Factor Investing

BUSI 722: Data-Driven Finance II

Kerry Back, Rice University

Overview

- Introduction to factors
- SQL Database
- Examples of constructing features
- Sorts

Introduction to factors

- Factor investing at BlackRock{.external target="_blank"}
- Factor investing at AQR{.external target="_blank"}

Some factors (features)

- Value
    - Price to book
    - Price to earnings
- Momentum / reversal
    - Last month or week return (short-term reversal)
    - Last six-months or year return excluding most recent month (momentum)
    - Last five-year return excluding most recent year (long-term reversal)

- Volatility
    - Standard deviation
    - Standard deviation of CAPM residual
    - Standard deviation of Fama-French residual
- Volume (liquidity)
- Profitability
    - Return on equity (quarterly or annual)
    - Operating profitability (Revenue - COGS - SG&A - Taxes) / assets
- Asset growth
- Accruals (net income - operating cash flow)

- Dividend announcements and yields
- Earnings announcements
- Sentiment (text analysis)
- Short interest
- Corporate insider (director/executive/large shareholder) trades

Some data from Ken French's data library

- Monthly returns of value-weighted portfolios constructed from sorts on characteristics
- Either (i) one characteristic at a time or (ii) size and another characteristic
- One at a time
- Size and another

SQL database for this course

- Annual and quarterly reports, prices, volume
- On Rice server. Must be on campus or on Rice VPN.
- Data is downloaded daily from Nasdaq Data Link.
- Use either pyodbc or pymssql (pymssql is deprecated). For Macs, need to install Microsoft's ODBC Driver. There have been issues with Macs.

Establish a connection

Can always use this code to connect (I hope).

In [104]:

```python
from sqlalchemy import create_engine

server = 'fs.rice.edu'
database = 'stocks'
username = 'stocks'
password = '6LAZH1'
driver = 'SQL+Server'
string = f"mssql+pyodbc://{username}:{password}@{server}/{database}"
try:
    conn = create_engine(string + "?driver='SQL+Server'").connect()
except:
    try:
        conn = create_engine(string + "?driver='ODBC+Driver+18+for+SQL+Server
    except:
        import pymssql
        string = f"mssql+pymssql://{username}:{password}@{server}/{database}"
        conn = create_engine(string).connect()
```

Overview of tables in the database

In [105]:

```python
import pandas as pd
pd.read_sql("select * from information_schema.tables", conn)
```

Out[105]:

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
0	stocks	dbo	sf1	BASE TABLE
1	stocks	dbo	sep_weekly	BASE TABLE
2	stocks	dbo	weekly	BASE TABLE
3	stocks	dbo	today	BASE TABLE
4	stocks	dbo	ghz	BASE TABLE
5	stocks	dbo	indicators	BASE TABLE
6	stocks	dbo	tickers	BASE TABLE

tickers table

tickers has one row for each ticker, with general company information

In [106]:
```
tickers = pd.read_sql("select top 3 * from tickers", conn)
tickers
```

Out[106]:

	permaticker	siccode	lastupdated	firstadded	firstpricedate	lastpricedate	fi
0	196290	3826	2023-12-20	2014-09-26	1999-11-18	2024-01-30	1
1	124392	3334	2023-10-26	2016-11-01	2016-11-01	2024-01-30	2
2	122827	6022	2019-07-29	2017-09-09	1998-09-25	2003-01-28	1

3 rows × 26 columns

In [107]:
```
for col in tickers.columns: print(col)
```

```
permaticker
siccode
lastupdated
firstadded
firstpricedate
lastpricedate
firstquarter
lastquarter
isdelisted
ticker
name
exchange
cusips
sicsector
sicindustry
famasector
famaindustry
sector
industry
scalemarketcap
scalerevenue
relatedtickers
currency
location
secfilings
companysite
```

indicators

indicators has one row for each variable in the other tables with definitions

In [108]:
```python
indicators = pd.read_sql("select * from indicators", conn)
indicators.head()
```

Out[108]:

	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
					Research and	[Income Statement] A	

In [109]:

```
indicators.to_excel("indicators.xlsx")
```

In [110]:

```
for col in indicators.columns: print(col)
```

```
tbl
indicator
isfilter
isprimarykey
title
description
unittype
```

sf1

sf1 has annual and quarterly reports for all NYSE/Nasdaq stocks since 2000

- ARQ = as reported quarterly
- ARY = as reported yearly
- MRQ = modified (includes restatements) quarterly
- MRY = modified (includes restatements) yearly

In [111]:

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

Out[111]:

	ticker	dimension	calendardate	datekey	reportperiod	lastupdated	a
0	MET	ARQ	2011-03-31	2011-05-10	2011-03-31	2023-11-02	1.11500(
1	MET	ARQ	2011-06-30	2011-08-05	2011-06-30	2023-11-02	3.35600(
2	MET	ARQ	2011-09-30	2011-11-04	2011-09-30	2023-11-02	6.81300(

3 rows × 111 columns

In [112]:
```python
for col in sf1.columns: print(col)
```

```
ticker
dimension
calendardate
datekey
reportperiod
lastupdated
accoci
assets
assetsavg
assetsc
assetsnc
assetturnover
bvps
capex
cashneq
cashnequsd
cor
consolinc
currentratio
de
debt
debtc
debtnc
debtusd
deferredrev
depamor
deposits
```

sep_weekly

sep_weekly has weekly open (opn), high, low, closeadj, closeunad, and average daily volume

In [113]:

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

weekly

weekly has end-of-week enterprise value, enterprise value to ebit, enterprise value to ebitda, marketcap, price to book, price to earnings, and price to sales

In [114]:
```
pd.read_sql("select top 3 * from weekly", conn)
```

Out[114]:

	ticker	date	lastupdated	ev	evebit	evebitda	marketcap	pb	pe
0	A	2000-01-07	2019-03-28	32040.0	47.9	28.9	32040.0	10.0	62.6
1	A	2000-01-14	2019-03-28	30678.3	45.9	27.7	30678.3	9.5	59.9
2	A	2000-01-21	2019-03-28	31817.5	47.6	28.7	31817.5	9.9	62.1

Examples of constructing features

- Momentum, price-to-book, marketcap, ROE, asset growth
- Tables
    - sep_weekly: closeadj $\longrightarrow$ returns and momentum, closeunadj $\longrightarrow$ exclude penny stocks
    - weekly: price-to-book and marketcap
    - sf1: assets $\longrightarrow$ asset growth, netinc and equity $\longrightarrow$ roe
- We will limit the date range to 2010 on for speed.
- Rarely, there are strange data entries - two rows for the same ticker/date. We'll keep the last updated row in this case.

sep_weekly

In [115]:

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

ret = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_change()
ret.name = "ret"

price = sep_weekly.closeunadj
price.name = "price"
```

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.

Calculating momentum

- Each week, we want to look back one year and compound the returns, excluding the most recent month.
- Count the weeks in the prior year as 1, 2, ..., 52.
- We want to calculate $(1+r_1)\cdots(1+r_{48})$.
- We can do this as

$$\frac{(1+r_1)\cdots(1+r_{52})}{(1+r_{49})\cdots(1+r_{52})}$$

- In other words,

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

In [116]:

```
ret_annual = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_chan
ret_monthly = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_cha
mom = (1 + ret_annual) / (1 + ret_monthly) - 1
mom.name = "mom"
```

Value

- Value means cheap relative to quality. Value investing has a very long tradition.
- Conventional measures are price-to-earnings (PE) and price-to-book (PB).
- Low PE or low PB stocks are value stocks. High PE or PB stocks are "growth stocks" or "glamour stocks."
- We'll get PB, but PE is also worth exploring (also price-to-sales, price-to-clicks, ...)

weekly

In [117]:

```
weekly = pd.read_sql(
    """
    select date, ticker, pb, marketcap, lastupdated from weekly
    where date>='2010-01-01'
    order by ticker, date, lastupdated
    """,
    conn,
)
weekly = weekly.groupby(["ticker", "date"]).last()
weekly = weekly.drop(columns=["lastupdated"])

pb = weekly.pb
pb.name = "pb"
marketcap = weekly.marketcap
marketcap.name = "marketcap"
```

Asset growth and ROE

- Fast growing firms in terms of % change in assets have historically been poor investments.
- Get total assets from sf1 (dimension=ARY) and compute % change year to year.
- High ROE firms have historically been good investments. Define ROE as net income / lagged book equity.

Combining data of different frequencies

- sf1 data is quarterly or annual. date is date of posting on SEC website.
- Other data is weekly = Fridays.
- Convert sf1 dates to Fridays.

sf1

In [118]:

```python
sf1 = pd.read_sql(
    """
    select datekey as date, ticker, assets, netinc, equity, lastupdated from
    where datekey>='2010-01-01' and dimension='ARY' and assets>0 and equity>0
    order by ticker, datekey, lastupdated
    """,
    conn,
)
sf1 = sf1.groupby(["ticker", "date"]).last()
sf1 = sf1.drop(columns=["lastupdated"])

# change dates to Fridays
from datetime import timedelta
sf1 = sf1.reset_index()
sf1.date =sf1.date.map(
    lambda x: x + timedelta(4 - x.weekday())
)
sf1 = sf1.set_index(["ticker", "date"])
sf1 = sf1[~sf1.index.duplicated()]

assets = sf1.assets
assets.name = "assets"
netinc = sf1.netinc
netinc.name = "netinc"
equity = sf1.equity
equity.name = "equity"
```

Sorts

Returns of portfolios based on sorts

- Merge a feature or multiple features with returns.
- Shift returns backwards.
    - Return on each Friday is return ending on close of that Friday.
    - Features are also known by Friday close.
    - We want to use features to predict future returns, so shift returns backwards, so the following week's return is aligned with features.
- Exclude penny stocks (e.g., price >= 5).
- Sort each week into groups based on feature(s) - e.g., deciles.
- Compute average (following week) return in each decile. This is the return of the portfolio that is equally weighted (same $ investment in each stock).

Sorting on momentum

In [119]:

```
df = pd.concat((ret, mom, price), axis=1)
df["ret"] = df.groupby("ticker", group_keys=False).ret.shift(-1)
df = df[df.price >= 5]
df = df.dropna()

df["mom10"] = df.groupby("date", group_keys=False).mom.apply(
    lambda x: pd.qcut(x, 10, labels=range(1, 11))
)
mom10 = df.groupby(
    ["date", "mom10"],
    observed=True,
    group_keys=True
).ret.mean().unstack()

mom10.head()
```

Out[119]:

mom10	1	2	3	4	5	6	
date							
2010-12-31	0.016544	0.014900	0.008000	0.008447	0.005864	0.005382	0.0080
2011-01-07	-0.002317	-0.002727	-0.001495	-0.005026	-0.005653	-0.005736	-0.0017
2011-01-14	0.016414	0.018445	0.015778	0.014730	0.012619	0.011439	0.0124
2011-	-0.023235	-0.016761	-0.016063	-0.008631	-0.012379	-0.011164	-0.0153

In [120]:

```
(100 * 52 * mom10.mean()).round(2)
```

Out[120]:

```
mom10
1      3.83
2      8.82
3     10.68
4     11.91
5     13.25
6     12.76
7     11.34
8     11.34
9     13.91
10    14.47
dtype: float64
```

Does size matter?

Repeat for small caps, defined as not in the top 1,000 by marketcap.

In [121]:

```
df = pd.concat((ret, mom, price, marketcap), axis=1)
df["ret"] = df.groupby("ticker", group_keys=False).ret.shift(-1)
df = df[df.price >= 5]
df["rnk"] = df.groupby("date", group_keys=False).marketcap.rank(ascending=Fal
df = df[df.rnk>1000]
df = df.dropna()

df["mom10"] = df.groupby("date", group_keys=False).mom.apply(
    lambda x: pd.qcut(x, 10, labels=range(1, 11))
)
mom10 = df.groupby(
    ["date", "mom10"],
    observed=True,
    group_keys=True
).ret.mean().unstack()

(100 * 52 * mom10.mean()).round(2)
```

Out[121]:

```
mom10
1      2.02
2      7.71
3     10.56
4     10.37
5     12.59
6     12.99
7     11.68
8     10.90
9     13.44
10    14.58
```

Exercise

- Sort into deciles based on marketcap (using all stocks, not just small caps).
- Compute equally weighted portfolio returns.

Double sort on momentum and price-to-book

- Sort into quintiles on mom and pb separately
- Intersect the quintiles to get 25 groups each week
- Compute equally weighted portfolio returns

In [122]:

```
df = pd.concat((ret, mom, pb, price), axis=1)
df["ret"] = df.groupby("ticker", group_keys=False).ret.shift(-1)
df = df[df.price >= 5]
df = df.dropna()

df["mom5"] = df.groupby("date", group_keys=False).mom.apply(
    lambda x: pd.qcut(x, 5, labels=range(1, 6))
)
df["pb5"] = df.groupby("date", group_keys=False).pb.apply(
    lambda x: pd.qcut(x, 5, labels=range(1, 6))
)

mom5_pb5 = df.groupby(
    ["date", "mom5", "pb5"],
    observed=True,
    group_keys=True
).ret.mean().unstack(level=["pb5", "mom5"])

(100 * 52 * mom5_pb5.mean()).round(2).unstack()
```

Out[122]:

mom5	1	2	3	4	5
pb5					
1	4.20	13.23	16.01	13.85	15.04
2	7.47	10.68	10.72	10.71	12.76
3	8.19	10.27	11.47	11.12	14.55
4	7.42	11.34	13.33	11.56	11.82

Exercise

Intersect quintile sorts on momentum and marketcap and compute mean portfolio returns.

Sorting on ROE

- Compute roe = netinc / lagged equity
- Merge with returns and prices
- Forward fill roe into weeks. Each week will show the most recently reported roe. roe will change only once per year when a new annual report comes out.
- roe will be missing until a firm has filed two annual reports. So we start the data in 2012 (2 years after 2010).

In [123]:

```
equity = equity.groupby("ticker", group_keys=False).shift()
roe = netinc / equity
roe.name = "roe"

df = pd.concat((ret, roe, price), axis=1)
df["ret"] = df.groupby("ticker", group_keys=False).ret.shift(-1)

## forward fill
df["roe"] = df.groupby("ticker", group_keys=False).roe.ffill()

df = df[df.price >= 5]
df = df[df.index.get_level_values("date").astype(str) >= "2012-01-01"]
df = df.dropna()

df["roe10"] = df.groupby("date", group_keys=False).roe.apply(
    lambda x: pd.qcut(x, 10, labels=range(1, 11))
)
roe10 = df.groupby(
    ["date", "roe10"],
    observed=True,
    group_keys=True
).ret.mean().unstack()

(100 * 52 * roe10.mean()).round(2)
```

Out[123]:

```
roe10
1       6.78
2      10.69
3      11.34
```

Sorting on asset growth

- % change in assets
- Forward fill and subset to date >= 2012-01-01 as for roe

In [124]:

```python
assetgr = assets.groupby("ticker", group_keys=False).pct_change()
assetgr.name = "assetgr"

df = pd.concat((ret, assetgr, price), axis=1)
df["ret"] = df.groupby("ticker", group_keys=False).ret.shift(-1)

## forward fill
df["assetgr"] = df.groupby("ticker", group_keys=False).assetgr.ffill()

df = df[df.price >= 5]
df = df[df.index.get_level_values("date").astype(str) >= "2012-01-01"]
df = df.dropna()

df["assetgr10"] = df.groupby("date", group_keys=False).assetgr.apply(
    lambda x: pd.qcut(x, 10, labels=range(1, 11))
)
assetgr10 = df.groupby(
    ["date", "assetgr10"],
    observed=True,
    group_keys=True
).ret.mean().unstack()

(100 * 52 * assetgr10.mean()).round(2)
```

Out[124]:

```
assetgr10
1      11.65
2      12.88
3      11.04
4      12.78
```