Session 1.2: Working with Text (Strings)

Text Processing for Beginners: Week 1, Day 2

Today's Journey

- Part 1: What Are Strings? Text Basics
- Part 2: Workshop - Text Processing Practice
- Break - You're Getting Good at This!
- Part 3: Git Workflow - Daily Practice
- Part 4: Workshop - Save Everything with Git
- Part 5: Quiz & Homework

Part 1: Strings - Working with Text

Text is everywhere in data!

What Are Strings? (Simple Explanation)

- String = any text in Python
- Examples:
  - Names: 'Sarah Johnson'
  - Emails: 'user@email.com'
  - Messages: 'Hello, World!'
  - Data labels: 'positive', 'negative'
- In ML, strings are everywhere:
  - Text classification (spam detection)
  - Sentiment analysis (positive/negative reviews)
  - Named Entity Recognition (finding names in text)

You must master strings for data science!

Creating Strings (Three Ways)

1. Single quotes: 'Hello'
2. Double quotes: "Hello"
3. Triple quotes: '''Hello''' or """Hello"""

Which to use?

- Single/double: Your choice (be consistent!)
- Use double when string has apostrophe:
  - "It's a beautiful day"
- Triple quotes: For long text (multiple lines)

All three create the exact same thing: a string!

String Examples - Your Turn to Try

```
# Single quotes (most common)
name = 'Alice'
job = 'Data Analyst'

# Double quotes (when you need apostrophes)
message = "It's Monday!"
sentence = "She said, 'Hello'"

# Triple quotes (long text)
paragraph = '''
This is a longer piece of text
that spans multiple lines.
Useful for descriptions!
'''
# Try these in your Python file!
```

Strings Are Sequences (Like a Necklace)

Think of strings like beads on a necklace

Each character has a position (index)

- Example: 'Python'
  - P y t h o n
  - 0 1 2 3 4 5 ← Position numbers (indices)
- Python counts from 0 (not 1!)
  - First character = position 0
  - Second character = position 1
  - And so on...
- This is called 'zero-indexing' - get used to it!

Accessing Individual Characters

```
word = 'Python'

# Get first letter (position 0)
print(word[0])    # P

# Get second letter (position 1)
print(word[1])    # y

# Get last letter (position 5)
print(word[5])    # n

# Shortcut: negative numbers count from end!
print(word[-1])   # n (last letter)
print(word[-2])   # o (second to last)
# Tip: Use negative indexing to get last characters easily
```

String Slicing - Extracting Pieces

- Slicing = getting a portion of a string
- Format: string[start:stop]
- Important rules:
  - start: included
  - stop: NOT included (goes up to, but not including)
  - Missing start means: from beginning
  - Missing stop means: to end
- Like cutting a piece of ribbon:
  - Start at position X, cut before position Y

Slicing Examples - Practice These!

```
text = 'Hello World'

# Get first 5 characters (positions 0-4)
print(text[0:5])    # 'Hello'

# Get from position 6 to end
print(text[6:])     # 'World'

# Get first 5 (shortcut - skip the 0)
print(text[:5])     # 'Hello'

# Get last 5 characters
print(text[-5:])    # 'World'
# This is CRITICAL for data processing!
# You'll use slicing constantly
```

Real-World Slicing Example

Imagine you have customer emails:

- email = 'john.smith@company.com'

Tasks you can do with slicing:

1. Extract username (before @)
2. Extract domain (after @)
3. Get file extension from filename
4. Extract dates from timestamps

This is real data processing!

Companies hire people who can do this!

Practical Slicing - Email Processing

```
email = 'sarah.johnson@techcompany.com'

# Find position of @
at_position = email.index('@')

# Result: 13
# Extract username (before @)
username = email[:at_position]
print(username)  # 'sarah.johnson'

# Extract domain (after @)
domain = email[at_position+1:]
print(domain)    # 'techcompany.com'

# This is data extraction!
# You're processing data like a pro!
```

String Methods - Built-In Tools

Python has pre-built tools for strings (methods)

Essential methods you'll use constantly:

- .upper() - Make all UPPERCASE
- .lower() - Make all lowercase
- .strip() - Remove spaces from ends
- .replace() - Replace text
- .split() - Cut into pieces

Format: string.method()

Example: name.upper()

These are like Excel functions but for text!

String Methods in Action

```
name = '  sarah johnson  '

# Remove spaces from ends
clean = name.strip()
print(clean)  # 'sarah johnson'

# Make uppercase
upper = clean.upper()
print(upper)  # 'SARAH JOHNSON'

# Make title case (First Letter Capital)
title = clean.title()
print(title)  # 'Sarah Johnson'

# Replace text
new = clean.replace('sarah', 'Sarah')
print(new)    # 'Sarah johnson'
```

Why String Cleaning Matters

Real data is MESSY:

- Bad data examples:
  - ' JOHN SMITH ' (extra spaces, all caps)
  - 'Alice   Johnson' (multiple spaces)
  - 'bob@EMAIL.COM' (inconsistent case)

Your job as data scientist:

1. Clean the data
2. Make it consistent
3. Process it correctly

String methods do this automatically!

Companies pay good money for this skill!

Data Cleaning Example

```
# Messy data from spreadsheet
raw_data = [
    ' JOHN  SMITH  ',
    ' alice  JOHNSON',
    'BOB  jones'
]

# Clean it up!
clean_data = []
for name in raw_data:
    # Remove spaces, make title case
    clean = name.strip().title()
    clean_data.append(clean)
print(clean_data)
# ['John Smith', 'Alice Johnson', 'Bob Jones']
# You just processed data professionally!
```

```
clean_data = [] # Another option for Cleaning the data
for name in raw_data:
    clean = " ".join(name.split()).title()
    clean_data.append(clean)
print(clean_data)
```

F-Strings - Modern Python Magic ✨

F-strings = best way to format text

Format: f'text {variable} more text'

Why use f-strings?

- Easy to read
- Fast to write
- Professional standard
- Industry best practice

You'll use these CONSTANTLY in ML:

- Logging training progress
- Reporting results
- Creating filenames
- Displaying metrics

F-Strings Are Incredible

```
# Old way (DON'T use this)
name = 'Sarah'
age = 32
message = 'My name is ' + name + ' and I am ' + str(age)

# F-STRING way (USE THIS!)
message = f'My name is {name} and I am {age}'

# Python handles everything automatically!
# Notice the 'f' before the quote
# Variables go in {curly braces}

# F-strings can even do math:
score = 87
print(f'You scored {score}% which is {score/100:.2f}')
# Output: You scored 87% which is 0.87
```

F-Strings for ML Progress

```
# Simulating ML training progress
epoch = 1 # Current full pass through dataset
total_epochs = 100
loss = 0.456 # Average prediction error
accuracy = 0.892 # 89.2% of correct predictions

# Professional logging (this is what real ML code looks like!)
print(f'Epoch {epoch}/{total_epochs}:')
print(f'  Loss: {loss:.3f}')
print(f'  Accuracy: {accuracy:.1%}')

# Output:
# Epoch 1/100:
#   Loss: 0.456
#   Accuracy: 89.2%
# This is professional ML output!
```

More Essential String Methods

- .split() - Cut string into list
- .join() - Combine list into string
- .startswith() - Check beginning
- .endswith() - Check ending
- 'text' in string - Check if contains

These are your text processing toolbox!

You'll use them every day in ML

Let's see them in action...

Split and Join - Processing Text

```
sentence = 'I love learning Python for ML'

# Split into words (creates a list)
words = sentence.split()
print(words)

# ['I', 'love', 'learning', 'Python', 'for', 'ML']
# Count words
word_count = len(words)
print(f'Word count: {word_count}')  # 6

# Join words back together
rejoined = ' '.join(words)
print(rejoined)
# 'I love learning Python for ML'
# This is text analysis!
```

Checking String Contents

```
filename = 'model_data.csv'

# Check if ends with .csv
if filename.endswith('.csv'):
    print('This is a CSV file!')

# Check if contains 'data'
if 'data' in filename:
    print('This file contains data')

# Check if starts with 'model'
if filename.startswith('model'):
    print('This is a model file')

# All three conditions are True!
# You're validating data programmatically!
```

Part 2: Workshop

Text Processing Practice

💻 Exercise 1: Name Formatter

Create file: name_formatter.py

You have messy names:

name1 = 'john smith'

name2 = ' ALICE JOHNSON '

name3 = 'bob   JONES'

Your task:

1. Clean each name (remove extra spaces)

2. Make proper title case (First Letters Capital)

3. Print formatted names

Expected output:

John Smith

Alice Johnson

Bob Jones

💻 Exercise 2: Email Validator

Create file: email_validator.py

Check if emails are valid:

email1 = 'user@company.com'

email2 = 'invalid.email'

email3 = ' USER@EMAIL.COM '

For each email:

1. Remove spaces
2. Convert to lowercase
3. Check if contains '@' and '.'
4. Print 'Valid' or 'Invalid'

Bonus: Extract username and domain from valid emails

This is real data validation!

Exercise 3: Text Statistics Tool

Create file: text_stats.py

Analyze this text:

```
text = 'I am learning Python and I love it!'
```

Calculate and print:

1. Total characters (including spaces)
2. Total characters (without spaces)
3. Number of words
4. Average word length
5. Uppercase version
6. Lowercase version

This is text analytics - NLP foundation!

Great Progress!

You're processing text like a pro! •

Part 3: Git Workflow

Daily Professional Practice

Review: The Three-Step Git Cycle

- Remember from last session:
    1. Edit - Write your code
    2. Stage - git add (prepare to save)
    3. Commit - git commit (actually save)

Today: Make this automatic

Goal: Git becomes muscle memory

By end of course: You won't think about it

It'll be as natural as Ctrl+S (Save)

The Daily Git Workflow

```
# Morning: Check where you are
git status

# After writing code:
git add .
git commit -m 'Add feature X'

# Before lunch:
git status  # See what you did

# Afternoon: More coding
git add .
git commit -m 'Fix bug in Y'

# End of day:
git log --oneline  # Review your work
# Commit 3-10 times per day!
```

When to Commit? (Simple Rules)

- ✅ COMMIT WHEN:
  - You finish a feature (even small!)
  - Code works (passes a test)
  - Before trying something risky
  - End of work session
  - Before taking a break
- ❌ DON'T COMMIT WHEN:
  - Code has errors (fix them first)
  - Half-way through a feature
  - You haven't tested it

Rule of thumb: 'Would I want to come back to this version?'

If yes → commit!

Writing Professional Commit Messages

- Your commit message is a note to future you
- Format: Start with a verb, be specific
- ✅ GOOD:
  - 'Add email validation function'
  - 'Fix bug in name formatting'
  - 'Update text cleaning logic'
  - 'Complete Exercise 2 from Session 1.2'
- ❌ BAD:
  - 'update'
  - 'changes'
  - 'stuff'
  - 'fixed it'

Good messages = professional portfolio!

Git Status - Your Best Friend

```
# ALWAYS use git status before any git command
git status

# It tells you:
# - What branch you're on
# - What files changed
# - What's staged (ready to commit)
# - What's not staged
# - What's new (untracked)

# Example output:
# On branch main
# Changes not staged:
#     modified: text_processor.py
# Untracked files:
#     new_exercise.py
# This guides you on what to do next!
```

Understanding git diff

git diff = see exactly what changed

Shows you:

- + Lines you added (green)
- - Lines you removed (red)
- Context around changes

Use before committing:

- 'Let me review what I changed...'
- 'Make sure I want to save this...'

Professional habit:

- git diff → git add → git commit

Viewing Your History

```
# See all commits
git log

# Easier to read (one line per commit)
git log --oneline

# Output example:
# a1b2c3d Add email validator
# e4f5g6h Complete text exercises
# i7j8k9l Add name formatter

# See last 5 commits only
git log -5

# This is your work timeline!
# Shows your progress
# Proves you did the work (portfolio!)
```

Fixing Common Git Mistakes

Everyone makes mistakes! Git helps you fix them:

- Added wrong file?
    - git restore --staged filename.py
- Want to undo changes to a file?
    - git restore filename.py
- Typo in commit message?
    - (We'll learn this later)

Remember: Git protects you!

Committed code is safe forever

Git Best Practices (Memorize These!)

1. *git status* before EVERY command
    - (Seriously, every single time)
2. *Review* with git diff before committing
    - (Know what you're saving)
3. *Write clear* commit messages
    - (Future you will thank you)
4. *Commit often* (3-10 times per day)
    - (Small, focused commits)
5. *Never commit* broken code
    - (Test first!)

Part 4: Workshop

Git Mastery Practice

💻 Exercise 4: Complete Git Cycle

Save ALL your work from today with Git!

Step 1: Set up (5 min)

- Create folder: session_1_2_work
- cd into folder
- git init

Step 2: Save exercises (10 min)

- Add name_formatter.py
- git add, git commit with message
- Add email_validator.py
- git add, git commit
- Add text_stats.py
- git add, git commit

Goal: Git becomes automatic!

Step 3: Make changes (5 min)

- Edit one file (add more features)
- git diff (see changes)
- git add, git commit
- git log --oneline (see history)

Part 5: Quiz & Wrap-up

Session Complete!

Session 1.2 Quiz

🎉 Amazing Progress! Session 1.2 Complete

What you accomplished today:

- ✓ Mastered string basics (creation, indexing, slicing)
- ✓ Used string methods (upper, lower, strip, split)
- ✓ Learned f-strings (professional formatting)
- ✓ Cleaned real messy data
- ✓ Built text processing tools
- ✓ Made Git workflow automatic

Week 1 COMPLETE! 🎊

You're officially a programmer who uses version control!

That's resume-worthy! 💼

Homework - Build Your Skills!

1. Complete the quiz (10 questions - required)
2. Project: Build a Text Analyzer
    - Create text_analyzer.py that:
        - Takes any text input
        - Cleans it (strip, lowercase)
        - Reports statistics (length, words, etc.)
        - Finds specific words
        - Formats output professionally with f-strings
3. Git Practice
    - Save your analyzer with Git
    - Make 3 improvements
    - Commit after each improvement
    - Review with git log
4. Prepare for Week 2: Read about lists (next topic)

Looking Ahead: Week 2 - Lists!

Next session: Session 2.1 - Lists Fundamentals

What are lists?

- – • Collections of values (like Excel columns)
- – • Store multiple items together
- – • Foundation for data science

- Why they matter:
  - – • ML works with collections of data
  - – • Lists → NumPy arrays → Tensors
  - – • You can't do ML without mastering lists!

Get excited - lists unlock everything! 🚀

Week 1 Reflections 💭

Take a moment to appreciate what you've learned:

- Two weeks ago: 'I don't know how to code'
- Now: 'I write Python programs and use Git!'

You learned:

- Variables, data types, operations
- String processing (critical skill!)
- Git workflow (professional tool!)
- Text cleaning (real data work!)

Next: Lists, loops, and real data processing