

Simple Finance Activities with a Microeconomics Lens by Prof. Khazra

This notebook contains graded exercises

This notebook contains **two beginner-friendly topics (topic A and B)**. Both start from a small `DataFrame` and use only a few core Python ideas:

- look at a table of data
- create new columns
- compute means/sums
- make one simple plot

Micro connection (theme): *choice under constraints* — households choosing consumption/savings, and/or agents choosing under risk (concavity).

Topic A — From Prices to Returns to “Consumption Tomorrow” (Intertemporal Choice)

We convert prices into returns, then interpret the return as the **gross interest factor** $R = 1 + r$ in a two-period budget constraint:

- If you save s today, you get $R \times s = Rs$ tomorrow.
- If you borrow b today, you repay Rb tomorrow.

Budget constraint (two-period, simple form):

- Today: $c_0 = y_0 - s$
 - Tomorrow: $c_1 = y_1 + Rs$
-

Topic B — Risk Basics as Expected Utility (Concavity)

We simulate a few returns, then compare:

- average return (a payoff level)
- volatility (dispersion)

Micro connection: with concave utility $u(c)$, risk lowers expected utility:

- Example: $u(c) = \sqrt{c}$ (risk-averse).
- We compare $\mathbb{E}[u(W)]$ across assets where W is wealth after investing.

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

np.set_printoptions(precision=4, suppress=True)
```

Topic A: Prices → Returns → Intertemporal Choice

Goal

1. Create a tiny price dataset for 2 assets
2. Compute simple returns $r_t = \frac{P_t - P_{t-1}}{P_{t-1}}$
3. Convert returns into a **gross return** $R_t = 1 + r_t$
4. Use R to compute a **simple consumption-savings** outcome

Interpretation (micro)

Suppose a household has income today y_0 and tomorrow y_1 and decides to save s today at interest factor R .

Then:

- $c_0 = y_0 - s$
- $c_1 = y_1 + Rs$

We'll compute c_1 for a few possible savings choices and **plot the budget line**.

```
In [2]: # Small, hand-sized dataset: 6 months of prices
dates = pd.date_range("2025-01-31", periods=6, freq="M")

prices = pd.DataFrame({
    "Date": dates,
    "Stock_Price": [100, 103, 101, 106, 108, 110],
    "Bond_Price": [100, 100.5, 100.7, 100.9, 101.2, 101.4],
})

prices
```

```
/tmp/ipykernel_500/1257664253.py:2: FutureWarning: 'M' is deprecated and
will be removed in a future version, please use 'ME' instead.
  dates = pd.date_range("2025-01-31", periods=6, freq="M")
```

Out[2]:

	Date	Stock_Price	Bond_Price
--	------	-------------	------------

0	2025-01-31	100	100.0
1	2025-02-28	103	100.5
2	2025-03-31	101	100.7
3	2025-04-30	106	100.9
4	2025-05-31	108	101.2
5	2025-06-30	110	101.4

```
In [3]: # Compute simple returns
#  $r_t = (P_t - P_{t-1}) / P_{t-1}$ 
prices["Stock_Return"] = prices["Stock_Price"].pct_change()
prices["Bond_Return"] = prices["Bond_Price"].pct_change()

# Gross returns  $R = 1 + r$ 
prices["Stock_R"] = 1 + prices["Stock_Return"]
prices["Bond_R"] = 1 + prices["Bond_Return"]

prices
```

Out[3]:

	Date	Stock_Price	Bond_Price	Stock_Return	Bond_Return	Stock_R	Bond_R
--	------	-------------	------------	--------------	-------------	---------	--------

0	2025-01-31	100	100.0	NaN	NaN	NaN	NaN
1	2025-02-28	103	100.5	0.030000	0.005000	1.030000	1.005000
2	2025-03-31	101	100.7	-0.019417	0.001990	0.980583	1.001990
3	2025-04-30	106	100.9	0.049505	0.001986	1.049505	1.001986
4	2025-05-31	108	101.2	0.018868	0.002973	1.018868	1.002973
5	2025-06-30	110	101.4	0.018519	0.001976	1.018519	1.001976

Pick a single interest factor R from the data

To keep things simple, we'll use the **average gross return** of the bond as our "interest factor":

- $R = \mathbb{E}[1 + r]$

(You can swap in the stock if you want a higher/rougher return.)

```
In [4]: R = prices["Bond_R"].dropna().mean()
r_avg = R - 1

print(f"Average bond gross return R ≈ {R:.5f}") #display 5 decimal poi
print(f"Average bond simple return r ≈ {r_avg:.5%}")
```

Average bond gross return R ≈ 1.00279

Average bond simple return r ≈ 0.27851%

Build an intertemporal budget line

Assume:

- income today: $y_0 = 100$
- income tomorrow: $y_1 = 105$

If the household chooses savings s today:

- $c_0 = y_0 - s$
- $c_1 = y_1 + R \times s$

We'll compute (c_0, c_1) for a range of s (including borrowing when $s < 0$) and plot the tradeoff.

```
In [5]: y0, y1 = 100, 105

# Savings grid: negative means borrowing, positive means saving
s_grid = np.arange(-40, 41, 10) # -40, -30, ..., 40

# Create savings column first
s = s_grid

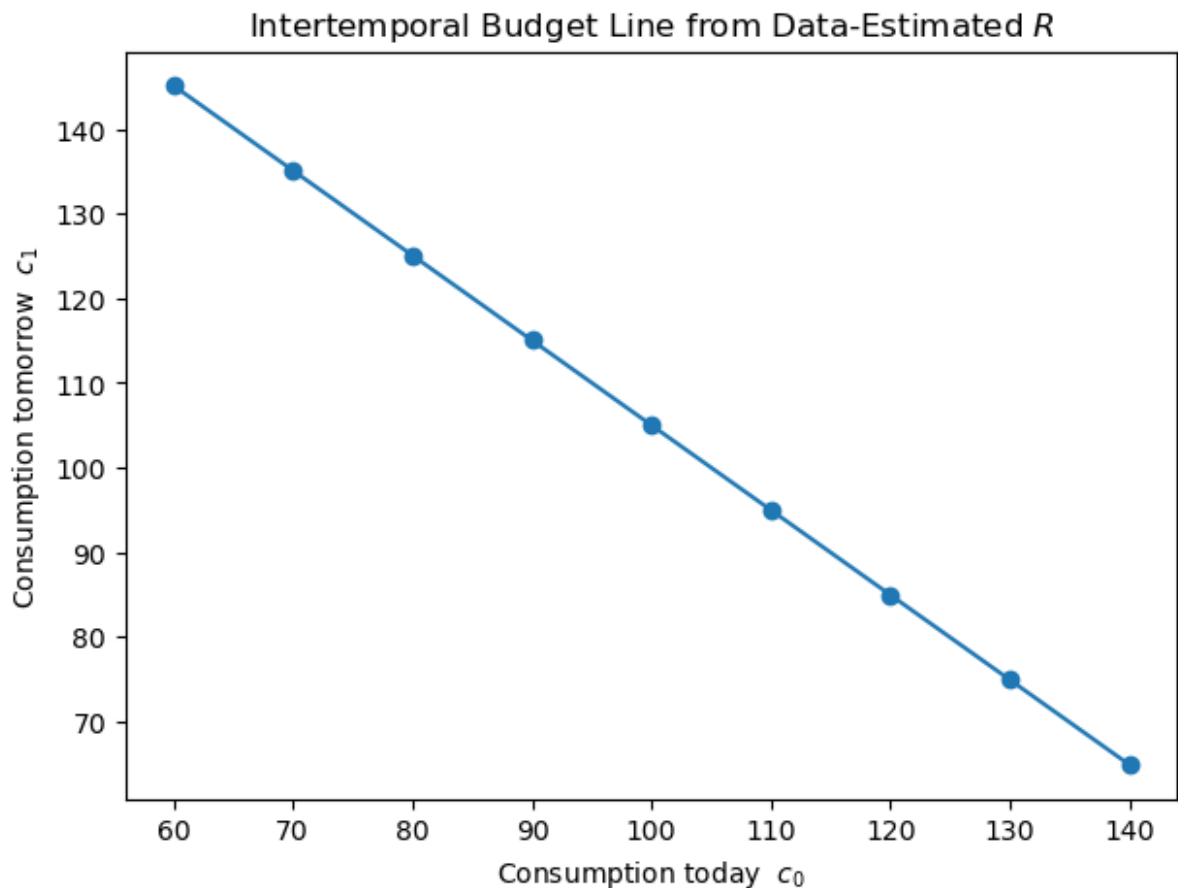
# Build DataFrame step-by-step (clear and readable)
budget = pd.DataFrame()
budget["s"] = s
budget["c0"] = y0 - s
budget["c1"] = y1 + R * s

budget
```

```
Out[5]:
```

	s	c0	c1
0	-40	140	64.888595
1	-30	130	74.916446
2	-20	120	84.944297
3	-10	110	94.972149
4	0	100	105.000000
5	10	90	115.027851
6	20	80	125.055703
7	30	70	135.083554
8	40	60	145.111405

```
In [6]: plt.figure(figsize=(7,5))
plt.plot(budget["c0"], budget["c1"], marker="o")
plt.xlabel("Consumption today $c_0$")
plt.ylabel("Consumption tomorrow $c_1$")
plt.title("Intertemporal Budget Line from Data-Estimated $R$")
#plt.gca().invert_xaxis() # optional: makes 'more today' appear to th
plt.show()
```



Take home mini-exercises for Topic A

1. Change the savings grid to include $s = 55$ and $s = -55$. What happens?
2. Replace the bond-based R with stock-based R :
 - set `R = prices["Stock_R"].dropna().mean()`
 - re-run the budget line plot
3. Micro question: if preferences are $U(c_0, c_1) = \ln(c_0) + \beta \ln(c_1)$, what does a higher R tend to do to optimal savings s ? (Intuition only.)

```
In [7]: # your code for (1)
```

```
In [8]: # your code for (2)
```

your answer to (3)

Topic B: Risk & Expected Utility with a Small Returns DataFrame

Goal

1. Create a tiny returns dataset (already in returns, not prices)
2. Compare “average” vs “risk” (volatility)
3. Compute **expected utility** for a risk-averse agent using a simple concave utility

Micro connection

Risk aversion is captured by concavity of $u(c)$.

We'll use:

- wealth today: $W_0 = 100$
- invest all wealth in one asset for one period
- wealth tomorrow: $W_1 = W_0(1 + r)$

Utility examples:

- $u(W) = \sqrt{W}$ (risk-averse, simple)
- Expected utility: $\mathbb{E}[u(W_1)]$

We'll rank assets by:

- average return
- volatility
- expected utility

```
In [9]: returns = pd.DataFrame({
    "Scenario": ["Bad", "Okay", "Good", "Great"],
    "Stock_r": [-0.10, 0.02, 0.05, 0.12],
    "Bond_r": [-0.01, 0.01, 0.015, 0.02],
    "Crypto_r": [-0.30, 0.05, 0.15, 0.50],
})

returns
```

```
Out[9]:
```

	Scenario	Stock_r	Bond_r	Crypto_r
0	Bad	-0.10	-0.010	-0.30
1	Okay	0.02	0.010	0.05
2	Good	0.05	0.015	0.15
3	Great	0.12	0.020	0.50

```
In [10]: stats = pd.DataFrame({
    "Mean return": returns[["Stock_r", "Bond_r", "Crypto_r"]].mean(),
    "Volatility (std)": returns[["Stock_r", "Bond_r", "Crypto_r"]].std()
})

stats
```

```
Out[10]:
```

	Mean return	Volatility (std)
Stock_r	0.02250	0.091788
Bond_r	0.00875	0.013150
Crypto_r	0.10000	0.329140

Expected utility calculation (super concrete)

Assume $W_0 = 100$ and invest all wealth in one asset.

Then each scenario implies:

- $W_1 = W_0(1 + r)$
- utility: $u(W_1) = \sqrt{W_1}$

Then expected utility is the simple average across scenarios:

- $\mathbb{E}[u(W_1)] = \frac{1}{N} \sum_{i=1}^N \sqrt{W_{1,i}}$

In [11]: `W0 = 100`

```
def expected_utility_of_asset(r_series, W0=100):
    W1 = W0 * (1 + r_series)
    u = np.sqrt(W1)
    return u.mean()

EU = pd.Series({
    "Stock": expected_utility_of_asset(returns["Stock_r"], W0),
    "Bond": expected_utility_of_asset(returns["Bond_r"], W0),
    "Crypto": expected_utility_of_asset(returns["Crypto_r"], W0),
}, name="Expected utility (u(W)=sqrt(W))")

EU
```

Out[11]:

Stock	10.104073
Bond	10.043494
Crypto	10.396201

Name: Expected utility (u(W)=sqrt(W)), dtype: float64

In [12]:

```
stats["Expected utility (u(W)=sqrt(W))"] = EU.values
ranking = stats.sort_values("Expected utility (u(W)=sqrt(W))", ascending=False)
ranking
```

Out[12]:

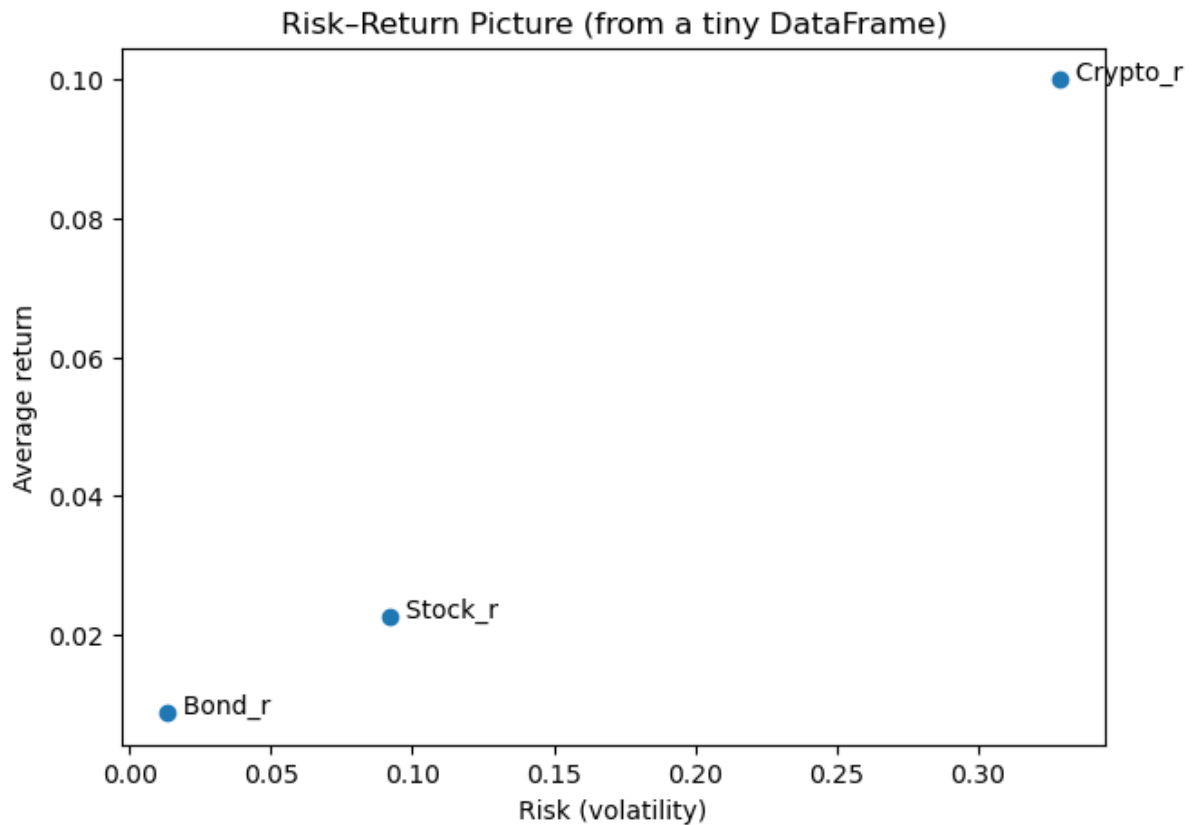
	Mean return	Volatility (std)	Expected utility (u(W)=sqrt(W))
Crypto_r	0.10000	0.329140	10.396201
Stock_r	0.02250	0.091788	10.104073
Bond_r	0.00875	0.013150	10.043494

In [13]:

```
plt.figure(figsize=(7,5))
plt.scatter(stats["Volatility (std)"], stats["Mean return"])

for name in stats.index:
    plt.text(stats.loc[name, "Volatility (std)"], stats.loc[name, "Mean return"], name)

plt.xlabel("Risk (volatility)")
plt.ylabel("Average return")
plt.title("Risk-Return Picture (from a tiny DataFrame)")
plt.show()
```



Take home mini-exercises for Topic B

1. Change the utility to $u(W) = \ln(W)$. Recompute and see if the ranking changes.
2. Add a new asset column (e.g., `Gold_r`) with your own scenario returns and see where it ranks.
3. Micro/finance question: why can an asset with a **higher mean return** still have **lower expected utility** for a risk-averse agent?

In [14]: `# your code for (1)`

In [15]: `# your code for (2)`

your text for (3)