

Python Pandas for Economics & Finance

Data Analysis Made Easy



For Undergraduate Economics & Finance Students

What is Pandas?

Pandas is a powerful Python library for data manipulation and analysis, particularly well-suited for:



Handling structured data like time series, tabular data, and statistical datasets



Analyzing financial markets, economic indicators, and business metrics



Transforming data between different formats (CSV, Excel, SQL, JSON)



Performing complex calculations on large datasets efficiently



Exploring and visualizing data patterns and relationships



Core Data Structures

Series

One-dimensional labeled array capable of holding any data type

```
# Stock closing prices (USD)
import pandas as pd

stock_prices = pd.Series([142.56, 143.21, 145.03,
                          144.87, 146.12],
                        index=['2023-01-01', '2023-01-02', '2023-01-03',
                              '2023-01-04', '2023-01-05'])

# Access by label
price = stock_prices['2023-01-03'] # 145.03
```

DataFrame

Two-dimensional labeled data structure with columns of potentially different types

```
# Economic indicators by country
import pandas as pd

data = {
    'Country': ['USA', 'China', 'Japan', 'Germany'],
    'GDP_Trillion': [23.0, 17.7, 4.9, 4.2],
    'Growth_Rate': [2.1, 5.2, 1.0, 1.5]
}

df = pd.DataFrame(data)
# Access column: df['GDP_Trillion']
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2018-01-01 07:21:51.000002
1	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16	2018-01-01 10:00:47.000004
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2018-01-01 13:24:57.999997
3	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2018-01-01 15:05:21.999995
4	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26	2018-01-01 23:26:55.000003
...
1502	424914	White-Trantow	B1-69924	37	42.77	1582.49	2018-11-27 14:29:02.000003
1503	424914	White-Trantow	S1-47412	16	65.58	1049.28	2018-12-19 15:15:41.000000
1504	424914	White-Trantow	B1-86481	75	28.89	2166.75	2018-12-29 13:03:54.000000
1505	424914	White-Trantow	S1-82801	20	95.75	1915.00	2018-12-22 03:31:35.999996
1506	424914	White-Trantow	S2-83881	100	88.19	8819.00	2018-12-16 00:46:26.000003

1507 rows × 7 columns

Loading Financial Data

From CSV Files

```
import pandas as pd

# Load stock price history
df = pd.read_csv('stock_prices.csv',
                 parse_dates=['Date'],
                 index_col='Date')
```

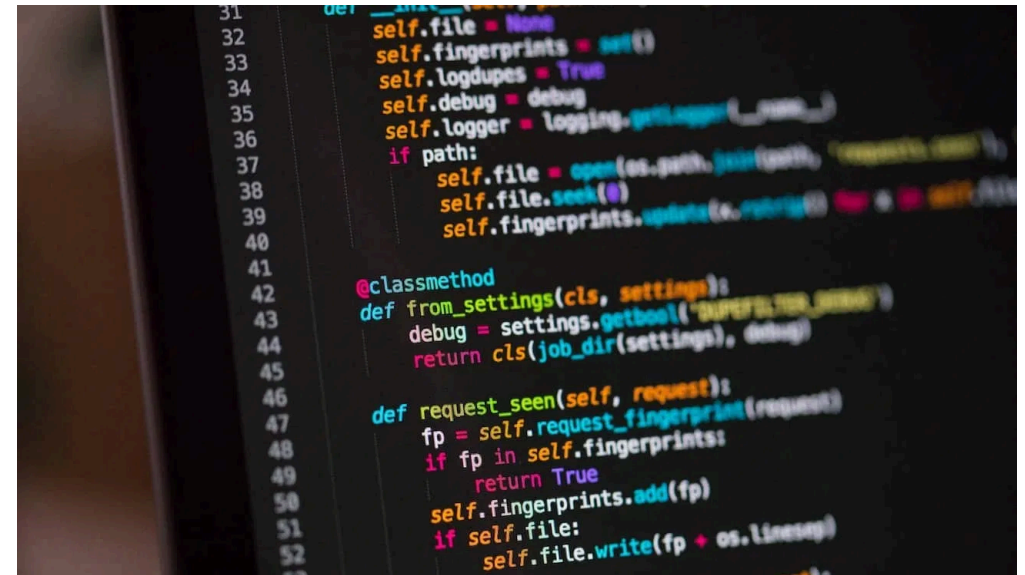
From Financial APIs

```
# Using pandas-datareader with Yahoo Finance
import pandas_datareader as pdr

aapl = pdr.get_data_yahoo('AAPL',
                          start='2022-01-01',
                          end='2022-12-31')
```

Best Practices

- ✓ Always specify data types and date formats
- ✓ Handle missing values during import
- ✓ Set appropriate index for time series data



Data Exploration & Inspection

`.head()` and `.tail()`

View first or last n rows of the DataFrame

```
df.head(5) # First 5 rows
```

`.info()`

Summary of DataFrame including data types and missing values

```
df.info()
```

`.describe()`

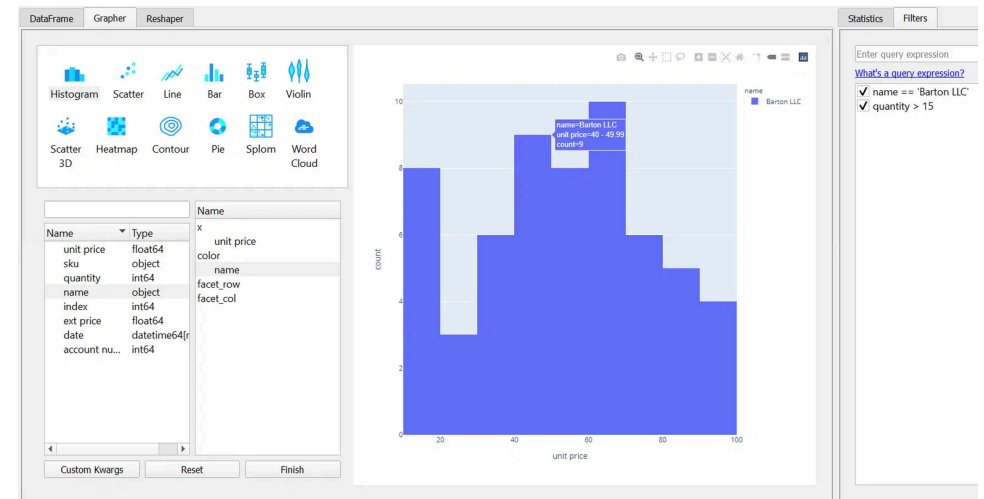
Statistical summary of numerical columns

```
df.describe()
```

`.isnull().sum()`

Count missing values in each column

```
df.isnull().sum()
```



Visual representation of pandas data exploration methods

Pro Tip: Always explore your data before analysis

Data Cleaning & Preparation

Handling Missing Data

🔍 Identify missing values with `df.isna()` or `df.isnull()`

🗑️ Remove rows with `df.dropna()` or columns with `df.dropna(axis=1)`

🔄 Fill missing values with `df.fillna(value)` or `df.fillna(method='ffill')`

Data Type Conversions

↔️ Convert types with `df.astype()` (e.g., string to numeric)

💵 Convert currency strings to numeric:
`pd.to_numeric(df['Price'].str.replace('$', ''))`

Date/Time Handling

📅 Convert to datetime: `pd.to_datetime(df['Date'])`

🕒 Set date index: `df.set_index('Date', inplace=True)`

📅 Resample time series: `df.resample('M').mean()` (monthly average)

```
In [4]: ebitda_margin = 0.14
        depr_percent = 0.032

        ebitda = sales * ebitda_margin
        depreciation = sales * depr_percent
        ebit = ebitda - depreciation
```

```
In [5]: nwc_percent = 0.24

        nwc = sales * nwc_percent
        change_in_nwc = nwc.shift(1) - nwc
        capex_percent = depr_percent
        capex = -(sales * capex_percent)

        tax_rate = 0.25

        tax_payment = -ebit * tax_rate
        tax_payment = tax_payment.apply(lambda x: min(x, 0))
        free_cash_flow = ebit + depreciation + tax_payment + capex + change_in_nwc

        free_cash_flow
```

```
Out[5]: 2018A      NaN
        2019B    2.018100
        2020P    2.219910
        2021P    2.441901
        2022P    2.686091
        2023P    2.954700
        dtype: float64
```

Basic Operations & Calculations

Common Financial Calculations

✂ Calculating Returns

```
# Daily returns calculation
df['daily_returns'] = df['close'].pct_change()

# Cumulative returns
df['cumulative_returns'] = (1 +
df['daily_returns']).cumprod() - 1
```

📈 Moving Averages

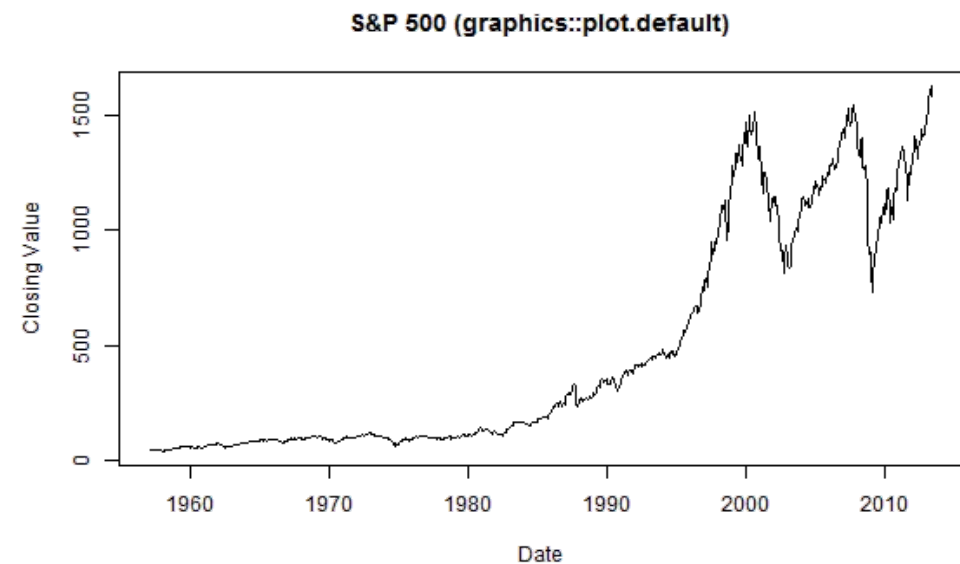
```
# 20-day moving average
df['MA20'] = df['close'].rolling(window=20).mean()

# Exponential moving average
df['EMA12'] = df['close'].ewm(span=12).mean()
```

Grouping & Aggregation

```
# Group by sector and calculate average P/E ratio
sector_pe = df.groupby('sector')['pe_ratio'].mean()

# Multiple aggregations
result = df.groupby('country').agg({
    'gdp': ['mean', 'std'],
    'inflation': ['min', 'max']
})
```



📊 Statistical Methods

```
# Correlation between assets
correlation = df[['AAPL', 'MSFT', 'GOOGL']].corr()

# Volatility (standard deviation of returns)
volatility = df['daily_returns'].std() * (252 ** 0.5) #
Annualized
```

Data Visualization with Pandas

Built-in Plotting Capabilities

Simple API: `df.plot()`

Multiple plot types

Financial plots

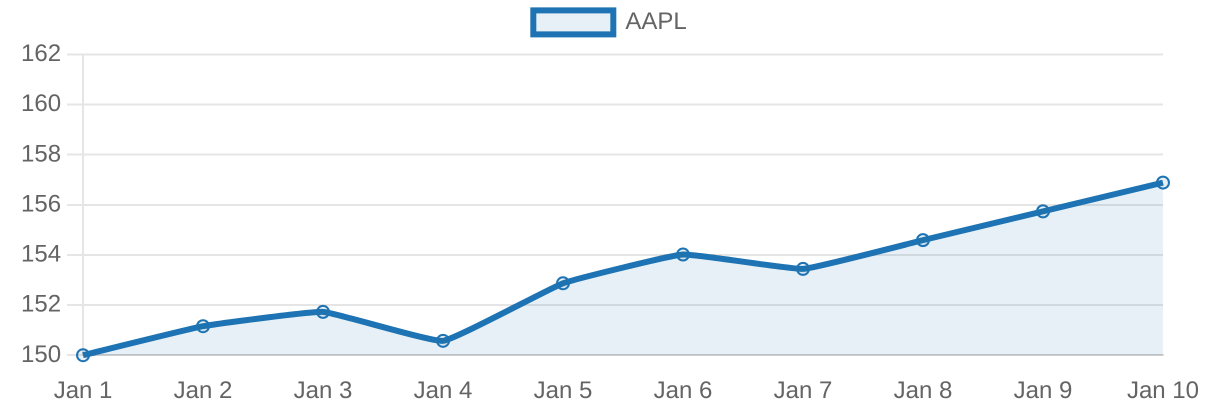
Customizable styles

Integration with Matplotlib

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

# Create sample stock data
data = {
    'Date': pd.date_range('2023-01-01', periods=10),
    'AAPL': [150, 152, 153, 151, 155, 157, 156, 158, 160, 162]
}
```

Stock Prices (USD)



Practical Example: Stock Analysis

Complete Workflow Example

1 Import libraries and load data

```
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader as pdr

# Get stock data
ticker = 'AAPL'
start_date = '2022-01-01'
end_date = '2022-12-31'
stock_data = pdr.get_data_yahoo(ticker, start_date, end_date)
```

2 Calculate key financial metrics

```
# Calculate daily returns
stock_data['Returns'] = stock_data['Adj Close'].pct_change()

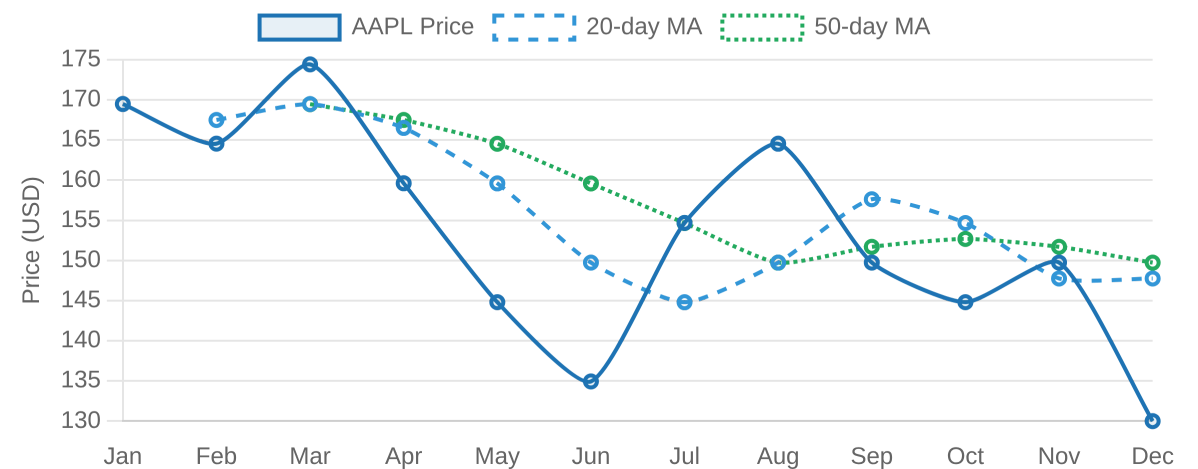
# Calculate moving averages
stock_data['MA20'] = stock_data['Adj Close'].rolling(window=20).mean()
stock_data['MA50'] = stock_data['Adj Close'].rolling(window=50).mean()

# Calculate volatility (20-day)
stock_data['Volatility'] =
stock_data['Returns'].rolling(window=20).std() * (252**0.5)
```

3 Create visualizations

```
# Plot price and moving averages
plt.figure(figsize=(10, 6))
plt.plot(stock_data['Adj Close'], label='Price')
plt.plot(stock_data['MA20'], label='20-day MA')
plt.plot(stock_data['MA50'], label='50-day MA')
plt.title(f'{ticker} Stock Price')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(True)
plt.show()
```

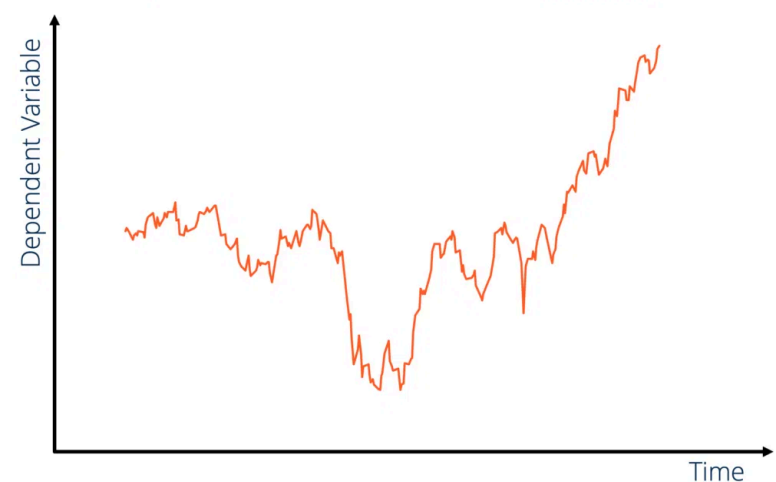
AAPL Stock Price with Moving Averages (2022)



Key Insights from Analysis





- Identify trends using moving averages
- Measure volatility for risk assessment
- Calculate returns for performance evaluation
- Detect trading signals (MA crossovers)

Time-Series Analysis



Next Steps & Resources

Advanced Pandas Features

-  MultilIndex for hierarchical data structures
-  Pivot tables and cross-tabulations
-  Merging, joining, and concatenating datasets
-  Advanced time series functionality

Integration with Other Libraries

NumPy - Mathematical operations

Matplotlib/Seaborn - Advanced visualization

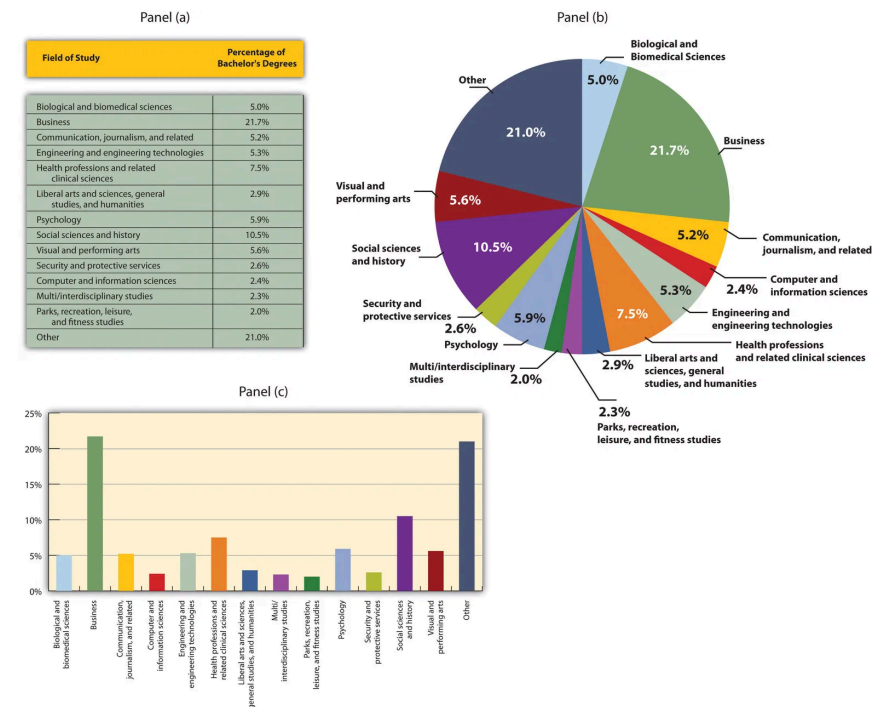
Scikit-learn - Machine learning

Statsmodels - Statistical modeling

PyMC3 - Bayesian analysis

Learning Resources

-  Official documentation: pandas.pydata.org
-  Practice datasets: [Kaggle.com/datasets](https://kaggle.com/datasets) (financial markets, economic indicators)
-  Online courses: DataCamp, Coursera "Python for Financial Analysis"



Start Your Data Analysis Journey Today!

Apply pandas to real economic and financial problems

Self-Check: Test Your Pandas Understanding

Practice Exercises

Exercise 1: Data Loading

Load 'stock_data.csv' with Date as index and parse dates properly.

💡 Hint: Use `parse_dates` and `index_col` parameters

Exercise 2: Financial Calculations CHALLENGE

For AAPL stock, calculate daily returns and 20-day moving average.

Exercise 3: Data Analysis

Filter data for volume > 50M shares and find average closing price by symbol.

Solutions

Solution 1:

```
import pandas as pd

df = pd.read_csv('stock_data.csv',
                 parse_dates=['Date'],
                 index_col='Date')
```

Solution 2:

```
# Filter for AAPL and calculate metrics
aapl = df[df['Symbol'] == 'AAPL'].copy()

# Daily returns
aapl['Returns'] = aapl['Close'].pct_change()
```

Solution 3:

```
# Filter and group data
high_vol = df[df['Volume'] > 50000000]

avg_prices = high_vol.groupby('Symbol')['Close'].mean()
print(avg_prices)
```