

Financial Data Structures and the Python Ecosystem

From Tick to OHLCV · Data Cleaning · Feature Engineering · Multi-Asset Analysis

Instructor

Assoc. Prof. Dr. Onur POLAT

Course

SEN605 — Elective

Prerequisite

Week 1 completed

What will we cover today?

01 Introduction to Financial Data Structures

02 Time Series Operations with Pandas

03 Data Cleaning Pipeline

04 Feature Engineering

05 Multi-Asset Data and Correlation

06 Lab: BIST-30 Analysis Pipeline

07 Complete Code: Return Matrix

08 Summary and Assignment



Complex System

Financial markets are nonlinear, feedback-driven, adaptive systems



Data Pyramid

Tick → Bar → Fundamental →
Alternative data layers — we deepen today



Python Setup

yfinance, pandas, numpy
environment should be ready
— heavy use today



Log Return

$r_t = \ln(P_t/P_{t-1})$ formula — we will use it as a feature today



Today's Goal

- Transform financial data from raw ticks to a model-ready feature matrix
- Build a complete data cleaning and feature engineering pipeline
- Download BIST-30 stocks and perform multi-asset correlation analysis

4

Financial Data Hierarchy

From raw data to processed features — data transformation chain

1

Raw Tick Data

Every trade, quote, and cancellation — microsecond resolution, full market depth

2

Bar Data (OHLCV)

1m / 5m / daily bars — time, trade, or volume-based aggregation

3

Cleaned Data

Missing value imputation (ffill), outlier removal, corporate action adjustment

4

Features (X)

Return series, volatility, momentum, technical indicators — model input

5

Model Input (X,Y)

Normalized, scaled, train/test split matrix — ready for machine learning

SECTION 1

Financial Data Structures

Tick data · OHLCV bar types · Bar construction methods

What Is Tick Data?

- A record of every executed trade or quote change
- Timestamp (microsecond), price, volume, aggressor (buyer/seller initiated?)
- Major exchanges can generate billions of rows per day
- Order book snapshots are also part of tick data
- Critical for high-frequency traders (HFT)
- Generally expensive and access-restricted for individual researchers

Sample Tick Data (AKBNK)

Time	Price (₺)	Volume	Direction
09:31:45.231	34.15	500	Buy
09:31:45.310	34.16	1200	Buy
09:31:45.891	34.14	300	Sell
09:31:46.012	34.15	800	Buy
09:31:46.240	34.13	2000	Sell
09:31:46.410	34.12	1500	Sell
09:31:47.005	34.14	600	Buy
09:31:47.312	34.16	400	Buy

O**Open**

First trade price of the period

H**High**

Highest price within the period

L**Low**

Lowest price within the period

C**Close**

Last trade price of the period — most commonly used

V**Volume (Volume)**

Total lots/shares traded during the period

Sample Daily Bar (XU100.IS — 2024-01-15)

Open	High	Low	Close	Volume
9.452,3	9.618,7	9.431,2	9.595,4	1.234.567.890

Accessing OHLCV data

```
# Accessing OHLCV with pandas
print(df.head())
#           Open      High      Low      Close
Volume
# 2024-01-15  9452.3  9618.7  9431.2  9595.4
1234567890
```



Time Bars

Definition:

Bars are formed at fixed time intervals (1m, 5m, 1 day)

Advantage:

Industry standard, widely used, easy comparison

Disadvantage:

Uninformative during low activity, insufficient during high activity

Usage:

Created with `resample('1D')` or `resample('5T')`



Trade Bars

Definition:

Bar closes when a specified number of trades occur (e.g., every 1000 trades)

Advantage:

Reflects market activity; more bars during high activity

Disadvantage:

Difficult to compare day vs. night; does not contain equal volume

Usage:

One bar every N ticks — `tick_bars(df, n=1000)`



Volume Bars

Definition:

Bar closes when a specified volume is traded (e.g., every 10M lots)

Advantage:

More balanced in information content; reduces HFT impact

Disadvantage:

High computational cost; requires tick data

Usage:

Proposed by Lopez de Prado — superior performance in ML models

SECTION 2

Pandas

DatetimeIndex · Resample · Rolling · Merge

SEN605 · Week 2

DatetimeIndex Features

- Indexing and slicing by timestamp: `df['2023']`
- Business day filtering: only workdays with `freq='B'`
- Timezone support: `tz_localize('Europe/Istanbul')`
- Shifting: `df.shift(1)` for previous day's value
- Differencing: `df.diff(1)` for daily change
- Date arithmetic: `BusinessDay`, `MonthEnd`, `YearEnd` offsets

DatetimeIndex operations

```
import pandas as pd
import yfinance as yf

df = yf.download("XU100.IS",
                 start="2020-01-01",
                 end="2024-12-31",
                 auto_adjust=True)

# Year-based slicing
df_2023 = df['2023']
df_q1    = df['2023-01':'2023-03']

# Business day frequency reindexing
idx_is = pd.date_range(
    start='2020-01-01',
    end='2024-12-31',
    freq='B')          # 'B' = business day
df_is = df.reindex(idx_is)

# Shift: previous close
df['prev_close'] = df['Close'].shift(1)

# Accessing date components
```

Why Is Resampling Important?

- Bringing data at different frequencies to a common time frame
- Reducing noise: daily → weekly bars have less noise
- Choosing modeling frequency: daily for LSTM; monthly for macro models
- Each OHLCV column aggregates differently: H=max, L=min, V=sum

Common Frequency Codes

'T' / 'min'	Minute (e.g., '5T' = 5 minutes)
'H'	Hourly
'B'	Business day
'W'	Weekly (Friday close)
'ME'	Month end
'QE'	Quarter end

Resample — OHLCV conversion

```
# OHLCV aggregation
agg_func = {
    'Open' : 'first',    # First price
    'High' : 'max',      # Highest
    'Low'  : 'min',      # Lowest
    'Close': 'last',     # Last price
    'Volume': 'sum'      # Total volume
}

# Daily → Weekly
weekly = df.resample('W').agg(agg_func)
weekly.dropna(inplace=True)

# Daily → Monthly
monthly = df.resample('ME').agg(agg_func)
monthly.dropna(inplace=True)

# Daily → 5-day
five_day = df.resample('5B').agg(agg_func)

print(f"Daily: {len(df)} bars")
print(f"Weekly: {len(weekly)} bars")
print(f"Monthly: {len(monthly)} bars")
```

rolling() — Fixed Window

A fixed-size window covering the last N observations. Advances one step with each new observation.

Usage: Moving average, rolling volatility, rolling correlation

expanding() — Expanding Window

Covers all data from the start of the series and expands with each step. Min periods can be specified.

Usage: Cumulative mean, cumulative max, cumulative standard deviation

ewm() — Exponentially Weighted Window

Gives more weight to recent observations. Controlled by span or halflife parameter.

Usage: EMA (exponential moving average), MACD calculation

Window operation examples

```
# Calculate log return
df['log_ret'] = np.log(
    df['Close'] / df['Close'].shift(1))

# --- rolling() examples ---
# 20-day moving average
df['ma20'] = df['Close'].rolling(20).mean()

# 21-day annualized volatility
df['vol21'] = (df['log_ret']
    .rolling(21).std() * np.sqrt(252))

# 63-day return correlation
# (between return and volatility)
df['ret_vol_corr'] = df['log_ret'].rolling(
    63).corr(df['vol21'])

# --- expanding() ---
df['cum_max'] = df['Close'].expanding().max()
df['current_drawdown'] = (
    df['Close'] / df['cum_max'] - 1)

# --- ewm() ---
```

SECTION 3

Data Cleaning Pipeline

Missing data · Outliers · Corporate actions · Volume filter

Sources of Missing Data

- Holidays / market closed days (national, religious holidays)
- Exchange technical issues (trading halts, system outages)
- Missing history after IPO or dilution events
- Data provider errors — especially in smaller exchanges
- Ticker changes after mergers/acquisitions

Imputation Strategies

- `ffill()` — forward fill: carry the last valid price (suitable for holidays)
- `bfill()` — backward fill: carry the next price (limited use)
- `interpolate()` — linear interpolation (for short gaps)
- `dropna()` — drop: remove all missing values before modeling
- ⚠ Never fill with 0 — price cannot be 0, returns become $\pm\infty$

Missing data detection and imputation

```
# Missing data analysis
print("Missing value count:")
print(df.isnull().sum())
print()
print("Missing percentage:")
print((df.isnull().sum()/len(df)*100).round(2))

# Detect holidays
biz_days = pd.date_range(
    df.index.min(), df.index.max(), freq='B')
missing_days = biz_days.difference(df.index)
print(f"Missing business days: {len(missing_days)} days")

# Forward fill (safest method)
df_clean = df.copy()
df_clean = df_clean.reindex(biz_days)
df_clean = df_clean.ffill()

# Remove zero-volume days
df_clean = df_clean[df_clean['Volume'] > 0]
print(f"Remaining rows: {len(df_clean)}")
```

Z-Skoru Directionemi

Flag values beyond ± 3 standard deviations from the mean.
Common for return series.

```
z = (df['log_ret'] - mean) / std
df[np.abs(z) > 3]
```

IQR (Interquartile Range)

Flag values outside $Q1 - 1.5 \times IQR$ and $Q3 + 1.5 \times IQR$. No distribution assumption required.

```
Q1, Q3 = df.quantile([0.25, 0.75])
IQR = Q3 - Q1
```

Rolling Z-Score

Windowed z-score: adapts to time-varying means. Preferred in finance.

```
rol_mean = df.rolling(63).mean()
rol_std = df.rolling(63).std()
```

Outlier detection and handling

```
import numpy as np

# Calculate log return
df['log_ret'] = np.log(
    df['Close'] / df['Close'].shift(1))

# 1) Standard Z-Score
mean = df['log_ret'].mean()
std = df['log_ret'].std()
df['z_score'] = (df['log_ret'] - mean) / std
outliers_z = df[np.abs(df['z_score']) > 4]
print(f"Z>4 outliers: {len(outliers_z)} days")

# 2) Rolling Z-Score (63-day window)
rol_m = df['log_ret'].rolling(63).mean()
rol_s = df['log_ret'].rolling(63).std()
df['roll_z'] = (df['log_ret'] - rol_m) / rol_s

# 3) Winsorize (clip outliers, don't drop)
pct_01 = df['log_ret'].quantile(0.01)
pct_99 = df['log_ret'].quantile(0.99)
df['ret_win'] = df['log_ret'].clip(
    lower=pct_01, upper=pct_99)
```

Stock Split

⚠️ **Problem:** In a 2:1 split, the stock price halves. Unadjusted data looks like a 50% crash. ✅ **Solution:** yfinance auto-adjusts with `auto_adjust=True`. Historical prices are multiplied by the split ratio.

📌 *Common in BIST; Apple did 4:1 in 2020, Tesla did 5:1 in 2020.*

Dividend

⚠️ **Problem:** On the ex-dividend date, stock price drops by the dividend amount. Creates artificial negative returns. ✅ **Solution:** Use adjusted close (Adj Close) with `auto_adjust=True`. Total return is preserved.

📌 *Turkish banks have high dividend yields — using unadjusted data is a major error.*

Capital Increase / Rights

⚠️ **Problem:** Bonus/rights share issuance creates dilution effects. Historical returns are distorted. ✅ **Solution:** Using fully adjusted close price is mandatory.

📌 *Preemptive rights in BIST — requires careful price history examination.*

💡 Rule: Always use `auto_adjust=True` or `actions=True`. Training models on unadjusted prices is the most common source of look-ahead bias.

SECTION 4

Feature Engineering

Returns · Volatility · Momentum · Technical indicators

SEN605 · Week 2

Feature Name	Formula	Meaning	Typical Usage
<code>ret_1d</code>	$\log(P_t/P_{t-1})$	Daily log return	Main signal feature
<code>ret_5d</code>	$\log(P_t/P_{t-5})$	5-day return	Weekly momentum
<code>ret_21d</code>	$\log(P_t/P_{t-21})$	Monthly return	Monthly momentum
<code>ret_63d</code>	$\log(P_t/P_{t-63})$	Quarterly return	Medium-term trend
<code>ret_252d</code>	$\log(P_t/P_{t-252})$	52-week return	Long-term momentum
<code>fw_ret_1d</code>	$\log(P_{t+1}/P_t)$	Next-day return	Target variable (Y)
<code>fw_ret_5d</code>	$\log(P_{t+5}/P_t)$	5-day forward return	Medium-term target (Y)
#			

Realized Volatility

$$\sigma_n = \text{std}(\log_ret) \times \sqrt{252}$$

Annualized standard deviation of n-day log returns. The most fundamental volatility measure.

```
df['vol_21d'] =
df['log_ret'].rolling(21).std() *
np.sqrt(252)
```

Parkinson Volatility

$$\sigma = \sqrt{(1/4n \cdot \ln 2 \times \Sigma [\ln(H/L)]^2)}$$

Calculated from the high-low price range. Captures intraday movement better.

```
df['park_vol'] =
np.sqrt((np.log(df['High']/df['Low'])
)**2 /
(4*np.log(2))).rolling(21).mean() *
252)
```

Volatility Ratio

$$VR = \text{vol_short} / \text{vol_long}$$

Ratio of short-term to long-term volatility. >1 indicates increasing uncertainty, <1 indicates calming.

```
df['vol_ratio'] = df['vol_5d'] /
df['vol_21d']
```

RSI (14)

$$RSI = 100 - 100/(1+RS)$$

Relative Strength Index: value between 0-100. >70 overbought, <30 oversold.

```
delta = df['Close'].diff()
gain = delta.clip(lower=0).rolling(14).mean()
loss = -delta.clip(upper=0).rolling(14).mean()
df['rsi'] = 100 - 100/(1+gain/loss)
```

Bollinger Bands

$$\text{Upper/Lower} = \text{MA}_{20} \pm 2\sigma$$

Middle band ± 2 standard deviations. Band width is a volatility indicator; price outside bands signals extreme movement.

```
df['bb_mid'] = df['Close'].rolling(20).mean()
df['bb_std'] = df['Close'].rolling(20).std()
df['bb_up'] = df['bb_mid'] + 2*df['bb_std']
df['bb_low'] = df['bb_mid'] - 2*df['bb_std']
df['bb_pos'] = (df['Close']-df['bb_low'])/(df['bb_up']-df['bb_low'])
```

MACD

$$\text{MACD} = \text{EMA}_{12} - \text{EMA}_{26}; \text{Sinyal} = \text{EMA}_9(\text{MACD})$$

Difference between two EMAs. Crossing above signal line is buy, below is sell signal.

```
df['ema12'] = df['Close'].ewm(span=12).mean()
df['ema26'] = df['Close'].ewm(span=26).mean()
df['macd'] = df['ema12'] - df['ema26']
df['macd_sig'] = df['macd'].ewm(span=9).mean()
```

ATR (14)

$$\text{ATR} = \text{rolling}(14).mean(\text{True Range})$$

Average True Range: a price-based risk measure used for volatility and stop-loss calculations.

```
df['tr'] = np.maximum(
    df['High']-df['Low'],
    np.maximum(abs(df['High']-df['Close'].shift()),
               abs(df['Low']-df['Close'].shift())))
df['atr14'] = df['tr'].rolling(14).mean()
```

SECTION 5

Multi-Asset Data and Correlation

Portfolio perspective · Correlation matrix · Covariance

Importance of Multi-Asset Analysis

- Modeling a single stock in isolation is unrealistic — correlations exist
- Covariance matrix is essential for portfolio optimization
- Possible to include common factors (market, sector) as features
- Reference needed for index return comparison (alpha calculation)
- Cross-asset signals: the effect of currency movements on

Multi-symbol download

```
# BIST-30 large companies
tickers = {
    'AKBNK': 'AKBNK.IS', 'GARAN': 'GARAN.IS',
    'THYAO': 'THYAO.IS', 'ASELS': 'ASELS.IS',
    'BIMAS': 'BIMAS.IS', 'FROTO': 'FROTO.IS',
}
raw = yf.download(list(tickers.values()),
    start="2021-01-01", end="2024-12-31",
    auto_adjust=True)['Close']
```

Return matrix and basic statistics

```
# Log return matrix
returns = np.log(raw / raw.shift(1)).dropna()
print(f"Boyut: {getiriler.shape}")
# Shape: (756, 6)

# Annual return (for each stock)
ann_return = returns.mean() * 252
print("Annual Mean Return:")
print(ann_return.round(4))

# Annual volatility
ann_vol = returns.std() * np.sqrt(252)
print("\nAnnual Volatility:")
print(ann_vol.round(4))

# Sharpe ratio (risk-free rate ~25% TRY)
rf = 0.25 / 252 # Daily risk-free rate
sharpe = (returns.mean()-rf) / returns.std() *
np.sqrt(252)
print("\nSharpe Ratio:")
print(sharpe.round(2))
```

Pearson Correlation Interpretation

- +1.0: Perfect positive correlation — rise/fall together
- +0.7 to +0.9: Strong positive — same-sector stocks
- +0.3 to +0.6: Moderate positive — similar market sensitivity
- Near 0: Uncorrelated — ideal for portfolio diversification
- Negative: Opposite movement — can be used as a hedge

Limitations of Correlation

- Measures linear relationship — may miss nonlinear dependencies
- Correlations converge toward 1 during crises (covariance trap)
- Sensitive to sampling period — 1 year vs. 5 years give different results
- Spurious correlation risk: a common trend can move both
- Copula models capture tail dependence better

Correlation matrix and heatmap

```
import matplotlib.pyplot as plt
import seaborn as sns

# Pearson correlation matrix
corr = returns.corr()

# Heatmap (visualization)
fig, ax = plt.subplots(figsize=(8, 6))
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(corr, annot=True, fmt='.2f',
            cmap='RdYlGn', center=0,
            mask=mask, # Lower triangle only
            linewidths=0.5,
            ax=ax,
            vmin=-1, vmax=1)
ax.set_title('BIST-30 Selected Stocks\nReturn
Correlation (2021-2024)',
            fontweight='bold', pad=12)
plt.tight_layout()
plt.savefig('korelasyon.png', dpi=150)
plt.show()
```

Highest/lowest correlation pairs

SECTION 6

Lab: BIST-30 Analiz Pipeline

A complete end-to-end data preparation exercise

Pipeline Steps 1–2: Data Download and Cleaning

Lab exercise — Open your Jupyter Notebook

1

Import Libraries

```
#  
  
import yfinance as yf  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
plt.style.use('seaborn-v0_8-whitegrid')  
print("✅ Libraries ready")
```

2

Download and Clean BIST-30 Data

```
#  
  
tickers = {  
    'AKBNK': 'AKBNK.IS', 'GARAN': 'GARAN.IS',  
    'THYAO': 'THYAO.IS', 'ASELS': 'ASELS.IS',  
    'BIMAS': 'BIMAS.IS', 'FROTO': 'FROTO.IS',  
    'EREGL': 'EREGL.IS', 'SISE': 'SISE.IS',  
    'TTKOM': 'TTKOM.IS', 'KCHOL': 'KCHOL.IS',  
}  
  
raw = yf.download(list(tickers.values()),
```

3

Calculate Return Matrix

```
# Log return matrix
returns = np.log(raw / raw.shift(1)).dropna()
# Basic statistics for each stock
stats = pd.DataFrame({
    'Annual Return (%)': (returns.mean()*252*100).round(2),
    'Annual Volatility (%)': (returns.std()*np.sqrt(252)*100).round(2),
    'Sharpe (rf=25%)': ((returns.mean()-(0.25/252)) / returns.std() * np.sqrt(252)).round(2),
    'Min Return (%)': (returns.min()*100).round(2),
    'Max Return (%)': (returns.max()*100).round(2),
})
print(stats)
```

Feature Engineering for a Single Stock

4

```
# Feature matrix for AKBNK
df = raw[['AKBNK']].copy()
df.columns = ['Close']

ret = np.log(df['Close']/df['Close'].shift(1))
df['ret_1d'] = ret
df['ret_5d'] = np.log(df['Close']/df['Close'].shift(5))
df['ret_21d'] = np.log(df['Close']/df['Close'].shift(21))
df['vol_10d'] = ret.rolling(10).std()*np.sqrt(252)
```

Pipeline Steps 5–6: Correlation Analysis and Visualization

5

Correlation Matrix and Heatmap

```
#

# Correlation matrix
corr = returns.corr()

# Plot heatmap
fig, ax = plt.subplots(figsize=(10, 8))
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(corr, annot=True, fmt='.2f',
```

6

Cumulative Return Chart

```
cm = 'bgylg'
sns.heatmap(corr, annot=True, fmt='.2f',
            cmap=cm, center=0, mask=mask,
            linewidths=0.5, ax=ax, vmin=-1, vmax=1,

#

# Cumulative return (each stock starts at 100)
cumulative = (1 + returns).cumprod() * 100

fig, ax = plt.subplots(figsize=(12, 6))
colors = ['#1A2A5E', '#0D7A8C', '#C9922A',
          '#C2540A', '#15803D', '#6B3FA0',
          '#DC2626', '#1CA8C0', '#64748B', '#E8B84B']
for i, ticker in enumerate(cumulative.columns):
```

SECTION 7

Full Pipeline: Code Architecture

Production-quality financial data processing class

FinancialDataPipeline class (part 1)

```
class FinancialDataPipeline:
    """Complete data preparation pipeline for BIST stocks."""

    def __init__(self, tickers: dict, start: str, end: str):
        self.tickers = tickers # {'AKBNK': 'AKBNK.IS', ...}
        self.start = start
        self.end = end
        self.raw_data = None
        self.clean_data = None
        self.features = None

    def download_data(self):
        """Download raw OHLCV data."""
        raw = yf.download(
            list(self.tickers.values()),
            start=self.start, end=self.end,
            auto_adjust=True)['Close']
        raw.columns = list(self.tickers.keys())
        self.raw_data = raw
        print(f"✅ Downloaded: {raw.shape}")
        return self
```

```
    def clean(self):
```

FinancialDataPipeline class (part 2)

```
def extract_features(self, target_ticker: str):
    """Create full feature matrix for a single stock."""
    df = self.clean_data[[target_ticker]].copy()
    df.columns = ['Close']
    r = np.log(df['Close'] / df['Close'].shift(1))

    # Return features
    df['ret_1d'] = r
    df['ret_5d'] = np.log(df['Close'] / df['Close'].shift(5))
    df['ret_21d'] = np.log(df['Close'] / df['Close'].shift(21))

    # Volatility features
    df['vol_10d'] = r.rolling(10).std() * np.sqrt(252)
    df['vol_21d'] = r.rolling(21).std() * np.sqrt(252)
    df['vol_oran'] = df['vol_10d'] / df['vol_21d']

    # Momentum features
    df['ma10_oran'] = df['Close'] / df['Close'].rolling(10).mean() - 1
    df['ma50_oran'] = df['Close'] / df['Close'].rolling(50).mean() - 1

    # Target variable: tomorrow's log return
    df['target_return'] = r.shift(-1)
    df['target_dir'] = (df['target_return'] > 0).astype(int)
```



Veri Structures

- Tick → OHLCV transformation hierarchy
- Time / trade / volume bars
- Information loss and gain in bar construction



Pandas Tools

- Time series operations with DatetimeIndex
- Frequency conversion with resample()
- rolling() / expanding() / ewm() windows



Data Cleaning

- Gap filling strategy with ffill()
- Outlier detection with Z-score and IQR
- auto_adjust=True — corporate action requirement



Feature Eng.

- Multi-scale return features (1d, 5d, 21d)
- Volatility: realized, Parkinson, ratio
- RSI, MACD, Bollinger, ATR calculations



Week 3: Financial Time Series — Stationarity, Autocorrelation, and ARIMA Models

Week 2 Assignment

1

Apply a complete cleaning pipeline for 5 stocks of your choice from BIST-30 (ffill, outlier detection, corporate action verification)

25 points

2

Calculate the following features for each stock: ret_1d, ret_5d, ret_21d, vol_10d, vol_21d, vol_ratio, ma10_ratio, RSI_14

25 points

3

Calculate the return correlation matrix for the 5 stocks and visualize with a heatmap. Interpret the highest and lowest correlated pairs

25 points

4

Use the FinancialDataPipeline class to run all steps in a single function call and save the output DataFrame (.csv)

25 points