Momentum and Data

MGMT 638: Data-Driven Investments: Equity

Kerry Back, Rice University

Momentum

- What people have found in equities and other markets (see "Value and Momentum Everywhere" by Asness and other AQR people) is
  - long-term reversals (5 year returns reverse somewhat)
  - medium-term momentum (1 year or 6 month returns continue)
  - short-term reversals (1 month or 1 week returns reverse)
- The conventional definition of momentum in academic work (including the Asness paper) is last year's return excluding the most recent month
  - In other words, the return over the first 11 of the previous 12 months.

Plan for today

- Look for momentum at the industry level using the Fama-French 49 industries
- Get acquainted with the JGSB SQL database of stock fundamentals and prices
- Sort stocks on momentum and look at portfolio returns

Fama-French 49 Industries

- Start by finding the filename to download.

In [1]:
```python
from pandas_datareader.famafrench import get_available_datasets
get_available_datasets()
```

Out[1]:
```
['F-F_Research_Data_Factors',
 'F-F_Research_Data_Factors_weekly',
 'F-F_Research_Data_Factors_daily',
 'F-F_Research_Data_5_Factors_2x3',
 'F-F_Research_Data_5_Factors_2x3_daily',
 'Portfolios_Formed_on_ME',
 'Portfolios_Formed_on_ME_Wout_Div',
 'Portfolios_Formed_on_ME_Daily',
 'Portfolios_Formed_on_BE-ME',
 'Portfolios_Formed_on_BE-ME_Wout_Div',
 'Portfolios_Formed_on_BE-ME_Daily',
 'Portfolios_Formed_on_OP',
 'Portfolios_Formed_on_OP_Wout_Div',
 'Portfolios_Formed_on_OP_Daily',
 'Portfolios_Formed_on_INV',
 'Portfolios_Formed_on_INV_Wout_Div',
 'Portfolios_Formed_on_INV_Daily',
 '6_Portfolios_2x3',
 '6_Portfolios_2x3_Wout_Div',
 '6_Portfolios_2x3_weekly',
 '6_Portfolios_2x3_daily',
 '25_Portfolios_5x5',
 '25_Portfolios_5x5_Wout_Div',
 '25_Portfolios_5x5_Daily',
 '100_Portfolios_10x10',
 '100_Portfolios_10x10_Wout_Div'
```

Get daily returns of all industries

In [2]:

```
from pandas_datareader import DataReader as pdr
rets = pdr("49_Industry_Portfolios_daily", "famafrench", start=1970)[0]/100
rets.head(3)
```

Out[2]:

	Agric	Food	Soda	Beer	Smoke	Toys	Fun	Books	Hshlc
Date									
1970-01-02	0.0305	0.0083	0.0018	0.0043	0.0000	0.0224	0.0220	-0.0004	-0.0025
1970-01-05	0.0367	0.0071	-0.0068	0.0037	0.0138	0.0028	-0.0047	-0.0026	0.0007
1970-01-06	0.0138	-0.0038	0.0032	0.0012	0.0005	-0.0172	-0.0002	-0.0091	-0.0104

3 rows × 49 columns

Calculating momentum

- Each day, we want to look back one year and compound the returns, excluding the most recent month.
- Count the days in the past year as 1, 2, ..., 252.
- We want to calculate (1+r_1)\cdots (1_r_{231}).
- We can do this as

$$\frac{(1+r_1)\cdots(1+r_{231})(1+r_{232})\cdots(1+r_{252})}{(1+r_{232})\cdots(1+r_{252})}$$

- In other words,

$$\frac{1+\text{last year's return}}{1+\text{last month's return}}$$

In [4]:
```
mom.head(3)
```

Out[4]:

	Agric	Food	Soda	Beer	Smoke	Toys	Fun
Date							
1970-12-31	-0.268268	-0.028023	0.040410	-0.080067	0.292730	-0.179984	-0.168475
1971-01-04	-0.300242	-0.025224	0.049104	-0.080067	0.253838	-0.176059	-0.169394
1971-01-05	-0.308732	-0.011133	0.055588	-0.065458	0.268250	-0.154429	-0.160920

3 rows × 49 columns

Rank by momentum each day (at close of prior day)

In [6]:

```
df.head()
```

Out[6]:

		mom	ret	rank
Date				
1970-12-31	**Agric**	-0.268268	0.0078	13.0
	Food	-0.028023	0.0051	35.0
	Soda	0.040410	-0.0069	43.0
	Beer	-0.080067	-0.0034	32.0
	Smoke	0.292730	-0.0135	48.0

Trading strategy

- Long the 5 industries with highest momentum
- Short the 5 industries with lowest momentum

In [7]:

```
long = df[df["rank"]>=45].groupby("Date").ret.mean()    # ranks 45, 46, 47, 4
short = df[df["rank"]<=5].groupby("Date").ret.mean()    # ranks 1, 2, 3, 4, 5
long_minus_short = long - short
print(f"long-minus-short annualized mean return is {252*long_minus_short.mean
```

```
long-minus-short annualized mean return is 11.15%
```

JGSB SQL Database

- Must be on RiceOwls or the Rice VPN.
- Neet to pip install or conda install pymssql and maybe sqlalchemy
- Following is boilerplate to create a connection. Always the same code for this.
- Can close the connection with conn.close().

In [8]:

```
from sqlalchemy import create_engine
import pymssql
server = 'fs.rice.edu'
database = 'stocks'
username = 'stocks'
password = '6LAZH1'
string = "mssql+pymssql://" + username + ":" + password + "@" + server + "/" 
conn = create_engine(string).connect()
```

Database tables

- tickers has one row for each ticker, with general company information
- indicators has one row for each variable in the other tables with definitions
- sf1 has annual and quarterly reports for all NYSE/Nasdaq stocks back to 2000
- sep has daily open, high, low, close and adjusted close for same stocks
- daily has marketcap, pb, pe, ps, ev, evebit, evebitda for same stocks
- sep_weekly is a weekly version of sep
- weekly is a weekly version of daily

Basic SQL

- select [] from [] join [] on [] where [] order by []
- select * means select all columns
- select top 3 * means select all columns for top 3 rows
- join [] on [] where [] order by [] are all optional
- a table that always exists in information_schema.tables. It lists the other tables.

In [9]:
```
pd.read_sql("select * from information_schema.tables", conn)
```

Out[9]:

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
0	stocks	dbo	prices_weekly	BASE TABLE
1	stocks	dbo	sep	BASE TABLE
2	stocks	dbo	sf1	BASE TABLE
3	stocks	dbo	daily	BASE TABLE
4	stocks	dbo	today	BASE TABLE
5	stocks	dbo	ghz	BASE TABLE
6	stocks	dbo	indicators	BASE TABLE
7	stocks	dbo	tickers	BASE TABLE
8	stocks	dbo	weekly	BASE TABLE
9	stocks	dbo	sep_weekly	BASE TABLE
10	stocks	dbo	sep2	BASE TABLE

In [10]:
```python
pd.read_sql("select top 3 * from tickers", conn)
```

Out[10]:

	permaticker	siccode	lastupdated	firstadded	firstpricedate	lastpricedate	fi
0	196290	3826	2023-08-31	2014-09-26	1999-11-18	2023-10-27	1
1	124392	3334	2023-10-26	2016-11-01	2016-11-01	2023-10-27	2
2	122827	6022	2019-07-29	2017-09-09	1998-09-25	2003-01-28	1

3 rows × 26 columns

In [11]:
```
pd.read_sql("select top 3 * from indicators", conn)
```

Out[11]:

	tbl	indicator	isfilter	isprimarykey	title	description	unittype
0	SF1	revenue	N	N	Revenues	[Income Statement] The amount of Revenue recog...	currency
1	SF1	cor	N	N	Cost of Revenue	[Income Statement] The aggregate cost of goods...	currency
2	SF1	sgna	N	N	Selling General and Administrative Expense	[Income Statement] A component of [OpEx] repre...	currency

In [12]:

```
pd.read_sql("select top 3 * from sep", conn)
```

Out[12]:

	ticker	date	lastupdated	opn	high	low	cls	volume	closea
0	BOOT	2021-09-10	2021-09-10	85.12	86.40	83.340	83.44	257625.000	83.4
1	BNTC	2021-09-10	2023-07-26	64.94	67.32	64.175	67.15	2597.529	67.1
2	JBHT	2023-04-05	2023-08-04	170.55	171.82	168.240	170.26	621922.000	169.5

In [13]:

```
pd.read_sql("select top 3 * from daily", conn)
```

Out[13]:

	ticker	date	lastupdated	ev	evebit	evebitda	marketcap	pb	pe	ps
0	FKLT	2000-10-10	2021-06-14	30.4	-2.7	-2.8	30.9	6.3	-2.7	9.6
1	FFDF	2000-10-10	2022-07-12	40.5	29.3	26.7	12.5	0.8	13.6	3.4
2	FL	2000-10-10	2019-03-28	2161.2	9.9	5.5	1776.2	1.5	15.6	0.4

In [14]:

```
pd.read_sql("select top 3 * from sf1", conn)
```

Out[14]:

	ticker	dimension	calendardate	datekey	reportperiod	lastupdated	accoci
0	ACHV	ARQ	1999-03-31	1999-05-13	1999-03-31	2023-08-15	0.0
1	ACHV	ARQ	1999-06-30	1999-08-06	1999-06-30	2023-08-15	0.0
2	ACHV	ARQ	1999-09-30	1999-11-16	1999-09-30	2023-08-15	0.0

3 rows × 111 columns

In [15]:
```
pd.read_sql("select top 3 * from sep_weekly", conn)
```

Out[15]:

	ticker	date	lastupdated	opn	high	low	volume	closeadj	closeunadj
0	ALX	2002-07-26	2023-08-07	73.0	73.52	68.7	5140.0	27.327	72.80
1	ALX	2002-08-02	2023-08-07	69.1	71.20	69.1	1100.0	26.712	71.16
2	ALX	2002-08-09	2023-08-07	70.0	70.00	67.1	1300.0	25.586	68.16

In [16]:
```
pd.read_sql("select top 3 * from weekly", conn)
```

Out[16]:

	ticker	date	lastupdated	ev	evebit	evebitda	marketcap	pb	pe
0	CADMQ	2000-04-14	2019-06-13	56.6	-4.2	-4.6	53.1	-126.4	-3.8
1	CADMQ	2000-04-20	2019-06-13	42.0	-3.1	-3.4	38.5	-91.7	-2.7
2	CADMQ	2000-04-28	2019-06-13	40.7	-3.0	-3.3	37.2	-88.6	-2.7

sep_weekly data

- get ticker, date, closeadj, closeunadj from sep_weekly
- keep only last updated row for each ticker, date (in case multiple updates)

In [17]:

```python
df = pd.read_sql(
    """
    select date, ticker, closeadj, closeunadj, lastupdated from sep_weekly
    where date>='2010-01-01'
    order by ticker, date, lastupdated
    """,
    conn,
)
df = df.groupby(["ticker", "date", "lastupdated"]).last()
df = df.droplevel("lastupdated")
```

In [18]:

```
df.head()
```

Out[18]:

		closeadj	closeunadj
ticker	**date**		
A	**2010-01-08**	20.187	30.96
	2010-01-15	19.855	30.45
	2010-01-22	19.900	30.52
	2010-01-29	18.707	28.69
	2010-02-05	19.235	29.50

Calculate weekly returns and momentum

- Compute weekly return as closeadj.pct_change()
- Compute annual returns (through end of prior week)
- Compute monthly returns (through end of prior week)
- Momentum $= (1 + \text{annual})/(1 + \text{monthly}) - 1$
- Momentum is through end of prior week so can be used to predict this week's returns
- Also, shift closeunadj by one week because we want to use it to filter out penny stocks.

In [20]:
```
df.head()
```

Out[20]:

ticker	date	closeadj	closeunadj	weekly	adj_shift	annual	monthly	
A	2011-01-14	27.529	42.22	0.008130	27.307	0.352702	0.127922	0.1
	2011-01-21	28.918	44.35	0.050456	27.529	0.386502	0.090949	0.2
	2011-01-28	26.721	40.98	-0.075973	28.918	0.453166	0.086204	0.3
	2011-02-04	26.727	40.99	0.000225	26.721	0.428396	-0.022426	0.4
	2011-02-11	28.318	43.43	0.059528	26.727	0.389498	-0.021240	0.4

Filter out penny stocks

- Penny stocks have an undue influence on equally weighted portfolio returns
- We could do value weighted returns instead
- Instead, we'll impose a common filter: price at the time of portfolio formation > $5.00

In [21]:

```
df = df[df.unadj_shift > 5].copy()
df.head()
```

Out[21]:

ticker	date	closeadj	closeunadj	weekly	adj_shift	annual	monthly	
A	2011-01-14	27.529	42.22	0.008130	27.307	0.352702	0.127922	0.1
	2011-01-21	28.918	44.35	0.050456	27.529	0.386502	0.090949	0.2
	2011-01-28	26.721	40.98	-0.075973	28.918	0.453166	0.086204	0.3
	2011-02-04	26.727	40.99	0.000225	26.721	0.428396	-0.022426	0.4
	2011-02-11	28.318	43.43	0.059528	26.727	0.389498	-0.021240	0.4

Form decile portfolios

- We want to go long the top momentum stocks and short the low momentum stocks and see if we earn alpha.
- Let's try top = top 10% and bottom = bottom 10%
- Could try top = top 100 and bottom = bottom 100 instead or other things.

In [31]:

```
df["decile"] = df.groupby("date", group_keys=False).mom.apply(
    lambda x: pd.qcut(x, 10, labels=range(1, 11))
)
df.head(3)
```

Out[31]:

		closeadj	closeunadj	weekly	adj_shift	annual	monthly	ı
ticker	**date**							
A	**2011-01-14**	27.529	42.22	0.008130	27.307	0.352702	0.127922	0.19
	2011-01-21	28.918	44.35	0.050456	27.529	0.386502	0.090949	0.27
	2011-01-28	26.721	40.98	-0.075973	28.918	0.453166	0.086204	0.33

Returns of equally-weighted portfolios

In [32]:

```
port_rets = df.groupby(["date", "decile"], group_keys=True).weekly.mean()
port_rets = port_rets.unstack()
port_rets.head()
```

Out[32]:

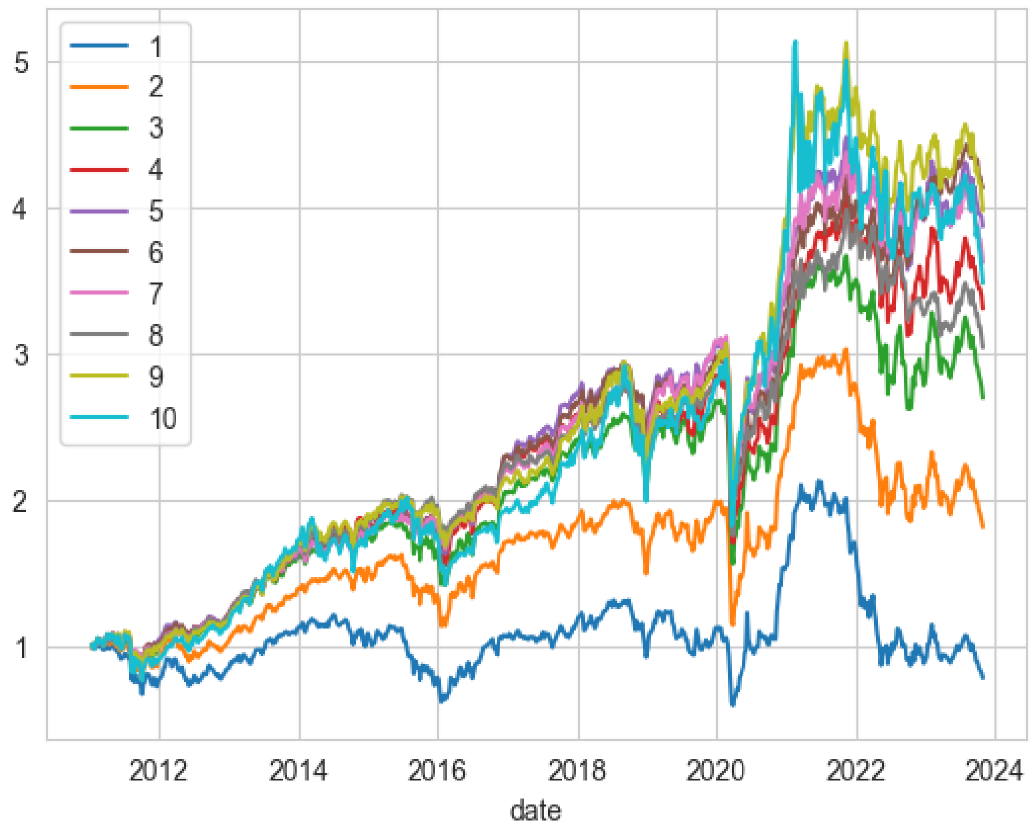
decile	1	2	3	4	5	6	7
date							
2011-01-14	-0.003574	-0.003988	-0.000700	-0.003290	-0.005264	-0.007649	0.00055
2011-01-21	0.017434	0.018820	0.015247	0.014496	0.011896	0.012560	0.01154
2011-01-28	-0.024139	-0.017535	-0.014224	-0.009462	-0.012853	-0.011438	-0.01580
2011-02-04	-0.005927	-0.001812	0.001808	0.000847	0.001169	0.002995	0.00554
2011-02-11	0.016875	0.023576	0.018955	0.026951	0.019405	0.023759	0.025541

In [33]:
```
port_rets.mean()
```

Out[33]:
```
decile
1     0.000457
2     0.001333
3     0.001823
4     0.002074
5     0.002281
6     0.002367
7     0.002174
8     0.001941
9     0.002400
10    0.002452
dtype: float64
```

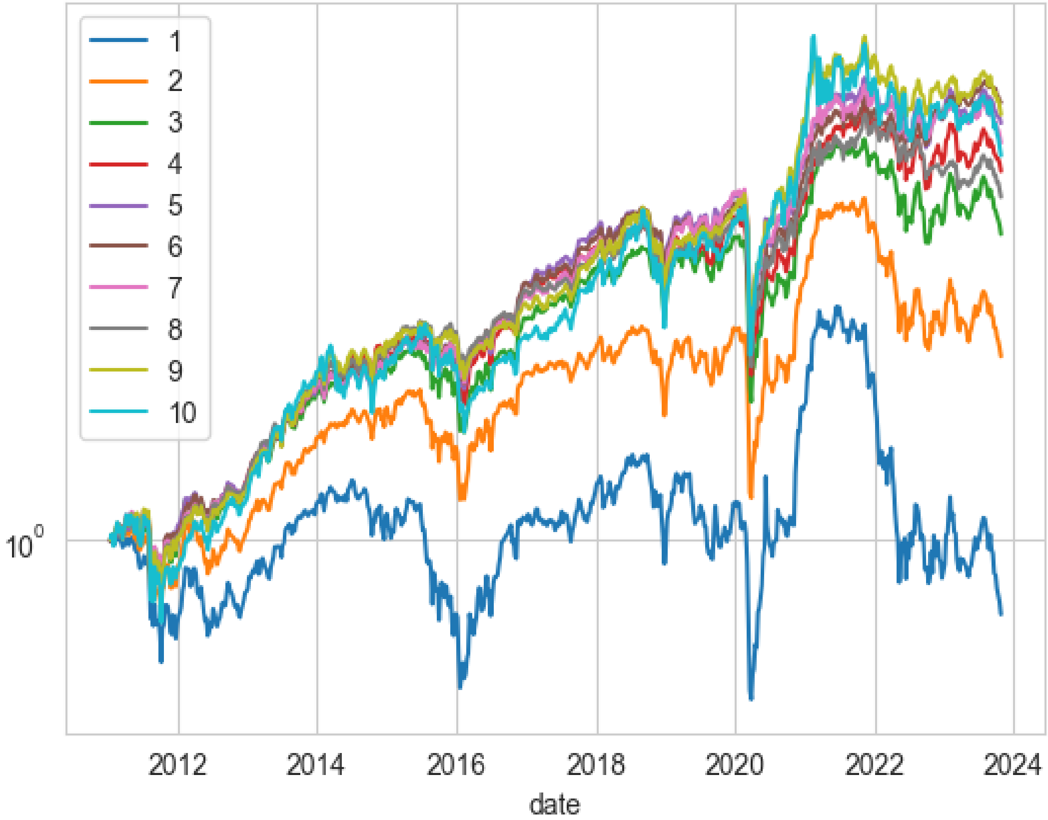
In [41]:

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")

(1+port_rets).cumprod().plot()
plt.legend()
plt.show()
```



In [42]:

```python
# log scale

(1+port_rets).cumprod().plot(logy=True)
plt.legend()
plt.show()
```



Long-minus short returns

In [36]:

```
long_minus_short = port_rets[10] - port_rets[1]
print(f"annualized long-minus-short return is {52*long_minus_short.mean():.2%
```

```
annualized long-minus-short return is 10.38%
```

In [43]:
```
(1+long_minus_short).cumprod().plot()
```

Out[43]: <AxesSubplot: xlabel='date'>

Database: industries and other information

- famaindustry and siccode, etc. are in tickers table
- join tables in SQL to get this info in
- or download tickers table and merge in pandas

Joining in SQL Example

In [37]:

```
df2 = pd.read_sql(
    """
    select date, a.ticker, closeadj, closeunadj, a.lastupdated,
    famaindustry, siccode
    from sep_weekly as a join tickers as b
    on a.ticker=b.ticker
    where date>='2020-01-01'
    order by a.ticker, date, a.lastupdated
    """,
    conn,
)
df2 = df2.groupby(["ticker", "date", "lastupdated"]).last()
df2 = df2.droplevel("lastupdated")
```

In [38]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 972409 entries, ('A', datetime.date(2020, 1, 3)) to ('ZYX
I', datetime.date(2023, 10, 27))
Data columns (total 4 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   closeadj      972409 non-null  float64
 1   closeunadj    972409 non-null  float64
 2   famaindustry  970284 non-null  object 
 3   siccode       972409 non-null  int64  
dtypes: float64(2), int64(1), object(1)
memory usage: 33.4+ MB
```

Merging in Pandas Example

In [39]:

```
df3 = pd.read_sql(
    """
    select date, ticker, closeadj, closeunadj, lastupdated
    from sep_weekly
    where date>='2020-01-01'
    order by ticker, date, lastupdated
    """,
    conn,
)
df3 = df3.groupby(["ticker", "date", "lastupdated"]).last()
df3 = df3.droplevel("lastupdated")
```

In [40]:

```
tickers = pd.read_sql("select ticker, siccode, famaindustry from tickers", co
df3 = df3.reset_index().merge(tickers, on="ticker")
df3 = df3.set_index(["ticker", "date"])
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 972409 entries, ('A', datetime.date(2020, 1, 3)) to ('ZYX
I', datetime.date(2023, 10, 27))
Data columns (total 4 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   closeadj      972409 non-null  float64
 1   closeunadj    972409 non-null  float64
 2   siccode       972409 non-null  int64
 3   famaindustry  970284 non-null  object
dtypes: float64(2), int64(1), object(1)
memory usage: 33.6+ MB
```