Value and Momentum Portfolio Construction

This code will create portfolios based on E/P and create portfolios based on momentum. We will then walk through various ways to combine these factors into a single portfolio.

As usual, we import Pandas and Numpy. I'm also going to use a nice function called MonthEnd, which I will explain below.

```
import pandas as pd
import numpy as np
from pandas.tseries.offsets import MonthEnd
```

Here I'm reading in CRSP and Compustat data. We have used these before. Make sure you have installed pyarrow to read the feather format.

```
stocks = pd.read_feather('crsp_monthly_stocks.feather')
cstat  = pd.read_feather('compustat_annual.feather')
```

+ Code + Text

Let's take a look at the stocks dataframe first. This is data from CRSP, which contains stock returns (RET), closing prices (PRC), volume (VOL), shares outstanding (SHROUT), a code describing the issue type (SHRCD), a code for the primary exchange (EXCHCD), and an industry code (SICCD).

Firms are identified by PERMNO, which remains constant over a firm's life. Data are monthly, and the date is equal to the last trading day of the month.

```
stocks.head(10)
```

	PERMNO	DATE	SHRCD	EXCHCD	SICCD	PRC	VOL	RET	SPREAD	RETX	SHROUT
0	10000.0	1986-01-31	10.0	3.0	3990.0	-4.375000	1771.0	NaN	0.25000	NaN	3680.0
1	10000.0	1986-02-28	10.0	3.0	3990.0	-3.250000	828.0	-0.257143	0.25000	-0.257143	3680.0
2	10000.0	1986-03-31	10.0	3.0	3990.0	-4.437500	1078.0	0.365385	0.12500	0.365385	3680.0
3	10000.0	1986-04-30	10.0	3.0	3990.0	-4.000000	957.0	-0.098592	0.25000	-0.098592	3793.0
4	10000.0	1986-05-30	10.0	3.0	3990.0	-3.109375	1074.0	-0.222656	0.09375	-0.222656	3793.0
5	10000.0	1986-06-30	10.0	3.0	3990.0	-3.093750	1069.0	-0.005025	0.06250	-0.005025	3793.0
6	10000.0	1986-07-31	10.0	3.0	3990.0	-2.843750	1163.0	-0.080808	0.06250	-0.080808	3793.0
7	10000.0	1986-08-29	10.0	3.0	3990.0	-1.093750	3049.0	-0.615385	0.06250	-0.615385	3793.0
8	10000.0	1986-09-30	10.0	3.0	3990.0	-1.031250	3551.0	-0.057143	0.06250	-0.057143	3793.0
9	10000.0	1986-10-31	10.0	3.0	3990.0	-0.781250	1903.0	-0.242424	0.06250	-0.242424	3843.0

We're going to clean up the data a bit. The code below does the following:

1. Shift the date so that it is always the last day of the month, rather than the last trading day. This will make it easier to merge in with other datasets.
2. Take the absolute value of the closing price. For shares that don't trade, CRSP sets the price equal to the closing bid-ask midpoint, but it makes the price negative as a warning about this.
3. Define market value (MV) as the product of shares outstanding and closing price.
4. Drop shares outstanding, which we won't use again, and the share code. We use the share code when we download data from WRDS. Selecting share codes of 10 or 11 means that we will be downloading common equity and not other securities (ETFs, REITS, etc.). Also drop EXCHCD, SICCD, PRC, and VOL to make the dataframe easier to display.
5. Set the index to PERMNO/DATE.
6. Sort by the index.
7. Look at the dataframe.

```
stocks['DATE'] = stocks['DATE'] + MonthEnd(0)
stocks['PRC']  = np.abs(stocks['PRC'])
stocks['MV'] = stocks['SHROUT']*stocks['PRC']
```

```
stocks.drop(['SHROUT','SHRCD','EXCHCD','SICCD','PRC','VOL'], axis=1, inplace=True)
stocks.set_index(['PERMNO','DATE'], inplace=True)
stocks.sort_index(inplace=True)
stocks.head()
```

		RET	SPREAD	RETX	MV
PERMNO	**DATE**				
10000.0	**1986-01-31**	NaN	0.25000	NaN	16100.000000
	1986-02-28	-0.257143	0.25000	-0.257143	11960.000000
	1986-03-31	0.365385	0.12500	0.365385	16330.000000
	1986-04-30	-0.098592	0.25000	-0.098592	15172.000000
	1986-05-31	-0.222656	0.09375	-0.222656	11793.859375

Looks good. Now let's look at the Compustat data. This is annual. It contains a variety of variables that are described in compustat_variables.xlsx. The one we will use here is earnings before extraordinary items (IB). For one small purpose I will also look at book equity (SEQ).

Since I used the CRSP/Compustat merged version of Compustat data, I have a variable LPERMNO that is equivalent to the PERMNO variable in CRSP. Dates in this file represent the last date of the fiscal year. They are *not* the dates at which the data became public.

```
cstat.head(5)
```

	DATADATE	FYEAR	LPERMNO	AT	CEQ	CHE	LT	PSTK	SEQ	DVT	IB	SALE	CAPX
0	1970-12-31	1970.0	25881.0	33.450	10.544	1.660	22.906	0.000	10.544	0.000	1.878	45.335	2.767
1	1971-12-31	1971.0	25881.0	29.330	8.381	2.557	20.948	0.000	8.382	0.000	0.138	47.033	1.771
2	1972-12-31	1972.0	25881.0	19.907	7.021	2.027	12.886	0.000	7.021	0.000	1.554	34.362	1.254
3	1973-12-31	1973.0	25881.0	21.771	8.567	1.357	13.204	0.000	8.567	0.000	1.863	37.750	1.633
4	1974-12-31	1974.0	25881.0	25.638	9.843	1.338	15.381	0.414	10.257	0.021	1.555	50.325	1.313

Let's rename LPERMNO to PERMNO:

```
cstat.rename(columns={"LPERMNO":"PERMNO"}, inplace=True)
cstat.head(5)
```

	DATADATE	FYEAR	PERMNO	AT	CEQ	CHE	LT	PSTK	SEQ	DVT	IB	SALE	CAPX
0	1970-12-31	1970.0	25881.0	33.450	10.544	1.660	22.906	0.000	10.544	0.000	1.878	45.335	2.767
1	1971-12-31	1971.0	25881.0	29.330	8.381	2.557	20.948	0.000	8.382	0.000	0.138	47.033	1.771
2	1972-12-31	1972.0	25881.0	19.907	7.021	2.027	12.886	0.000	7.021	0.000	1.554	34.362	1.254
3	1973-12-31	1973.0	25881.0	21.771	8.567	1.357	13.204	0.000	8.567	0.000	1.863	37.750	1.633
4	1974-12-31	1974.0	25881.0	25.638	9.843	1.338	15.381	0.414	10.257	0.021	1.555	50.325	1.313

To use this data, we must make an assumption about the first date on which this data would be available. Standard practice is to assume that by 6 months after the fiscal year end we will for sure have access to the annual report.

I therefore create a new column, DATE, that is designed to represent the date that the data become known. In the first line I set date equal to the fiscal year end plus six months. In the second line I make sure that the date is the last day of the month. As before, this will help when I merge this with the other datasets.

```
cstat['DATE'] = cstat['DATADATE'] + MonthEnd(0)
cstat.head()
```

	DATADATE	FYEAR	PERMNO	AT	CEQ	CHE	LT	PSTK	SEQ	DVT	IB	SALE	CAPX	DATE
0	1970-12-31	1970.0	25881.0	33.450	10.544	1.660	22.906	0.000	10.544	0.000	1.878	45.335	2.767	1970-12-31
1	1971-12-31	1971.0	25881.0	29.330	8.381	2.557	20.948	0.000	8.382	0.000	0.138	47.033	1.771	1971-12-31
2	1972-12-31	1972.0	25881.0	19.907	7.021	2.027	12.886	0.000	7.021	0.000	1.554	34.362	1.254	1972-12-31
3	1973-12-31	1973.0	25881.0	21.771	8.567	1.357	13.204	0.000	8.567	0.000	1.863	37.750	1.633	1973-12-31
4	1974-12-31	1974.0	25881.0	25.638	9.843	1.338	15.381	0.414	10.257	0.021	1.555	50.325	1.313	1974-12-31

With the date defined how I want it, we are ready to set indexes and sort:

```
cstat.set_index(['PERMNO','DATE'], inplace=True)
cstat.sort_index(inplace=True)
cstat.head()
```

PERMNO	DATE	DATADATE	FYEAR	AT	CEQ	CHE	LT	PSTK	SEQ	DVT	IB	SALE	CAPX
10000.0	**1986-10-31**	1986-10-31	1986.0	2.115	0.418	0.348	1.697	0.0	0.418	0.000	-0.730	1.026	0.240
10001.0	**1986-06-30**	1986-06-30	1986.0	12.242	5.432	0.746	6.810	0.0	5.432	0.365	0.669	21.460	0.551
	1987-06-30	1987-06-30	1987.0	11.771	5.369	0.729	6.402	0.0	5.369	0.416	0.312	16.621	0.513
	1988-06-30	1988-06-30	1988.0	11.735	5.512	0.744	6.223	0.0	5.512	0.427	0.542	16.978	0.240
	1989-06-30	1989-06-30	1989.0	18.565	6.321	1.177	12.244	0.0	6.321	0.459	1.208	22.910	0.444

We now need to merge these data. Unfortunately, the data occasionally have multiple rows with the same PERMNO and DATE. So we are going to have to eliminate duplicate PERMNO/DATE pairs.

There are many ways to do this. My thought here is that we should assume that if there is more than one PERMNO on the same date, then the bigger one is probably more important and therefore more likely to be correct.

I am therefore going to sort the dataframe in ascending order by PERMNO, then in ascending order by DATE, and then in descending order by size (either MV or SEQ). This is how you do it:

```
stocks = stocks.sort_values(by = ['PERMNO','DATE','MV'], ascending = [True, True, False])
cstat  = cstat.sort_values(by = ['PERMNO','DATE','SEQ'], ascending = [True, True, False])
```

Since the first observation for each PERMNO/DATE is the one I want to keep, I eliminate duplicates as follows:

```
stocks = stocks.groupby(['PERMNO','DATE']).head(1)
cstat  = cstat.groupby(['PERMNO','DATE']).head(1)
```

Let's create momentum here. I will first lag returns two months, and then I will compute the 11-month moving average of the lagged returns. This will create the version of momentum that skips a month between the formation period and the holding period.

Note the use of droplevel in the second step.

```
stocks['lag2 RET'] = stocks['RET'].groupby('PERMNO').shift(2)
stocks['momentum'] = stocks['lag2 RET'].groupby('PERMNO').rolling(11).mean().droplevel(0)
```

In the previous step, you will notice I used the droplevel method. This is because of some unexpected behavior by groupby. If I don't use droplevel, I get the following result:

```
stocks['lag2 RET'].groupby('PERMNO').rolling(11).mean()
```

```
PERMNO   PERMNO   DATE
10000.0  10000.0  1986-01-31         NaN
                  1986-02-28         NaN
                  1986-03-31         NaN
                  1986-04-30         NaN
                  1986-05-31         NaN
                                      ...
93436.0  93436.0  2023-02-28   -0.064176
                  2023-03-31   -0.020812
```

```
                    2023-04-30   -0.025397
                    2023-05-31   -0.007174
                    2023-06-30   -0.014338
Name: lag2 RET, Length: 3452888, dtype: float64
```

For some reason, groupby creates a redundant PERMNO index. This is eliminated by droplevel(0), which drops the first (which is 0) level of the index.

Now it's time to combine the CRSP and Compustat data. Because of the way we have indexed each dataframe, specifically making sure that all dates are the last days of the month, this is super easy. Just type:

stocks[['IB','SEQ']] = cstat[['IB','SEQ']]

However, this is a bit slow. A much faster alternative is to use the merge method:

```
stocks = stocks.merge(cstat[['IB','SEQ']], how='left', on=['PERMNO','DATE'])
```

To see what happened, let's take a look at one stock, Apple, which has PERMNO=14593:

```
stocks.loc[14593].tail(24)
```

DATE	RET	SPREAD	RETX	MV	lag2 RET	momentum	IB	SEQ
2021-07-31	0.064982	NaN	0.064982	2.411090e+09	-0.050434	0.034278	NaN	NaN
2021-08-31	0.042438	NaN	0.040930	2.509775e+09	0.099109	0.028276	NaN	NaN
2021-09-30	-0.068037	NaN	-0.068037	2.324390e+09	0.064982	0.014519	94680.0	63090.0
2021-10-31	0.058657	NaN	0.058657	2.457678e+09	0.042438	0.027697	NaN	NaN
2021-11-30	0.104940	NaN	0.103471	2.711977e+09	-0.068037	0.026968	NaN	NaN
2021-12-31	0.074229	NaN	0.074229	2.902368e+09	0.058657	0.023620	NaN	NaN
2022-01-31	-0.015712	NaN	-0.015712	2.852312e+09	0.104940	0.022744	NaN	NaN
2022-02-28	-0.054011	NaN	-0.055270	2.694666e+09	0.074229	0.029992	NaN	NaN
2022-03-31	0.057473	NaN	0.057473	2.830003e+09	-0.015712	0.035794	NaN	NaN
2022-04-30	-0.097131	NaN	-0.097131	2.551594e+09	-0.054011	0.030216	NaN	NaN
2022-05-31	-0.054424	NaN	-0.055883	2.409002e+09	0.057473	0.028512	NaN	NaN
2022-06-30	-0.081430	NaN	-0.081430	2.200560e+09	-0.097131	0.024267	NaN	NaN
2022-07-31	0.188634	NaN	0.188634	2.611658e+09	-0.054424	0.010310	NaN	NaN
2022-08-31	-0.031137	NaN	-0.032552	2.526644e+09	-0.081430	-0.003001	NaN	NaN
2022-09-30	-0.120977	NaN	-0.120977	2.203381e+09	0.188634	0.010290	99803.0	50672.0
2022-10-31	0.109551	NaN	0.109551	2.439351e+09	-0.031137	0.013644	NaN	NaN
2022-11-30	-0.033129	NaN	-0.034629	2.354879e+09	-0.120977	-0.002686	NaN	NaN
2022-12-31	-0.122273	NaN	-0.122273	2.058404e+09	0.109551	-0.002267	NaN	NaN
2023-01-31	0.110521	NaN	0.110521	2.282948e+09	-0.033129	-0.012027	NaN	NaN
2023-02-28	0.023217	NaN	0.021623	2.332313e+09	-0.122273	-0.021714	NaN	NaN
2023-03-31	0.118649	NaN	0.118649	2.592790e+09	0.110521	-0.006756	NaN	NaN
2023-04-30	0.028987	NaN	0.028987	2.668846e+09	0.023217	-0.009871	NaN	NaN
2023-05-31	0.046028	NaN	0.044613	2.787912e+09	0.118649	0.009746	NaN	NaN
2023-06-30	0.094330	NaN	0.094330	3.050896e+09	0.028987	0.017329	NaN	NaN

Apple's fiscal year ends in September. That's why we see IB in those months and no others.

Now we will compute the E/P ratio in two different ways.

The first is to use the most recent known value of E (column IB) and divide it by the contemporaneous observation of P (column MV), meaning the value of MV that corresponds to the most recent fiscal year end. We will then lag the ratio 6 months to account for the fact that earnings are not known for some time after the end of the quarter.

We will need to multiply the E/P ratio by 1000 for it to make sense. The reason is that CRSP and Compustat are in different units. In CRSP, the shares outstanding series used to create market values (MV) was in 1000s of shares. Thus, the MV column is too small by a factor of 1000. In Compustat, earnings (IB) are in millions of dollars. Multiplying by 1000 makes these numbers comparable.

```
stocks['lag EP v1'] = stocks['IB'].groupby('PERMNO').shift(6) / stocks['MV'].groupby('PERMNO').shift(6) * 1000
```

Looking again at Apple, we can see what we have done:

```
stocks.loc[14593].tail(24)
```

DATE	RET	MV	lag2 RET	momentum	IB	SEQ	lag EP v1
2020-07-31	0.165132	1.817316e+09	0.084956	0.048177	NaN	NaN	NaN
2020-08-31	0.216309	2.206911e+09	0.147386	0.054631	NaN	NaN	NaN
2020-09-30	-0.102526	1.966079e+09	0.165132	0.071149	57411.0	65339.0	NaN
2020-10-31	-0.060012	1.850816e+09	0.216309	0.084181	NaN	NaN	NaN
2020-11-30	0.095490	2.024065e+09	-0.102526	0.064798	NaN	NaN	NaN
2020-12-31	0.114574	2.232279e+09	-0.060012	0.052304	NaN	NaN	NaN
2021-01-31	-0.005502	2.215357e+09	0.095490	0.052004	NaN	NaN	NaN
2021-02-28	-0.079532	2.035725e+09	0.114574	0.057510	NaN	NaN	NaN
2021-03-31	0.007340	2.038232e+09	-0.005502	0.067402	NaN	NaN	0.029201
2021-04-30	0.076218	2.193756e+09	-0.079532	0.066513	NaN	NaN	NaN
2021-05-31	-0.050434	2.079446e+09	0.007340	0.053056	NaN	NaN	NaN
2021-06-30	0.099109	2.267639e+09	0.076218	0.052261	NaN	NaN	NaN
2021-07-31	0.064982	2.411090e+09	-0.050434	0.034278	NaN	NaN	NaN
2021-08-31	0.042438	2.509775e+09	0.099109	0.028276	NaN	NaN	NaN
2021-09-30	-0.068037	2.324390e+09	0.064982	0.014519	94680.0	63090.0	NaN
2021-10-31	0.058657	2.457678e+09	0.042438	0.027697	NaN	NaN	NaN
2021-11-30	0.104940	2.711977e+09	-0.068037	0.026968	NaN	NaN	NaN
2021-12-31	0.074229	2.902368e+09	0.058657	0.023620	NaN	NaN	NaN
2022-01-31	-0.015712	2.852312e+09	0.104940	0.022744	NaN	NaN	NaN
2022-02-28	-0.054011	2.694666e+09	0.074229	0.029992	NaN	NaN	NaN
2022-03-31	0.057473	2.830003e+09	-0.015712	0.035794	NaN	NaN	0.040733
2022-04-30	-0.097131	2.551594e+09	-0.054011	0.030216	NaN	NaN	NaN
2022-05-31	-0.054424	2.409002e+09	0.057473	0.028512	NaN	NaN	NaN
2022-06-30	-0.081430	2.212838e+09	-0.097131	0.024267	NaN	NaN	NaN

The calculations look right, but they only result in one E/P ratio per year. We will fill in the rest using the *fillna* method with the *pad* option. This uses older data to fill in for missing values. Because older data are used, we don't have to worry about look-ahead bias. The groupby('PERMNO') step makes sure that we never fill in one firm's earnings with those of another firm. The limit=15 option says that we will not use a prior value if it is more than 15 months old, which should be unusual situations.

```
stocks['lag EP v1']  = stocks['lag EP v1'].groupby('PERMNO').fillna(method='pad', limit=15)
```

The second approach will be to use the most recent known E divided by the most recent known P, even if these two variables are observed at very different times.

To start with this, lets compute the most recent E (column IB) that we would observe. We'll then use fillna in the same way to fill in missing values with older data.

```
stocks['lag IB']  = stocks['IB'].groupby('PERMNO').shift(6).fillna(method='pad', limit=15)
```

Again taking a look at Apple, it seems to have worked as expected:

```
stocks.loc[14593].tail(24)
```

DATE	RET	SPREAD	RETX	MV	lag2 RET	momentum	IB	SEQ	lag EP v1	lag IB
2021-07-31	0.064982	NaN	0.064982	2.411090e+09	-0.050434	0.034278	NaN	NaN	0.029201	57411.0
2021-08-31	0.042438	NaN	0.040930	2.509775e+09	0.099109	0.028276	NaN	NaN	0.029201	57411.0
2021-09-30	-0.068037	NaN	-0.068037	2.324390e+09	0.064982	0.014519	94680.0	63090.0	0.029201	57411.0
2021-10-31	0.058657	NaN	0.058657	2.457678e+09	0.042438	0.027697	NaN	NaN	0.029201	57411.0
2021-11-30	0.104940	NaN	0.103471	2.711977e+09	-0.068037	0.026968	NaN	NaN	0.029201	57411.0
2021-12-31	0.074229	NaN	0.074229	2.902368e+09	0.058657	0.023620	NaN	NaN	0.029201	57411.0
2022-01-31	-0.015712	NaN	-0.015712	2.852312e+09	0.104940	0.022744	NaN	NaN	0.029201	57411.0
2022-02-28	-0.054011	NaN	-0.055270	2.694666e+09	0.074229	0.029992	NaN	NaN	0.029201	57411.0
2022-03-31	0.057473	NaN	0.057473	2.830003e+09	-0.015712	0.035794	NaN	NaN	0.040733	94680.0
2022-04-30	-0.097131	NaN	-0.097131	2.551594e+09	-0.054011	0.030216	NaN	NaN	0.040733	94680.0
2022-05-31	-0.054424	NaN	-0.055883	2.409002e+09	0.057473	0.028512	NaN	NaN	0.040733	94680.0
2022-06-30	-0.081430	NaN	-0.081430	2.200560e+09	-0.097131	0.024267	NaN	NaN	0.040733	94680.0
2022-07-31	0.188634	NaN	0.188634	2.611658e+09	-0.054424	0.010310	NaN	NaN	0.040733	94680.0
2022-08-31	-0.031137	NaN	-0.032552	2.526644e+09	-0.081430	-0.003001	NaN	NaN	0.040733	94680.0
2022-09-30	-0.120977	NaN	-0.120977	2.203381e+09	0.188634	0.010290	99803.0	50672.0	0.040733	94680.0
2022-10-31	0.109551	NaN	0.109551	2.439351e+09	-0.031137	0.013644	NaN	NaN	0.040733	94680.0
2022-11-30	-0.033129	NaN	-0.034629	2.354879e+09	-0.120977	-0.002686	NaN	NaN	0.040733	94680.0
2022-12-31	-0.122273	NaN	-0.122273	2.058404e+09	0.109551	-0.002267	NaN	NaN	0.040733	94680.0
2023-01-31	0.110521	NaN	0.110521	2.282948e+09	-0.033129	-0.012027	NaN	NaN	0.040733	94680.0
2023-02-28	0.023217	NaN	0.021623	2.332313e+09	-0.122273	-0.021714	NaN	NaN	0.040733	94680.0
2023-03-31	0.118649	NaN	0.118649	2.592790e+09	0.110521	-0.006756	NaN	NaN	0.045295	99803.0
2023-04-30	0.028987	NaN	0.028987	2.668846e+09	0.023217	-0.009871	NaN	NaN	0.045295	99803.0
2023-05-31	0.046028	NaN	0.044613	2.787912e+09	0.118649	0.009746	NaN	NaN	0.045295	99803.0
2023-06-30	0.094330	NaN	0.094330	3.050896e+09	0.028987	0.017329	NaN	NaN	0.045295	99803.0

To compute the E/P ratio in this approach, we divide the lagged IB column by the most recently observed value of MV, which is the value in the previous row:

```
stocks['lag EP v2'] = stocks['lag IB'] / stocks['MV'].groupby('PERMNO').shift(1) * 1000
```

Again, Apple:

```
stocks.loc[14593].tail(24)
```

DATE	RET	SPREAD	RETX	MV	lag2 RET	momentum	IB	SEQ	lag EP v1	lag IB	lag EP v2
2021-07-31	0.064982	NaN	0.064982	2.411090e+09	-0.050434	0.034278	NaN	NaN	0.029201	57411.0	0.025318
2021-08-31	0.042438	NaN	0.040930	2.509775e+09	0.099109	0.028276	NaN	NaN	0.029201	57411.0	0.023811
2021-09-30	-0.068037	NaN	-0.068037	2.324390e+09	0.064982	0.014519	94680.0	63090.0	0.029201	57411.0	0.022875
2021-10-31	0.058657	NaN	0.058657	2.457678e+09	0.042438	0.027697	NaN	NaN	0.029201	57411.0	0.024699
2021-11-30	0.104940	NaN	0.103471	2.711977e+09	-0.068037	0.026968	NaN	NaN	0.029201	57411.0	0.023360
2021-12-31	0.074229	NaN	0.074229	2.902368e+09	0.058657	0.023620	NaN	NaN	0.029201	57411.0	0.021169
2022-01-31	-0.015712	NaN	-0.015712	2.852312e+09	0.104940	0.022744	NaN	NaN	0.029201	57411.0	0.019781
2022-02-28	-0.054011	NaN	-0.055270	2.694666e+09	0.074229	0.029992	NaN	NaN	0.029201	57411.0	0.020128
2022-03-31	0.057473	NaN	0.057473	2.830003e+09	-0.015712	0.035794	NaN	NaN	0.040733	94680.0	0.035136
2022-04-30	-0.097131	NaN	-0.097131	2.551594e+09	-0.054011	0.030216	NaN	NaN	0.040733	94680.0	0.033456
2022-05-31	-0.054424	NaN	-0.055883	2.409002e+09	0.057473	0.028512	NaN	NaN	0.040733	94680.0	0.037106
2022-06-30	-0.081430	NaN	-0.081430	2.200560e+09	-0.097131	0.024267	NaN	NaN	0.040733	94680.0	0.039303
2022-07-31	0.188634	NaN	0.188634	2.611658e+09	-0.054424	0.010310	NaN	NaN	0.040733	94680.0	0.043025
2022-08-31	-0.031137	NaN	-0.032552	2.526644e+09	-0.081430	-0.003001	NaN	NaN	0.040733	94680.0	0.036253
2022-09-30	-0.120977	NaN	-0.120977	2.203381e+09	0.188634	0.010290	99803.0	50672.0	0.040733	94680.0	0.037473
2022-10-31	0.109551	NaN	0.109551	2.439351e+09	-0.031137	0.013644	NaN	NaN	0.040733	94680.0	0.042970
2022-11-30	-0.033129	NaN	-0.034629	2.354879e+09	-0.120977	-0.002686	NaN	NaN	0.040733	94680.0	0.038814
2022-12-31	-0.122273	NaN	-0.122273	2.058404e+09	0.109551	-0.002267	NaN	NaN	0.040733	94680.0	0.040206
2023-01-31	0.110521	NaN	0.110521	2.282948e+09	-0.033129	-0.012027	NaN	NaN	0.040733	94680.0	0.045997
2023-02-28	0.023217	NaN	0.021623	2.332313e+09	-0.122273	-0.021714	NaN	NaN	0.040733	94680.0	0.041473
2023-03-31	0.118649	NaN	0.118649	2.592790e+09	0.110521	-0.006756	NaN	NaN	0.045295	99803.0	0.042791
2023-04-30	0.028987	NaN	0.028987	2.668846e+09	0.023217	-0.009871	NaN	NaN	0.045295	99803.0	0.038493
2023-05-31	0.046028	NaN	0.044613	2.787912e+09	0.118649	0.009746	NaN	NaN	0.045295	99803.0	0.037396
2023-06-30	0.094330	NaN	0.094330	3.050896e+09	0.028987	0.017329	NaN	NaN	0.045295	99803.0	0.035798

Note that the two E/P ratios are not the same.

Our primary analysis will be on the RET, lag EP v1, lag EP v2, and momentum columns. Let's make sure all four variables are observed:

```
stocks = stocks.dropna(subset=['RET','lag EP v1','lag EP v2','momentum'])
```

Since we are going to combine different firms, on the same date, into portfolios, the next step is to sort by DATE first and then by PERMNO.

```
stocks = stocks.reorder_levels(['DATE','PERMNO'])
stocks.sort_index(inplace=True)
```

One final step before portfolios are constructed is to eliminate observations that look funny. These could be the result of earnings being close to zero. They could also be database errors or errors in merging the different databases.

In any case, it is entirely feasible for an investor to say that he or she is not going to hold stocks that have P/E ratios below 5 or above 100, so I will exclude all stocks that are outside that range (using either P/E measure).

I also remove stocks with momentum below -.1 or above .1. These are very extreme values. Removing them has little effect on the results below, though it does improve them slightly.

```
stocks = stocks.loc[(stocks['lag EP v1']>0)  & (stocks['lag EP v2']>0)  & \
                    (stocks['lag EP v1']<.5) & (stocks['lag EP v2']<.5) & \
                    (stocks['momentum']>-.1) & (stocks['momentum']<.1)]
```

Finally, time to compute portfolios. I will compute quintile portfolios using the rank method. This will split the sample into percentile groups based upon the input variable. I can then transform the percentiles into an integer group number.

However, I do NOT want to split the sample into groups across all dates. Rather, I need to, for each date, split the sample into five groups by the P/E measures. I can do this using the "groupby" command.

We're going to look at quintiles formed on two different measures. Rather than type all the code twice, let's create functions that do all the necessary calculations. The first one examines quintile portfolios and returns the HML long/short portfolio. The second computes performance statistics for the HML portfolio returns.

```
# the function takes the name of the sorting variable we want to use as its single input
def perf(sortvar):

    # creating the quintiles
    stocks['Q'] = (stocks[sortvar].groupby('DATE').rank(pct=True)*5 - 0.00001).astype(int) +1

    # computing quintile portfolio returns
    ports = stocks.groupby(['Q','DATE'])['RET'].mean()

    # printing out basic statistics on each portfolio
    print(ports.groupby('Q').describe())

    # computing high minus low portfolios
    hml = ports.loc[5] - ports.loc[1]

    return hml


def stats(hml):
    # printing basic statistics plus sharpe and t-stat
    stats = hml.describe()
    stats.loc['tstat']  = stats.loc['mean'] / stats.loc['std'] * np.sqrt(stats.loc['count'])
    stats.loc['sharpe'] = stats.loc['mean'] / stats.loc['std'] * np.sqrt(12)
    print(stats)
```

Now we just have to call the function twice:

```
hml_ep = perf('lag EP v2')
stats(hml_ep)
```

```
   count      mean       std       min       25%       50%       75%       max
Q
1  726.0  0.009966  0.057195 -0.311634 -0.023591  0.013187  0.045671  0.194511
2  726.0  0.010355  0.051053 -0.290474 -0.017858  0.013183  0.042021  0.242052
3  726.0  0.010938  0.048940 -0.272163 -0.014901  0.014308  0.038728  0.279161
4  726.0  0.013435  0.049738 -0.256460 -0.011694  0.016641  0.040227  0.295669
5  726.0  0.015067  0.059774 -0.308834 -0.014842  0.014627  0.043939  0.366360
count     726.000000
mean        0.005101
std         0.030248
min        -0.140727
25%        -0.011790
50%         0.003667
75%         0.018950
max         0.171849
tstat       4.543539
sharpe      0.584139
Name: RET, dtype: float64
```

```
hml_mom = perf('momentum')
stats(hml_mom)
```

```
   count      mean       std       min       25%       50%       75%       max
Q
1  726.0  0.007959  0.065184 -0.270553 -0.026189  0.007782  0.041152  0.358567
2  726.0  0.010462  0.051163 -0.237515 -0.014231  0.011856  0.036784  0.306753
3  726.0  0.011902  0.047159 -0.266068 -0.012271  0.015384  0.038523  0.254245
4  726.0  0.013239  0.048364 -0.279561 -0.012318  0.016257  0.041824  0.243608
5  726.0  0.016193  0.056987 -0.306080 -0.016374  0.018080  0.052673  0.214873
count     726.000000
mean        0.008234
std         0.037464
min        -0.247366
25%        -0.008493
50%         0.011929
75%         0.029542
max         0.137134
tstat       5.922308
```

```
sharpe      0.761401
Name: RET, dtype: float64
```

Combining Portfolios

One way to combine the value and momentum strategies is just to put half of your money in each long/short portfolio. The performance of this strategy is as follows:

```
stats(.5*hml_ep + .5*hml_mom)
```

```
count     726.000000
mean        0.006668
std         0.014943
min        -0.062660
25%        -0.001072
50%         0.006161
75%         0.014828
max         0.055236
tstat      12.022683
sharpe      1.545695
Name: RET, dtype: float64
```

An alternative way to combine signals is to normalize them and take the sum, i.e.

$$score_{i,t} = \frac{ep_{i,t} - \mu_t^{ep}}{\sigma_t^{ep}} + \frac{mom_{i,t} - \mu_t^{mom}}{\sigma_t^{mom}},$$

where μ_t^x and σ_t^x are the mean and SD of x across all firms at date t.

Note that subtracting the means affects the score, but it does not change the rankings of different stocks. So in the next calculation, I don't bother to remove them.

```
stocks['score'] = stocks['lag EP v2']/stocks['lag EP v2'].groupby('DATE').std() + stocks['momentum']/stocks['momentum'].groupby(
```

Now I form quintile portfolios based on $score$:

```
hml_score = perf('score')
stats(hml_score)
```

```
   count      mean       std       min       25%       50%       75%       max
Q
1  726.0  0.006399  0.060011 -0.292449 -0.027516  0.007847  0.040818  0.276472
2  726.0  0.009597  0.050035 -0.269568 -0.017341  0.011568  0.037587  0.271461
3  726.0  0.012021  0.047994 -0.254787 -0.014350  0.014967  0.040753  0.258152
4  726.0  0.014672  0.049482 -0.261427 -0.011947  0.018249  0.042329  0.283848
5  726.0  0.017065  0.057263 -0.281546 -0.012677  0.018642  0.051218  0.287986
count     726.000000
mean        0.010665
std         0.023797
min        -0.098474
25%        -0.001538
50%         0.010148
75%         0.023695
max         0.094464
tstat      12.075750
sharpe      1.552517
Name: RET, dtype: float64
```

Another possibility is to do a two-way sort. I will do independent quintile sorts on EP and momentum. I will then go long stocks that are in the 5th quintile of both variables and short stocks that are in both 1st quintiles.

```
# creating the quintiles
stocks['Qep']  = (stocks['lag EP v2'].groupby('DATE').rank(pct=True)*5 - 0.00001).astype(int) +
stocks['Qmom'] = (stocks['momentum'].groupby('DATE').rank(pct=True)*5 - 0.00001).astype(int) +1

# computing quintile portfolio returns
ports = stocks.groupby(['Qep','Qmom','DATE'])['RET'].mean()

# printing out basic statistics on each portfolio
print(ports.groupby(['Qep','Qmom']).describe())

# computing high minus low portfolios
hml = ports.loc[5,5] - ports.loc[1,1]
```

```
stats(hml)
```

```
             count      mean       std       min       25%       50%       75%  \
Qep Qmom
1   1        726.0  0.003397  0.069312 -0.327723 -0.035262  0.002538  0.042094
    2        725.0  0.007208  0.058415 -0.293033 -0.025590  0.008583  0.041263
    3        726.0  0.008544  0.054175 -0.307528 -0.023488  0.012407  0.041745
    4        726.0  0.009816  0.054004 -0.299929 -0.020168  0.012894  0.042287
    5        726.0  0.015063  0.061736 -0.317323 -0.020362  0.019007  0.054977
2   1        726.0  0.004644  0.066284 -0.316259 -0.031477  0.004948  0.041096
    2        726.0  0.007933  0.053009 -0.276134 -0.018374  0.008442  0.038619
    3        726.0  0.009823  0.048137 -0.268819 -0.015240  0.013900  0.036712
    4        726.0  0.011400  0.048579 -0.278437 -0.014804  0.013758  0.043399
    5        726.0  0.014947  0.055794 -0.306281 -0.014711  0.017143  0.052116
3   1        726.0  0.006674  0.062558 -0.268635 -0.028083  0.007493  0.039667
    2        726.0  0.008733  0.049503 -0.267926 -0.014909  0.011838  0.033938
    3        726.0  0.010321  0.046252 -0.265387 -0.013531  0.015618  0.037923
    4        726.0  0.013177  0.047334 -0.274877 -0.012461  0.016816  0.041965
    5        726.0  0.015898  0.056107 -0.284165 -0.013916  0.018256  0.049343
4   1        726.0  0.010335  0.061888 -0.261531 -0.021616  0.009256  0.040102
    2        726.0  0.011505  0.049861 -0.229210 -0.011947  0.013753  0.036346
    3        726.0  0.013542  0.046628 -0.247418 -0.010446  0.016140  0.040273
    4        726.0  0.015449  0.048937 -0.269256 -0.009792  0.018812  0.042819
    5        726.0  0.018243  0.059267 -0.327886 -0.013529  0.021066  0.052559
5   1        726.0  0.010721  0.070706 -0.339999 -0.028165  0.009349  0.045260
    2        726.0  0.014759  0.056030 -0.294325 -0.014329  0.016399  0.041880
    3        726.0  0.016801  0.054395 -0.271452 -0.010464  0.017702  0.045708
    4        726.0  0.017705  0.055953 -0.304036 -0.010255  0.019831  0.047812
    5        726.0  0.020650  0.062376 -0.266946 -0.012273  0.020551  0.055450

                  max
Qep Qmom
1   1        0.314436
    2        0.249003
    3        0.204233
    4        0.177621
    5        0.209763
2   1        0.352277
    2        0.303274
    3        0.206218
    4        0.202148
    5        0.203778
3   1        0.358475
    2        0.314629
    3        0.261886
    4        0.228307
    5        0.239924
4   1        0.352171
    2        0.318708
    3        0.271649
    4        0.299127
    5        0.239315
5   1        0.408924
    2        0.378606
    3        0.339344
    4        0.355978
    5        0.304707
count      726.000000
mean         0.017253
std          0.042471
```