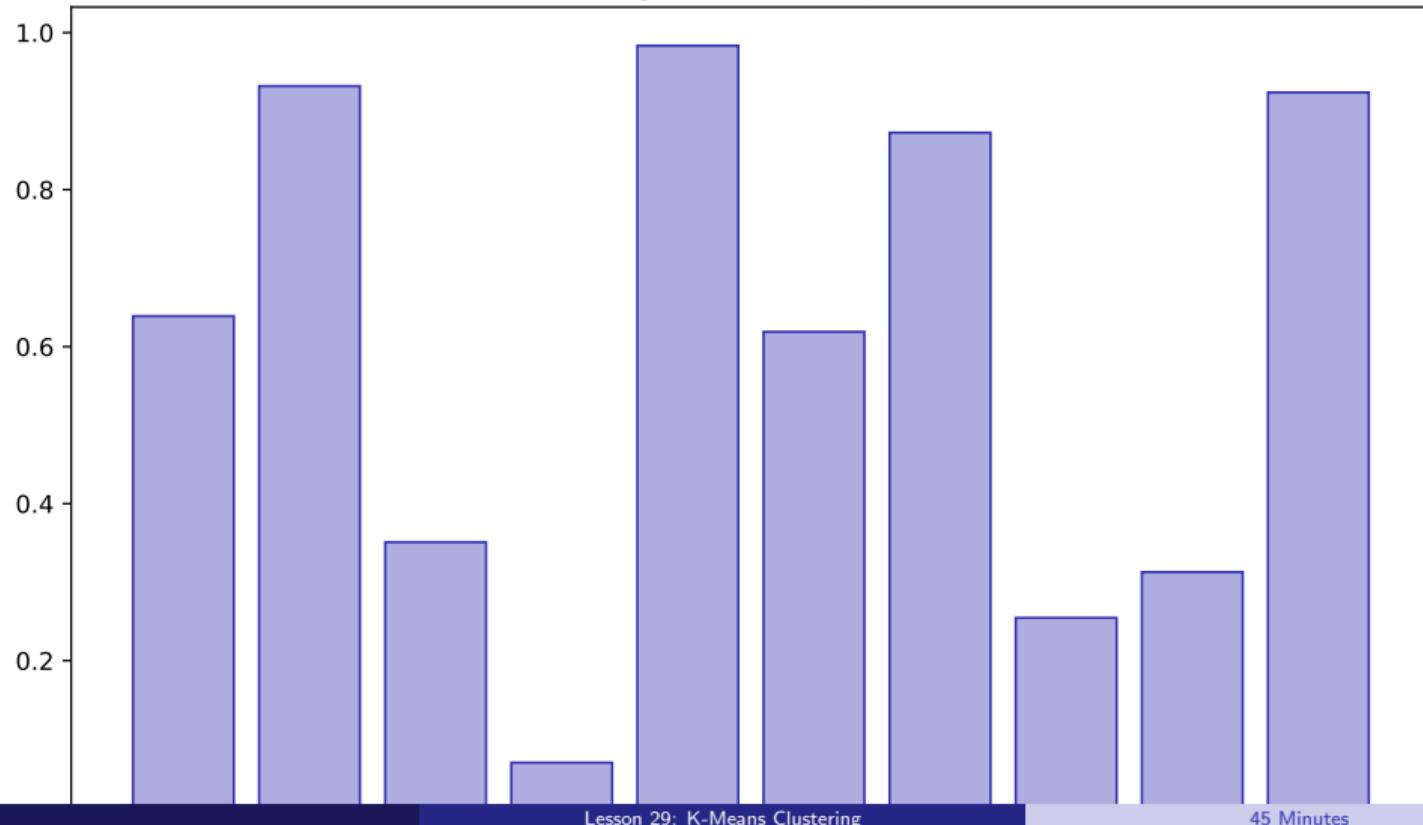
Kmeans Concept

Kmeans Concept

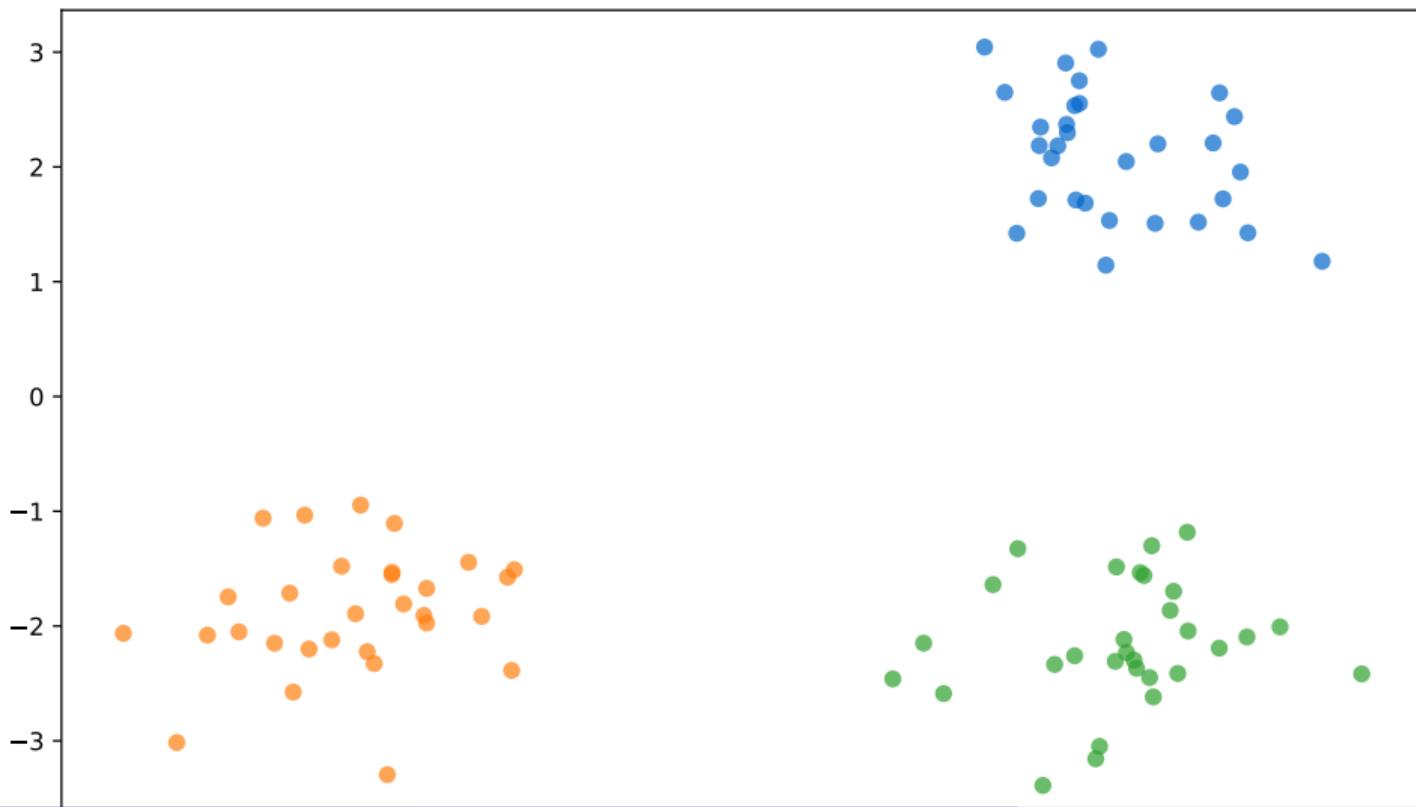


Algorithm Steps

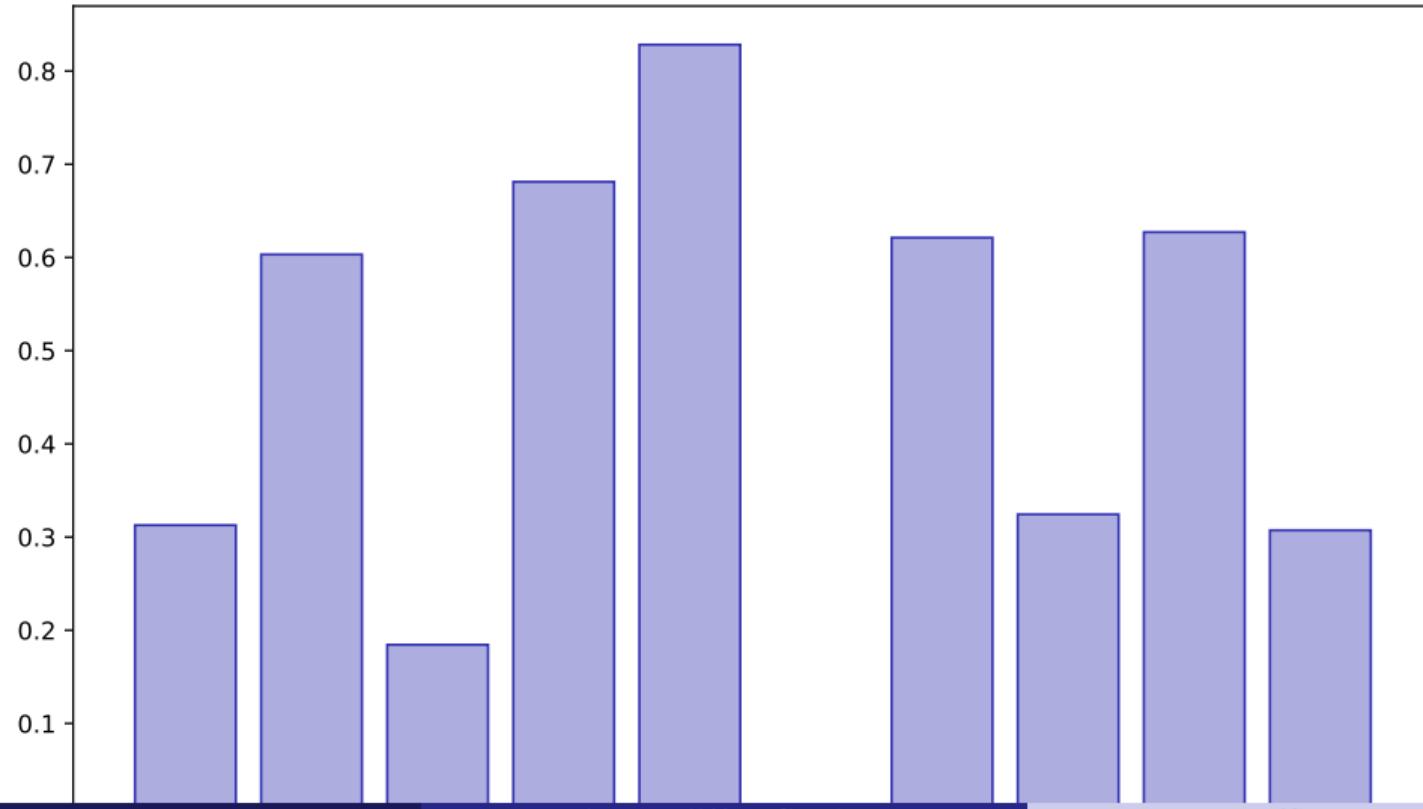
Algorithm Steps



Sklearn Kmeans

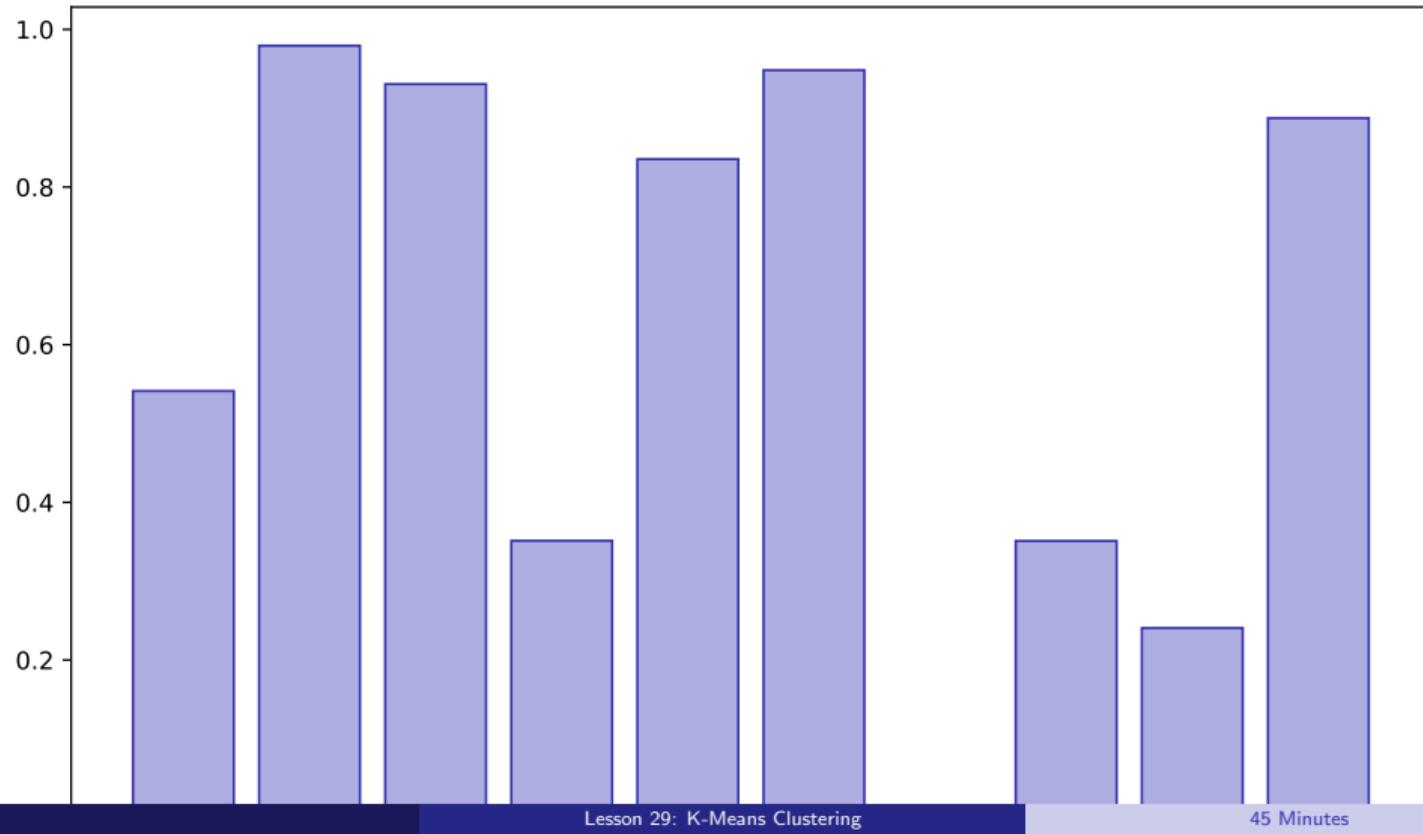


Elbow Method

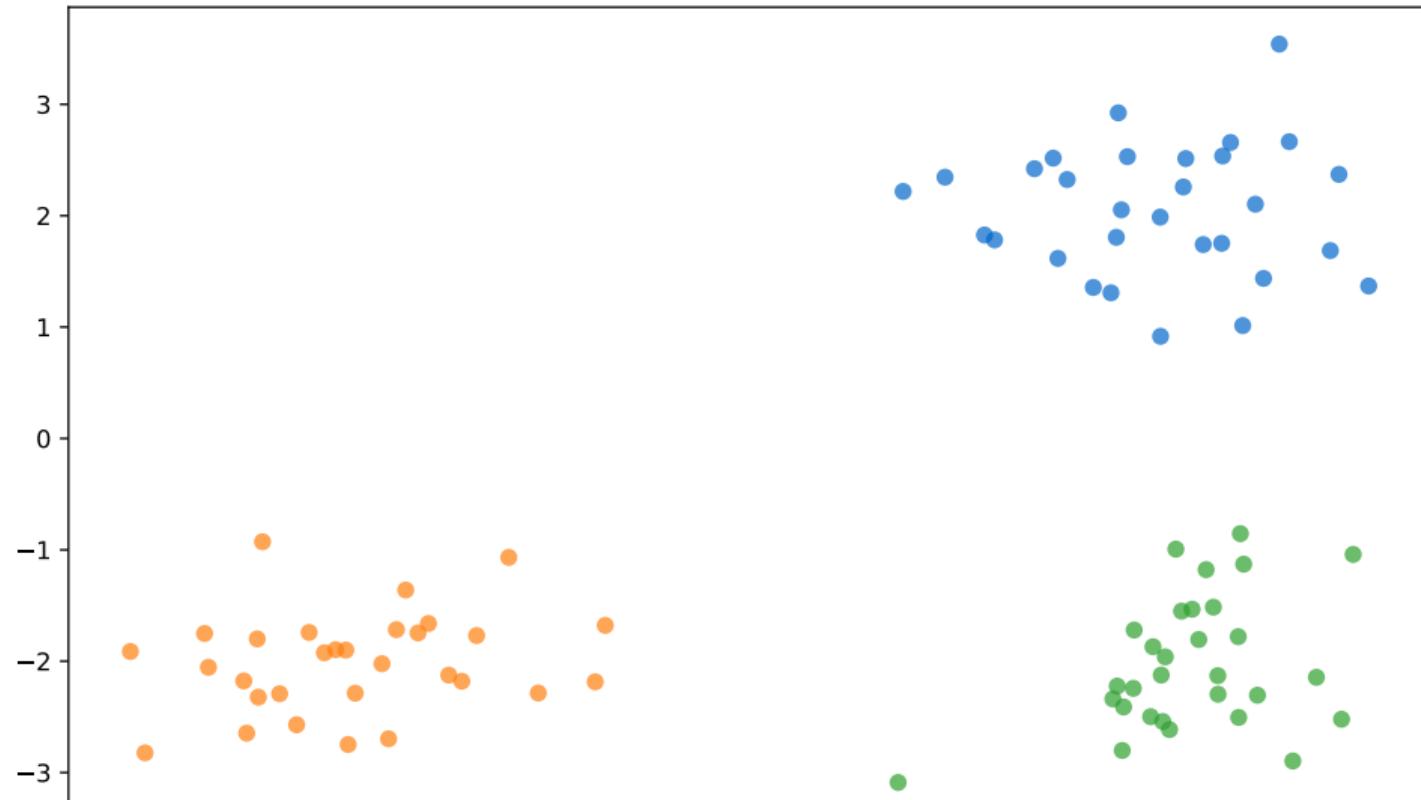


Silhouette Score

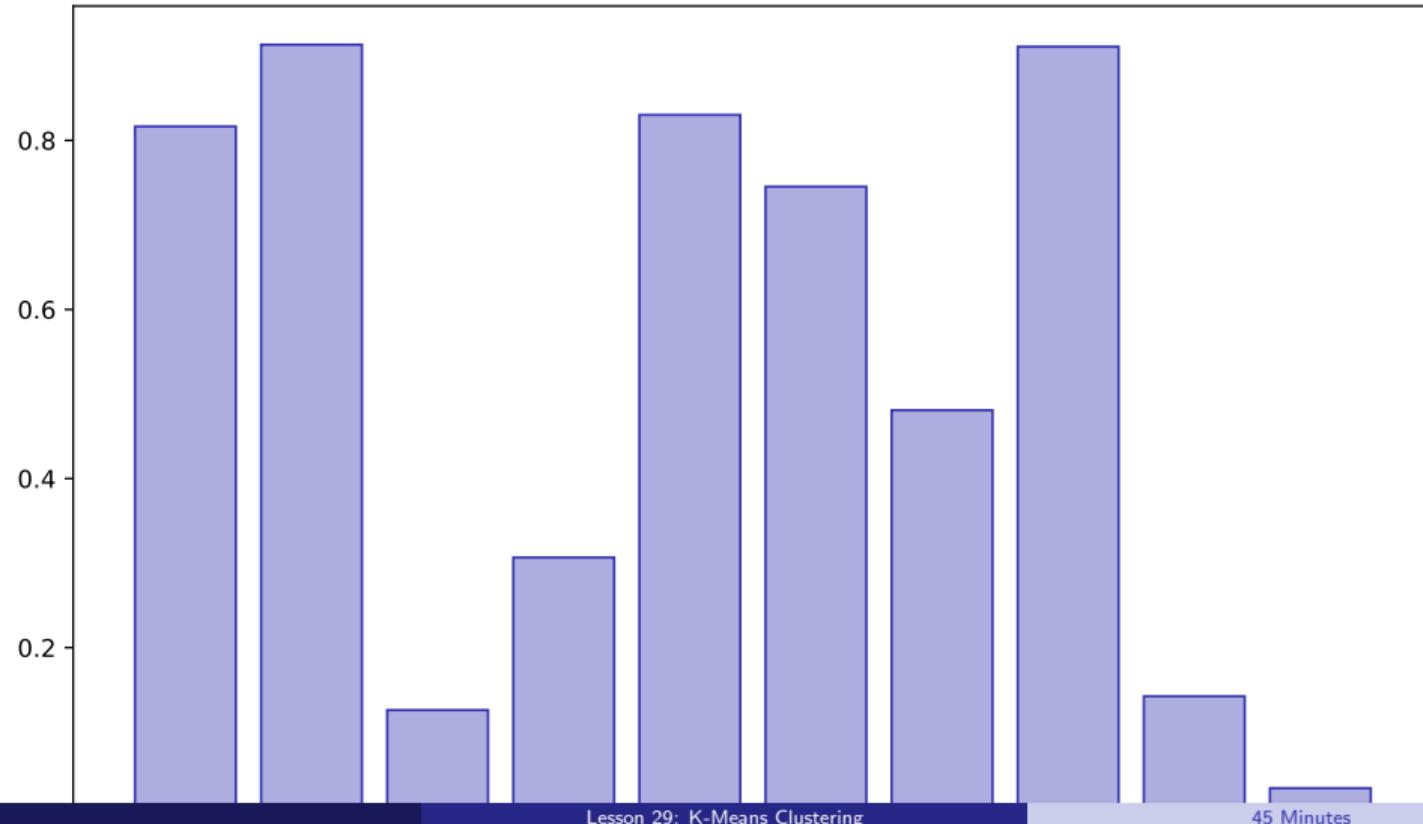
Silhouette Score



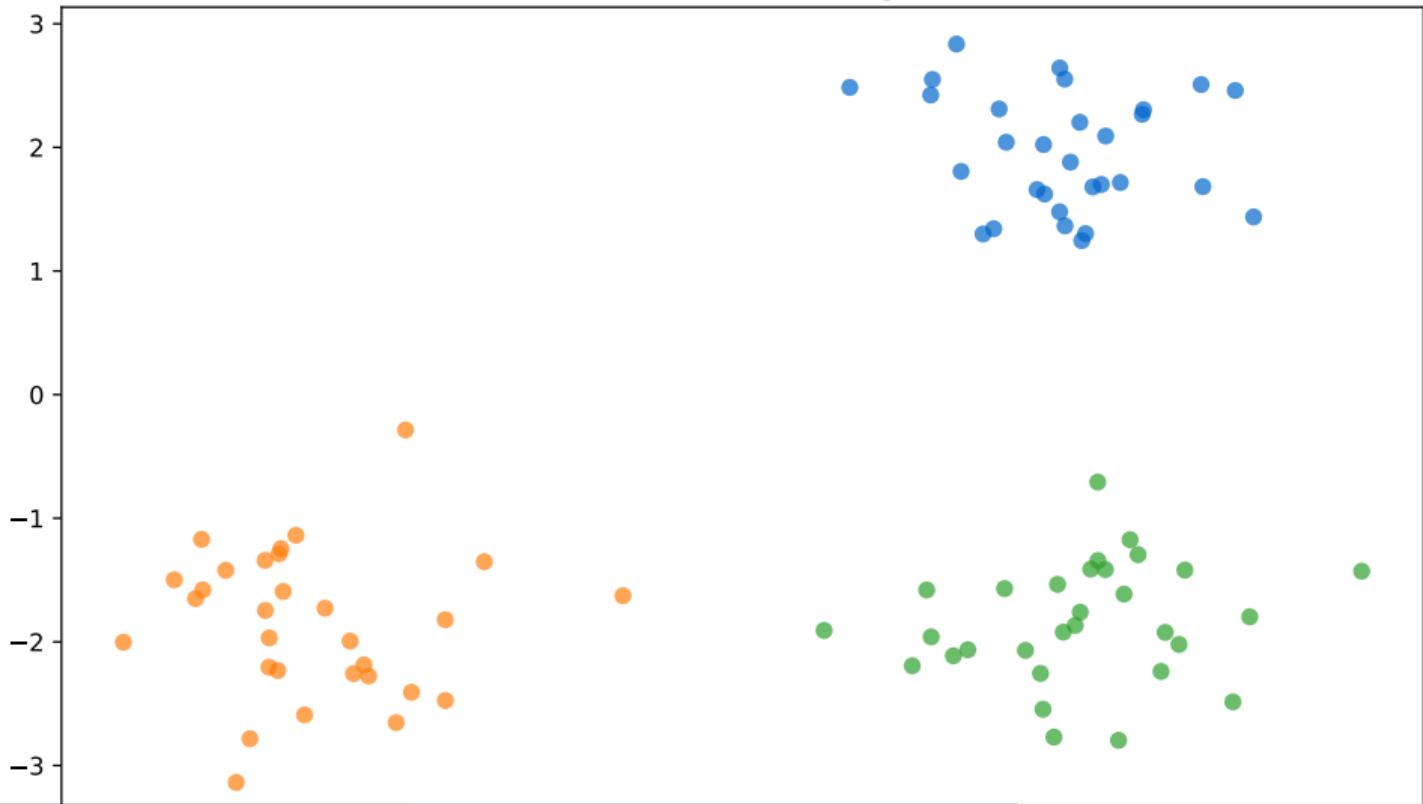
Cluster Visualization



Centroid Interpretation



Asset Clustering



Lesson Summary

Key Takeaways:

- Apply K-Means algorithm
- Choose optimal K (elbow method)
- Interpret cluster centers
- Segment financial assets

Apply these skills in your final project

Lesson 30: Hierarchical Clustering

Data Science with Python – BSc Course

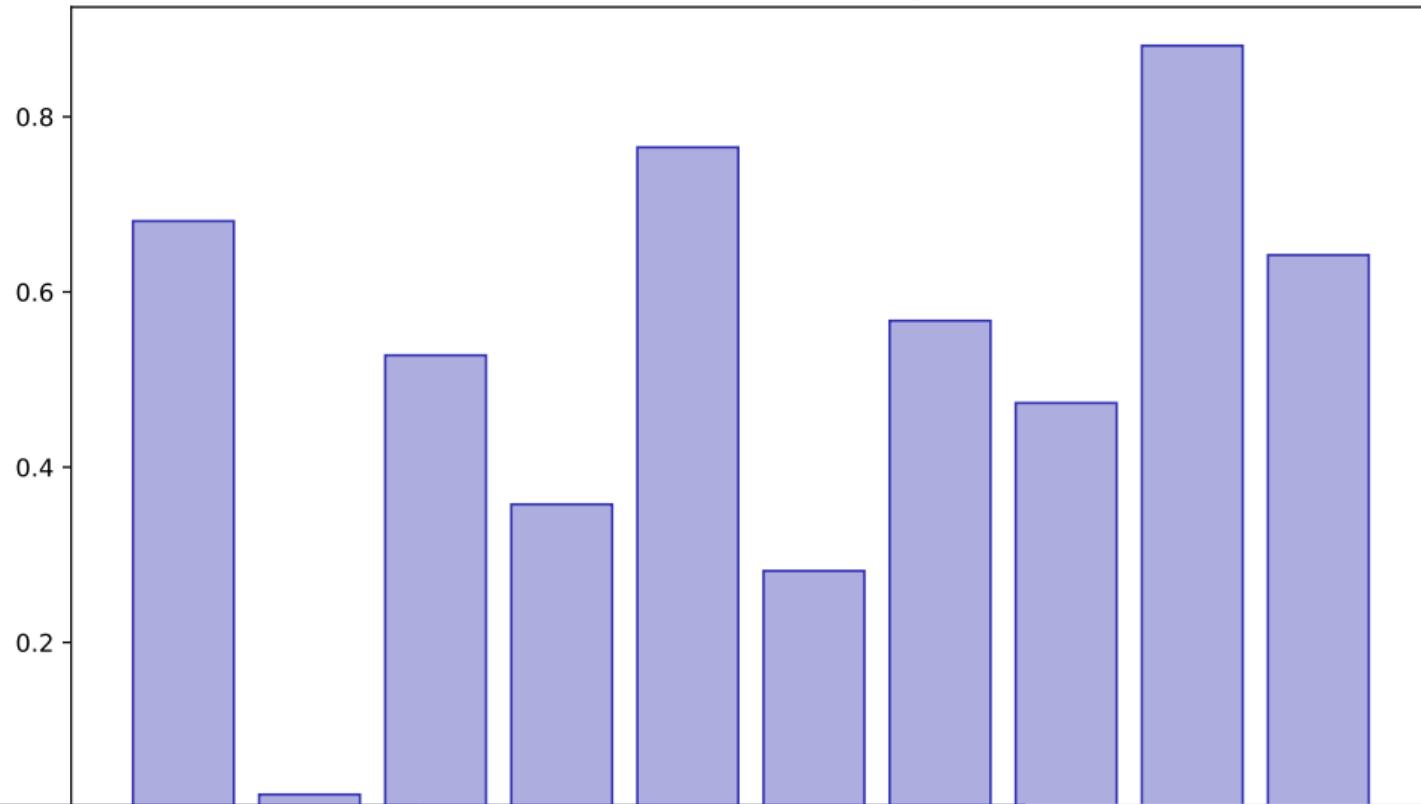
45 Minutes

After this lesson, you will be able to:

- Build dendrograms
- Choose linkage methods
- Cut dendrograms for clusters
- Apply to portfolio construction

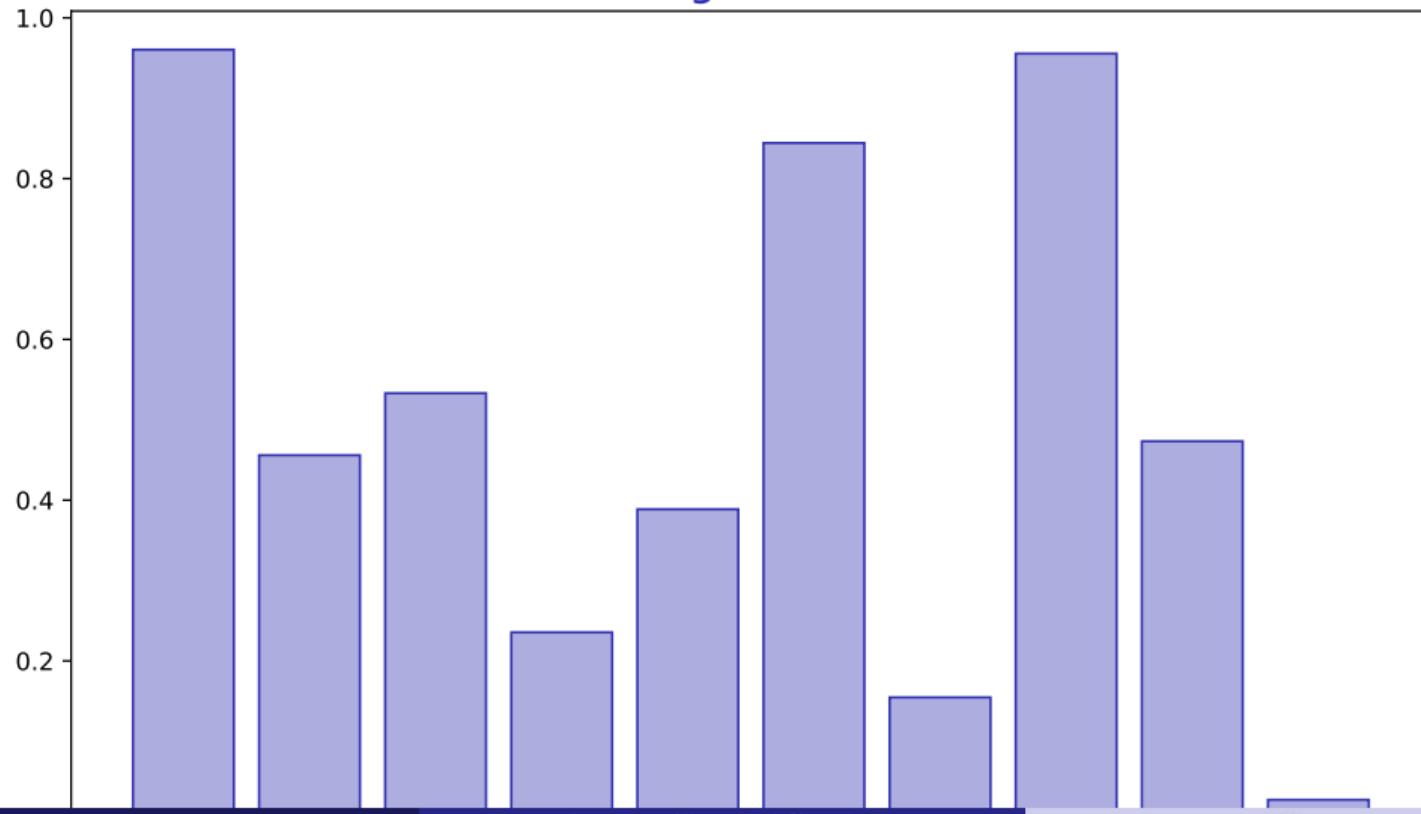
Building towards your final project

Hierarchical Concept



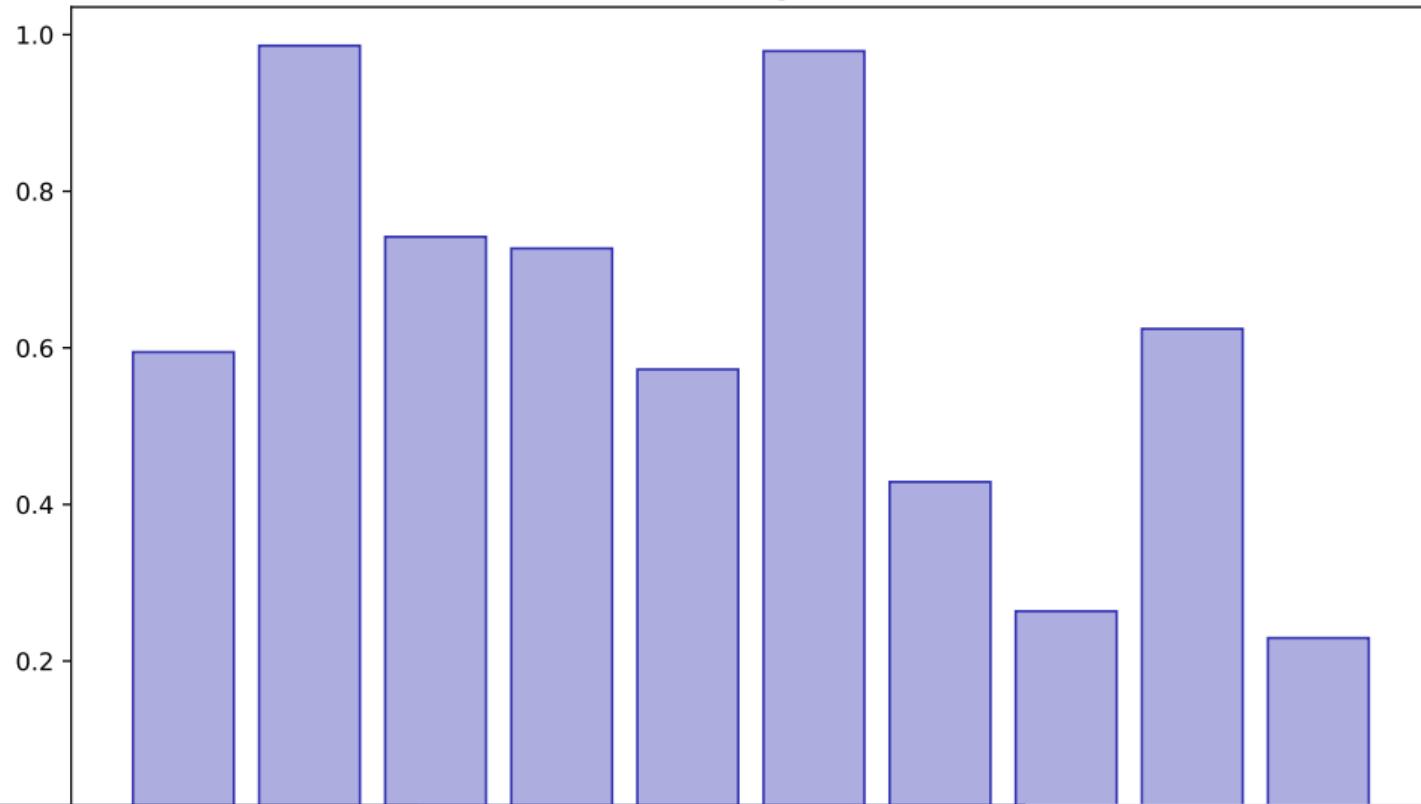
Linkage Methods

Linkage Methods

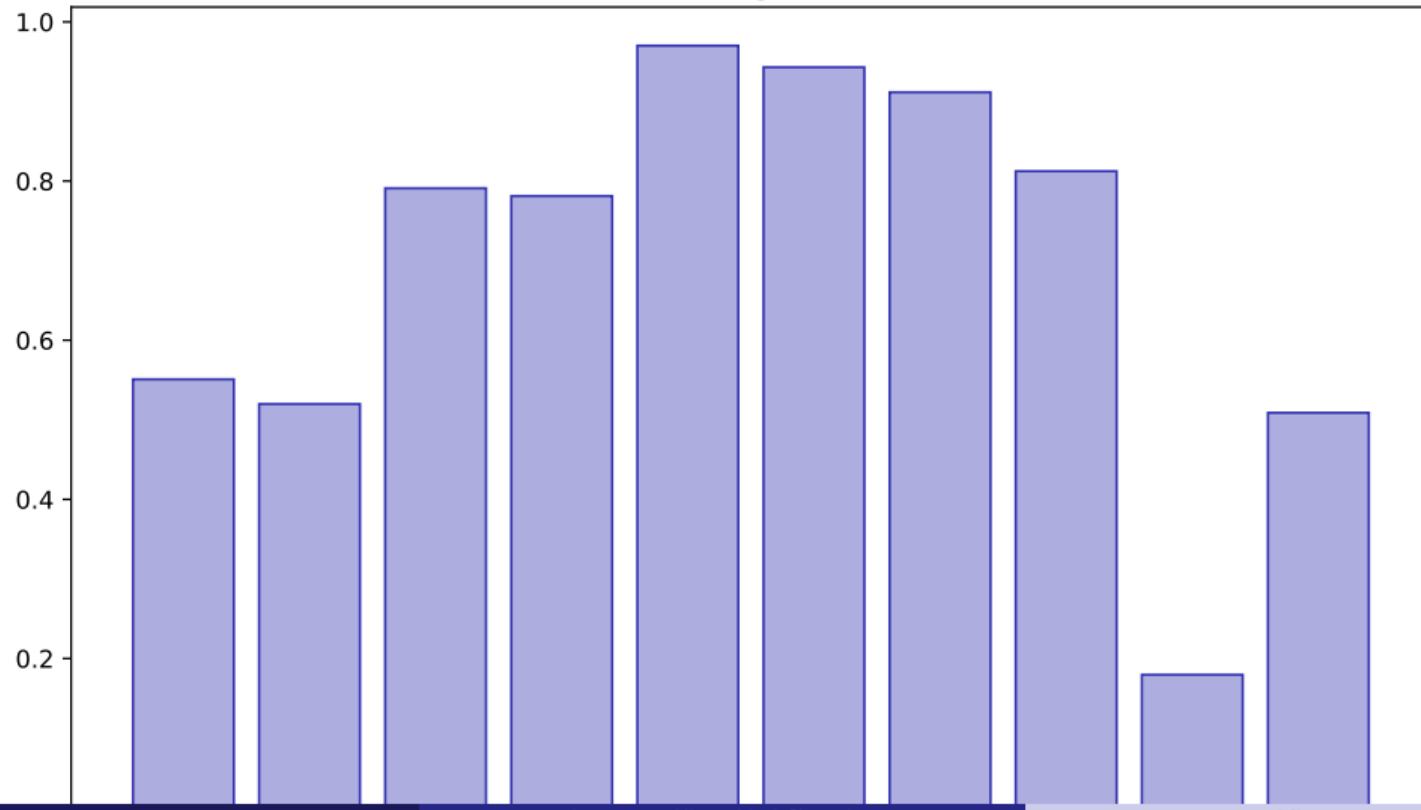


Dendrogram

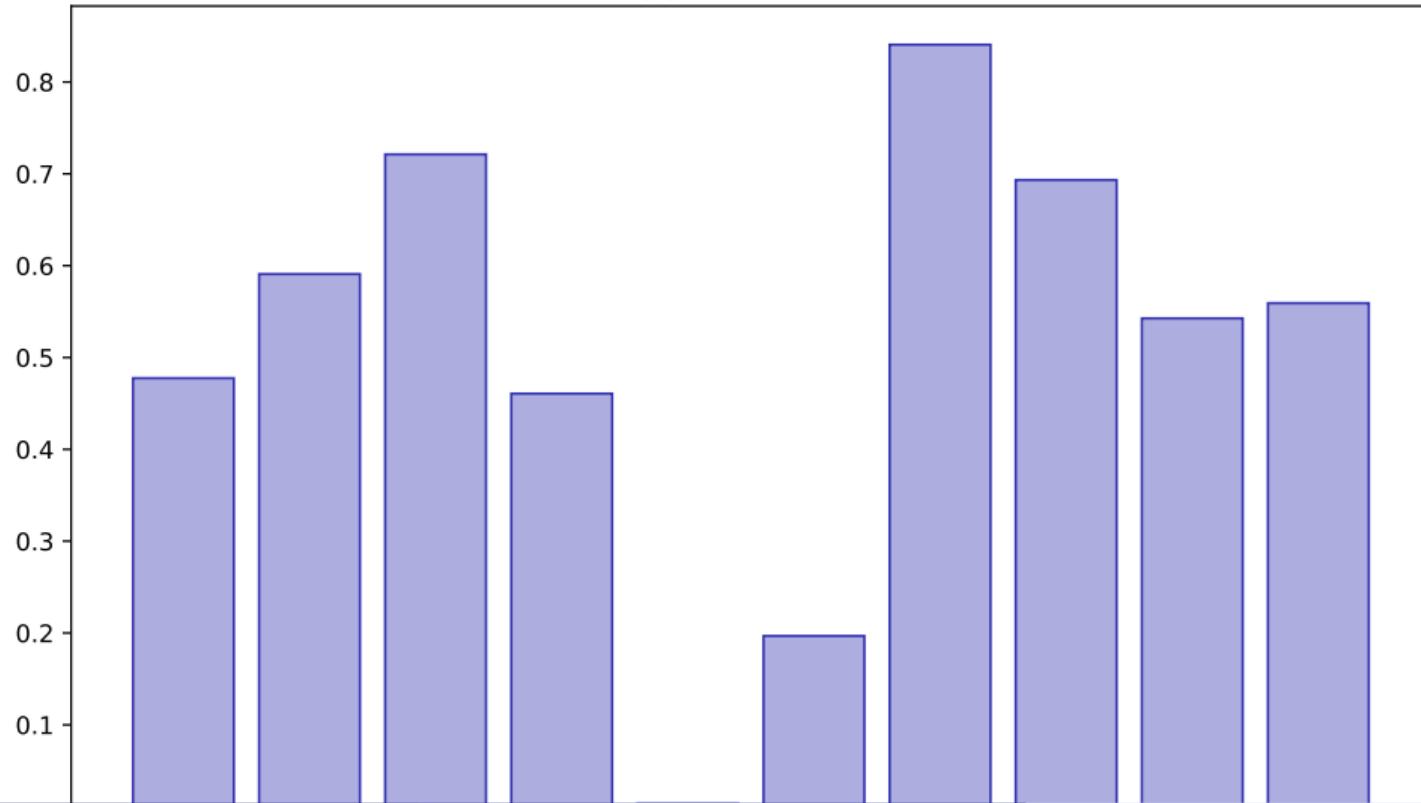
Dendrogram



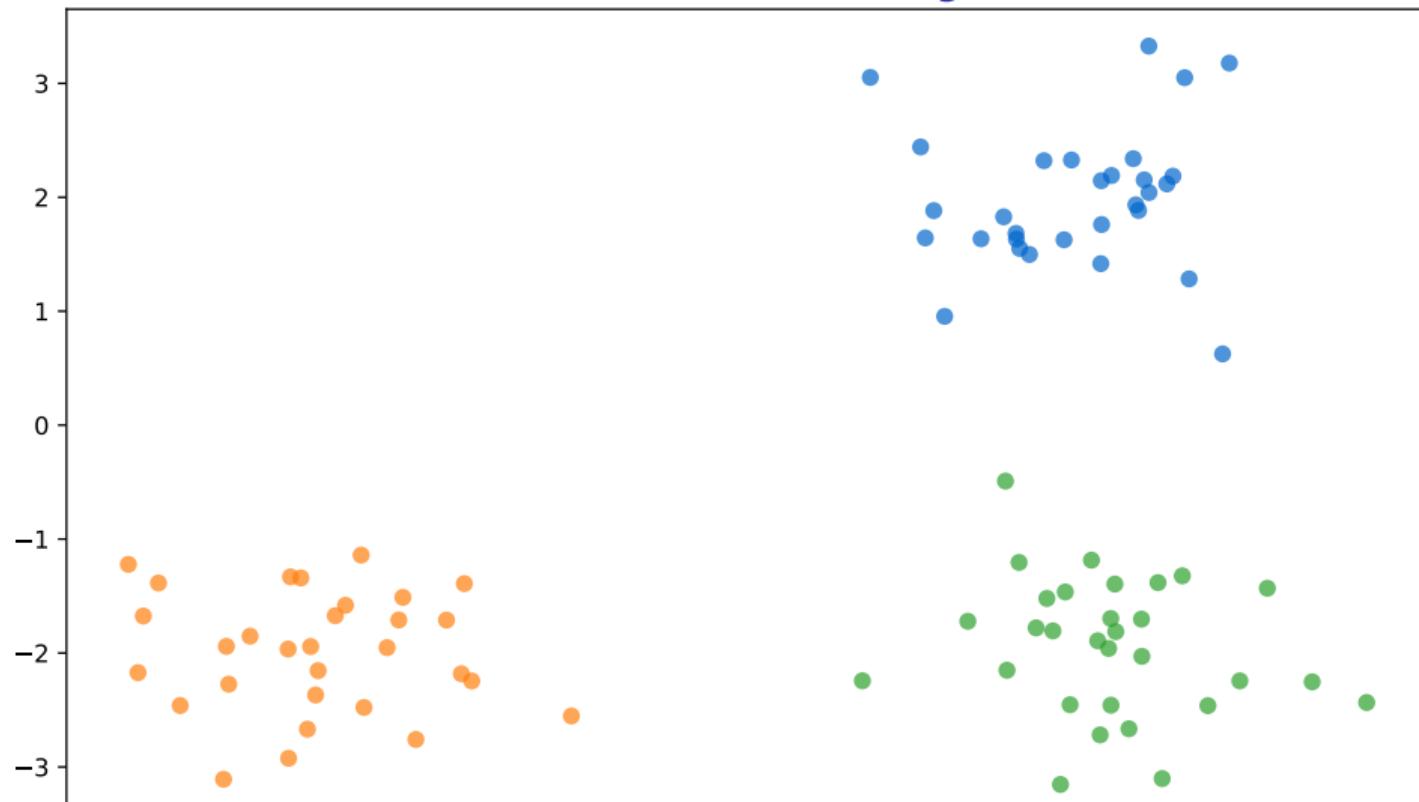
Cutting Tree



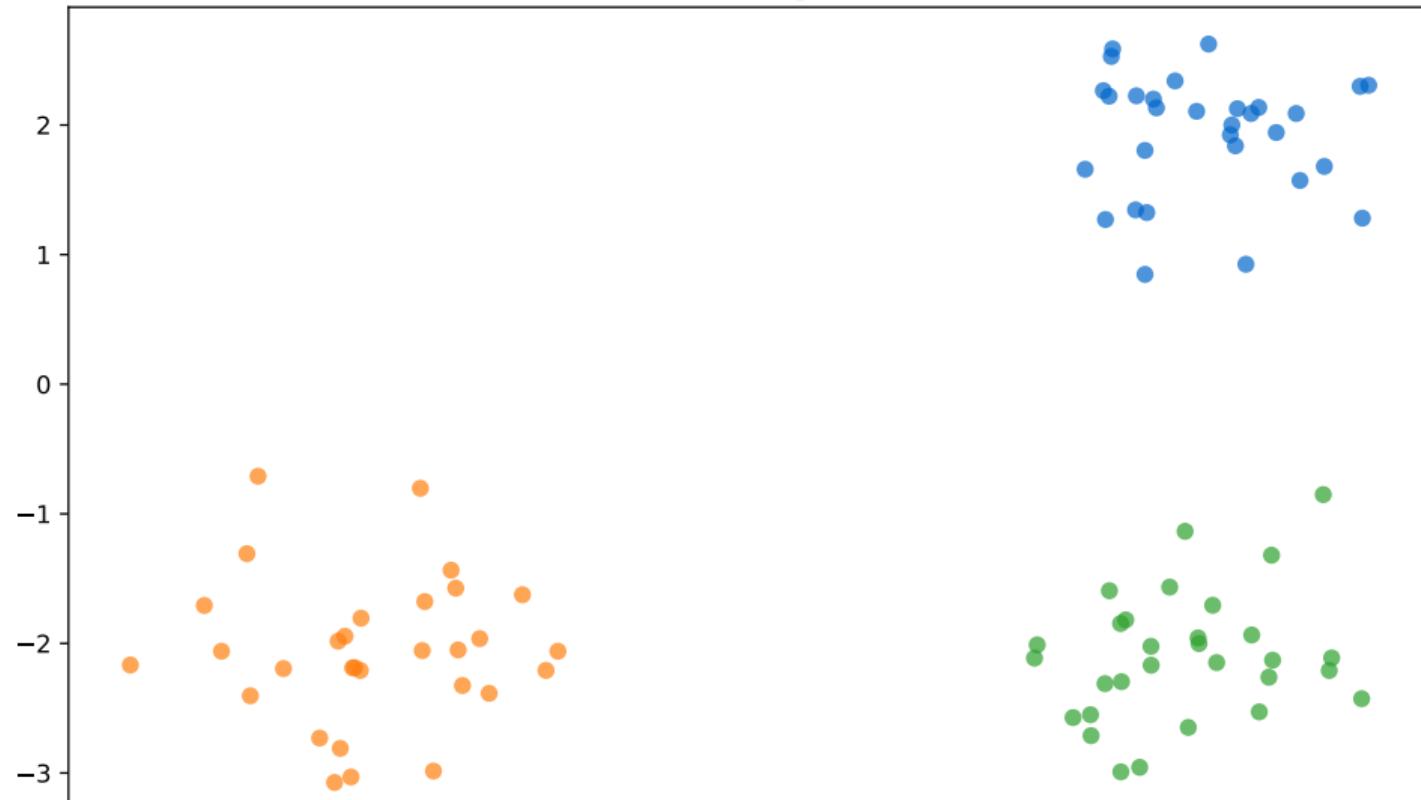
Agglomerative



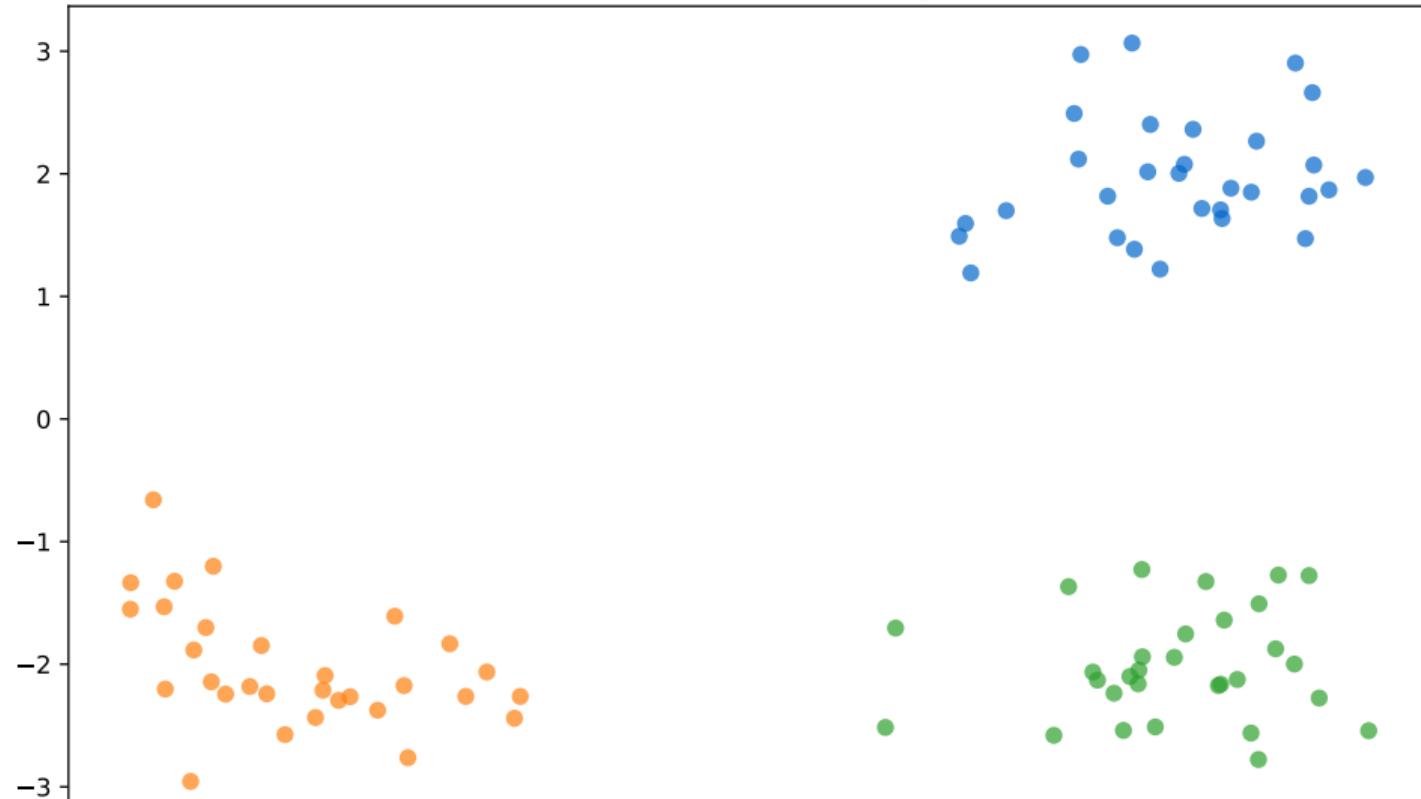
Correlation Clustering



Cluster Comparison



Portfolio Clustering



Key Takeaways:

- Build dendograms
- Choose linkage methods
- Cut dendograms for clusters
- Apply to portfolio construction

Apply these skills in your final project

Lesson 31: PCA Dimensionality Reduction

Data Science with Python – BSc Course

45 Minutes

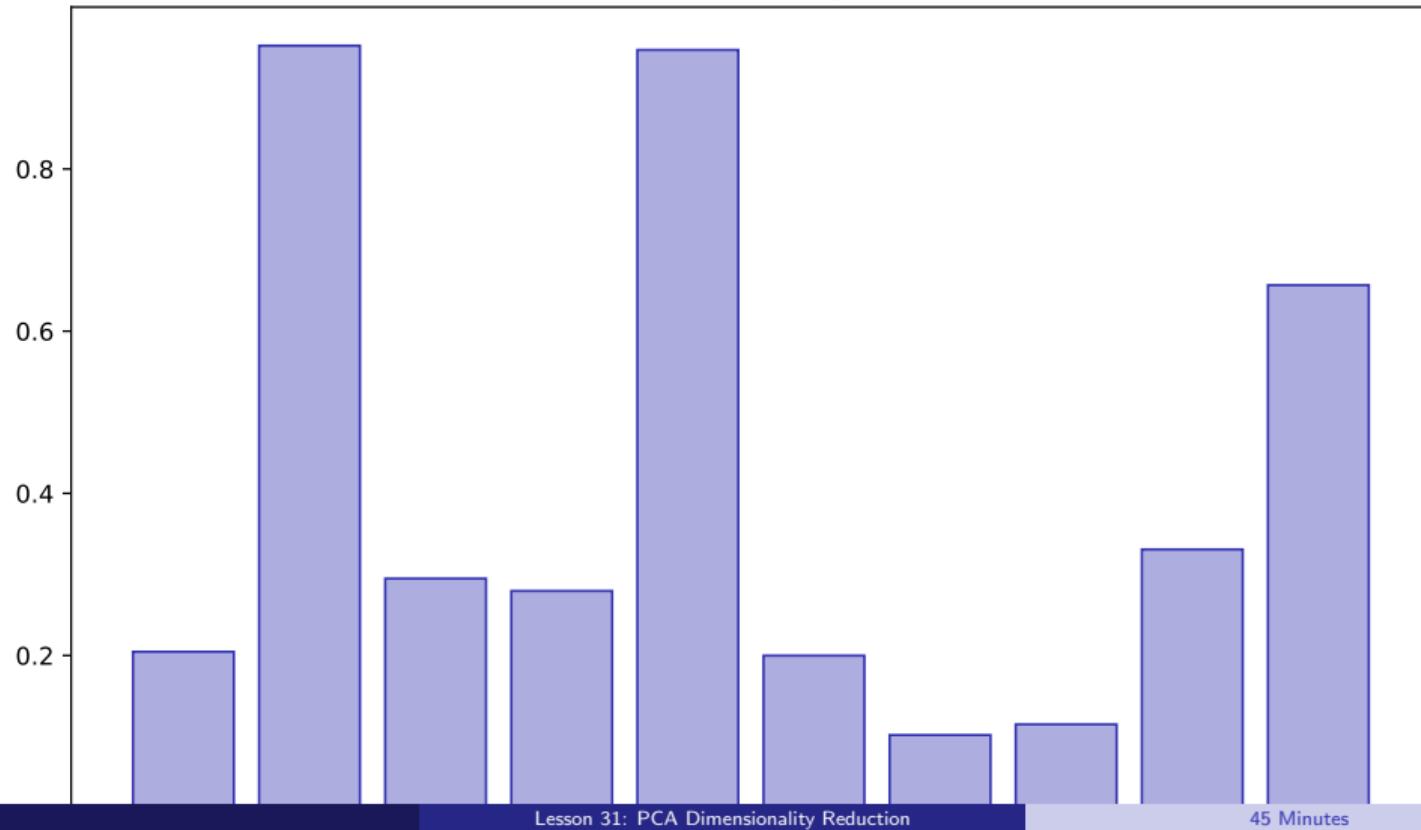
Learning Objectives

After this lesson, you will be able to:

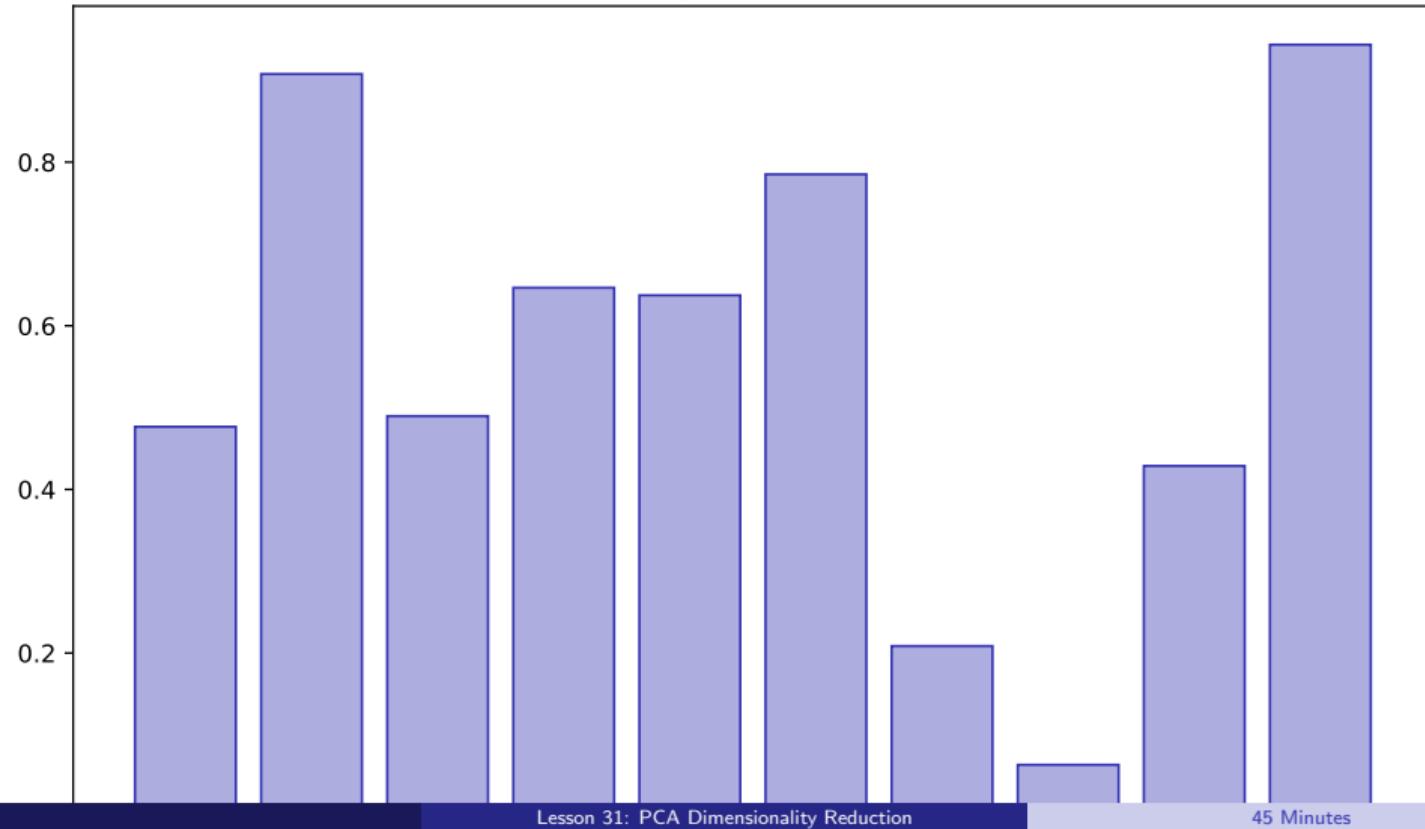
- Understand principal components
- Apply PCA with sklearn
- Interpret explained variance
- Reduce feature dimensions

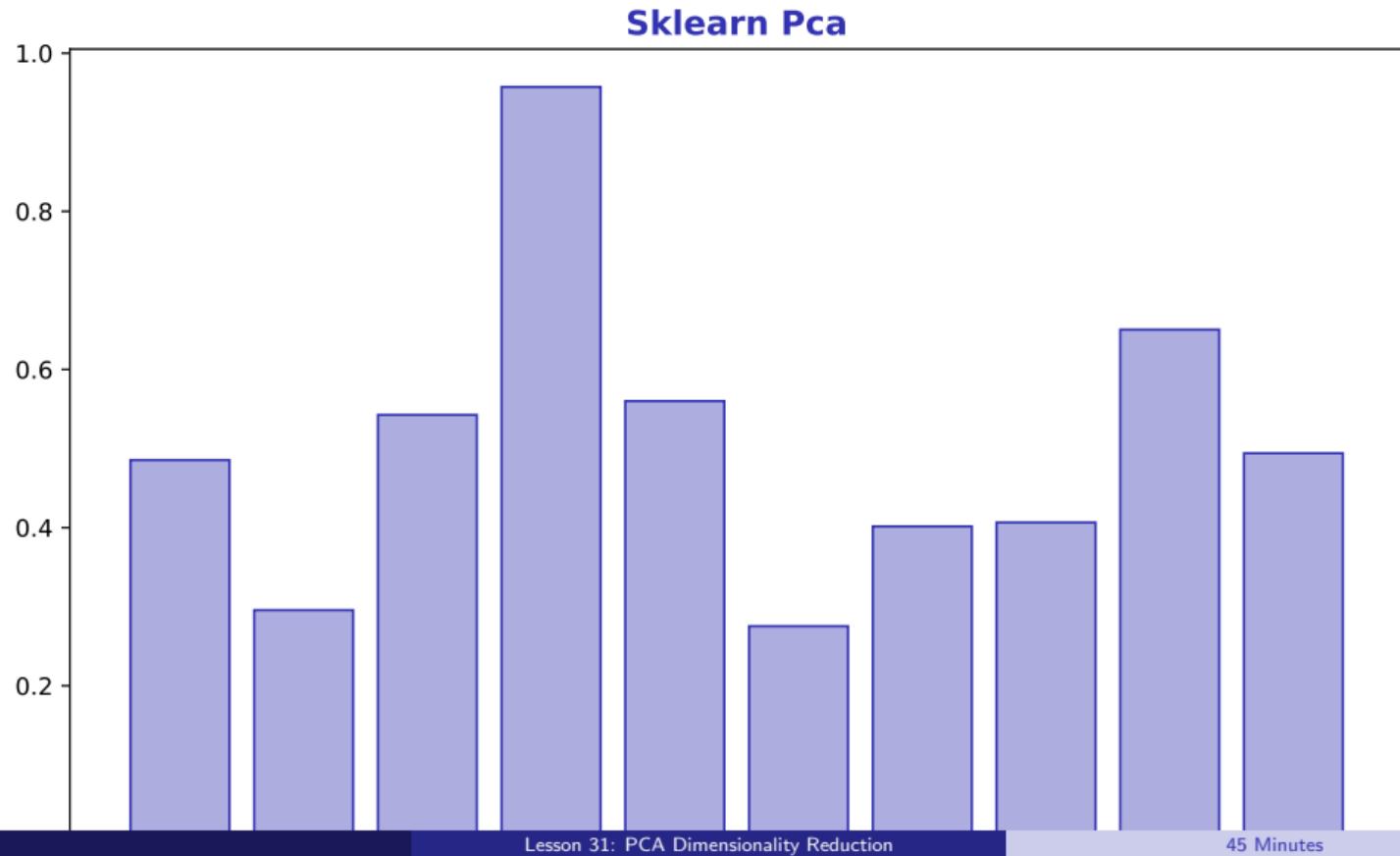
Building towards your final project

Pca Concept

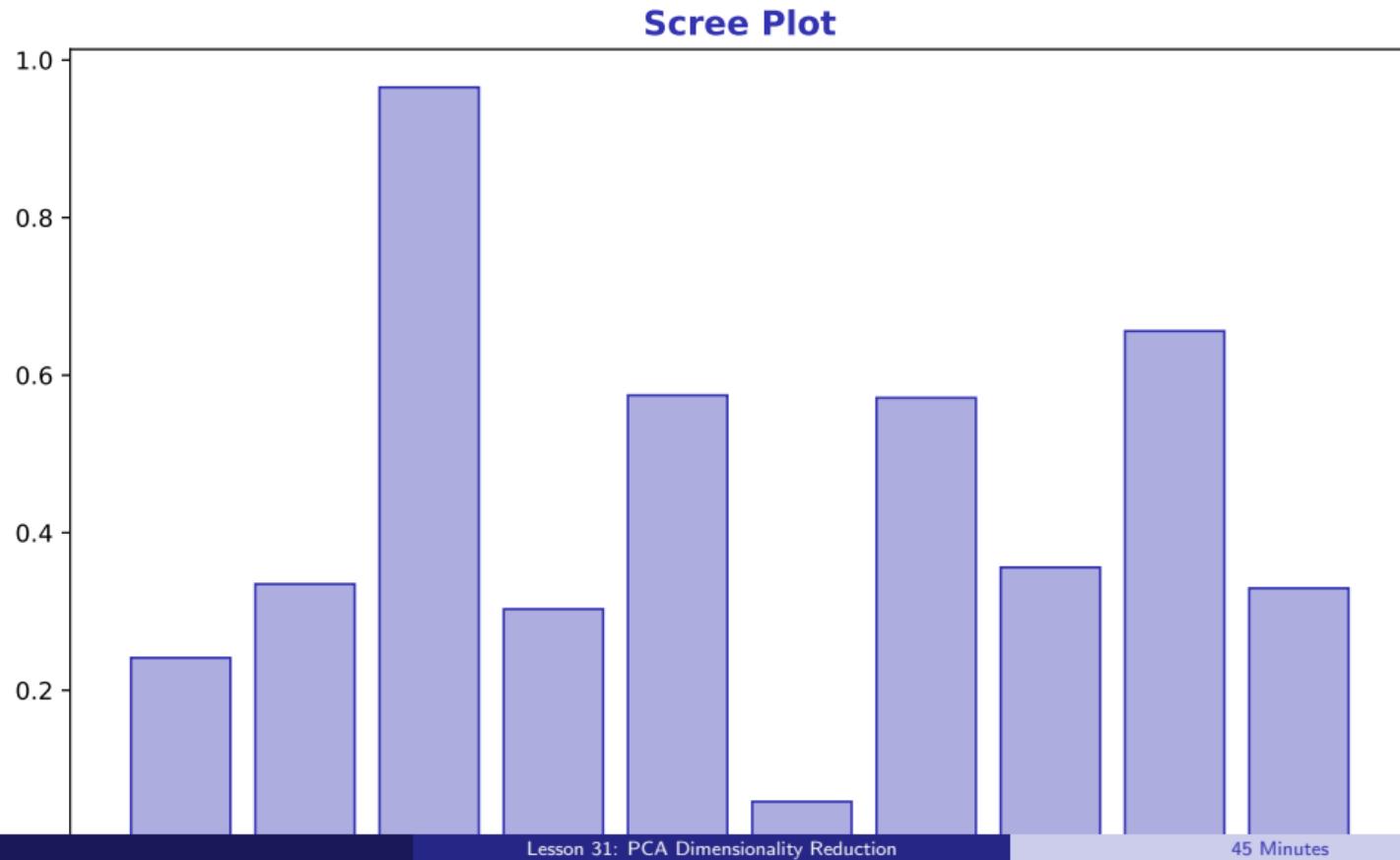


Eigenvalues



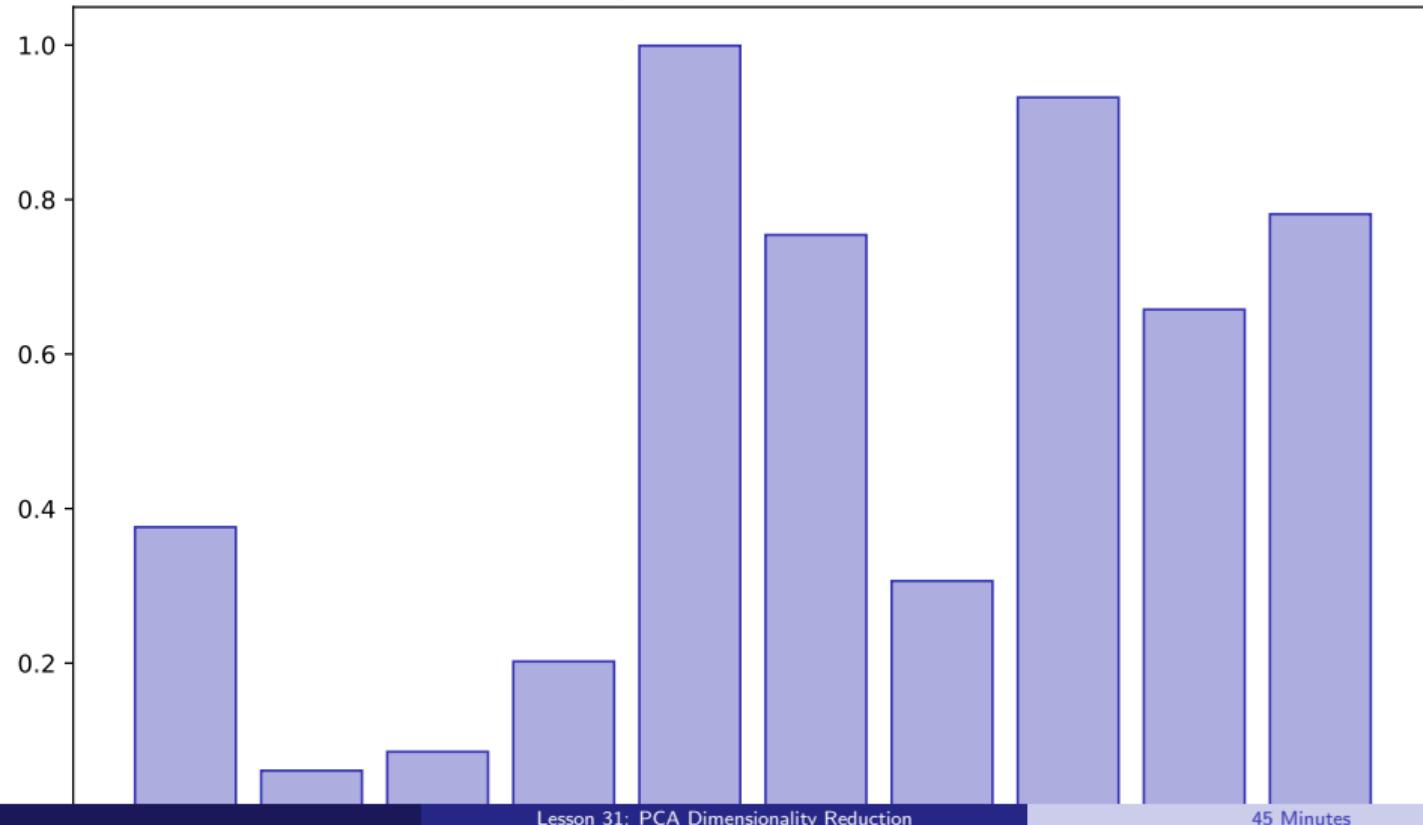


Scree Plot

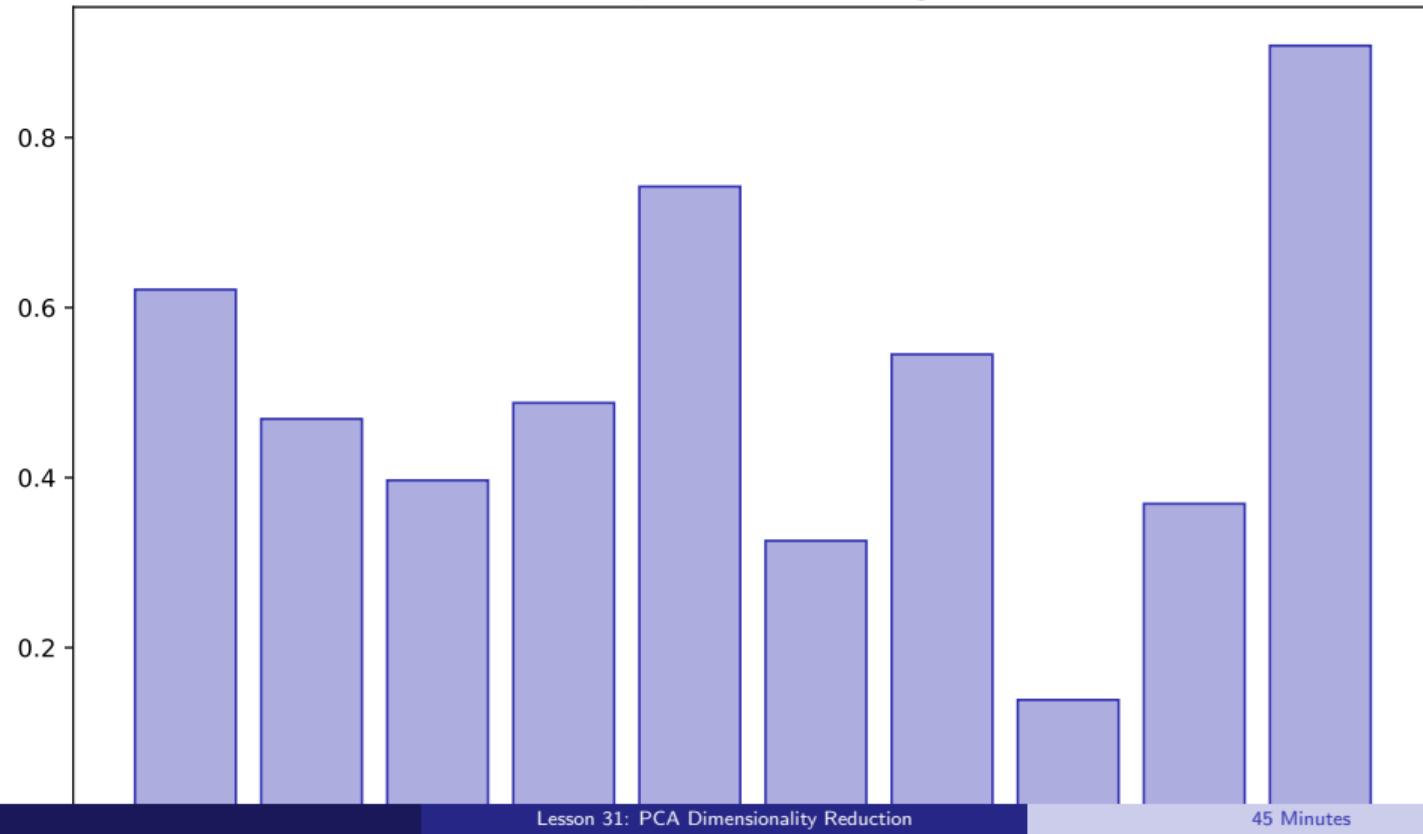


Explained Variance

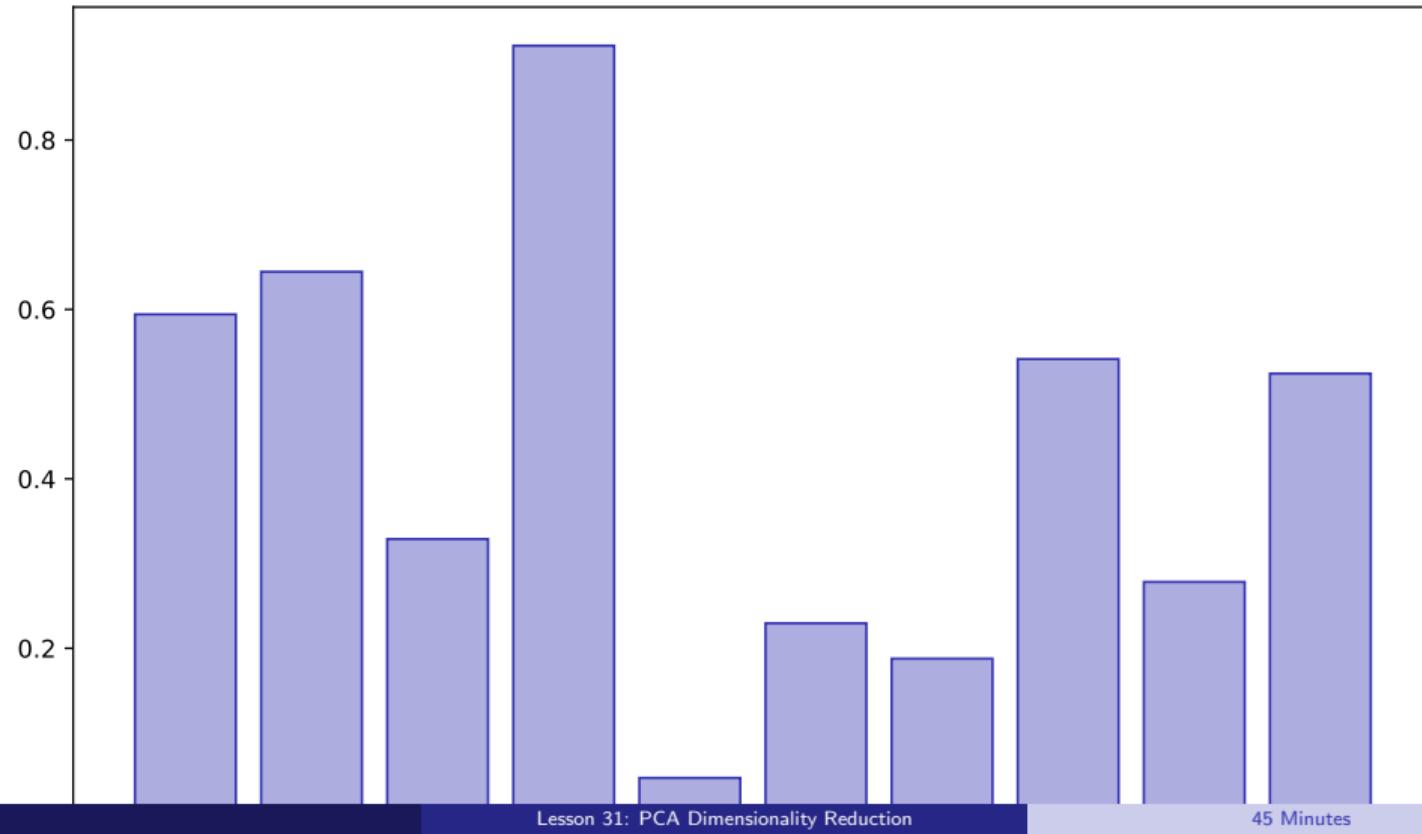
Explained Variance



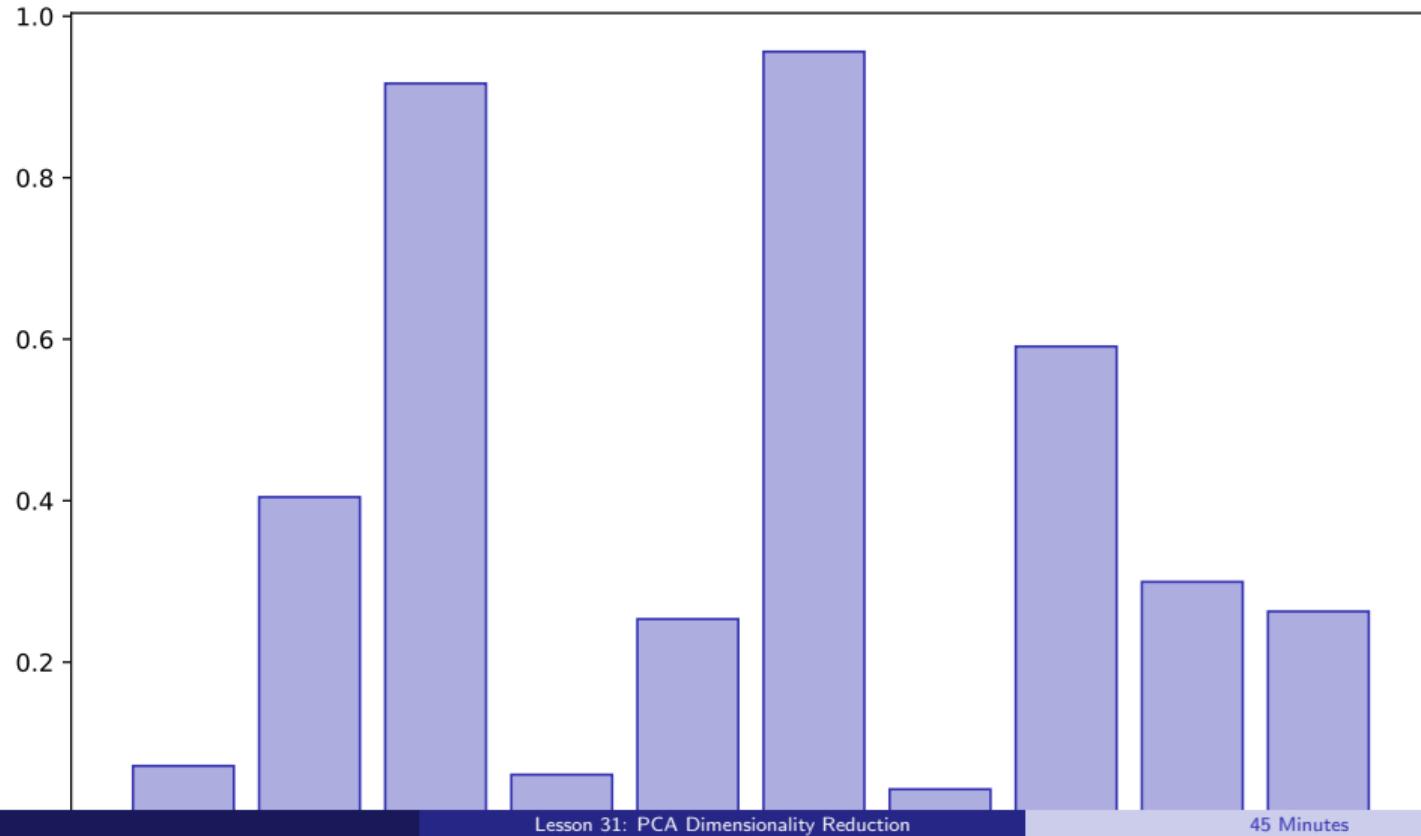
Component Loadings



Visualization



Factor Extraction



Lesson Summary

Key Takeaways:

- Understand principal components
- Apply PCA with sklearn
- Interpret explained variance
- Reduce feature dimensions

Apply these skills in your final project

Lesson 32: Complete ML Pipeline

Data Science with Python – BSc Course

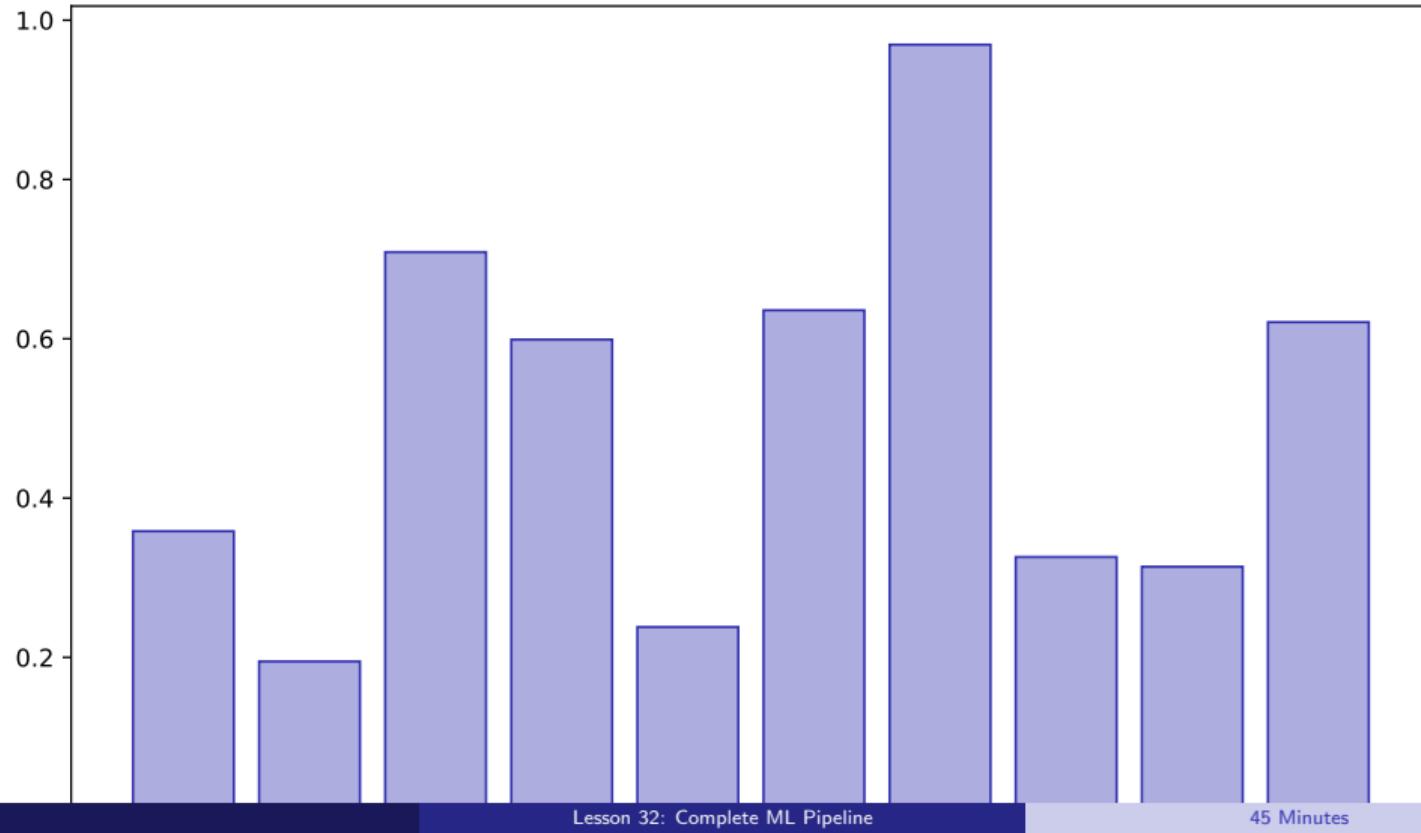
45 Minutes

After this lesson, you will be able to:

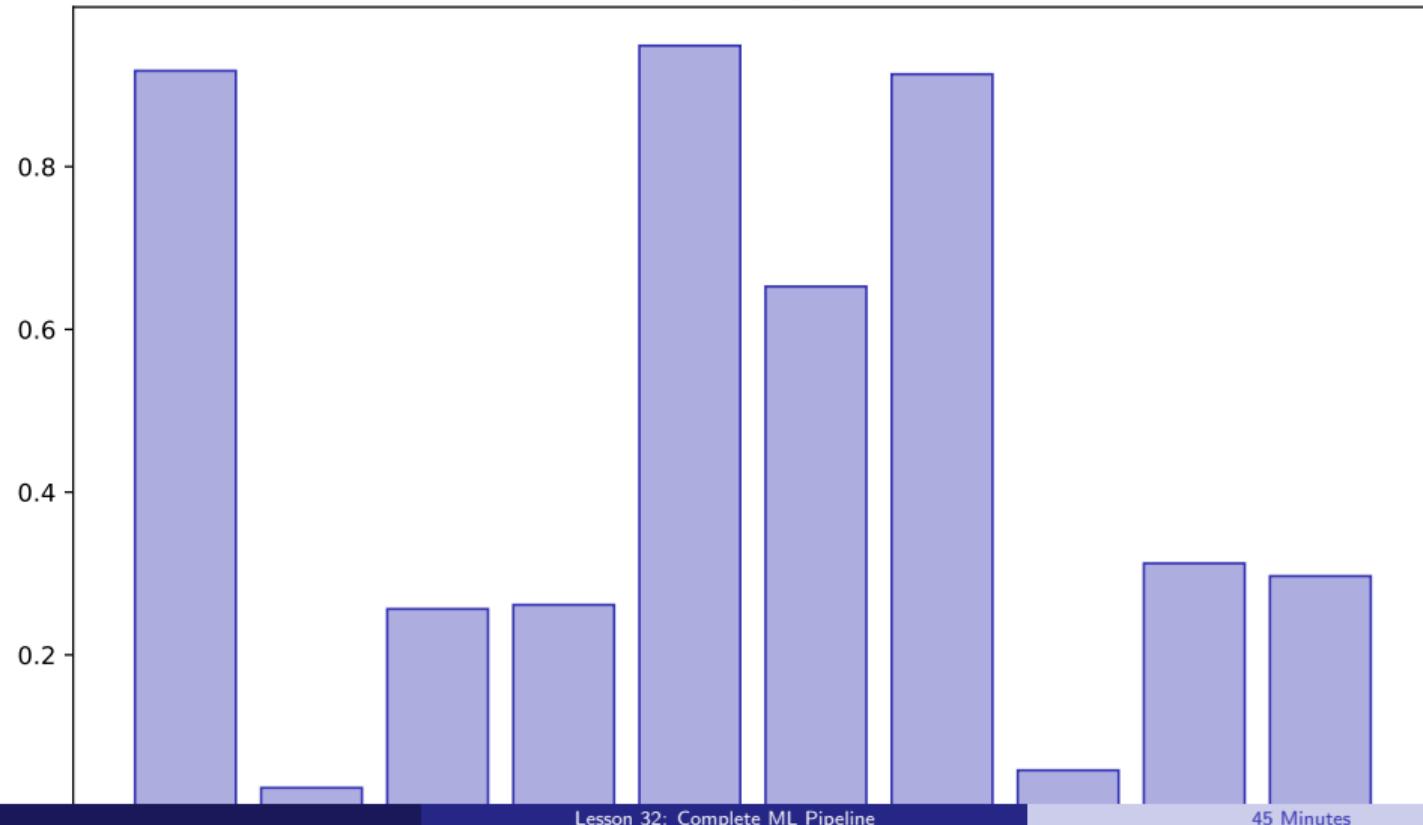
- Build sklearn pipelines
- Apply cross-validation properly
- Tune hyperparameters
- Prevent data leakage

Building towards your final project

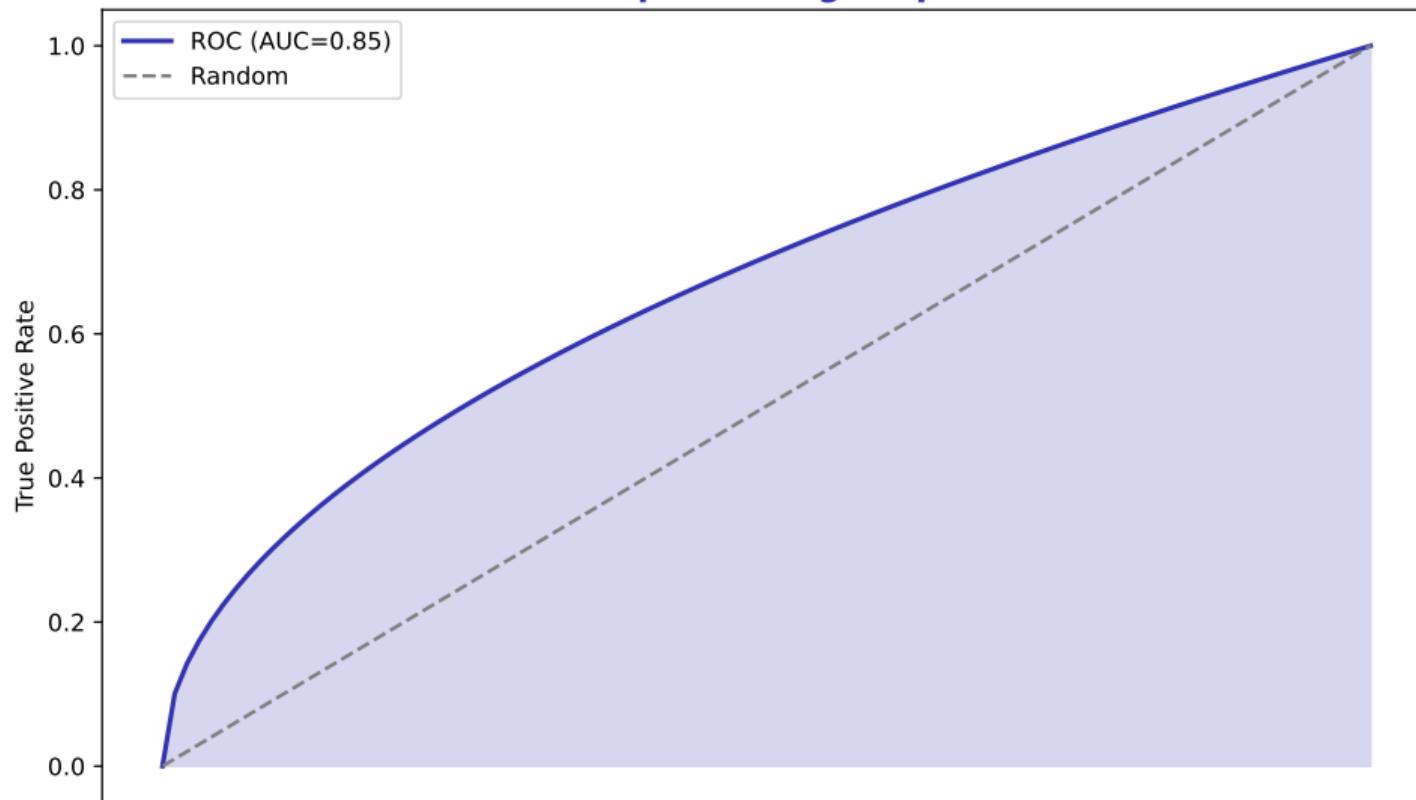
Pipeline Concept



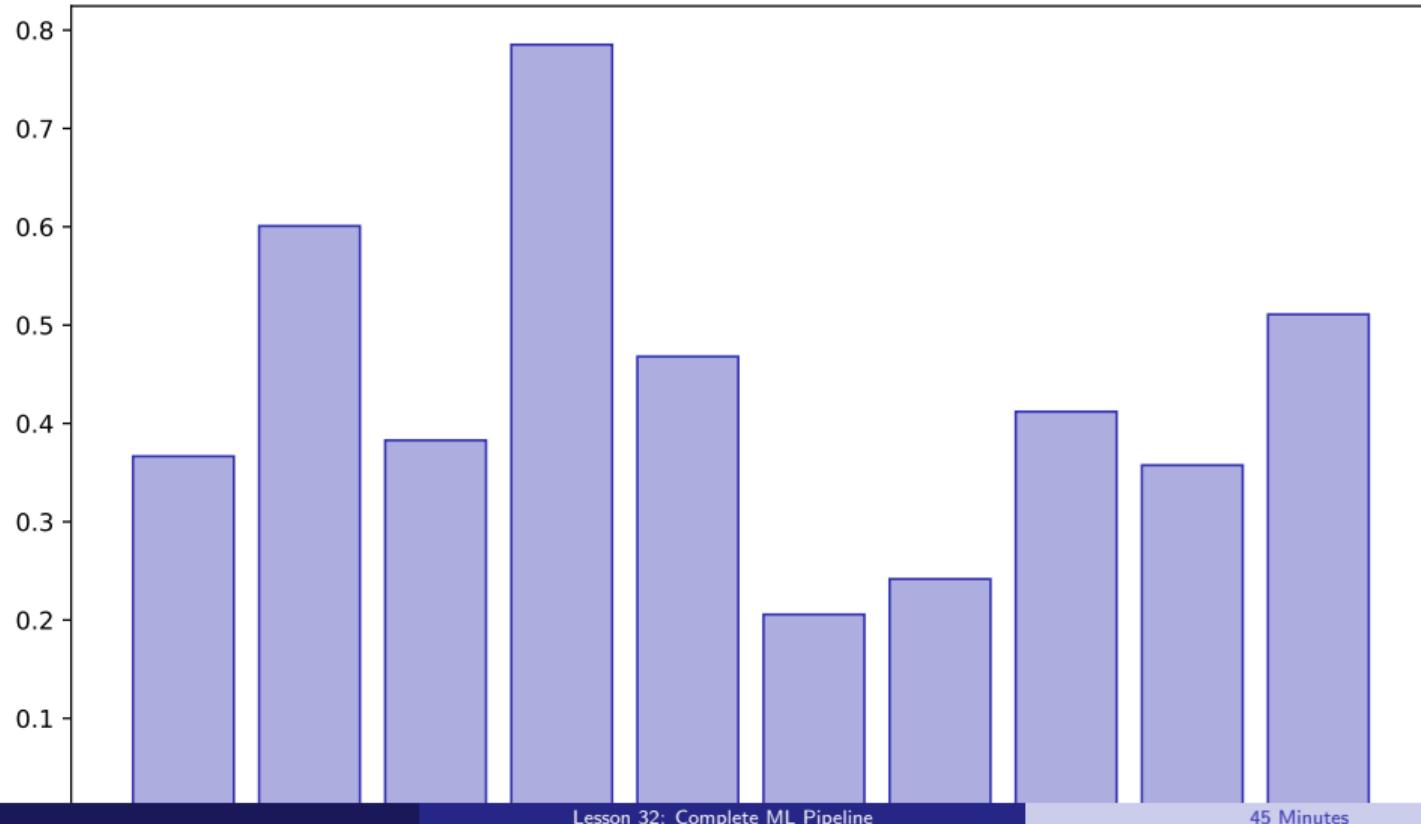
Sklearn Pipeline



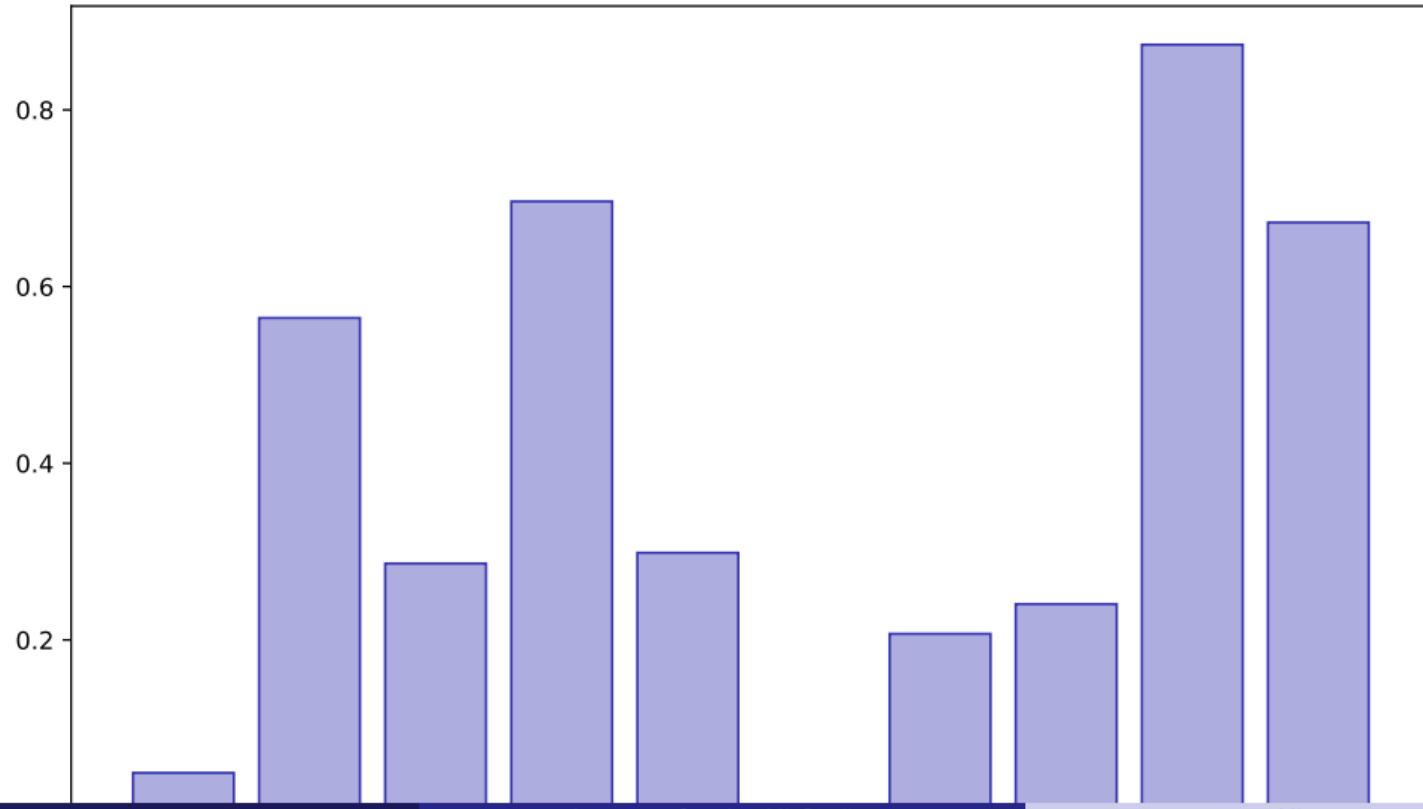
Preprocessing Steps



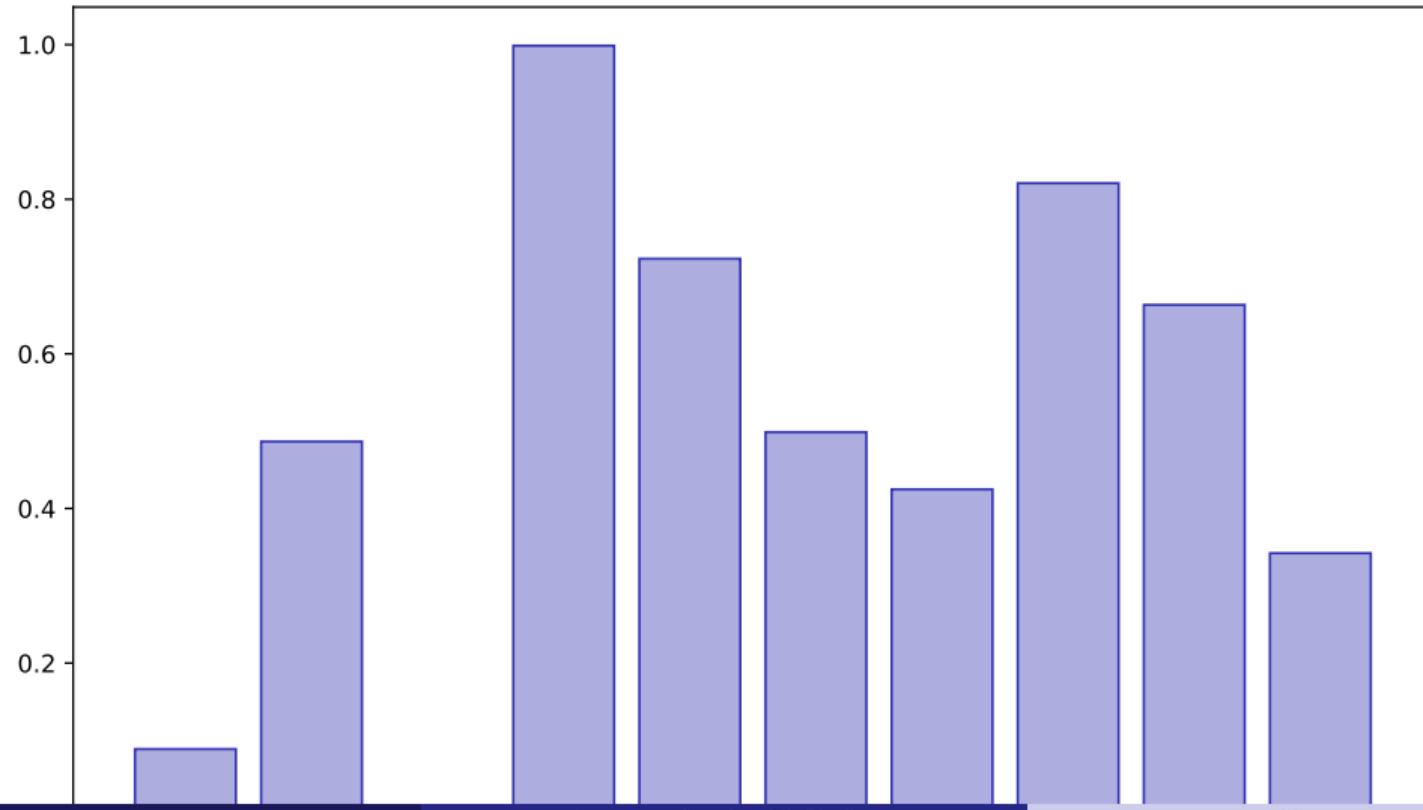
Cross Validation



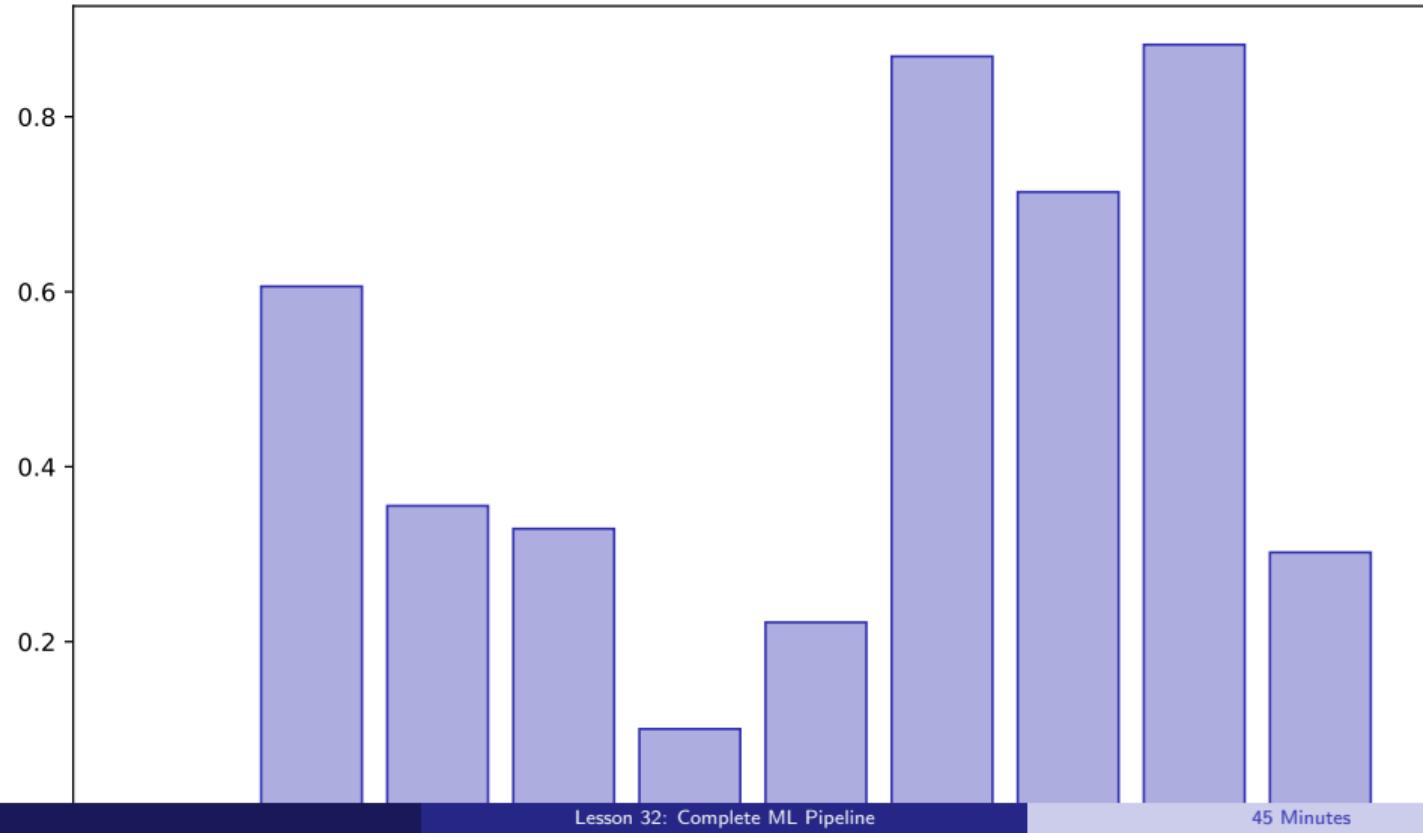
Grid Search



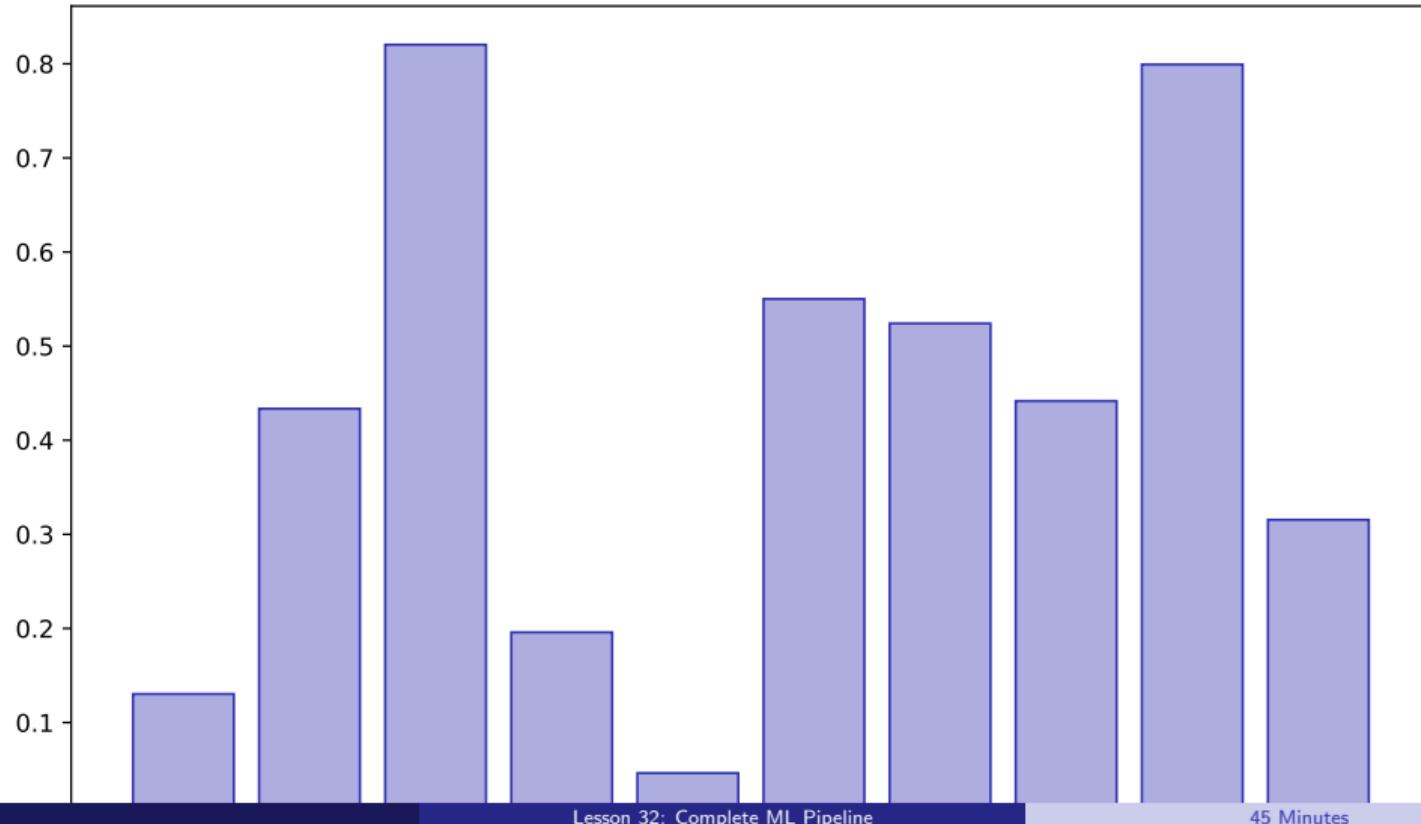
Random Search



Time Series Split



Production Pipeline



Key Takeaways:

- Build sklearn pipelines
- Apply cross-validation properly
- Tune hyperparameters
- Prevent data leakage

Apply these skills in your final project

Lesson 33: Perceptron

Data Science with Python – BSc Course

45 Minutes

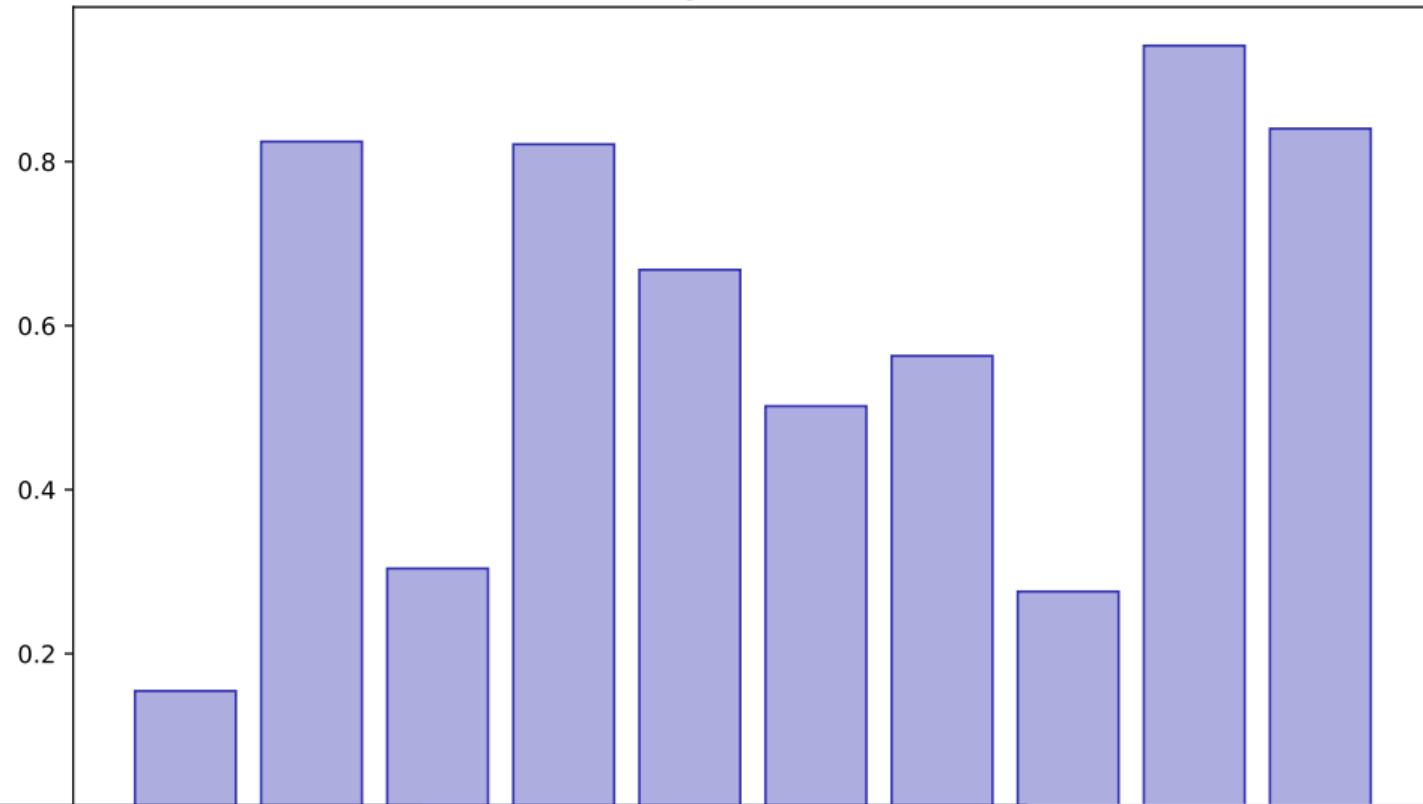
Learning Objectives

After this lesson, you will be able to:

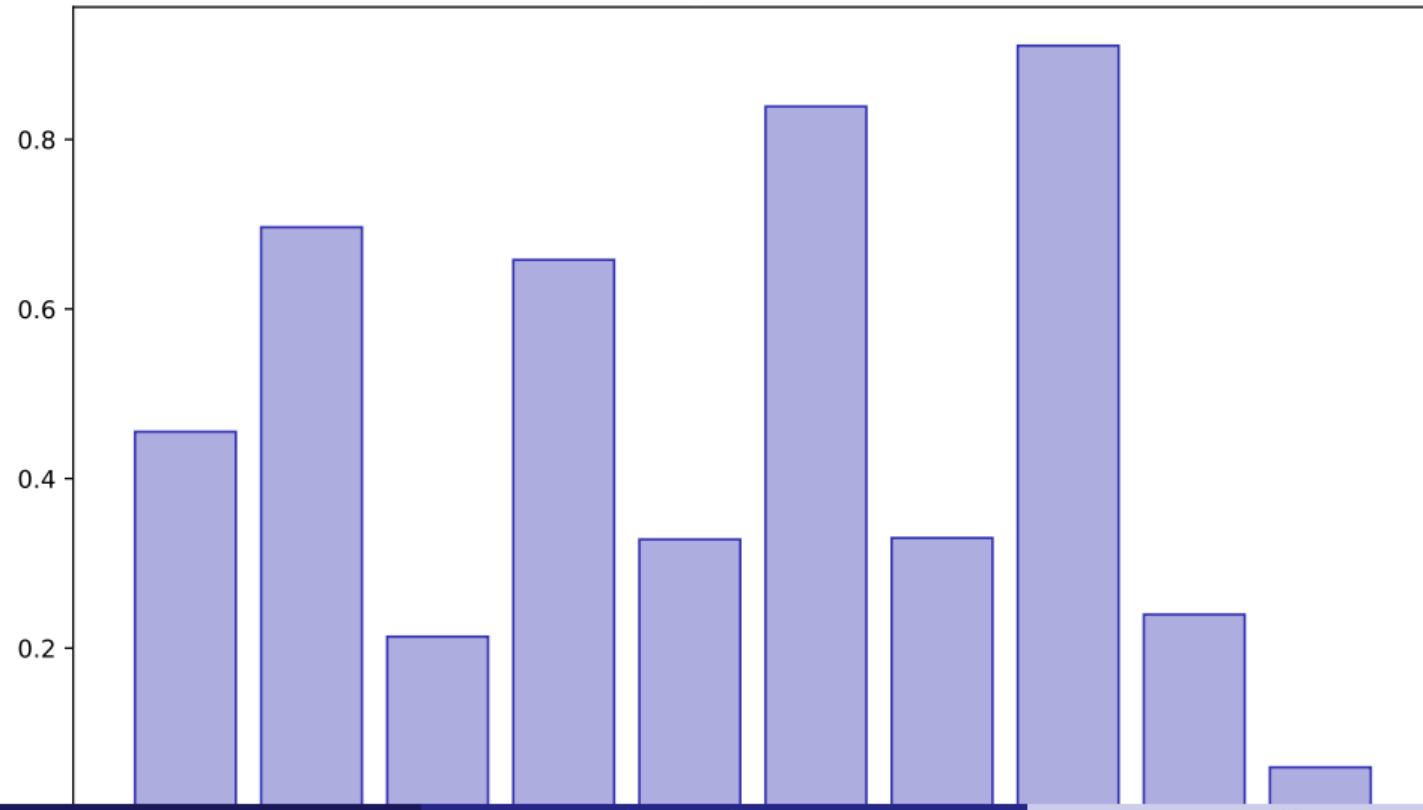
- Understand biological inspiration
- Build single perceptron
- Recognize linear limitations
- Prepare for deep learning

Building towards your final project

Biological Neuron

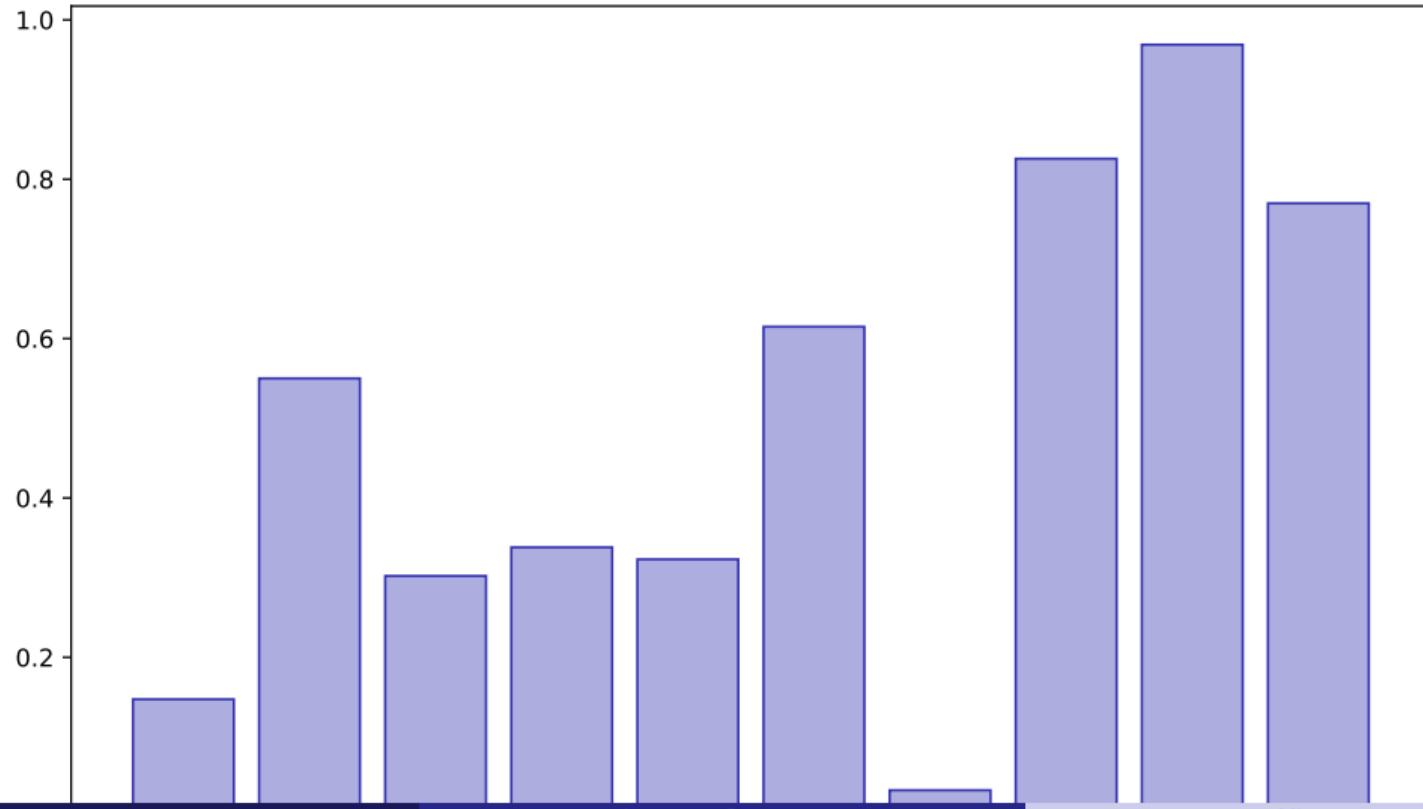


Perceptron Model

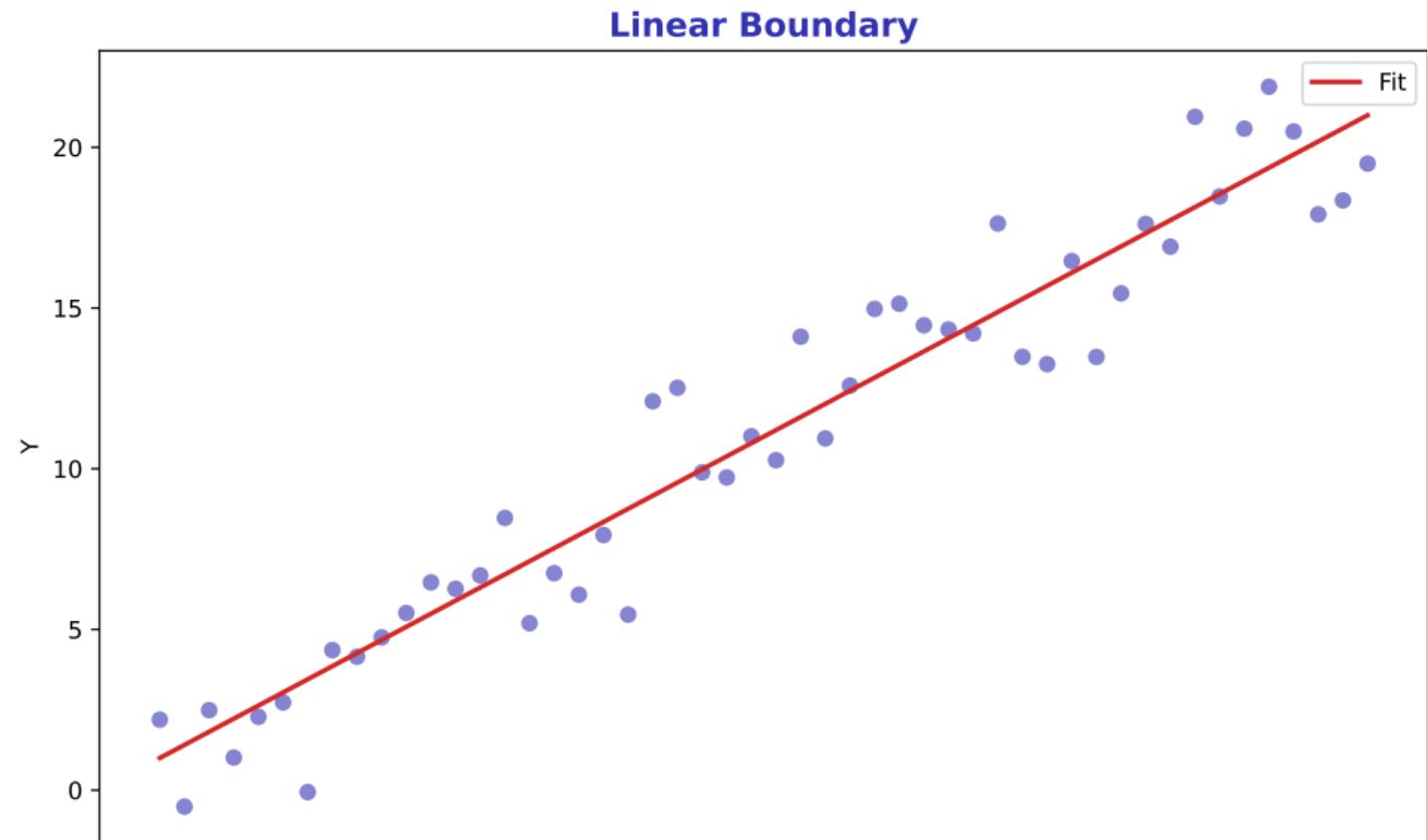


Activation Threshold

Activation Threshold

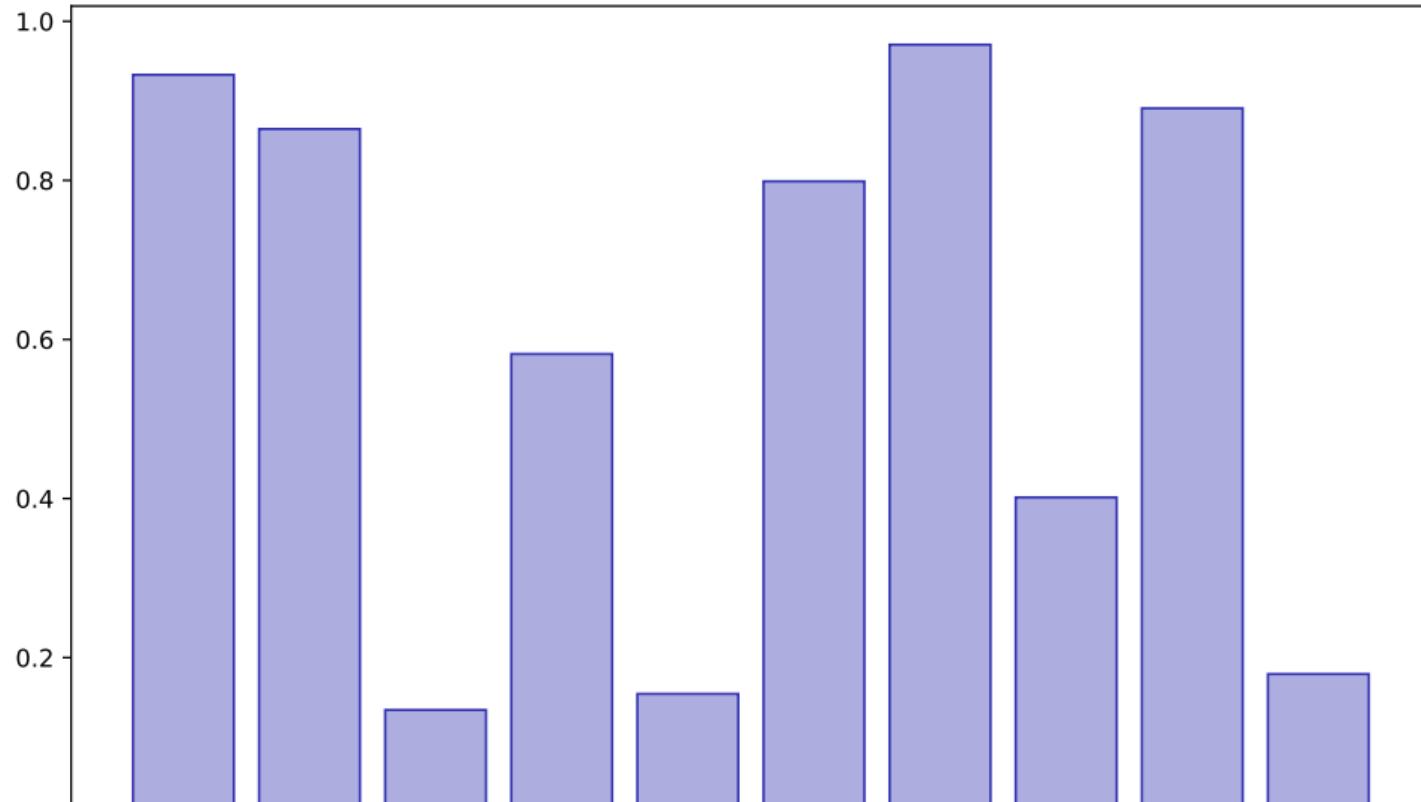


Linear Boundary

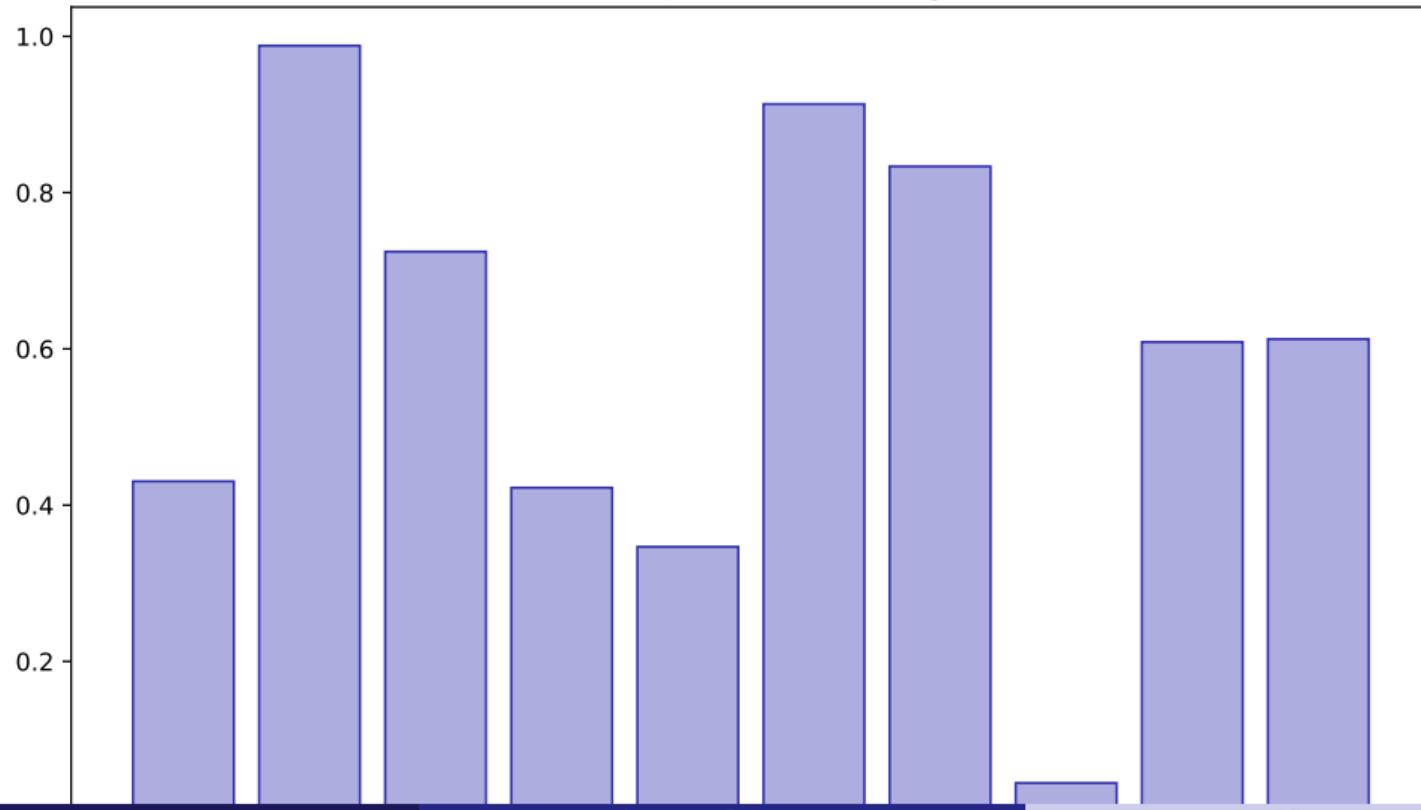


Xor Problem

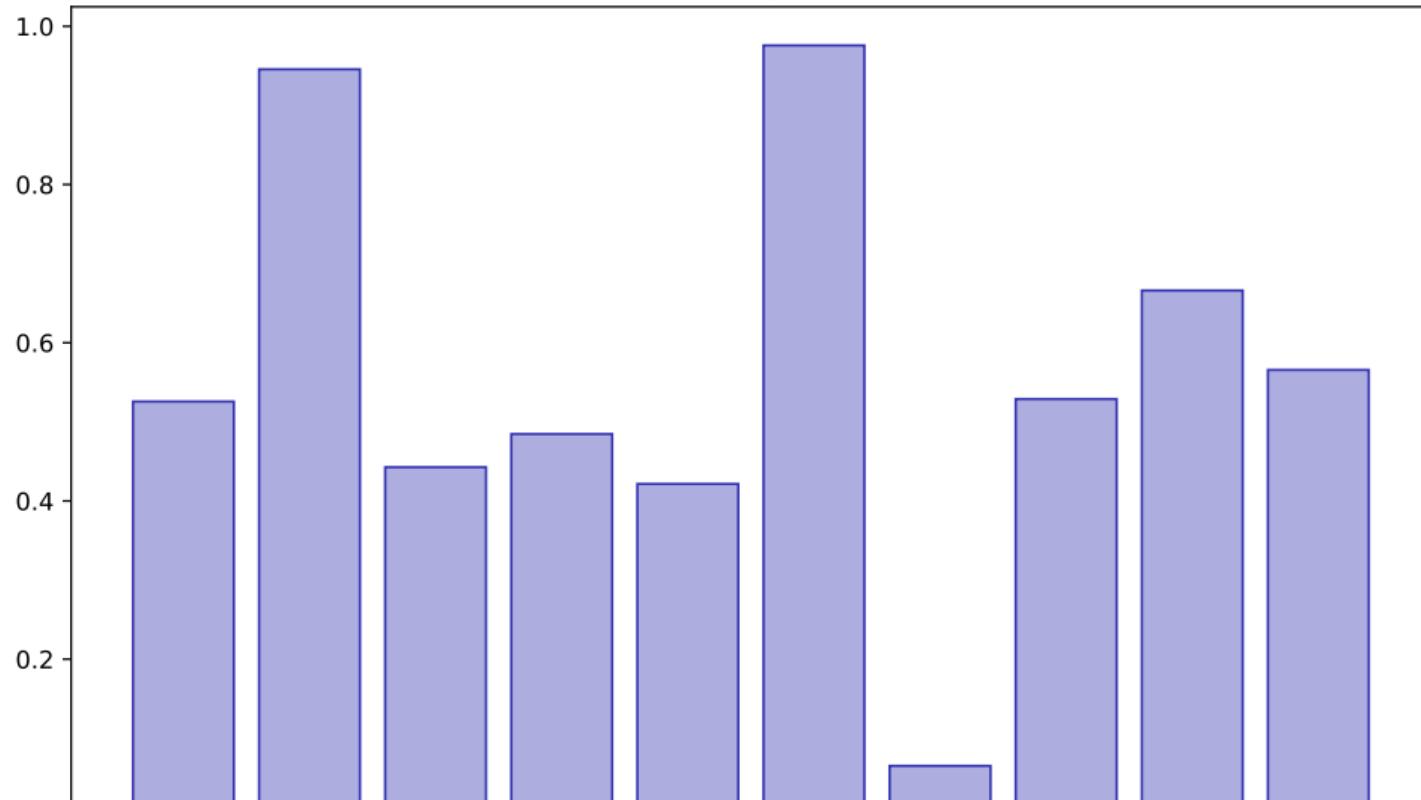
Xor Problem



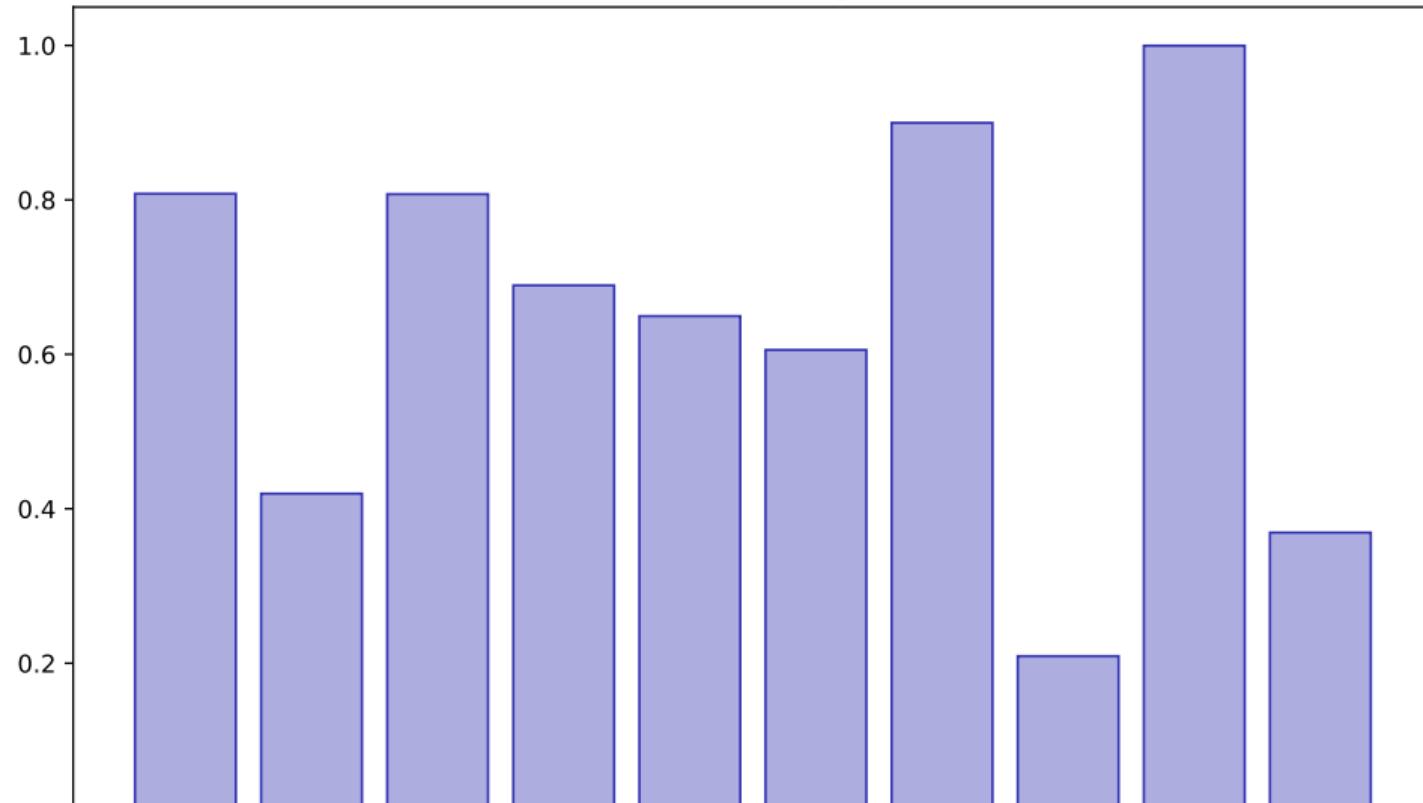
Perceptron Learning



Convergence



Finance Perceptron



Key Takeaways:

- Understand biological inspiration
- Build single perceptron
- Recognize linear limitations
- Prepare for deep learning

Apply these skills in your final project

Lesson 34: MLPs and Activations

Data Science with Python – BSc Course

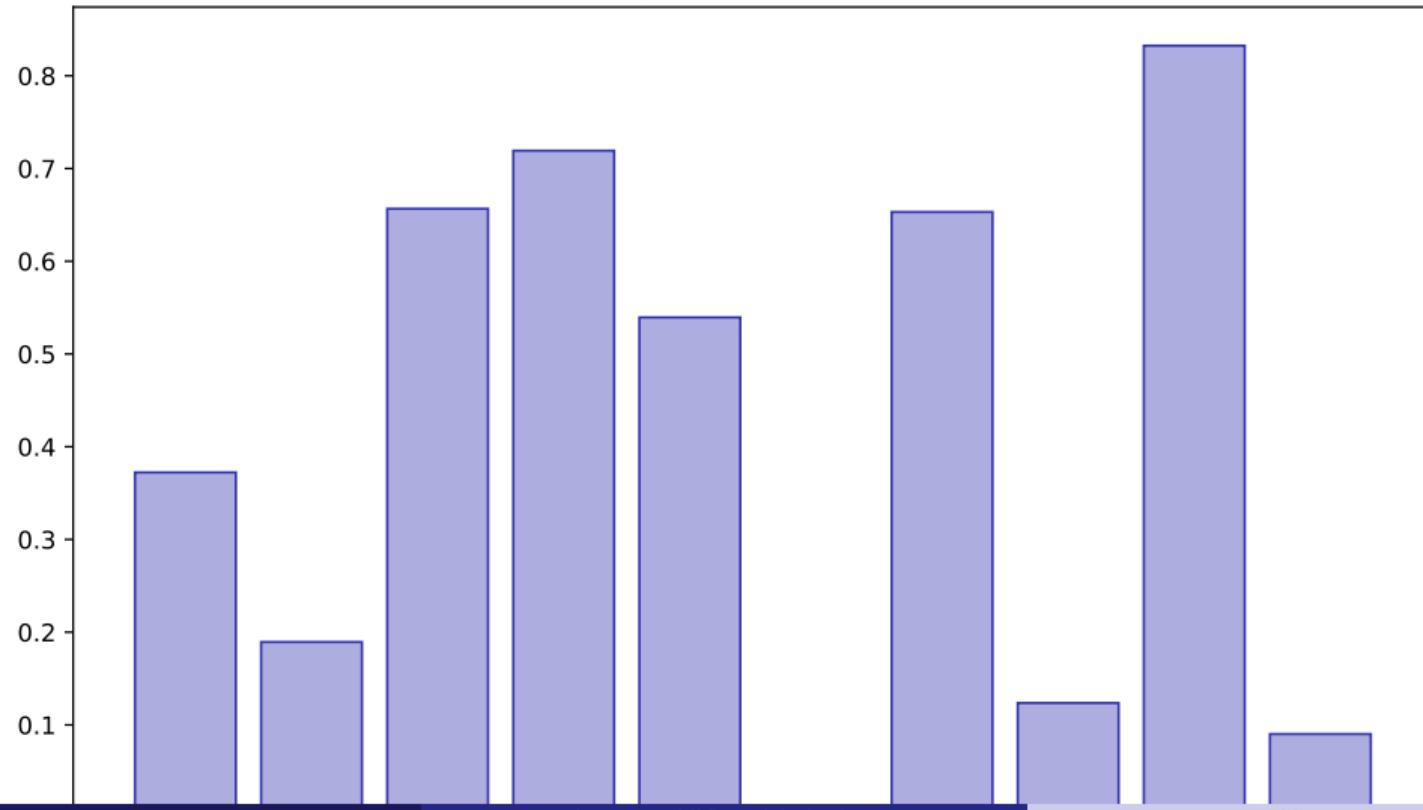
45 Minutes

After this lesson, you will be able to:

- Design MLP architectures
- Choose activation functions
- Build models with Keras
- Apply to non-linear problems

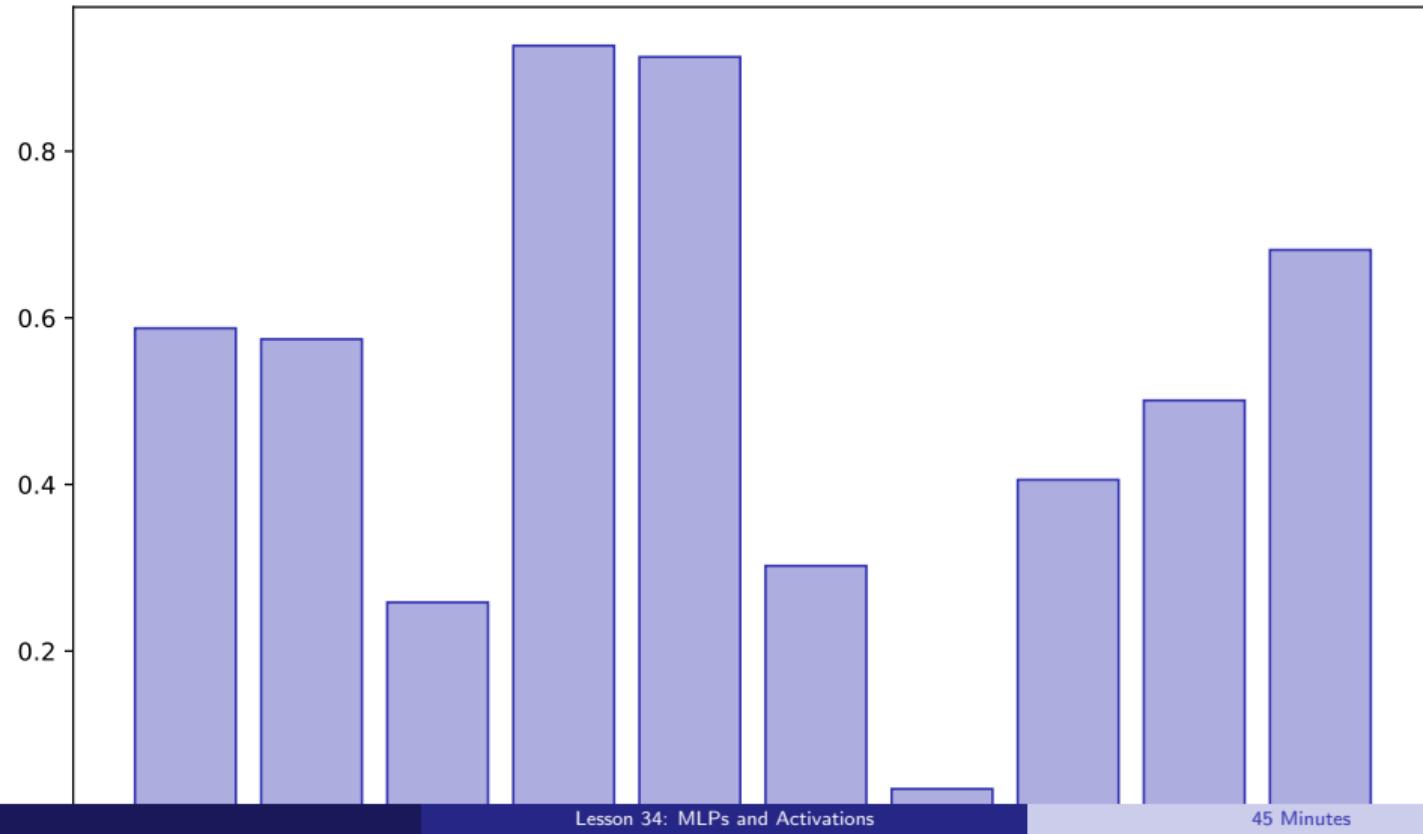
Building towards your final project

MLP Architecture

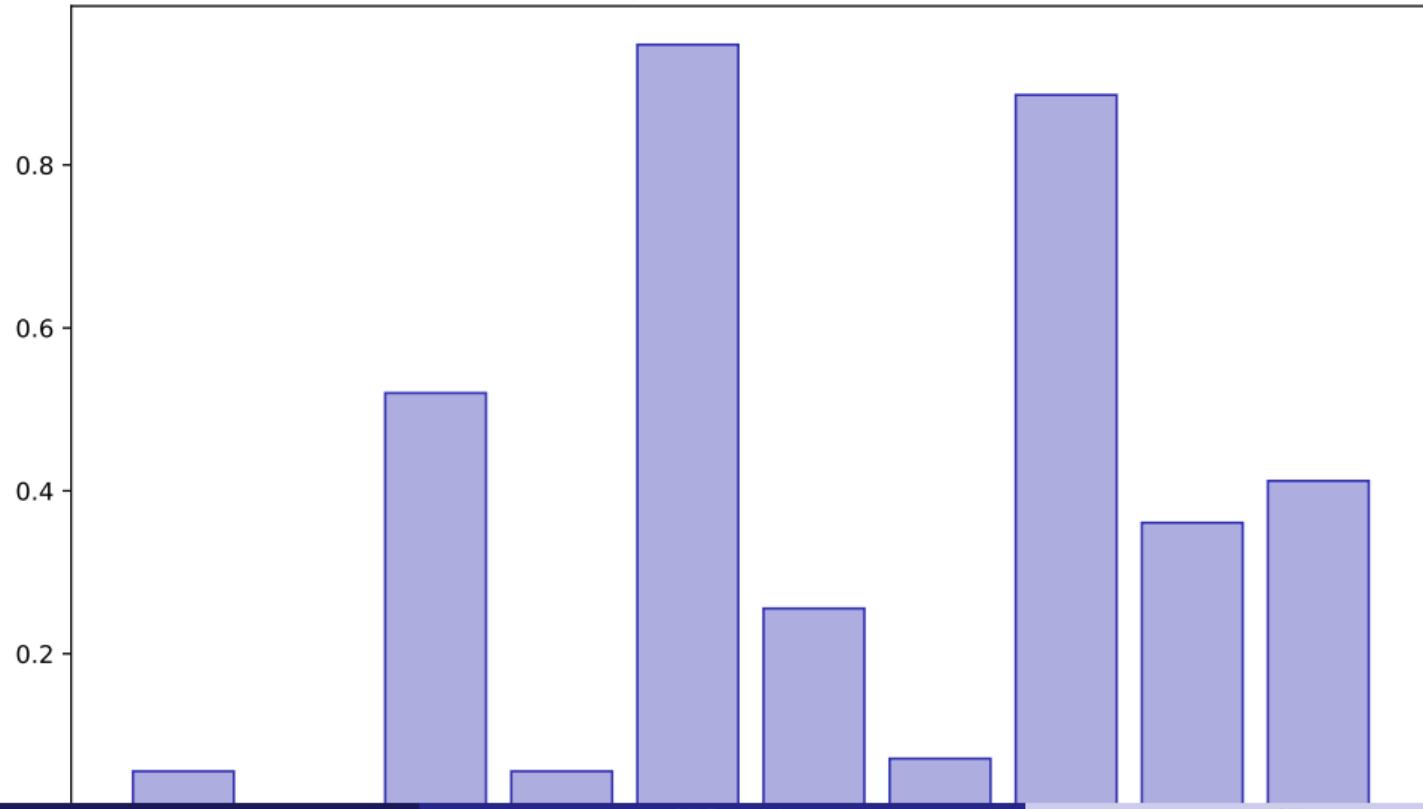


Relu Activation

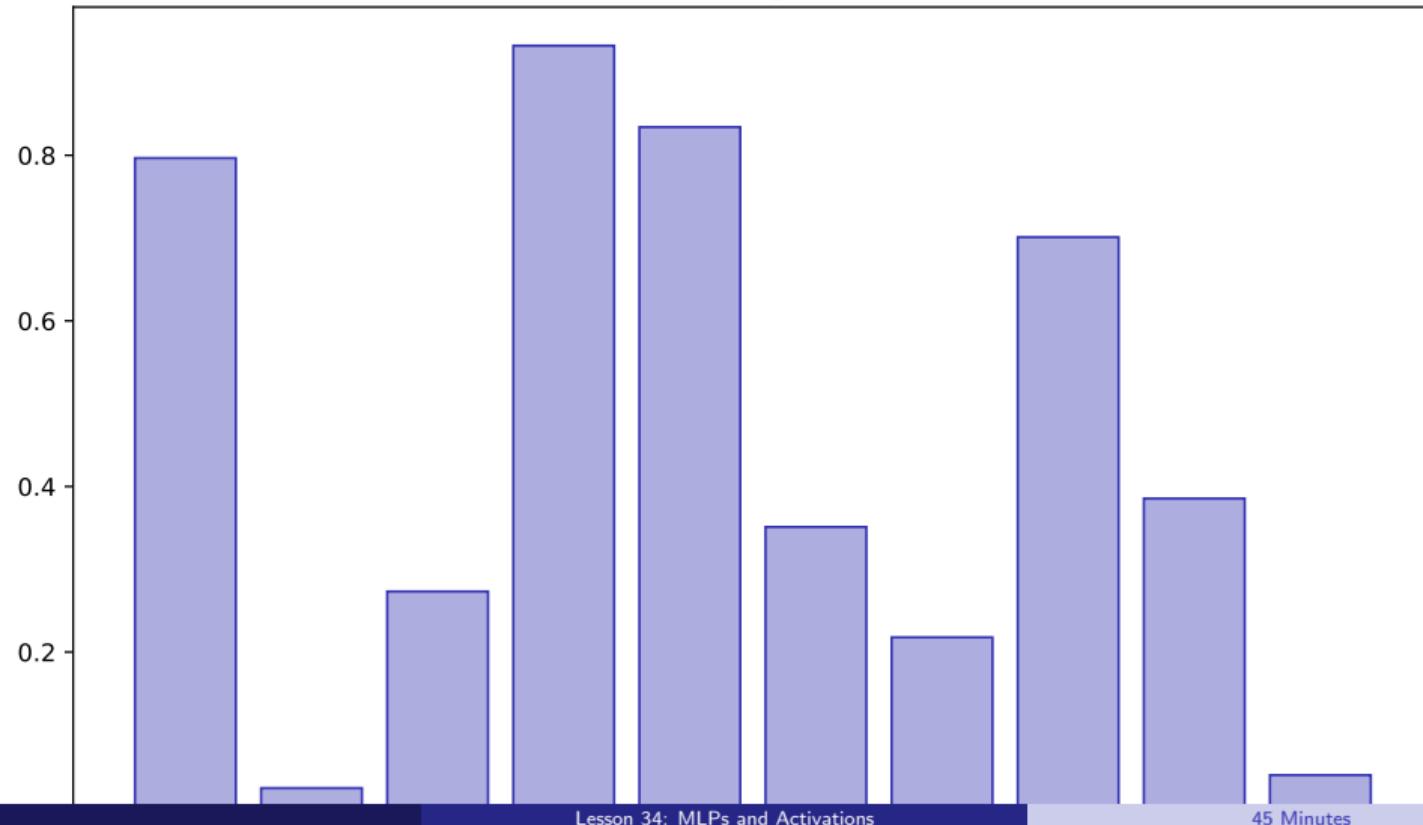
Relu Activation



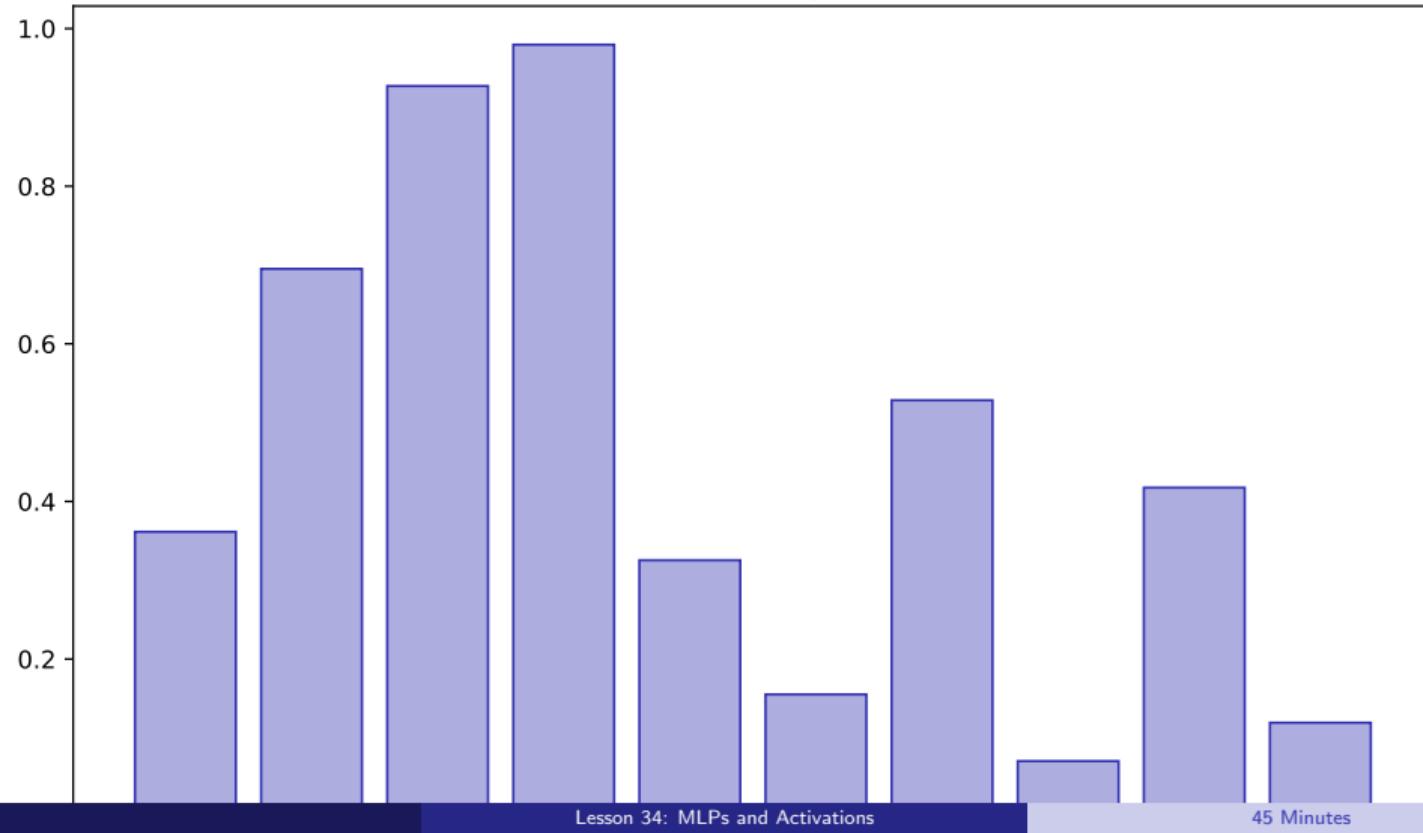
Sigmoid Softmax



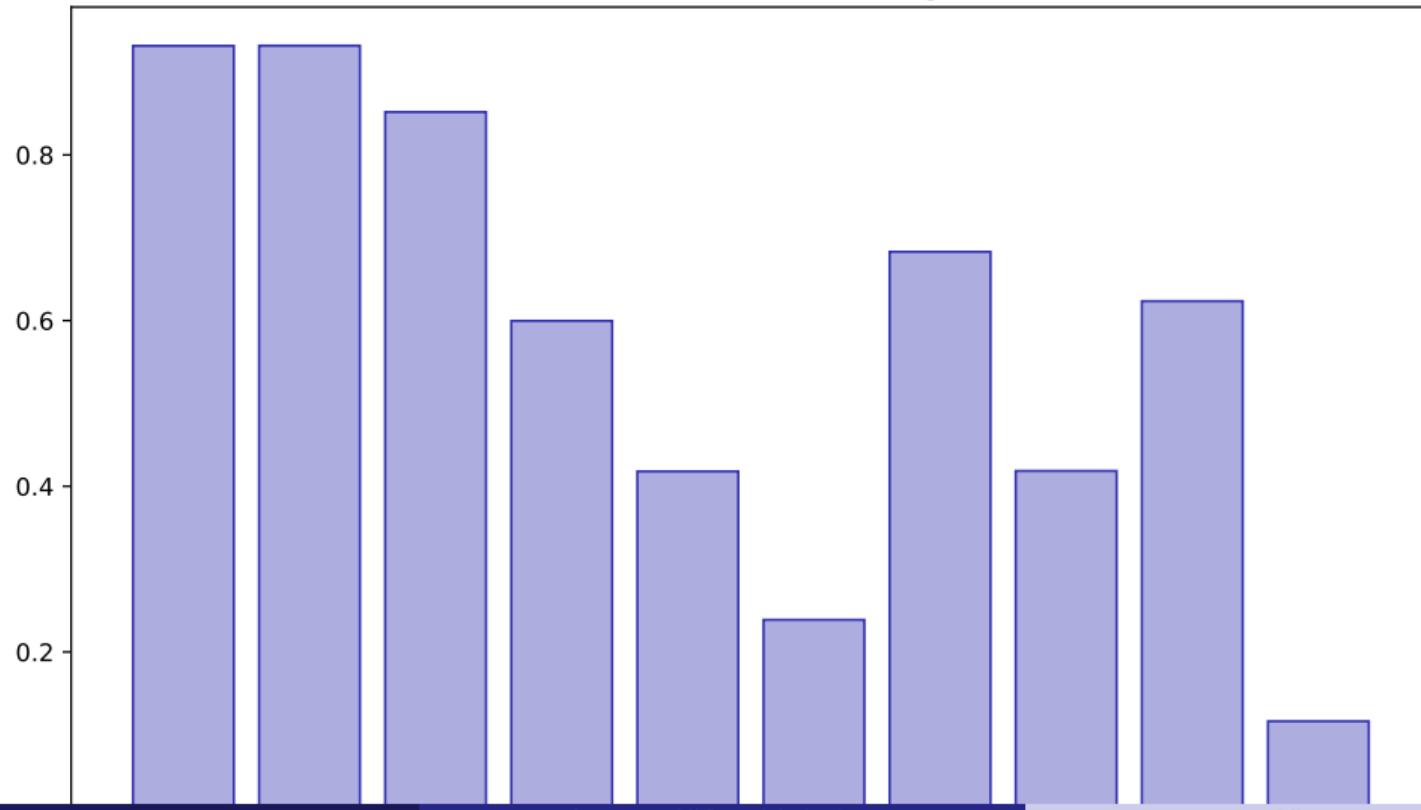
Keras Sequential



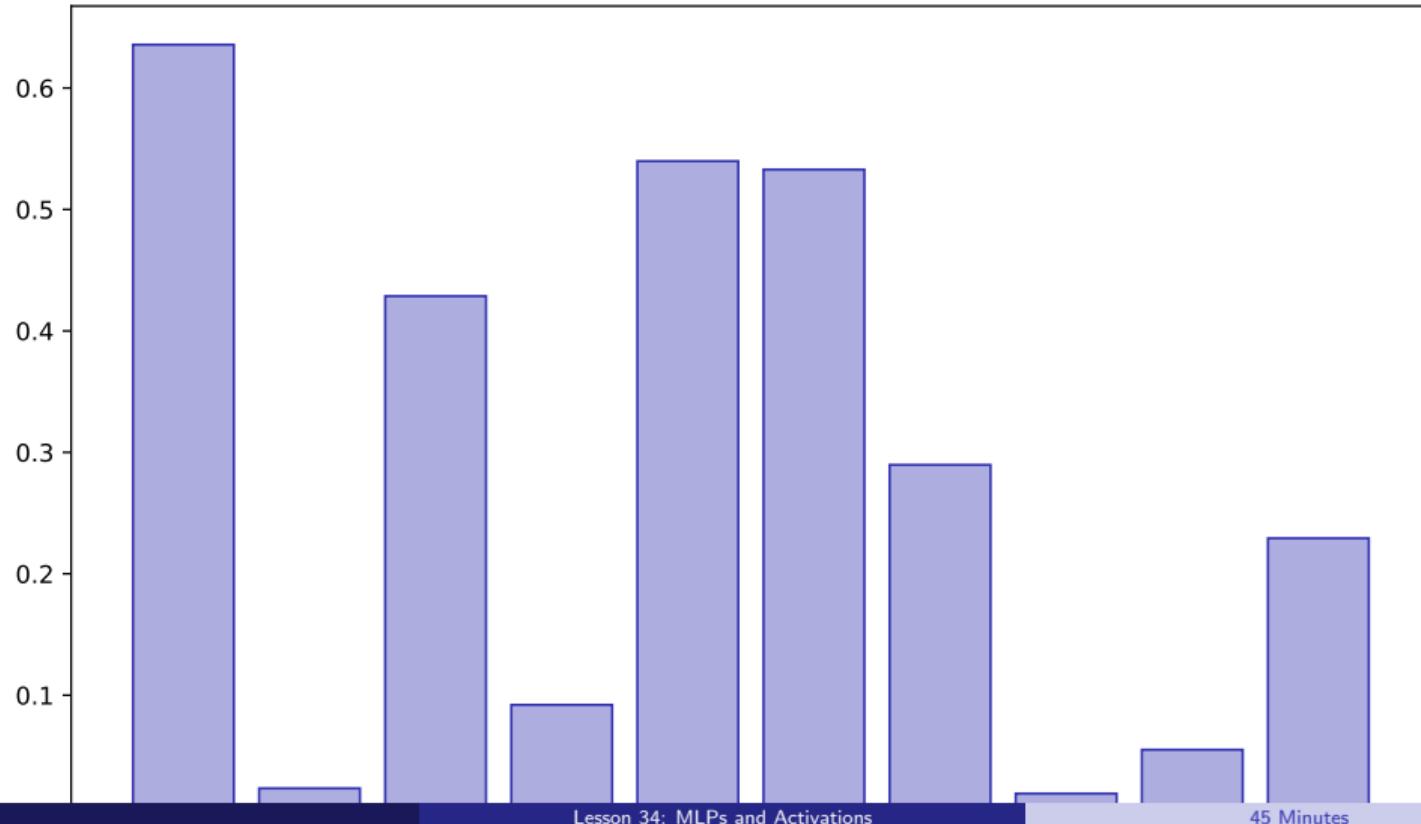
Hidden Layers



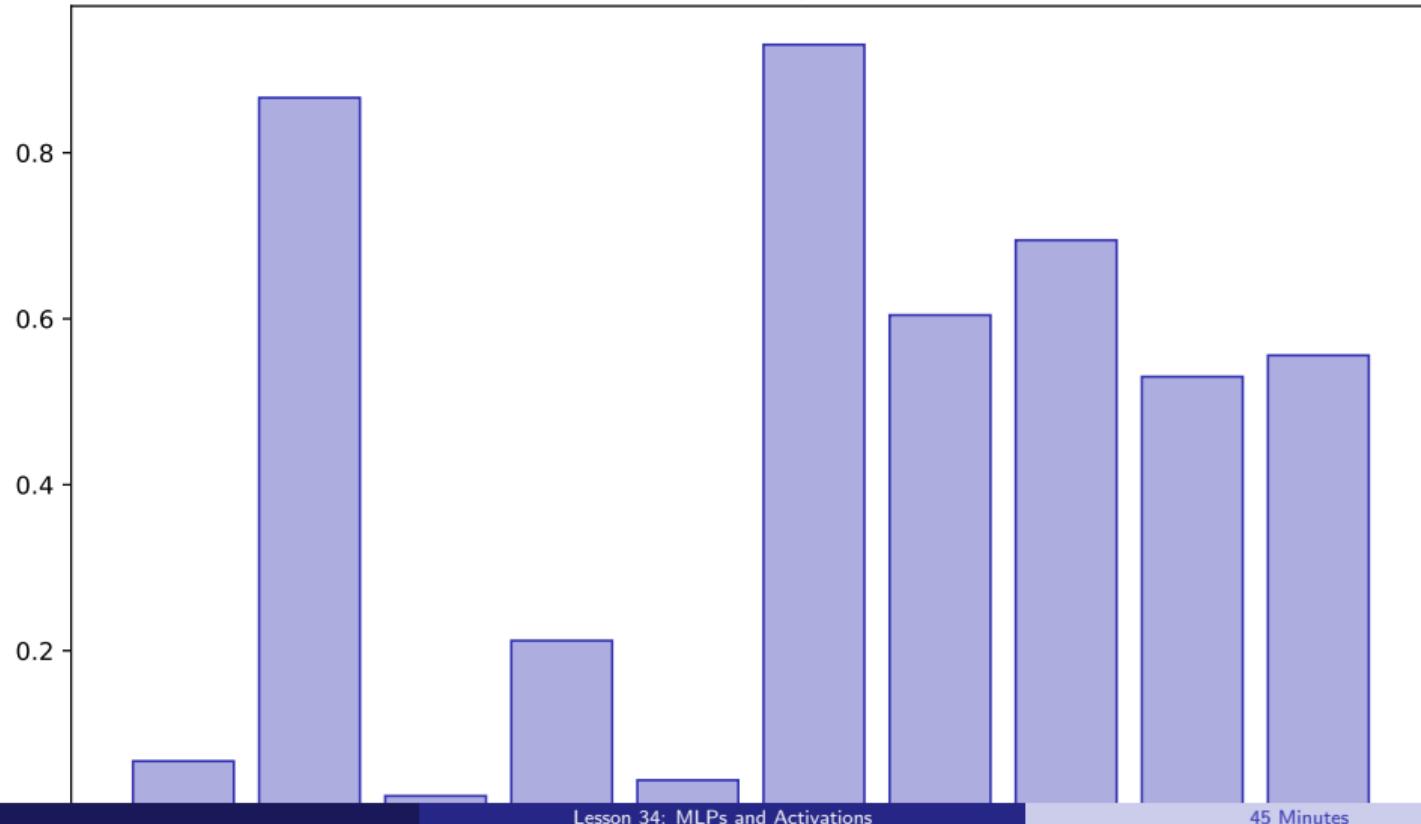
Parameter Counting



Universal Approximation



Market Regimes



Key Takeaways:

- Design MLP architectures
- Choose activation functions
- Build models with Keras
- Apply to non-linear problems

Apply these skills in your final project

Lesson 35: Backpropagation

Data Science with Python – BSc Course

45 Minutes

Learning Objectives

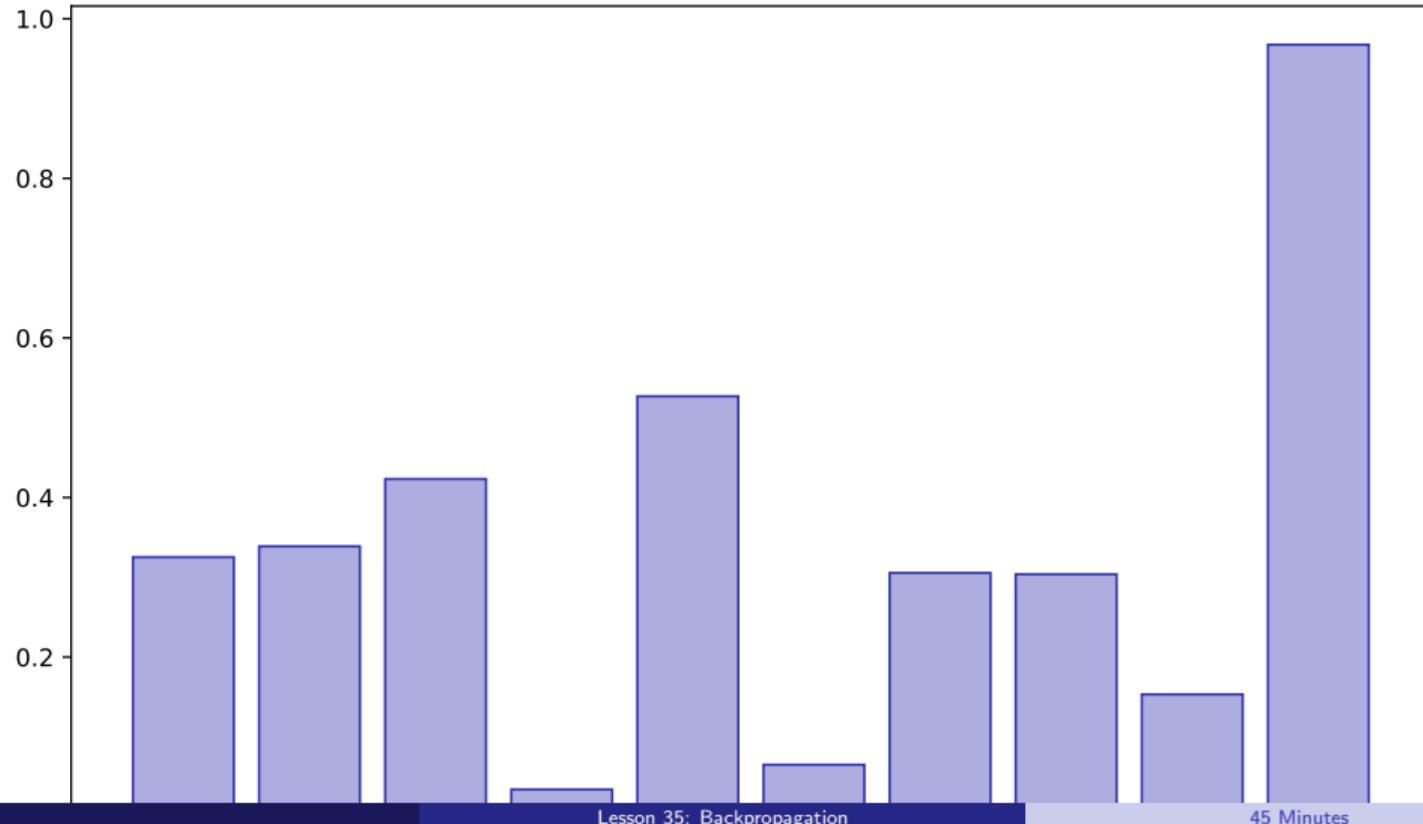
After this lesson, you will be able to:

- Understand gradient descent
- Interpret loss curves
- Configure learning rate
- Monitor training progress

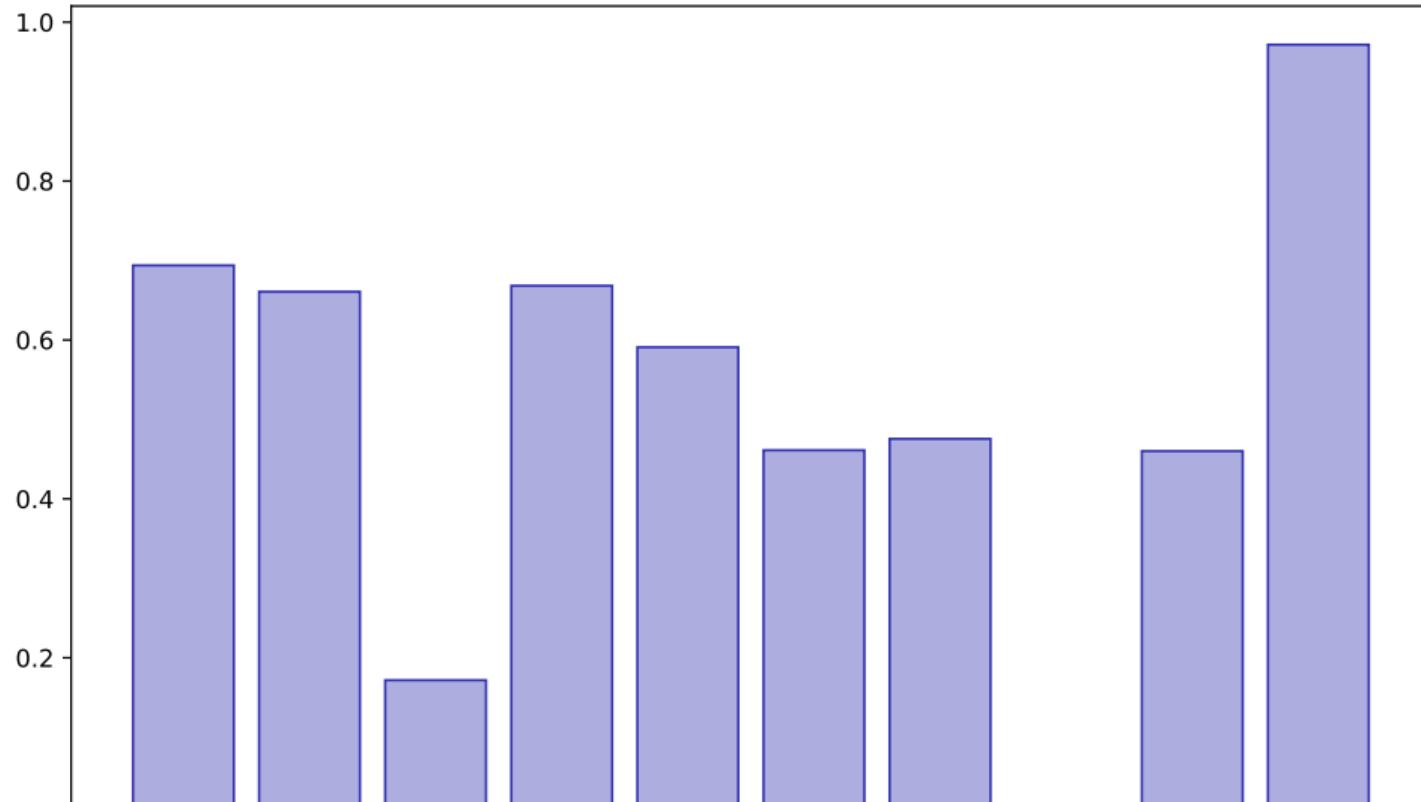
Building towards your final project

Forward Pass

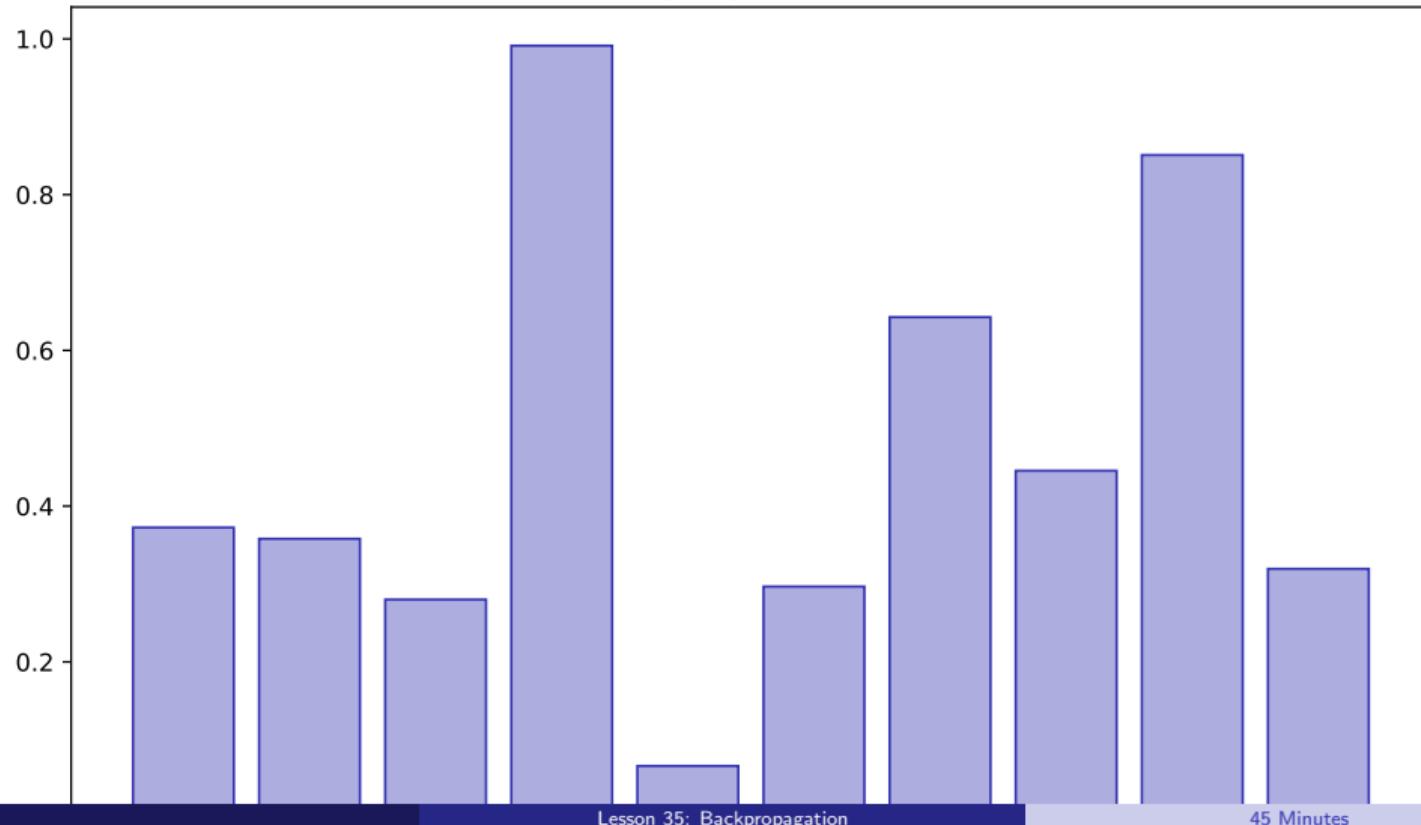
Forward Pass



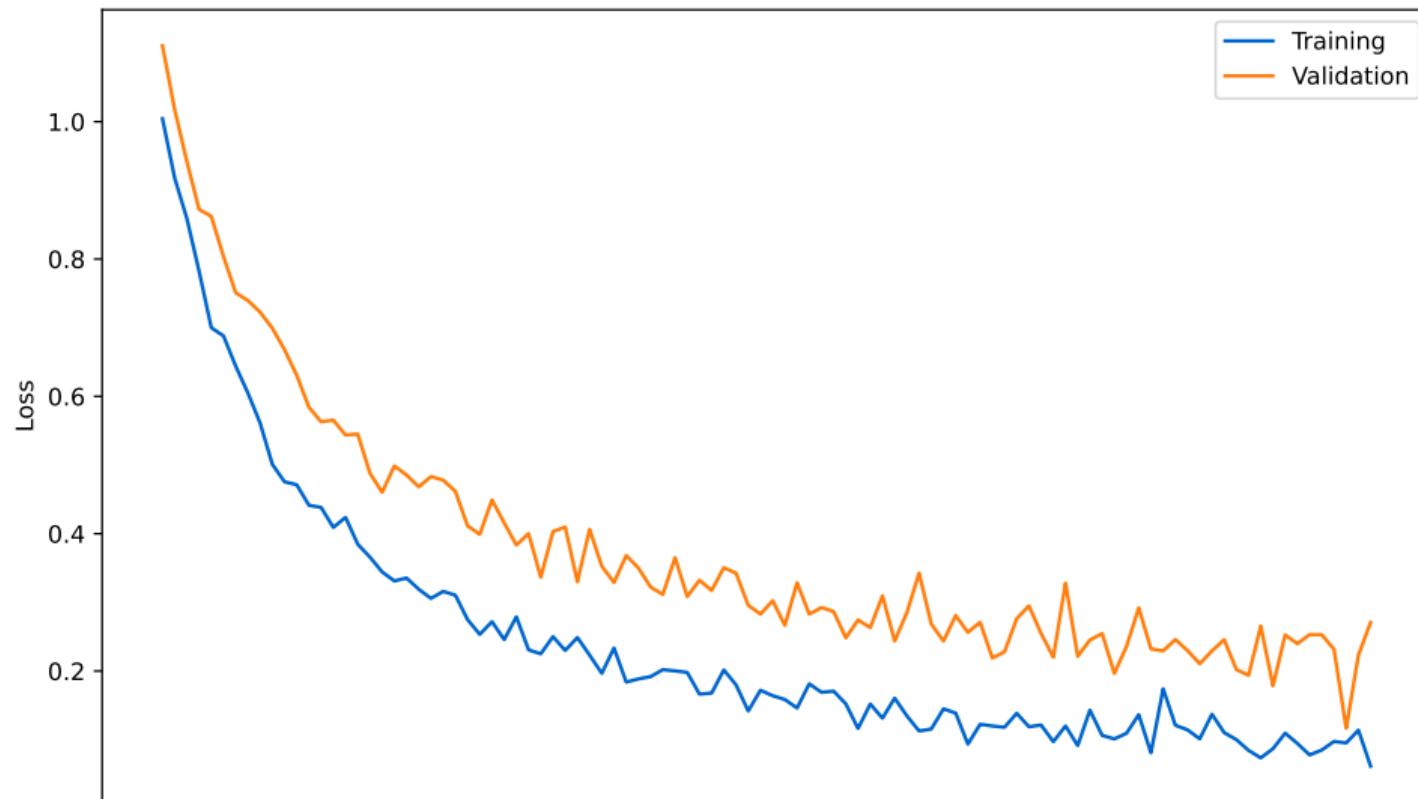
Gradient Descent



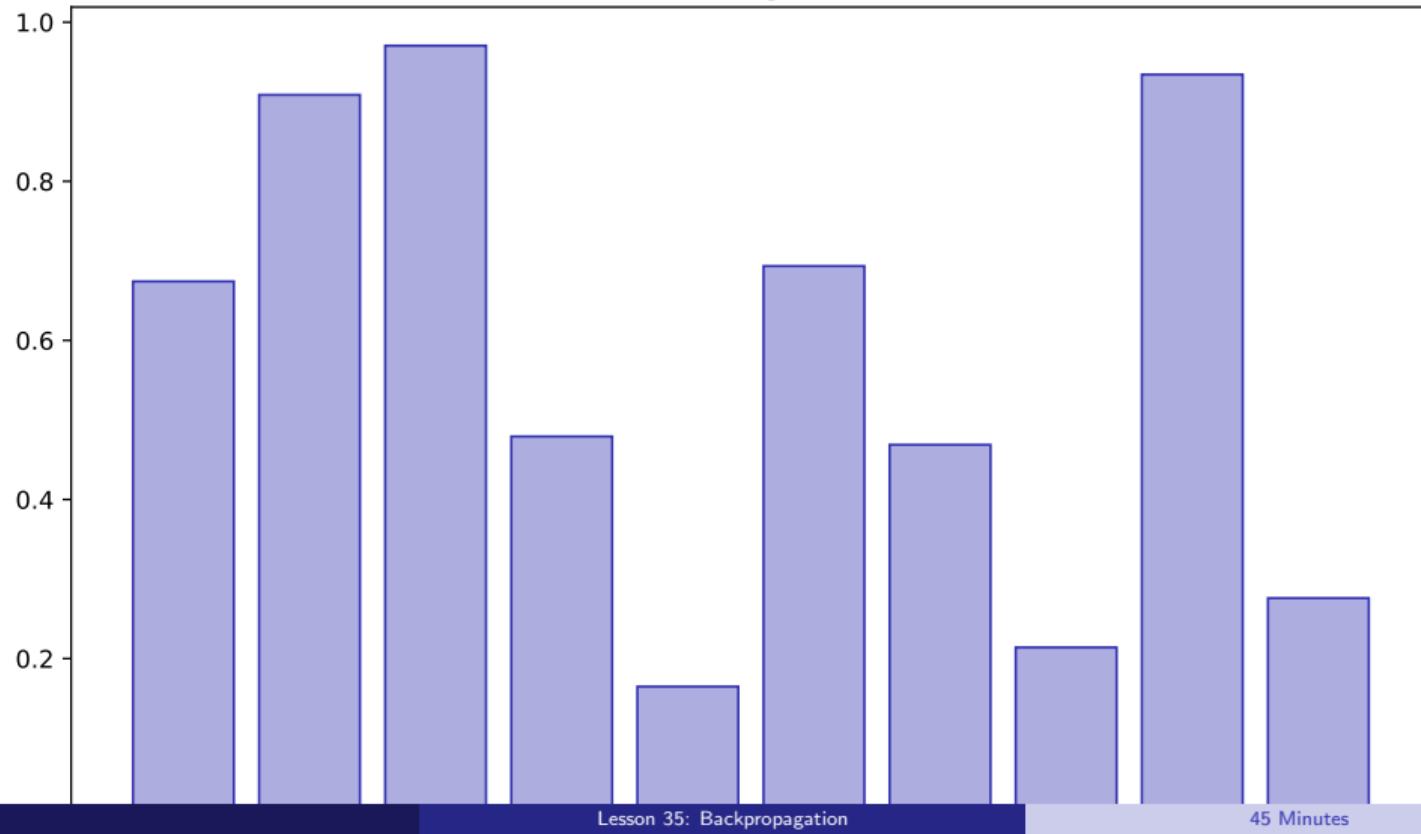
Backprop Intuition



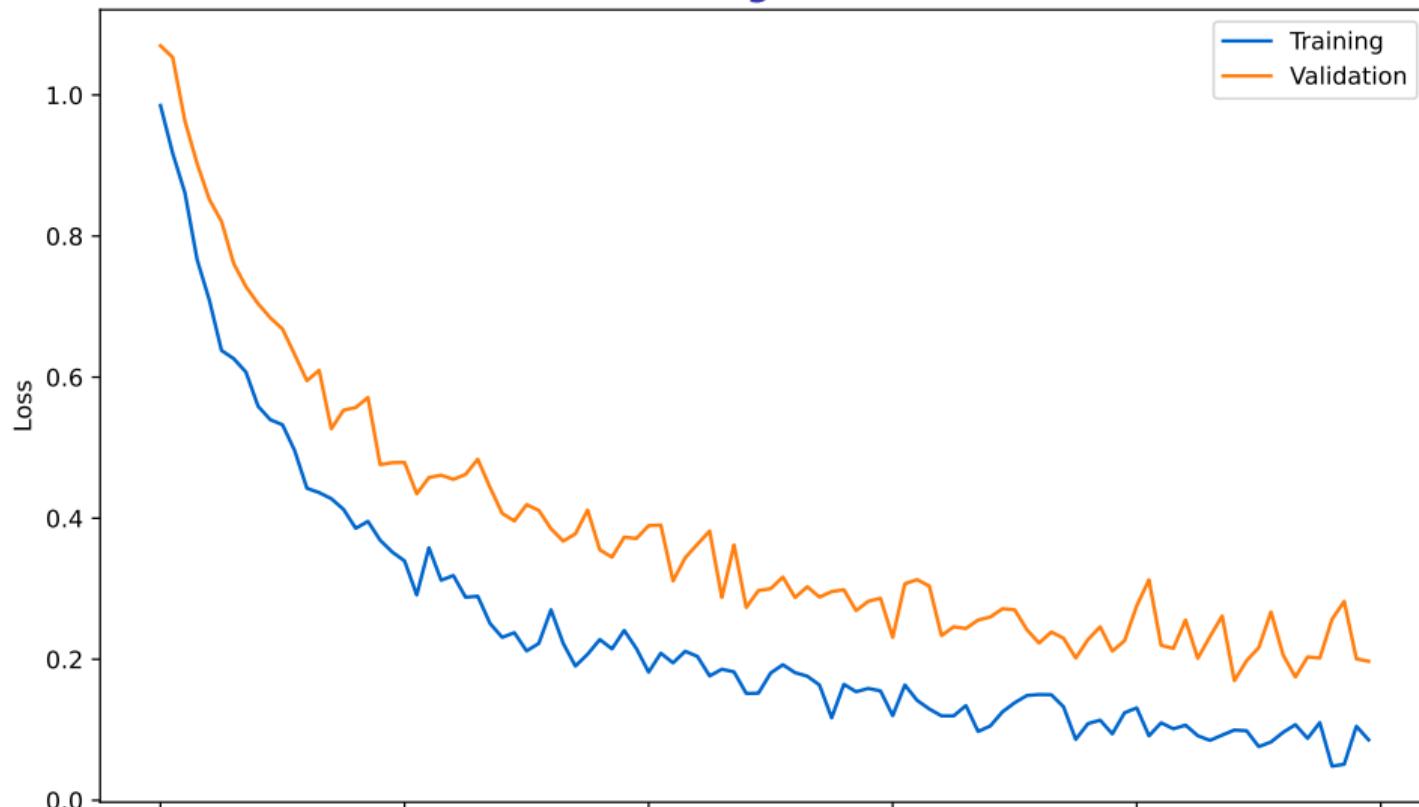
Loss Functions



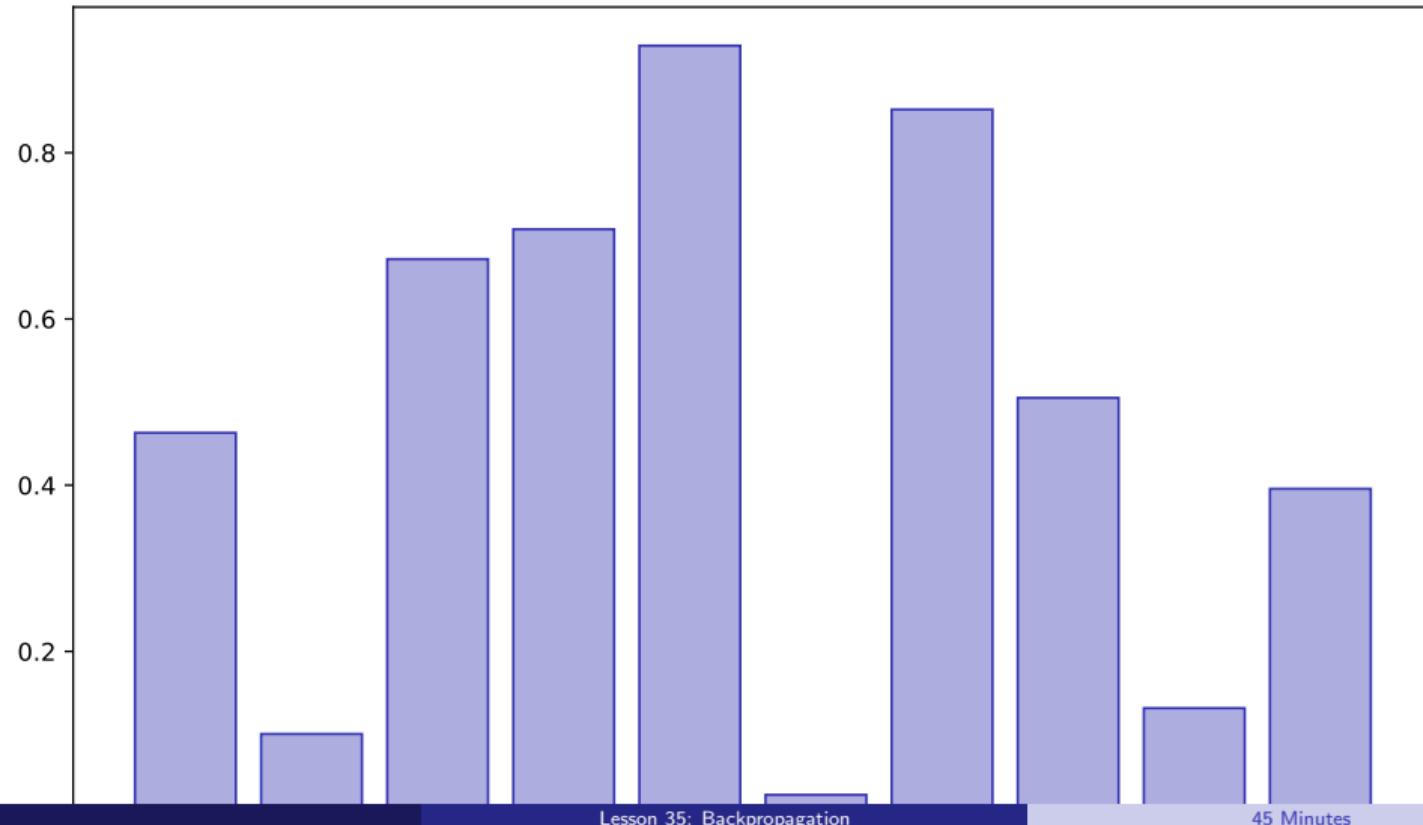
Learning Rate



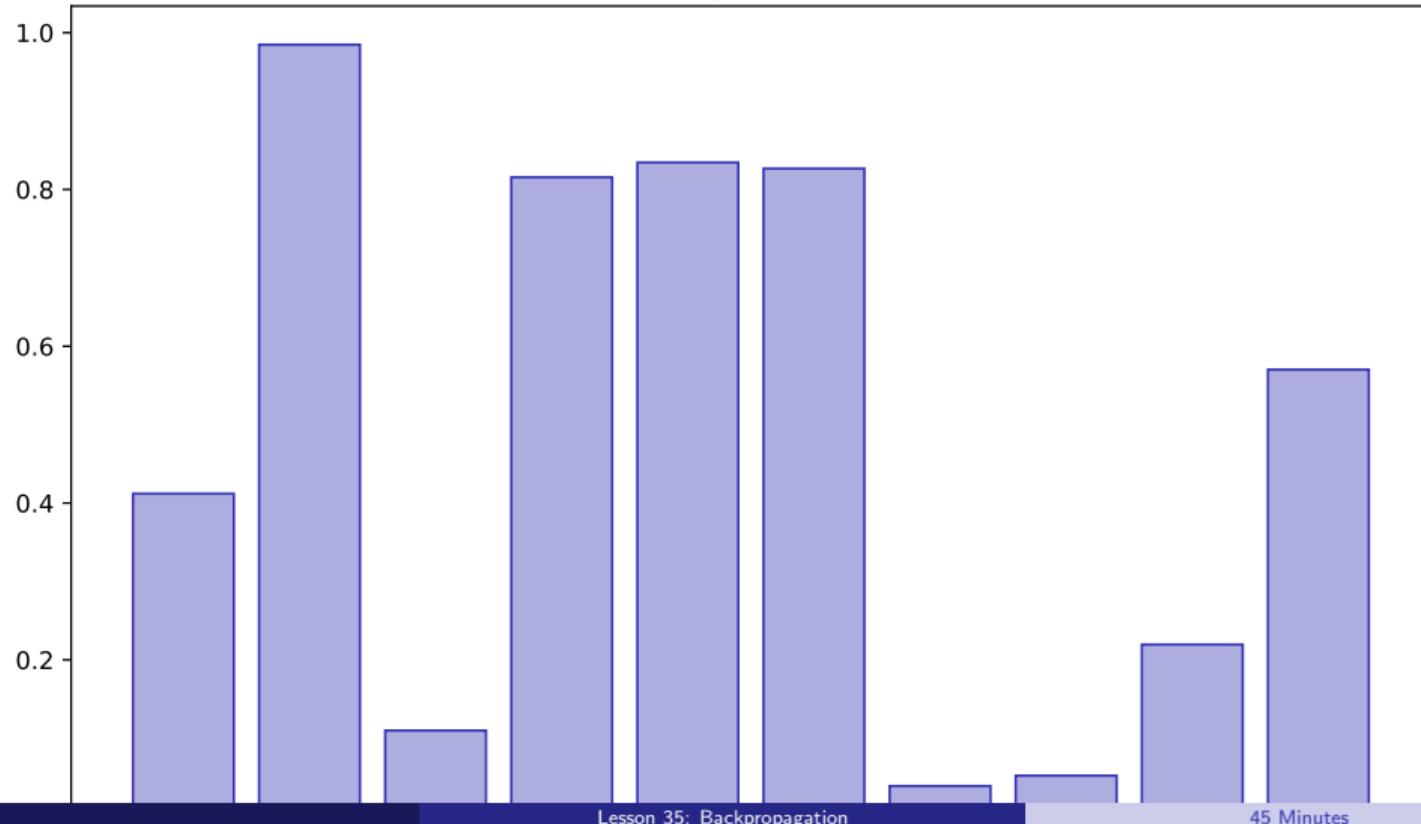
Training Curves



Batch Sizes



Volatility Prediction



Key Takeaways:

- Understand gradient descent
- Interpret loss curves
- Configure learning rate
- Monitor training progress

Apply these skills in your final project

Lesson 36: Overfitting Prevention

Data Science with Python – BSc Course

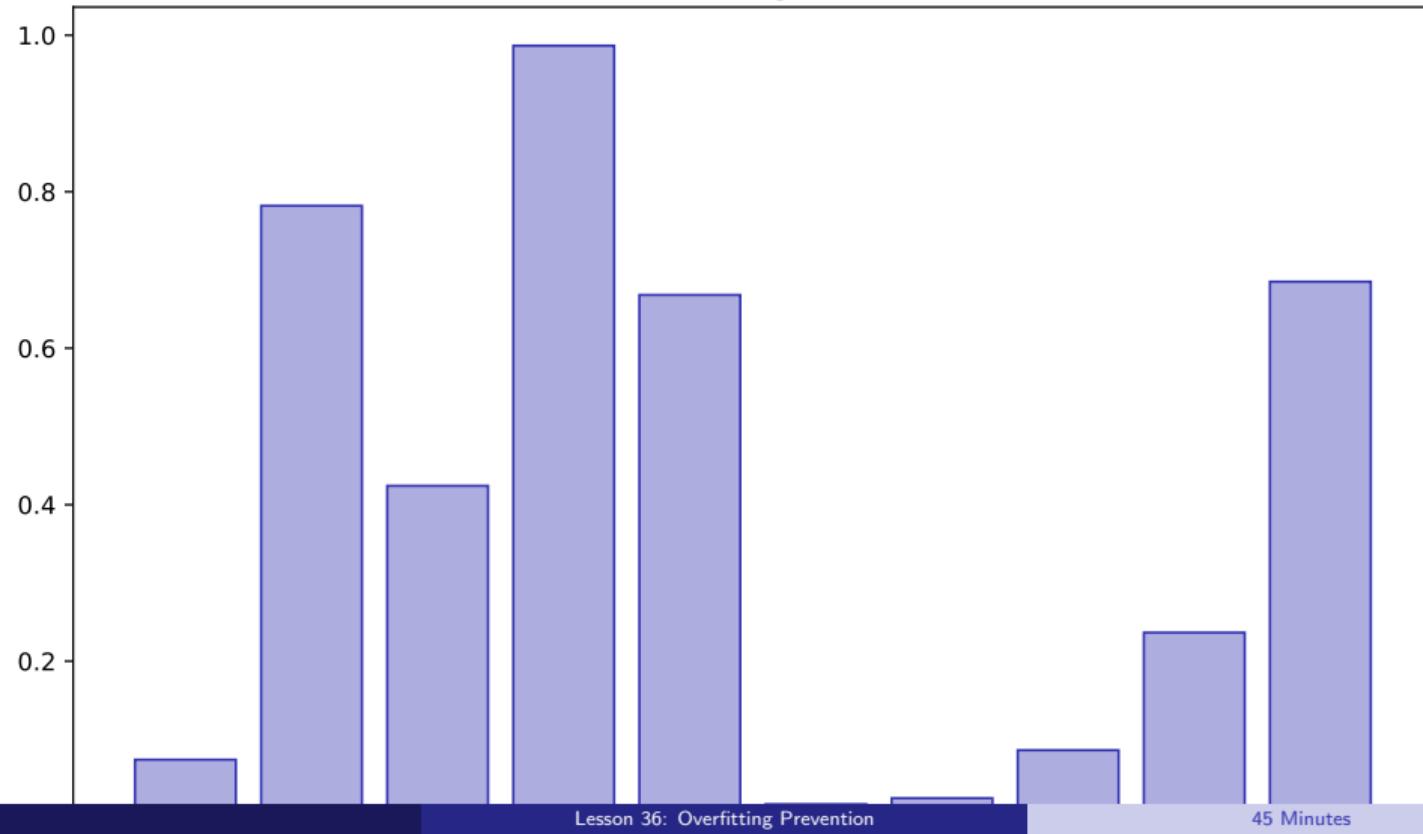
45 Minutes

After this lesson, you will be able to:

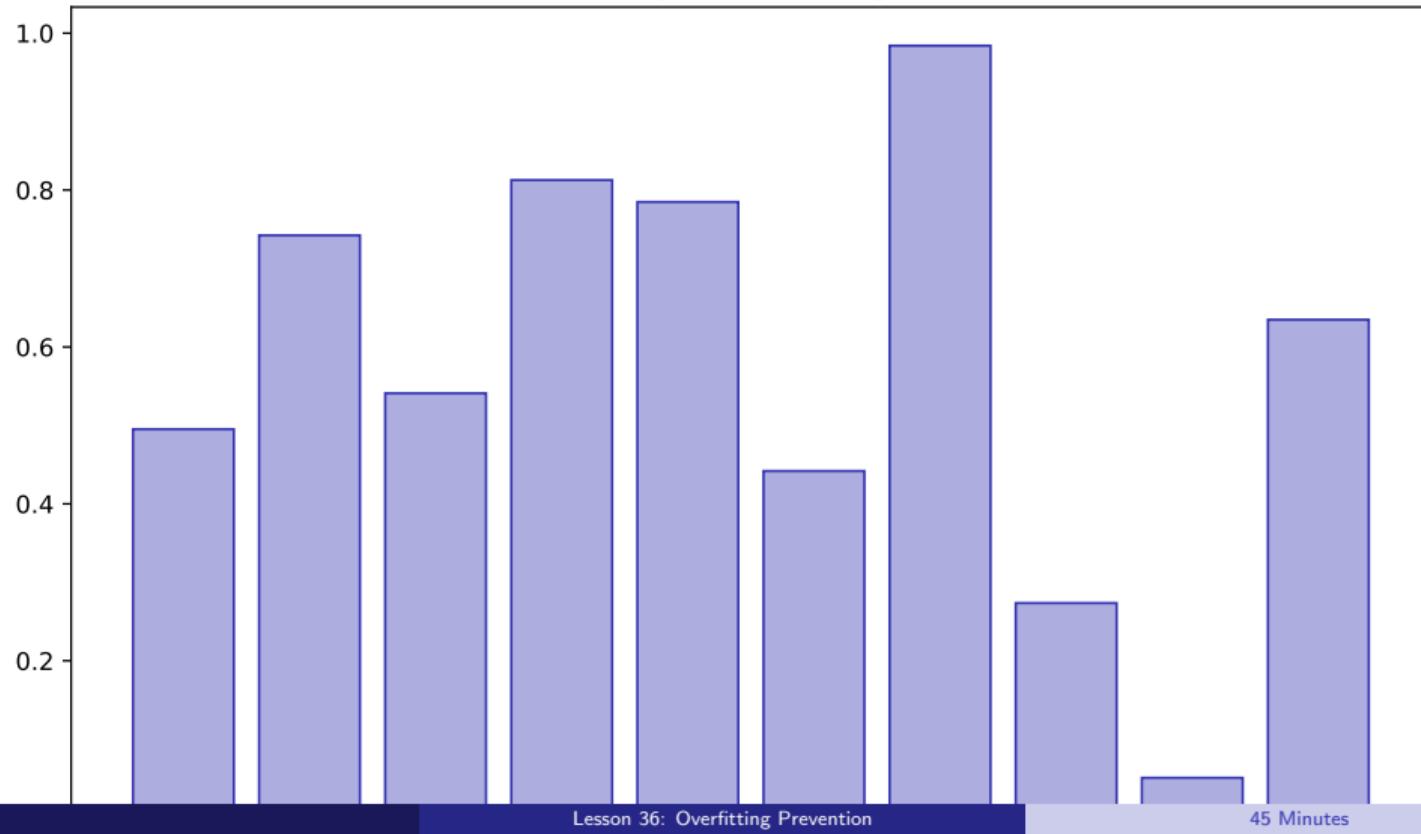
- Apply dropout regularization
- Use early stopping
- Diagnose overfitting
- Build robust models

Building towards your final project

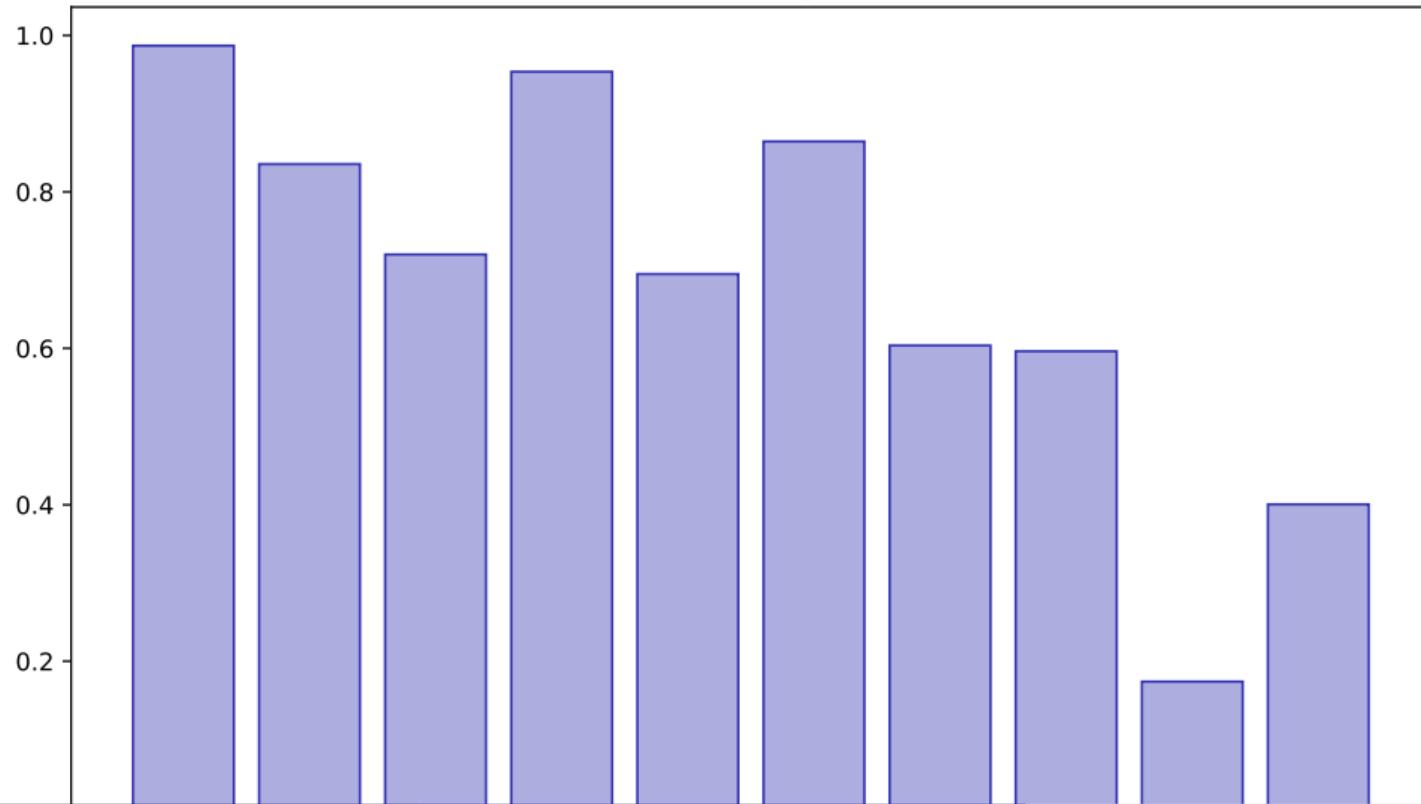
Overfitting Visual



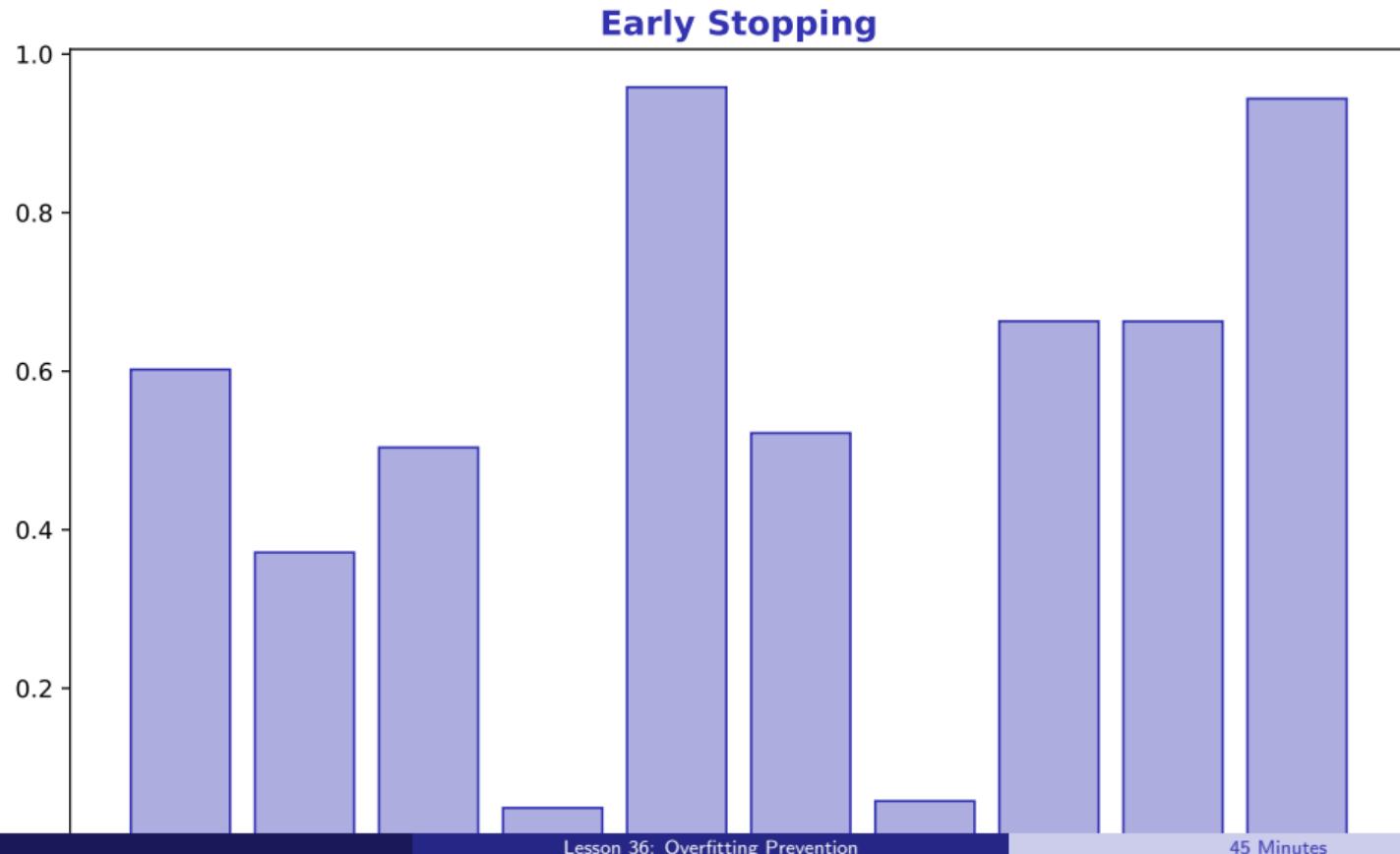
Dropout Concept



Keras Dropout

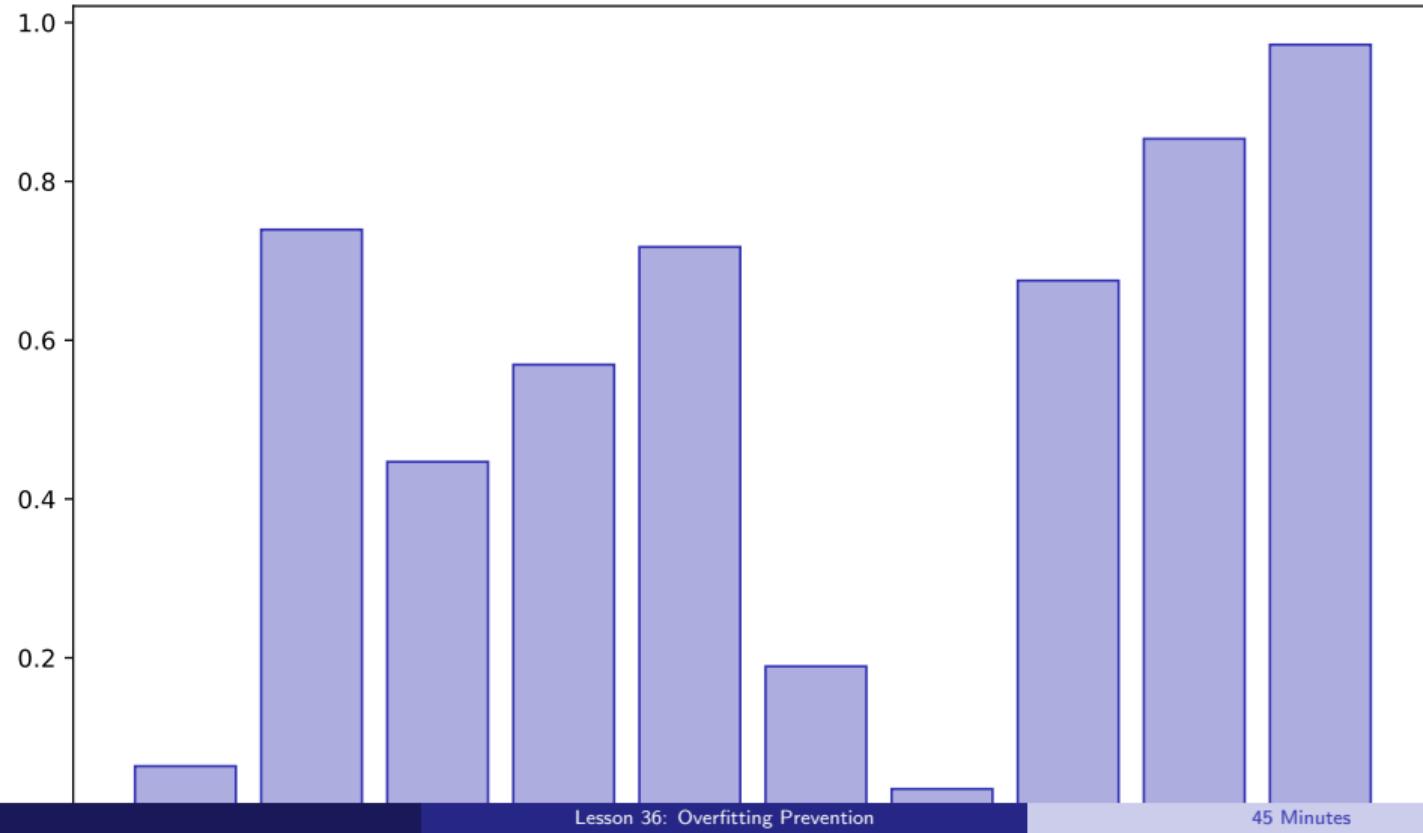


Early Stopping

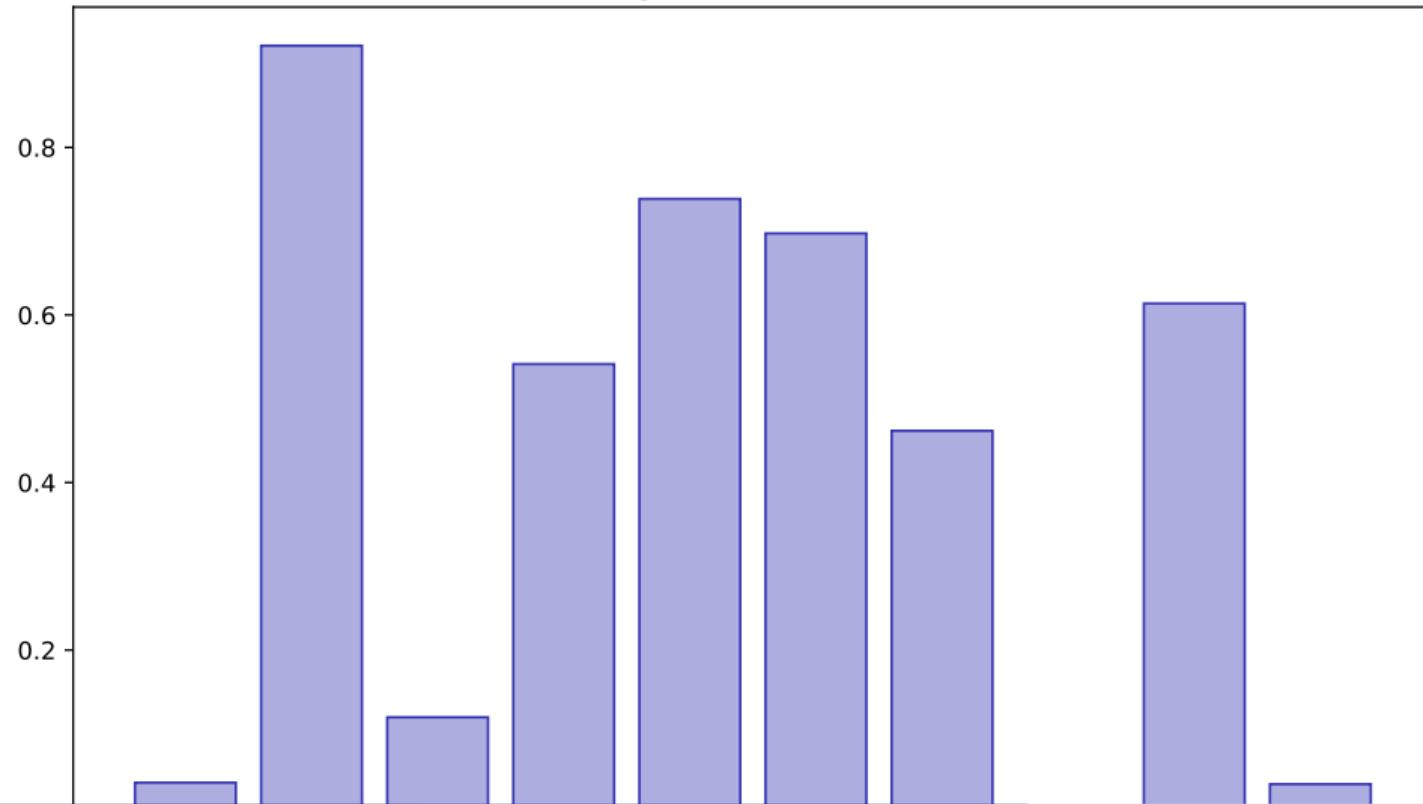


Validation Curves

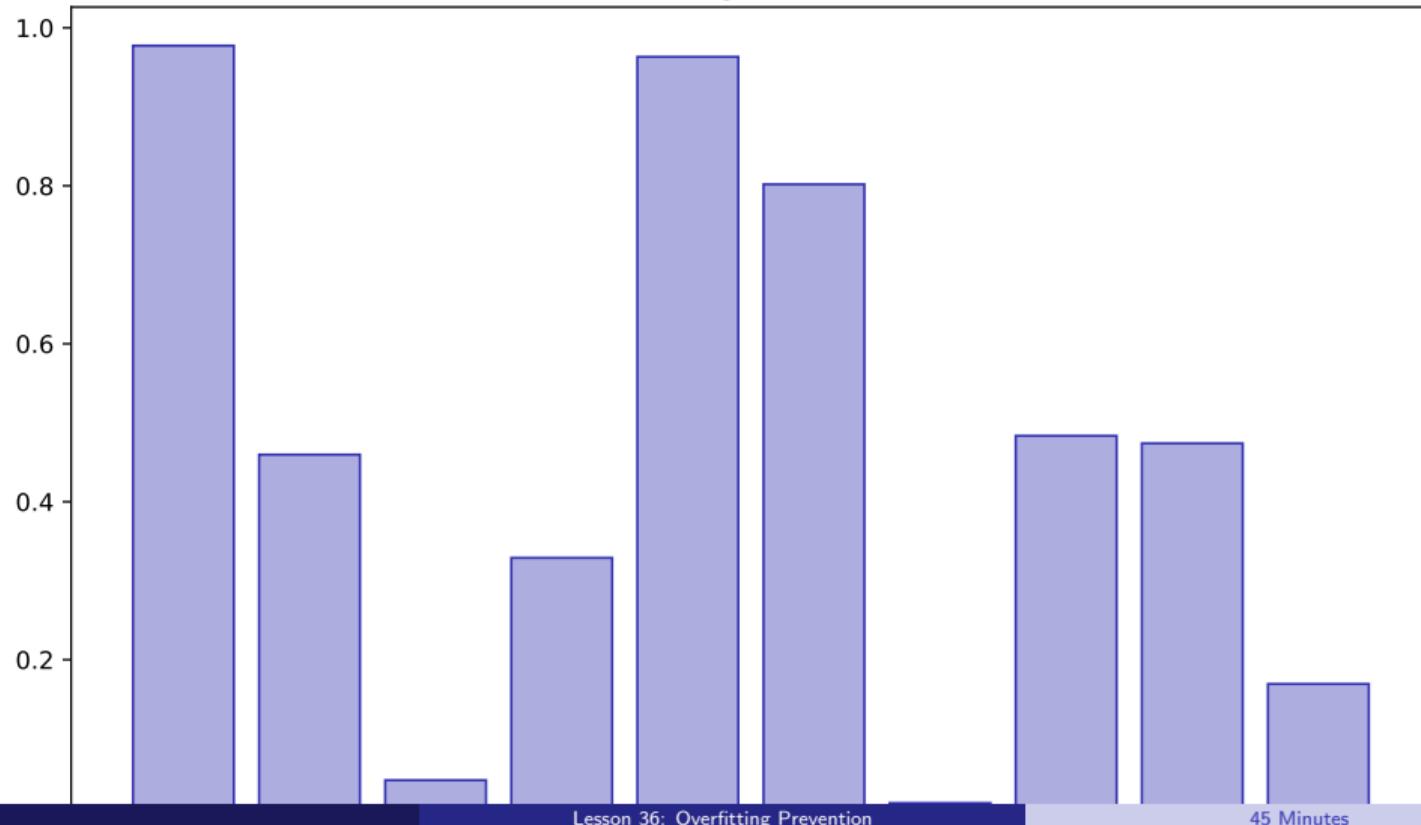
Validation Curves



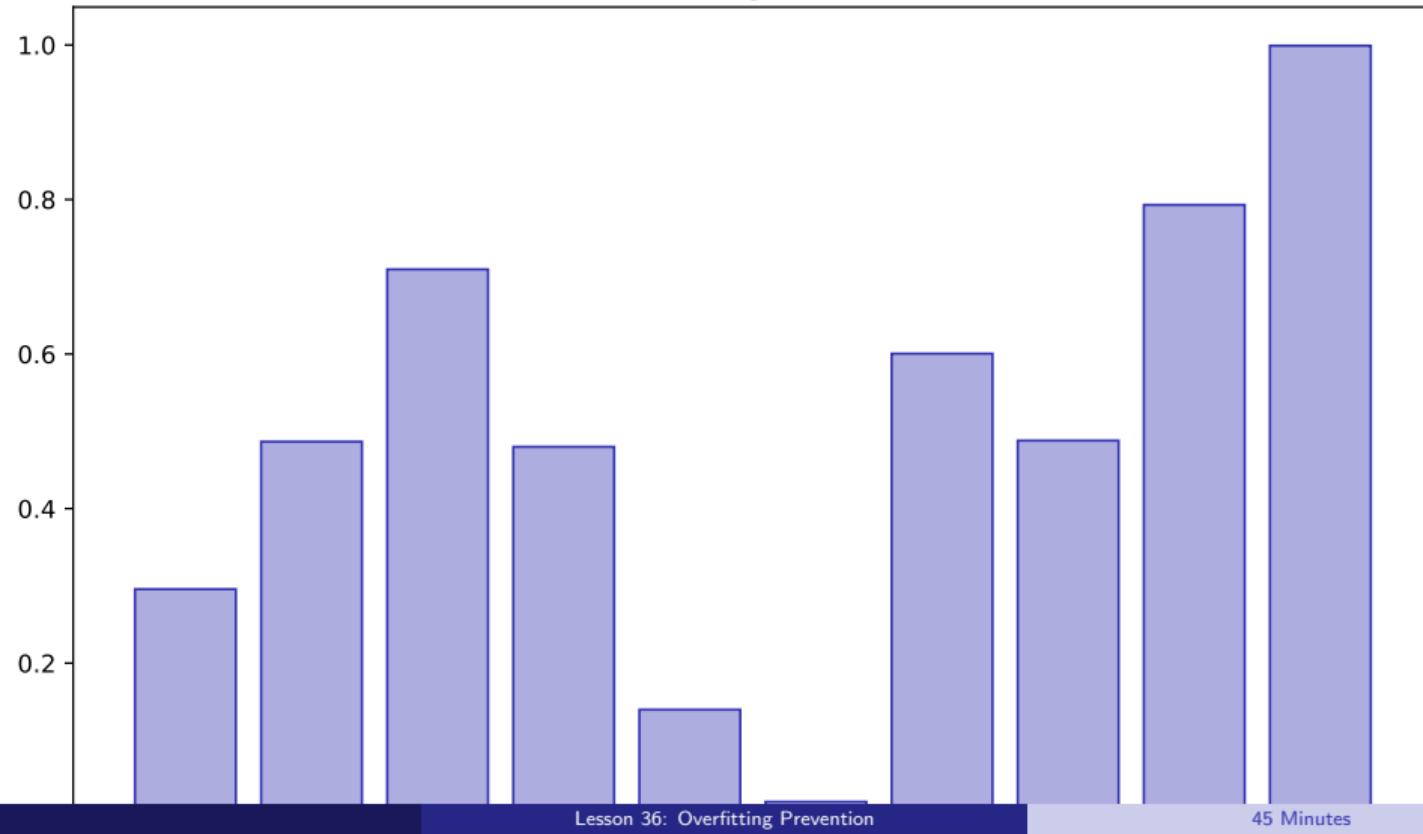
Regularization L2



Data Augmentation



Finance Regularization



Key Takeaways:

- Apply dropout regularization
- Use early stopping
- Diagnose overfitting
- Build robust models

Apply these skills in your final project

Lesson 37: Text Preprocessing

Data Science with Python – BSc Course

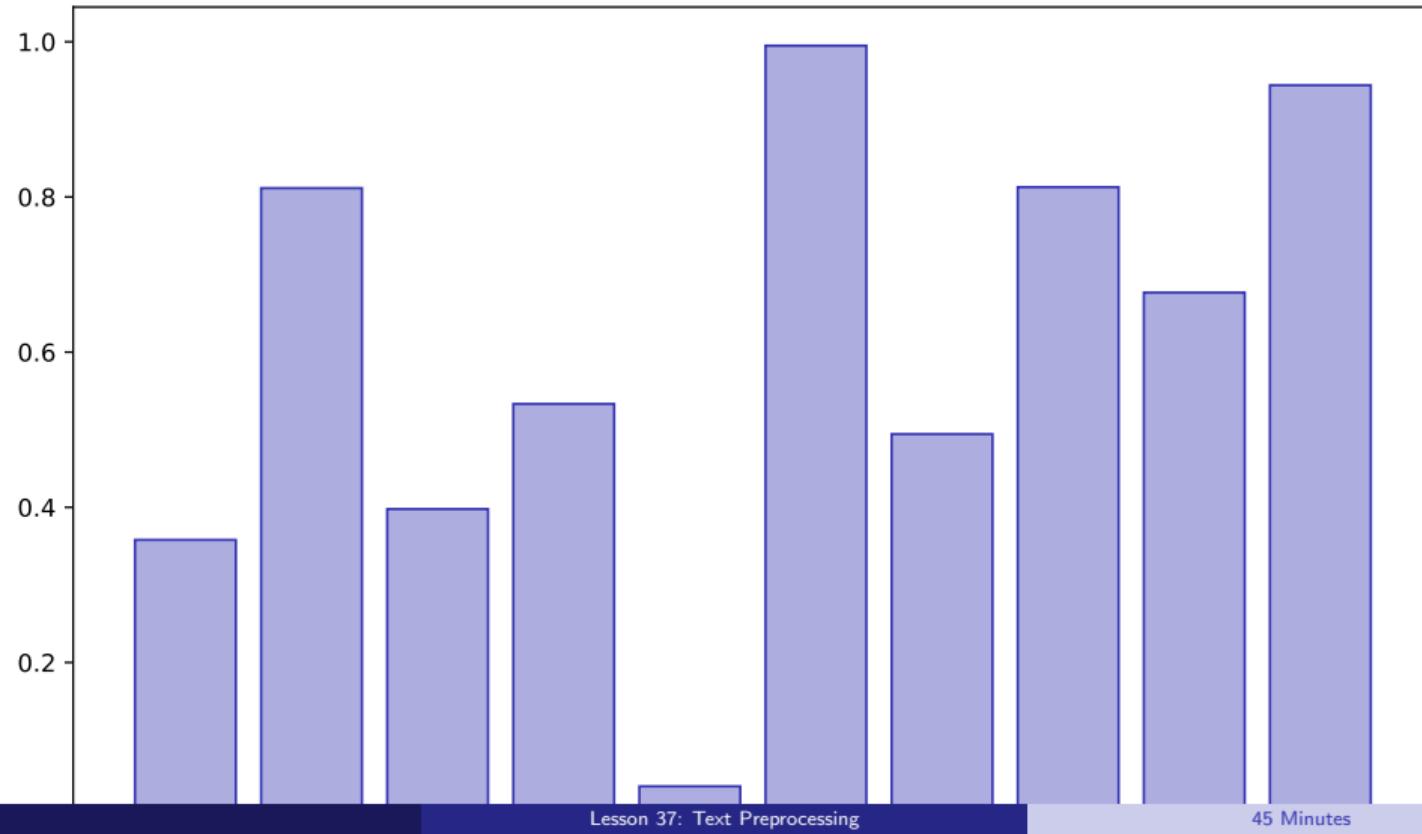
45 Minutes

After this lesson, you will be able to:

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

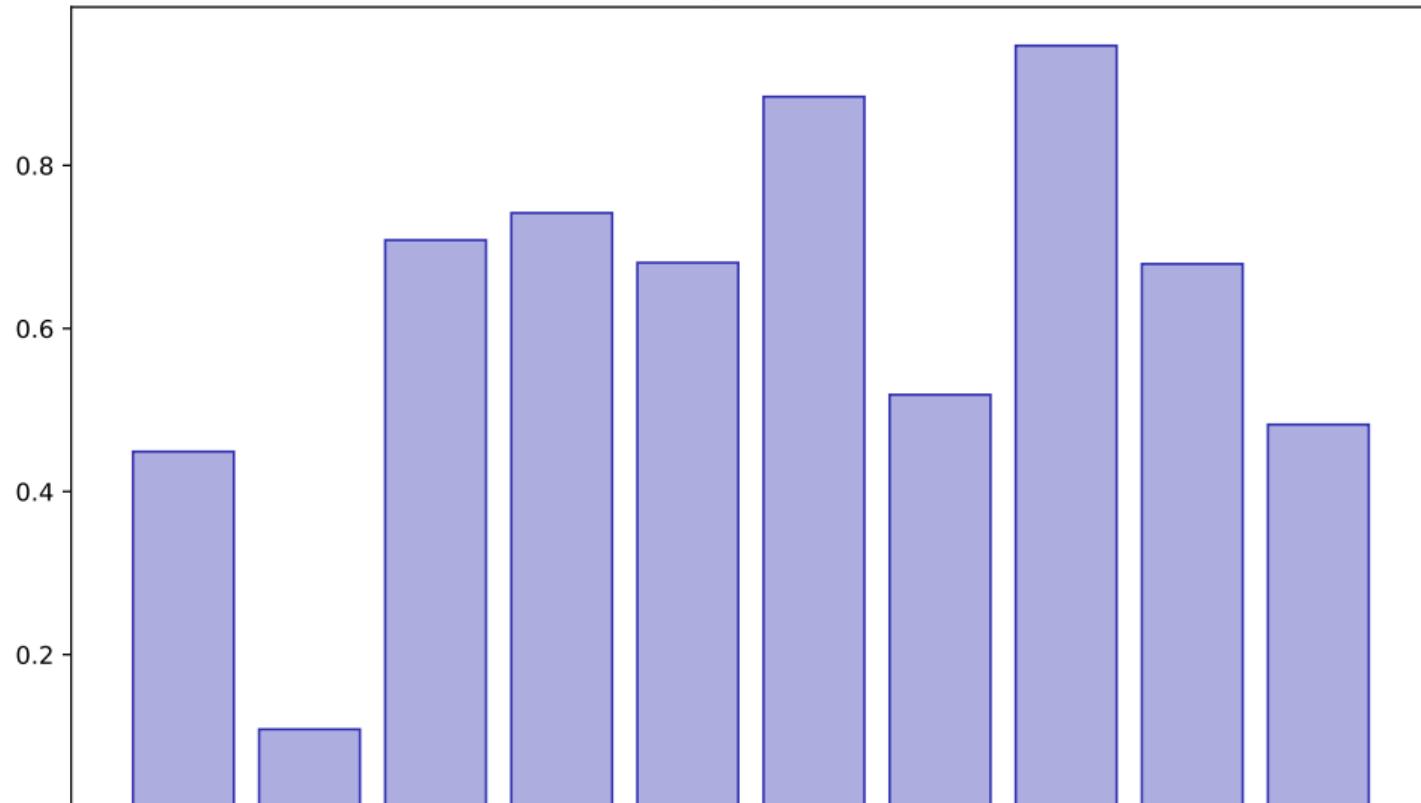
Building towards your final project

Tokenization

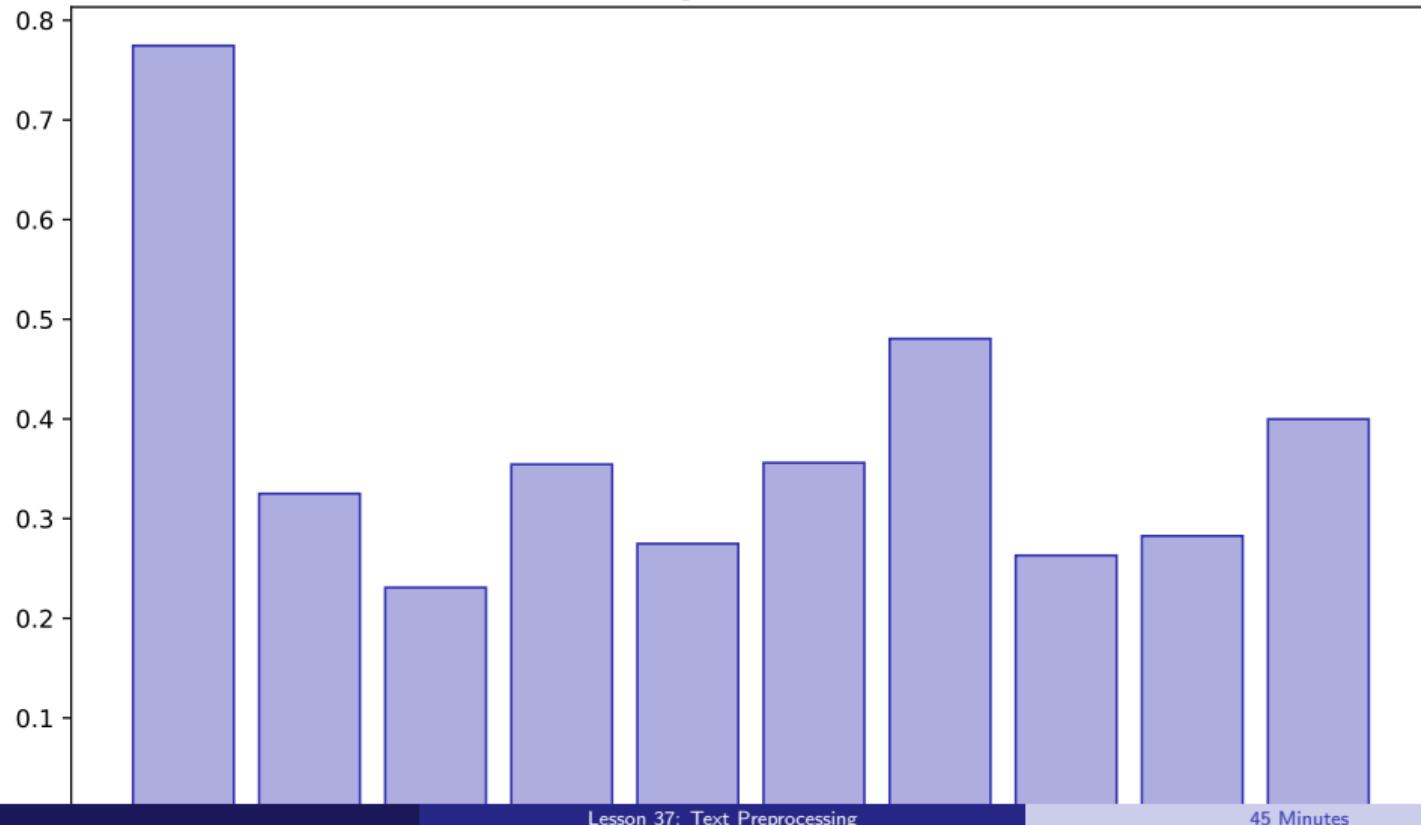


Stopwords

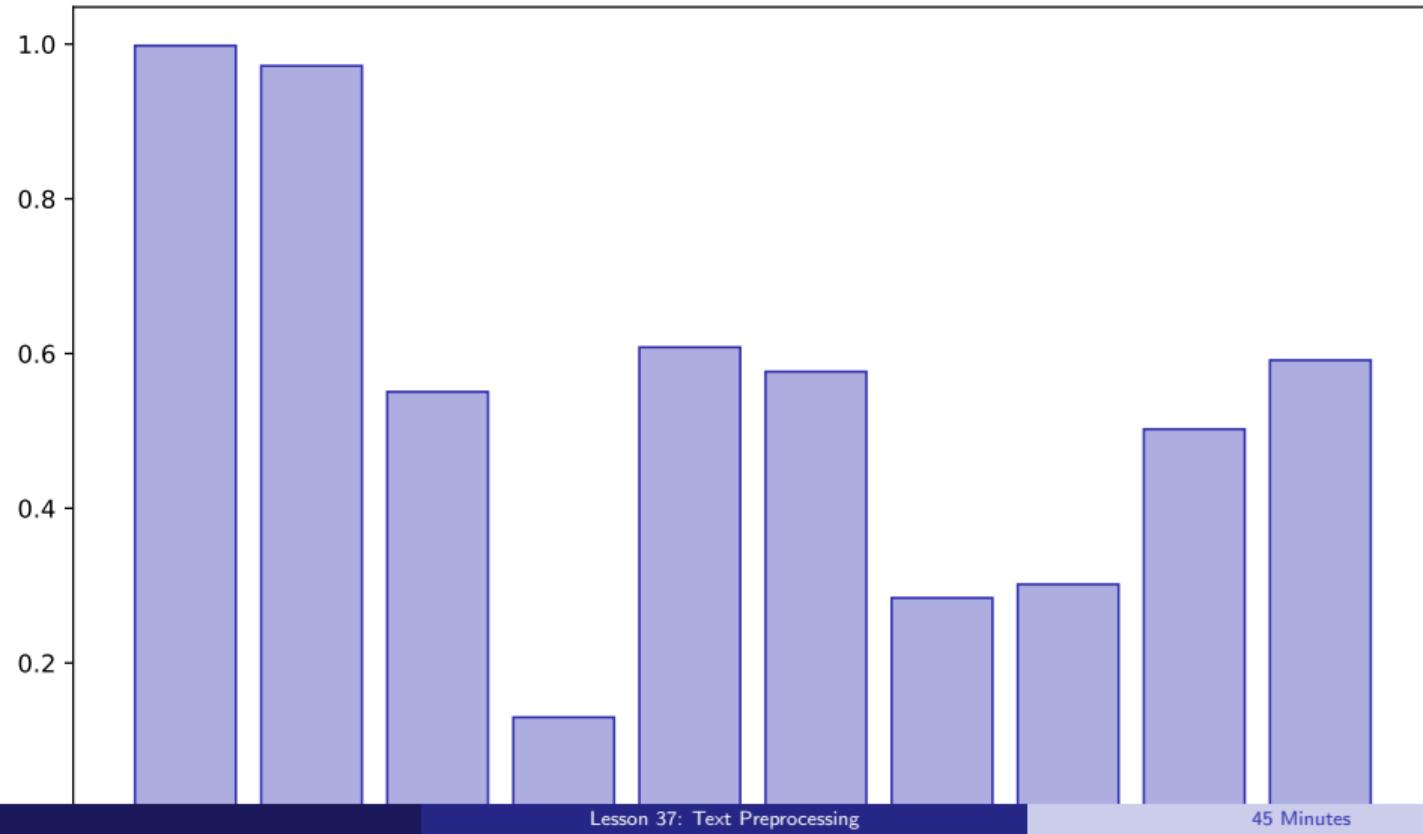
Stopwords



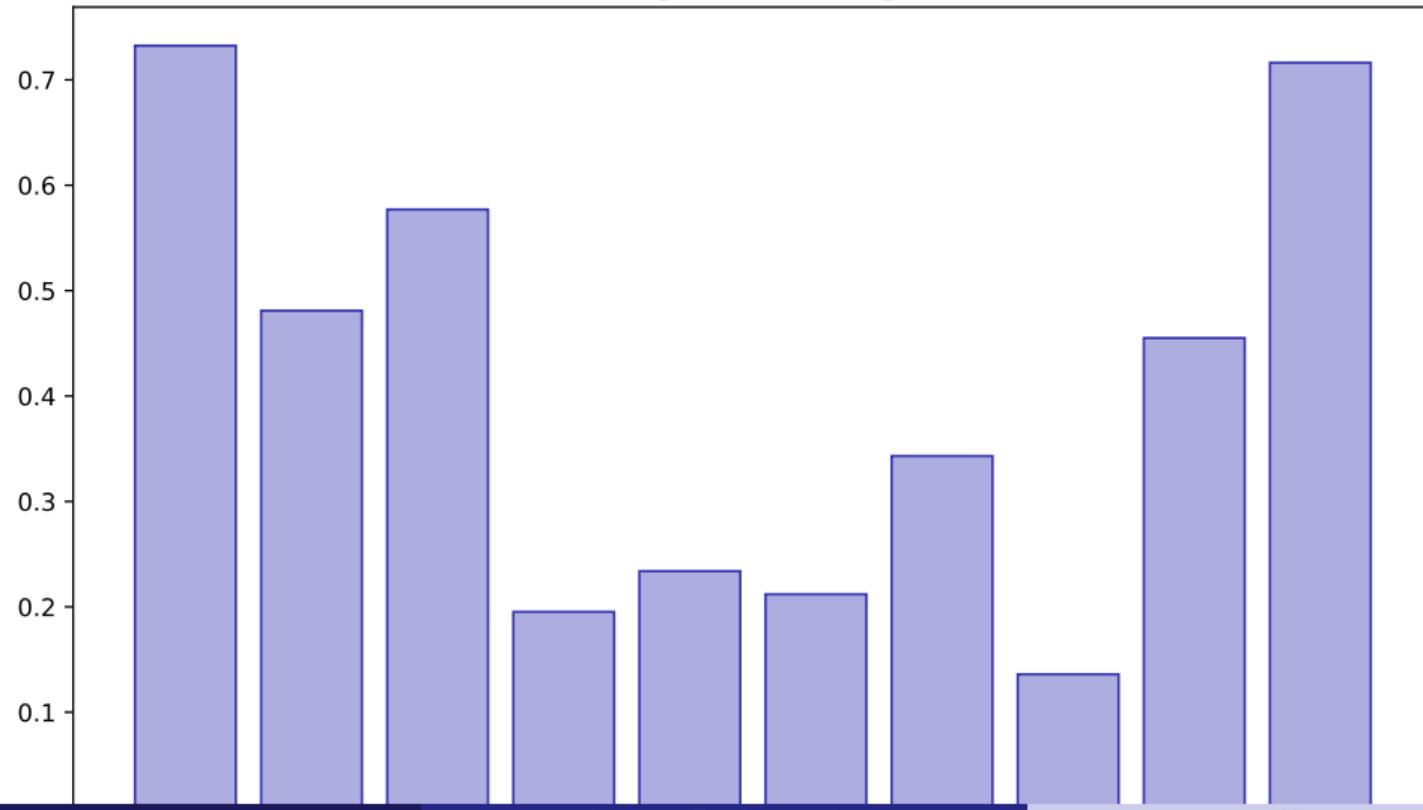
Stemming Lemmatization



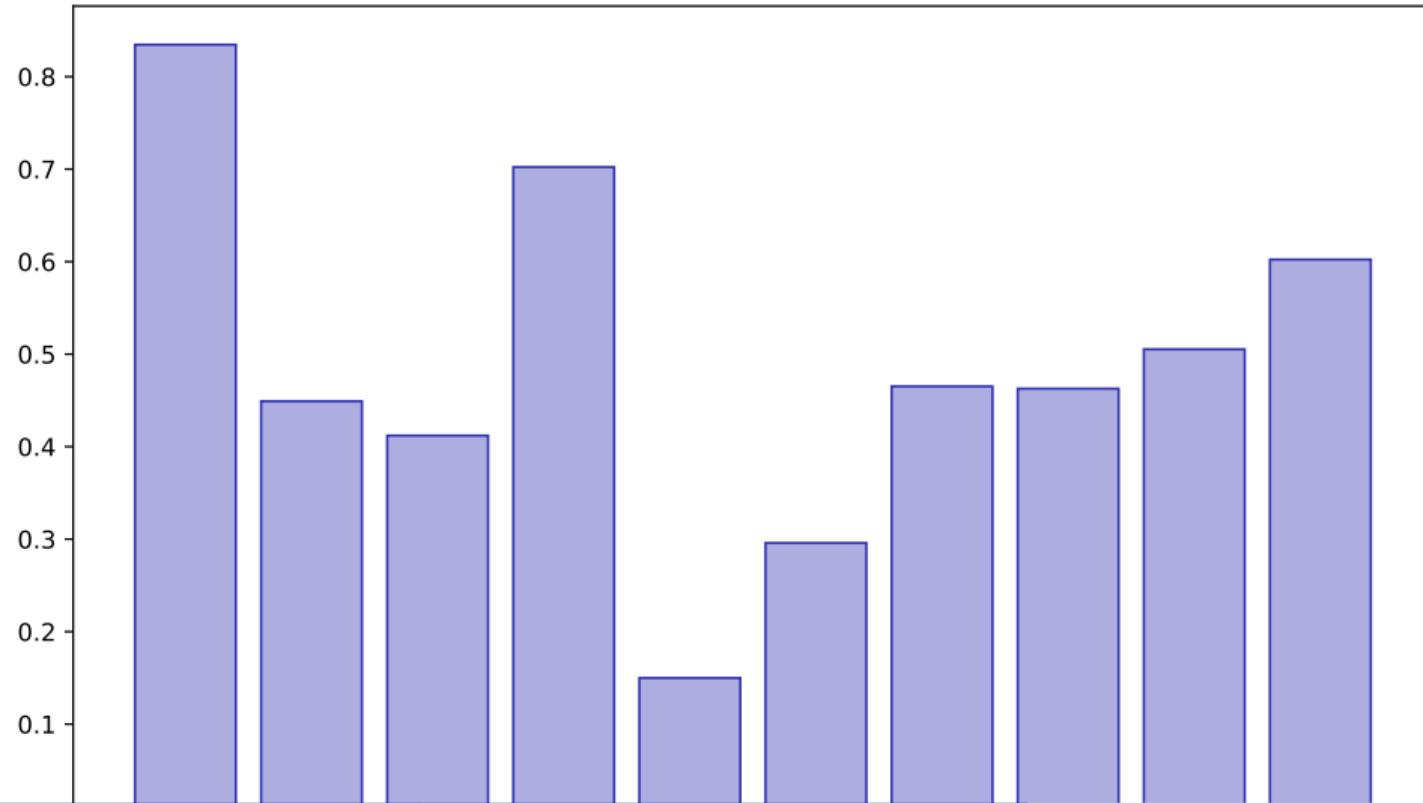
Nltk Basics



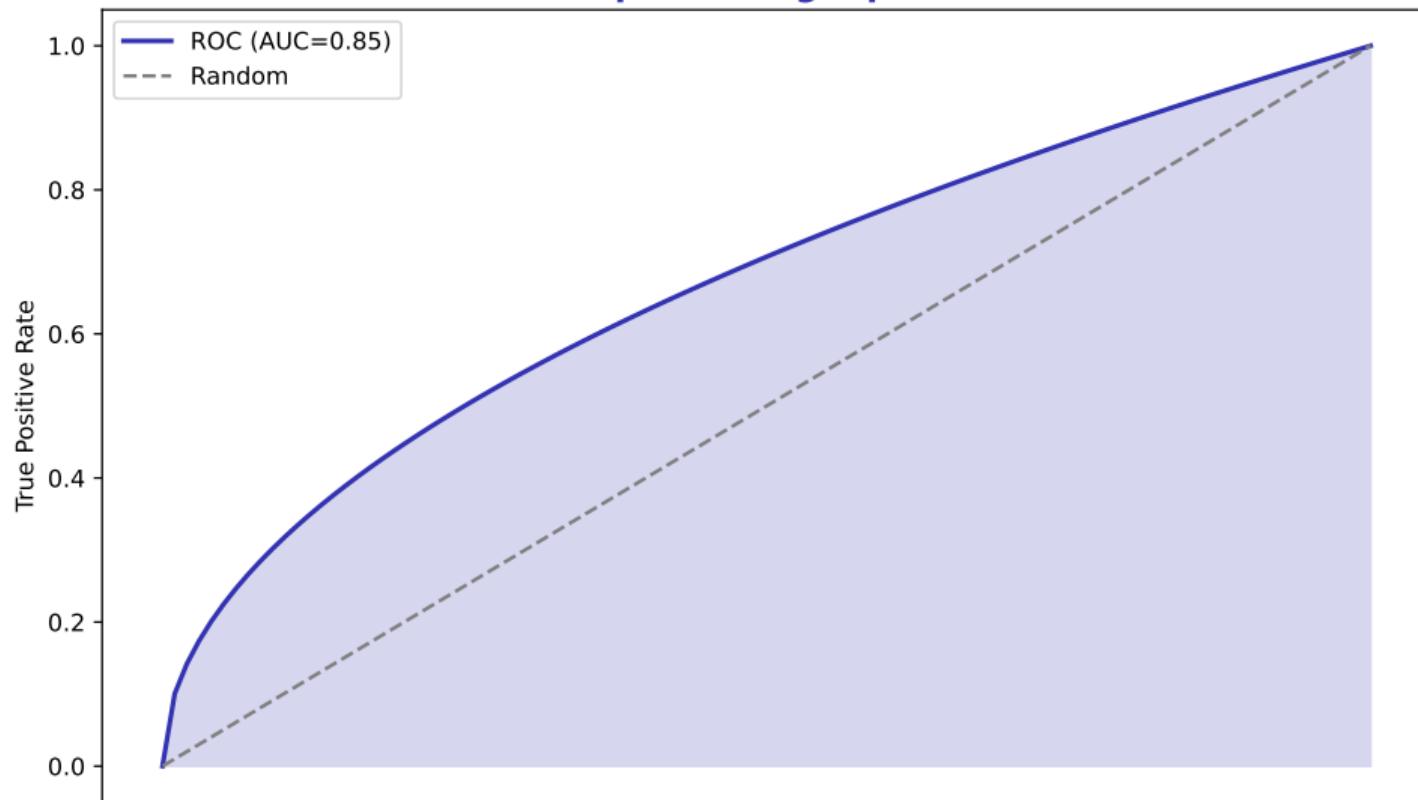
Regex Cleaning



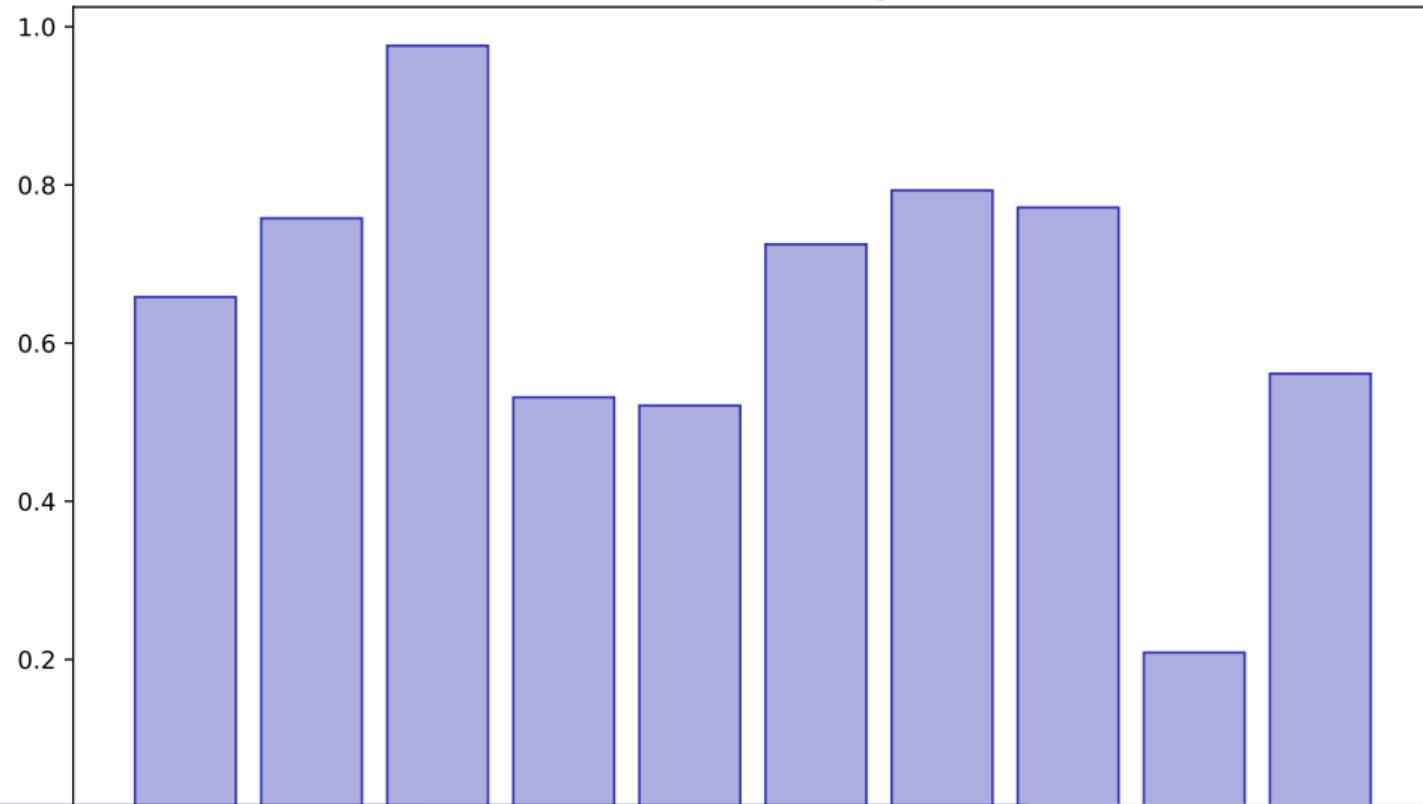
Financial Text



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

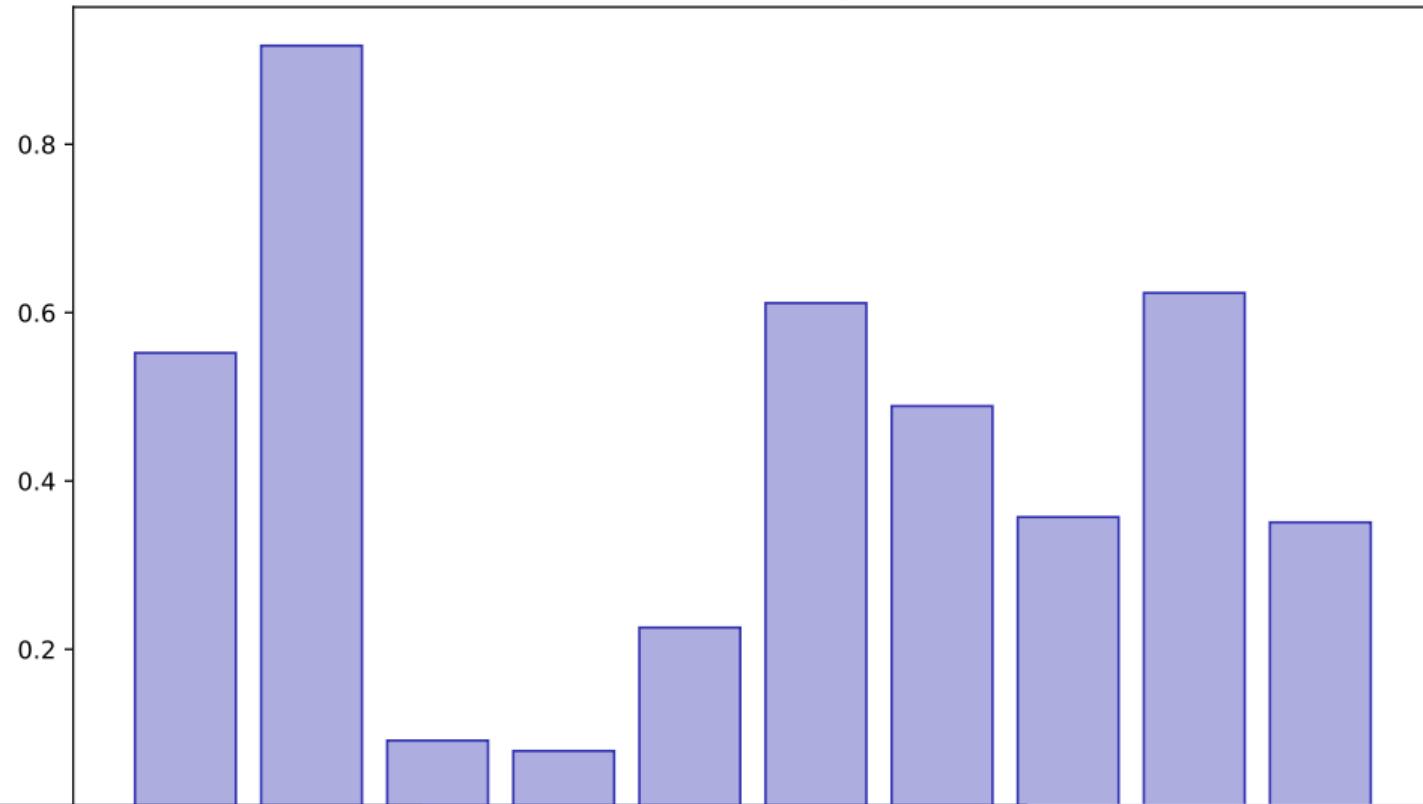
45 Minutes

After this lesson, you will be able to:

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

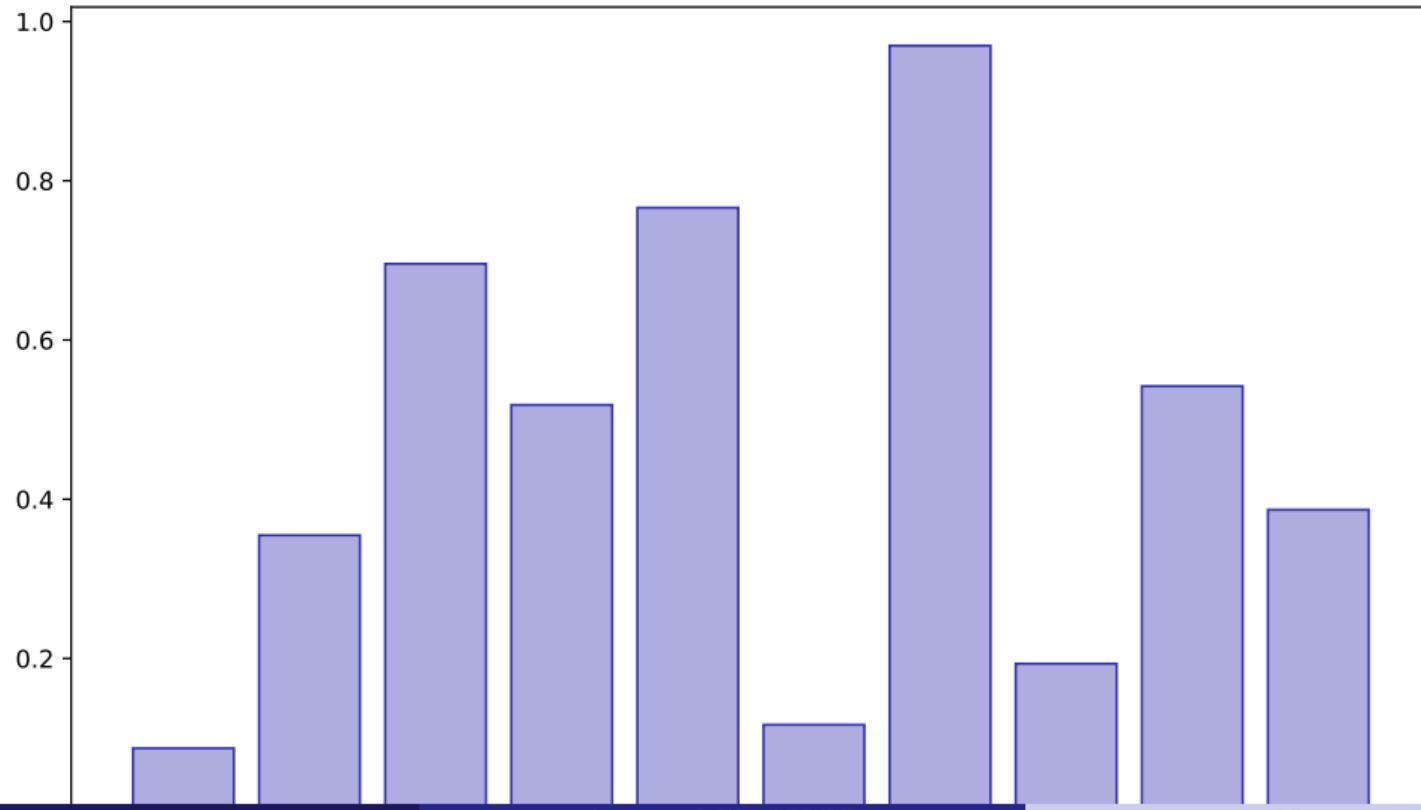
Building towards your final project

Bow Concept

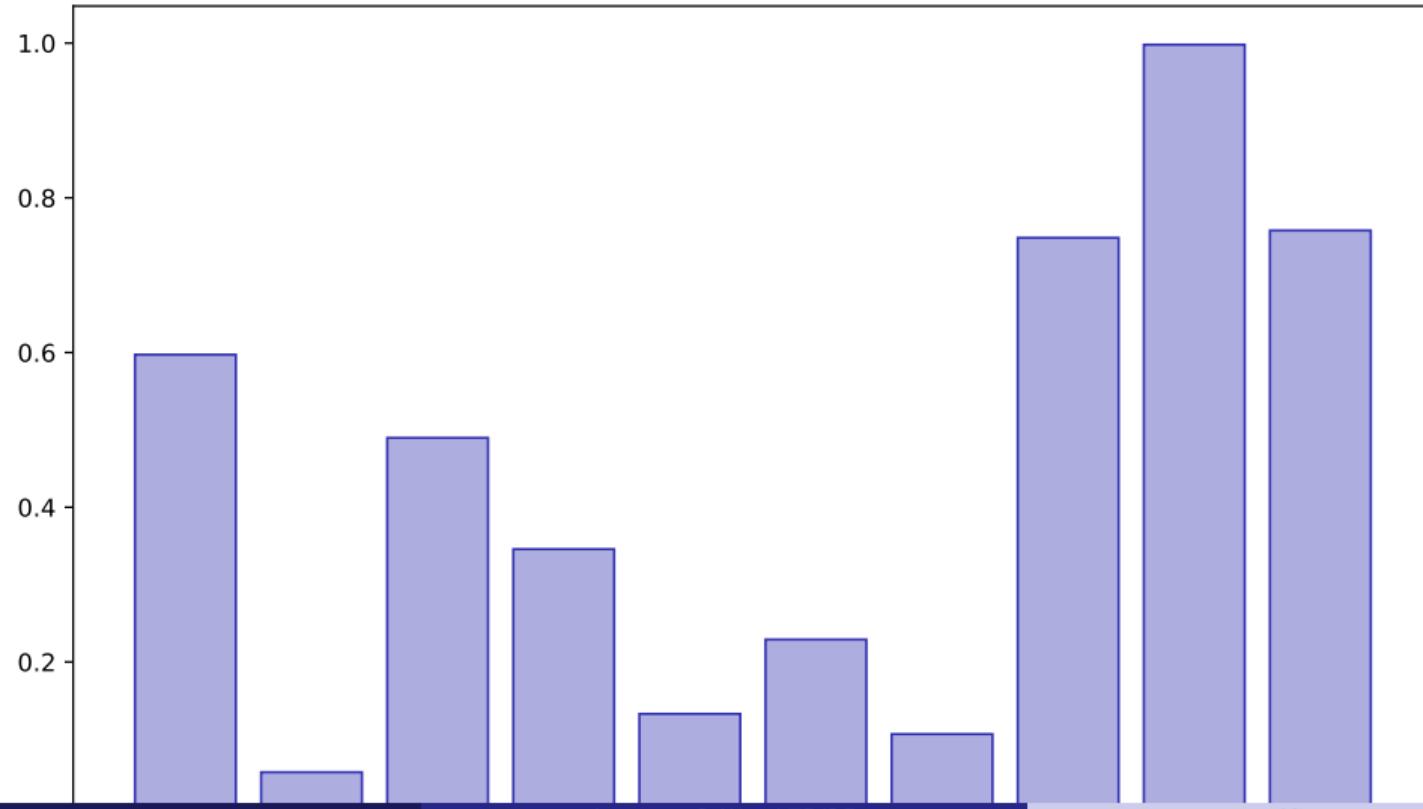


Count Vectorizer

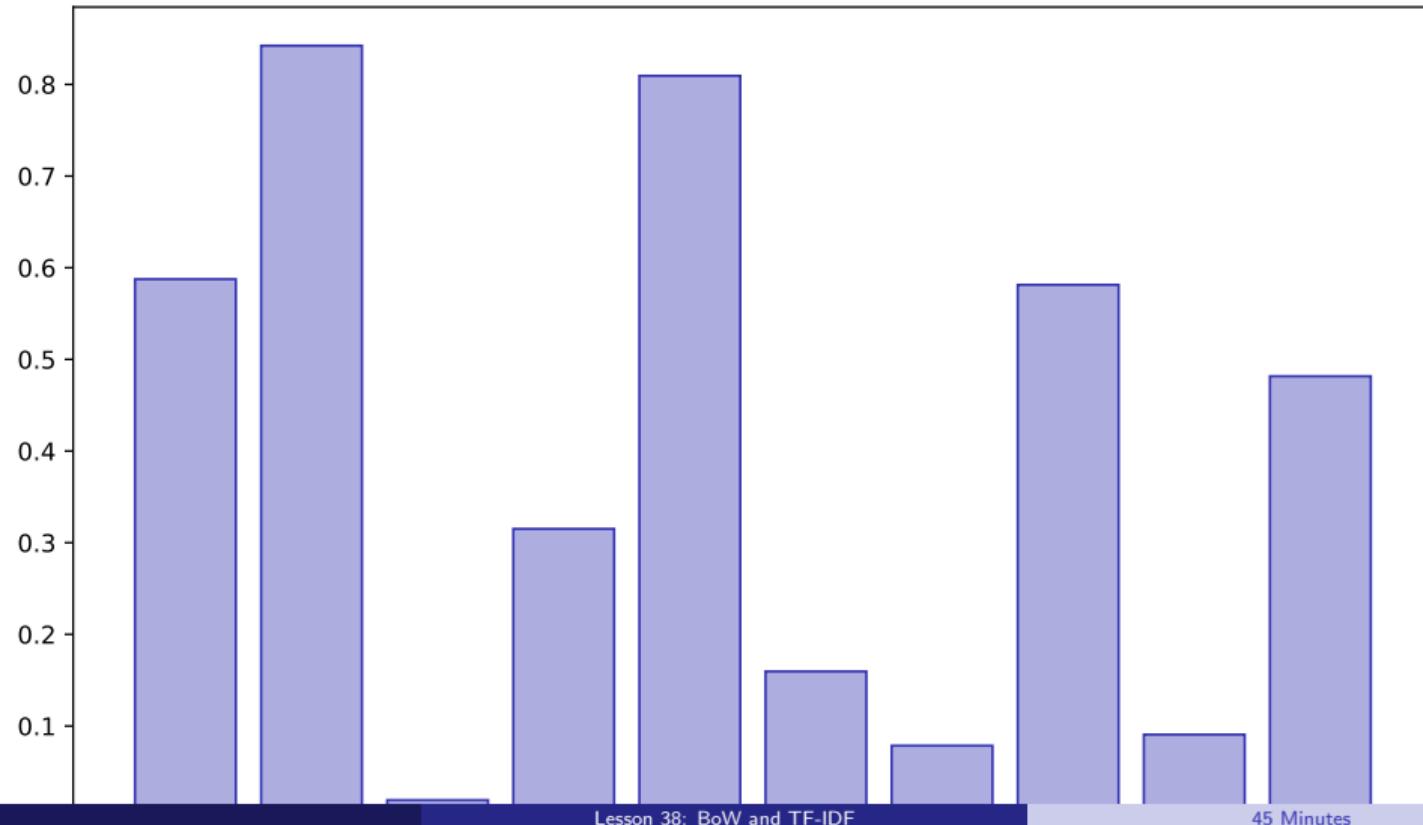
Count Vectorizer



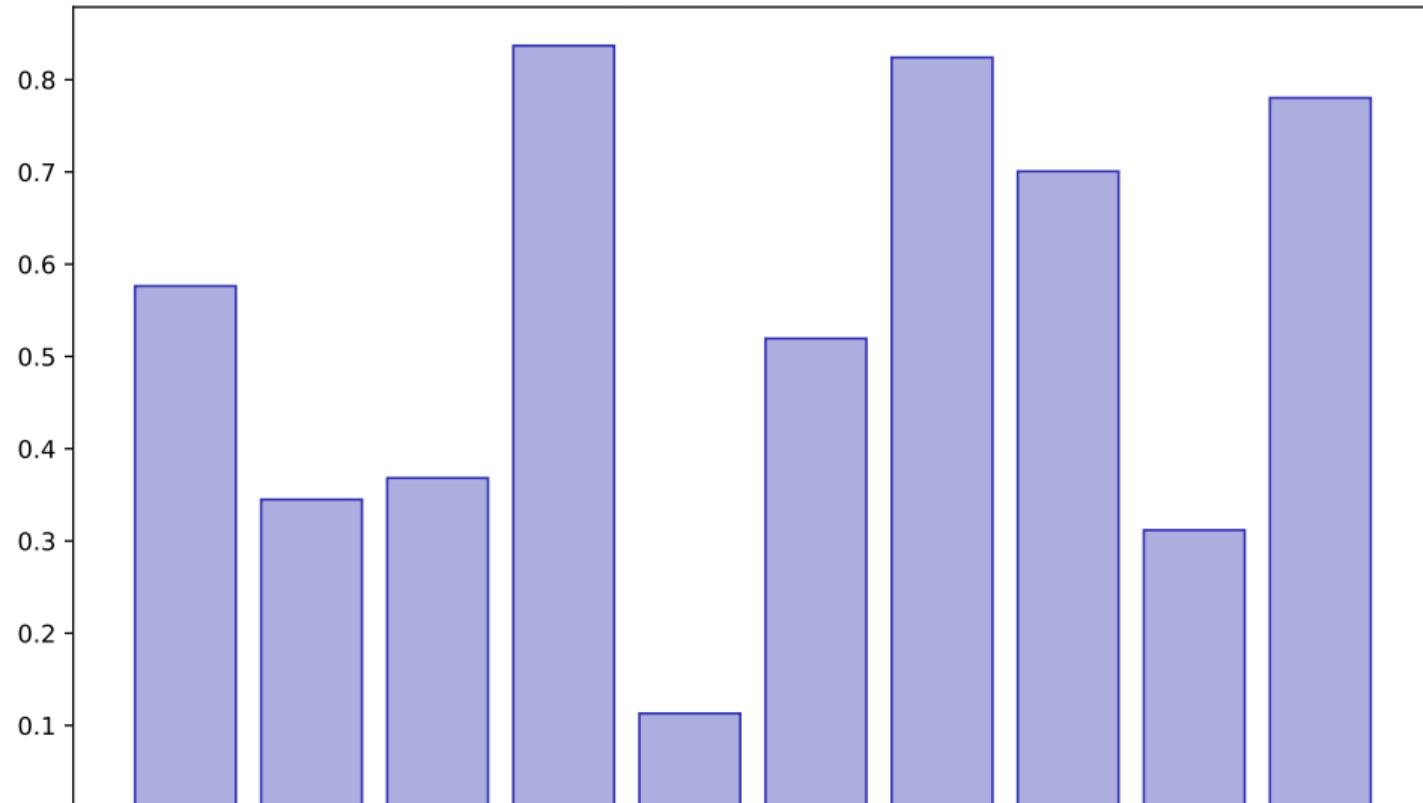
Tfidf Formula



Tfidf Vectorizer

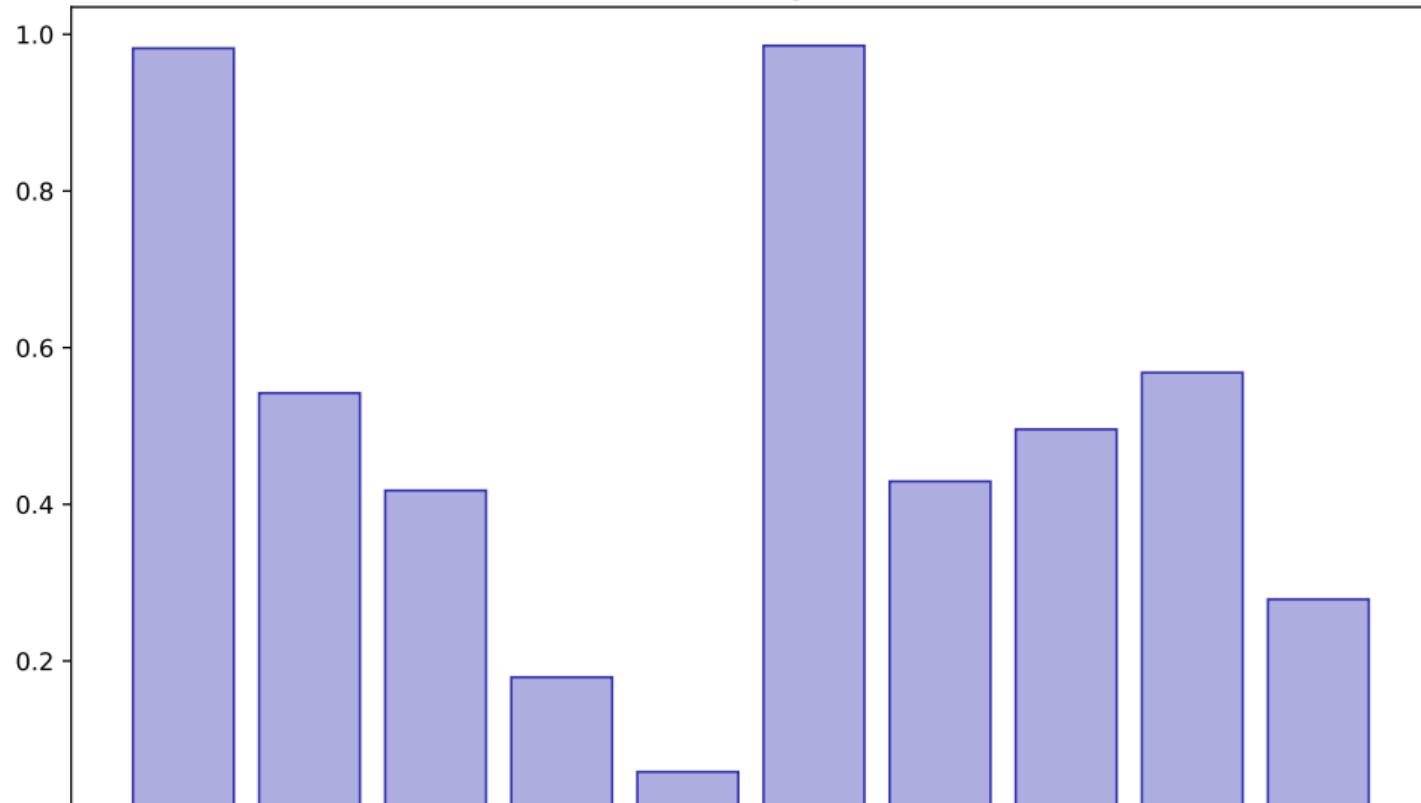


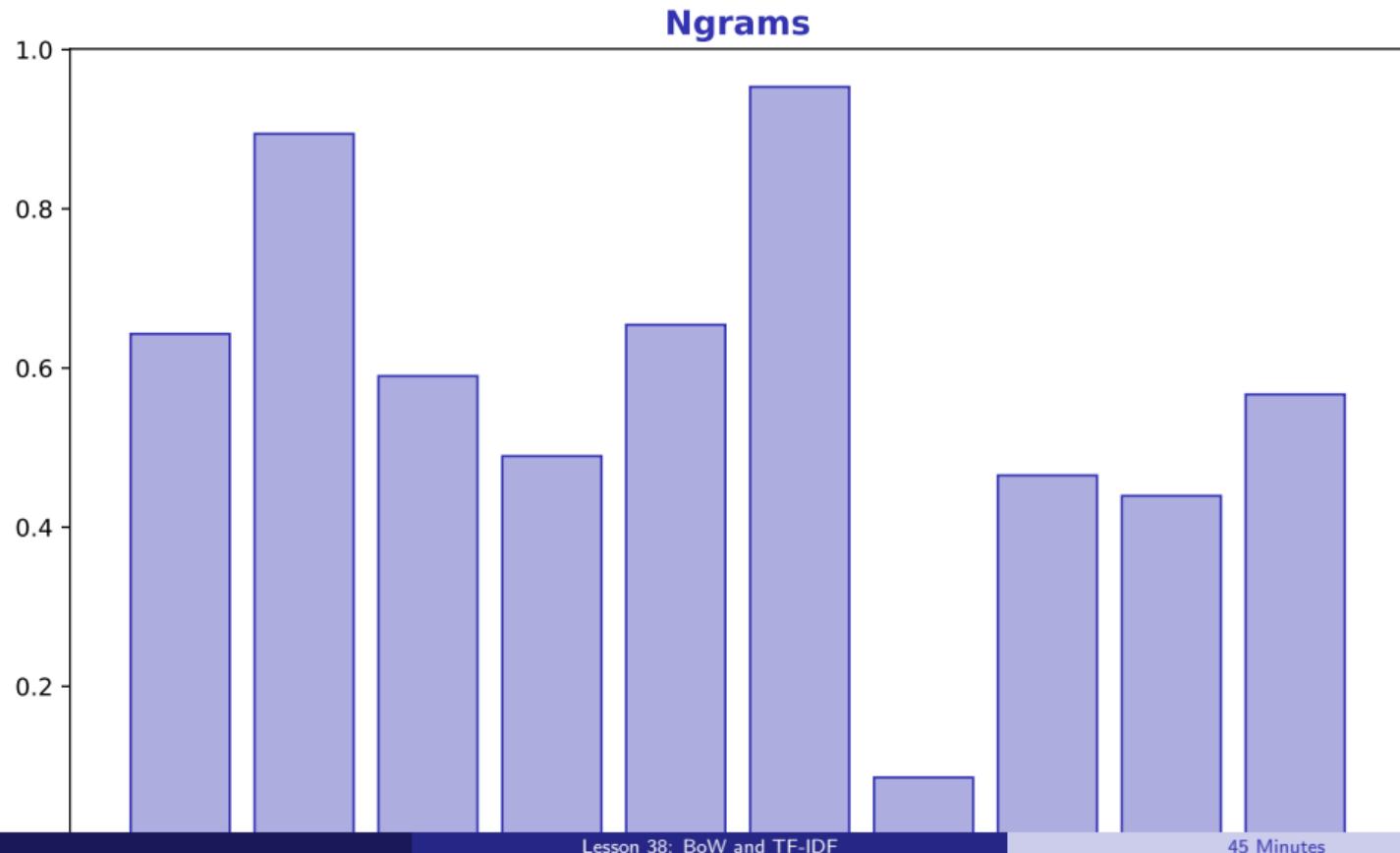
Sparse Matrices



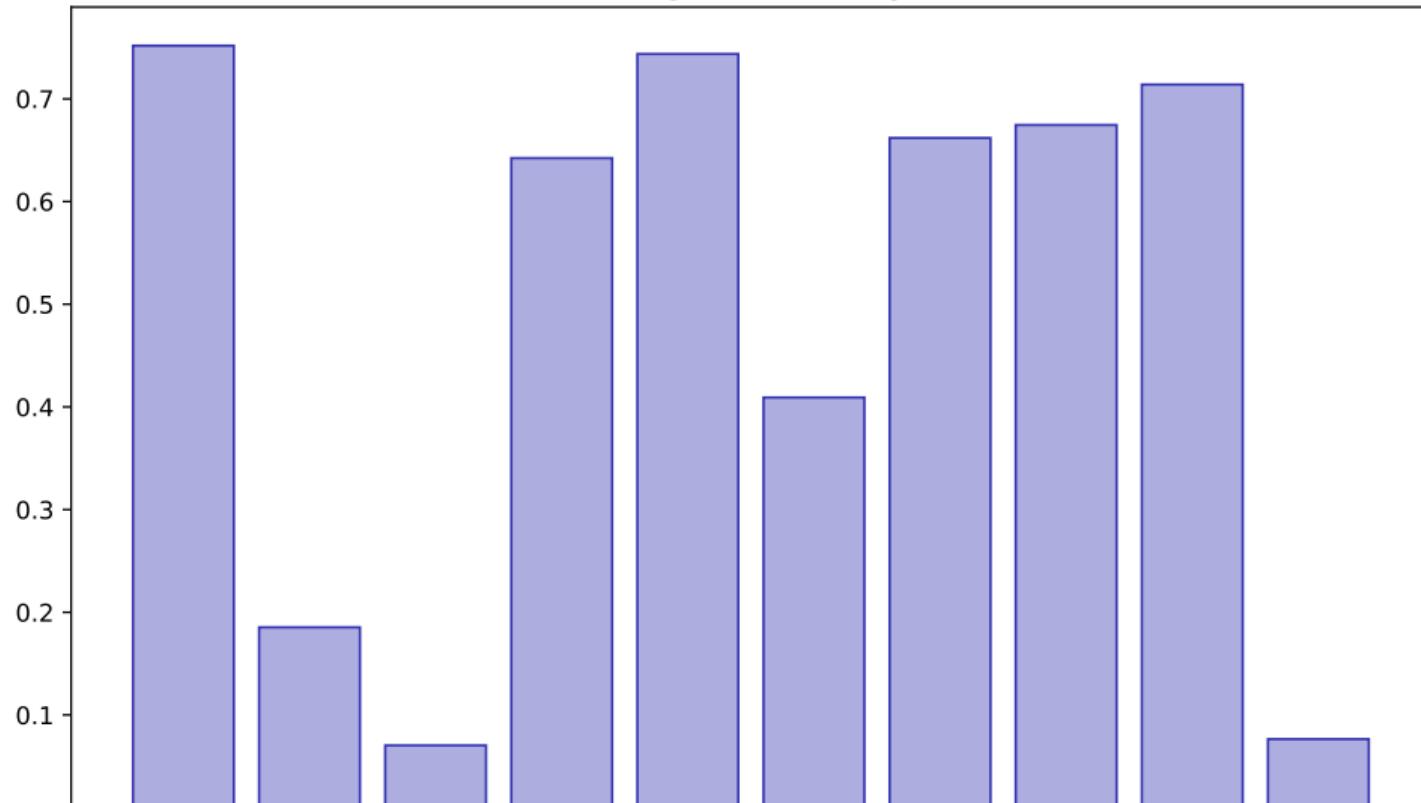
Vocabulary Size

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

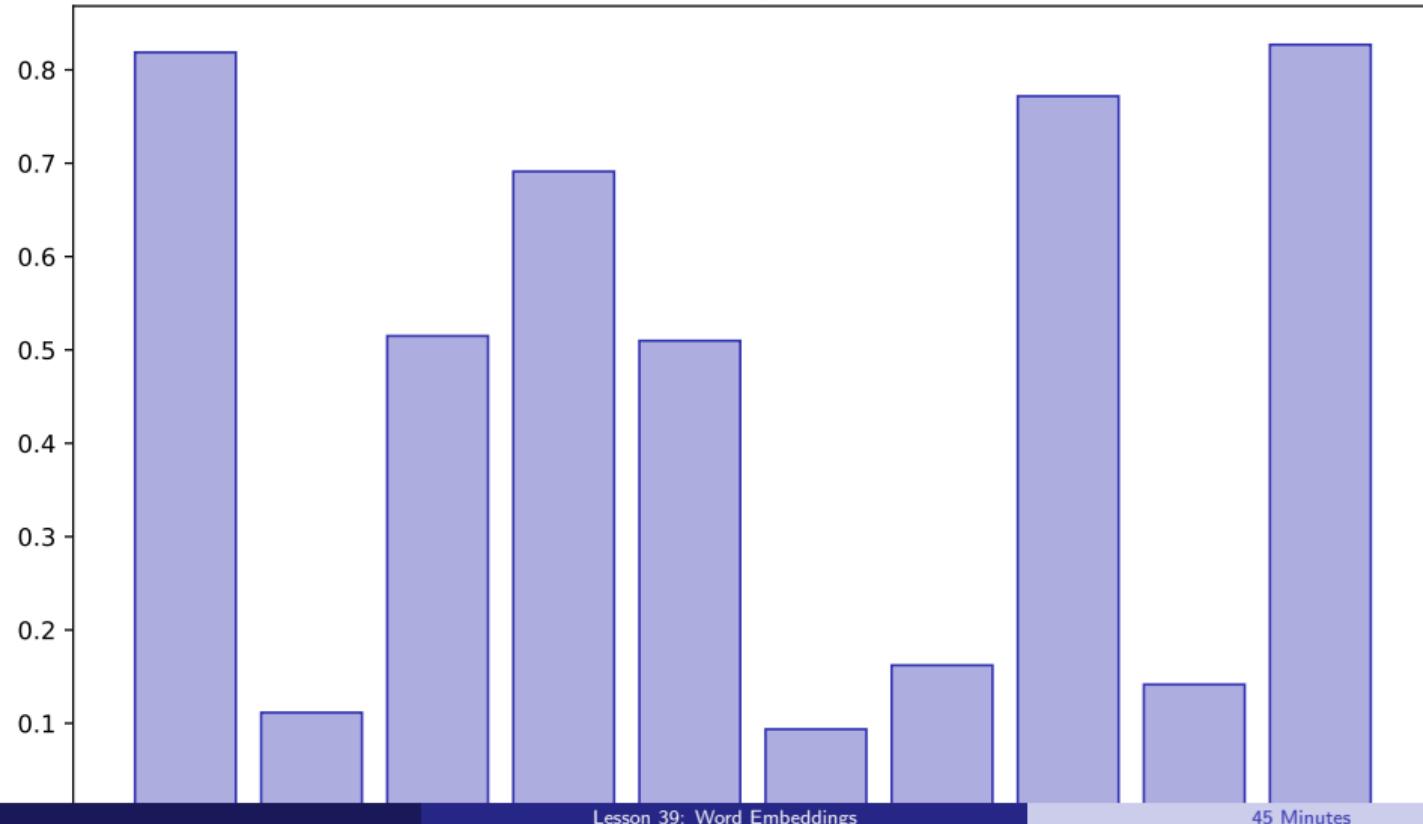
Learning Objectives

After this lesson, you will be able to:

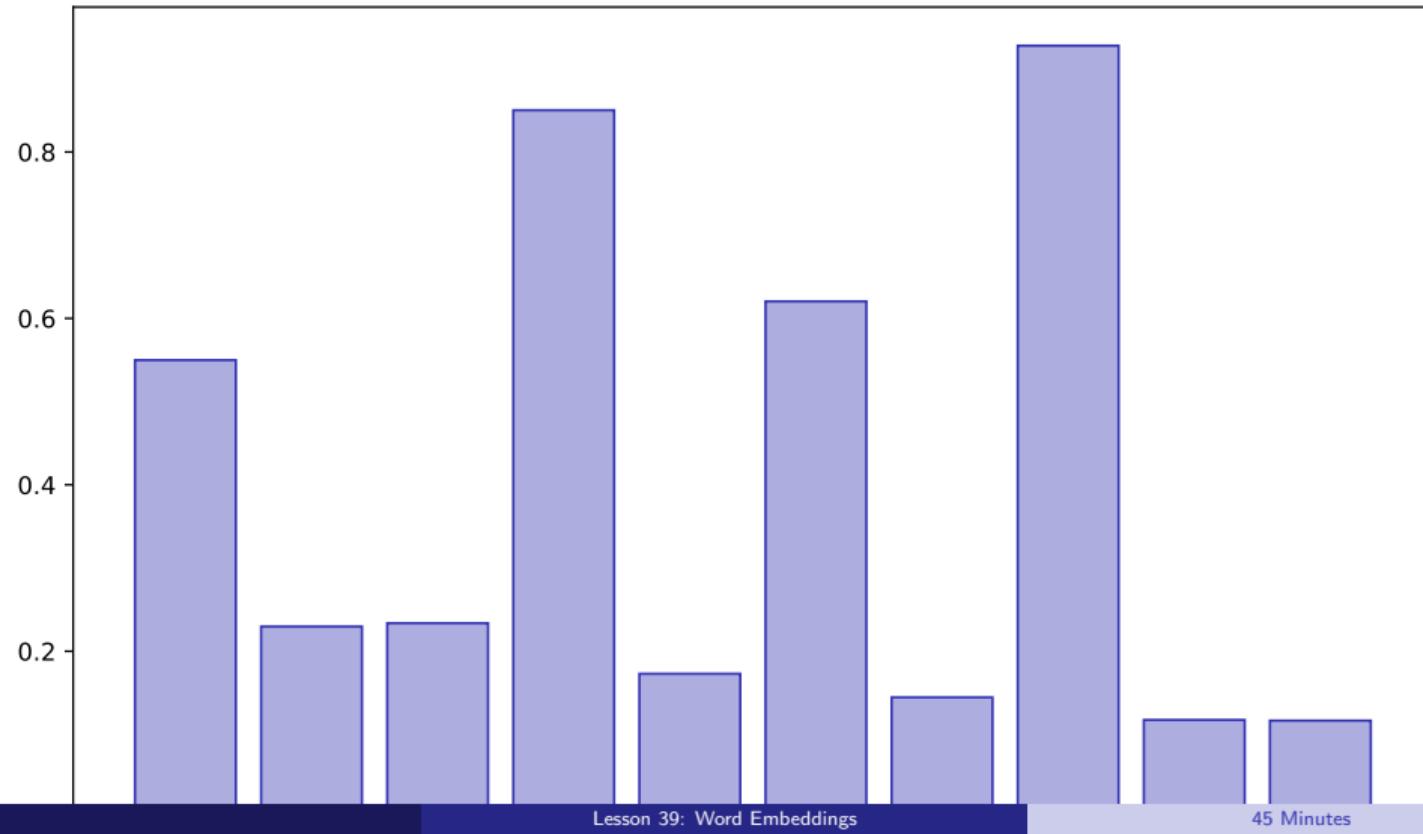
- Understand word vectors
- Use pre-trained Word2Vec
- Find similar words
- Create document embeddings

Building towards your final project

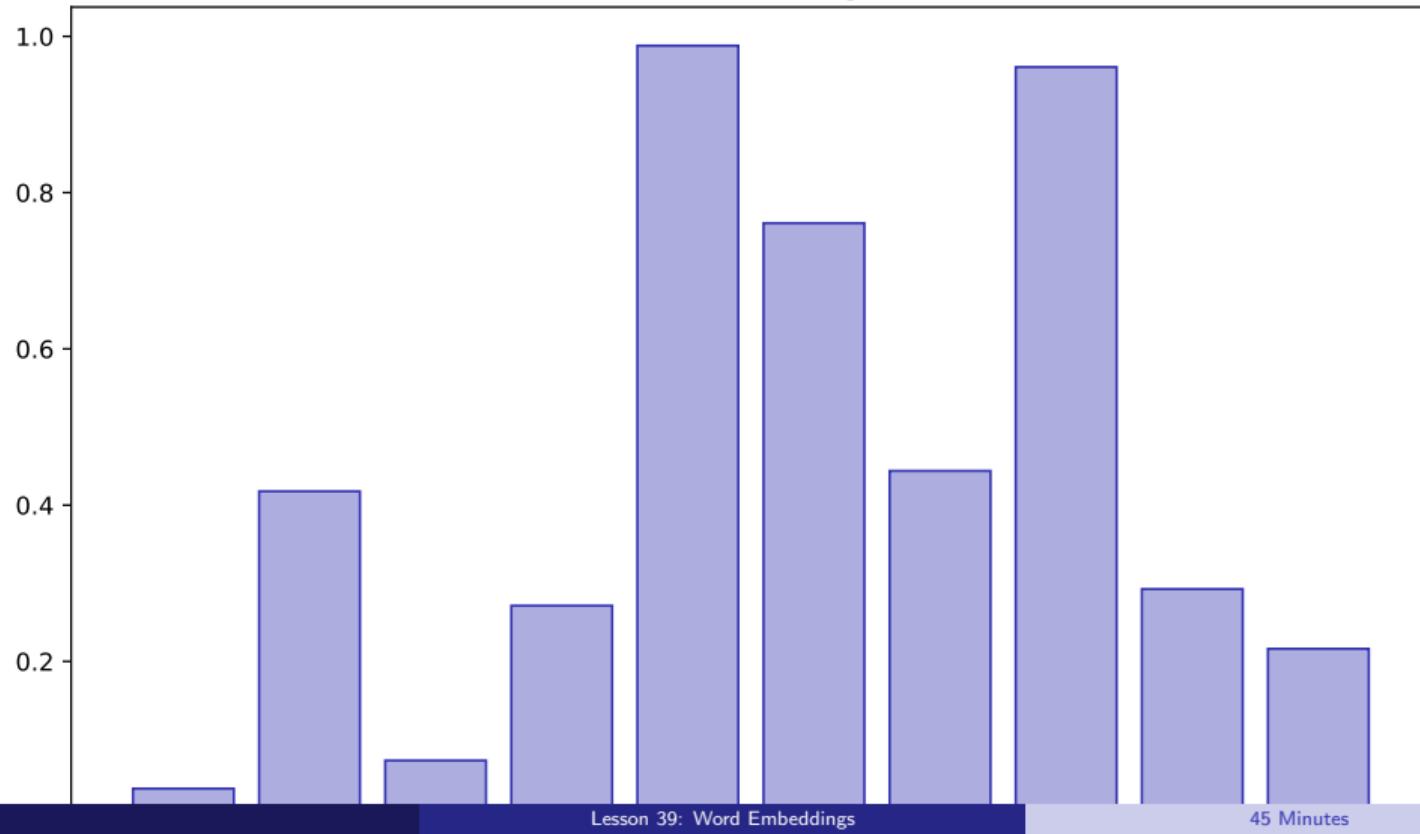
Embedding Concept



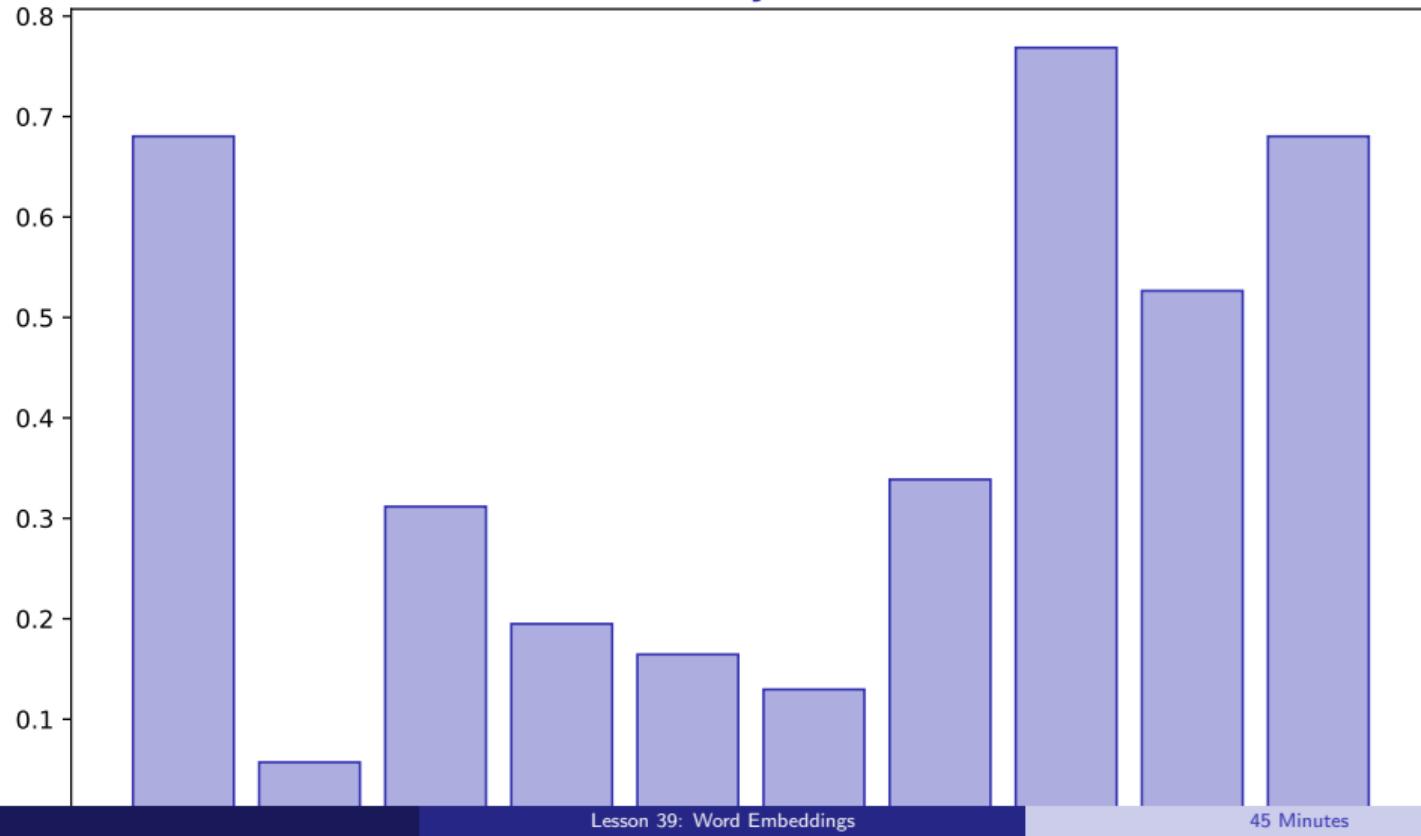
Word2Vec



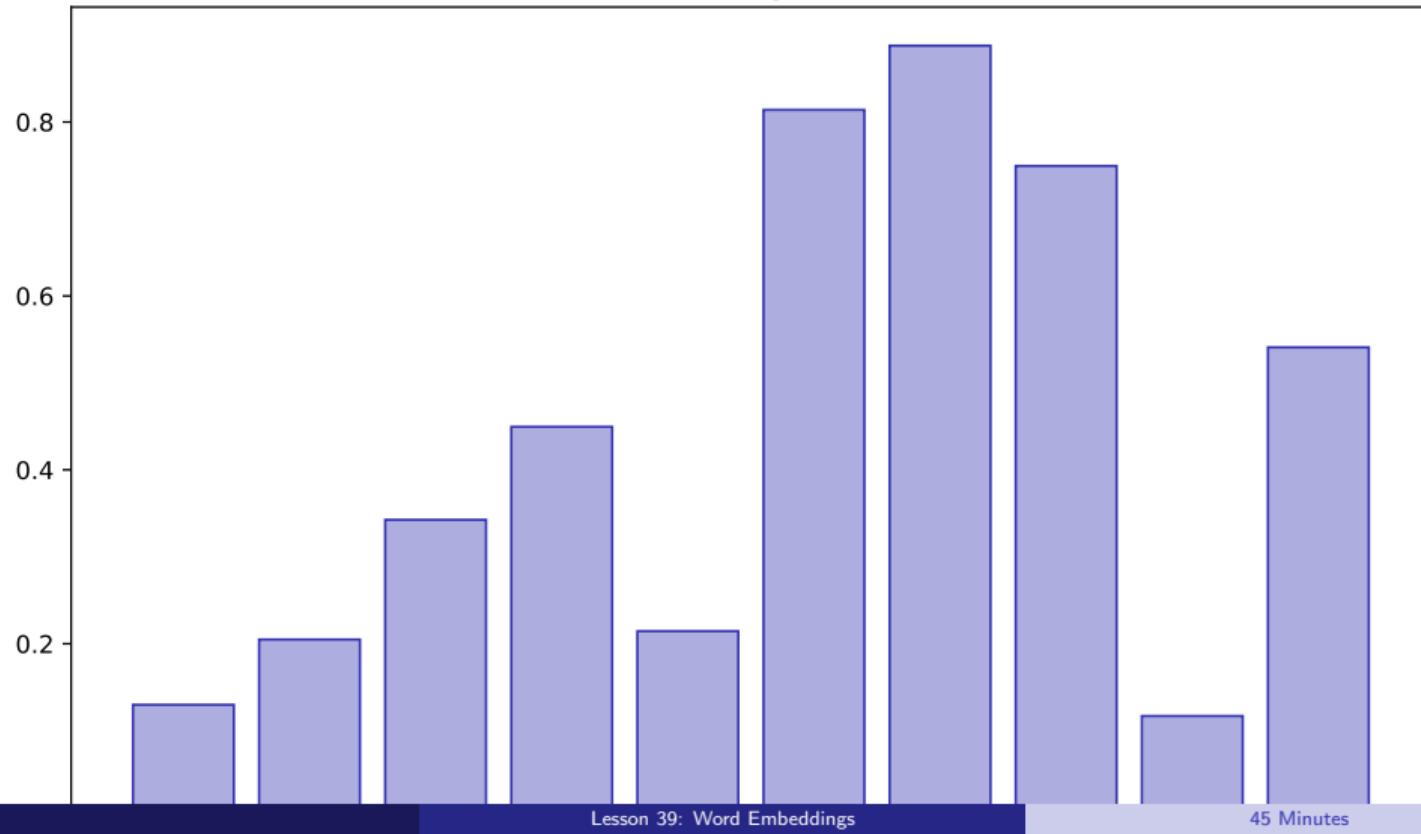
Gensim Usage



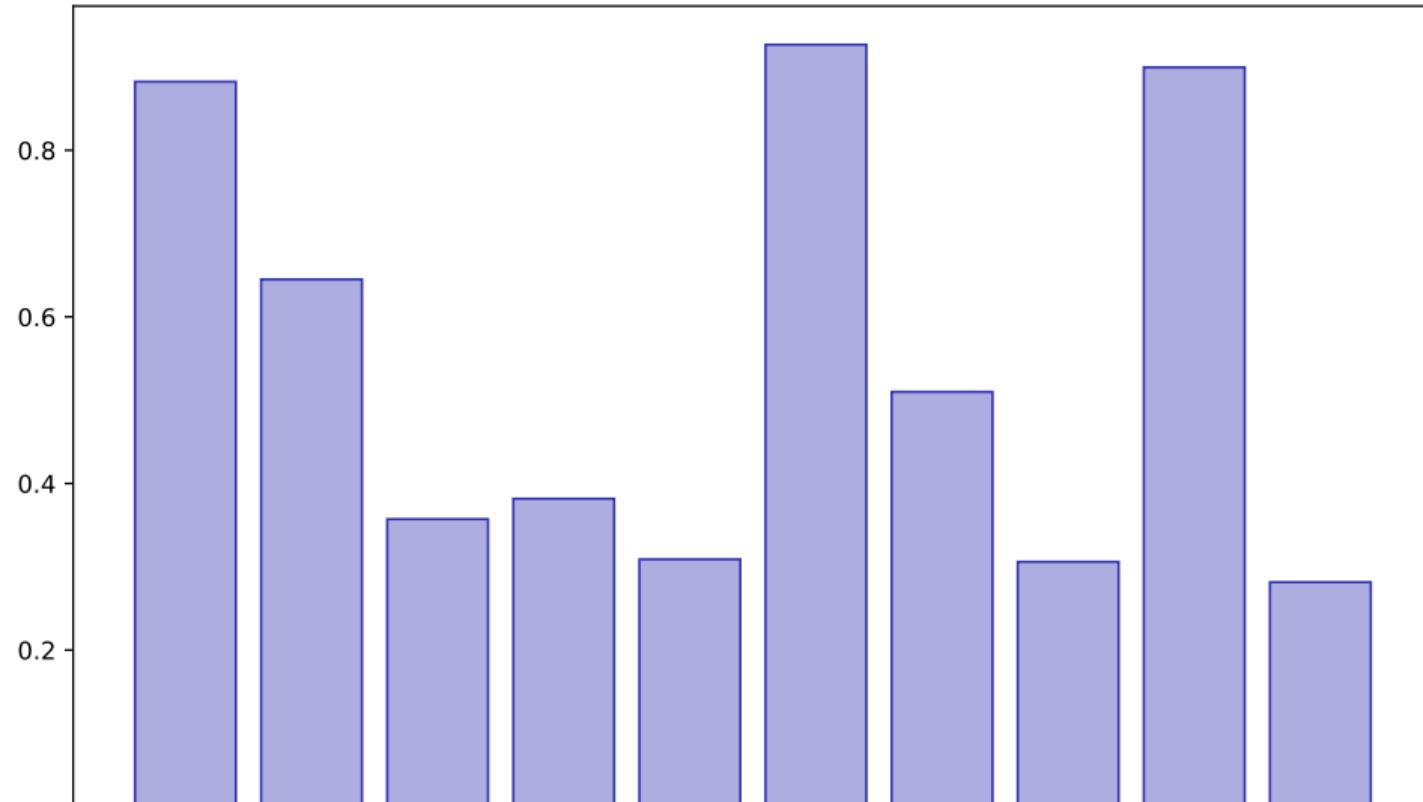
Similarity Search



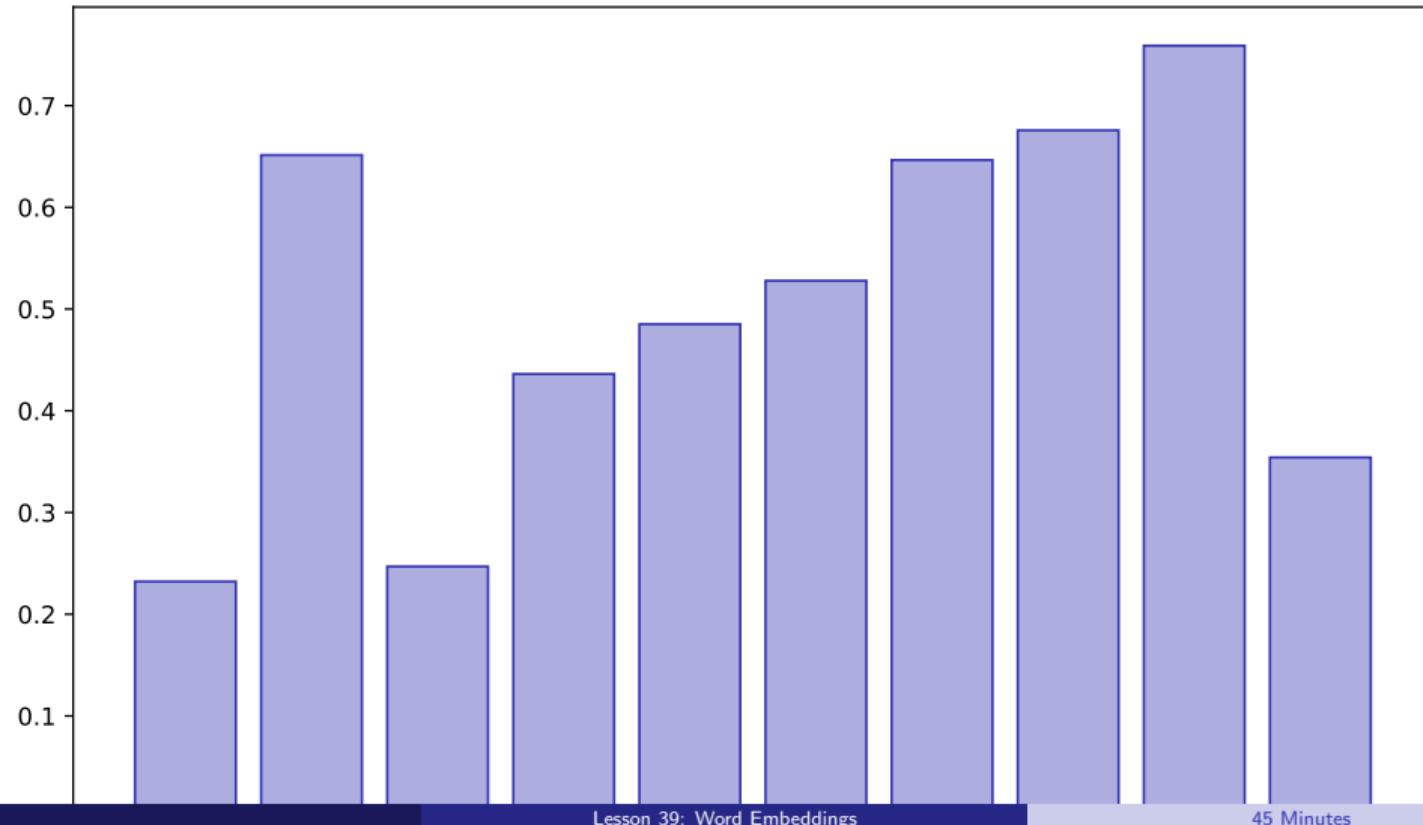
Analogies



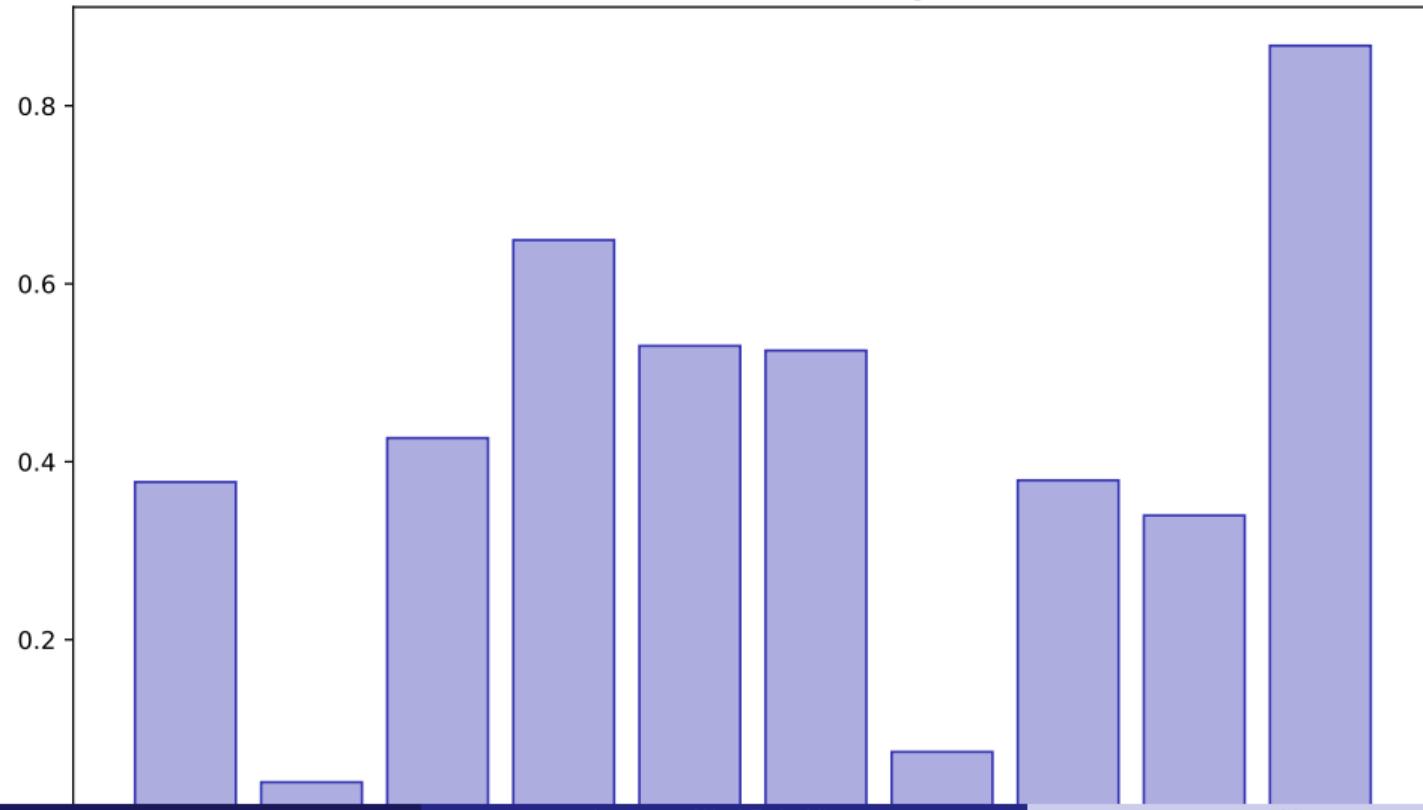
Document Vectors



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

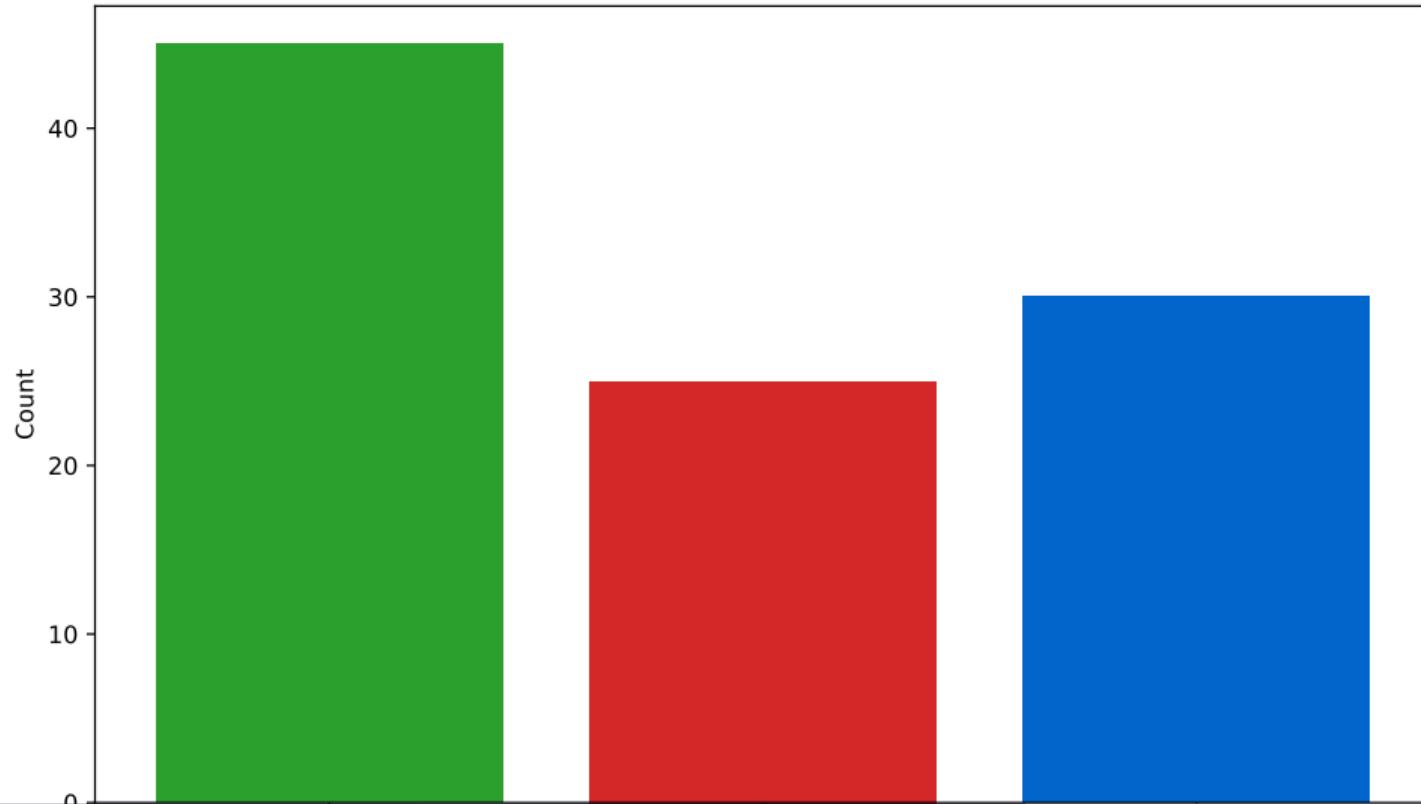
45 Minutes

After this lesson, you will be able to:

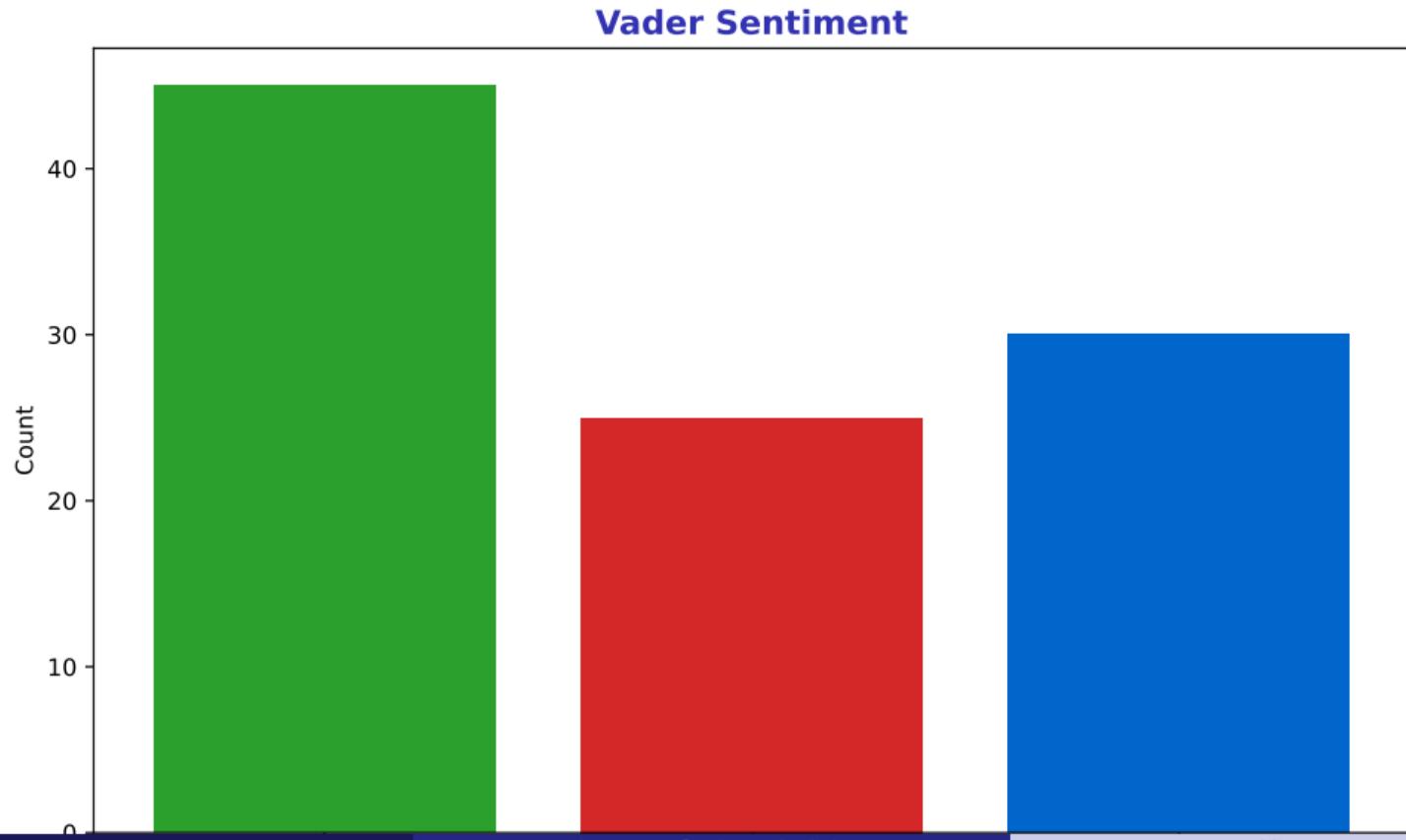
- Apply VADER sentiment
- Use FinBERT for finance
- Build sentiment classifiers
- Analyze market sentiment

Building towards your final project

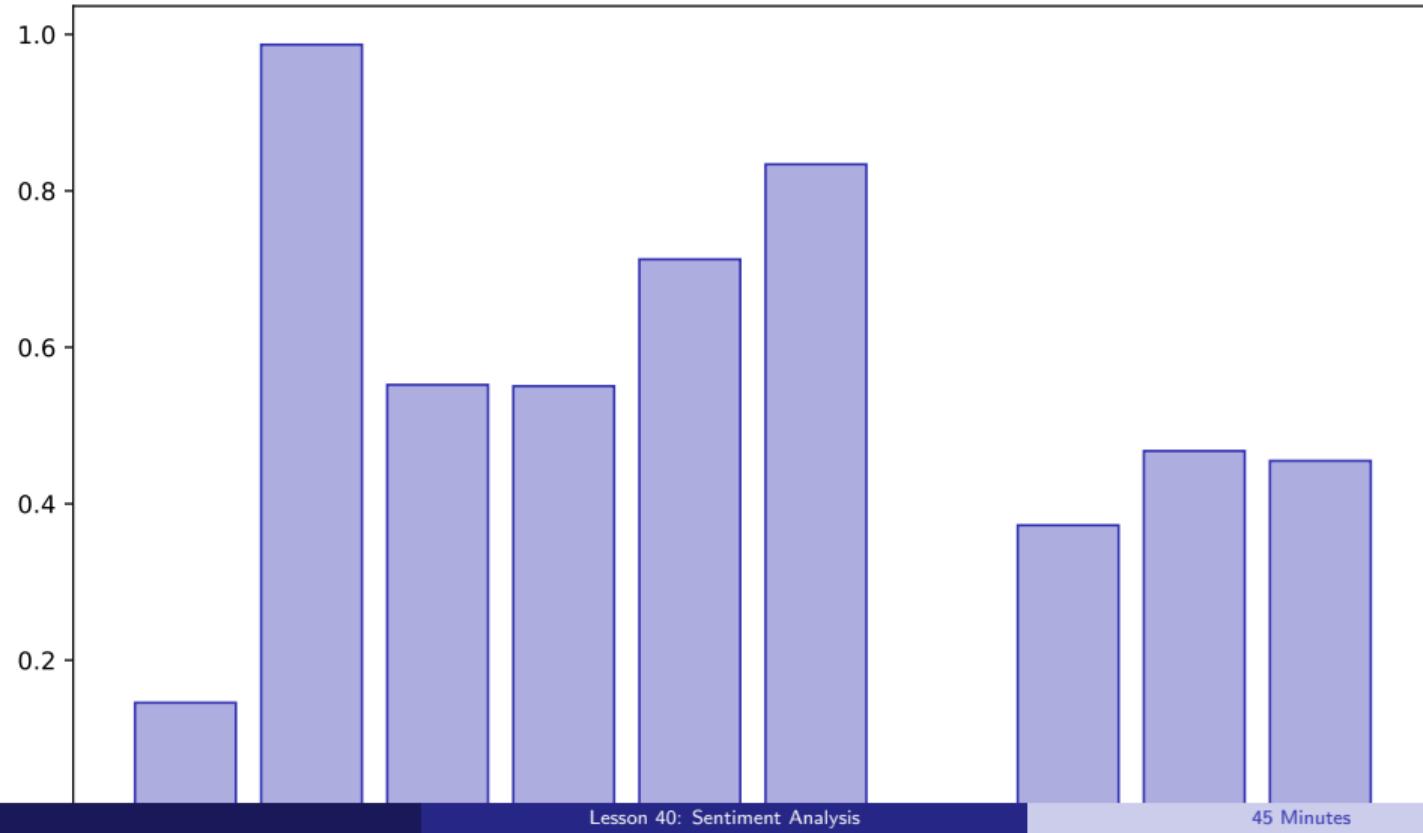
Sentiment Concept



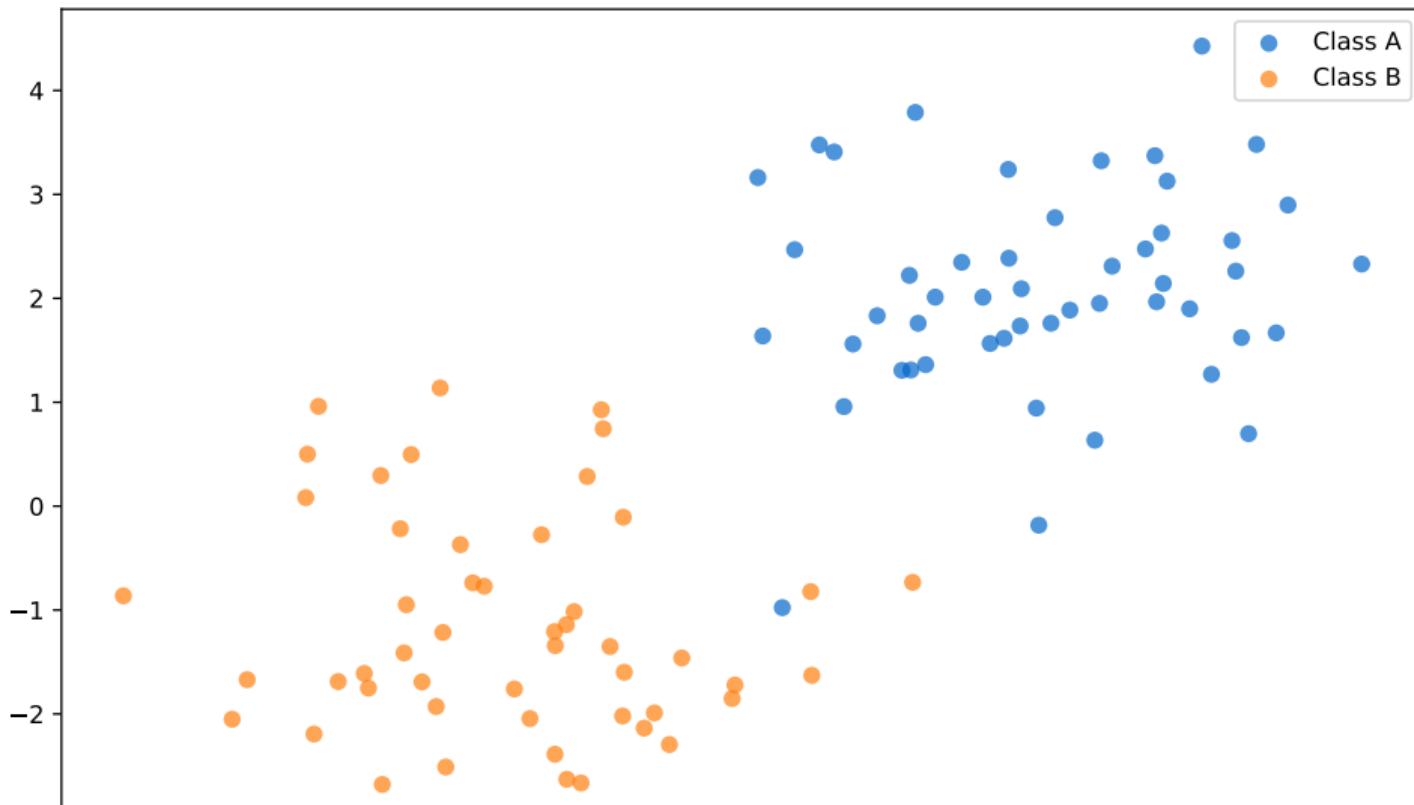
Vader Sentiment

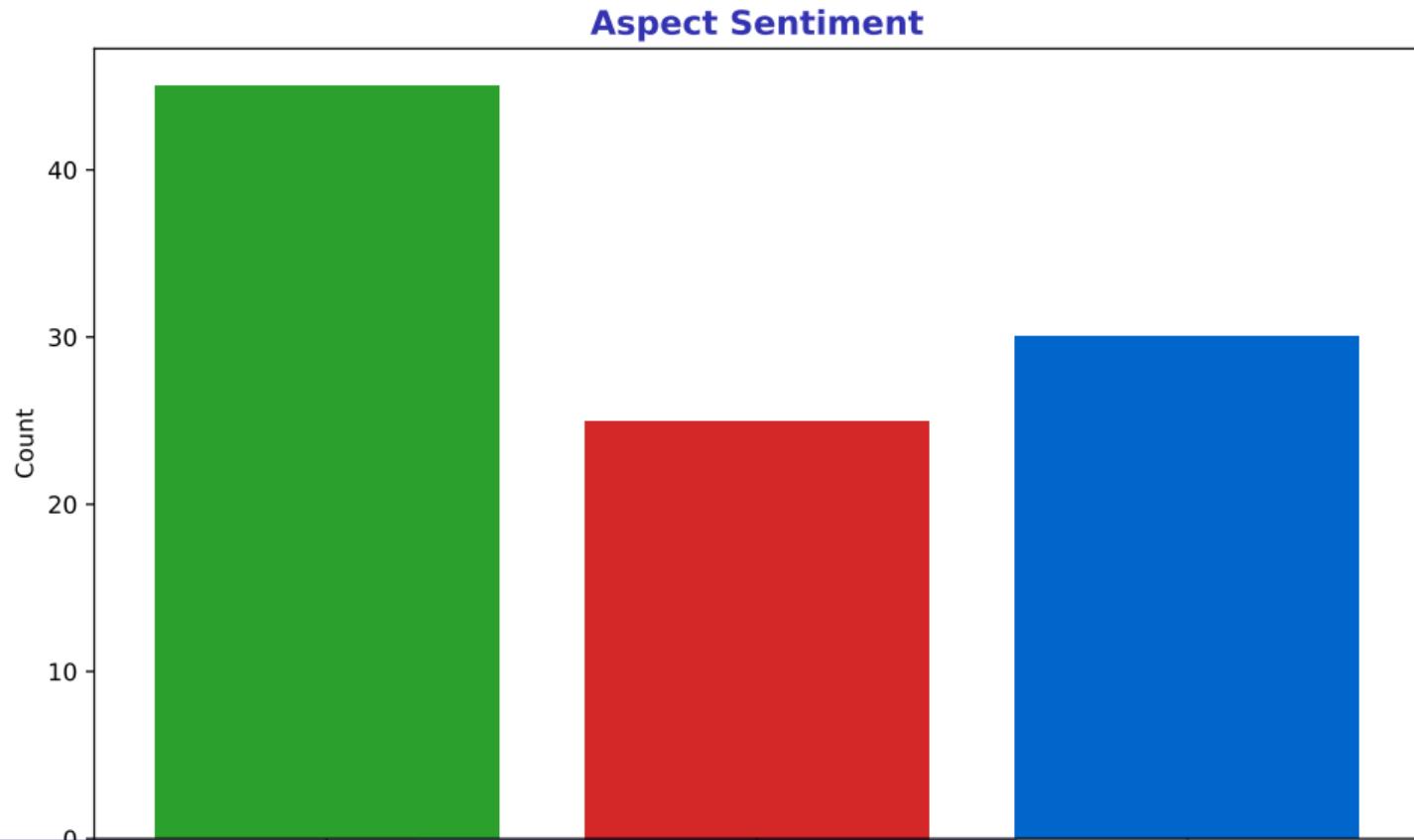


Finbert Intro

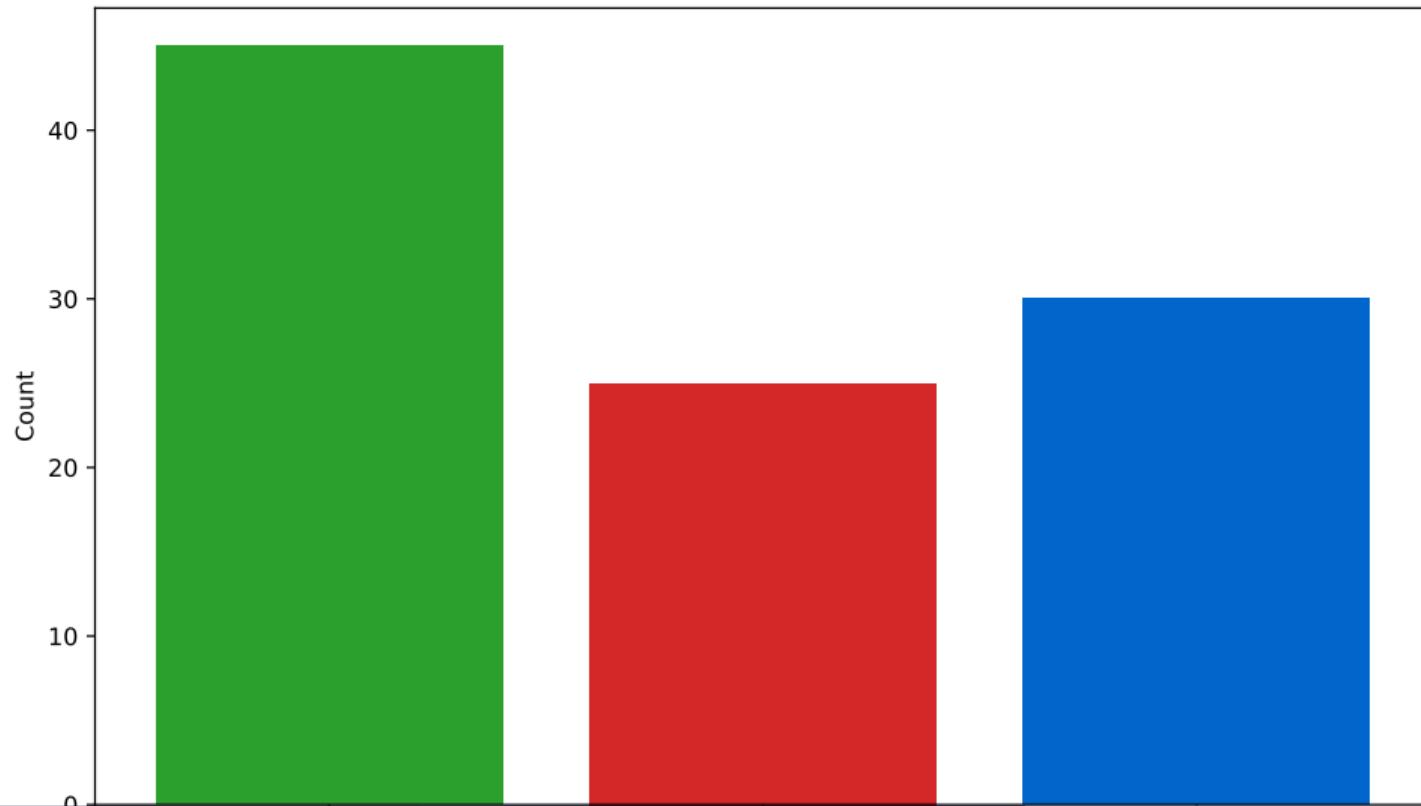


Sentiment Classification

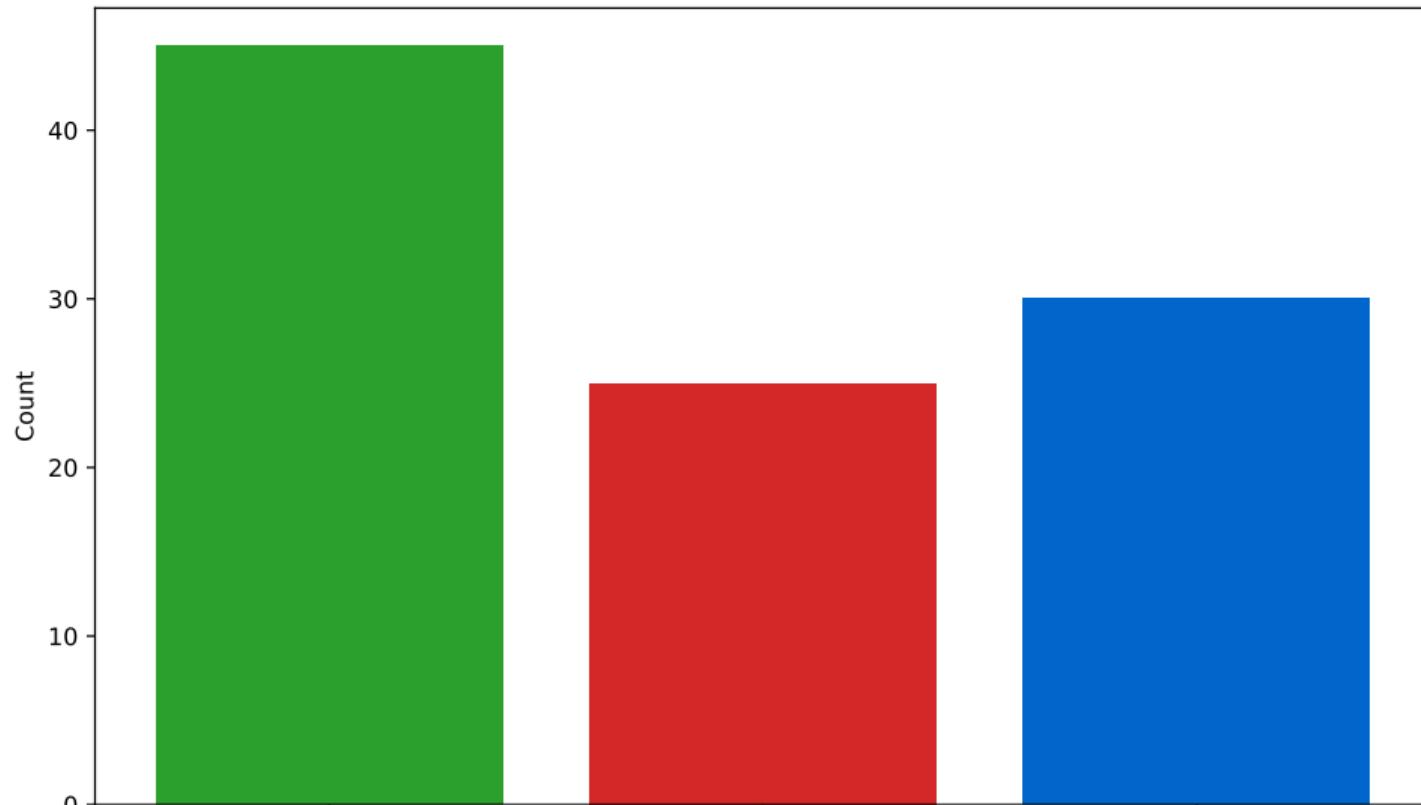




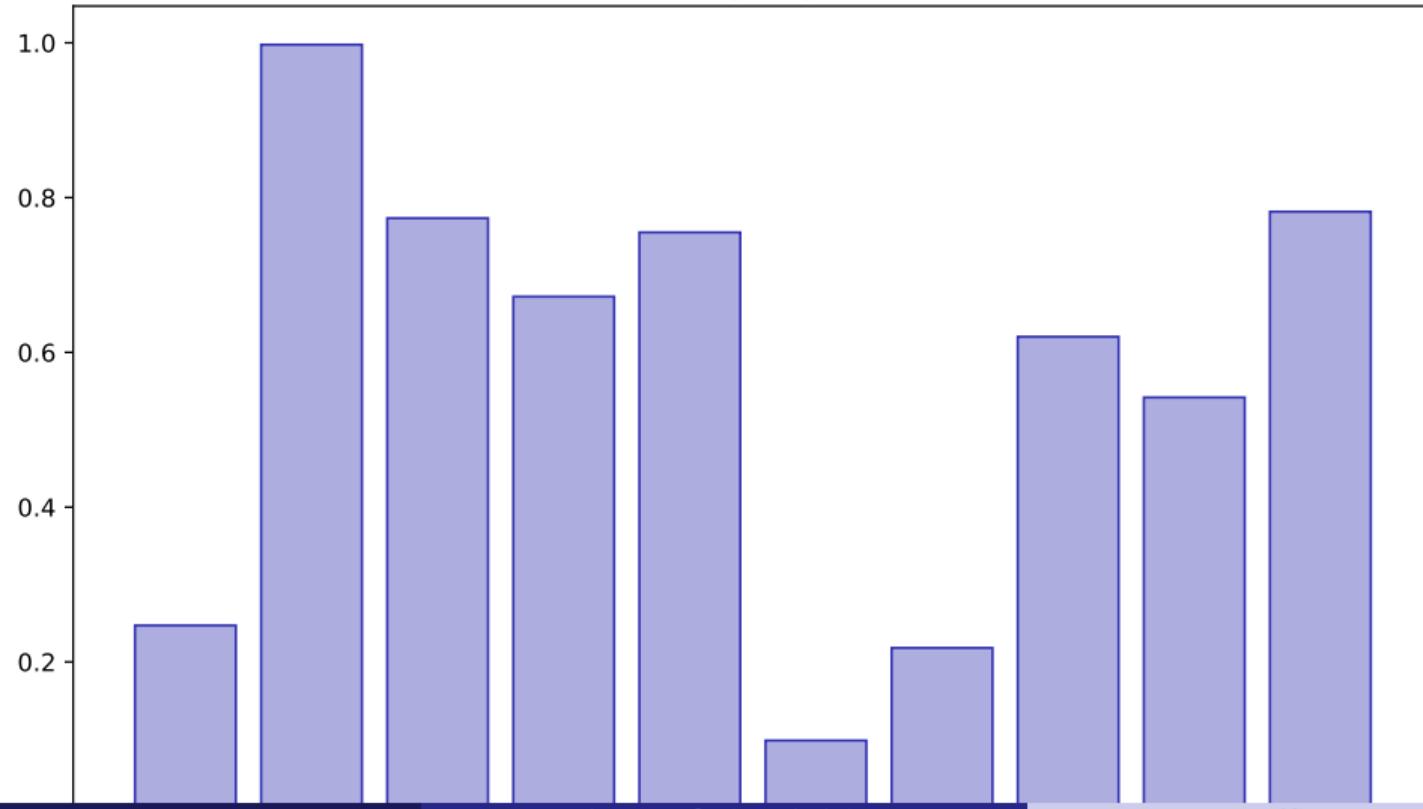
Time Series Sentiment



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

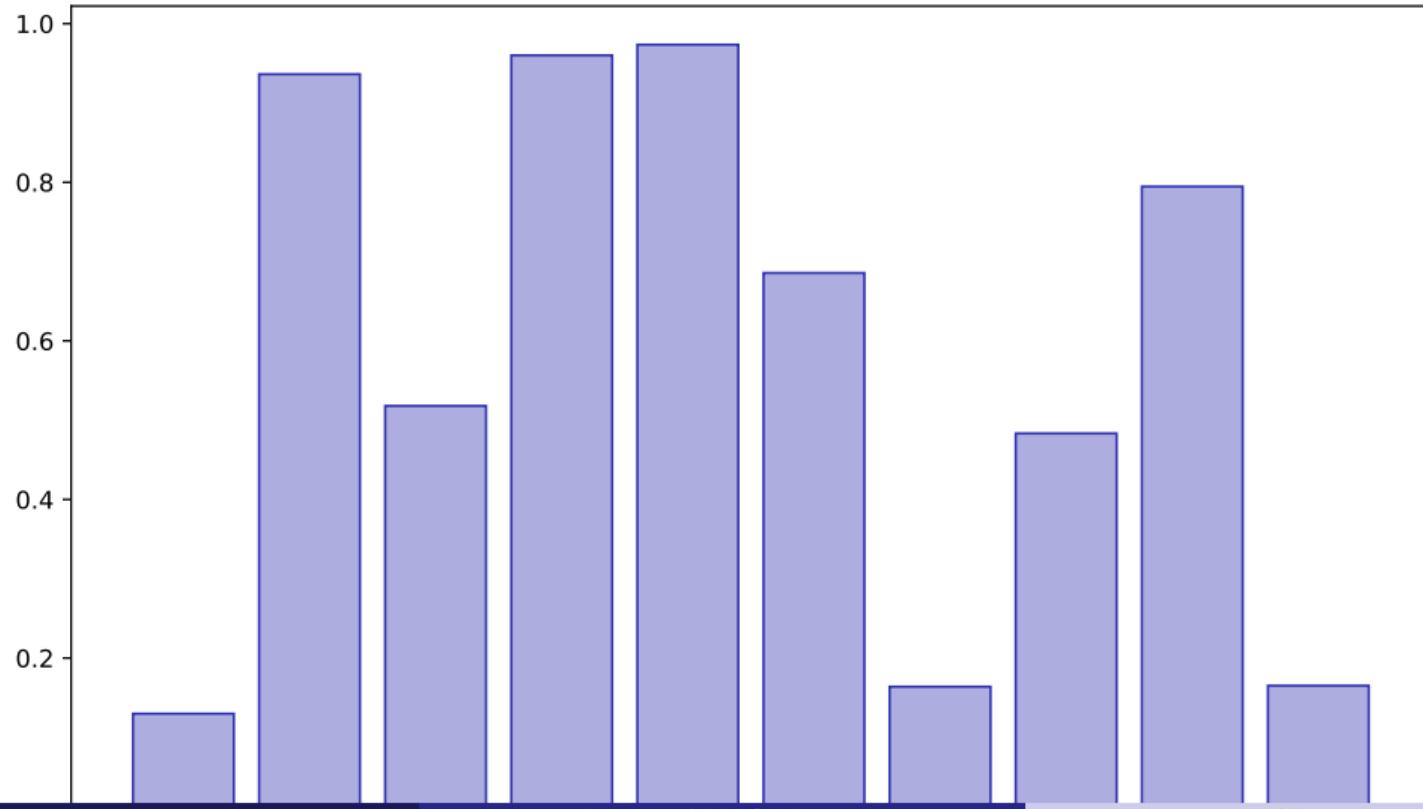
Learning Objectives

After this lesson, you will be able to:

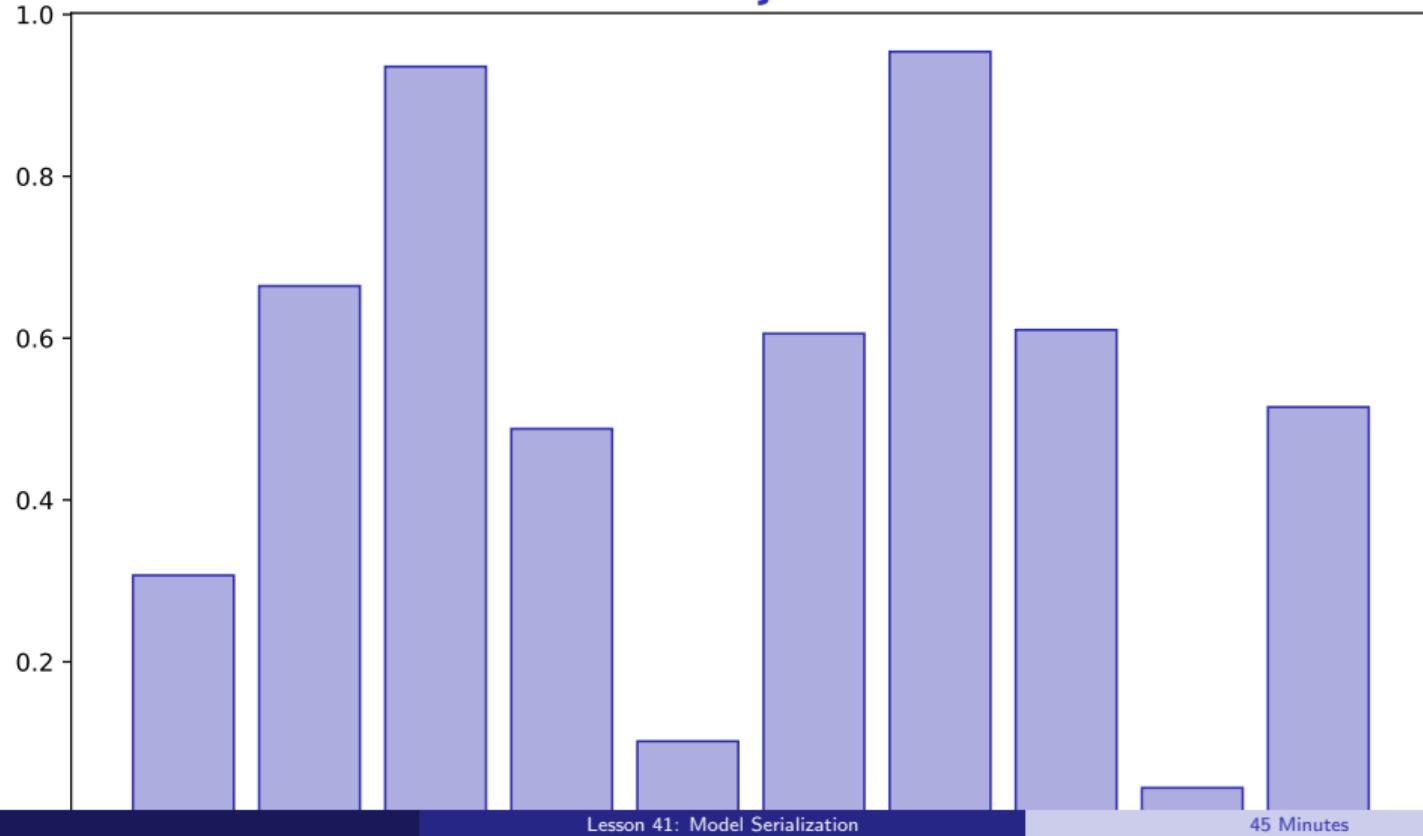
- Save models with joblib
- Load and predict
- Version models
- Prepare for deployment

Building towards your final project

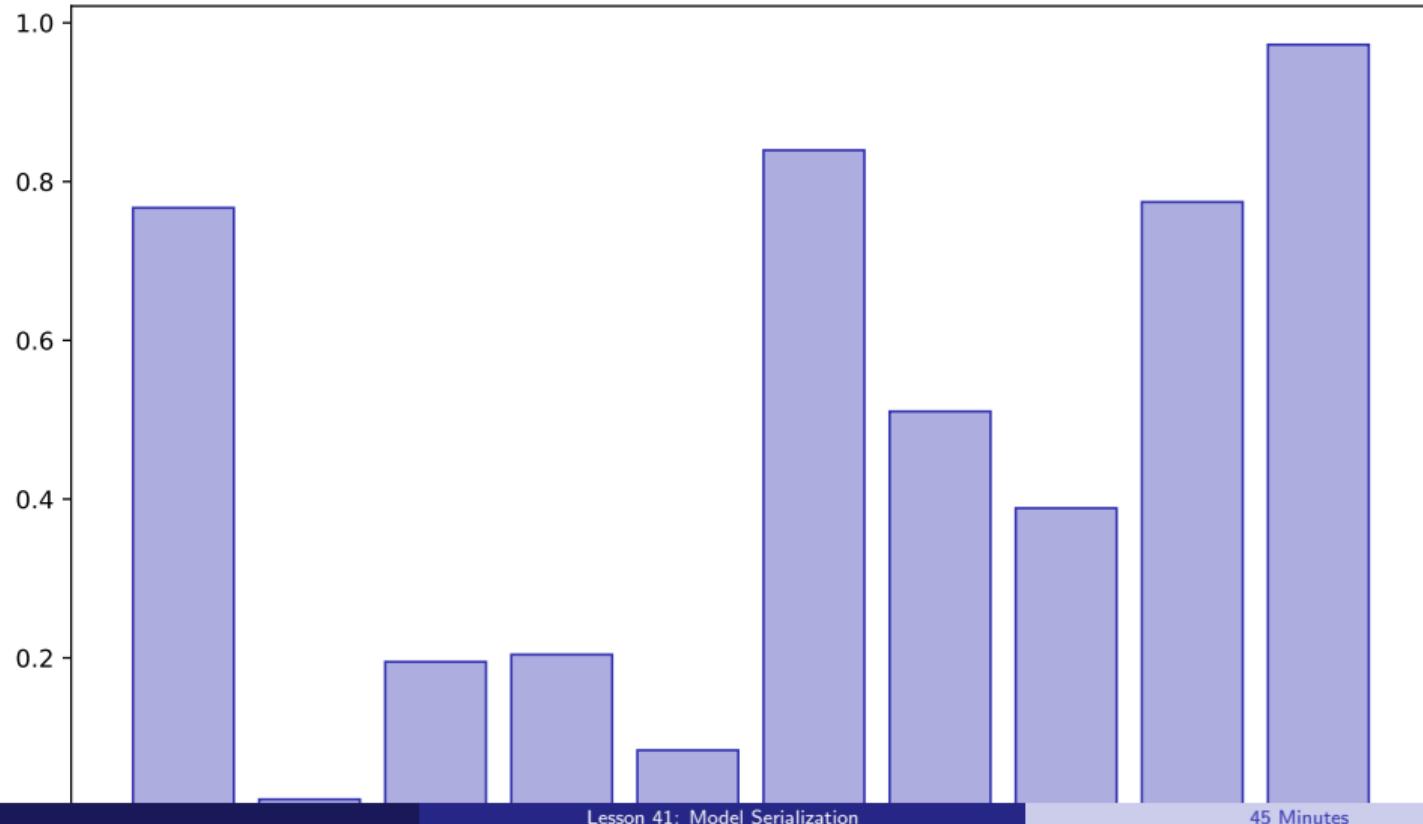
Serialization Concept



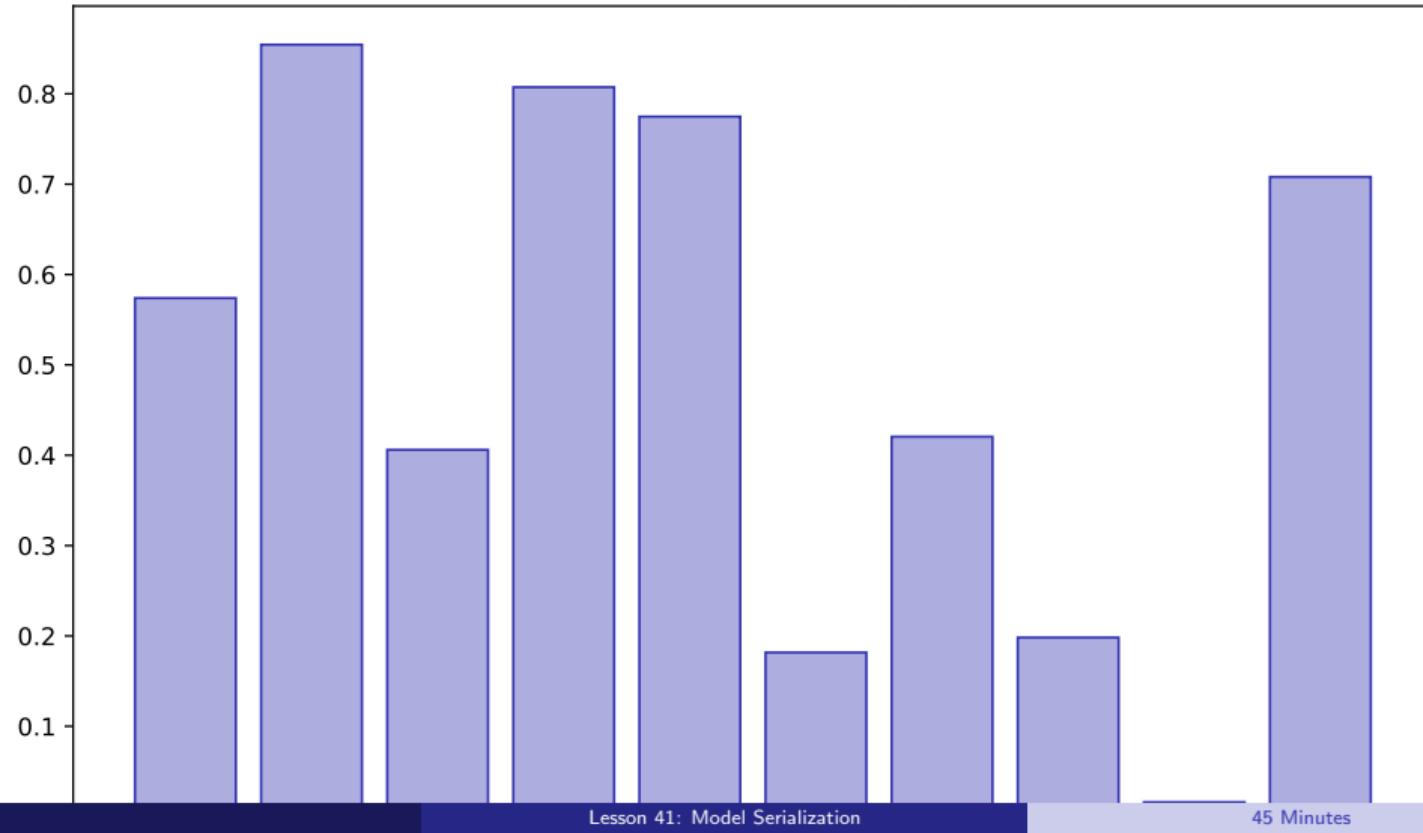
Pickle Joblib



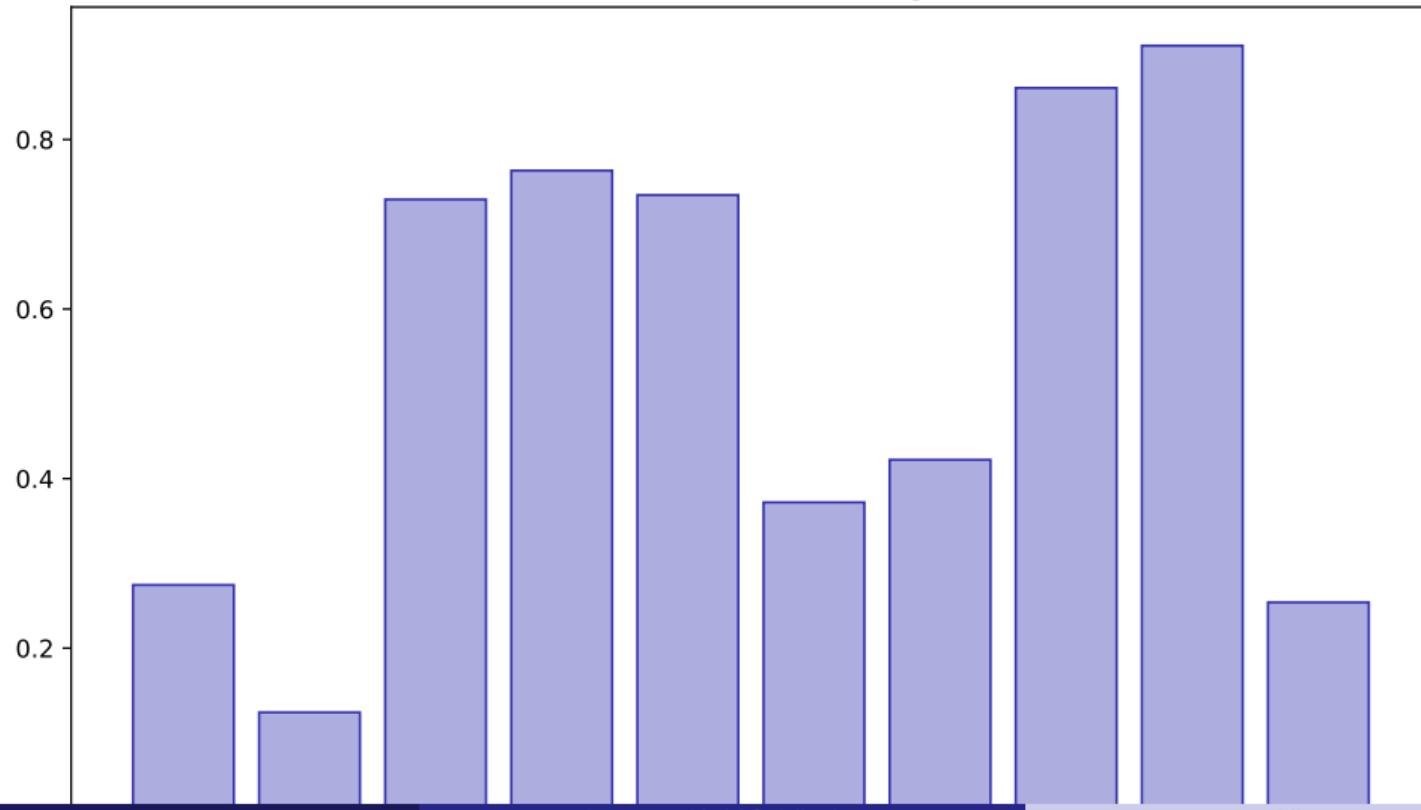
Save Load Workflow



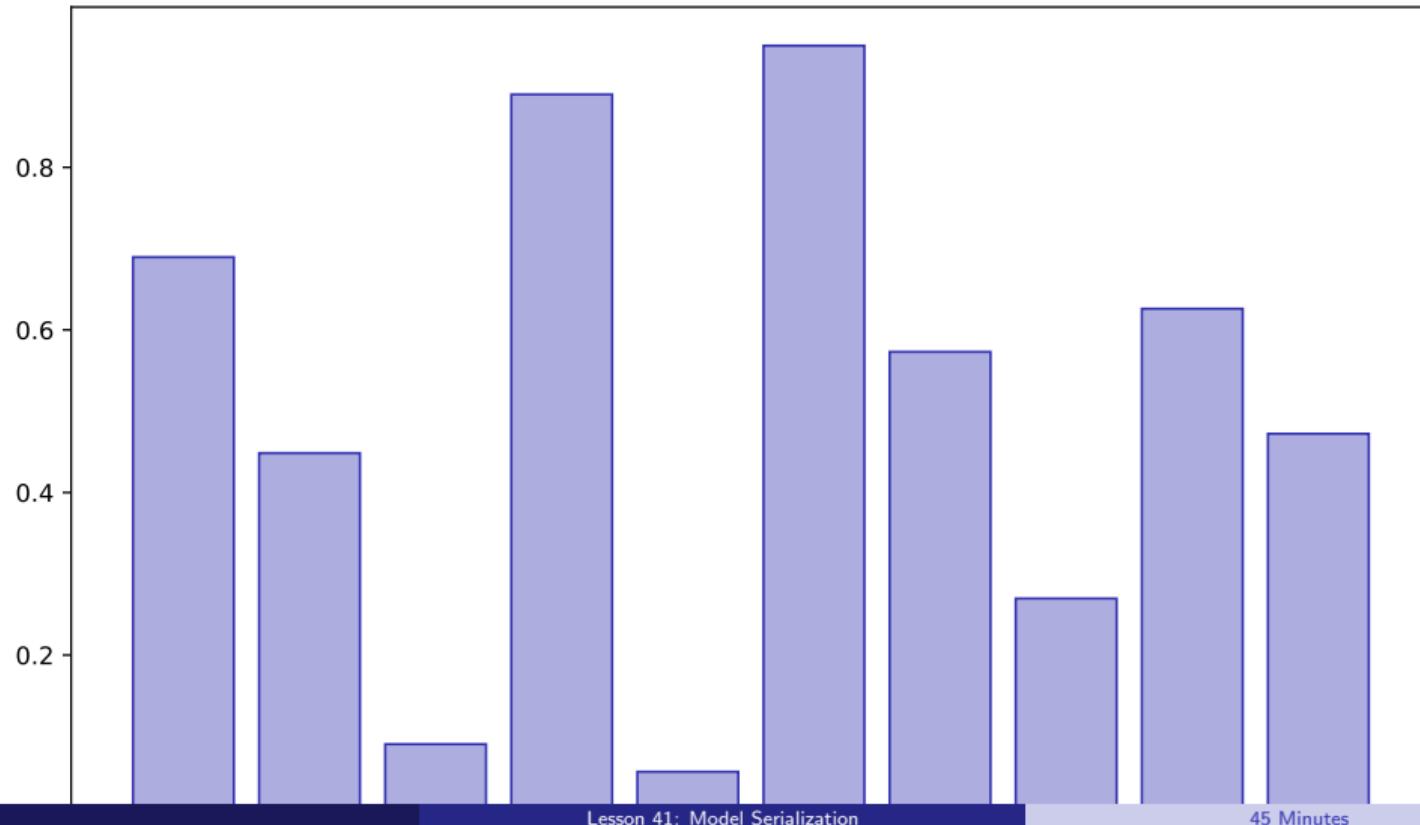
Model Versioning



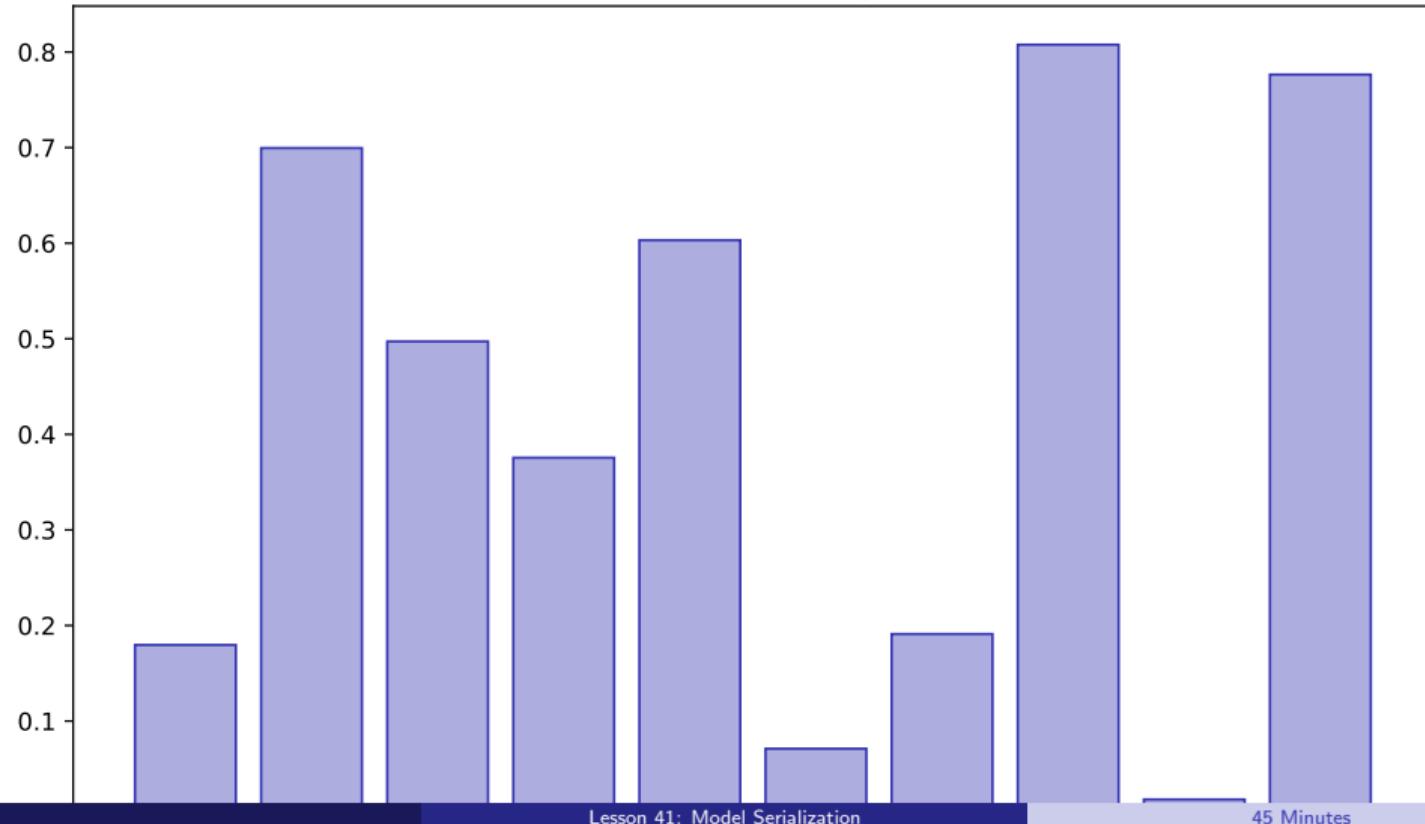
Metadata Tracking



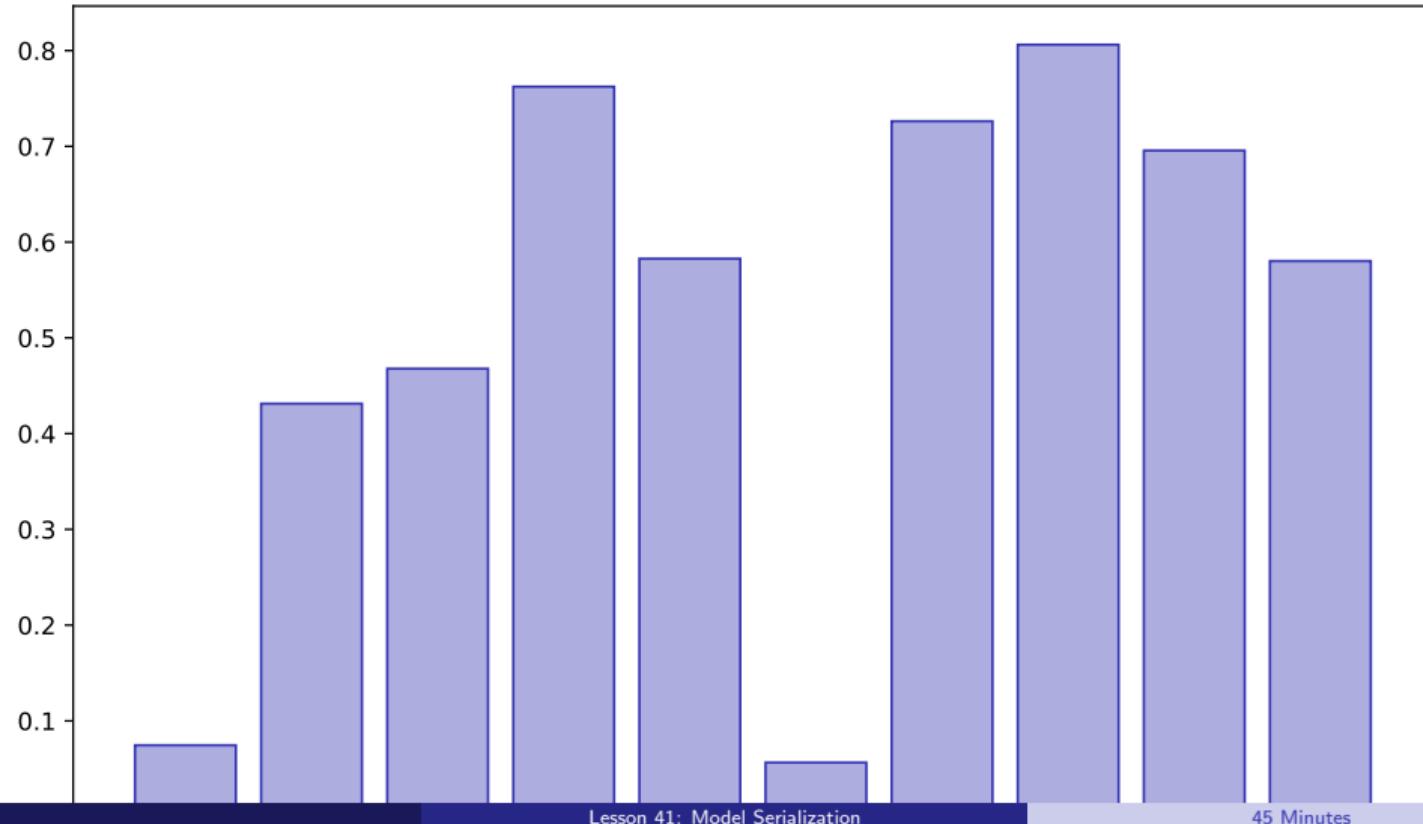
Sklearn Persistence



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

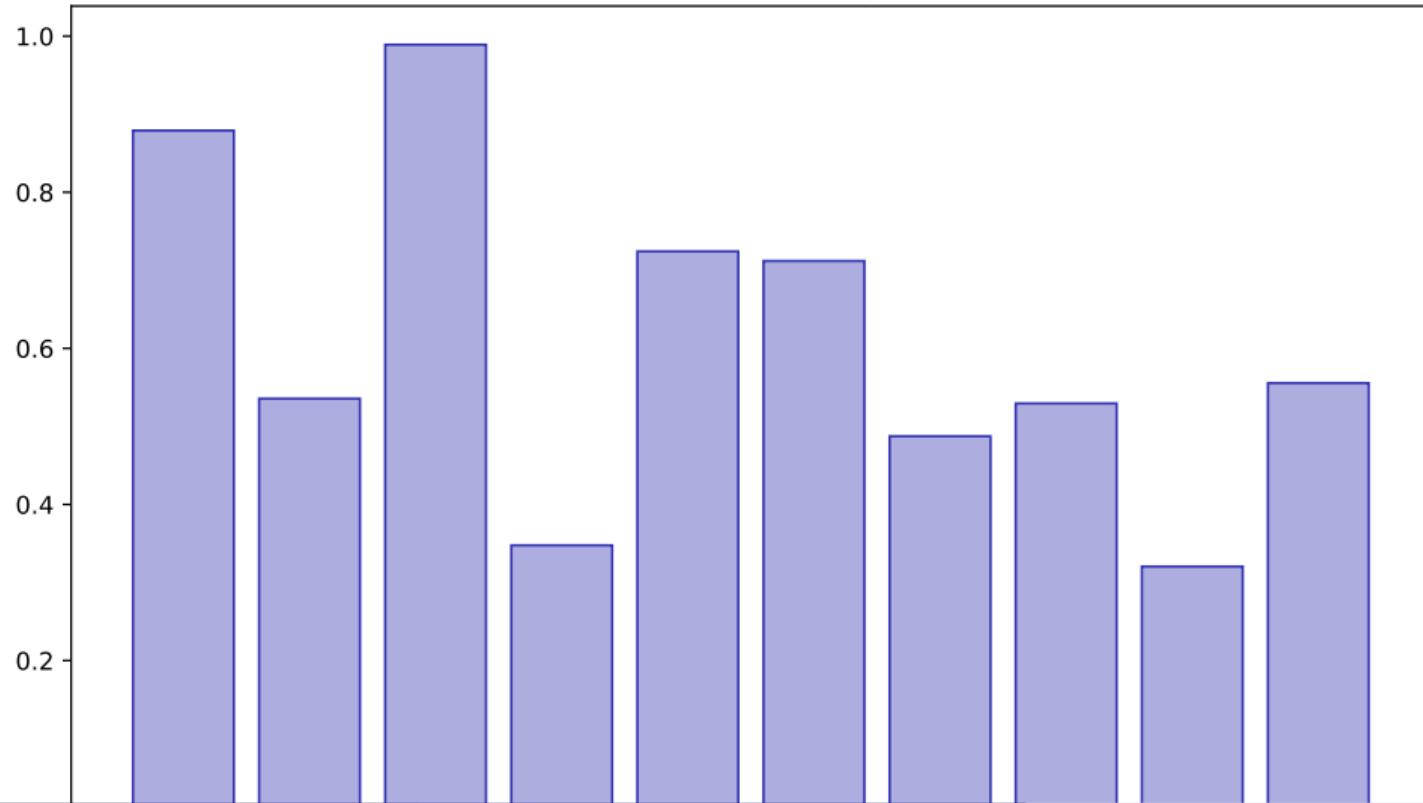
Api Concept



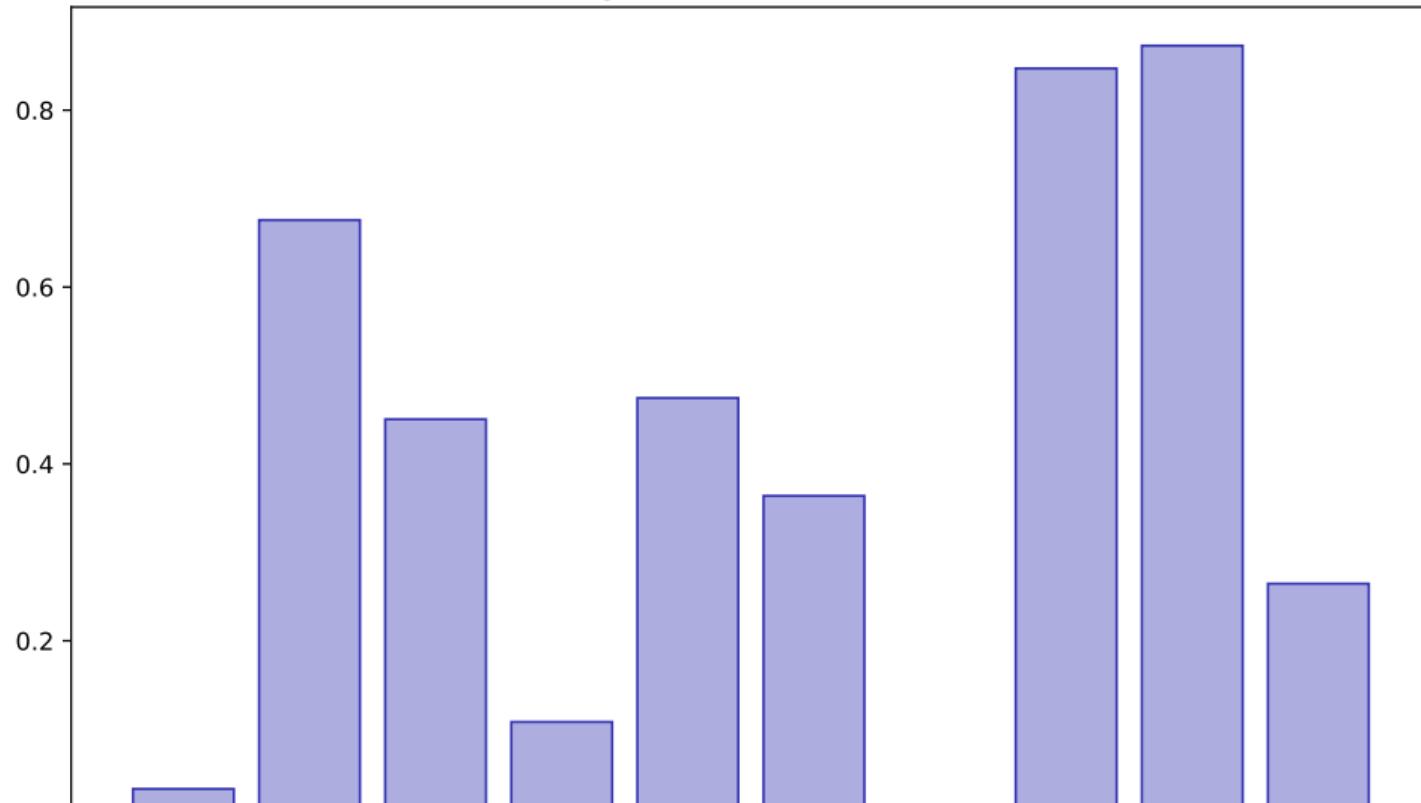
Fastapi Basics



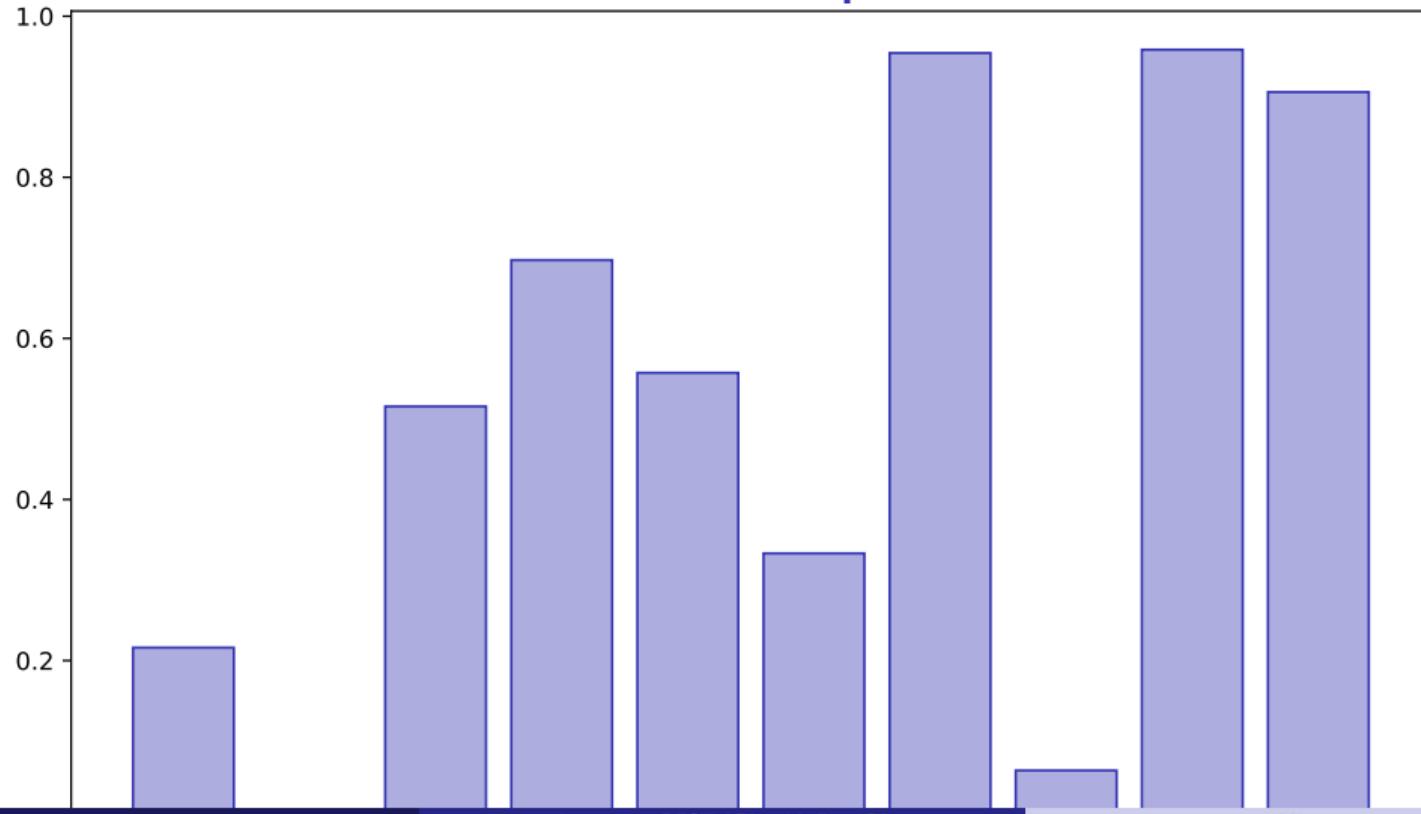
Endpoint Design



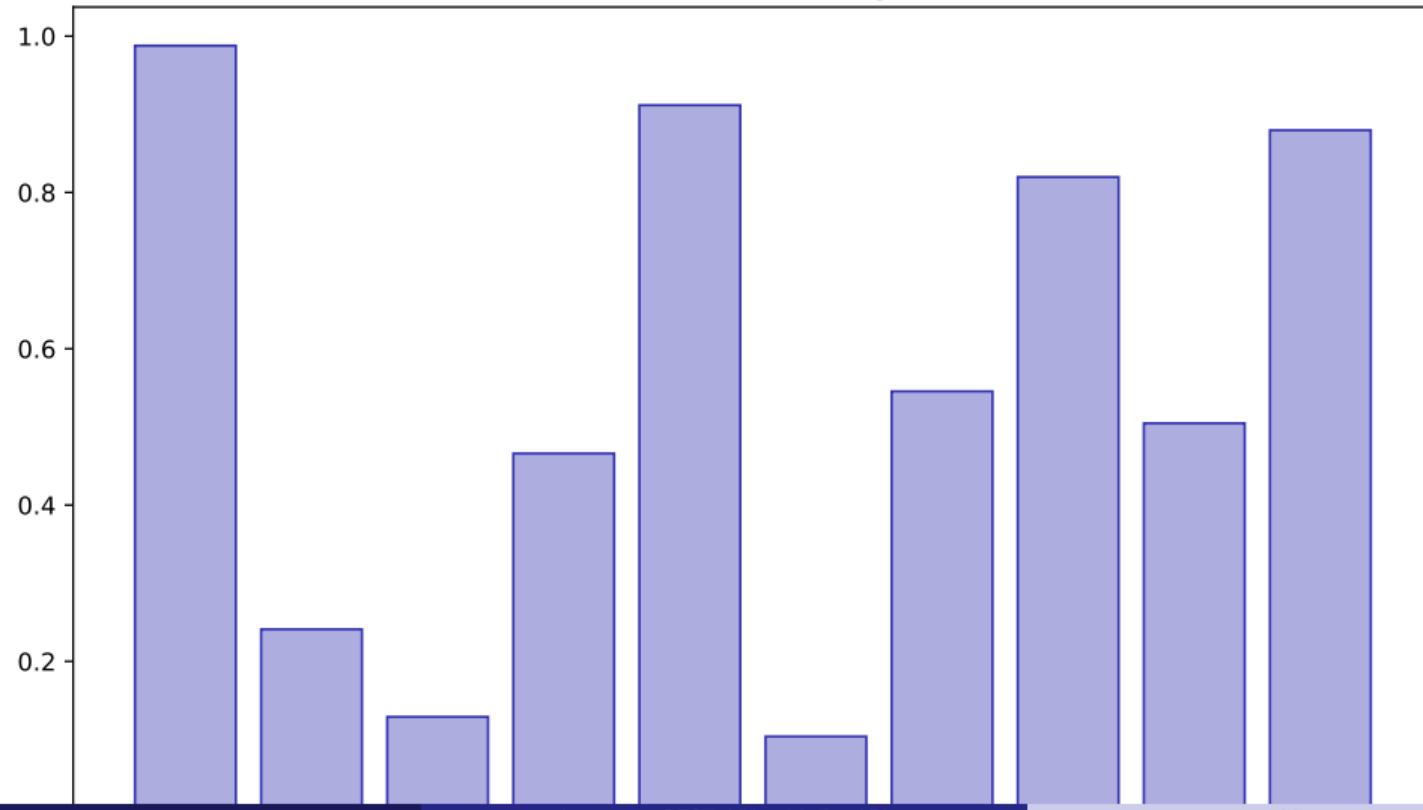
Pydantic Schemas



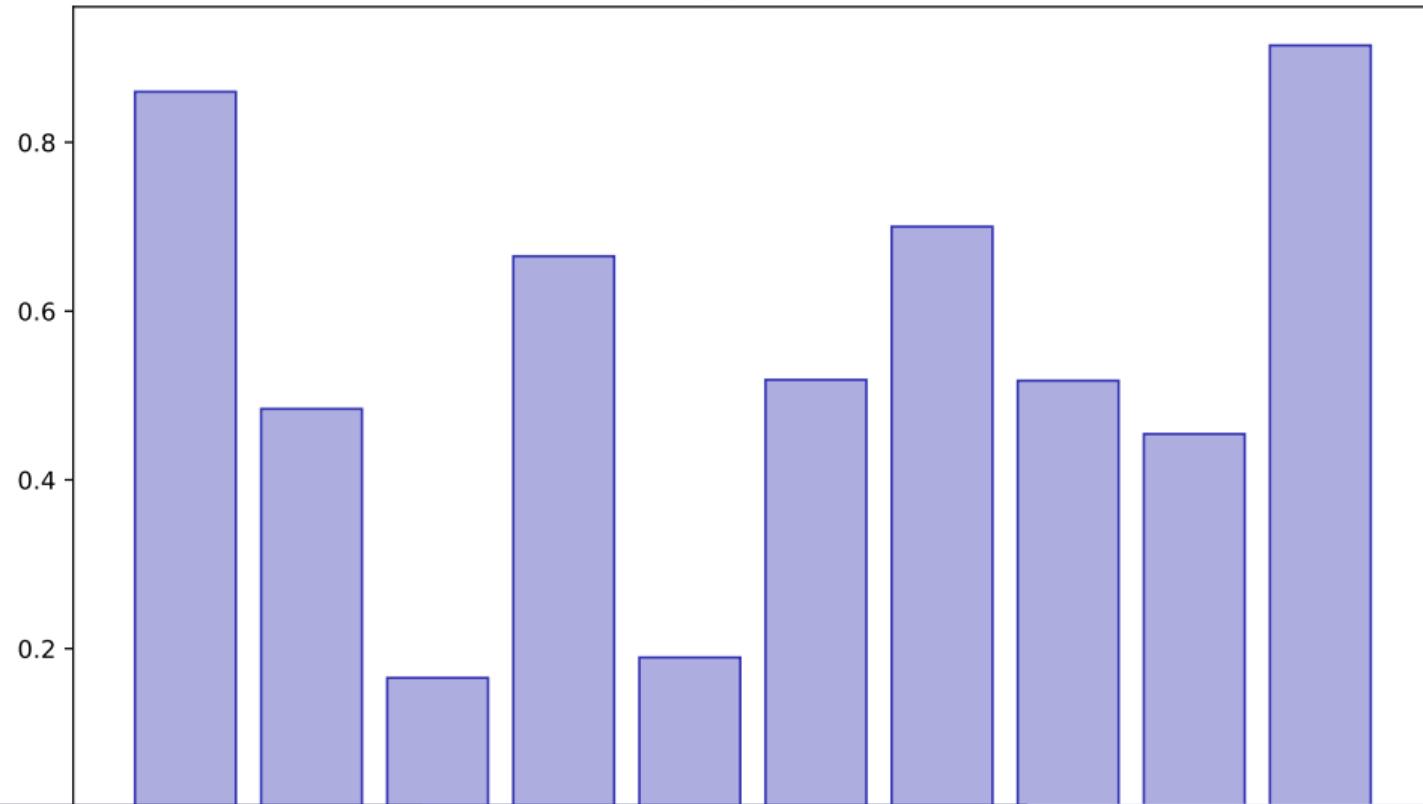
Prediction Endpoint



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

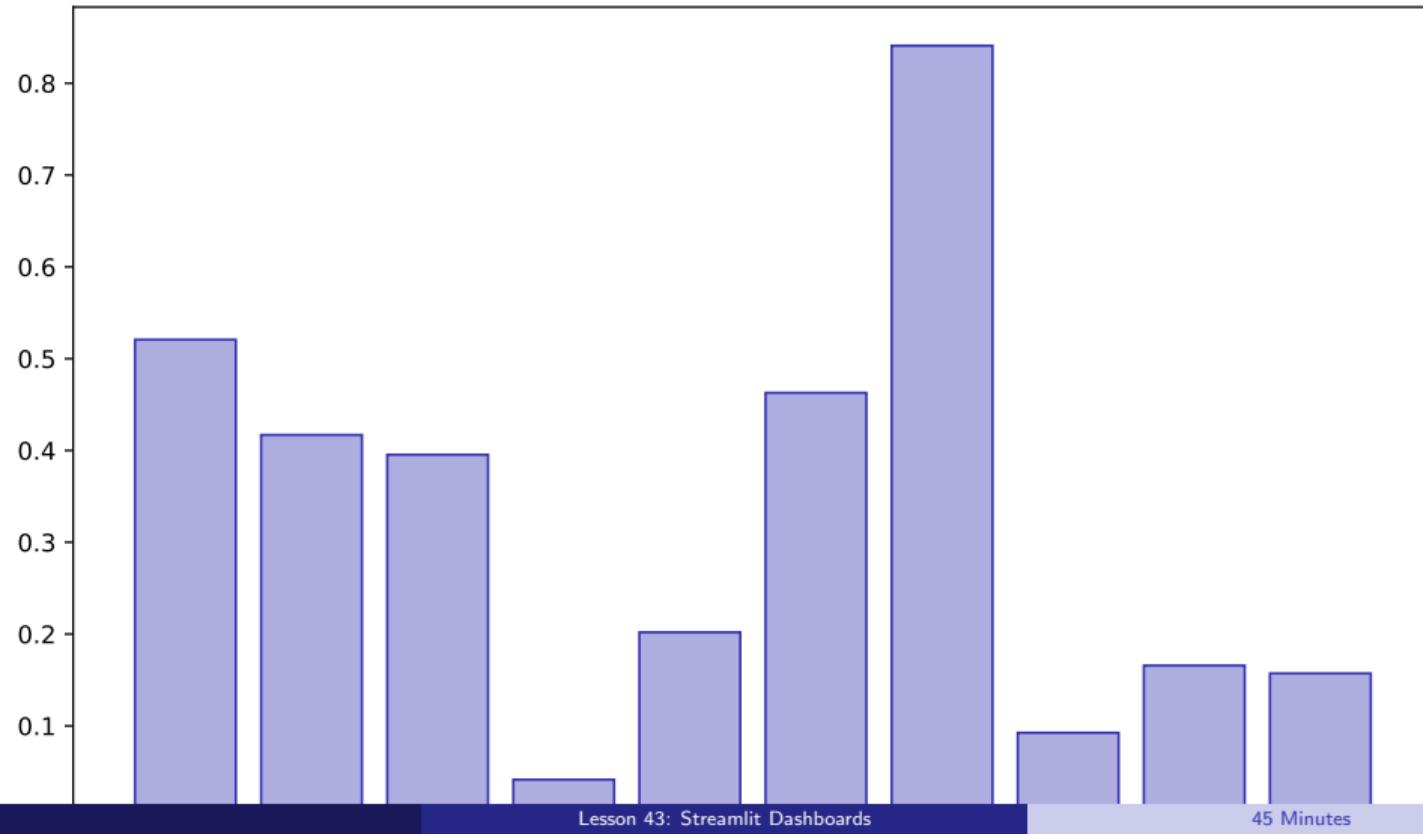
Learning Objectives

After this lesson, you will be able to:

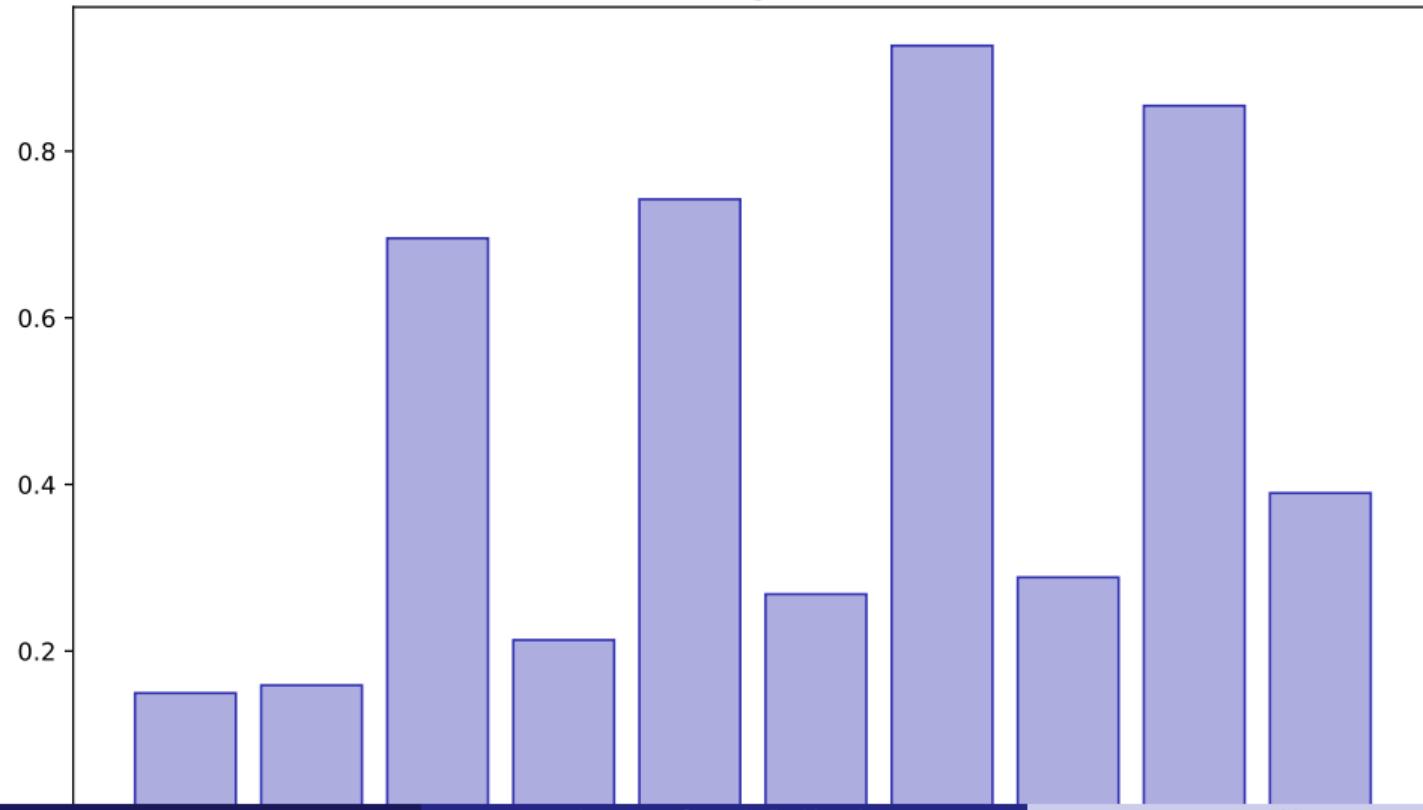
- Build interactive apps
- Add widgets and inputs
- Display predictions
- Create financial dashboards

Building towards your final project

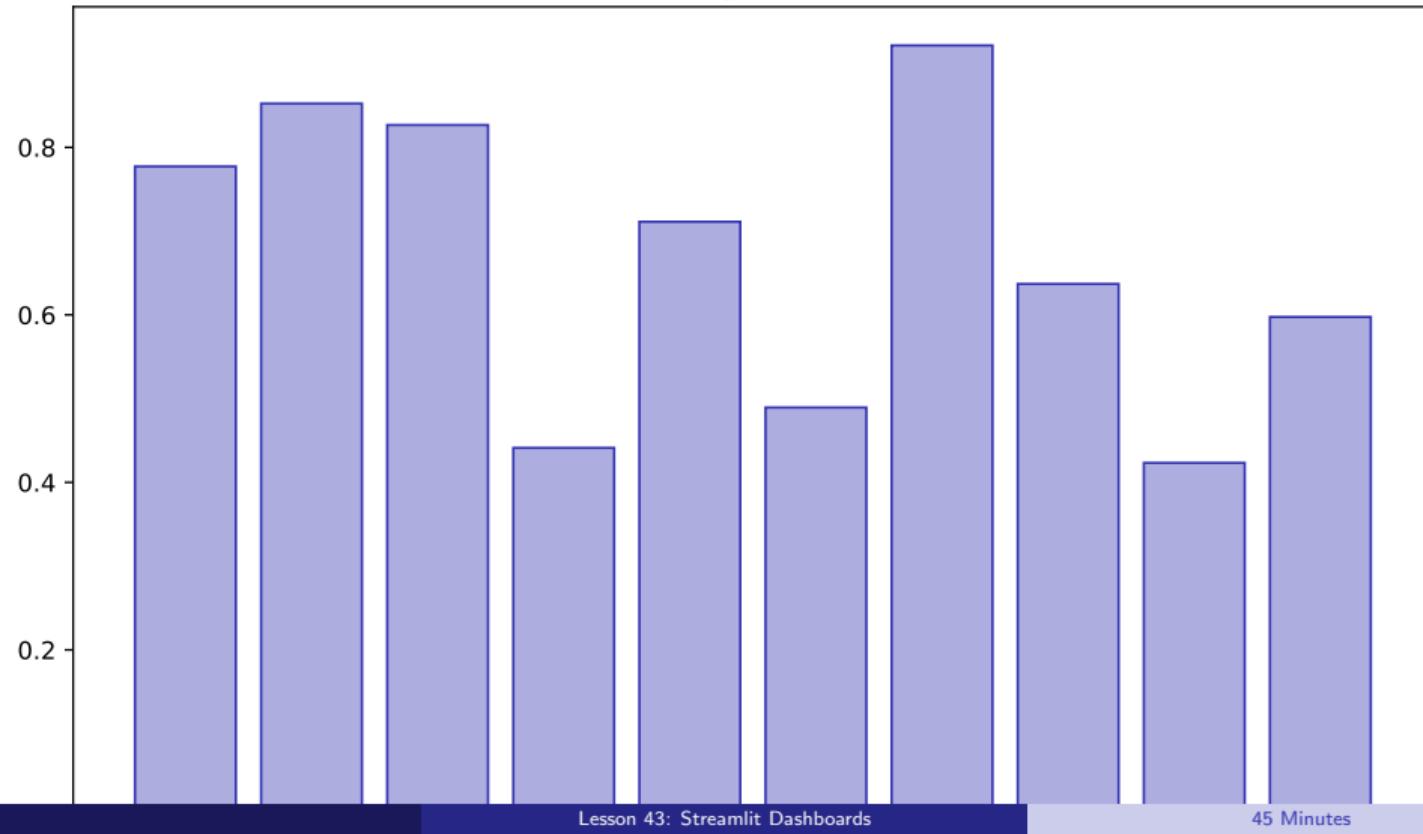
Streamlit Intro



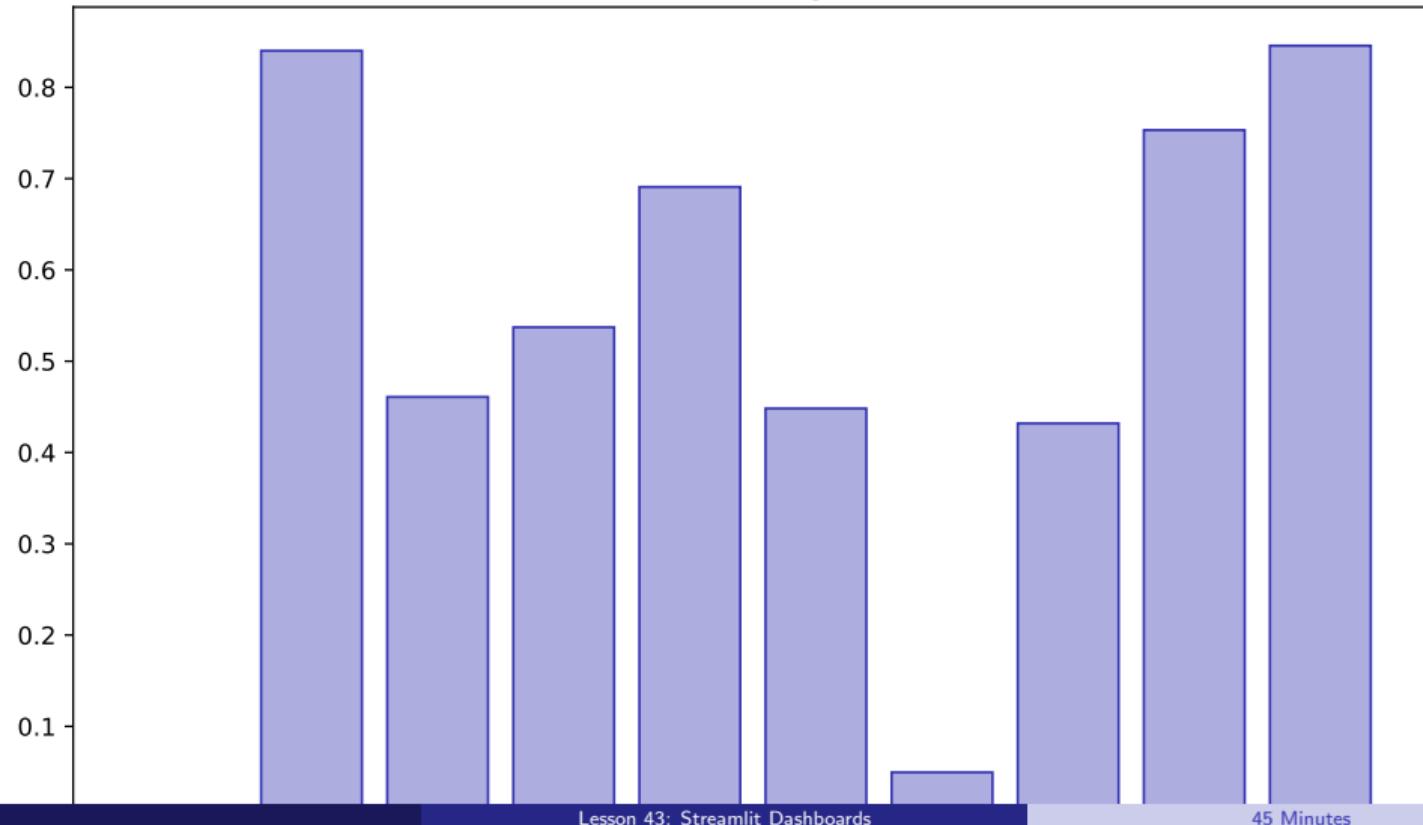
Widgets



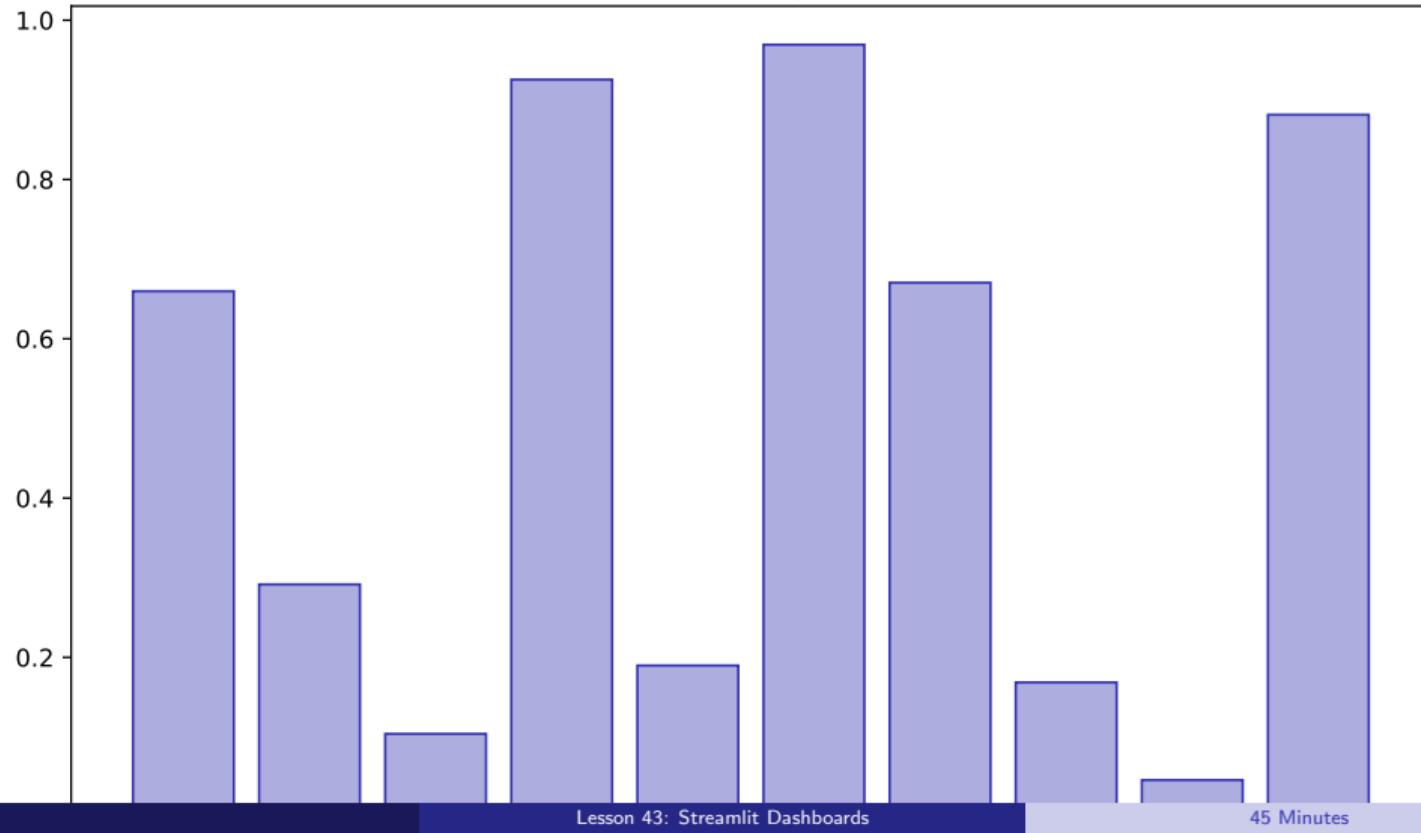
Layouts



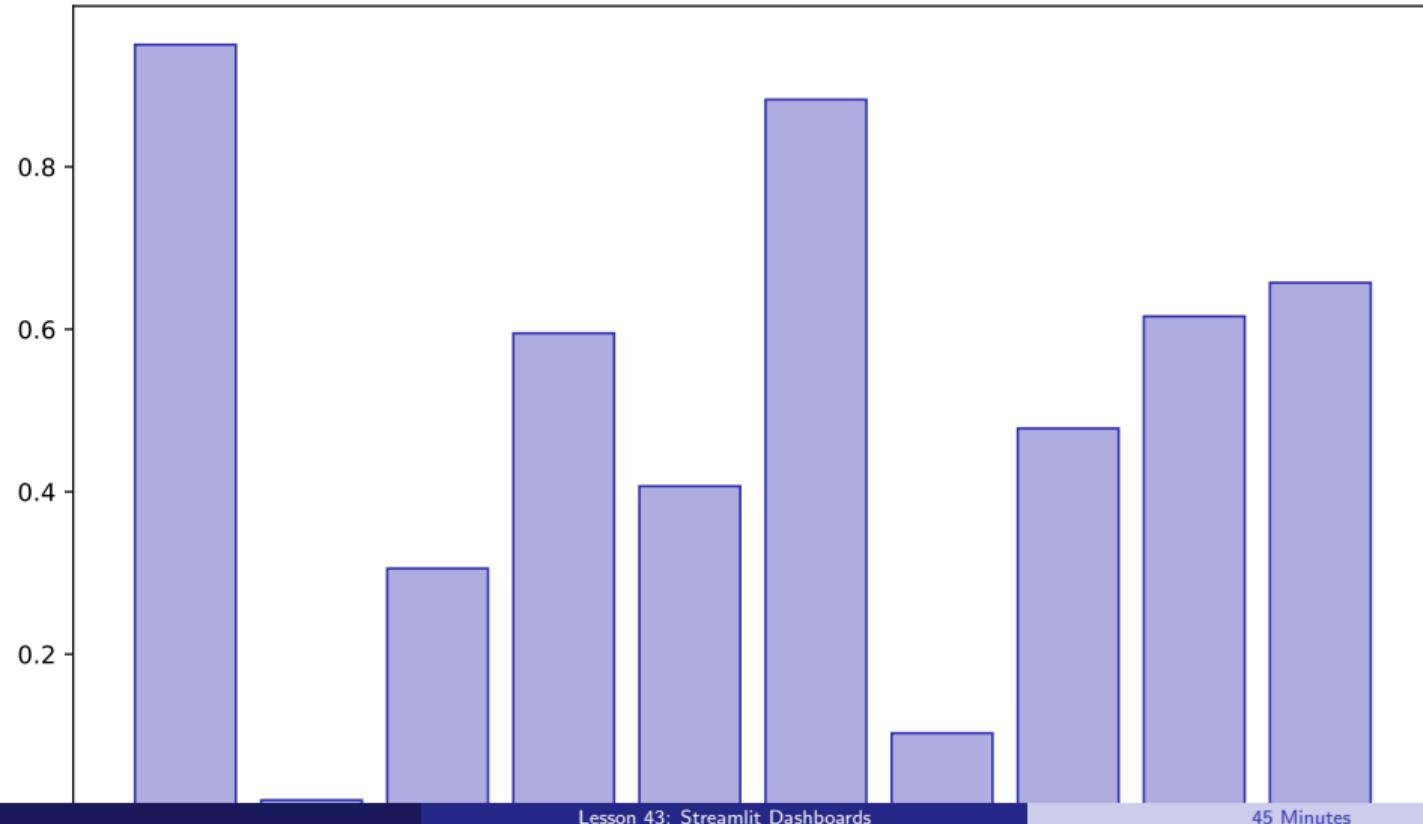
Caching



Charts Integration



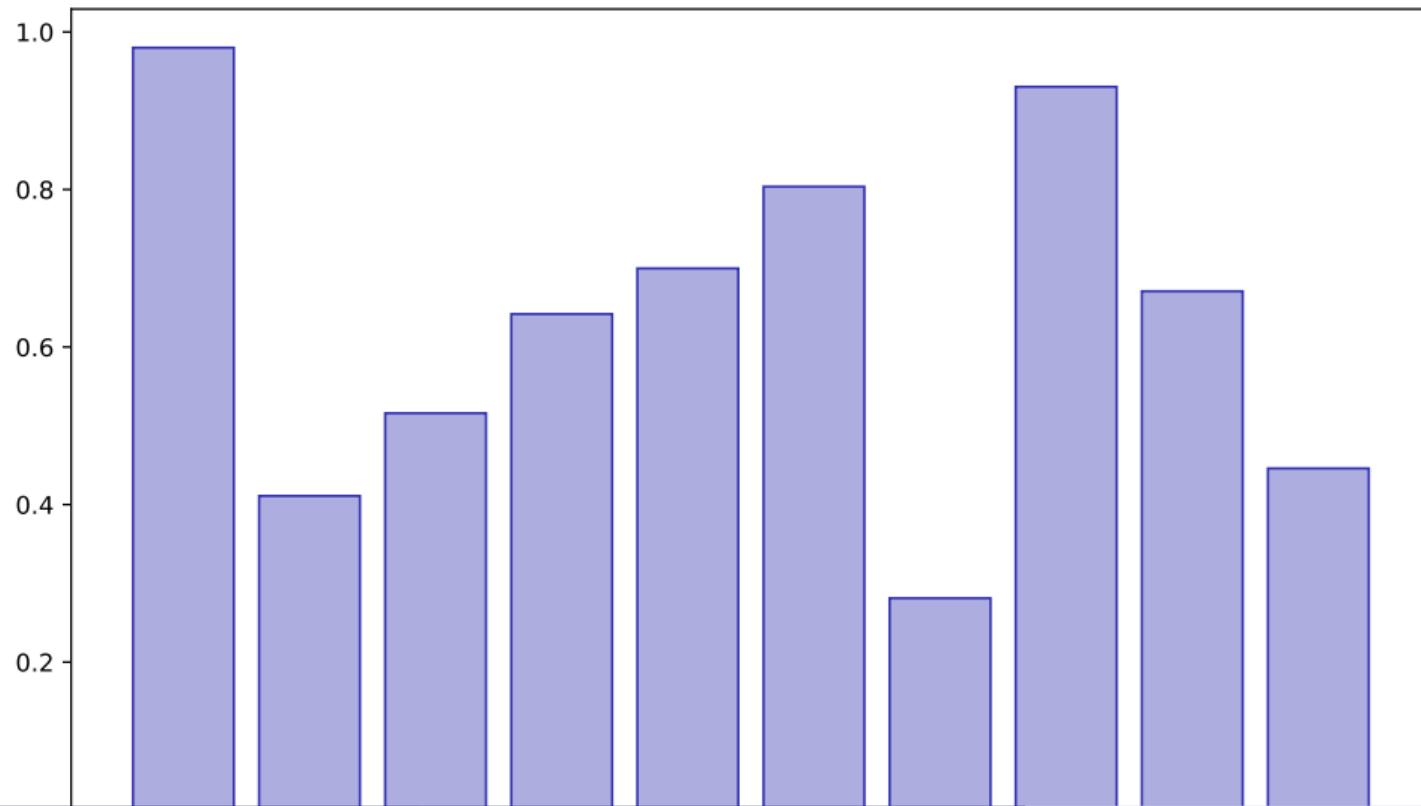
Model Integration



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

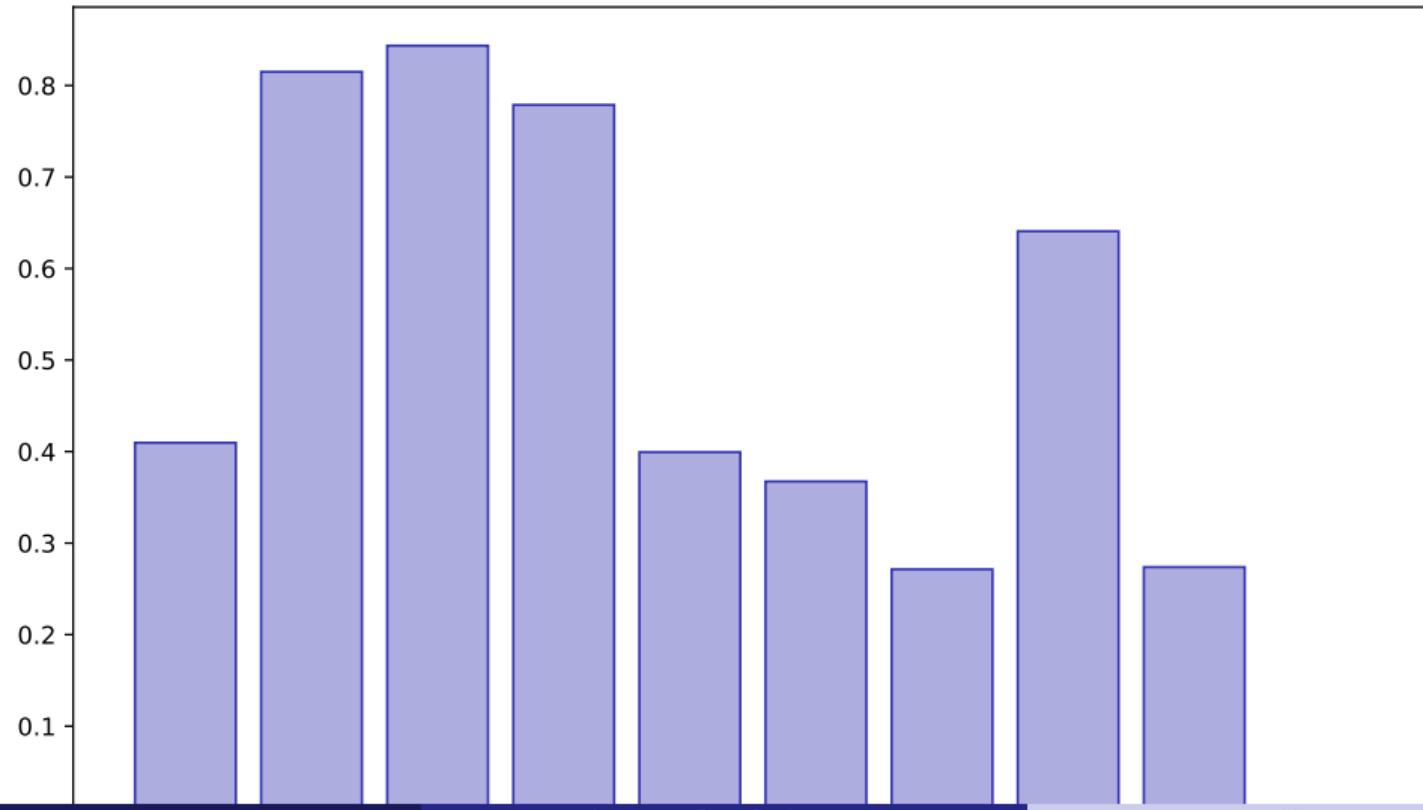
Learning Objectives

After this lesson, you will be able to:

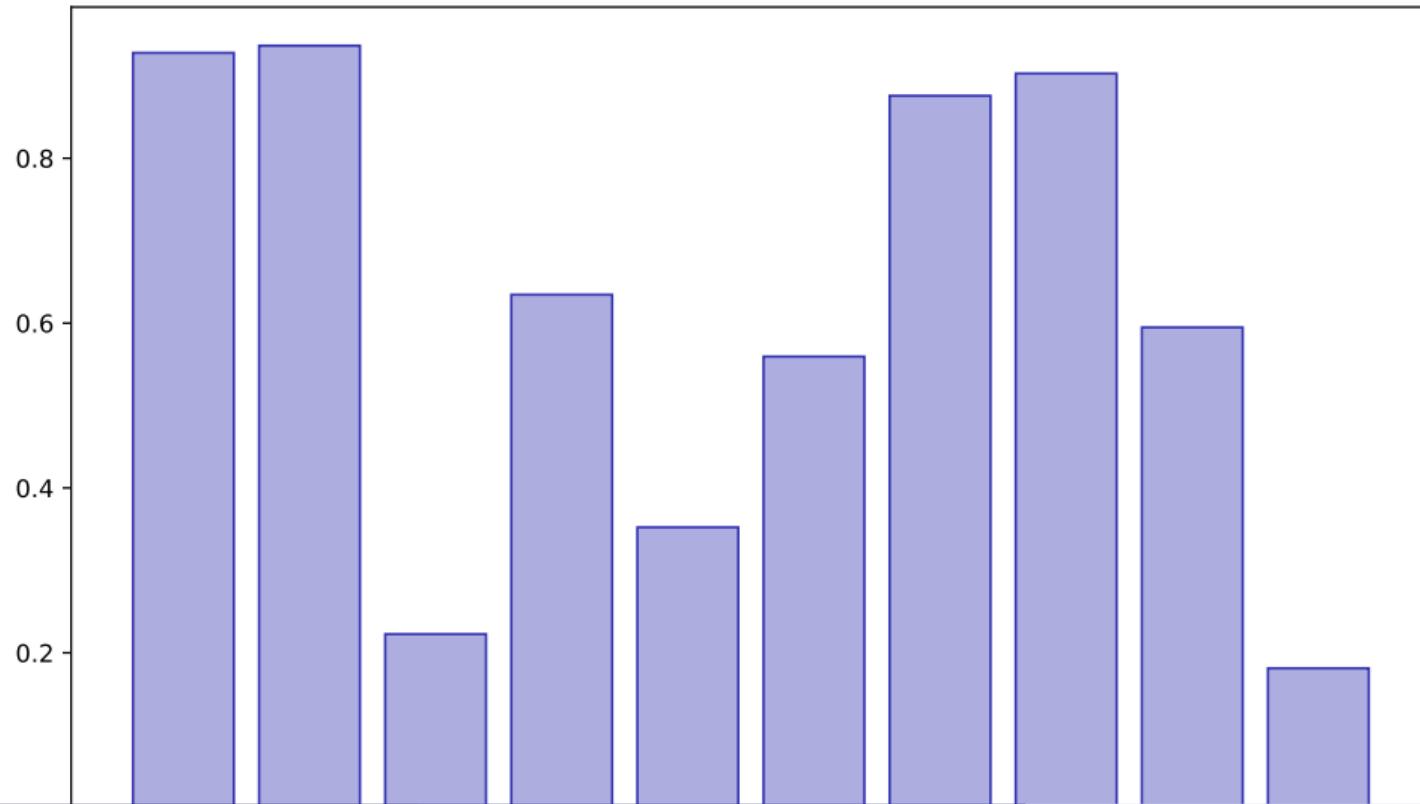
- Deploy to Streamlit Cloud
- Configure requirements
- Manage secrets
- Monitor applications

Building towards your final project

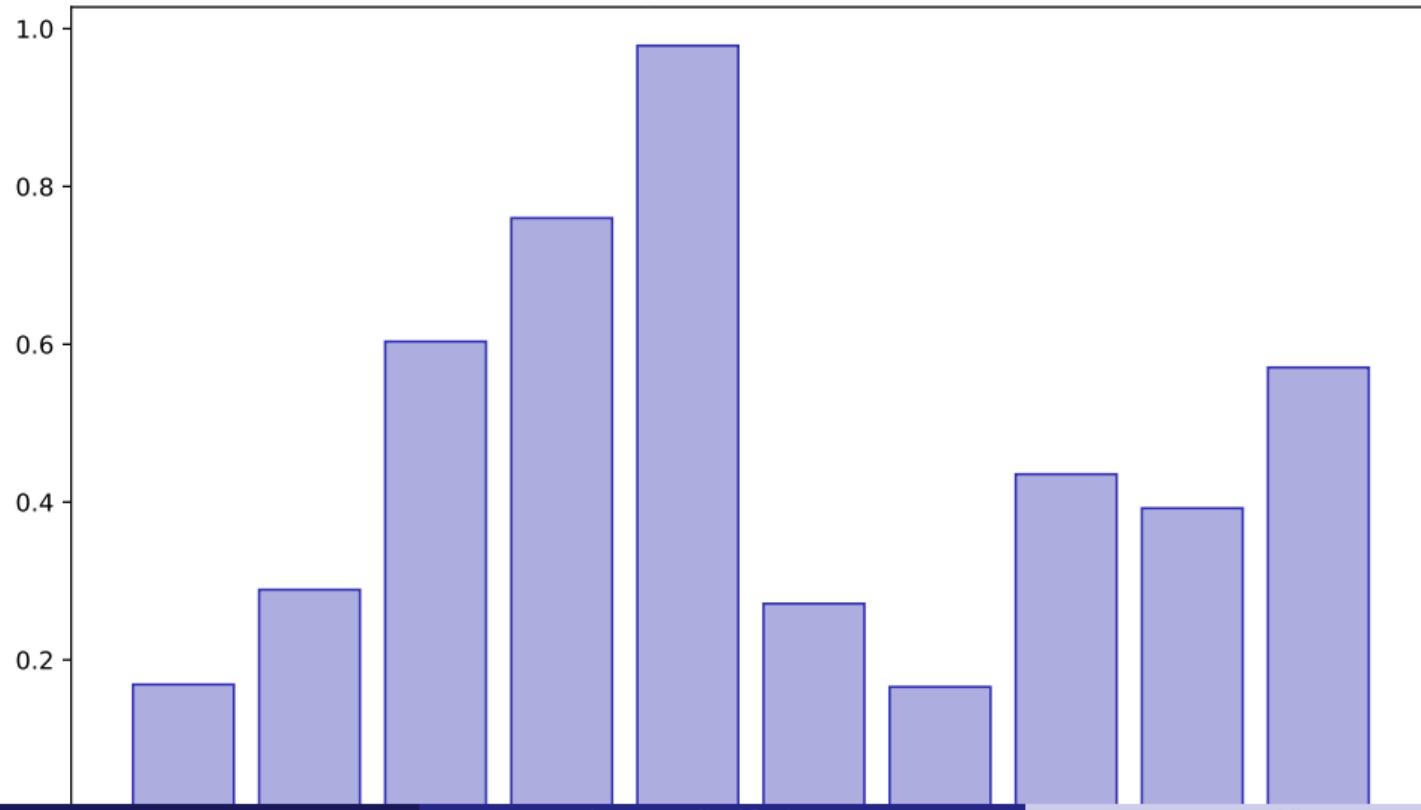
Cloud Options



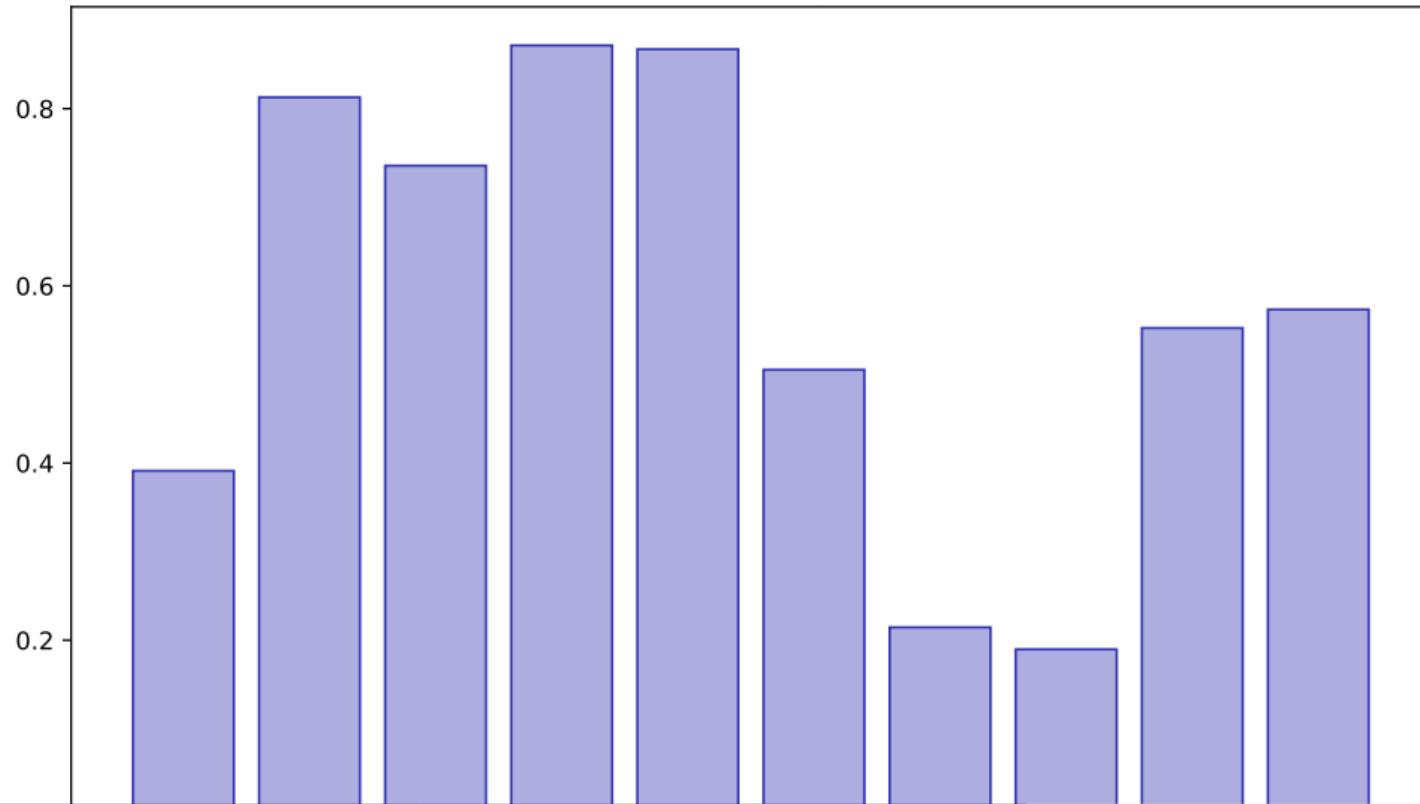
Streamlit Cloud



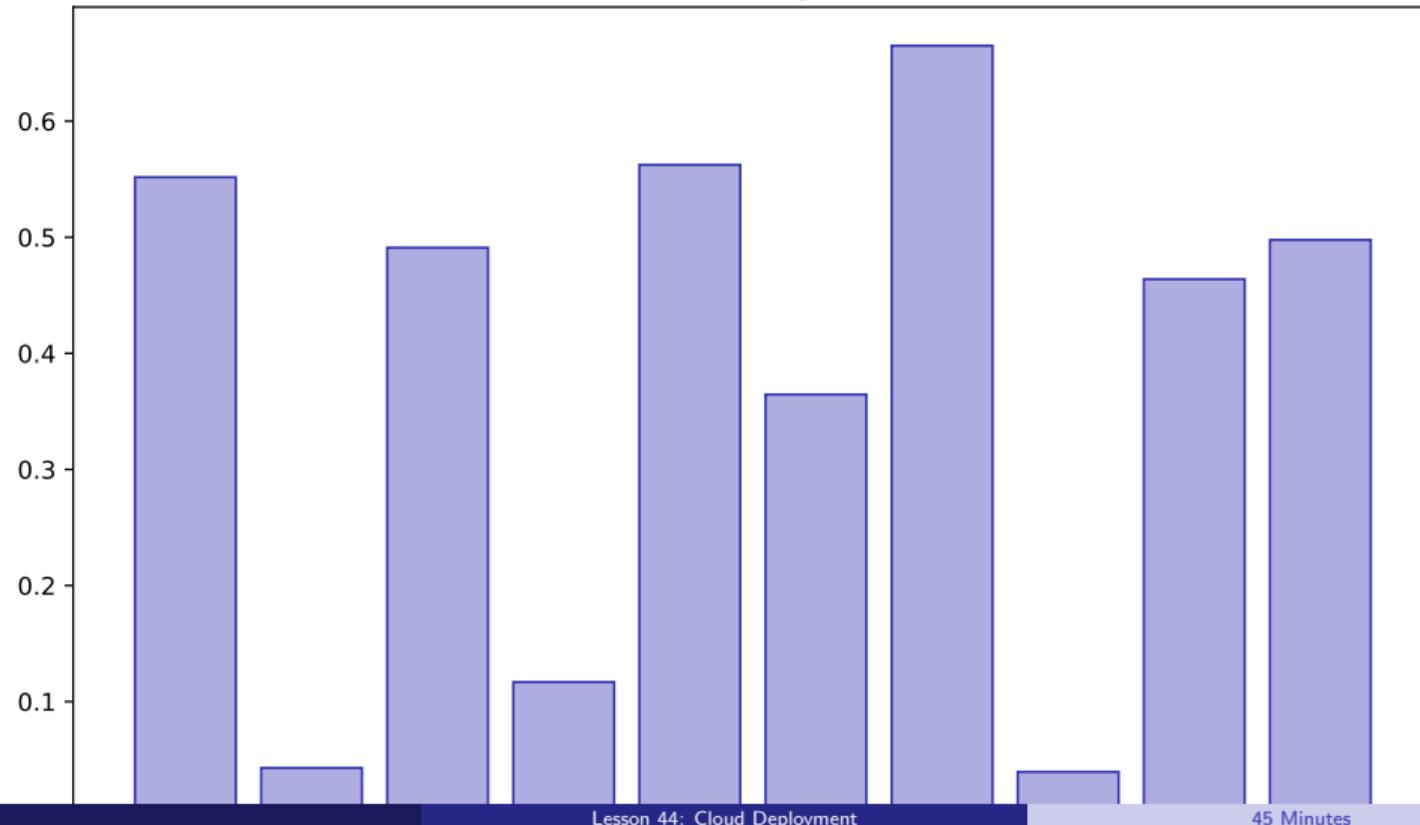
Requirements Txt



Secrets Management



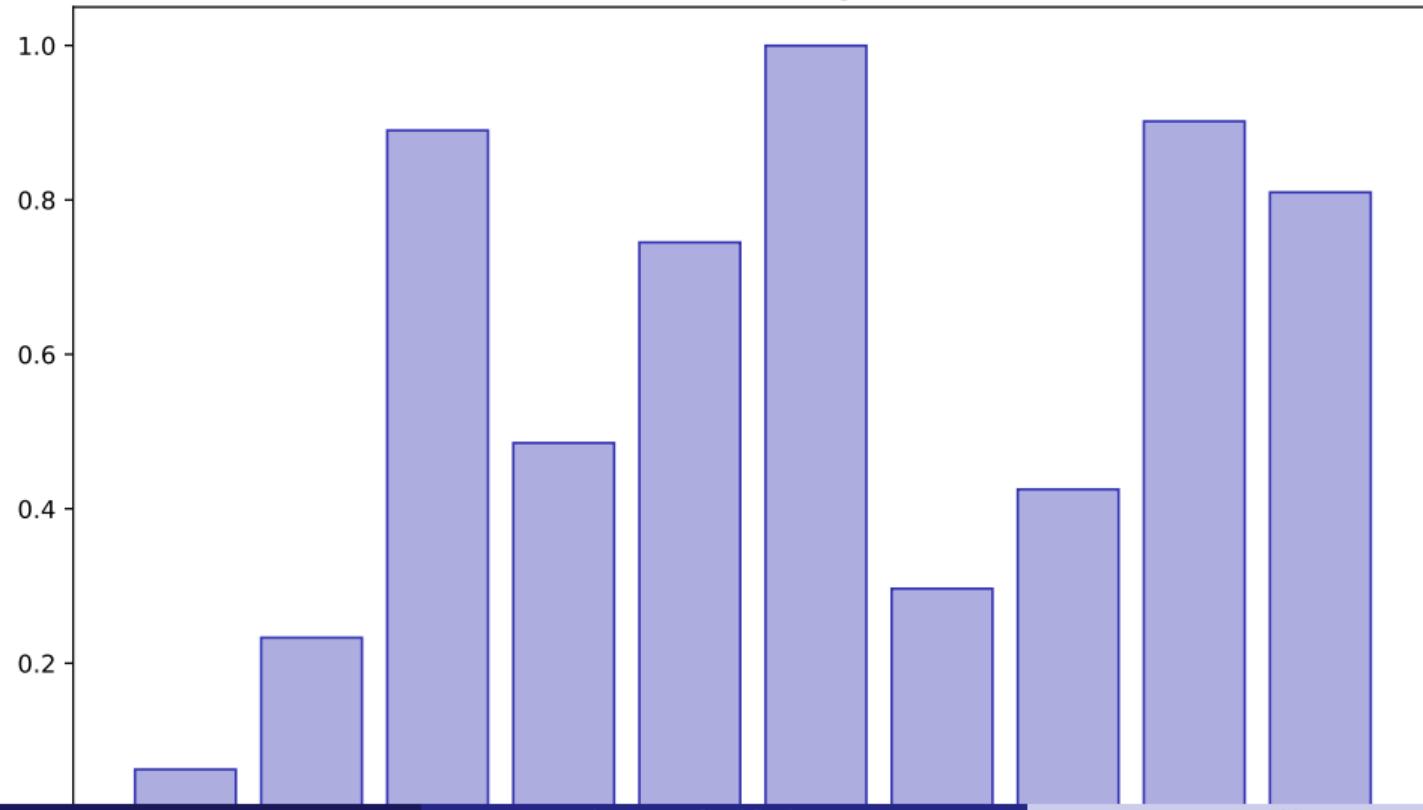
Github Integration



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

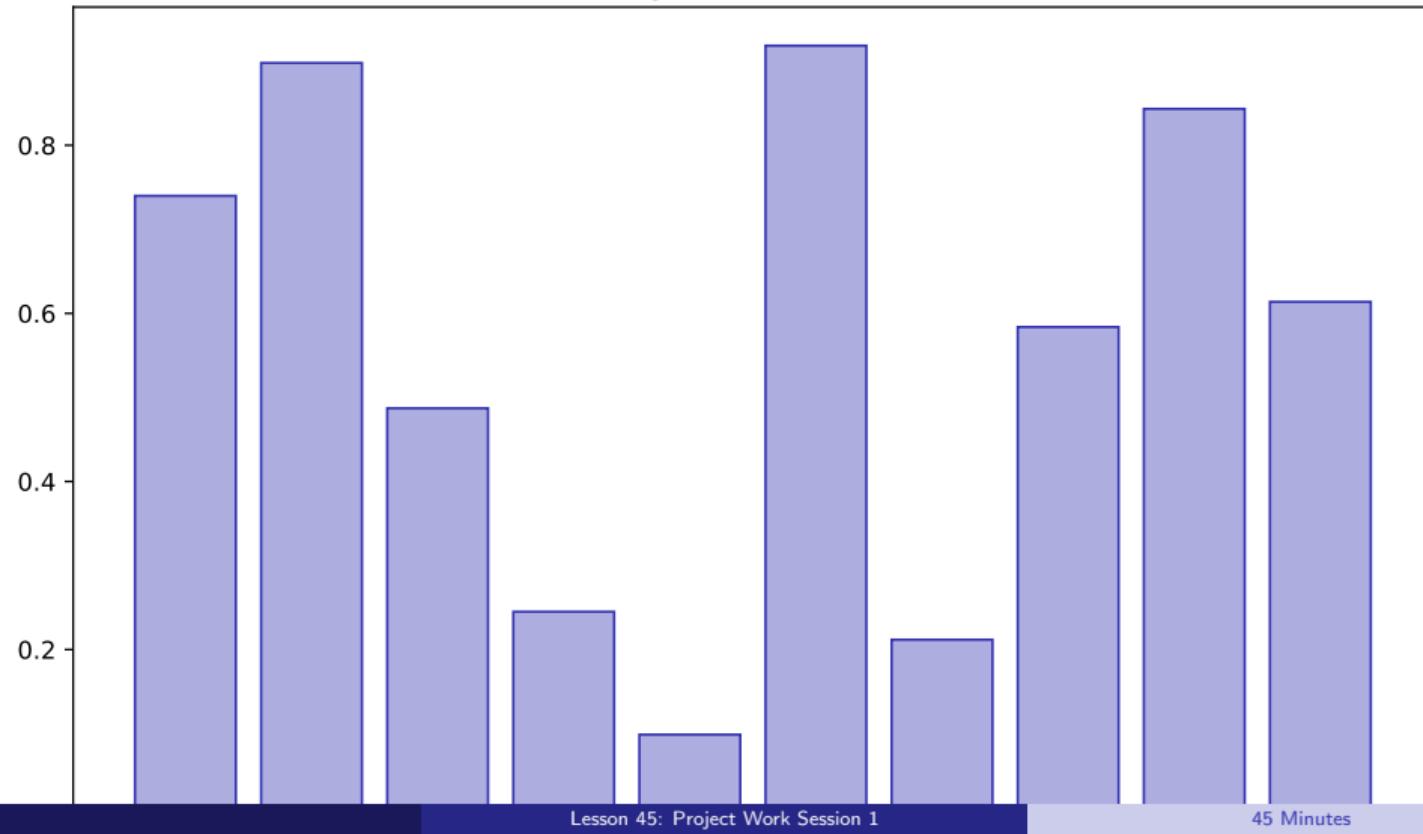
Learning Objectives

After this lesson, you will be able to:

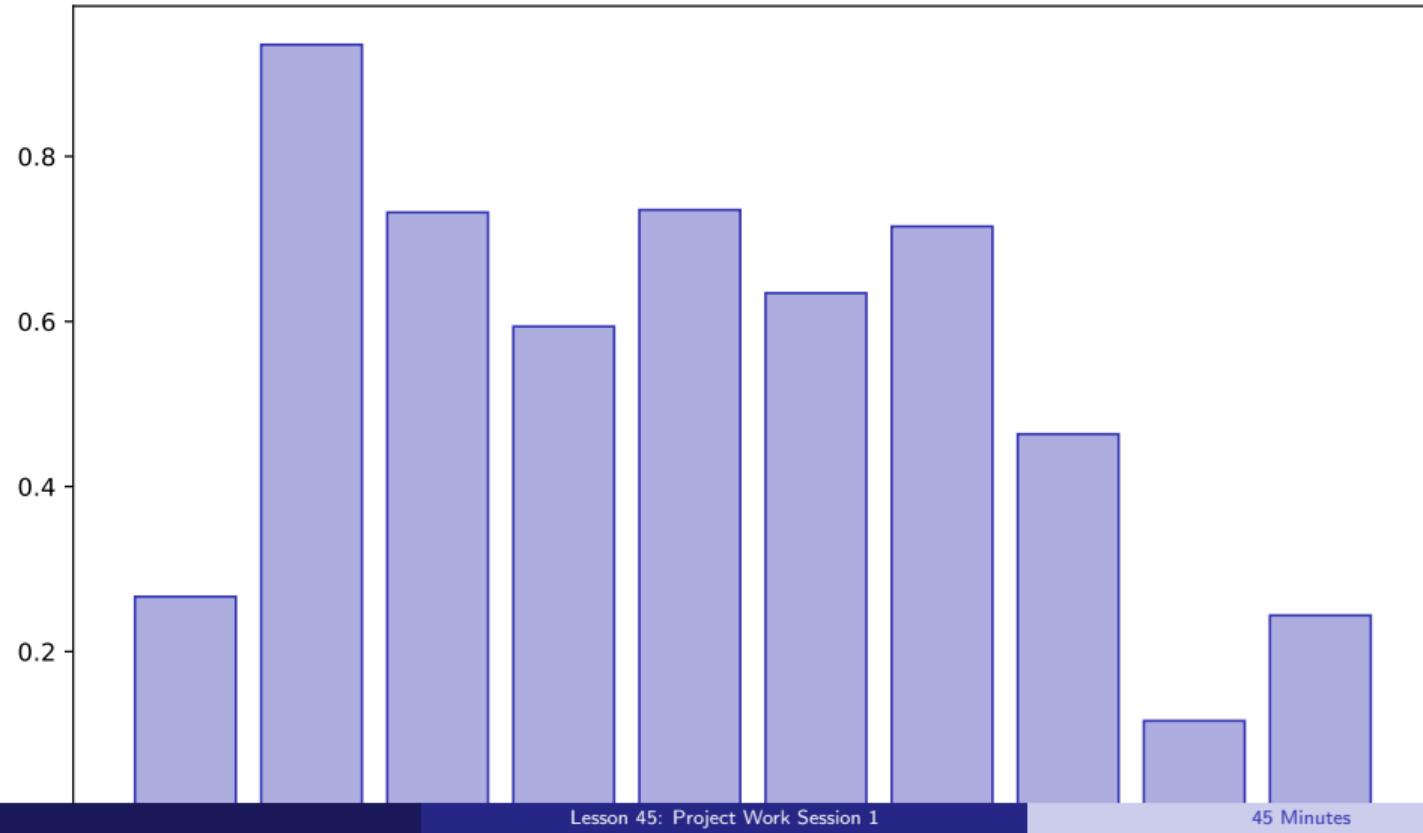
- Review project requirements
- Plan implementation
- Start coding
- Get feedback

Building towards your final project

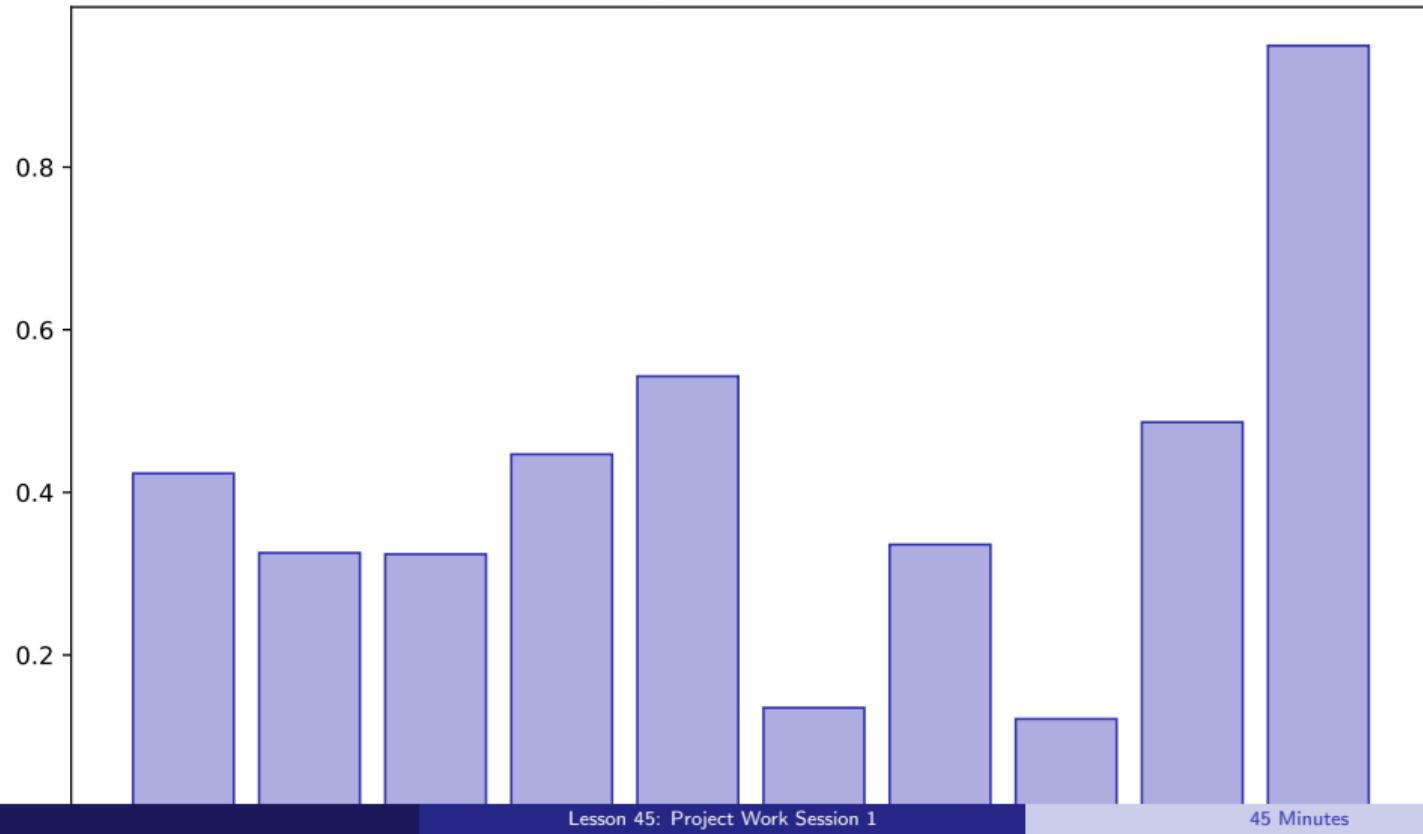
Project Overview



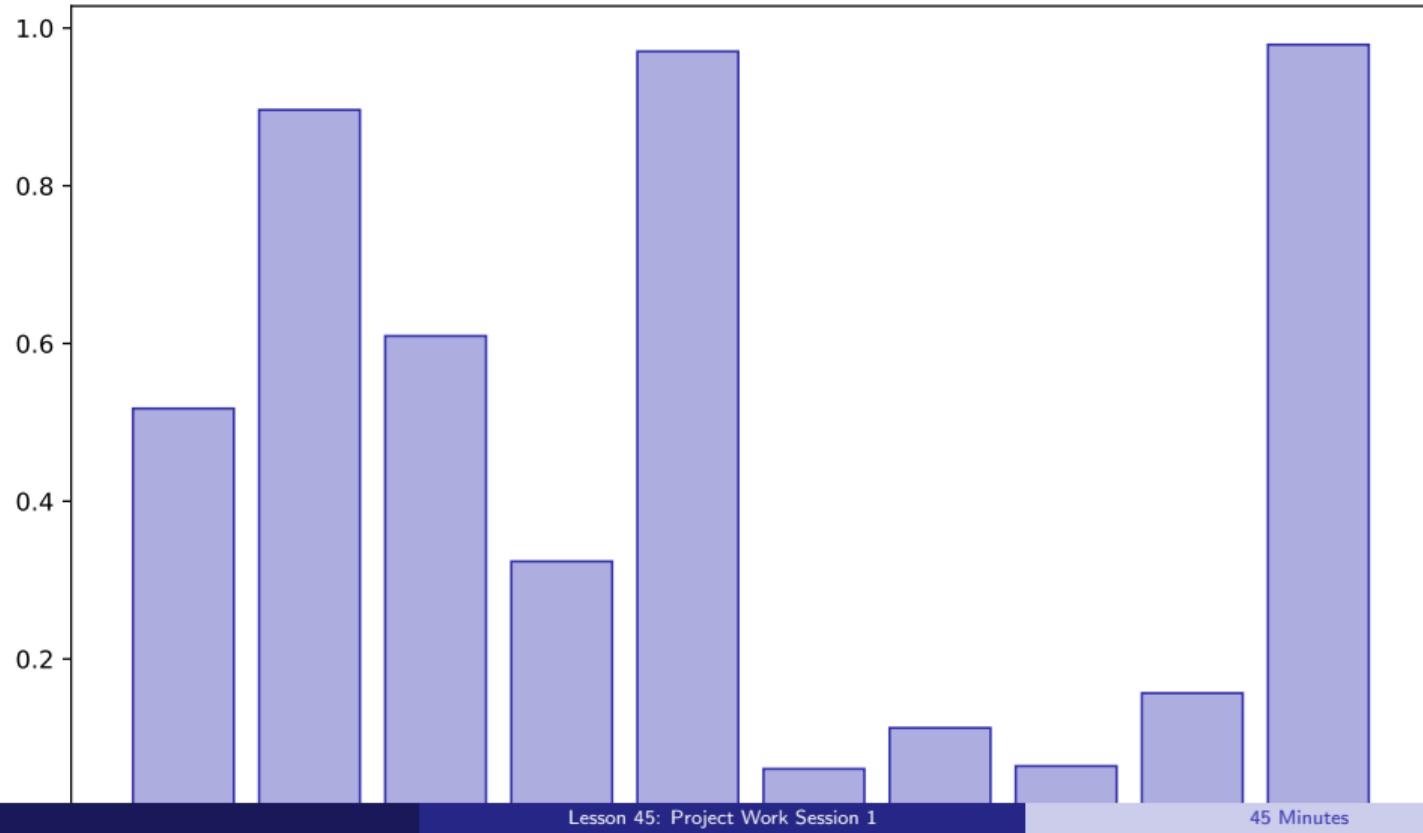
Rubric Review



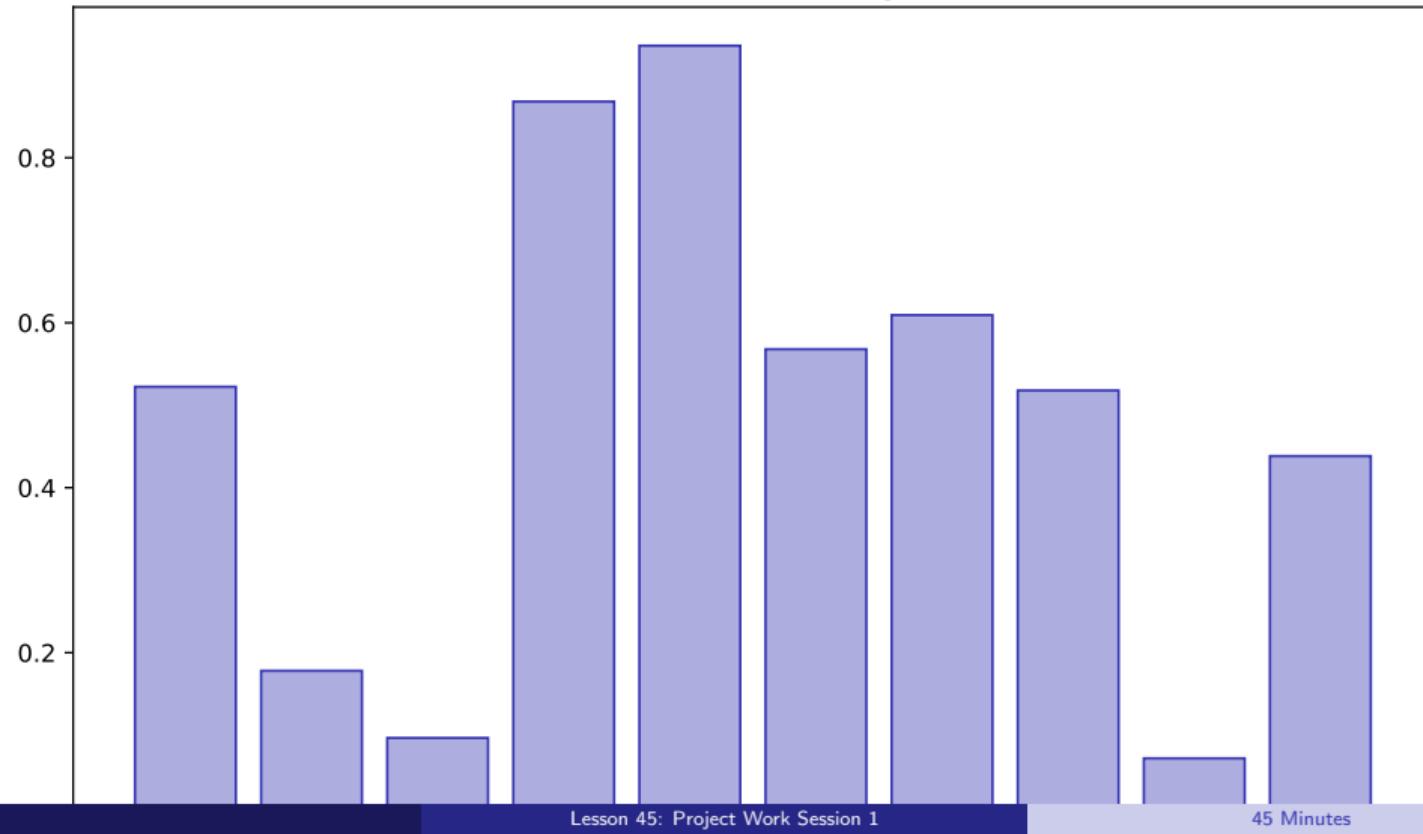
Topic Selection



Data Requirements

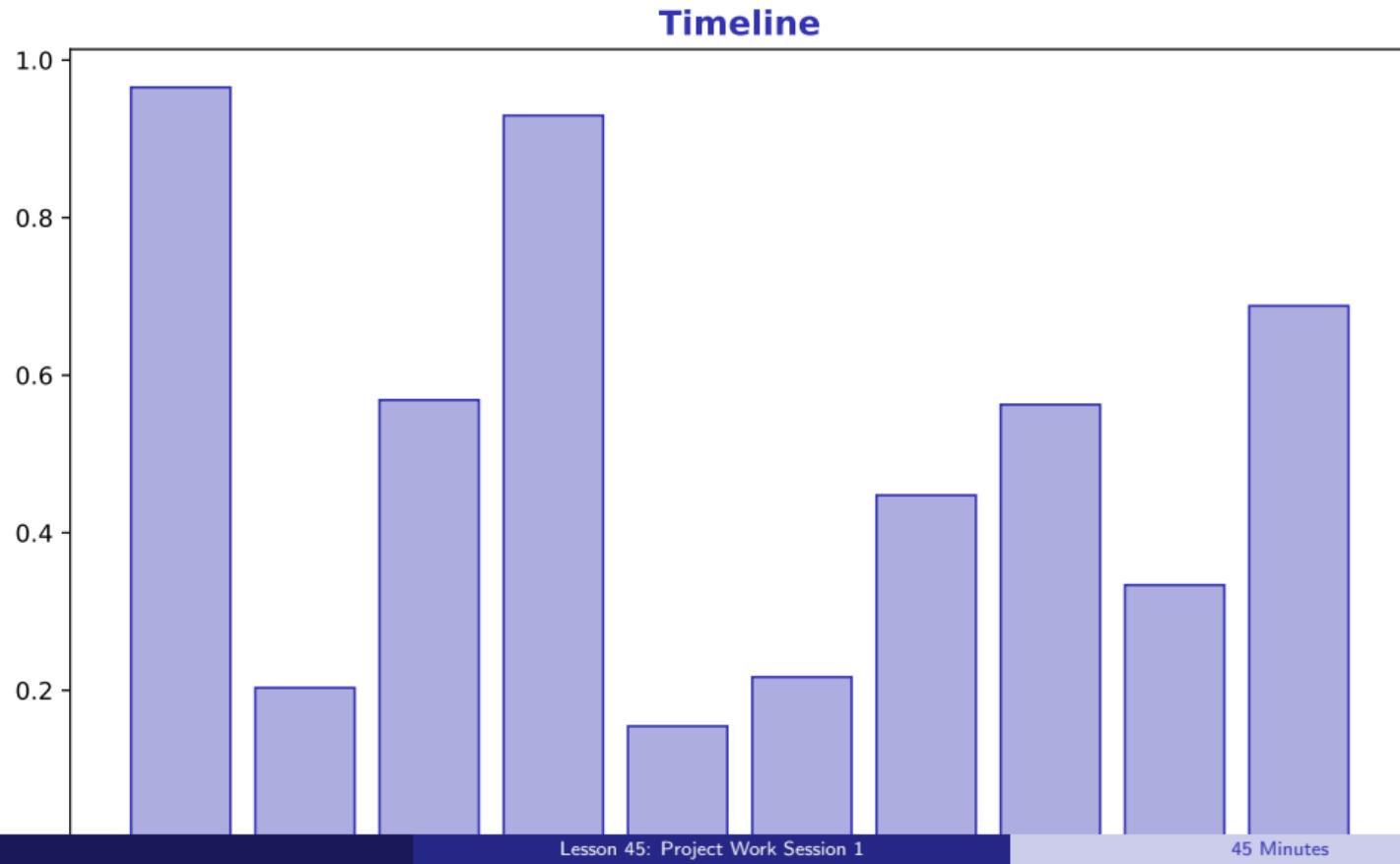


Model Planning



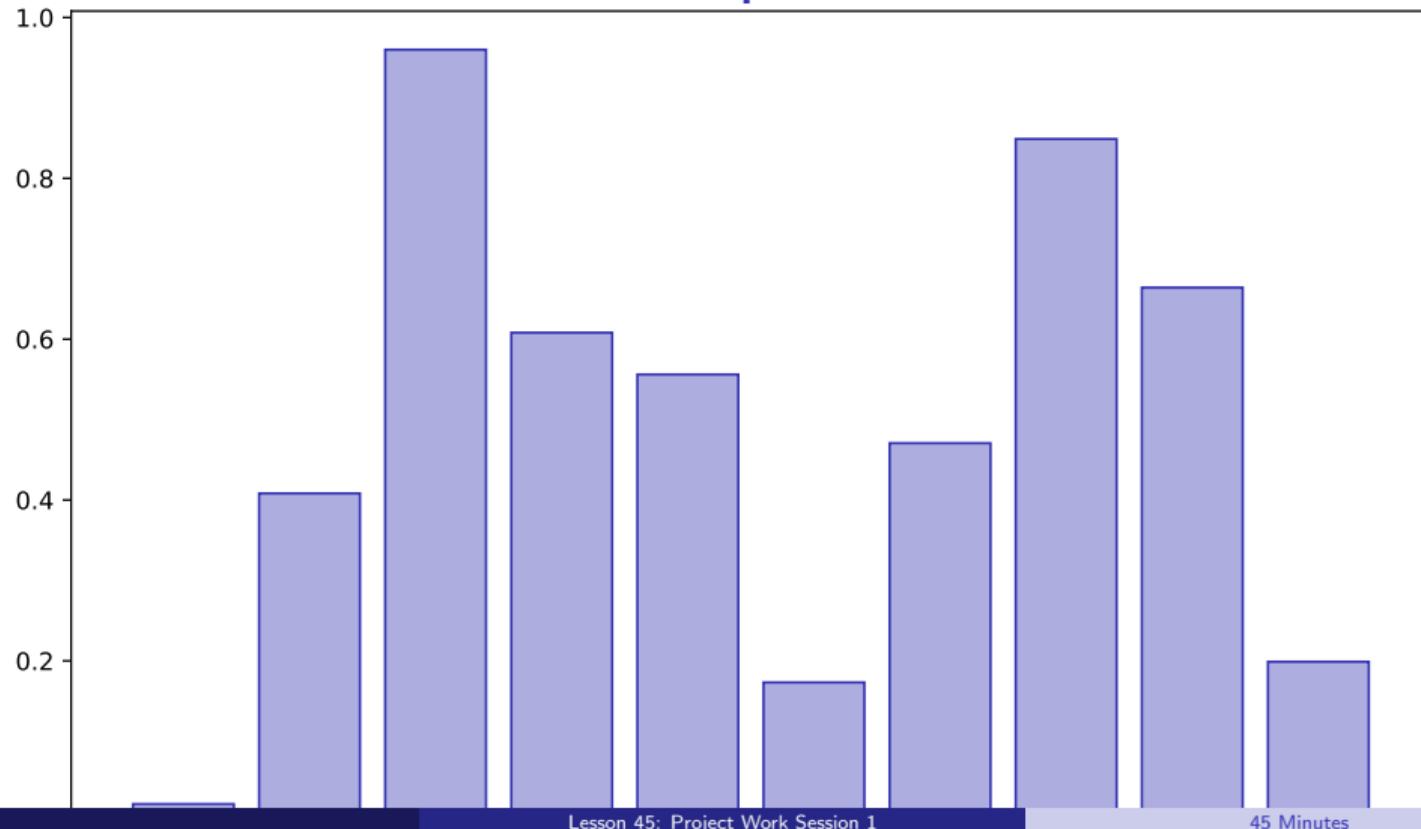
Deployment Planning





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

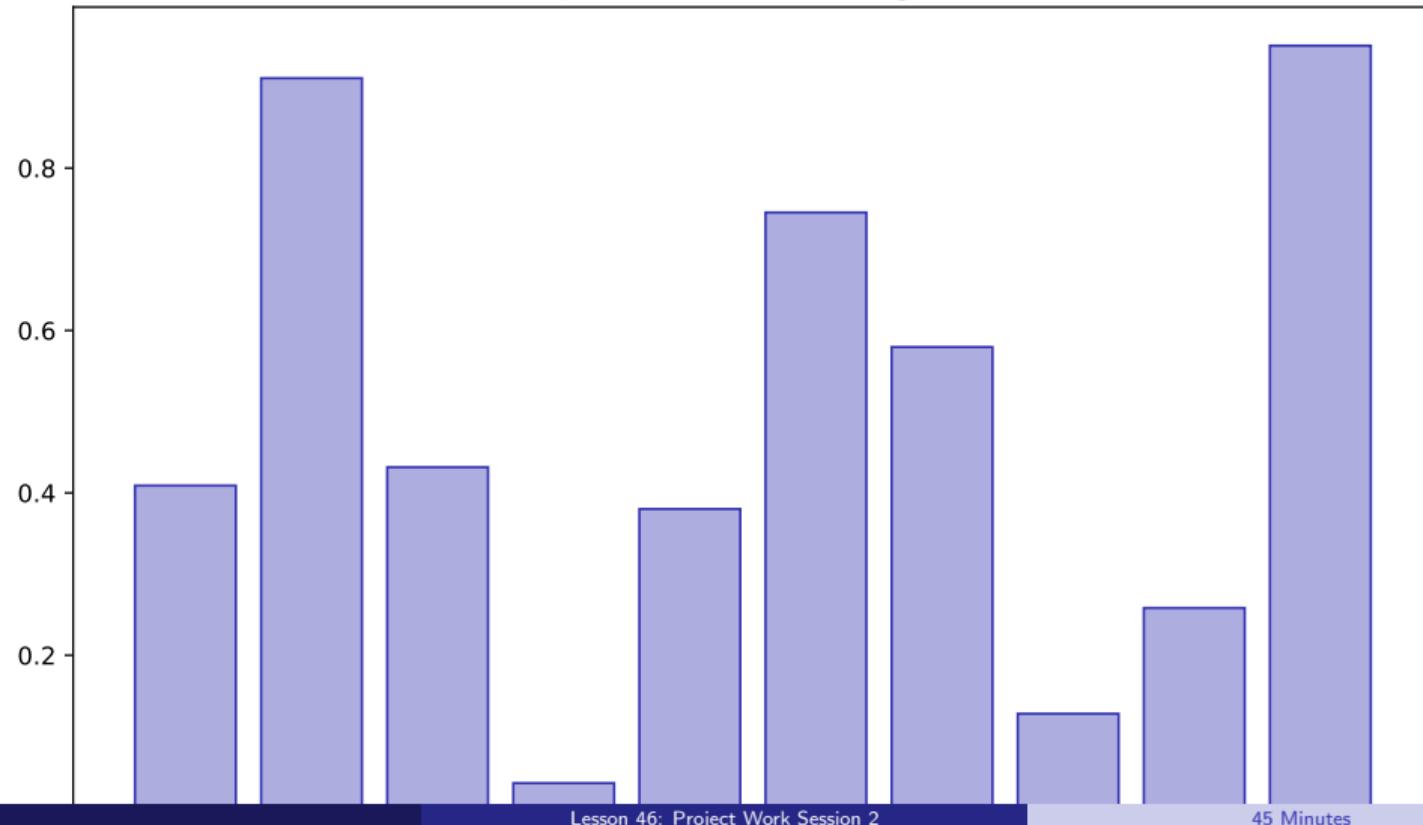
Learning Objectives

After this lesson, you will be able to:

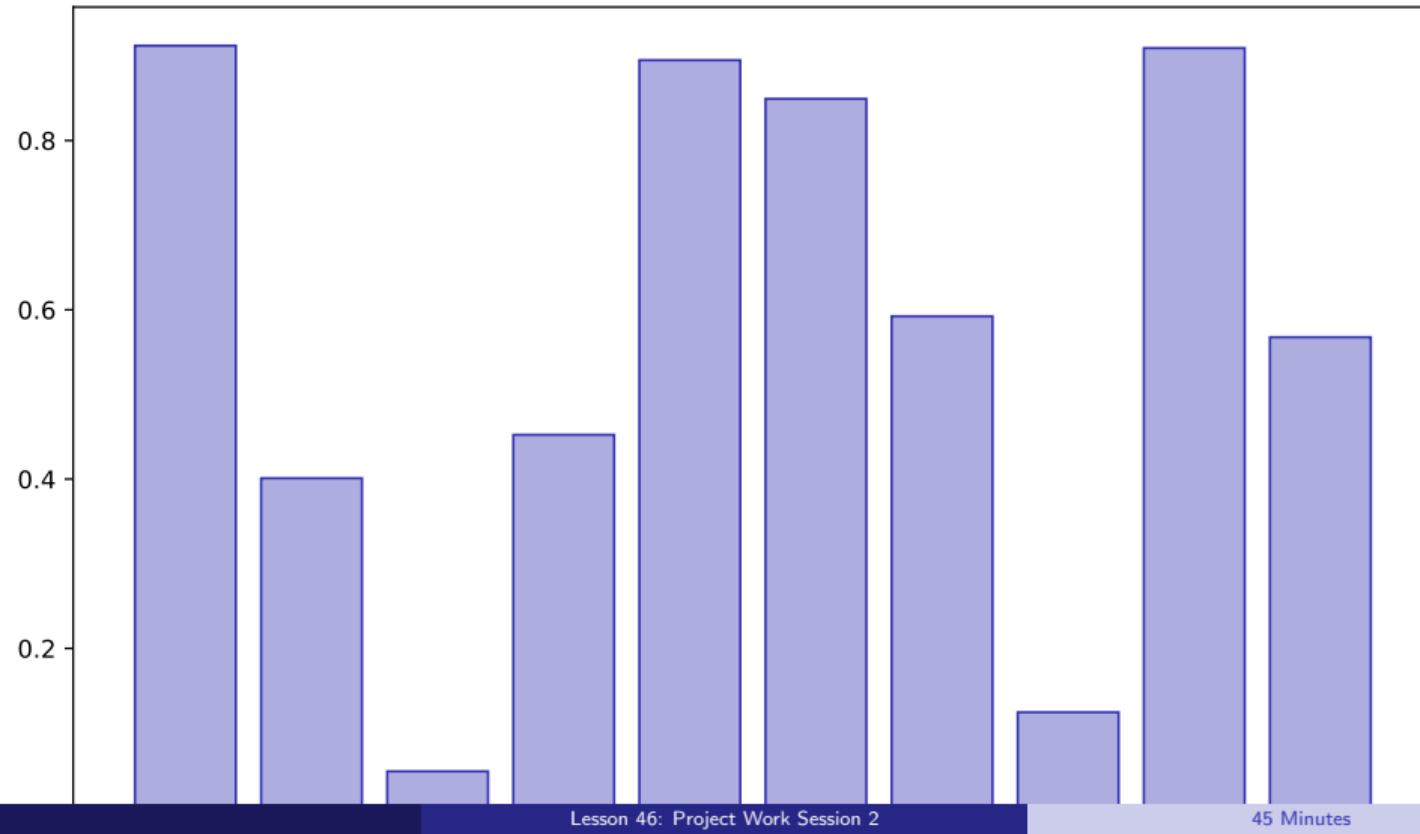
- Continue implementation
- Prepare presentation
- Practice demo
- Finalize code

Building towards your final project

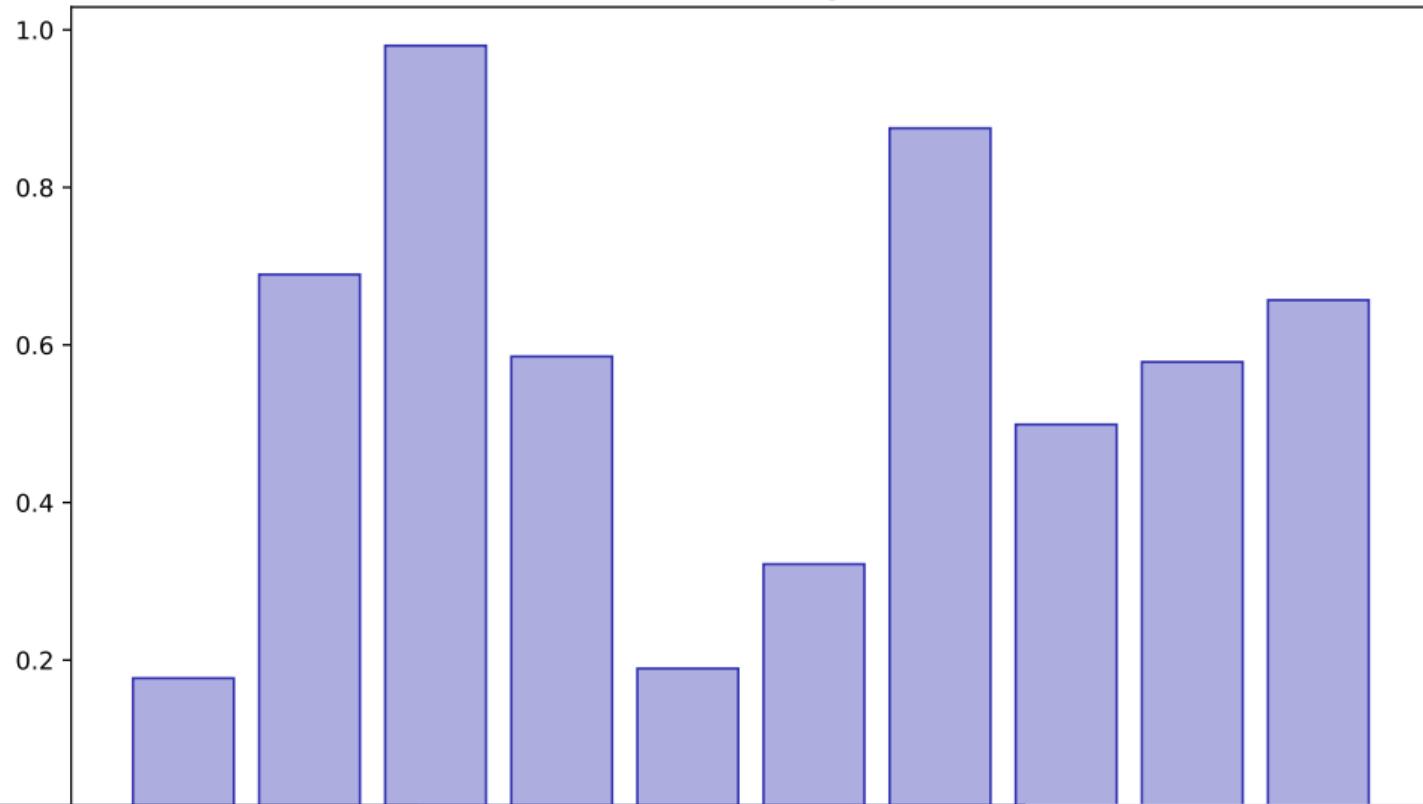
Implementation Progress



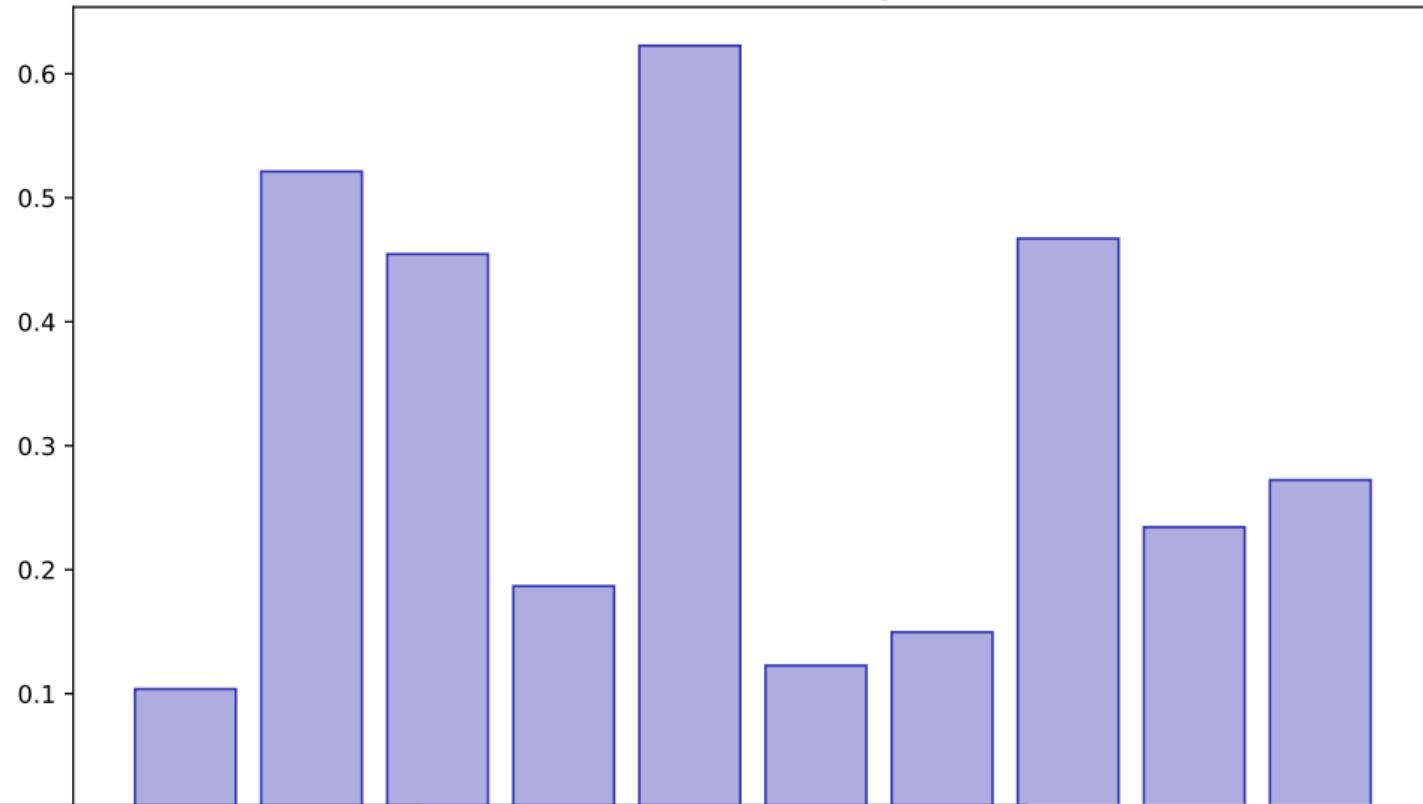
Presentation Structure



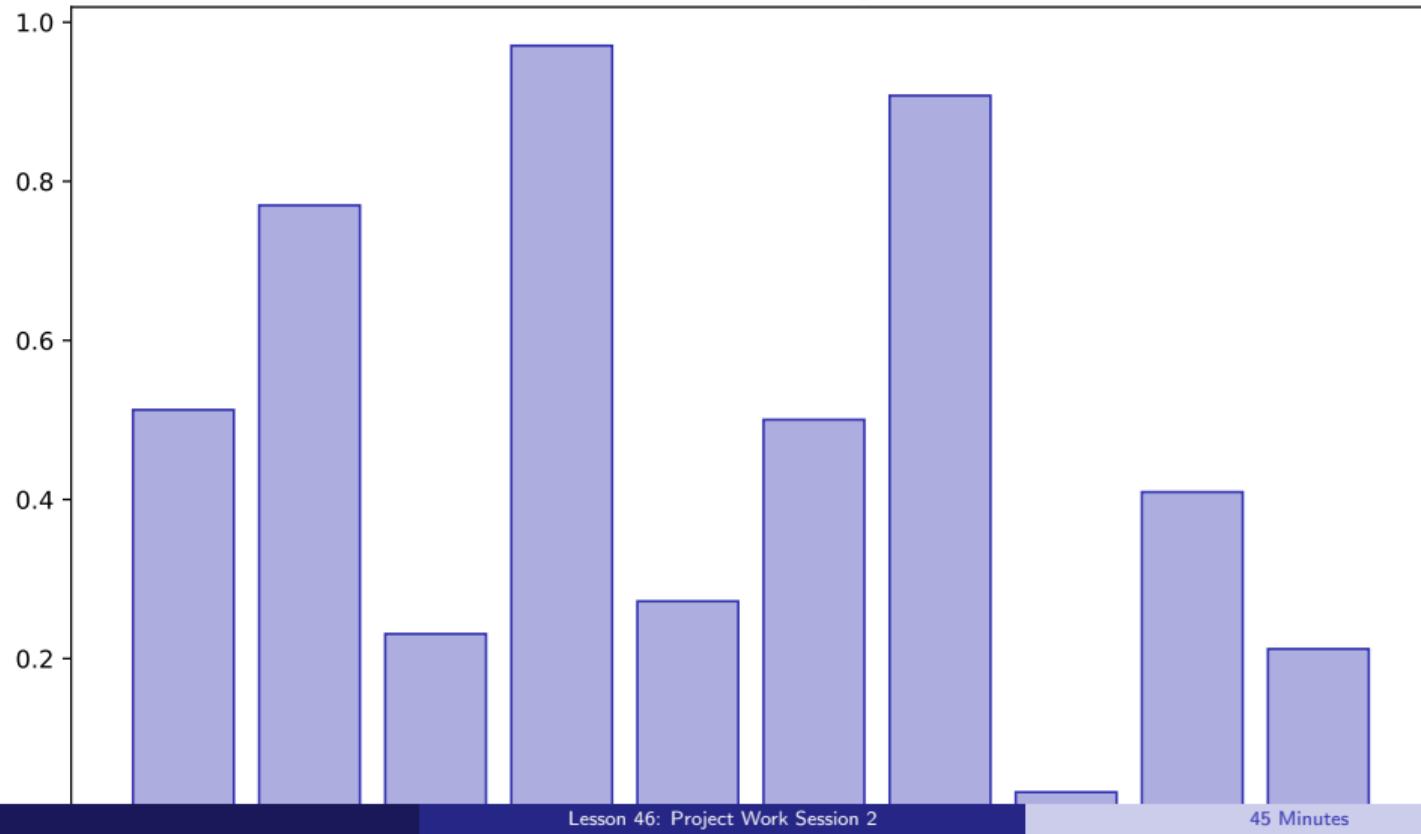
Slide Design



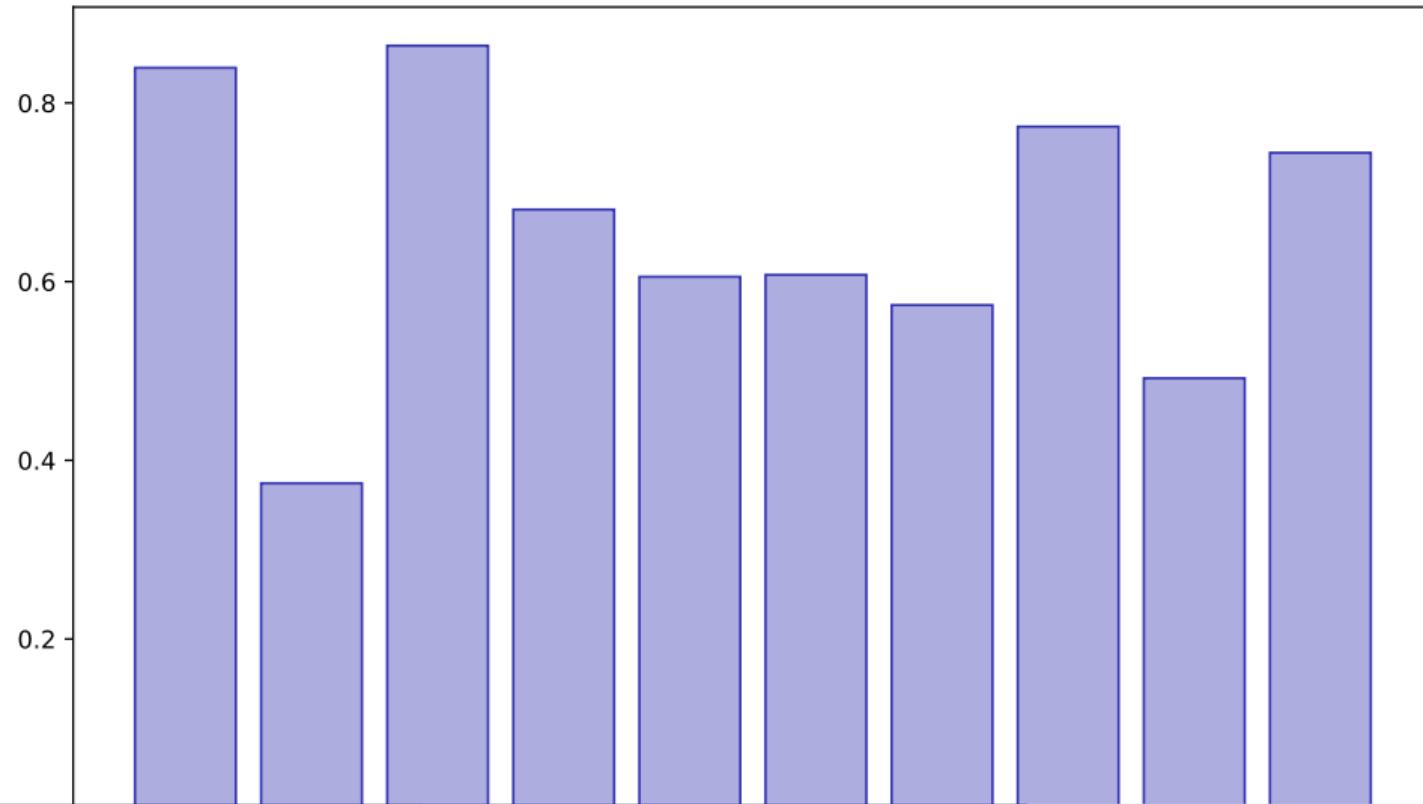
Demo Planning



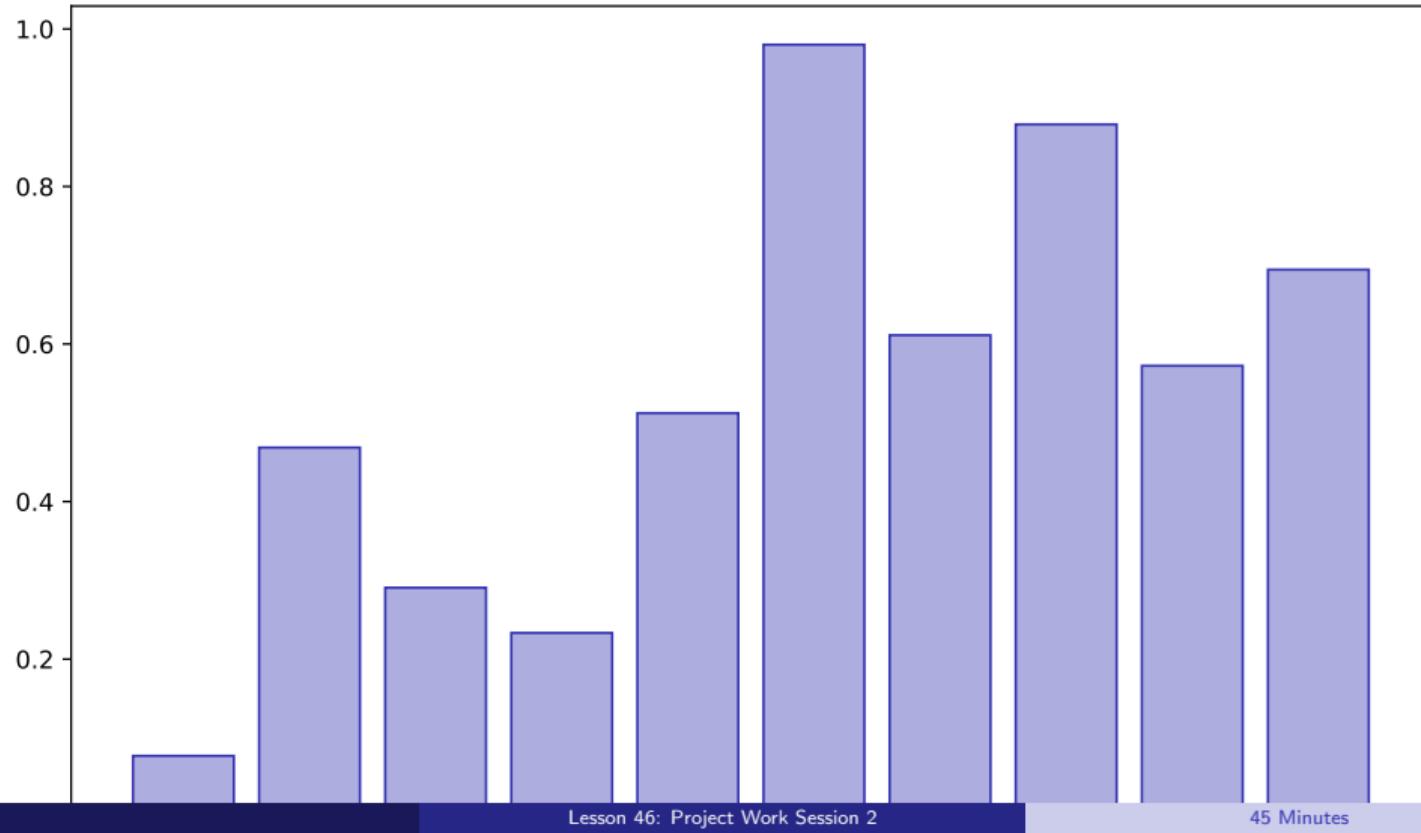
Code Cleanup



Documentation

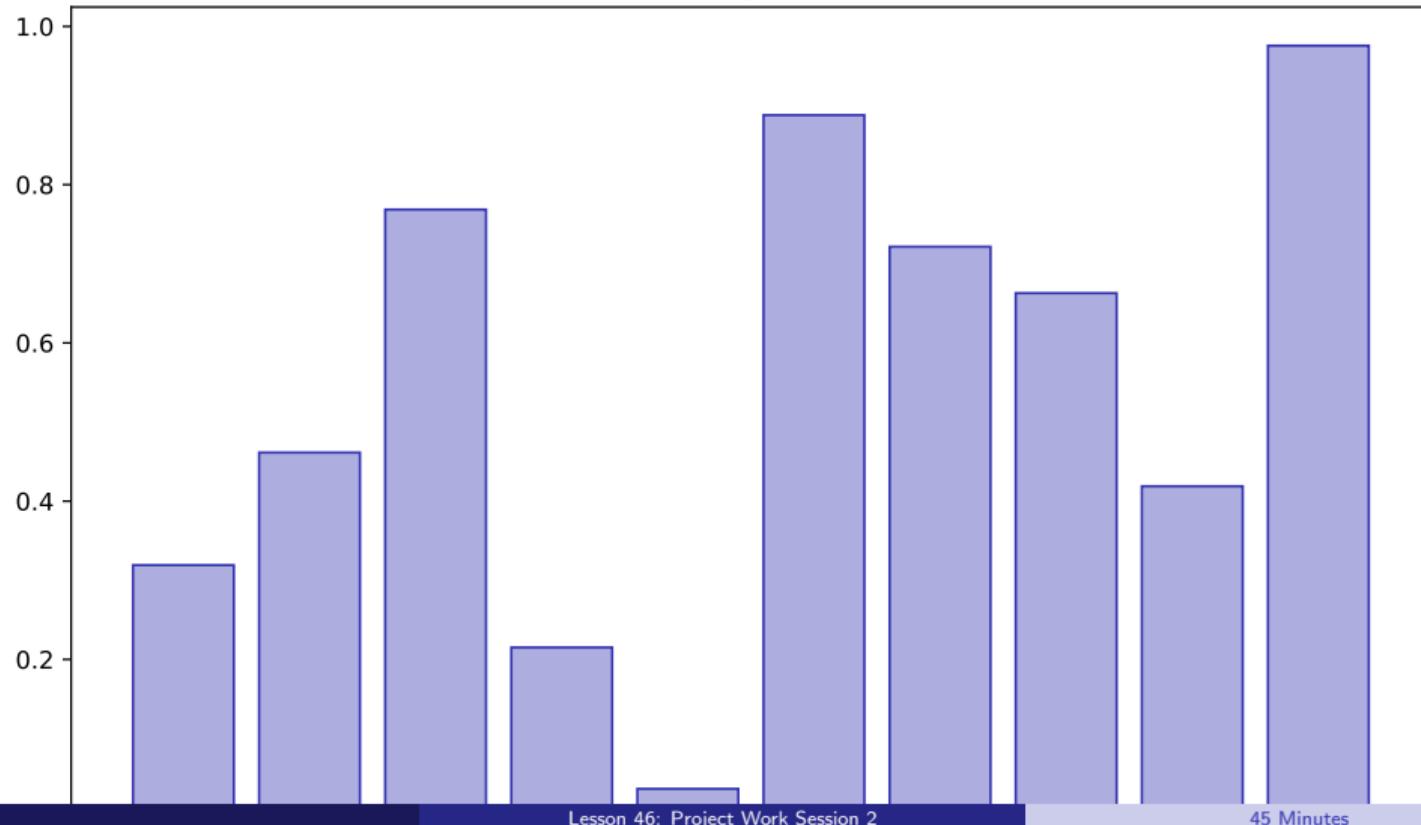


Peer Feedback



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

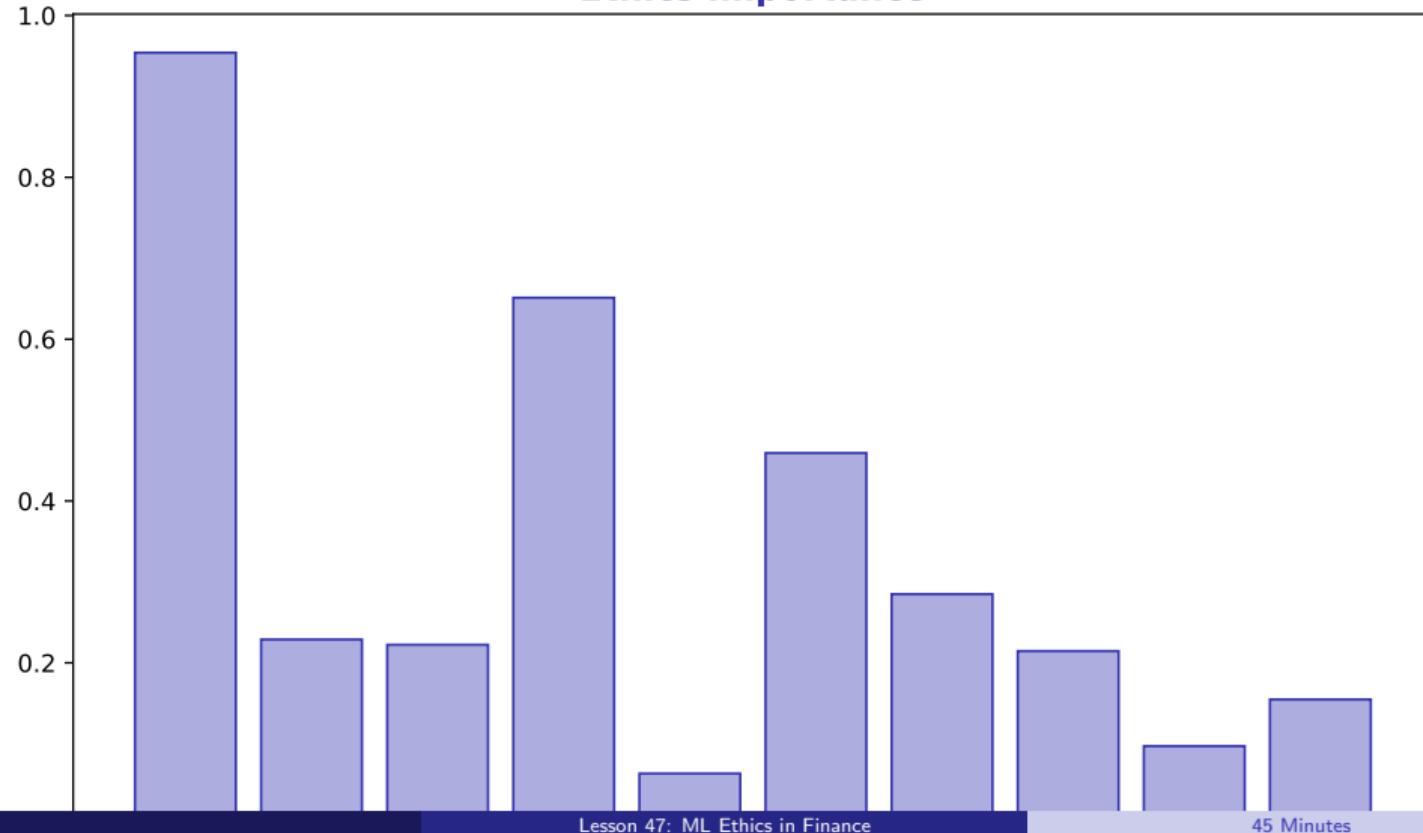
45 Minutes

After this lesson, you will be able to:

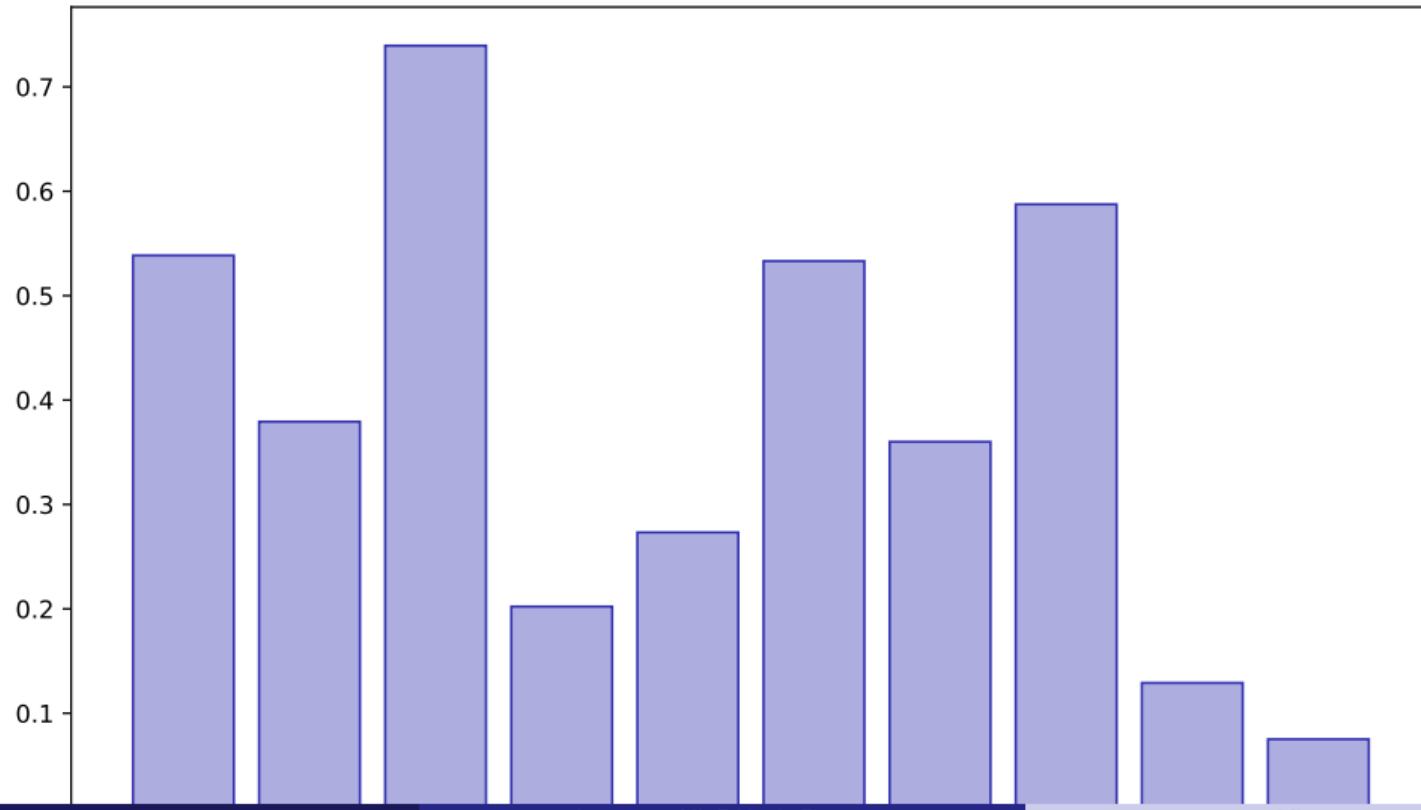
- Identify bias in models
- Ensure explainability
- Consider fairness
- Follow regulations

Building towards your final project

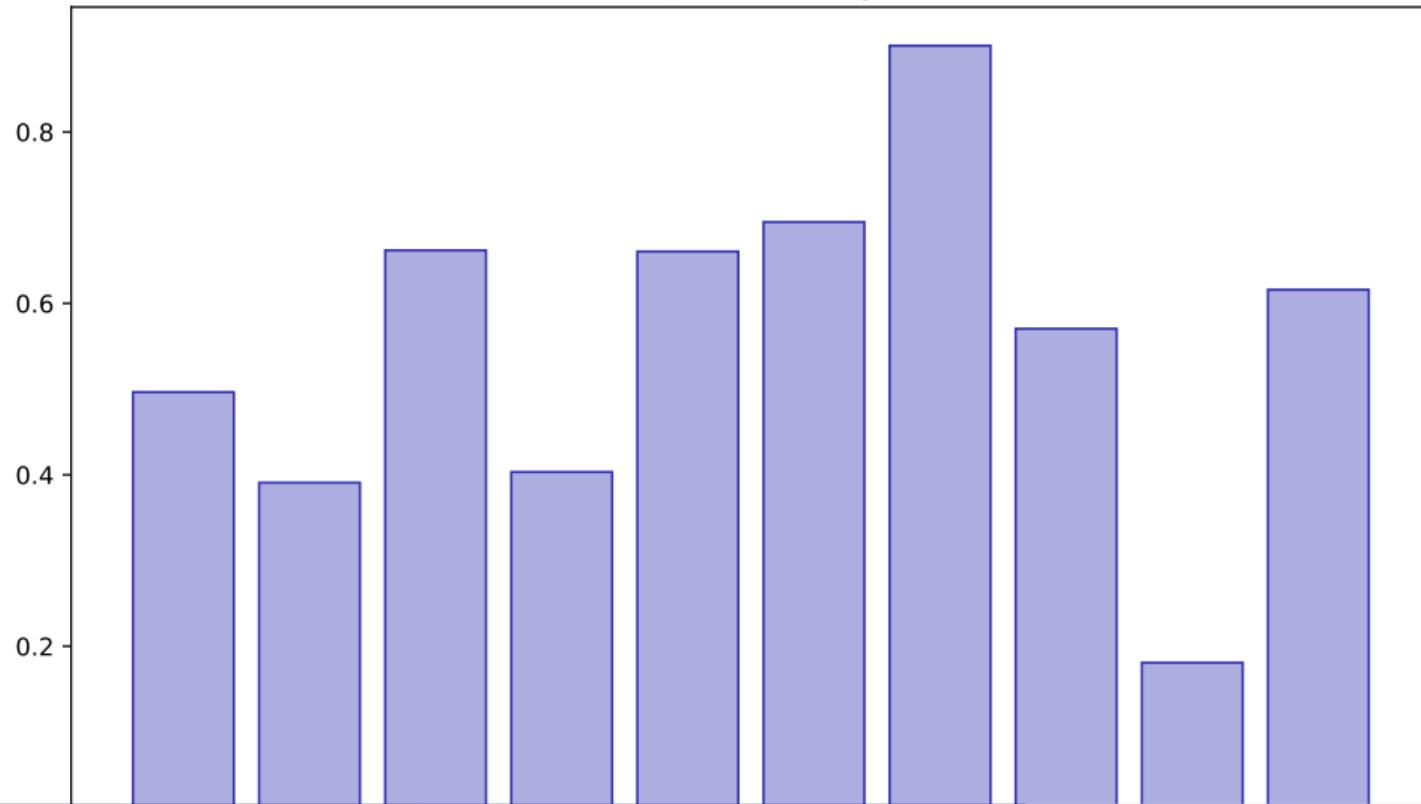
Ethics Importance



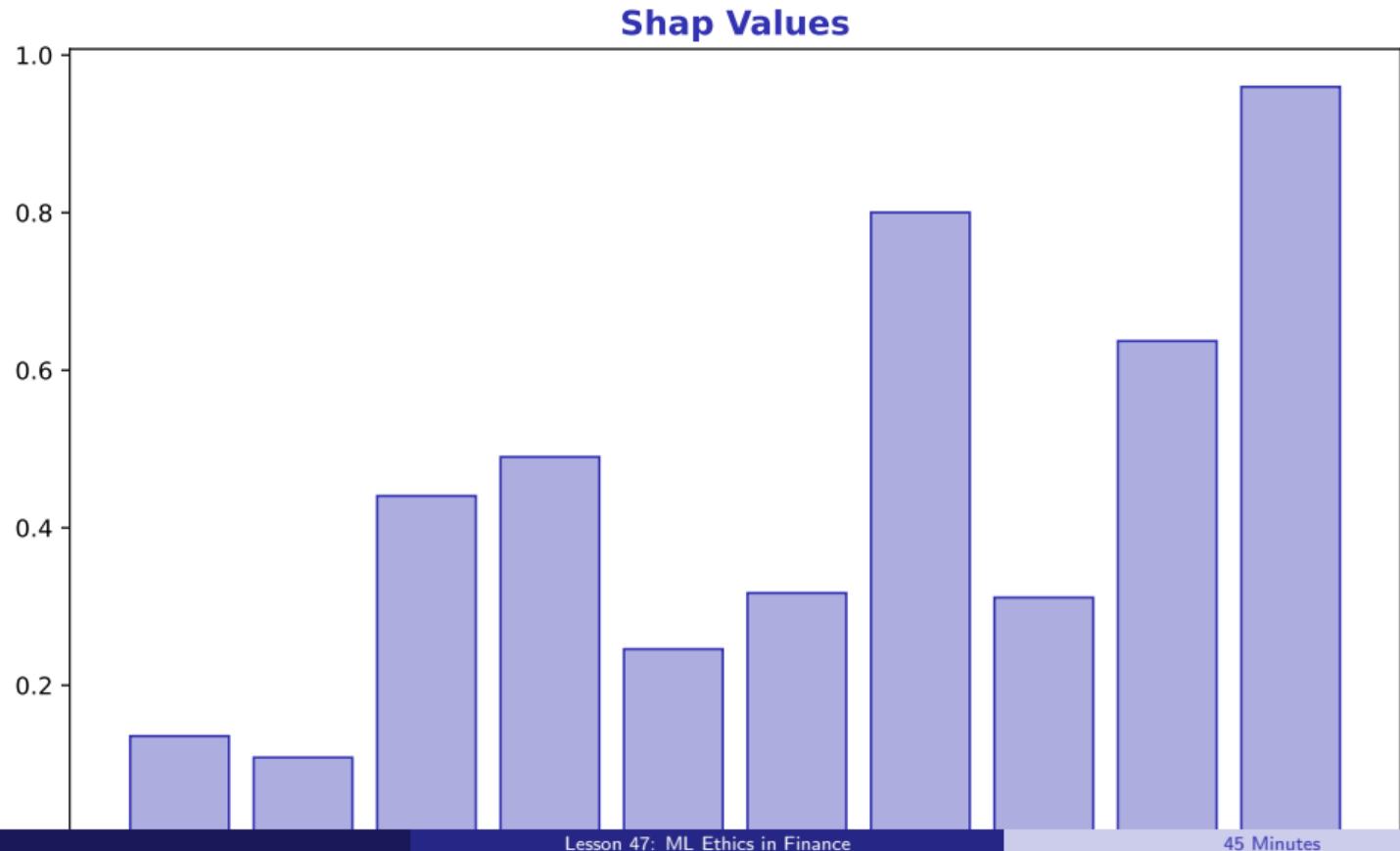
Bias Sources



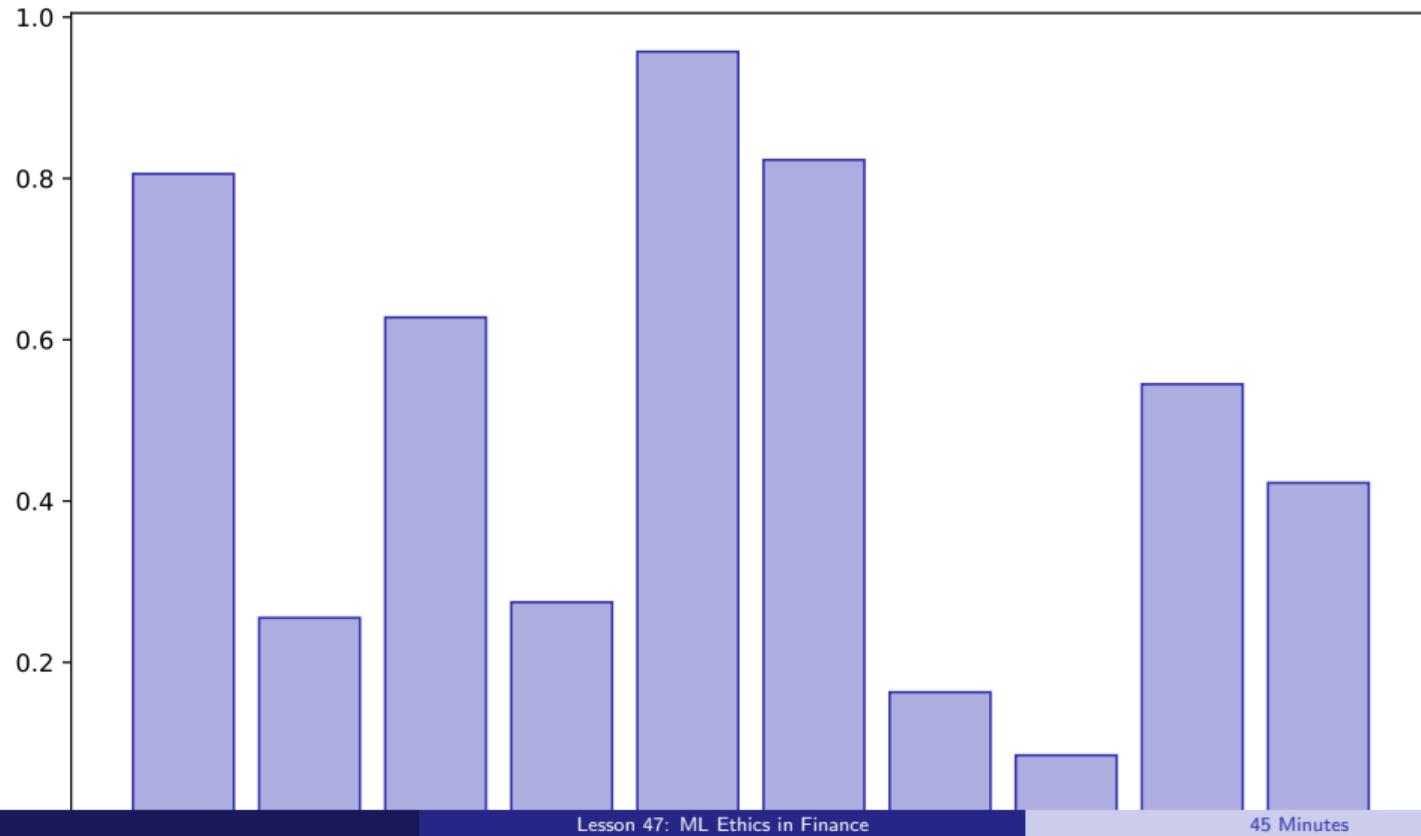
Explainability



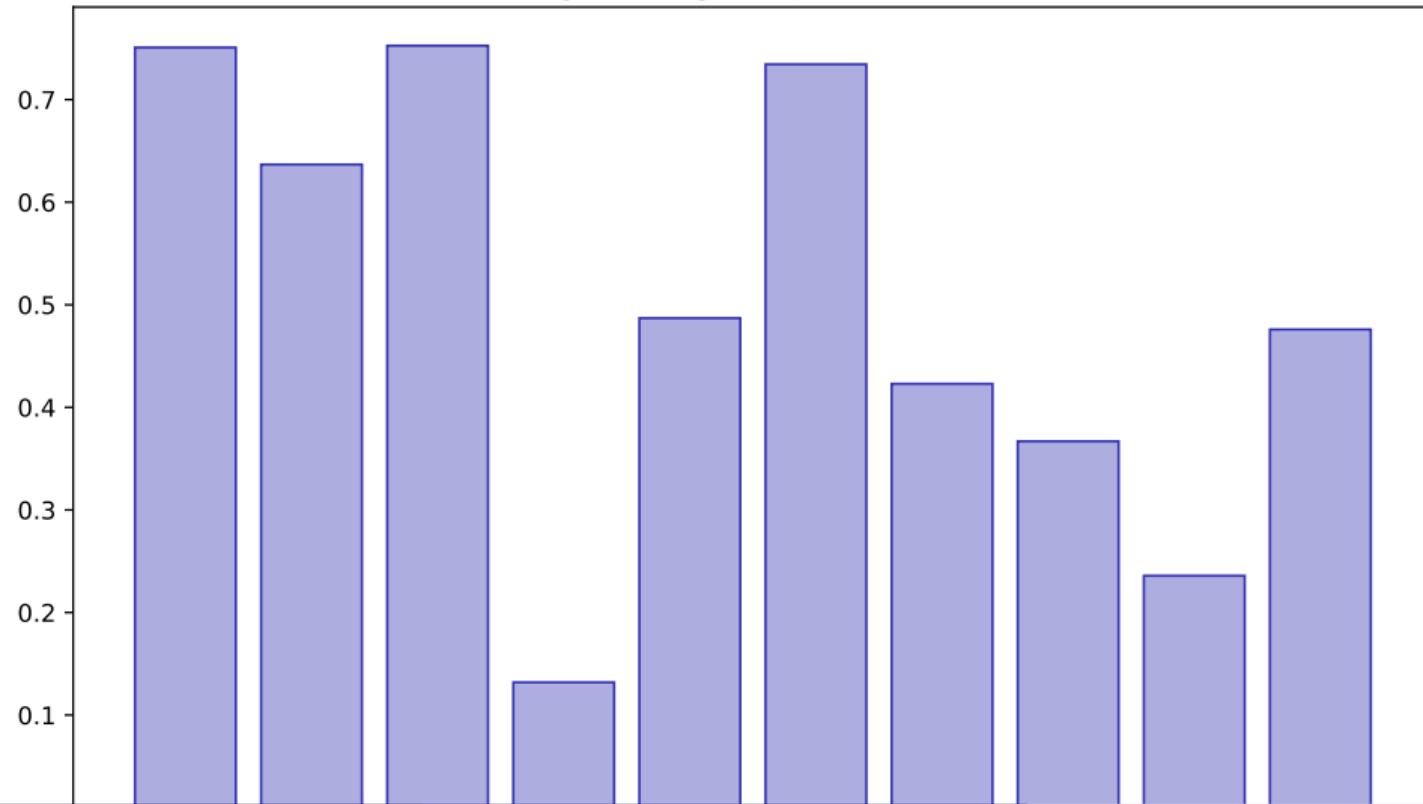
Shap Values



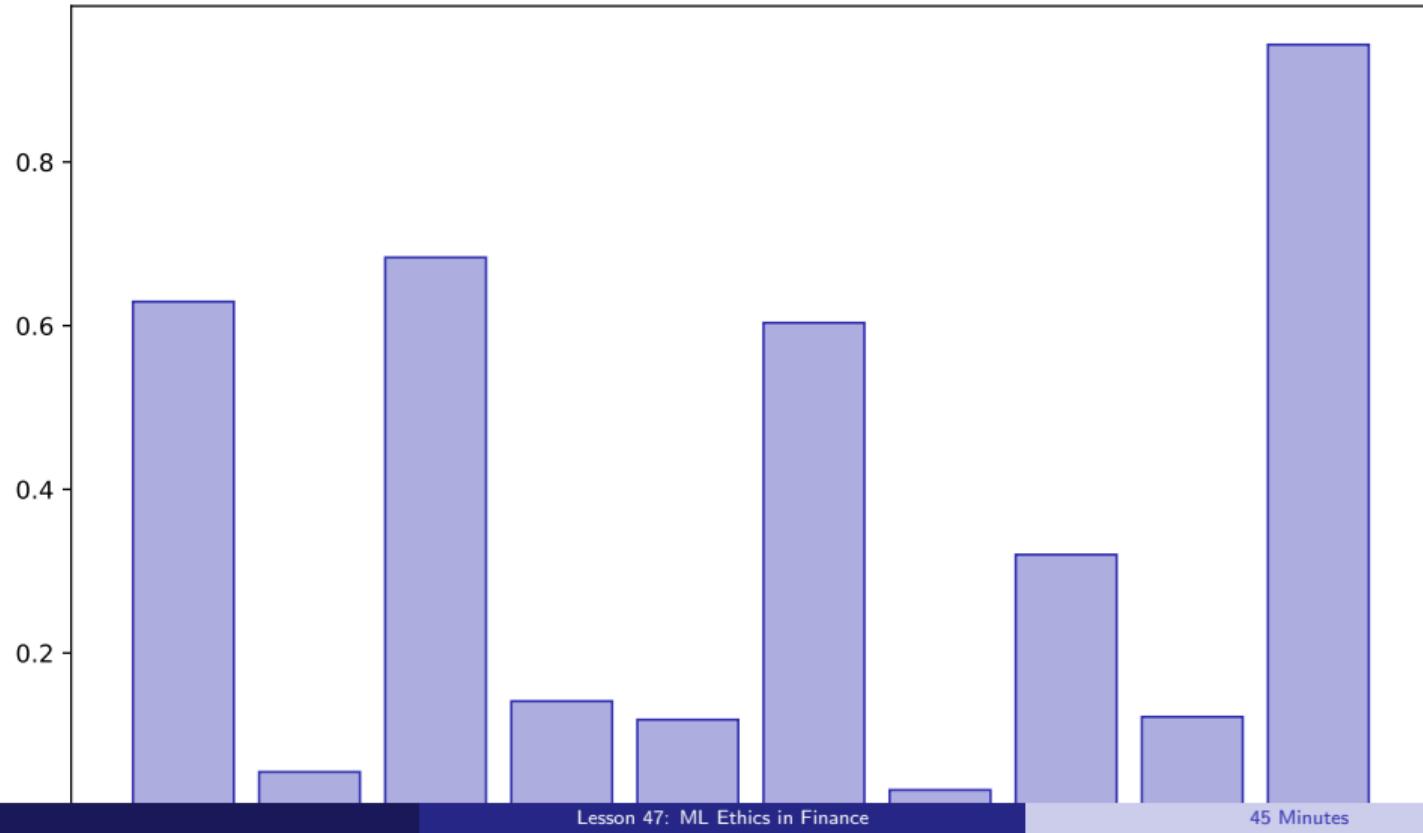
Fairness Metrics



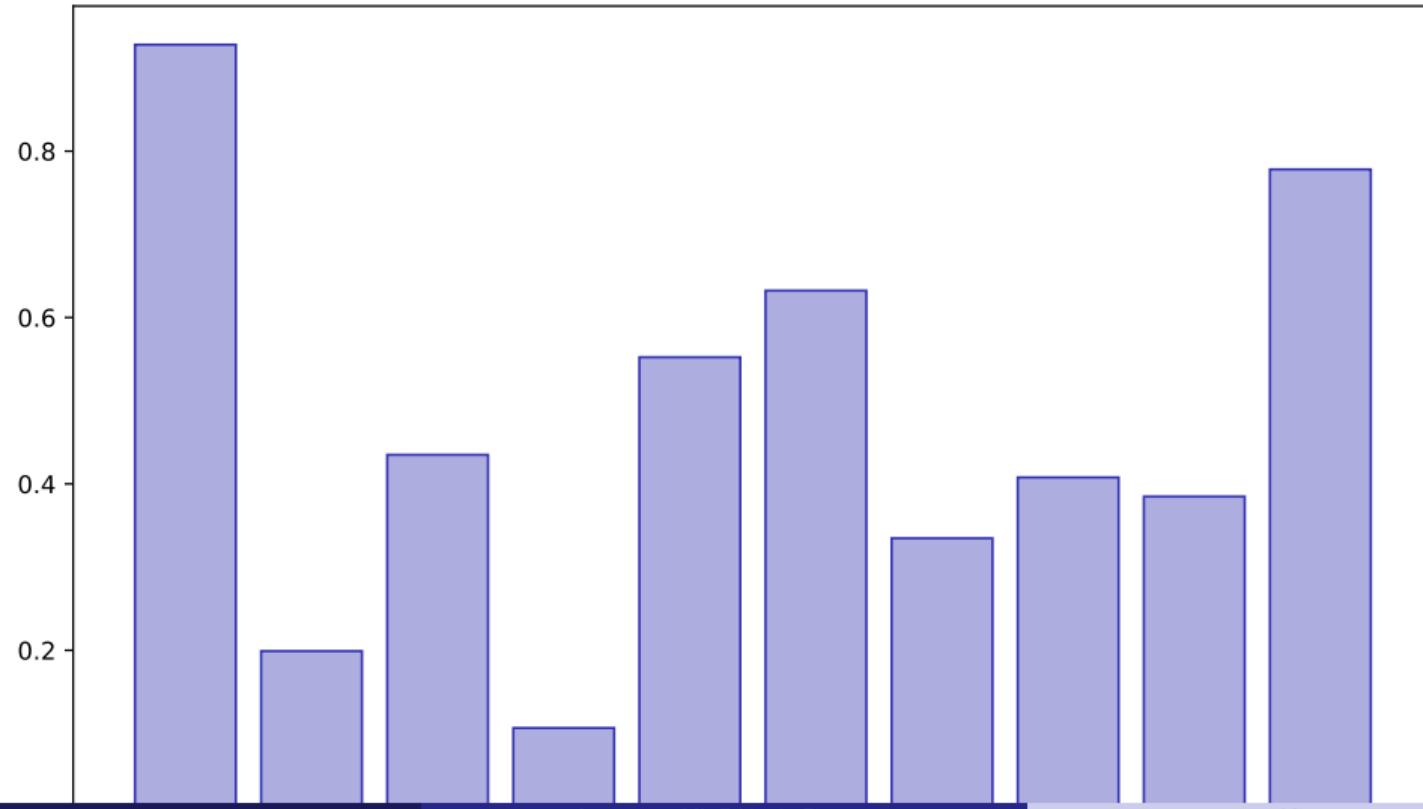
Regulatory Requirements



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

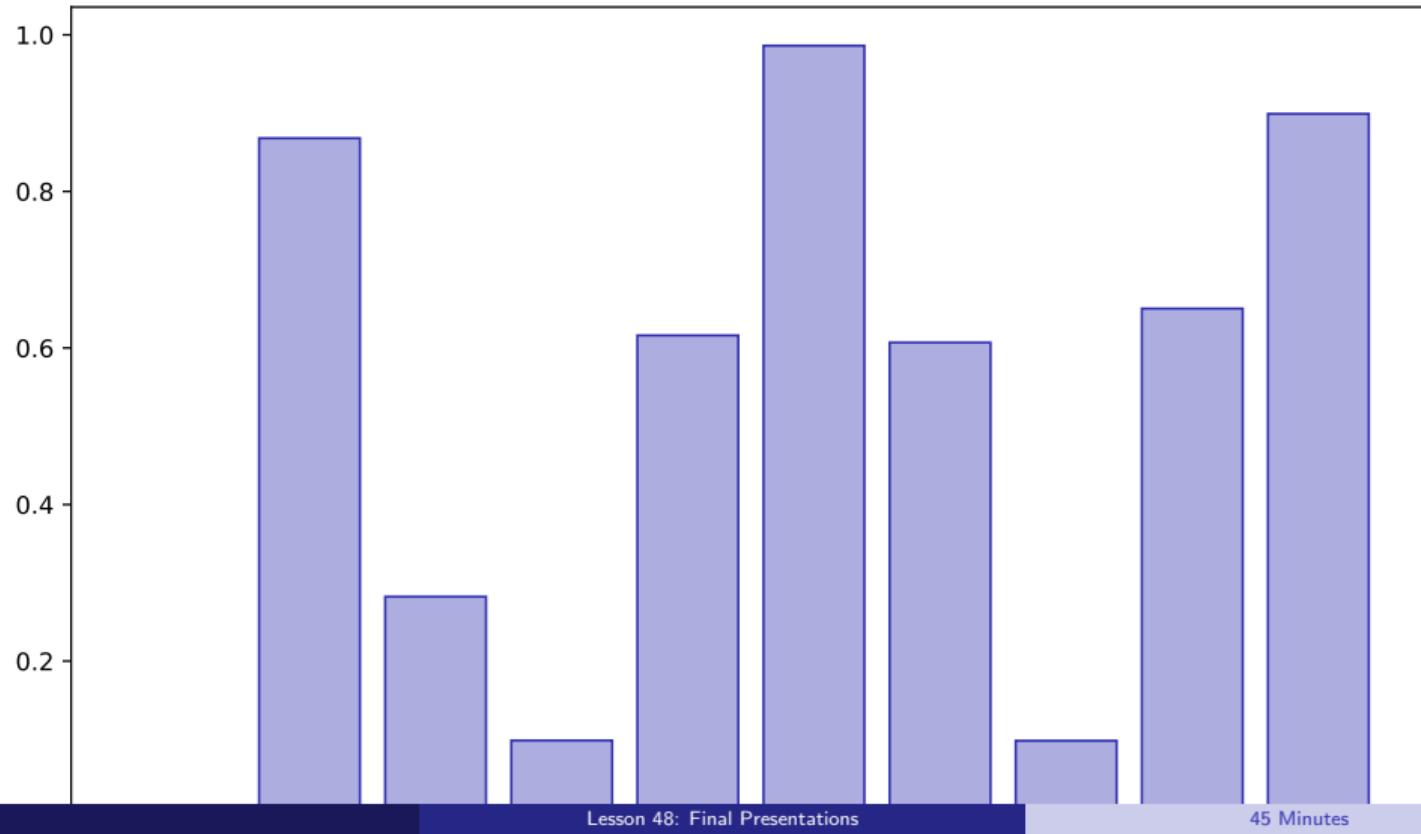
45 Minutes

After this lesson, you will be able to:

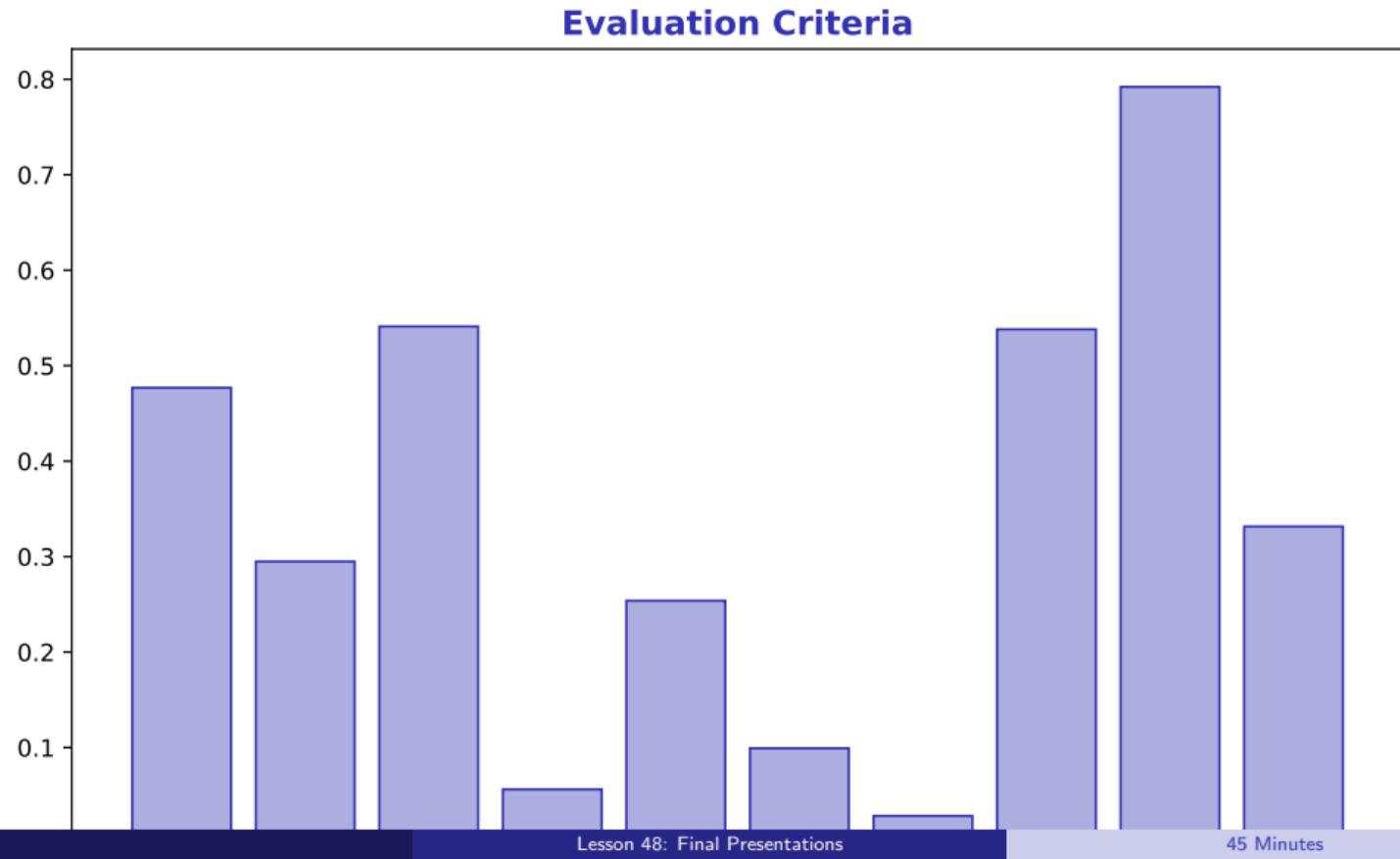
- Present projects
- Demonstrate models
- Answer questions
- Receive feedback

Building towards your final project

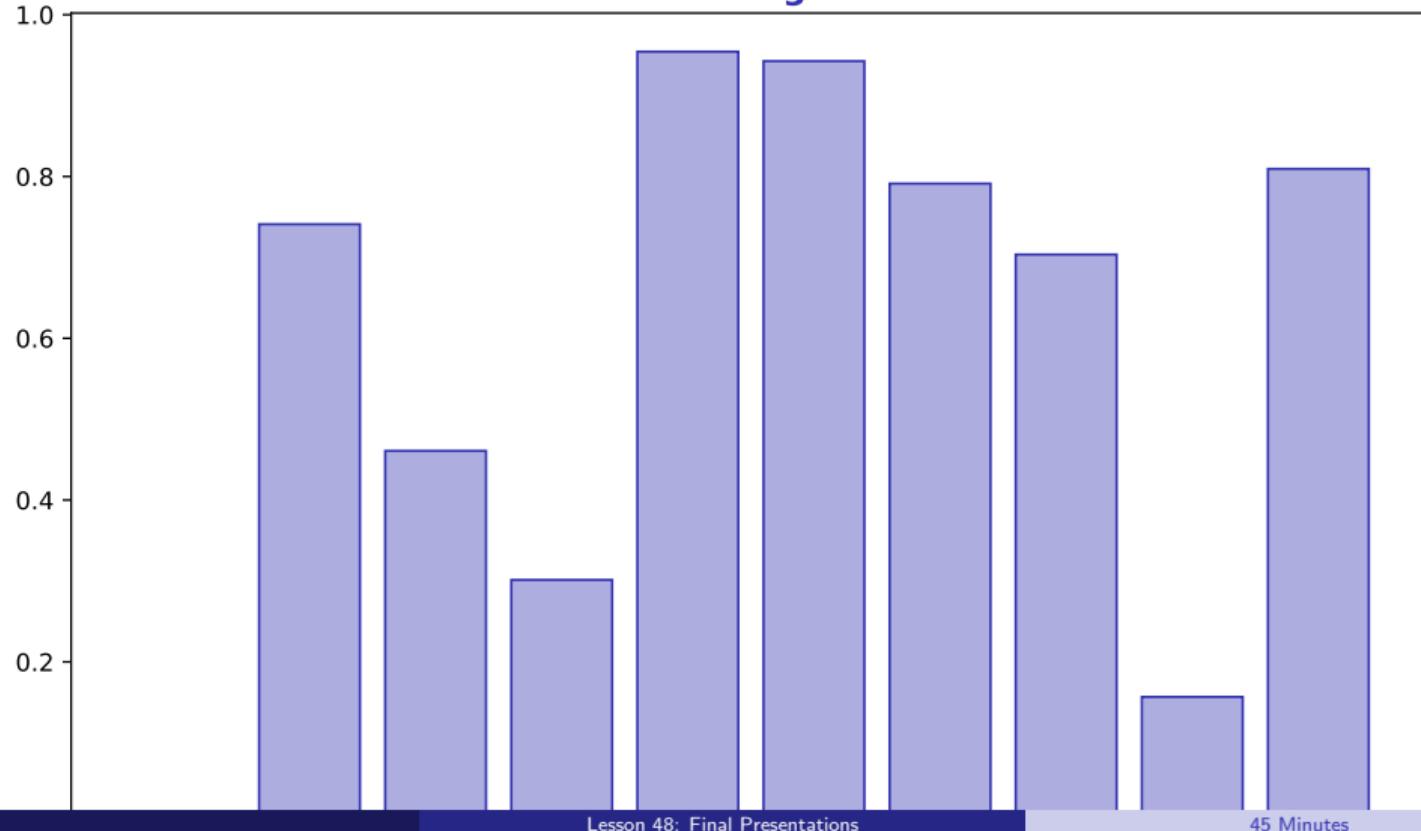
Presentation Guidelines



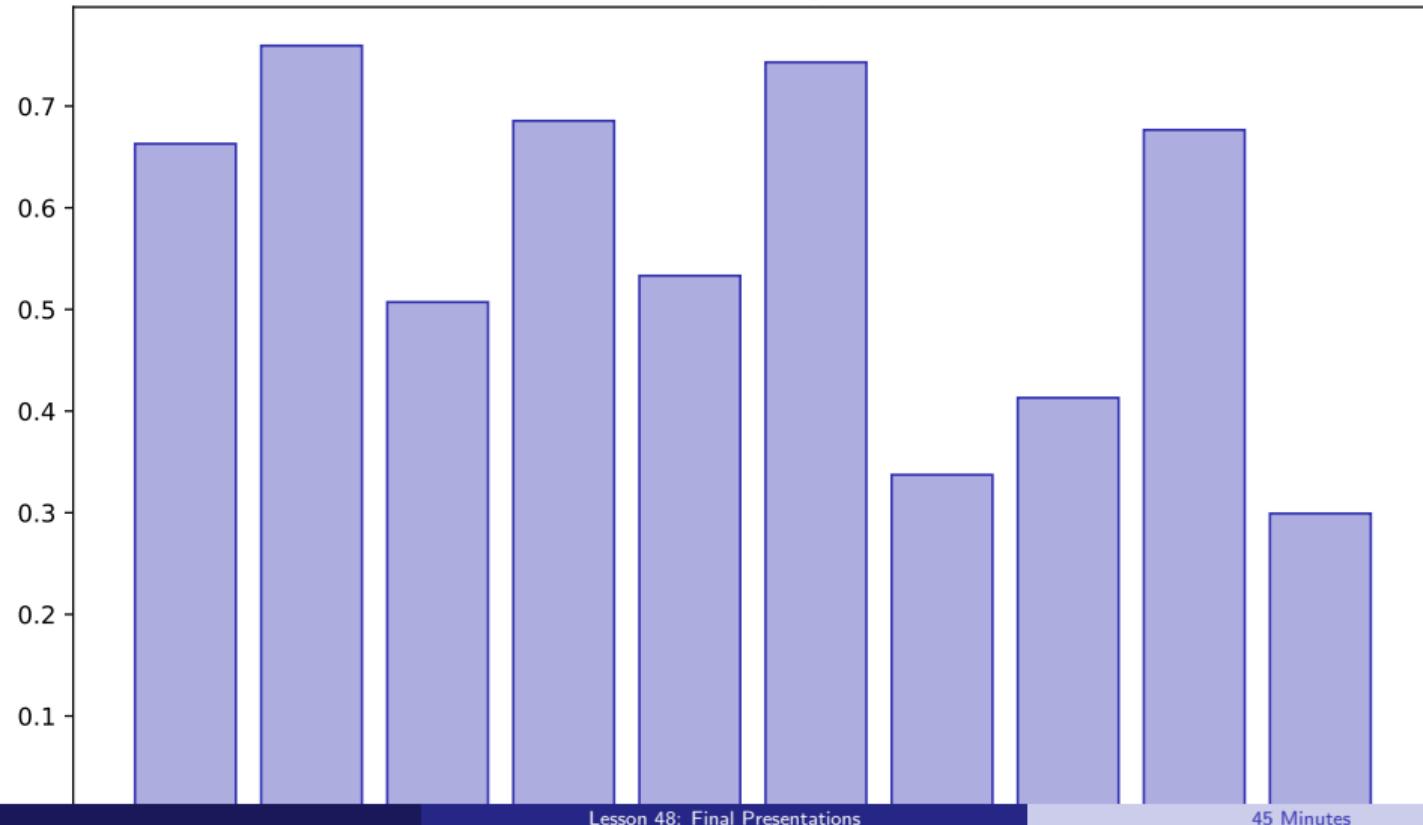
Evaluation Criteria



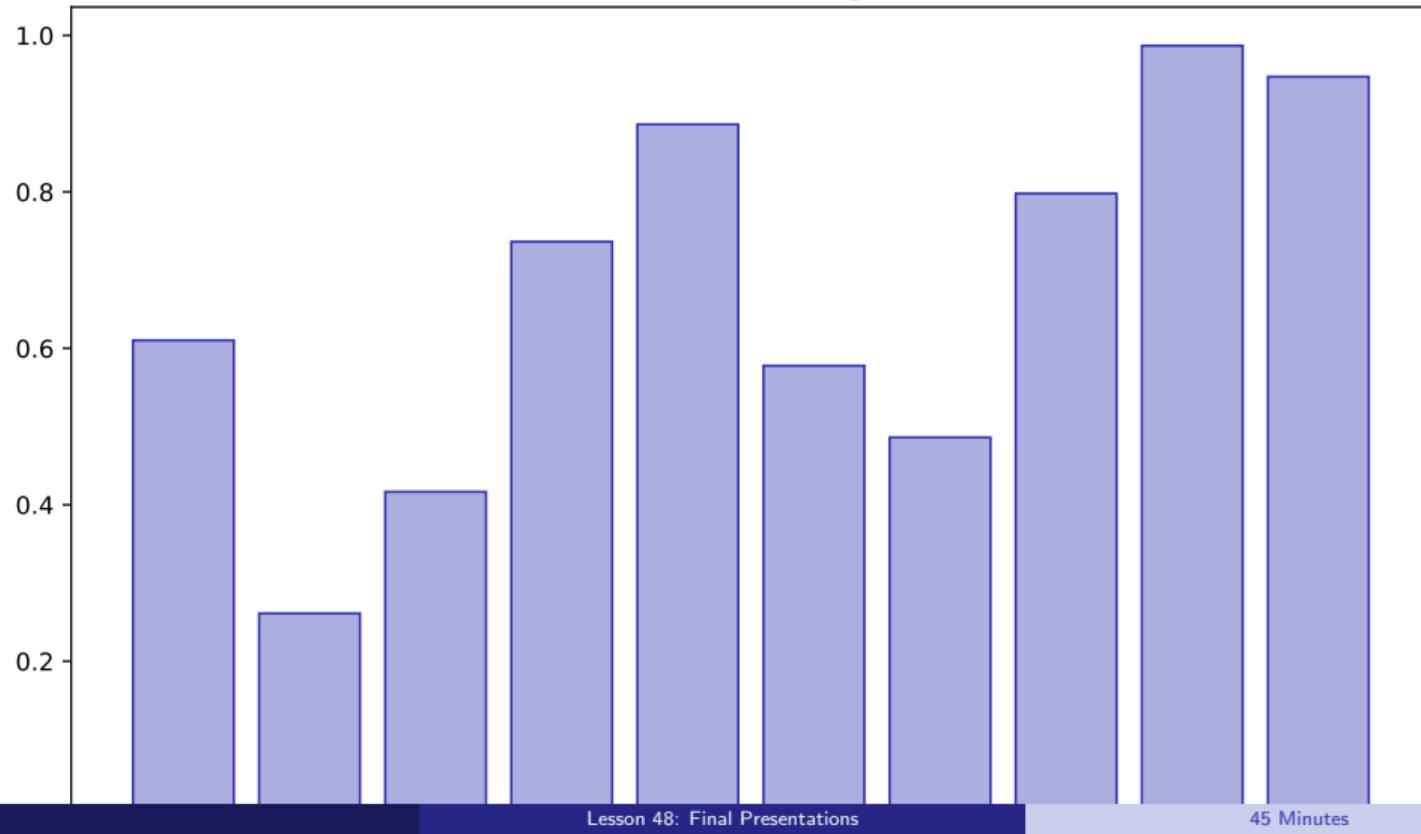
Time Management



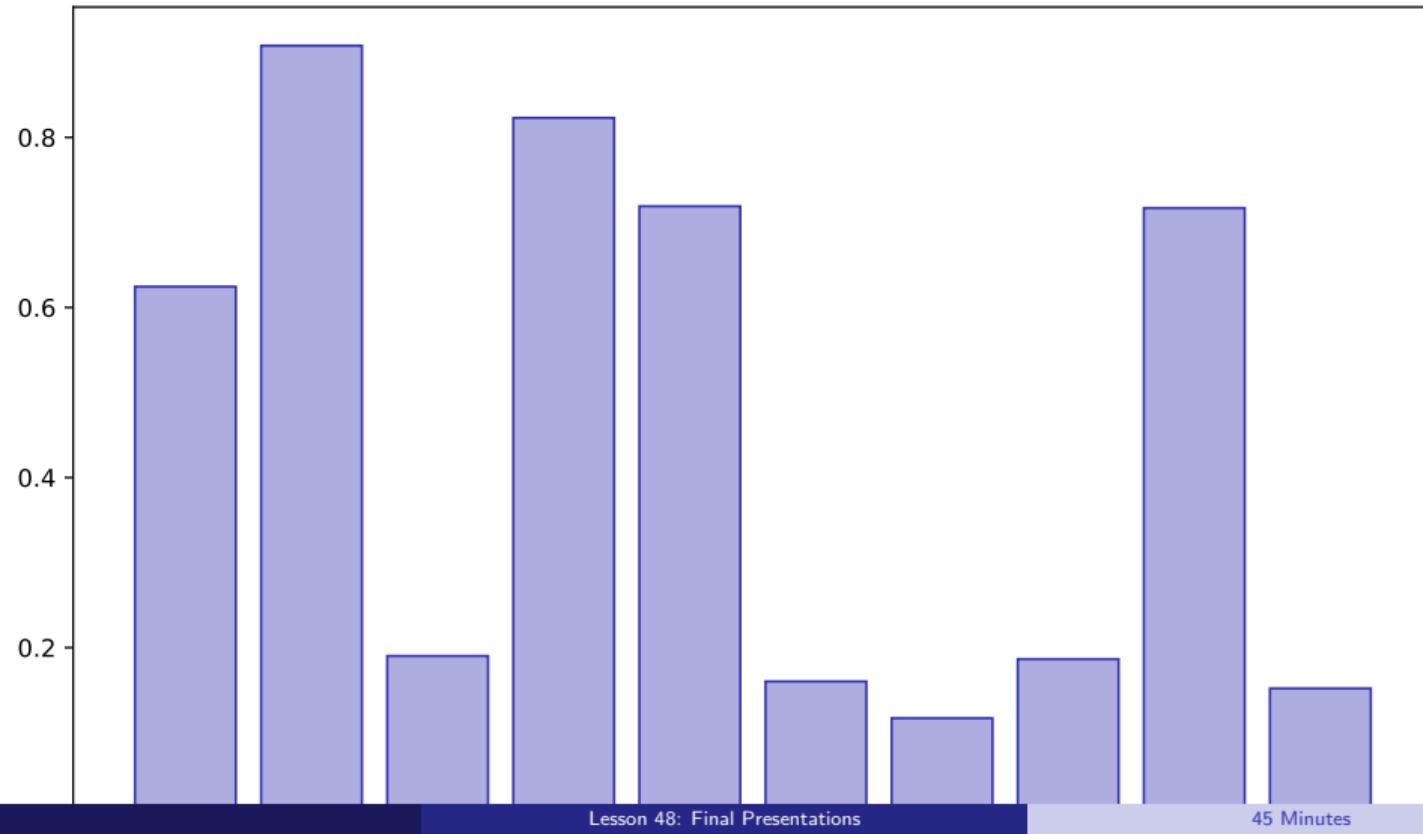
Demo Tips



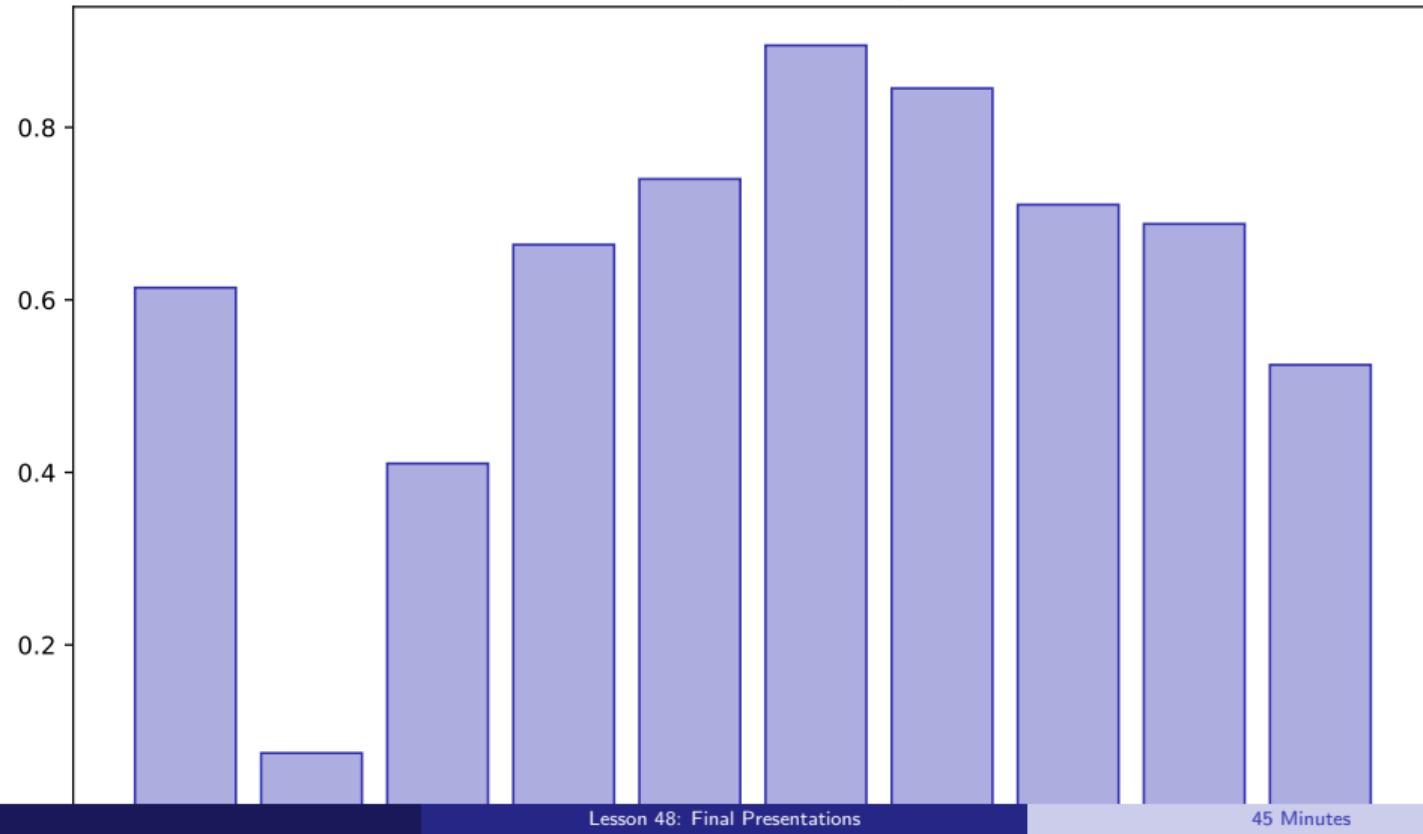
Qa Handling



Peer Evaluation

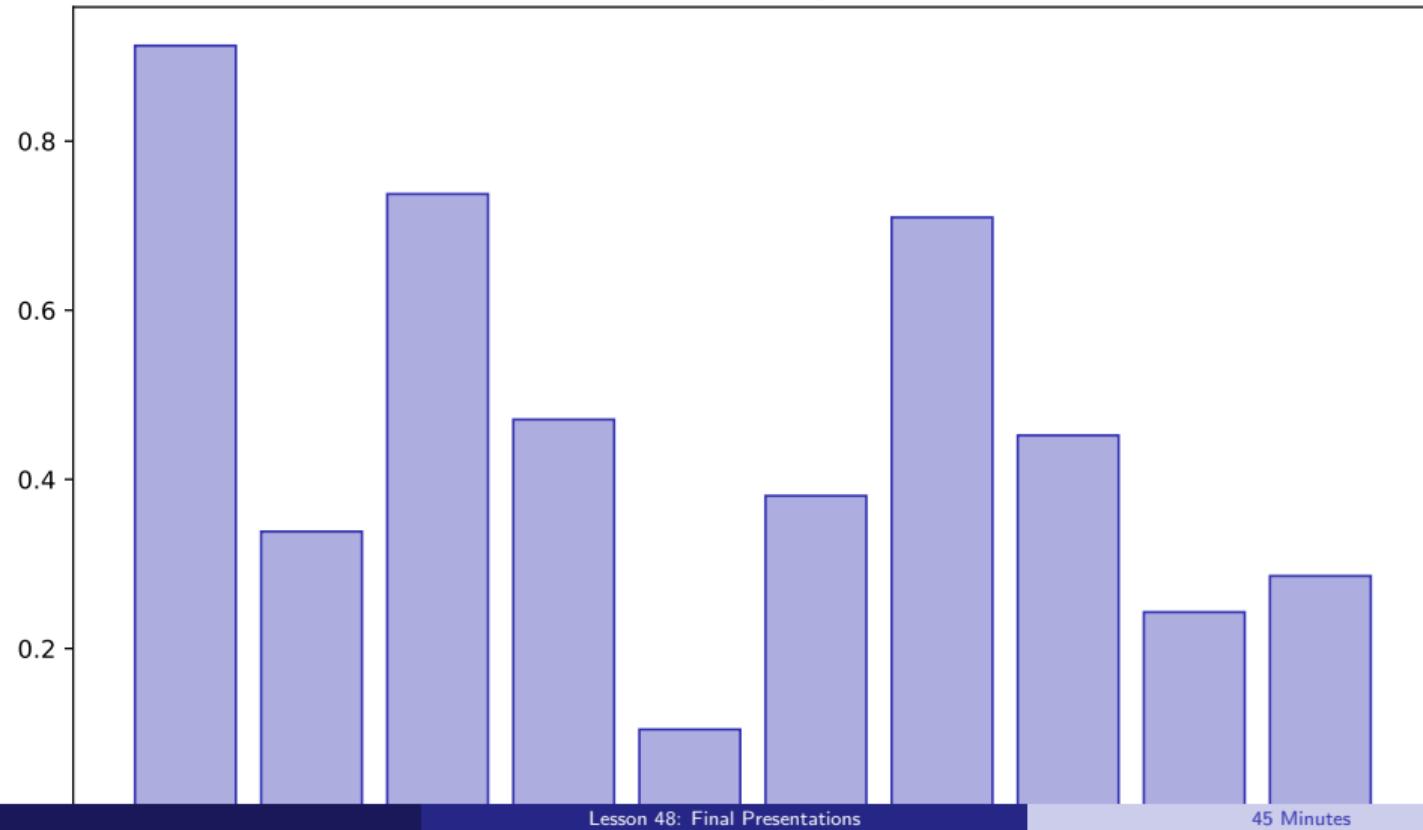


Course Summary



Next Steps

Next Steps



Lesson Summary

Key Takeaways:

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

Apply these skills in your final project