# Contents

# Tutorial 3: VS Code Data Analysis with Python

## □ What You'll Learn (90 minutes)

Master data analysis in VS Code using: - Pandas for financial data manipulation - Excel integration for corporate finance - Data visualization with Matplotlib & Plotly - Interactive Jupyter notebooks - Real financial statement analysis - Stock market data analysis

Prerequisites: - Completed Tutorials 1 & 2 - Python environment configured - Basic Python knowledge

---

## Part 1: Setting Up Data Analysis Environment (15 minutes)

### Install Required Libraries

Open VS Code terminal ('Ctrl+"):

```
# Activate virtual environment
venv\Scripts\activate

# Install data analysis stack
pip install pandas numpy matplotlib seaborn plotly yfinance openpyxl jupyter ipykernel

# Verify installations
python -c "import pandas; print(f'Pandas {pandas.__version__}')"
python -c "import matplotlib; print(f'Matplotlib {matplotlib.__version__}')"
python -c "import yfinance; print('yfinance installed!')"
```

### Install VS Code Extensions

`Ctrl+Shift+X` → Search and install: - □ Jupyter (Microsoft) - □ Jupyter Keymap (Microsoft) - □ Jupyter Notebook Renderers (Microsoft) - □ Data Wrangler (Microsoft) - Excel-like data viewer!

### Configure Jupyter in VS Code

1. Press `Ctrl+Shift+P`
2. Type: “Python: Select Interpreter”
3. Choose your venv interpreter: `.\venv\Scripts\python.exe`

---

## Part 2: Create Your First Jupyter Notebook (15 minutes)

### Create Notebook

1. `Ctrl+Shift+P` → “Create: New Jupyter Notebook”
2. Save as: `financial_analysis_intro.ipynb`

### Notebook Interface Tour

Key Components: - □ Code Cell: Write Python code - □ Markdown Cell: Write notes, explanations - ▸ Run Button: Execute cell - □ Add Cell: Insert new cell

### Your First Analysis

Cell 1 (Markdown - click `+Markdown`):

```markdown
# My First Financial Analysis

Learning data analysis in VS Code!

## Objectives
1. Load financial data
2. Calculate metrics
3. Visualize results
```

Cell 2 (Code):

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

print("  Libraries loaded!")
print(f"Pandas version: {pd.__version__}")
```

Run: `Shift+Enter`

Cell 3 (Code):

```python
# Create sample financial data
data = {
    'Year': [2020, 2021, 2022, 2023, 2024],
    'Revenue': [100, 120, 150, 180, 220],  # millions
    'EBITDA': [20, 28, 38, 50, 66],
    'Net_Income': [10, 15, 22, 32, 45]
}

df = pd.DataFrame(data)
print(df)
```

Run: `Shift+Enter`

Cell 4 (Code):

```
# Calculate growth rates
df['Revenue_Growth'] = df['Revenue'].pct_change() * 100
df['EBITDA_Margin'] = (df['EBITDA'] / df['Revenue']) * 100

df
```

Run: `Shift+Enter`

☐ You just did financial analysis in VS Code!

Jupyter Keyboard Shortcuts

| Shortcut | Action |
|---|---|
| `Shift+Enter` | Run cell, move to next |
| `Ctrl+Enter` | Run cell, stay in place |
| `Alt+Enter` | Run cell, insert below |
| `A` | Insert cell above |
| `B` | Insert cell below |
| `DD` | Delete cell |
| `M` | Convert to Markdown |
| `Y` | Convert to Code |
| `Ctrl+S` | Save notebook |

---

## Part 3: Working with Excel Data (20 minutes)

Create Sample Financial Statement

Create Excel file: `financial_statements.xlsx`

Using Excel: - Sheet1: “Income_Statement” - Add this data:

| Account | 2022 | 2023 | 2024 |
|---|---|---|---|
| Revenue | 500 | 600 | 750 |
| COGS | -300 | -345 | -420 |
| Gross Profit | 200 | 255 | 330 |
| SG&A | -80 | -90 | -105 |
| EBITDA | 120 | 165 | 225 |
| D&A | -20 | -25 | -30 |
| EBIT | 100 | 140 | 195 |
| Interest | -15 | -18 | -20 |
| EBT | 85 | 122 | 175 |
| Taxes | -21 | -30 | -44 |
| Net Income | 64 | 92 | 131 |

Save in your project folder.

Load and Analyze in VS Code

New notebook: `excel_analysis.ipynb`

Cell 1:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Load Excel file
df = pd.read_excel('financial_statements.xlsx',
                   sheet_name='Income_Statement',
                   index_col='Account')

print("  Financial Statements Loaded!")
df
```

Cell 2:

```
# Transpose for easier viewing
df_t = df.T
df_t.index.name = 'Year'
df_t
```

Cell 3:

```
# Calculate key metrics
metrics = pd.DataFrame()

metrics['Revenue_Growth'] = df_t['Revenue'].pct_change() * 100
metrics['Gross_Margin'] = (df_t['Gross Profit'] / df_t['Revenue']) * 100
metrics['EBITDA_Margin'] = (df_t['EBITDA'] / df_t['Revenue']) * 100
metrics['Net_Margin'] = (df_t['Net Income'] / df_t['Revenue']) * 100

metrics.round(2)
```

Cell 4:

```
# Visualize margin trends
plt.figure(figsize=(10, 6))

plt.plot(metrics.index, metrics['Gross_Margin'],
         marker='o', label='Gross Margin', linewidth=2)
plt.plot(metrics.index, metrics['EBITDA_Margin'],
         marker='s', label='EBITDA Margin', linewidth=2)
plt.plot(metrics.index, metrics['Net_Margin'],
         marker='^', label='Net Margin', linewidth=2)

plt.title('Profitability Margins Over Time', fontsize=14, fontweight='bold')
plt.xlabel('Year')
plt.ylabel('Margin (%)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()

plt.show()
```

Use Data Wrangler (Excel-Like Interface)

1. Click on `df` variable in notebook
2. Right-click → “Open in Data Wrangler”
3. See Excel-like interface!
4. Try:
    - Sorting columns
    - Filtering rows
    - Creating calculated columns
    - Export operations as code

This is HUGE for finance professionals! □

---

## Part 4: Real Stock Market Data Analysis (20 minutes)

### Download Stock Data

New notebook: `stock_analysis.ipynb`

Cell 1:

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Download Apple stock data (last 2 years)
ticker = 'AAPL'
stock = yf.Ticker(ticker)
df = stock.history(period='2y')

print(f"  Downloaded {len(df)} days of {ticker} data")
df.head()
```

Cell 2:

```
# Stock info
info = stock.info
print(f"Company: {info.get('longName', 'N/A')}")
print(f"Sector: {info.get('sector', 'N/A')}")
print(f"Market Cap: ${info.get('marketCap', 0) / 1e9:.2f}B")
print(f"P/E Ratio: {info.get('trailingPE', 'N/A')}")
print(f"Dividend Yield: {info.get('dividendYield', 0) * 100:.2f}%")
```

Cell 3:

```
# Calculate returns
df['Daily_Return'] = df['Close'].pct_change()
df['Cumulative_Return'] = (1 + df['Daily_Return']).cumprod() - 1

# Calculate moving averages
df['MA_50'] = df['Close'].rolling(window=50).mean()
df['MA_200'] = df['Close'].rolling(window=200).mean()

df[['Close', 'MA_50', 'MA_200', 'Cumulative_Return']].tail()
```

Cell 4:

```
# Visualize stock price with moving averages
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

# Price chart
ax1.plot(df.index, df['Close'], label='Close Price', linewidth=1.5)
ax1.plot(df.index, df['MA_50'], label='50-day MA', alpha=0.7)
ax1.plot(df.index, df['MA_200'], label='200-day MA', alpha=0.7)
ax1.set_title(f'{ticker} Stock Price with Moving Averages', fontsize=14, fontweight='bold')
ax1.set_ylabel('Price ($)')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Volume chart
ax2.bar(df.index, df['Volume'], alpha=0.5, color='blue')
ax2.set_title('Trading Volume', fontsize=12)
ax2.set_ylabel('Volume')
ax2.set_xlabel('Date')
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

Cell 5:

```
# Calculate volatility and risk metrics
daily_returns = df['Daily_Return'].dropna()

print("  Risk Metrics")
print(f"Average Daily Return: {daily_returns.mean() * 100:.3f}%")
print(f"Daily Volatility: {daily_returns.std() * 100:.3f}%")
print(f"Annualized Volatility: {daily_returns.std() * np.sqrt(252) * 100:.2f}%")
print(f"Sharpe Ratio (0% risk-free): {daily_returns.mean() / daily_returns.std() * np.sqrt(252):.2f}")
print(f"Max Daily Gain: {daily_returns.max() * 100:.2f}%")
print(f"Max Daily Loss: {daily_returns.min() * 100:.2f}%")
```

Cell 6:

```
# Distribution of returns
plt.figure(figsize=(10, 6))

plt.hist(daily_returns * 100, bins=50, alpha=0.7, edgecolor='black')
plt.axvline(daily_returns.mean() * 100, color='red',
            linestyle='--', linewidth=2, label=f'Mean: {daily_returns.mean()*100:.3f}%')
plt.title(f'{ticker} Daily Returns Distribution', fontsize=14, fontweight='bold')
plt.xlabel('Daily Return (%)')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True, alpha=0.3)

plt.show()
```

---

## Part 5: Multi-Stock Portfolio Analysis (15 minutes)

Cell 1:

```python
# Download multiple stocks
tickers = ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA']
data = yf.download(tickers, period='1y', group_by='ticker')

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

Cell 2:

```python
# Extract closing prices
closes = pd.DataFrame()
for ticker in tickers:
    closes[ticker] = data[ticker]['Close']

closes.head()
```

Cell 3:

```python
# Calculate normalized performance (starting at 100)
normalized = (closes / closes.iloc[0]) * 100

plt.figure(figsize=(12, 6))

for ticker in tickers:
    plt.plot(normalized.index, normalized[ticker], label=ticker, linewidth=2)

plt.title('Stock Performance Comparison (Normalized)', fontsize=14, fontweight='bold')
plt.ylabel('Normalized Price (Start = 100)')
plt.xlabel('Date')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()

plt.show()
```

Cell 4:

```python
# Calculate correlation matrix
returns = closes.pct_change().dropna()
correlation = returns.corr()

print("  Correlation Matrix")
correlation.round(2)
```

Cell 5:

```python
# Visualize correlation heatmap
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.heatmap(correlation, annot=True, fmt='.2f', cmap='coolwarm',
            square=True, linewidths=1, cbar_kws={'label': 'Correlation'})
plt.title('Stock Returns Correlation Matrix', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

Cell 6:

```python
# Portfolio performance metrics
metrics = pd.DataFrame({
    'Total Return (%)': ((closes.iloc[-1] / closes.iloc[0]) - 1) * 100,
    'Volatility (%)': returns.std() * np.sqrt(252) * 100,
    'Sharpe Ratio': (returns.mean() / returns.std()) * np.sqrt(252)
})

metrics.round(2).sort_values('Sharpe Ratio', ascending=False)
```

## Part 6: Interactive Visualizations with Plotly (10 minutes)

Cell 1:

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Get Apple data
aapl = yf.Ticker('AAPL')
df = aapl.history(period='6mo')
```

Cell 2:

```
# Create interactive candlestick chart
fig = go.Figure(data=[go.Candlestick(
    x=df.index,
    open=df['Open'],
    high=df['High'],
    low=df['Low'],
    close=df['Close'],
    name='AAPL'
)])

fig.update_layout(
    title='Apple Stock - Interactive Candlestick Chart',
    xaxis_title='Date',
    yaxis_title='Price ($)',
    height=600,
    xaxis_rangeslider_visible=False
)

fig.show()
```

Hover over chart, zoom, pan - it's interactive! □

Cell 3:

```
# Create dashboard with subplots
fig = make_subplots(
    rows=2, cols=1,
    subplot_titles=('Stock Price', 'Trading Volume'),
    row_heights=[0.7, 0.3],
    vertical_spacing=0.1
)

# Price line
fig.add_trace(
    go.Scatter(x=df.index, y=df['Close'], name='Close Price',
               line=dict(color='blue', width=2)),
    row=1, col=1
)

# Volume bars
fig.add_trace(
    go.Bar(x=df.index, y=df['Volume'], name='Volume',
           marker=dict(color='lightblue')),
    row=2, col=1
)

fig.update_layout(
    title_text='AAPL Stock Dashboard',
    height=700,
    showlegend=True
)

fig.show()
```

Cell 4:

```
# Financial statement visualization
# Load our Excel data
df_fin = pd.read_excel('financial_statements.xlsx',
                       sheet_name='Income_Statement',
                       index_col='Account')

revenue_data = df_fin.loc['Revenue']
ebitda_data = df_fin.loc['EBITDA']
net_income_data = df_fin.loc['Net Income']

fig = go.Figure()

fig.add_trace(go.Bar(
    x=revenue_data.index,
    y=revenue_data.values,
    name='Revenue',
    marker_color='lightblue'
))

fig.add_trace(go.Bar(
    x=ebitda_data.index,
    y=ebitda_data.values,
    name='EBITDA',
    marker_color='orange'
))

fig.add_trace(go.Bar(
    x=net_income_data.index,
    y=net_income_data.values,
    name='Net Income',
    marker_color='green'
))

fig.update_layout(
    title='Income Statement Trends',
    xaxis_title='Year',
    yaxis_title='Amount ($M)',
    barmode='group',
    height=500
)

fig.show()
```

---

## Part 7: Real Financial Analysis Project (15 minutes)

Project: Company Valuation Comparison

New notebook: `company_comparison.ipynb`

Cell 1:

```
import yfinance as yf
import pandas as pd
import plotly.graph_objects as go

# Compare tech giants
tickers = ['AAPL', 'MSFT', 'GOOGL', 'META', 'AMZN']

# Get company info
companies = []
for ticker in tickers:
    stock = yf.Ticker(ticker)
    info = stock.info
    companies.append({
        'Ticker': ticker,
        'Name': info.get('longName', ticker),
        'Market Cap ($B)': info.get('marketCap', 0) / 1e9,
        'P/E Ratio': info.get('trailingPE', None),
        'Revenue ($B)': info.get('totalRevenue', 0) / 1e9,
        'Profit Margin (%)': info.get('profitMargins', 0) * 100,
        'ROE (%)': info.get('returnOnEquity', 0) * 100,
        'Debt/Equity': info.get('debtToEquity', 0) / 100
    })

df_comp = pd.DataFrame(companies)
df_comp
```

Cell 2:

```
# Calculate EV/Revenue multiples
df_comp['EV/Revenue'] = df_comp['Market Cap ($B)'] / df_comp['Revenue ($B)']

df_comp[['Ticker', 'Name', 'Market Cap ($B)', 'P/E Ratio',
         'EV/Revenue', 'Profit Margin (%)']].round(2)
```

Cell 3:

```
# Visualize valuation multiples
fig = go.Figure()
```

```
fig.add_trace(go.Bar(
    x=df_comp['Ticker'],
    y=df_comp['P/E Ratio'],
    name='P/E Ratio',
    marker_color='lightblue'
))

fig.update_layout(
    title='P/E Ratio Comparison - Big Tech',
    xaxis_title='Company',
    yaxis_title='P/E Ratio',
    height=500
)

fig.show()
```

Cell 4:

```
# Profitability vs Valuation scatter
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=df_comp['Profit Margin (%)'],
    y=df_comp['P/E Ratio'],
    mode='markers+text',
    marker=dict(
        size=df_comp['Market Cap ($B)'] / 50,  # Size by market cap
        color=df_comp['ROE (%)'],  # Color by ROE
        colorscale='Viridis',
        showscale=True,
        colorbar=dict(title='ROE (%)')
    ),
    text=df_comp['Ticker'],
    textposition='top center'
))

fig.update_layout(
    title='Profitability vs Valuation (Bubble Size = Market Cap)',
    xaxis_title='Profit Margin (%)',
    yaxis_title='P/E Ratio',
    height=600
)

fig.show()
```

Cell 5:

```
# Summary insights
print("  INVESTMENT INSIGHTS")
print("=" * 60)

# Highest PE
highest_pe = df_comp.loc[df_comp['P/E Ratio'].idxmax()]
print(f"\n  Highest P/E: {highest_pe['Ticker']} ({highest_pe['P/E Ratio']:.1f}x)")

# Lowest PE
```

```
lowest_pe = df_comp.loc[df_comp['P/E Ratio'].idxmin()]
print(f"  Lowest P/E: {lowest_pe['Ticker']} ({lowest_pe['P/E Ratio']:.1f}x)")

# Best margins
best_margin = df_comp.loc[df_comp['Profit Margin (%)'].idxmax()]
print(f"\n  Best Profit Margin: {best_margin['Ticker']} ({best_margin['Profit Margin (%)']:.1f}%)")

# Highest ROE
best_roe = df_comp.loc[df_comp['ROE (%)'].idxmax()]
print(f"  Highest ROE: {best_roe['Ticker']} ({best_roe['ROE (%)']:.1f}%)")

# Largest company
largest = df_comp.loc[df_comp['Market Cap ($B)'].idxmax()]
print(f"\n  Largest Market Cap: {largest['Name']} (${largest['Market Cap ($B)']:.1f}B)")

print("\n" + "=" * 60)
```

---

## Part 8: Export and Share Results (5 minutes)

### Export to Excel

Cell:

```
# Create comprehensive report
with pd.ExcelWriter('tech_company_analysis.xlsx', engine='openpyxl') as writer:
    # Company comparison
    df_comp.to_excel(writer, sheet_name='Company_Comparison', index=False)

    # Stock returns (if you have the data)
    if 'returns' in locals():
        returns_summary = returns.describe().T
        returns_summary.to_excel(writer, sheet_name='Returns_Statistics')

    # Correlation matrix
    if 'correlation' in locals():
        correlation.to_excel(writer, sheet_name='Correlation_Matrix')

print("  Report exported to: tech_company_analysis.xlsx")
```

### Export Charts

Cell:

```
# Save interactive chart as HTML
fig.write_html('company_comparison_chart.html')
print("  Interactive chart saved: company_comparison_chart.html")

# Save static image (requires kaleido)
# pip install kaleido
# fig.write_image('company_comparison.png')
```

### Export Notebook as HTML

1. Click “…” (more actions) in notebook toolbar

2. Select “Export”
3. Choose “HTML”
4. Save as: `financial_analysis_report.html`

Share with colleagues who don’t use Python! □

---

## □ Skills Checklist

After this tutorial, you can:

- □ Create and use Jupyter notebooks in VS Code
- □ Load data from Excel files
- □ Manipulate financial data with Pandas
- □ Download real-time stock data
- □ Calculate financial metrics and returns
- □ Create static visualizations (Matplotlib)
- □ Create interactive charts (Plotly)
- □ Analyze multiple stocks
- □ Export results to Excel
- □ Share analysis reports

---

## □ Data Analysis Cheat Sheet

Pandas Essentials

```
# Loading Data
df = pd.read_excel('file.xlsx')
df = pd.read_csv('file.csv')

# Viewing Data
df.head()          # First 5 rows
df.tail()          # Last 5 rows
df.info()          # Data types and memory
df.describe()      # Statistics

# Selecting Data
df['Column']       # Single column
df[['Col1', 'Col2']]  # Multiple columns
df.loc[0]          # Row by index
df.loc[0:5]        # Rows 0-5
df.iloc[0]         # Row by position

# Calculations
df['New'] = df['A'] / df['B']  # New column
df['Growth'] = df['Rev'].pct_change()
df['MA'] = df['Price'].rolling(20).mean()

# Filtering
df[df['Revenue'] > 100]
df[df['Year'].isin([2023, 2024])]

# Aggregations
```

```
df.sum()
df.mean()
df.groupby('Year').sum()

# Sorting
df.sort_values('Revenue', ascending=False)
```

### Matplotlib Basics

```
import matplotlib.pyplot as plt

# Line chart
plt.plot(x, y, label='Label')
plt.title('Title')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.legend()
plt.grid()
plt.show()

# Bar chart
plt.bar(categories, values)

# Scatter plot
plt.scatter(x, y)

# Multiple subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))
ax1.plot(x, y1)
ax2.plot(x, y2)
```

---

## □ Pro Tips

### 1. Use Variable Inspector

- Bottom of notebook shows all variables
- Click to inspect values
- Right-click → "Open in Data Wrangler"

### 2. Clear Output

- Cell output cluttering notebook?
- Right-click cell → "Clear Outputs"
- Or: Clear All Outputs

### 3. Cell Execution Order

- Numbers in brackets show execution order: `[1]`, `[2]`, `[3]`
- Important for debugging!
- Restart kernel if confused: Top menu → Restart

4. Keyboard Shortcuts

- Learn these to work 10x faster!
- `Ctrl+Shift+P` → "Notebook: Show All Shortcuts"

5. Comments and Documentation

```python
# Use comments for complex calculations
df['ROIC'] = (
    df['NOPAT'] /
    (df['Total_Debt'] + df['Shareholders_Equity'])
)  # Return on Invested Capital
```

---

## □ Real-World Applications

### Investment Banking

- □ Comparable company analysis
- □ Precedent transaction analysis
- □ DCF sensitivity tables
- □ LBO returns analysis

### Private Equity

- □ Portfolio company monitoring
- □ Entry/exit multiple analysis
- □ IRR calculations
- □ Fund performance tracking

### Corporate Finance

- □ Budget vs actual analysis
- □ Financial statement modeling
- □ KPI dashboards
- □ Scenario analysis

### Equity Research

- □ Stock screening
- □ Valuation models
- □ Peer comparison
- □ Trend analysis

---

## □ Bonus: Advanced Techniques

1. Automated Reports

```python
# Run daily, export to Excel, email to team
import yfinance as yf
from datetime import datetime
```

```
def daily_portfolio_report(tickers):
    data = yf.download(tickers, period='1d')
    # ... analysis ...

    filename = f'portfolio_report_{datetime.now().strftime("%Y%m%d")}.xlsx'
    # ... export ...
    return filename

# Schedule this with Windows Task Scheduler!
```

2. Custom Functions

```
def calculate_financial_ratios(df):
    """Calculate all key ratios"""
    ratios = pd.DataFrame()
    ratios['Current_Ratio'] = df['Current_Assets'] / df['Current_Liabilities']
    ratios['Quick_Ratio'] = (df['Current_Assets'] - df['Inventory']) / df['Current_Liabilities']
    ratios['Debt_to_Equity'] = df['Total_Debt'] / df['Shareholders_Equity']
    ratios['ROE'] = df['Net_Income'] / df['Shareholders_Equity']
    ratios['ROA'] = df['Net_Income'] / df['Total_Assets']
    return ratios

# Use in any notebook!
ratios = calculate_financial_ratios(balance_sheet_df)
```

3. Templates

Save your best notebooks as templates: - Company analysis template - Stock comparison template - Financial statement template

Copy and reuse! □

---

## □ What's Next?

You now have: - □ Data analysis superpowers - □ Excel integration - □ Stock market data access - □ Visualization skills - □ Professional reporting capabilities

Next steps:

1. □ Practice with real data
2. □ Build your own analysis templates
3. □ Continue to: `Tutorials/04_Building_DCF_with_VS_Code.md`
4. □ Create portfolio tracking dashboard

---

□ You're now a VS Code data analyst!

Next: `Tutorials/04_Building_DCF_with_VS_Code.md` - Build DCF models step-by-step

---

Estimated completion time: 90 minutes Difficulty: Intermediate Next: Building DCF Models