## Contents

Tutorial 5: VS Code Power User for Finance . . . 1
☐ What You'll Learn (90 minutes) . . . 1
Part 1: Keyboard Shortcuts Mastery (15 minutes) . . . 1
Part 2: Code Snippets for Finance (15 minutes) . . . 3
Part 3: Workspace Optimization (15 minutes) . . . 5
Part 4: Advanced Debugging (15 minutes) . . . 8
Part 5: Essential Extensions for Finance (10 minutes) . . . 9
Part 6: Advanced Search & Replace (10 minutes) . . . 10
Part 7: Productivity Hacks (10 minutes) . . . 11
Part 8: Advanced Jupyter Tricks (10 minutes) . . . 12
Part 9: Collaboration Features (10 minutes) . . . 13
Part 10: Automation & Scripts (10 minutes) . . . 13
☐ Skills Checklist . . . 15
☐ Ultimate Shortcuts Reference Card . . . 15
☐ Pro Tips Summary . . . 16
☐ What's Next? . . . 17
☐ Bonus: Personal Productivity System . . . 17

# Tutorial 5: VS Code Power User for Finance

## ☐ What You'll Learn (90 minutes)

Master advanced VS Code techniques to become 10x more productive: - Advanced keyboard shortcuts and navigation - Multi-cursor editing for financial models - Code snippets and custom templates - Workspace organization and settings - Debugging Python financial code - Extensions ecosystem for finance - Remote development and collaboration - Automation and task running - Professional workflow optimization

Prerequisites: - Completed Tutorials 1-4 - Comfortable with basic VS Code usage - Ready to level up!

---

## Part 1: Keyboard Shortcuts Mastery (15 minutes)

### Essential Navigation

| Shortcut | Action | Finance Use Case |
|---|---|---|
| `Ctrl+P` | Quick open file | Jump to `dcf_model.py` |
| `Ctrl+Shift+P` | Command palette | Access any command |
| `Ctrl+Tab` | Switch between tabs | Navigate between files |
| `Ctrl+\` | Split editor | Code + Notebook side-by-side |
| `Ctrl+B` | Toggle sidebar | More screen space |
| `Alt+Z` | Toggle word wrap | Long financial formulas |
| `Ctrl+G` | Go to line | Jump to line 247 |
| `Ctrl+Shift+O` | Go to symbol | Find `calculate_wacc()` |

### Advanced Editing

| Shortcut | Action | Finance Use Case |
|---|---|---|
| `Alt+↑/↓` | Move line up/down | Reorder assumptions |

| Shortcut | Action | Finance Use Case |
|---|---|---|
| `Shift+Alt+↑/↓` | Copy line up/down | Duplicate year projections |
| `Ctrl+Shift+K` | Delete line | Remove old code quickly |
| `Ctrl+/` | Toggle comment | Comment out test code |
| `Ctrl+[` | Outdent | Fix indentation |
| `Ctrl+]` | Indent | Organize code blocks |
| `Ctrl+Shift+[` | Fold code | Collapse long functions |
| `Ctrl+Shift+]` | Unfold code | Expand collapsed code |

## Multi-Cursor Magic ☐

This is a GAME CHANGER for financial modeling!

| Shortcut | Action |
|---|---|
| `Alt+Click` | Add cursor |
| `Ctrl+Alt+↑/↓` | Add cursor above/below |
| `Ctrl+D` | Select next occurrence |
| `Ctrl+Shift+L` | Select all occurrences |
| `Alt+Shift+I` | Cursor at end of each line |
| `Esc` | Exit multi-cursor mode |

### Practice: Multi-Cursor Editing

Create file: `multi_cursor_practice.py`

```python
# Copy this:
revenue_2020 = 100
revenue_2021 = 120
revenue_2022 = 150
revenue_2023 = 180
revenue_2024 = 220
```

Task 1: Rename all at once 1. Put cursor on first `revenue` 2. Press `Ctrl+D` four times (selects all `revenue`) 3. Type `sales` - all renamed!

Task 2: Add calculations 1. Select all 5 lines 2. Press `Alt+Shift+I` (cursor at end of each line) 3. Type `* 1.10` - adds to all lines!

Task 3: Create from template

```python
# Type one line:
year_1 =

# Place cursor on `1`
# Press Ctrl+Alt+↓ four times (5 cursors)
# Press End, then type = 0
# Press Home, select "1"
# Type 1, then down arrow, type 2, down, 3, etc.
```

Result:

```python
year_1 = 0
year_2 = 0
year_3 = 0
```

```
year_4 = 0
year_5 = 0
```

☐ Use multi-cursor for: - Updating projection years - Adding consistent formulas - Batch renaming variables - Creating data structures

---

## Part 2: Code Snippets for Finance (15 minutes)

### Create Custom Snippets

`Ctrl+Shift+P` → "Preferences: Configure User Snippets" → python.json

```
{
  "DCF Model Template": {
    "prefix": "dcf",
    "body": [
      "class DCFModel:",
      "    \"\"\"Discounted Cash Flow Valuation Model\"\"\"",
      "    ",
      "    def __init__(self, company_name: str):",
      "        self.company_name = company_name",
      "        self.projection_years = 5",
      "        self.projections = None",
      "        ",
      "    def calculate_fcf(self):",
      "        \"\"\"Calculate free cash flow\"\"\"",
      "        ${1:pass}",
      "        ",
      "    def calculate_wacc(self):",
      "        \"\"\"Calculate weighted average cost of capital\"\"\"",
      "        ${2:pass}",
      "        ",
      "    def calculate_enterprise_value(self):",
      "        \"\"\"Calculate enterprise value\"\"\"",
      "        ${3:pass}",
      "$0"
    ],
    "description": "DCF model class template"
  },

  "Financial Imports": {
    "prefix": "finimp",
    "body": [
      "import pandas as pd",
      "import numpy as np",
      "import matplotlib.pyplot as plt",
      "import yfinance as yf",
      "from typing import Dict, List, Tuple",
      "$0"
    ],
    "description": "Standard financial analysis imports"
  },

  "Calculate Growth Rate": {
    "prefix": "cagr",
    "body": [
      "def calculate_cagr(beginning_value: float, ending_value: float, periods: int) -> float:",
      "    \"\"\"Calculate Compound Annual Growth Rate\"\"\"",
      "    return (ending_value / beginning_value) ** (1 / periods) - 1",
      "$0"
    ],
    "description": "CAGR calculation function"
  },

  "Financial Ratio": {
    "prefix": "ratio",
    "body": [
      "def calculate_${1:ratio_name}(${2:numerator}: float, ${3:denominator}: float) -> float:",
      "    \"\"\"Calculate ${1:ratio_name}\"\"\"",
      "    if ${3:denominator} == 0:",
      "        return np.nan",
      "    return ${2:numerator} / ${3:denominator}",
      "$0"
    ],
    "description": "Financial ratio template"
  },

  "Pandas DataFrame": {
    "prefix": "dfin",
    "body": [
      "${1:df} = pd.DataFrame({",
      "    'Year': [${2:2020, 2021, 2022, 2023, 2024}],",
      "    '${3:Revenue}': [${4:100, 120, 150, 180, 220}],",
      "    '${5:EBITDA}': [${6:20, 28, 38, 50, 66}]",
      "})",
      "$0"
    ],
    "description": "Financial DataFrame template"
  },

  "Excel Export": {
    "prefix": "xlsx",
    "body": [
      "with pd.ExcelWriter('${1:output}.xlsx', engine='openpyxl') as writer:",
      "    ${2:df}.to_excel(writer, sheet_name='${3:Sheet1}', index=${4:False})",
      "    ${5:# Add more sheets}",
      "$0"
    ],
    "description": "Export to Excel template"
  },

  "Stock Data Download": {
    "prefix": "yf",
    "body": [
      "import yfinance as yf",
      "",
      "ticker = '${1:AAPL}'",
      "stock = yf.Ticker(ticker)",
      "df = stock.history(period='${2:1y}')",
      "info = stock.info",
      "$0"
    ],
    "description": "Download stock data"
  },

  "Plotting Template": {
    "prefix": "plotfin",
    "body": [
      "plt.figure(figsize=(${1:12}, ${2:6}))",
      "plt.plot(${3:x}, ${4:y}, label='${5:Label}', linewidth=2)",
      "plt.title('${6:Title}', fontsize=14, fontweight='bold')",
      "plt.xlabel('${7:X Label}')",
      "plt.ylabel('${8:Y Label}')",
      "plt.legend()",
      "plt.grid(True, alpha=0.3)",
      "plt.tight_layout()",
      "plt.show()",
      "$0"
    ],
    "description": "Financial chart template"
  }
}
```

Use Snippets

Create new file: `test_snippets.py`

1. Type `finimp` + `Tab` → Auto-imports!
2. Type `dcf` + `Tab` → DCF class template!
3. Type `cagr` + `Tab` → CAGR function!
4. Type `dfin` + `Tab` → DataFrame template!

Customize for your workflow!

---

## Part 3: Workspace Optimization (15 minutes)

Multi-Root Workspace

Organize multiple projects:

`Ctrl+Shift+P` → “Add Folder to Workspace”

```
My Finance Workspace/
  DCF-Models/
     tech_company_dcf.py
     retail_dcf.py
  LBO-Models/
     lbo_template.py
  Data-Analysis/
     market_analysis.ipynb
  Utilities/
     helpers.py
```

Save as: `File` → `Save Workspace As...` → `finance_workspace.code-workspace`

## Workspace Settings

Create: `.vscode/settings.json` in workspace root

```
{
  "python.defaultInterpreterPath": "./venv/Scripts/python.exe",
  "python.formatting.provider": "black",
  "python.linting.enabled": true,
  "python.linting.pylintEnabled": true,
  "editor.formatOnSave": true,
  "editor.rulers": [88, 120],
  "files.exclude": {
    "**/__pycache__": true,
    "**/*.pyc": true,
    ".pytest_cache": true,
    "venv": true
  },
  "files.autoSave": "afterDelay",
  "files.autoSaveDelay": 1000,
  "editor.minimap.enabled": false,
  "workbench.colorTheme": "Visual Studio Dark",
  "terminal.integrated.defaultProfile.windows": "PowerShell",
  "jupyter.askForKernelRestart": false
}
```

## Launch Configurations

Create: `.vscode/launch.json`

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Run DCF Model",
      "type": "python",
      "request": "launch",
      "program": "${workspaceFolder}/dcf_model.py",
      "console": "integratedTerminal",
      "justMyCode": true
    },
    {
      "name": "Debug Current File",
      "type": "python",
      "request": "launch",
      "program": "${file}",
      "console": "integratedTerminal"
    },
    {
      "name": "Run Tests",
      "type": "python",
      "request": "launch",
      "module": "pytest",
      "args": [
        "tests/",
        "-v"
      ],
      "console": "integratedTerminal"
    }
  ]
}
```

Press `F5` to run configured tasks!

## Custom Tasks

Create: `.vscode/tasks.json`

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "Run DCF Analysis",
      "type": "shell",
      "command": "python",
      "args": ["${workspaceFolder}/dcf_model.py"],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "presentation": {
        "reveal": "always",
        "panel": "new"
      }
    },
    {
      "label": "Export All Models to Excel",
      "type": "shell",
      "command": "python",
      "args": ["${workspaceFolder}/batch_export.py"],
      "problemMatcher": []
    },
    {
      "label": "Update Market Data",
      "type": "shell",
      "command": "python",
      "args": ["-c", "import yfinance as yf; # download script"],
      "problemMatcher": []
    }
  ]
}
```

Run tasks: `Ctrl+Shift+P` → “Tasks: Run Task”

## Part 4: Advanced Debugging (15 minutes)

### Debug Financial Code

Create: `debug_example.py`

```python
import pandas as pd

def calculate_dcf(cash_flows: list, wacc: float, terminal_value: float) -> float:
    """Calculate DCF with debugging example"""

    pv_cash_flows = 0

    for year, cf in enumerate(cash_flows, start=1):
        discount_factor = (1 + wacc) ** year
        pv = cf / discount_factor
        pv_cash_flows += pv

        # Set breakpoint on next line
        print(f"Year {year}: CF={cf}, DF={discount_factor:.3f}, PV={pv:.2f}")

    pv_terminal = terminal_value / (1 + wacc) ** len(cash_flows)
    enterprise_value = pv_cash_flows + pv_terminal

    return enterprise_value

# Test
cash_flows = [100, 120, 150, 180, 220]
wacc = 0.10
terminal_value = 3000

result = calculate_dcf(cash_flows, wacc, terminal_value)
print(f"\nEnterprise Value: ${result:.2f}")
```

### Using Debugger

1. Set breakpoint: Click left of line number (red dot appears)
2. Start debugging: Press `F5`
3. Debug controls:
    - `F10` - Step Over (next line)
    - `F11` - Step Into (enter function)
    - `Shift+F11` - Step Out (exit function)
    - `F5` - Continue
    - `Shift+F5` - Stop
4. Inspect variables:
    - Hover over variable to see value
    - Check Variables panel (left side)
    - Check Watch panel (add expressions)
5. Debug Console:
    - Type variable names
    - Execute code: `pv_cash_flows + 100`
    - Test expressions: `wacc * 2`

### Conditional Breakpoints

Right-click breakpoint → Edit Breakpoint → Expression

```
year == 3  # Only break on year 3
cf > 150   # Only break when cash flow > 150
```

### Logpoints

Right-click breakpoint → Add Logpoint

```
Year {year}: CF={cf}, PV={pv}
```

Logs without stopping execution!

---

## Part 5: Essential Extensions for Finance (10 minutes)

### Must-Have Extensions

Already Installed: - ☐ Python (Microsoft) - ☐ Jupyter (Microsoft) - ☐ GitHub Copilot

Install These:

1. Excel Viewer
   - `Ctrl+Shift+X` → Search "Excel Viewer"
   - View .xlsx files in VS Code
   - Edit CSV files with table interface
2. Rainbow CSV
   - Colorizes CSV columns
   - Makes data files readable
   - Query CSV with SQL!
3. Better Comments
   - Color-coded comments
   - `# ! Important note` → Red
   - `# ? Question` → Blue
   - `# TODO Task` → Orange
4. Bookmarks
   - Mark important code locations
   - Jump between bookmarks
   - Perfect for large models
5. Code Spell Checker
   - Catches typos in comments
   - Professional documentation
   - Add finance terms to dictionary
6. GitLens
   - See who changed what
   - Inline git blame
   - Commit history visualization
7. Python Docstring Generator
   - Auto-generate docstrings
   - Type `"""` and press Enter
   - Professional documentation

### Extension Settings

Add to `settings.json`:

```
{
  "better-comments.tags": [
    {
      "tag": "!",
      "color": "#FF2D00",
      "strikethrough": false,
      "backgroundColor": "transparent"
    },
    {
      "tag": "?",
      "color": "#3498DB",
      "strikethrough": false,
      "backgroundColor": "transparent"
    },
    {
      "tag": "FINANCE",
      "color": "#00FF00",
      "strikethrough": false,
      "backgroundColor": "transparent"
    }
  ]
}
```

Use in code:

```
# ! CRITICAL: This formula must match Bloomberg methodology
# ? TODO: Verify beta calculation with research team
# FINANCE: WACC = (E/V × Re) + (D/V × Rd × (1-T))
```

---

## Part 6: Advanced Search & Replace (10 minutes)

### Powerful Search

`Ctrl+Shift+F` - Search across all files

Search options: - `.*` - Use regex - `Aa` - Match case - `ab|` - Match whole word - `{}` - Use exclude/include patterns

### Regex Examples for Finance

Find all dollar amounts:

```
\$[\d,]+\.?\d*
```

Find all percentages:

```
\d+\.?\d*%
```

Find function definitions:

```
def calculate_\w+
```

Find TODO comments:

```
# TODO:.*
```

Multi-File Replace

Scenario: Change all `revenue` to `sales`

1. `Ctrl+Shift+H` (Search and Replace)
2. Search: `revenue`
3. Replace: `sales`
4. Important: Click files to preview changes
5. `Ctrl+Shift+1` - Replace in all files

Regex replace example:

Search: `year_(\d+)` Replace: `projection_year_$1`

Transforms: - `year_1` → `projection_year_1` - `year_2` → `projection_year_2`

---

## Part 7: Productivity Hacks (10 minutes)

### 1. Command Palette Mastery

`Ctrl+Shift+P` then type:

- `>reopen closed` - Reopen last closed editor
- `>reload` - Reload window
- `>clear` - Clear recently opened
- `>zen` - Zen mode (distraction-free)
- `>settings sync` - Sync settings across machines

### 2. Quick Actions

Select text → `Ctrl+.` - See available actions

Examples: - Extract to variable - Extract to function - Add import statement - Generate docstring

### 3. Emmet in Python

Type and press `Tab`:

```
# Type: df.
# Copilot suggests: head(), describe(), info()

# Type: plt.
# Copilot suggests: plot(), show(), figure()
```

### 4. Integrated Terminal Tips

Multiple terminals: - `Ctrl+Shift+`` - New terminal` -Ctrl+Shift+5‘ - Split terminal - Dropdown - Switch between terminals

Terminal shortcuts:

```
# History search
Ctrl+R

# Clear terminal
Ctrl+L (or type: clear)
```

```
# Kill process
Ctrl+C
```

5. Side-by-Side Editing

Compare files: 1. Open `file1.py` 2. Right-click `file2.py` in explorer 3. Select "Compare with `file1.py`"

Split editors: - `Ctrl+\` - Split right - `Ctrl+K Ctrl+\` - Split down - Drag tabs to split

Use case: Compare DCF v1 vs v2

---

## Part 8: Advanced Jupyter Tricks (10 minutes)

### Interactive Widgets

```python
from ipywidgets import interact, FloatSlider
import matplotlib.pyplot as plt

@interact(
    wacc=FloatSlider(min=0.05, max=0.15, step=0.01, value=0.10),
    growth=FloatSlider(min=0.01, max=0.05, step=0.005, value=0.025)
)
def dcf_sensitivity(wacc, growth):
    """Interactive DCF calculator"""
    # Calculate enterprise value
    fcf = 100
    terminal_value = fcf * (1 + growth) / (wacc - growth)

    print(f"WACC: {wacc*100:.1f}%")
    print(f"Growth: {growth*100:.1f}%")
    print(f"Terminal Value: ${terminal_value:.0f}M")

    # Visualize
    plt.figure(figsize=(8, 4))
    plt.bar(['FCF', 'Terminal Value'], [fcf, terminal_value])
    plt.title('DCF Components')
    plt.ylabel('Value ($M)')
    plt.show()
```

Move sliders, results update instantly! ☐

### Table of Contents

Add to first cell:

```markdown
# Financial Analysis Notebook

## Table of Contents
1. [Data Loading](#loading)
2. [Analysis](#analysis)
3. [Visualization](#viz)
4. [Conclusions](#conclusions)
```

Use markdown headers as anchors!

### Cell Execution Control

Magic commands:

```
%%time
# Code here - shows execution time

%%timeit
# Code here - runs multiple times, shows average

%%capture output
# Captures output to variable

%%writefile model.py
# Writes cell to file
```

### Variable Inspector Enhancement

```
# At top of notebook
%load_ext autoreload
%autoreload 2

# Now changes to .py files reload automatically!
```

---

## Part 9: Collaboration Features (10 minutes)

### Live Share (Real-Time Collaboration)

Install: `Ctrl+Shift+X` → "Live Share" extension

Start session: 1. `Ctrl+Shift+P` → "Live Share: Start Collaboration Session" 2. Share link with colleague 3. They can: - Edit same files - See your cursor - Use their own debugger - Share terminal

Perfect for: - Code reviews - Pair programming - Teaching models - Troubleshooting

### GitHub Integration

Push changes: 1. `Ctrl+Shift+G` - Source Control 2. Stage changes (+ icon) 3. Write commit message 4. Click ✓ Commit 5. Click "…" → Push

Pull Requests in VS Code: - Install "GitHub Pull Requests" extension - Review PRs without leaving VS Code - Comment on code - Approve/merge

### Code Review

Review someone's code: 1. Open file 2. Add comments: Right-click line → "Add Comment" 3. Discuss inline 4. Mark as resolved

---

## Part 10: Automation & Scripts (10 minutes)

### Automated Tasks

Create: `automate_workflow.py`

```python
"""
Automated Financial Analysis Workflow
Run daily to update all models
"""

import os
import pandas as pd
import yfinance as yf
from datetime import datetime
from dcf_model import DCFModel

def update_market_data():
    """Download latest stock data"""
    tickers = ['AAPL', 'MSFT', 'GOOGL', 'AMZN']

    for ticker in tickers:
        stock = yf.Ticker(ticker)
        df = stock.history(period='1mo')
        df.to_csv(f'data/{ticker}_latest.csv')

    print(f"  Updated data for {len(tickers)} stocks")

def run_all_models():
    """Execute all DCF models"""
    model_dir = 'models/'

    for file in os.listdir(model_dir):
        if file.endswith('_config.json'):
            # Load config and run model
            print(f"Running {file}...")
            # model code here

    print("  All models executed")

def export_reports():
    """Export to Excel reports"""
    timestamp = datetime.now().strftime('%Y%m%d')

    # Create report
    filename = f'reports/daily_report_{timestamp}.xlsx'
    # export code here

    print(f"  Report saved: {filename}")

if __name__ == "__main__":
    print("  Starting automated workflow...")
    update_market_data()
    run_all_models()
    export_reports()
    print("  Workflow complete!")
```

Schedule with Windows Task Scheduler: 1. Open Task Scheduler 2. Create Basic Task 3. Trigger: Daily at 7 AM 4. Action: Start program - Program: `C:\path\to\venv\Scripts\python.exe` - Arguments: `C:\path\to\automate_workflow.py` 5. Done! Runs automatically

Batch Processing

```python
def batch_export_models():
    """Export all models to Excel"""
    models = {
        'Apple': DCFModel('AAPL'),
        'Microsoft': DCFModel('MSFT'),
        'Google': DCFModel('GOOGL')
    }

    for name, model in models.items():
        # Configure and run
        model.calculate_dcf()
        model.export_to_excel(f'{name}_DCF.xlsx')

    print(f"  Exported {len(models)} models")
```

---

## ☐ Skills Checklist

After this tutorial, you are a VS Code power user:

- ☐ Master keyboard shortcuts
- ☐ Use multi-cursor editing
- ☐ Create custom code snippets
- ☐ Optimize workspace settings
- ☐ Debug Python code effectively
- ☐ Use essential extensions
- ☐ Advanced search & replace with regex
- ☐ Productivity hacks and tricks
- ☐ Collaborate with Live Share
- ☐ Automate workflows
- ☐ Professional development environment

---

## ☐ Ultimate Shortcuts Reference Card

Print this and keep by your desk!

```
VS CODE SHORTCUTS FOR FINANCE PROFESSIONALS


NAVIGATION
Ctrl+P          Quick open file
Ctrl+Shift+P    Command palette
Ctrl+G          Go to line
Ctrl+Shift+O    Go to symbol
Ctrl+Tab        Switch files
Ctrl+B          Toggle sidebar

EDITING
Ctrl+D          Select next occurrence
Ctrl+Shift+L    Select all occurrences
```

```
Alt+Click       Add cursor
Ctrl+Alt+↑/↓    Add cursor above/below
Alt+↑/↓         Move line
Shift+Alt+↑/↓   Copy line
Ctrl+/          Toggle comment
Ctrl+Shift+K    Delete line

SEARCH
Ctrl+F          Find
Ctrl+H          Replace
Ctrl+Shift+F    Find in files
Ctrl+Shift+H    Replace in files

CODE
Ctrl+Space      Trigger suggest
Ctrl+.          Quick fix
F12             Go to definition
Shift+F12       Find references
F2              Rename symbol

DEBUG
F9              Toggle breakpoint
F5              Start/continue
F10             Step over
F11             Step into
Shift+F5        Stop

TERMINAL
Ctrl+`          Toggle terminal
Ctrl+Shift+`    New terminal
Ctrl+C          Kill process

JUPYTER
Shift+Enter     Run cell, next
Ctrl+Enter      Run cell, stay
Alt+Enter       Run cell, insert below
DD              Delete cell (command mode)
A               Insert above
B               Insert below

GIT
Ctrl+Shift+G    Source control
Ctrl+K Ctrl+C   Stage changes
Ctrl+Enter      Commit
```

---

## ☐ Pro Tips Summary

1. Learn One Shortcut Per Day

   - Start with `Ctrl+P` and `Ctrl+D`
   - Add one new shortcut each day

- In 30 days, you're 10x faster!

2. Customize Your Environment
    - Create snippets for common patterns
    - Configure workspace settings
    - Install extensions you need
3. Use Multi-Cursor Liberally
    - Renaming variables
    - Updating formulas
    - Creating data structures
    - Batch operations
4. Leverage Copilot
    - Write comments, get code
    - Ask questions in chat
    - Review suggestions critically
5. Automate Repetitive Tasks
    - Tasks.json for common operations
    - Scripts for batch processing
    - Scheduled runs for updates

---

## ☐ What's Next?

You now have: - ☐ Power user keyboard shortcuts - ☐ Professional workflow optimization - ☐ Advanced editing capabilities - ☐ Debugging mastery - ☐ Automation skills - ☐ Collaboration tools

Next steps:

1. ☐ Practice shortcuts daily
2. ☐ Build custom snippet library
3. ☐ Set up automated workflows
4. ☐ Start Module 4: DCF Modeling
5. ☐ Apply skills to real projects

---

## ☐ Bonus: Personal Productivity System

Daily Workflow

Morning (5 min):

```
# Open workspace
code finance_workspace.code-workspace

# Pull latest from team
git pull
```

```
# Update market data
python update_data.py
```

During Work: - Use Copilot for new code - Multi-cursor for updates - Commit frequently - Debug with breakpoints

End of Day (5 min):

```
# Stage all changes
git add .

# Commit
git commit -m "Daily update: [what you did]"

# Push
git push

# Export reports
python export_reports.py
```

Weekly Review

- Review Git history
- Update documentation
- Refactor messy code
- Learn new shortcuts

Monthly Goals

- Build new template
- Learn advanced technique
- Contribute to open source
- Teach someone else

---

☐ You’re now a VS Code power user!

Continue to Module 4 to apply these skills to DCF modeling!

---

Estimated completion time: 90 minutes Difficulty: Intermediate-Advanced Completion: All Tutorials Done! Start Main Modules!