Portfolio Analysis

Using portfolio sorts

Agenda

- Opening Notes
- Motivation
- Conceptual Overview
  - Portfolio Sorts
    - Univariate Portfolio Analysis
    - Bivariate Independent-Sort Analysis
    - Bivariate Dependent-Sort Analysis
    - Considerations
- Python Introduction
  - Portfolio Sorts
  - Important Notes for Practical Applications
  - Parallelization

Opening Notes

- This course closely follows the following book:

  Empirical Asset Pricing

  The Cross Section of Stock Returns

  Turan G. Bali, Robert F. Engle, Scott Murray

- Chapter 5 of the book covers portfolio analysis. The formulas, explanations etc. on the following slides are mostly taken from this chapter and occasionally complemented.

Motivation

- The objective of portfolio sorts is to examine the cross-sectional relation between two or more variables.
- Most commonly it is used to determine if a variable (“X”, e.g. market capitalization) can explain/predict stock returns (“Y”).
- Examples:
  - Do stocks with higher/lower market capitalization (X) show high/low returns (Y) over time?
  - Do stocks with high market capitalization (X) exhibit high skewness risk (Y)?
  - Do stocks with low market capitalization (X1) and high skewness risk (X2) exhibit high returns (Y) over time?

Conceptual Overview (1/3)

- Portfolio analysis is a commonly used statistical methodology in empirical asset pricing.
- The general approach is to form portfolios of stocks, where the stocks in each portfolio have different levels of the variable or variables posited to predict cross-sectional variation in future returns and to examine the returns of these portfolios.
- One can examine the cross-sectional relation between two or more variables (for example size and beta) and asses the predictive power of these variables.

Conceptual Overview (2/3)

- It is a **nonparametric** technique, i.e. it does not make any assumptions about the nature of the cross-sectional relations between the variables under investigation. (opposed to linear regression which assumes linear relations)
- The drawback is that it is difficult to control for a large number of variables when examining the cross-sectional relation of interest.
- There exist different variations of portfolio sorting algorithms, we will cover the most common ones, all others are derivative.
- Y will be the outcome variable, the variable of interest.
- X will be the sort variables

Conceptual Overview (3/3)

- The sorted portfolios are rebalanced/recalculated over multiple time periods t, for example every month.
- Each period t, all entities/stocks are sorted anew by their values of the sort variable(s) X. Since the values (beta) change over time, the composition of the sorted portfolios also changes over time.
- The sorting algorithm is of course fixed.

Univariate Portfolio Analysis (1/18)

- Is the most basic type of portfolio analysis, has only one sort variable X.
- The objective of the analysis is to assess the cross-sectional relation between X and the outcome variable Y.
- There are four steps:

1. Calculate breakpoints for each period t
2. Form portfolios for each period t
3. Calculate average value of Y within each portfolio for each period t
4. Examine variation in average values of Y across the different portfolios

Univariate Portfolio Analysis (2/18)

- (1) Breakpoints
- The periodic breakpoints will be used to group the stocks into different portfolios based on values of the sort variable X.
- Entities with values of X that are less than or equal to the first breakpoint will be placed into the first portfolio. Entities with values of X that are between (including) the first and second breakpoints will comprise the second portfolio, etc. Finally, entities with X values higher than or equal to the highest breakpoint will be placed in the last portfolio.
- n_p: number of portfolios
- Therefore, we need $n_p - 1$ breakpoints B.

Univariate Portfolio Analysis (3/18)

- (1) Breakpoints
- The number of portfolios to be formed and, thus, the number of breakpoints to be calculated is the same for all time periods.
- The value of the kth breakpoint will almost certainly vary for the time periods t.
- The kth breakpoint for period t will be denoted:
- $B_{k,t}$ for $k \in \{1, 2, \dots, n_p - 1\}$
- The breakpoints are determined by **percentiles** of the time t cross-sectional distribution of the sort variable X. The breakpoint for period t is calculated as the p_kth percentile of the values of X across all entities in the sample for which X is **available** in period t.

Univariate Portfolio Analysis (4/18)

- (1) Breakpoints
- The breakpoint is defined as:
- $B_{k,t} = Pctl_{pk}(\{X_t\})$
- The chosen percentiles p_k must be increasing, but it can be the case, that, if for a large number of entities the values of X are the same, two or more of the breakpoints are the same.
- Note: Sometimes, breakpoints are calculated using only a subset of the available entities. For example, in research where the entities are stocks, sometimes researchers form breakpoints using only stocks that trade on the NYSE, and then use those breakpoints to sort all stocks in the sample (including stocks that trade on other exchanges) into portfolios.

Univariate Portfolio Analysis (5/18)

- (1) Breakpoints
- Choosing an appropriate number of portfolios and choosing appropriate percentiles for the breakpoints are important decisions in portfolio analysis. Trade off between the number of entities in each portfolio against the dispersion of the sort variable among the portfolios.
- Most commonly, portfolios are formed using breakpoints that represent evenly spaced percentiles of the cross-sectional distribution of the sort variable.
- Example five portfolios: 20^{th}, 40^{th}, 60^{th}, 80^{th} percentiles of X
- But for a three portfolio split it is common to use 30^{th} and 70^{th} percentiles.
- When in doubt: Follow the literature!

Univariate Portfolio Analysis (6/18)

- (1) Breakpoints

TABLE 5.1 Univariate Breakpoints for β-Sorted Portfolios

This table presents breakpoints for β-sorted portfolios. Each year t, the first ($B_{1,t}$), second ($B_{2,t}$), third ($B_{3,t}$), fourth ($B_{4,t}$), fifth ($B_{5,t}$), and sixth ($B_{6,t}$) breakpoints for portfolios sorted on β are calculated as the 10th, 20th, 40th, 60th, 80th, and 90th percentiles, respectively, of the cross-sectional distribution of β. Each row in the table presents the breakpoints for the year indicated in the first column. The subsequent columns present the values of the breakpoints indicated in the first row.

t	$B_{1,t}$	$B_{2,t}$	$B_{3,t}$	$B_{4,t}$	$B_{5,t}$	$B_{6,t}$
1988	−0.05	0.07	0.29	0.51	0.86	1.11
1989	−0.11	0.05	0.29	0.54	0.89	1.17
1990	−0.06	0.10	0.37	0.68	1.07	1.37

Univariate Portfolio Analysis (7/18)

- (2) Portfolio Formation
- At the end of a period t, all stocks will be sorted into portfolios based on the breakpoints of the sort variable.
- Each time period t, all entities with values of the sort variable X that are **less than or equal to** the first breakpoint are put in portfolio one. Entities with values of X that are **greater than or equal to** the first breakpoint and less than or equal to the second breakpoint are sorted into portfolio two.
- The final portfolio n_p holds the entities with values of X that are greater than or equal to $B_{np\,-1,t}$.
- $P_{k,t} = \{i | B_{k-1,t} \leq X_{i,t} \leq B_{k,t}\}$ for $k \in \{1, 2, \dots, n_p - 1\}$
- Entities i

Univariate Portfolio Analysis (8/18)

- (2) Portfolio Formation
- Note: If a given entity i has a value of X during time period t that is exactly equal to the kth breakpoint, then this entity is included in both portfolio k and portfolio k + 1.
- Why we do this: It is possible that two (or more) consecutive breakpoints have exactly the same value, if there are a large number of entities with the same value of X. If we'd use a strict inequality, some or more portfolios would contain no entities.
- Example: Decile portfolios 10^{th} … 90^{th} percentile based on past returns. 30^{th} and 40^{th} percentile, i.e. breakpoints (return values), are both 0 in period t. If the stocks in the fourth portfolio are those that have returns that are greater than the third breakpoint, which is zero, and less than or equal to the fourth breakpoints, which is also zero, then there would be no stocks in the fourth portfolio.

Univariate Portfolio Analysis (9/18)

- (2) Portfolio Formation
- The problem is then, that some entities i will be included in more than one portfolio, but this issue is considered minor compared to having portfolios with no entities.
- If such a situation arises in your analysis, special attention should be paid to ensuring that this does not have an important impact on any of the conclusions drawn from the portfolio analysis. Your decision should be stated and justified.

Univariate Portfolio Analysis (10/18)

- (2) Portfolio Formation

TABLE 5.2 Number of Stocks per Portfolio

This table presents the number of stocks in each of the portfolios formed in each year during the sample period. The column labeled t indicates the year. The subsequent columns, labeled $n_{k,t}$ for $k \in \{1, 2, \ldots, 7\}$ present the number of stocks in the kth portfolio.

t	$n_{1,t}$	$n_{2,t}$	$n_{3,t}$	$n_{4,t}$	$n_{5,t}$	$n_{6,t}$	$n_{7,t}$
1988	569	569	1138	1138	1138	569	569
1989	552	552	1104	1103	1104	552	552
1990	541	541	1082	1081	1082	541	541

Univariate Portfolio Analysis (11/18)

- (3) Average Portfolio Values
- Calculate the average value of the outcome variable Y for each of the portfolios in each time period t.
- In most cases it is desirable to **weight the entities within each portfolio** according to some variable $W_{i,t}$ which is usually the **market capitalization**. The average is then referred to as the value-weighted average. The other commonly used case is **equally-weighted** portfolios.
- The weighting scheme most likely has substantial impact on the results (see later slides).
- General form: $\bar{Y}_{k,t} = \frac{\sum_{i \in P_{k,t}} W_{i,t} Y_{i,t}}{\sum_{i \in P_{k,t}} W_{i,t}}$ for $k \in \{1, 2, \ldots, n_p - 1\}$

Univariate Portfolio Analysis (12/18)

- (3) Average Portfolio Values
- Note: Value-weighting rather accounts for the stock market as a whole. Equal-weighting can point to phenomena of the average stock. The returns of equal-weighted portfolios are likely driven by low market capitalization stocks, which tend to be illiquid and expensive to trade.
- Note: It can be the case, that for a given entity i Y or W or both are not available (data). The summation should be taken over all entities for which values of both W and Y are available.

Univariate Portfolio Analysis (13/18)

- (3) Average Portfolio Values
- In addition to calculating the average value of Y for each portfolio, one can also calculate the difference in average values between portfolio n_p (the “highest” portfolio) and portfolio one (the “lowest” portfolio).
- $\bar{Y}_{Diff,t} = \bar{Y}_{n_p,t} - \bar{Y}_{1,t}$
- This difference is used to detect a cross-sectional relation between the sort variable X and the outcome variable Y, **which is the main objective of portfolio analysis.** It is often referred to as the average value of the difference portfolio.

Univariate Portfolio Analysis (14/18)

- (3) Average Portfolio Values

TABLE 5.3 Univariate Portfolio Equal-Weighted Excess Returns

This table presents the one-year-ahead excess returns of the equal-weighted portfolios formed by sorting on β. The column labeled t indicates the portfolio formation year. The column labeled $t+1$ indicates the portfolio holding year. The columns labeled 1 through 7 show the excess returns of the seven β-sorted portfolios. The column labeled 7-1 presents the difference between the return of portfolio seven and that of portfolio one.

t	$t+1$	1	2	3	4	5	6	7	7-1
1988	1989	−0.97	1.12	2.12	6.77	3.18	9.04	8.96	9.93
1989	1990	−30.21	−29.09	−28.72	−29.81	−27.85	−26.85	−25.75	4.45
1990	1991	56.81	28.99	36.22	40.42	54.51	64.44	66.49	9.68
1991	1992	55.37	30.93	29.45	21.99	19.16	15.95	20.12	−35.26

Univariate Portfolio Analysis (15/18)

- (3) Average Portfolio Values

TABLE 5.4 Univariate Portfolio Value-Weighted Excess Returns

This table presents the one-year-ahead excess returns of the value-weighted portfolios formed by sorting on β. The column labeled t indicates the portfolio formation year. The column labeled $t+1$ indicates the portfolio holding year. The columns labeled 1 through 7 show the excess returns of the seven β-sorted portfolios. The column labeled 7-1 presents the difference between the return of portfolio seven and that of portfolio one.

t	$t+1$	1	2	3	4	5	6	7	7-1
1988	1989	−2.98	11.14	9.04	16.29	20.25	29.16	18.21	21.19
1989	1990	−28.42	−33.78	−18.81	−19.12	−16.14	−11.51	−11.88	16.53
1990	1991	−0.45	18.50	12.28	17.79	22.08	31.14	51.51	51.96
1991	1992	−1.18	22.64	14.17	8.47	5.44	0.43	12.45	13.62

Univariate Portfolio Analysis (16/18)

- (4) Inspect results
- Calculate the time-series means of the period t average values of the outcome variable for each of the portfolios as well as the difference portfolio.
- $\bar{Y}_k = \frac{\sum_{t=1}^{T} \bar{Y}_{k,t}}{T}$
- Examine whether the time-series mean values are statistically distinguishable from zero using a hypothesis test. Calculate standard errors (Newey, West), t-statistics and p-values.
- A statistically nonzero mean **for the difference portfolio** is evidence that, in the average time period, a cross-sectional relation exists between the sort variable and the outcome variable.

Univariate Portfolio Analysis (17/18)

TABLE 5.5 Univariate Portfolio Equal-Weighted Excess Returns Summary

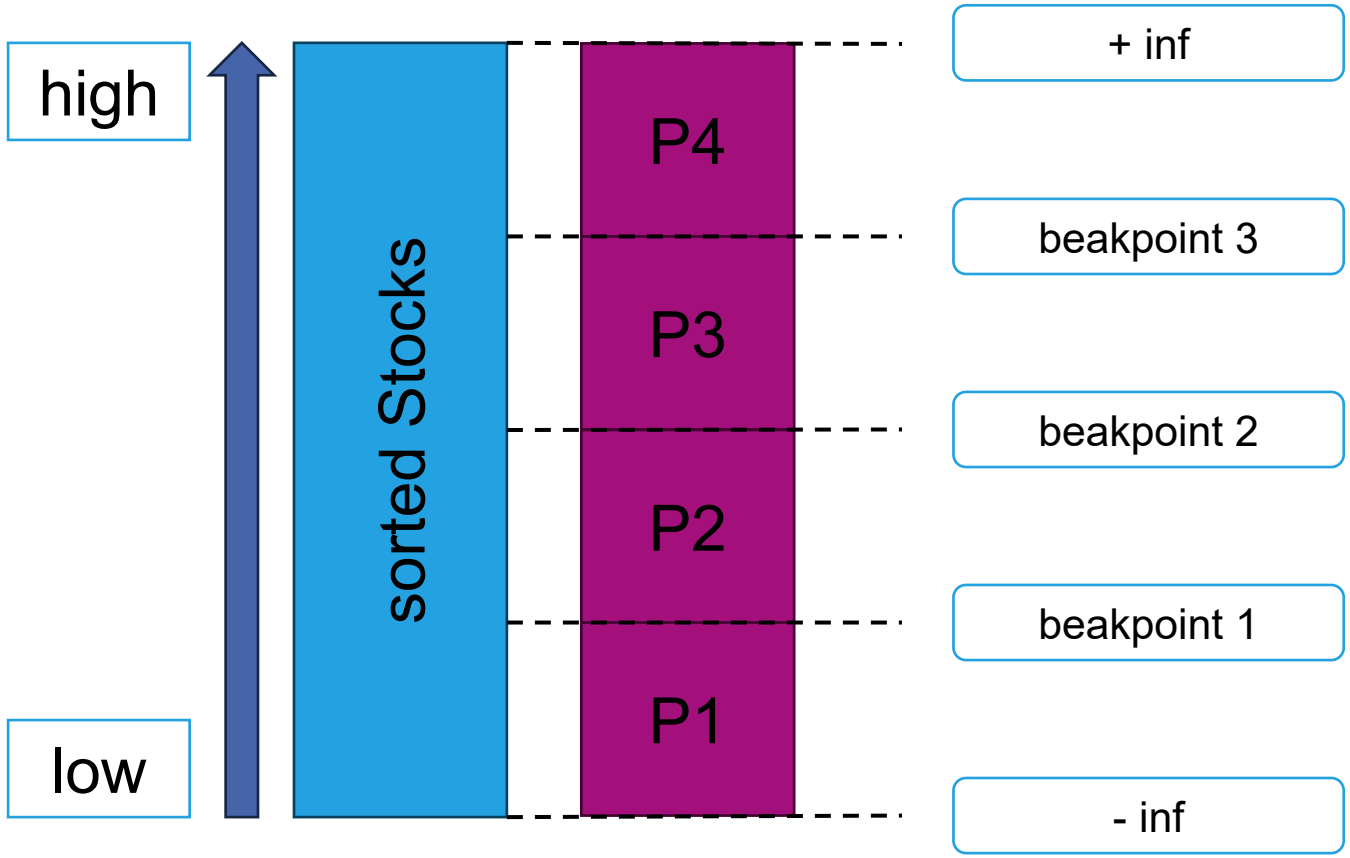
This table presents the results of a univariate portfolio analysis of the relation between beta (β) and future stock returns (r_{t+1}). The row labeled Average presents the equal-weighted average annual return for each of the portfolios. The row labeled Standard error presents the standard error of the estimated mean portfolio return. Standard errors are adjusted following Newey and West (1987) using six lags. The row labeled t-statistic presents the t-statistic (in parentheses) for the test with null hypothesis that the average portfolio excess return is equal to zero. The row labeled p-value presents the two-sided p-value for the test with null hypothesis that the average portfolio excess return is equal to zero. The columns labeled 1 through 7 show the excess returns of the seven β-sorted portfolios. The column labeled 7-1 presents the results for the difference between the return of portfolio seven and that of portfolio one.

	1	2	3	4	5	6	7	7-1
Average	16.47	13.89	14.55	12.79	11.99	10.92	10.43	−6.04
Standard error	3.62	2.42	2.50	1.90	1.83	1.80	3.04	4.61
t-statistic	4.55	5.74	5.83	6.73	6.57	6.06	3.43	−1.31
p-value	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20

Univariate Portfolio Analysis (18/18)

- (4) Inspect results
- Note: Excess return means “absolute” return minus risk free rate.
- In this case the average excess returns of the portfolios 1 through 7 are found to be highly statistically significant, however the average excess return **of the difference portfolio** is not statistically distinguishable from zero and therefore the analysis fails to detect a cross-sectional relation between β and the one-year-ahead excess stock return.
- Side note: It is common to control for sensitivity to systematic risk factors (CAPM, FF3, …). See book if interested.

Univariate Portfolio Analysis - Visual



This slide depicts the sorting logic for univariate sorting at one time step t

Bivariate Independent-Sort Analysis (1/11)

- Very similar to univariate portfolio analysis, except now there are two sort variables. Big surprise.
- Bivariate independent-sort portfolio analysis is designed to assess the cross-sectional relations between two sort variables X1 and X2, and an outcome variable Y.
- General steps are the same.
- All points to consider for univariate sorting remain for bivariate sorting.

Bivariate Independent-Sort Analysis (2/11)

- (1) Breakpoints
- As the name implies, in bivariate independent-sort portfolio analysis, portfolios are formed by sorting on two variables independently. Thus, in each period, two sets of breakpoints will be calculated, one corresponding to X1 and the other to X2. The breakpoints of X2 are calculated completely **independently** of the breakpoints of X1. Therefore, there is no order in the sort variables X1 and X2. Switching them does not affect the results. (Unconditional Sort)
- The entities are sorted into groups according to each sort variable.
- Portfolios will later represent intersections of groups.

Bivariate Independent-Sort Analysis (3/11)

- (1) Breakpoints
- n_{P1} are the number of groups for the first sort variable, n_{P2} for the second.
- We will have $n_{P1} \times n_{P2}$ portfolios.
- Note: Frequently the set of entities for which values of X1 are available may differ from those for which X2 is available. The researcher must decide whether the breakpoints are formed using only entities for which valid values of both variables are available or whether the breakpoints are formed using all available data for each variable.

Bivariate Independent-Sort Analysis (4/11)

- (1) Breakpoints
- How many breakpoints and which percentiles: If the sort variables are highly positively correlated, it may result in many entities being put into the portfolio that holds entities with high values of both sort variables as well as the portfolio holding entities with low values of both sort variables. Portfolios that hold entities with low values of one sort variable and high values of the other will contain relatively fewer entities. The situation is reversed when the sort variables are negatively correlated. The number of groups to form based on each sort variable, therefore, should take this correlation into account and ensure that for each time period, each portfolio contains a sufficient number of entities.

Bivariate Independent-Sort Analysis (5/11)

- (1) Breakpoints

TABLE 5.10 Bivariate Independent-Sort Breakpoints

This table presents the breakpoints for a bivariate independent-sort portfolio analysis The first sort variable is β and the second sort variable is *MktCap*. The sample is split into three groups (and thus two breakpoints) based on the 30th and 70th percentiles of β, and four groups (and thus three breakpoints) based on the 25th, 50th, and 75th percentiles of *MktCap*. The column labeled t indicates the year for which the breakpoints are calculated. The columns labeled $B1_{1,t}$ and $B1_{2,t}$ present the first and second β breakpoints, respectively. The columns labeled $B2_{1,t}$, $B2_{2,t}$, and $B3_{3,t}$ present the first, second, and third *MktCap* breakpoints, respectively.

t	$B1_{1,t}$	$B1_{2,t}$	$B2_{1,t}$	$B2_{2,t}$	$B2_{3,t}$
1988	0.18	0.66	9.65	34.83	159.85
1989	0.17	0.70	9.77	37.04	184.14

Bivariate Independent-Sort Analysis (6/11)

- (2) Portfolio formation
- $P_{j,k,t} = \{i|B1_{j-1,t} \leq X1_{i,t} \leq B1_{j,t}\} \cap \{i|B2_{k-1,t} \leq X2_{i,t} \leq B2_{k,t}\}$
- for j $\in \{1, 2, \dots, n_{P1}\}$ and $k \in \{1, 2, \dots, n_{P2}\}$
- $B1_{0,t} = B2_{0,t} = -\infty$ and $B1_{n_{P1},t} = B2_{n_{P2},t} = \infty$
- $\cap$ is intersection operator
- For an entity i to be held in Portfolio $P_{j,k,t}$ it must have a value of X1 in period t between the j-1st and jth (inclusive) period t breakpoints for X1 **and** have a period t value of X2 between the k-1st and kth (inclusive) period t breakpoints for X2.

Bivariate Independent-Sort Analysis (7/11)

- (2) Portfolio formation

TABLE 5.11 Bivariate Independent-Sort Number of Stocks per Portfolio

This table presents the number of stocks in each of the 12 portfolios formed by sorting independently into three β groups and four *MktCap* groups. The columns labeled t indicate the year of portfolio formation. The columns labeled β 1, β 2, and β 3 indicate the β group. The rows labeled *MktCap* 1, *MktCap* 2, *MktCap* 3, and *MktCap* 4 indicate the *MktCap* groups.

t		β 1	β 2	β 3	t		β 1	β 2	β 3
1988	*MktCap* 1	736	468	217	1998	*MktCap* 1	842	557	252
	MktCap 2	539	585	297		*MktCap* 2	622	667	361
	MktCap 3	335	683	403		*MktCap* 3	368	709	574
	MktCap 4	95	538	788		*MktCap* 4	149	708	794
1989	*MktCap* 1	736	419	224	1999	*MktCap* 1	785	484	253
	MktCap 2	537	574	268		*MktCap* 2	635	608	279
	MktCap 3	298	663	418		*MktCap* 3	338	729	455
	MktCap 4	84	549	746		*MktCap* 4	69	613	840

Bivariate Independent-Sort Analysis (8/11)

- (3) Average Portfolio Values
- As before: equal-weighted or weighted according to some variable W, usually market capitalization.
- One now calculates portfolio average and difference values:
  - Average portfolios (e.g. average of all “low X1” portfolios)
  - Difference portfolios (e.g. “high X2” – “low X2 portfolio given low X1)
  - Average of averages
  - Difference in difference portfolio
  - **See next two slides!**

Bivariate Independent-Sort Analysis (9/11)

- (4) Inspect Results (as before)

TABLE 5.12 Average Value for the Difference in Difference Portfolio

This diagram describes how the difference in difference portfolio for a bivariate-sort portfolio analysis is constructed.

	$X1\ 1$	$\cdots$	$X1\ n_{P1}$	$X1\ Diff$
$X2\ 1$	$\overline{Y}_{1,1,t}$ A	$\cdots$	$\overline{Y}_{n_{P1},1,t}$ B	$\overline{Y}_{Diff,1,t}$ $B-A$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$X2\ n_{P2}$	$\overline{Y}_{1,n_{P2},t}$ C	$\cdots$	$\overline{Y}_{n_{P1},n_{P2},t}$ D	$\overline{Y}_{Diff,n_{P2},t}$ $D-C$
$X2\ Diff$	$\overline{Y}_{1,Diff,t}$ $C-A$	$\cdots$	$\overline{Y}_{n_{P1},Diff,t}$ $D-B$	$(D-C)-(B-A)$ $=$ $(D-B)-(C-A)$ $=$ $D-C-B+A$

Bivariate Independent-Sort Analysis (10/11)

- (4) Inspect Results (as before)

TABLE 5.13 Bivariate Independent-Sort Portfolio Excess Returns

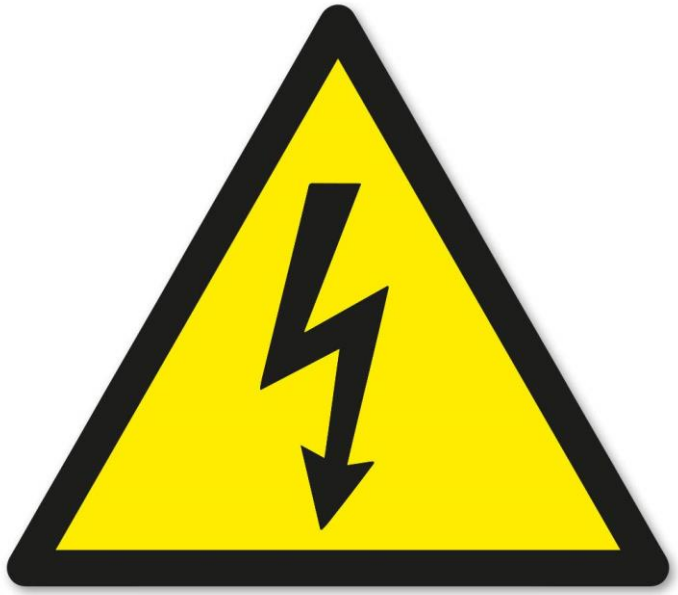
This table presents the equal-weighted excess returns for each of the 12 portfolios formed by sorting independently into three β groups and four *MktCap* groups, as well as for the difference and average portfolios. The columns labeled $t/t+1$ indicate the year of portfolio formation (t) and the portfolio holding period ($t+1$). The columns labeled β 1, β 2, β 3, β Diff, and β Avg indicate the β groups. The rows labeled *MktCap* 1, *MktCap* 2, *MktCap* 3, *MktCap* 4, *MktCap* Diff, and *MktCap* Avg indicate the *MktCap* groups.

$t/t+1$		β 1	β 2	β 3	β Diff	β Avg
1988/1989	*MktCap* 1	1.13	−0.96	1.69	0.56	0.62
	MktCap 2	−2.02	−1.49	−10.49	−8.47	−4.67
	MktCap 3	4.48	3.62	5.45	0.97	4.52
	MktCap 4	9.92	14.45	17.08	7.15	13.82
	MktCap Diff	8.80	15.40	15.39	6.59	13.20
	MktCap Avg	3.38	3.90	3.43	0.05	3.57

Bivariate Independent-Sort Analysis (11/11)

- (4) Inspect Result
- In most cases, the focal results of the portfolio analysis are in the differences between the portfolios that contain high and low values of a given variable.
- The differences in average Y values between portfolios with high values of X1 and low values of X1 (X1 Diff) indicate whether a cross-sectional relation between X1 and Y exists after controlling for the effect of X2.
- For example: Among low-β stocks, high-MarketCap stocks have significantly lower average returns than low-MarketCap stocks.

Bivariate Independent-Sort Analysis - Visual



Bivariate Dependent-Sort Analysis (1/4)

- Similar to independent sort, but the breakpoints for X2 are formed **within each group** of X1. (Conditional Sort)
- It is used to understand the relation between X2 and Y conditional on X1.
- The relation between X1 and Y is not examined, X1 is only a control variable.
- (1) Breakpoints
- $B2_{j,k,t} = Pctl_{p2_k}\left(\{X2_t | B1_{j-1,t} \leq X1_t \leq B1_{j,t}\}\right)$

Bivariate Dependent-Sort Analysis (2/4)

- (1) Breakpoints

TABLE 5.20 Bivariate Dependent-Sort Breakpoints

This table presents the breakpoints for portfolios formed by sorting all stocks in the sample into three groups based on the 30th and 70th percentiles of β, and then, within each β group, into four groups based on the 25th, 50th, and 75th percentiles of *MktCap* among only stocks in the given β groups. The columns labeled t indicates the year of the breakpoints. The columns labeled $B1_{1,t}$ and $B1_{2,t}$ present the β breakpoints. The columns labeled $B2_{1,k,t}$, $B2_{2,k,t}$, and $B2_{3,k,t}$ indicate the kth *MktCap* breakpoint for stocks in the first, second, and third β group, respectively, where k is indicated in the columns labeled k.

t	k	$B2_{1,k,t}$	$B1_{1,t}$	$B2_{2,k,t}$	$B1_{2,t}$	$B2_{3,k,t}$	t	k	$B2_{1,k,t}$	$B1_{1,t}$	$B2_{2,k,t}$	$B1_{2,t}$	$B2_{3,k,t}$
1988	1	4.57	0.18	12.49	0.66	25.31	2001	1	16.71	0.31	45.87	0.95	91.00
	2	12.36		39.76		126.79		2	44.75		231.86		350.97
	3	35.38		142.60		796.57		3	113.32		886.00		1162.14

Bivariate Dependent-Sort Analysis (3/4)

- (2) Portfolio Formation
- $P_{j,k,t} = \{i|B1_{j-1,t} \leq X1_{i,t} \leq B1_{j,t}\} \cap \{i|B2_{j,k-1,t} \leq X2_{i,t} \leq B2_{j,k,t}\}$
- for j $\in \{1, 2, \ldots, n_{P1}\}$ and $k \in \{1, 2, \ldots, n_{P2}\}$

TABLE 5.21 Bivariate Dependent-Sort Number of Stocks per Portfolio

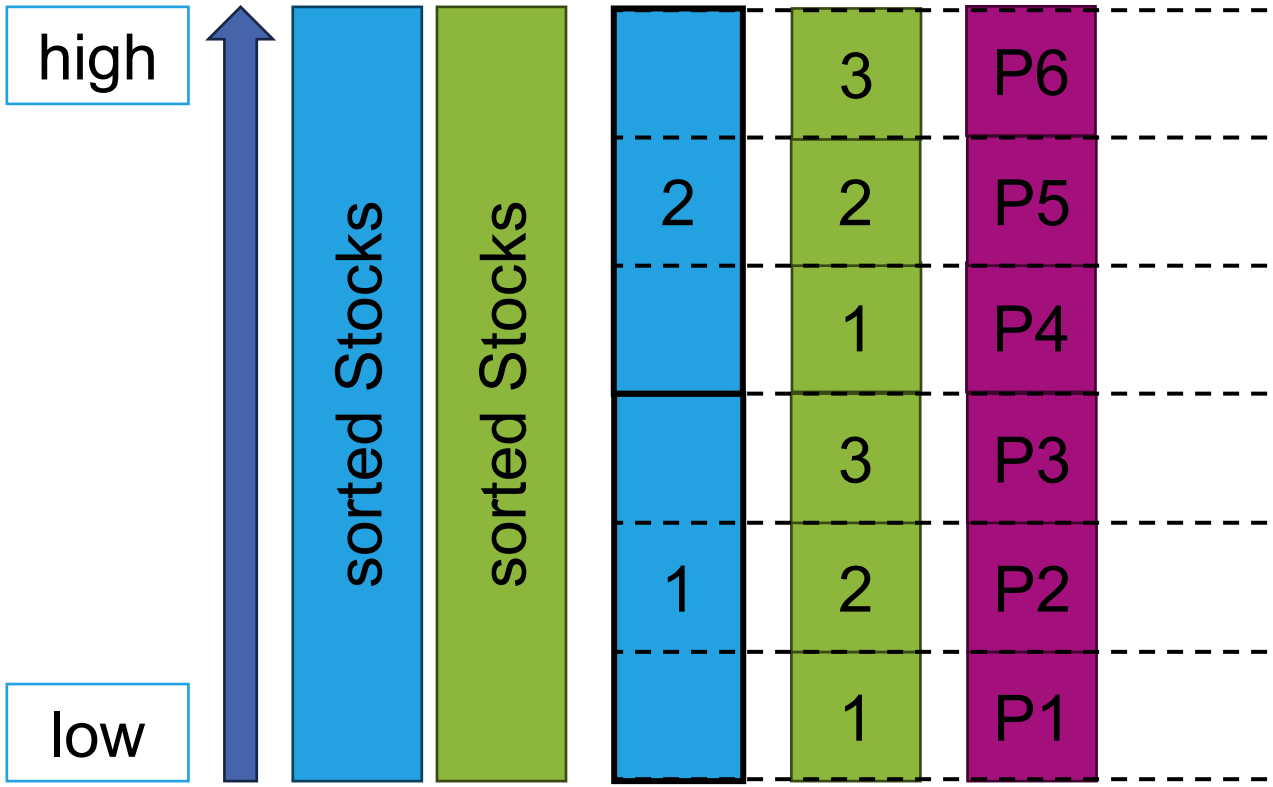
This table presents the number of stocks in each of the 12 portfolios formed by sorting dependently into three β groups and then into four *MktCap* groups. The columns labeled t indicate the year of portfolio formation. The columns labeled β 1, β 2, and β 3 indicate the β group. The rows labeled *MktCap* 1, *MktCap* 2, *MktCap* 3, and *MktCap* 4 indicate the *MktCap* groups.

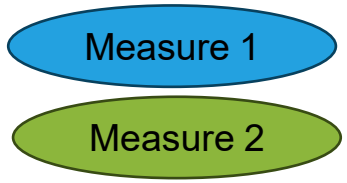
t		β 1	β 2	β 3	t		β 1	β 2	β 3
1988	*MktCap* 1	426	568	426	1998	*MktCap* 1	496	661	495
	MktCap 2	426	569	426		*MktCap* 2	496	660	496
	MktCap 3	427	568	426		*MktCap* 3	496	660	495
	MktCap 4	426	569	427		*MktCap* 4	496	661	495

Bivariate Dependent-Sort Analysis (4/4)

- (3) Average Portfolio Values
  - Identical to independent-sorts
- (4) Inspect Results
  - We’re only interested in the relation between X2 and Y after controlling for X1.
  - Therefore, we will only focus on the difference portfolios for the second sort variable.
  - Otherwise, interpretation is the same.

Bivariate Dependent-Sort Analysis - Visual





This slide depicts the sorting logic for univariate sorting at one time step t

Considerations

- Independent vs. Dependent Sort
  - In most cases both procedures produce qualitatively similar results.
  - As mentioned before: In dependent sorts we examine in the relation between X2 and Y after controlling for X1.
  - For independent sorts, when taking the difference in returns between the high-MktCap portfolio and the low-MktCap portfolio, we are comparing average returns for stocks with **unconditionally** high values of MktCap and stocks with **unconditionally** low values of MktCap, among stocks with low values of β.
- Trivariate Sort Analysis and other constellations exist, but are uncommon.

Python Introduction (1/12)

```python
# Import libraries
import os
import time
import pickle
import multiprocessing as mp
import pandas as pd
import numpy as np
import statsmodels.api as sm
from loguru import logger
from statsmodels.regression.linear_model import OLS
from statsmodels.tools.tools import add_constant
```

- Import relevant libraries
- “os” to access directories and files
- “time” to log timestamps
- “pickle” for storing intermediary results
- “multiprocessing” for parallelization
- “pandas” and “numpy” to manipulate data
- “statsmodels” for statistical tests of results
- “loguru” to enable logging from parallelized processes

Python Introduction (2/12)

- Define script parameters

```
# Directories directly relevant to this script
baseFolder = '/mnt/beegfs/phd_projects/aw/courseMaterialPortfolioSorts' # Project folder, relevant data is stored here
logFilePath = baseFolder + '/logfilePortfolioSortsPipeline.log' # The log file goes here
resultsPath = baseFolder + '/solution' # Solutions go here

# Parameters for parallelization (warning: check RAM load on your machine)
num_processes = 50 # Number of processes that are run in parallel
chunk_size = 6 # number of objects that each process works on

# Other important parameters
startDate = pd.to_datetime('01.01.2005', format='%d.%m.%Y') # start date for the procedure
endDate = pd.to_datetime('01.01.2023', format='%d.%m.%Y') # end date for the procedure
sortingType = "BivariateIndependent" # alternatively "BivariateDependent" or "BivariateIndependent"
marketCapColumn = "MCap" # name of the column that contains the market capitalization
sortVariable1 = "bakshi_mu_tau_30" # name of the first sorting variable
sortVariable2 = "bakshiSkew_tau_30" # name of the second sorting variable
outcomeVariable = "return_month_ahead_excess" # name of the outcome variable
sortPercentilesVar1 = [30, 70] # percentiles for the split of sort variable 1
sortPercentilesVar2 = [30, 70] # percentiles for the split of sort variable 2
weightType = "MarketCapWeighted" # weighting scheme within the portfolios, alternatively "EquallyWeighted"
filterPortfolios = None # Optional parameter for filtering out sorted portfolios by index
backtrackingDays = 0 # Variable to handle the number of days of backtracking for portfolio sorts, values other than 0 are
lags = 6 # the number of lags for the Newey-West adjustment for the statistical tests
testmode = True # if test mode is set to true, the sorting of portfolios will only be applied to a small number of dates
log_results = True # Variable to control whether the results of the portfolio sorting are printed in the log
```

Python Introduction (3/12)

- Preprocessing
- Filter out relevant risk free rates

```
### A - Risk Free Rates ###
# Filter for daystomaturity = 30 and drop unnecessary columns
riskFreeRatesData = riskFreeRatesData[riskFreeRatesData['daystomaturity'] == 30]
riskFreeRatesData = riskFreeRatesData.drop(columns=['yld_pct_annual', 'yld_pct_daily'])
```

- Calculate adjusted close prices and monthly returns

```
# Adjusted Close Price Calculation
pricesData['adjustedclose'] = (pricesData['closeprice'] * pricesData['adjustmentfactor2']) / pricesData.groupby('securityid')['adjustmentfactor2'].transform('last')

# Monthly Discrete Returns (20-trading-day rolling return)
pricesData['return_month'] = pricesData.groupby('securityid')['adjustedclose'].pct_change(periods=20)
```

- $close_{adj,i,t} = close_{unajd,i,t} \frac{adjfactor_{i,t}}{adjfactor_{i,T}}$

Python Introduction (4/12)

- Preprocessing cont.
- Calculate excess returns

```
# Calculate Monthly Excess Return
pricesData['return_month_excess'] = pricesData['return_month'] - pricesData['yld_pct_monthly']
```

- Shift monthly returns back to get 1 month ahead excess return

```
# Offset Monthly Excess Return by 20 trading days into the past
pricesData['return_month_ahead_excess'] = pricesData.groupby('securityid')['return_month_excess'].shift(-20)
```

- Merge all relevant input data into one file!
- Determine the first available date for each month, the dates to sort on

```
# Find the first available date for each month
unique_dates_df['year_month'] = unique_dates_df['loctimestamp'].dt.to_period('M')  # Extract year-month
first_of_month_dates = unique_dates_df.groupby('year_month')['loctimestamp'].min().reset_index(drop=True)
```

Python Introduction (5/12)

- Set up structures to collect results in

```
portfolioWeightsDict = {} # Key: portfolioID, Value: {date: weights}
outcomeResultsDict = {} # Key: portfolioID, Value: {date: averageOutcome}
numberOfStocksDict = {} # Number of stocks per portfolio
breakpointsDict = {} # Breakpoints per date
missingDataDict = {} # Key: date, Value: {errortype: count}
missingRebalancingDatesList = [] # List containing missing rebalancing dates
```

- Setup parallelization: Split list of sorting dates into chunks

```
chunk_size = len(sortingDates) // num_processes + 1 # Divide dates into chunks and process each chunk in a separate process
date_chunks = [sortingDates[i:i + chunk_size] for i in range(0, len(sortingDates), chunk_size)]
results_queue = mp.Queue() # Queue setup for inter-process communication
```

```
for date_chunk in date_chunks:
    p = mp.Process(
        target=self.processDatesChunk,
        args=(
            instrumentData, date_chunk, entitiesByDates,
            sortingType, marketCapColumn, sortVariable1, sortVariable2, outcomeVariable,
            sortPercentilesVar1, sortPercentilesVar2,
            weightType, filterPortfolios, results_queue, missing_data_queue
        )
    )
    processes.append(p)
    p.start()
```

Pass all parameters to the child processes

Python Introduction (6/12)

- Child process
- Iterate over the sorting dates in the chunk; set up structure to collect information on missing data
- Differentiate between sorting types
- For each instrument, check if all necessary data is available, log error cases and only proceed with that instrument, if all data is available
- Handle the case, when there is no data at all

```
# Iterate over each date and perform sorting
for date in dateChunk:
    workingDate = date
    validDataFound = False
```

```
# Try to find matching data
dateDataframe = instrumentData[ # filter the dataframe in a vectorized manner
    (instrumentData['loctimestamp'] == workingDate) &
    (instrumentData['instrumentid'].isin(constituents))
][['instrumentid', sortVariable1, marketCapColumn, outcomeVariable]]
validDataframe = dateDataframe.dropna(subset=[sortVariable1, marketCapColumn, outcomeVariable])
instrumentDataValid = list(validDataframe.itertuples(index=False, name=None)) # Get instrument data
```

```
if len(sort1_values) == 0:
    logger.error(f"{pid} ERROR Failed to find valid CRAM data for {date}.")
    missingRebalancingDates.append(date)
    continue # skip over to the next date
```

Python Introduction (7/12)

- Calculate breakpoints

```
breakpoints = np.percentile(sort1_values, sortPercentilesVar1) # Step 2: Calculate breakpoints
breakpoints = [float("-inf")] + list(breakpoints) + [float("inf")]
portfolios = {i + 1: [] for i in range(len(breakpoints) - 1)} # Initialize dictionary structure to put portfolios in
```

- $B_{k,t} = Pctl_{pk}(\{X_t\})$

- Sort instruments into portfolios

```
for instrumentID, sort1Value, marketCap, outcomeValue in instrumentDataValid: # Step 3: Sort
    for i in range(len(breakpoints) - 1):
        if breakpoints[i] <= sort1Value <= breakpoints[i + 1]:
            portfolios[i + 1].append((instrumentID, sort1Value, marketCap, outcomeValue))
            # Notice: No break of the loop, the instrument can be in multiple portfolios
```

- $P_{k,t} = \{i | B_{k-1,t} \leq X_{i,t} \leq B_{k,t}\}$ for $k \in \{1, 2, \ldots, n_p - 1\}$

Python Introduction (8/12)

- Calculate portfolio weights (optional) and average outcome variable; differentiate between weighting schemes

```python
# Step 5: Calculate Weights & average outcome variable
portfolio_weights = {}
portfolio_outcome = {}
for portfolioID, instruments in filtered_portfolios.items():
    if weightType == "EquallyWeighted":
        weight = 1 / len(instruments)
        portfolio_weights[portfolioID] = {ins_id: weight for ins_id, _, _, _ in instruments}
        weighted_sum = sum(weight * outc for _, _, _, outc in instruments)
    elif weightType == "MarketCapWeighted":
        total_mcap = sum(mcap for _, _, mcap, _ in instruments)
        portfolio_weights[portfolioID] = {ins_id: mcap / total_mcap for ins_id, _, mcap, _ in instruments}
        weighted_sum = sum((mcap / total_mcap) * outc for _, _, mcap, outc in instruments)
    portfolio_outcome[portfolioID] = weighted_sum
```

- $\bar{Y}_{k,t} = \frac{\sum_{i \in P_{k,t}} W_{i,t} Y_{i,t}}{\sum_{i \in P_{k,t}} W_{i,t}}$ for $k \in \{1, 2, \ldots, n_p - 1\}$
- For equal weighting: $W_{i,t} = 1 \; \forall i, t$

Python Introduction (9/12)

- Pass results through the queue to the parent process

```
# Pass results to queue
results_queue.put((portfolioResults, breakpointsDict, outcomeResults))
missing_data_queue.put((missingDataDict, missingRebalancingDates))
logger.info(f'Process {pid} done.')
```

- Catch results in the parent process and sort by date

```
# Fetch results from queue
for _ in range(len(date_chunks)): # Process results for each chunk
    portfolioResults, breakpoints, outcomeResults = results_queue.get()
    missingData, missingRebalancingDates = missing_data_queue.get()
```

```
# Collect average outcome variables
for portfolioID, outcomeDict in outcomeResults.items():
    if portfolioID not in outcomeResultsDict:
        outcomeResultsDict[portfolioID] = {}
    for date, outcome in outcomeDict.items():
        outcomeResultsDict[portfolioID][date] = outcome
```

```
for p in processes: # Wait for all processes to finish
    p.join()
```

Python Introduction (10/12)

- Sort results chronologically

```
# Sort outcome results by date ascending for each portfolio
sortedOutcomeResults = {
    portfolioID: dict(sorted(outcomeByDate.items()))
    for portfolioID, outcomeByDate in outcomeResultsDict.items()
    }
```

- Save the results into pkl files

```
# Dictionary of variables to save
variables_to_save = {
    "sortedPortfolioResults": sortedPortfolioResults,
    "sortedOutcomeResults": sortedOutcomeResults,
    "numberOfStocks": numberOfStocks,
    "breakpointsDict": breakpointsDict,
    "missingDataDict": missingDataDict,
    "missingRebalancingDatesList": missingRebalancingDatesList
}

# Loop through dictionary and save each variable as a .pkl file
for var_name, var_value in variables_to_save.items():
    file_path = os.path.join(resultsPath, f"{var_name}.pkl")
    with open(file_path, "wb") as f:
        pickle.dump(var_value, f)

    logger.info(f"Saved {var_name} to {file_path}")
logger.info('Completed calculation of sorted portfolios.')
```

Python Introduction (10/12)

- Calculate the Diff portfolios

```python
if sortingType == "Univariate":

    # Determine the portfolio names for the highest and lowest portfolios based on sortVariable1
    high_portfolio = f"{sortVariable1}_{len(sortPercentilesVar1) + 1}"  # Highest portfolio
    low_portfolio = f"{sortVariable1}_1"  # Lowest portfolio

    # Check if the high and low portfolios exist in the columns
    if high_portfolio not in outcome_df.columns or low_portfolio not in outcome_df.columns:
        logger.error(f"Portfolio columns {high_portfolio} or {low_portfolio} not found in the DataFrame.")
        return

    # Add the new column: sortVariable1_Diff (difference between high and low portfolio)
    outcome_df[f"{sortVariable1}_Diff"] = outcome_df[high_portfolio] - outcome_df[low_portfolio]
```

- $\bar{Y}_{Diff,t} = \bar{Y}_{n_p,t} - \bar{Y}_{1,t}$

Python Introduction (11/12)

- Save results to table (parquet)

	date	bakshi_mu_tau_30_1	bakshi_mu_tau_30_2	bakshi_mu_tau_30_3	bakshi_mu_tau_30_Diff
0	2005-01-03	-0.062426	-0.006505	0.003624	0.066050
1	2005-02-01	0.014196	0.016229	-0.008394	-0.022590
2	2005-03-01	-0.048409	-0.077814	-0.022491	0.025919
3	2005-04-01	-0.018835	-0.030317	0.019979	0.038814
4	2005-05-02	0.065485	0.039178	0.008646	-0.056839
...	...	...	...	...	...
211	2022-08-01	-0.012281	-0.022829	-0.023869	-0.011588
212	2022-09-01	-0.103917	-0.108302	-0.080283	0.023634
213	2022-10-03	0.023949	0.041456	0.058026	0.034077
214	2022-11-01	0.045912	0.064593	0.054317	0.008404
215	2022-12-01	-0.115730	-0.074699	-0.031089	0.084640

Python Introduction (12/12)

- Calculate average and standard deviation of outcome variable time series
- Test the time series for significance (statsmodels OLS with constant)
- Perform Newey-West adjustment for standard errors (6 lags)

```
# Perform OLS regression with constant (Newey-West adjustment with 6 lags)
model = OLS(timeseries, add_constant(np.ones(len(timeseries)))) # OLS regression with constant
results = model.fit(cov_type='HAC', cov_kwds={'maxlags': lags})
```

- Extract t-stats and p-values from the fitted model
- save results for all portfolios of the measure

	Average	StdErr	t-stat	p-value
bakshi_mu_tau_30_1	0.001295	0.005981	0.180802	0.856523
bakshi_mu_tau_30_2	0.005458	0.003880	1.410424	0.158415
bakshi_mu_tau_30_3	0.006037	0.002748	2.249662	0.024470
bakshi_mu_tau_30_Diff	0.004742	0.004163	0.963918	0.335087

Important Notes for Practical Applications (1/2)

- **Preprocess** all data carefully (you will spend a lot of time on this step!)
  - Filter out everything you don’t need
  - Sensibly organize your data (Folder structure)
- Keep in mind: Loading many big files takes a lot of time and consumes memory, which might stall your machine, I/O kills performance!
- **Modularize** the code and save intermediary results (.parquet /.pq for raw data, .json or .pkl for more complex data structures), so you don’t have to start all over every time; this makes life = debugging easier
- Implement logging, to always know what’s going on

Important Notes for Practical Applications (2/2)

- Keep in mind, handle and log all edge end error cases: Missing data, missing columns, not enough instruments, …
- Write a **documentation** for and ideally in your code, to make it easier to use for others and your future self
- Use **parallelization** and **vectorization** to very significantly speed up computations for large datasets

Parallelization (1/4)

- Usually when you execute code, the calculations are performed using only one CPU core on your computer
- All modern computers have more than one CPU core that can be utilized; this will drastically speed up your overall processing time; if you do the same calculations on two instead of one core, you will need about half the time
- One can see that therein lies huge potential for bigger projects
- To make use of more than one CPU core, one can parallelize the computational tasks and distribute them to different cores
- This can be done with different python libraries, for example “multiprocessing”

Parallelization (2/4)

- Parallelization makes sense, if you do the **same operation for a large set of objects**, for example stocks, dates, etc.
- Since the algorithm of forming a portfolio is the same each time period t, it makes sense to distribute **chunks of dates** over multiple cores
- Generally, a **parent process** (the process that handles the code file you execute) distributes the calculations to multiple **child processes** (also called workers), which are each associated with one CPU core
- The child processes place their results into a **queue** which is monitored by the parent process
- When all child processes are done, the parent process assembles all results it got from the queue

Parallelization (3/4)

- The parent process spawns multiple child processes, passing the function (target) that contains the logic for the calculations and the necessary parameters (args)

```
for date_chunk in date_chunks:
    p = mp.Process(
        target=self.processDatesChunk,
        args=(
            instrumentData, date_chunk, entitiesByDates,
            sortingType, marketCapColumn, sortVariable1, sortVariable2, outcomeVariable,
            sortPercentilesVar1, sortPercentilesVar2,
            weightType, filterPortfolios, results_queue, missing_data_queue
        )
    )
    processes.append(p)
    p.start()
```

- The parent process also fetches the results from the queues, where each child process sends its results

```
# Fetch results from queue
for _ in range(len(date_chunks)): # Process results for each chunk
    portfolioResults, breakpoints, outcomeResults = results_queue.get()
    missingData, missingRebalancingDates = missing_data_queue.get()
```

Parallelization (4/4)

- Make sure to monitor hardware load, CPU and especially RAM while performing distributed calculations to avoid overloading your machine; you can do this using the task manager on windows, the htop command on linux or using other python libraries
- When first setting up parallelized calculations, go slow, step by step and be prepared to kill all your processes, if necessary, before your machine becomes unresponsive; this can also be done using the task manager or a kill command in your command line/shell

Vectorization (1/2)

- Many datasets, such as financial data, is naturally represented in 2D structures (tables/matrices) with rows and columns
- A common (but very bad) method is to iterate over rows using for-loops, for example, to filter for rows, where a column's value meets a condition
- This is extremely slow in high-level languages like python and scales poorly with larger datasets

```python
df = pd.DataFrame({
    "name": ["Alice", "Bob", "Charlie", "David", "Eva"],
    "age": [25, 40, 35, 22, 45],
    "income": [48000, 55000, 60000, 39000, 51000]
})
```

```python
filtered_rows = []
for _, row in df.iterrows():
    if row["age"] > 30 and row["income"] > 50000:
        filtered_rows.append(row)

filtered_df = pd.DataFrame(filtered_rows)
```

Vectorization (2/2)

- Instead use vectorization, it’s faster and cleaner!
- Vectorization operates on entire arrays (e.g. columns) at once without an explicit loop
- This can be done with libraries like NumPy, pandas, TensorFlow, etc.
- For large (financial) datasets **the gain is very substantial**

```
filtered_df = df[(df["age"] > 30) & (df["income"] > 50000)]
```

Questions?

Reach out to

- alexander.walter@partner.kit.edu or
- alexander.walter.1@web.de