

01_feature_engineering

September 29, 2021

1 How to transform data into factors

Based on a conceptual understanding of key factor categories, their rationale and popular metrics, a key task is to identify new factors that may better capture the risks embodied by the return drivers laid out previously, or to find new ones.

In either case, it will be important to compare the performance of innovative factors to that of known factors to identify incremental signal gains.

We create the dataset here and store it in our [data](#) folder to facilitate reuse in later chapters.

1.1 Imports & Settings

```
[1]: import warnings
      warnings.filterwarnings('ignore')
```

```
[2]: %matplotlib inline

      from datetime import datetime
      import pandas as pd
      import pandas_datareader.data as web

      # replaces pyfinance.ols.PandasRollingOLS (no longer maintained)
      from statsmodels.regression.rolling import RollingOLS
      import statsmodels.api as sm

      import matplotlib.pyplot as plt
      import seaborn as sns
```

```
[3]: sns.set_style('whitegrid')
      idx = pd.IndexSlice
```

1.2 Get Data

The `assets.h5` store can be generated using the the notebook [create_datasets](#) in the [data](#) directory in the root directory of this repo for instruction to download the following dataset.

We load the Quandl stock price datasets covering the US equity markets 2000-18 using `pd.IndexSlice` to perform a slice operation on the `pd.MultiIndex`, select the adjusted close price

and unpivot the column to convert the DataFrame to wide format with tickers in the columns and timestamps in the rows:

Set data store location:

```
[4]: DATA_STORE = '../data/assets.h5'
```

```
[5]: START = 2000
     END = 2018
```

```
[6]: with pd.HDFStore(DATA_STORE) as store:
      prices = (store['quandl/wiki/prices']
                .loc[idx[str(START):str(END), :], 'adj_close']
                .unstack('ticker'))
      stocks = store['us_equities/stocks'].loc[:, ['marketcap', 'ipoyear', 'sector']]
```

```
[7]: prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 4706 entries, 2000-01-03 to 2018-03-27
Columns: 3199 entries, A to ZUMZ
dtypes: float64(3199)
memory usage: 114.9 MB
```

```
[8]: stocks.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6834 entries, PIH to ZYME
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   marketcap   5766 non-null   float64
1   ipoyear     3038 non-null   float64
2   sector      5288 non-null   object
dtypes: float64(2), object(1)
memory usage: 213.6+ KB
```

1.2.1 Keep data with stock info

Remove stocks duplicates and align index names for later joining.

```
[9]: stocks = stocks[~stocks.index.duplicated()]
     stocks.index.name = 'ticker'
```

Get tickers with both price information and metdata

```
[10]: shared = prices.columns.intersection(stocks.index)
```

```
[11]: stocks = stocks.loc[shared, :]  
stocks.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Index: 2412 entries, A to ZUMZ  
Data columns (total 3 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   marketcap   2407 non-null   float64  
1   ipoyear      1065 non-null   float64  
2   sector       2372 non-null   object  
dtypes: float64(2), object(1)  
memory usage: 75.4+ KB
```

```
[12]: prices = prices.loc[:, shared]  
prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 4706 entries, 2000-01-03 to 2018-03-27  
Columns: 2412 entries, A to ZUMZ  
dtypes: float64(2412)  
memory usage: 86.6 MB
```

```
[13]: assert prices.shape[1] == stocks.shape[0]
```

1.3 Create monthly return series

To reduce training time and experiment with strategies for longer time horizons, we convert the business-daily data to month-end frequency using the available adjusted close price:

```
[14]: monthly_prices = prices.resample('M').last()
```

To capture time series dynamics that reflect, for example, momentum patterns, we compute historical returns using the method `.pct_change(n_periods)`, that is, returns over various monthly periods as identified by lags.

We then convert the wide result back to long format with the `.stack()` method, use `.pipe()` to apply the `.clip()` method to the resulting `DataFrame`, and winsorize returns at the [1%, 99%] levels; that is, we cap outliers at these percentiles.

Finally, we normalize returns using the geometric average. After using `.swaplevel()` to change the order of the `MultiIndex` levels, we obtain compounded monthly returns for six periods ranging from 1 to 12 months:

```
[15]: monthly_prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 219 entries, 2000-01-31 to 2018-03-31  
Freq: M
```

Columns: 2412 entries, A to ZUMZ
dtypes: float64(2412)
memory usage: 4.0 MB

```
[16]: outlier_cutoff = 0.01
data = pd.DataFrame()
lags = [1, 2, 3, 6, 9, 12]
for lag in lags:
    data[f'return_{lag}m'] = (monthly_prices
                             .pct_change(lag)
                             .stack()
                             .pipe(lambda x: x.clip(lower=x.
→quantile(outlier_cutoff),
                                                    upper=x.
→quantile(1-outlier_cutoff)))
                             .add(1)
                             .pow(1/lag)
                             .sub(1)
                             )
data = data.swaplevel().dropna()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 399525 entries, ('A', Timestamp('2001-01-31 00:00:00', freq='M')) to
('ZUMZ', Timestamp('2018-03-31 00:00:00', freq='M'))
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  -
0   return_1m    399525 non-null   float64
1   return_2m    399525 non-null   float64
2   return_3m    399525 non-null   float64
3   return_6m    399525 non-null   float64
4   return_9m    399525 non-null   float64
5   return_12m   399525 non-null   float64
dtypes: float64(6)
memory usage: 19.9+ MB
```

1.4 Drop stocks with less than 10 yrs of returns

```
[17]: min_obs = 120
nobs = data.groupby(level='ticker').size()
keep = nobs[nobs>min_obs].index

data = data.loc[idx[keep,:], :]
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 360752 entries, ('A', Timestamp('2001-01-31 00:00:00', freq='M')) to
```

```
('ZUMZ', Timestamp('2018-03-31 00:00:00', freq='M'))
```

Data columns (total 6 columns):

#	Column	Non-Null Count	Dtype
0	return_1m	360752 non-null	float64
1	return_2m	360752 non-null	float64
2	return_3m	360752 non-null	float64
3	return_6m	360752 non-null	float64
4	return_9m	360752 non-null	float64
5	return_12m	360752 non-null	float64

dtypes: float64(6)

memory usage: 18.0+ MB

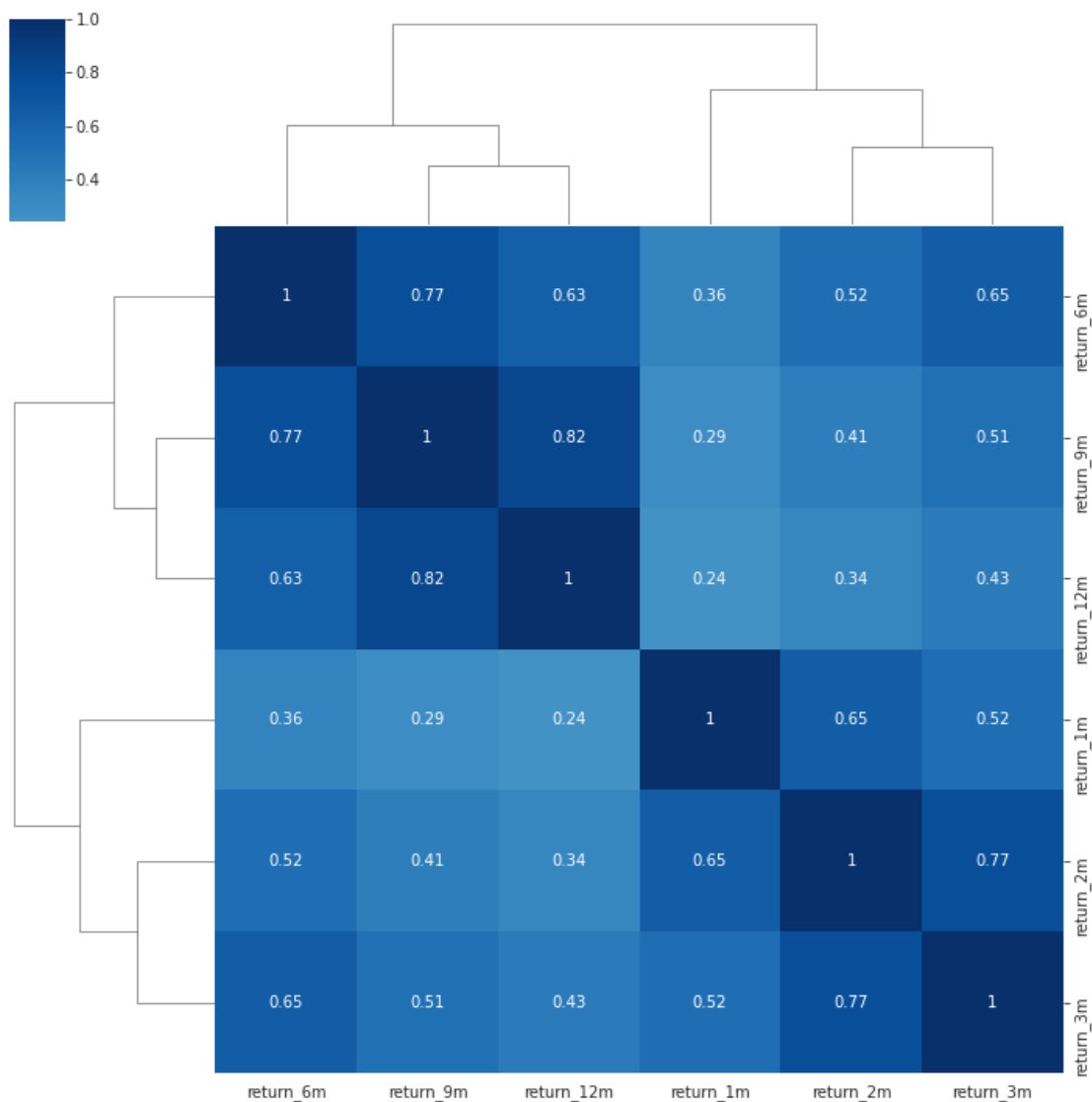
```
[18]: data.describe()
```

```
[18]:
```

	return_1m	return_2m	return_3m	return_6m	\
count	360752.000000	360752.000000	360752.000000	360752.000000	
mean	0.012255	0.009213	0.008181	0.007025	
std	0.114236	0.081170	0.066584	0.048474	
min	-0.329564	-0.255452	-0.214783	-0.162063	
25%	-0.046464	-0.030716	-0.023961	-0.014922	
50%	0.009448	0.009748	0.009744	0.009378	
75%	0.066000	0.049249	0.042069	0.031971	
max	0.430943	0.281819	0.221789	0.154555	

	return_9m	return_12m
count	360752.000000	360752.000000
mean	0.006552	0.006296
std	0.039897	0.034792
min	-0.131996	-0.114283
25%	-0.011182	-0.009064
50%	0.008982	0.008726
75%	0.027183	0.024615
max	0.124718	0.106371

```
[19]: # cmap = sns.diverging_palette(10, 220, as_cmap=True)
sns.clustermap(data.corr('spearman'), annot=True, center=0, cmap='Blues');
```



We are left with 1,670 tickers.

```
[20]: data.index.get_level_values('ticker').nunique()
```

[20]: 1838

1.5 Rolling Factor Betas

We will introduce the Fama—French data to estimate the exposure of assets to common risk factors using linear regression in [Chapter 9, Time Series Models](#).

The five Fama—French factors, namely market risk, size, value, operating profitability, and investment have been shown empirically to explain asset returns and are commonly used to assess the risk/return profile of portfolios. Hence, it is natural to include past factor exposures as financial

features in models that aim to predict future returns.

We can access the historical factor returns using the `pandas-datareader` and estimate historical exposures using the `RollingOLS` rolling linear regression functionality in the `statsmodels` library as follows:

Use Fama-French research factors to estimate the factor exposures of the stock in the dataset to the 5 factors market risk, size, value, operating profitability and investment.

```
[21]: factors = ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA']
factor_data = web.DataReader('F-F_Research_Data_5_Factors_2x3', 'famafrench',
                             start='2000')[0].drop('RF', axis=1)
factor_data.index = factor_data.index.to_timestamp()
factor_data = factor_data.resample('M').last().div(100)
factor_data.index.name = 'date'
factor_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 254 entries, 2000-01-31 to 2021-02-28
Freq: M
Data columns (total 5 columns):
#   Column   Non-Null Count  Dtype
---  -
0   Mkt-RF    254 non-null    float64
1   SMB       254 non-null    float64
2   HML       254 non-null    float64
3   RMW       254 non-null    float64
4   CMA       254 non-null    float64
dtypes: float64(5)
memory usage: 11.9 KB
```

```
[22]: factor_data = factor_data.join(data['return_1m']).sort_index()
factor_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 360752 entries, ('A', Timestamp('2001-01-31 00:00:00', freq='M')) to
('ZUMZ', Timestamp('2018-03-31 00:00:00', freq='M'))
Data columns (total 6 columns):
#   Column   Non-Null Count  Dtype
---  -
0   Mkt-RF    360752 non-null  float64
1   SMB       360752 non-null  float64
2   HML       360752 non-null  float64
3   RMW       360752 non-null  float64
4   CMA       360752 non-null  float64
5   return_1m 360752 non-null  float64
dtypes: float64(6)
memory usage: 18.0+ MB
```

```
[23]: T = 24
betas = (factor_data.groupby(level='ticker',
                             group_keys=False)
         .apply(lambda x: RollingOLS(endog=x.return_1m,
                                     exog=sm.add_constant(x.drop('return_1m',
                                     ↪axis=1))),
         window=min(T, x.shape[0]-1))
         .fit(params_only=True)
         .params
         .drop('const', axis=1)))
```

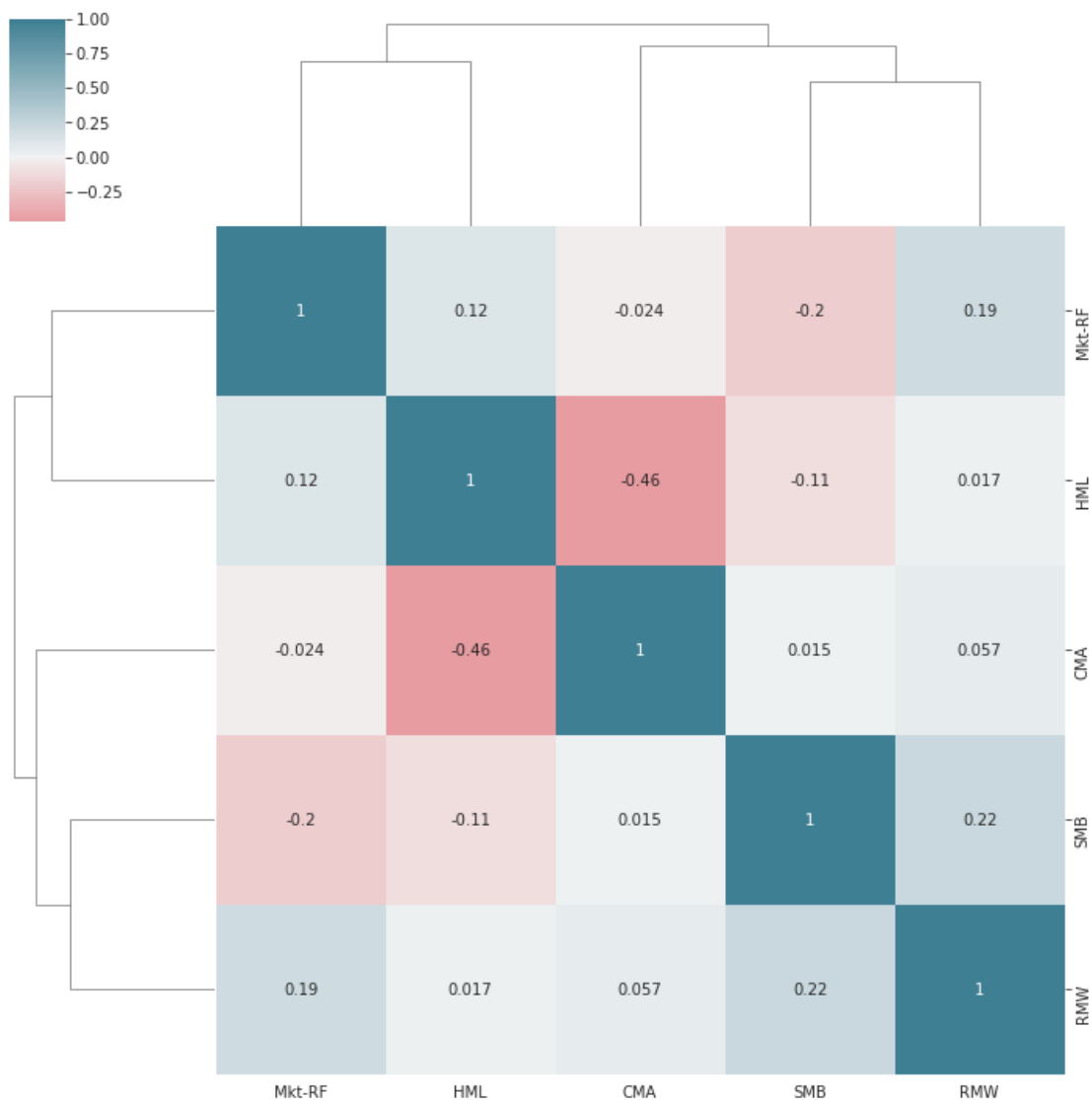
```
[24]: betas.describe().join(betas.sum(1).describe().to_frame('total'))
```

```
[24]:
```

	Mkt-RF	SMB	HML	RMW \
count	318478.000000	318478.000000	318478.000000	318478.000000
mean	0.981855	0.628163	0.128131	-0.059253
std	0.918800	1.248071	1.615972	1.919938
min	-9.922641	-10.212033	-17.654894	-22.925165
25%	0.465445	-0.114605	-0.710399	-0.979216
50%	0.932070	0.542998	0.101903	0.039229
75%	1.447381	1.303487	0.955263	0.955362
max	10.916430	10.373043	14.558920	17.413382

	CMA	total
count	318478.000000	360752.000000
mean	0.013774	1.494318
std	2.182730	3.291402
min	-18.182706	-31.429456
25%	-1.086919	0.000000
50%	0.032834	1.214277
75%	1.140431	3.145515
max	17.626042	33.316296

```
[25]: cmap = sns.diverging_palette(10, 220, as_cmap=True)
sns.clustermap(betas.corr(), annot=True, cmap=cmap, center=0);
```

```
[26]: data = (data
        .join(betas
              .groupby(level='ticker')
              .shift()))
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 360752 entries, ('A', Timestamp('2001-01-31 00:00:00', freq='M')) to
('ZUMZ', Timestamp('2018-03-31 00:00:00', freq='M'))
Data columns (total 11 columns):
#   Column      Non-Null Count  Dtype
---  -
0   return_1m    360752 non-null float64
```

```

1  return_2m    360752 non-null    float64
2  return_3m    360752 non-null    float64
3  return_6m    360752 non-null    float64
4  return_9m    360752 non-null    float64
5  return_12m   360752 non-null    float64
6  Mkt-RF       316640 non-null    float64
7  SMB          316640 non-null    float64
8  HML          316640 non-null    float64
9  RMW          316640 non-null    float64
10 CMA          316640 non-null    float64
dtypes: float64(11)
memory usage: 39.8+ MB

```

1.5.1 Impute mean for missing factor betas

```

[27]: data.loc[:, factors] = data.groupby('ticker')[factors].apply(lambda x: x.
    ↪fillna(x.mean()))
data.info()

```

```

<class 'pandas.core.frame.DataFrame'>
MultiIndex: 360752 entries, ('A', Timestamp('2001-01-31 00:00:00', freq='M')) to
('ZUMZ', Timestamp('2018-03-31 00:00:00', freq='M'))
Data columns (total 11 columns):
#   Column          Non-Null Count  Dtype
---  -
0   return_1m       360752 non-null  float64
1   return_2m       360752 non-null  float64
2   return_3m       360752 non-null  float64
3   return_6m       360752 non-null  float64
4   return_9m       360752 non-null  float64
5   return_12m      360752 non-null  float64
6   Mkt-RF          360752 non-null  float64
7   SMB             360752 non-null  float64
8   HML             360752 non-null  float64
9   RMW             360752 non-null  float64
10  CMA             360752 non-null  float64
dtypes: float64(11)
memory usage: 39.8+ MB

```

1.6 Momentum factors

We can use these results to compute momentum factors based on the difference between returns over longer periods and the most recent monthly return, as well as for the difference between 3 and 12 month returns as follows:

```

[28]: for lag in [2,3,6,9,12]:
    data[f'momentum_{lag}'] = data[f'return_{lag}m'].sub(data.return_1m)
data[f'momentum_3_12'] = data[f'return_12m'].sub(data.return_3m)

```

1.7 Date Indicators

```
[29]: dates = data.index.get_level_values('date')
      data['year'] = dates.year
      data['month'] = dates.month
```

1.8 Lagged returns

To use lagged values as input variables or features associated with the current observations, we use the `.shift()` method to move historical returns up to the current period:

```
[30]: for t in range(1, 7):
      data[f'return_1m_t-{t}'] = data.groupby(level='ticker').return_1m.shift(t)
      data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 360752 entries, ('A', Timestamp('2001-01-31 00:00:00', freq='M')) to
('ZUMZ', Timestamp('2018-03-31 00:00:00', freq='M'))
Data columns (total 25 columns):
#   Column                Non-Null Count  Dtype
---  -
0   return_1m             360752 non-null float64
1   return_2m             360752 non-null float64
2   return_3m             360752 non-null float64
3   return_6m             360752 non-null float64
4   return_9m             360752 non-null float64
5   return_12m            360752 non-null float64
6   Mkt-RF                 360752 non-null float64
7   SMB                   360752 non-null float64
8   HML                   360752 non-null float64
9   RMW                   360752 non-null float64
10  CMA                    360752 non-null float64
11  momentum_2             360752 non-null float64
12  momentum_3             360752 non-null float64
13  momentum_6             360752 non-null float64
14  momentum_9             360752 non-null float64
15  momentum_12            360752 non-null float64
16  momentum_3_12          360752 non-null float64
17  year                   360752 non-null int64
18  month                  360752 non-null int64
19  return_1m_t-1          358914 non-null float64
20  return_1m_t-2          357076 non-null float64
21  return_1m_t-3          355238 non-null float64
22  return_1m_t-4          353400 non-null float64
23  return_1m_t-5          351562 non-null float64
24  return_1m_t-6          349724 non-null float64
dtypes: float64(23), int64(2)
memory usage: 78.3+ MB
```

1.9 Target: Holding Period Returns

Similarly, to compute returns for various holding periods, we use the normalized period returns computed previously and shift them back to align them with the current financial features

```
[31]: for t in [1,2,3,6,12]:
        data[f'target_{t}m'] = data.groupby(level='ticker')[f'return_{t}m'].
        ↪shift(-t)
```

```
[32]: cols = ['target_1m',
              'target_2m',
              'target_3m',
              'return_1m',
              'return_2m',
              'return_3m',
              'return_1m_t-1',
              'return_1m_t-2',
              'return_1m_t-3']

data[cols].dropna().sort_index().head(10)
```

```
[32]:
```

		target_1m	target_2m	target_3m	return_1m	return_2m	\
ticker	date						
A	2001-04-30	-0.140220	-0.087246	-0.098192	0.269444	0.040966	
	2001-05-31	-0.031008	-0.076414	-0.075527	-0.140220	0.044721	
	2001-06-30	-0.119692	-0.097014	-0.155847	-0.031008	-0.087246	
	2001-07-31	-0.073750	-0.173364	-0.080114	-0.119692	-0.076414	
	2001-08-31	-0.262264	-0.083279	0.009593	-0.073750	-0.097014	
	2001-09-30	0.139130	0.181052	0.134010	-0.262264	-0.173364	
	2001-10-31	0.224517	0.131458	0.108697	0.139130	-0.083279	
	2001-11-30	0.045471	0.054962	0.045340	0.224517	0.181052	
	2001-12-31	0.064539	0.045275	0.070347	0.045471	0.131458	
	2002-01-31	0.026359	0.073264	-0.003306	0.064539	0.054962	

		return_3m	return_1m_t-1	return_1m_t-2	return_1m_t-3
ticker	date				
A	2001-04-30	-0.105747	-0.146389	-0.329564	-0.003653
	2001-05-31	-0.023317	0.269444	-0.146389	-0.329564
	2001-06-30	0.018842	-0.140220	0.269444	-0.146389
	2001-07-31	-0.098192	-0.031008	-0.140220	0.269444
	2001-08-31	-0.075527	-0.119692	-0.031008	-0.140220
	2001-09-30	-0.155847	-0.073750	-0.119692	-0.031008
	2001-10-31	-0.080114	-0.262264	-0.073750	-0.119692
	2001-11-30	0.009593	0.139130	-0.262264	-0.073750
	2001-12-31	0.134010	0.224517	0.139130	-0.262264
	2002-01-31	0.108697	0.045471	0.224517	0.139130

```
[33]: data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
MultiIndex: 360752 entries, ('A', Timestamp('2001-01-31 00:00:00', freq='M')) to
('ZUMZ', Timestamp('2018-03-31 00:00:00', freq='M'))
Data columns (total 30 columns):
#   Column                Non-Null Count  Dtype
---  -
0   return_1m             360752 non-null float64
1   return_2m             360752 non-null float64
2   return_3m             360752 non-null float64
3   return_6m             360752 non-null float64
4   return_9m             360752 non-null float64
5   return_12m            360752 non-null float64
6   Mkt-RF                360752 non-null float64
7   SMB                   360752 non-null float64
8   HML                   360752 non-null float64
9   RMW                   360752 non-null float64
10  CMA                    360752 non-null float64
11  momentum_2            360752 non-null float64
12  momentum_3            360752 non-null float64
13  momentum_6            360752 non-null float64
14  momentum_9            360752 non-null float64
15  momentum_12           360752 non-null float64
16  momentum_3_12         360752 non-null float64
17  year                   360752 non-null int64
18  month                  360752 non-null int64
19  return_1m_t-1         358914 non-null float64
20  return_1m_t-2         357076 non-null float64
21  return_1m_t-3         355238 non-null float64
22  return_1m_t-4         353400 non-null float64
23  return_1m_t-5         351562 non-null float64
24  return_1m_t-6         349724 non-null float64
25  target_1m             358914 non-null float64
26  target_2m             357076 non-null float64
27  target_3m             355238 non-null float64
28  target_6m             349724 non-null float64
29  target_12m            338696 non-null float64
dtypes: float64(28), int64(2)
memory usage: 92.1+ MB

```

1.10 Create age proxy

We use quintiles of IPO year as a proxy for company age.

```

[34]: data = (data
          .join(pd.qcut(stocks.ipoyear, q=5, labels=list(range(1, 6))))
          .astype(float)
          .fillna(0)
          .astype(int)

```

```

        .to_frame('age'))
data.age = data.age.fillna(-1)

```

1.11 Create dynamic size proxy

We use the marketcap information from the NASDAQ ticker info to create a size proxy.

```
[35]: stocks.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Index: 2412 entries, A to ZUMZ
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   marketcap    2407 non-null   float64
1   ipoyear      1065 non-null   float64
2   sector       2372 non-null   object
dtypes: float64(2), object(1)
memory usage: 139.9+ KB

```

Market cap information is tied to current prices. We create an adjustment factor to have the values reflect lower historical prices for each individual stock:

```
[36]: size_factor = (monthly_prices
                    .loc[data.index.get_level_values('date').unique(),
                        data.index.get_level_values('ticker').unique()]
                    .sort_index(ascending=False)
                    .pct_change()
                    .fillna(0)
                    .add(1)
                    .cumprod())
size_factor.info()
```

```

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 207 entries, 2018-03-31 to 2001-01-31
Columns: 1838 entries, A to ZUMZ
dtypes: float64(1838)
memory usage: 2.9 MB

```

```
[37]: msize = (size_factor
              .mul(stocks
                  .loc[size_factor.columns, 'marketcap']))
              .dropna(axis=1, how='all')
```

1.11.1 Create Size indicator as deciles per period

Compute size deciles per month:

```
[38]: data['msize'] = (msize
                    .apply(lambda x: pd.qcut(x, q=10, labels=list(range(1, 11)))
                        .astype(int), axis=1)
                    .stack()
                    .swaplevel())
data.msize = data.msize.fillna(-1)
```

1.12 Combine data

```
[39]: data = data.join(stocks[['sector']])
data.sector = data.sector.fillna('Unknown')
```

```
[40]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 360752 entries, ('A', Timestamp('2001-01-31 00:00:00', freq='M')) to
('ZUMZ', Timestamp('2018-03-31 00:00:00', freq='M'))
Data columns (total 33 columns):
#   Column                Non-Null Count  Dtype
---  -
0   return_1m             360752 non-null  float64
1   return_2m             360752 non-null  float64
2   return_3m             360752 non-null  float64
3   return_6m             360752 non-null  float64
4   return_9m             360752 non-null  float64
5   return_12m            360752 non-null  float64
6   Mkt-RF                360752 non-null  float64
7   SMB                   360752 non-null  float64
8   HML                   360752 non-null  float64
9   RMW                   360752 non-null  float64
10  CMA                   360752 non-null  float64
11  momentum_2            360752 non-null  float64
12  momentum_3            360752 non-null  float64
13  momentum_6            360752 non-null  float64
14  momentum_9            360752 non-null  float64
15  momentum_12           360752 non-null  float64
16  momentum_3_12         360752 non-null  float64
17  year                  360752 non-null  int64
18  month                 360752 non-null  int64
19  return_1m_t-1         358914 non-null  float64
20  return_1m_t-2         357076 non-null  float64
21  return_1m_t-3         355238 non-null  float64
22  return_1m_t-4         353400 non-null  float64
23  return_1m_t-5         351562 non-null  float64
24  return_1m_t-6         349724 non-null  float64
25  target_1m             358914 non-null  float64
26  target_2m             357076 non-null  float64
```

```

27 target_3m      355238 non-null float64
28 target_6m      349724 non-null float64
29 target_12m     338696 non-null float64
30 age            360752 non-null int64
31 msize          360752 non-null float64
32 sector         360752 non-null object
dtypes: float64(29), int64(3), object(1)
memory usage: 100.4+ MB

```

1.13 Store data

We will use the data again in several later chapters, starting in [Chapter 7 on Linear Models](#).

```

[41]: with pd.HDFStore(DATA_STORE) as store:
        store.put('engineered_features', data.sort_index().loc[idx[:, :
        ↪datetime(2018, 3, 1)], :])
        print(store.info())

```

```

<class 'pandas.io.pytables.HDFStore'>
File path: ../data/assets.h5
/engineered_features      frame      (shape->[358914,33])
/quandl/wiki/prices       frame      (shape->[15389314,12])
/quandl/wiki/stocks       frame      (shape->[1,2])
/sp500/fred               frame      (shape->[2609,1])
/sp500/sp500_stooq       frame      (shape->[17700,5])
/sp500/stocks            frame      (shape->[1,7])
/sp500/stooq            frame      (shape->[17700,5])
/stooq/jp/tse/stocks/prices frame_table (typ->appendable_multi,
nrows->10283141,ncols->7,indexers->[index],dc->[date,ticker])
/stooq/jp/tse/stocks/tickers frame_table
(typ->appendable,nrows->3732,ncols->2,indexers->[index],dc->[])
/stooq/us/nasdaq/etfs/prices frame_table (typ->appendable_multi,
nrows->359912,ncols->7,indexers->[index],dc->[date,ticker])
/stooq/us/nasdaq/etfs/tickers frame_table
(typ->appendable,nrows->171,ncols->2,indexers->[index],dc->[])
/stooq/us/nasdaq/stocks/prices frame_table (typ->appendable_multi,
nrows->6415760,ncols->7,indexers->[index],dc->[date,ticker])
/stooq/us/nasdaq/stocks/tickers frame_table
(typ->appendable,nrows->3570,ncols->2,indexers->[index],dc->[])
/stooq/us/nyse/etfs/prices frame_table (typ->appendable_multi,
nrows->2435526,ncols->7,indexers->[index],dc->[date,ticker])
/stooq/us/nyse/etfs/tickers frame_table
(typ->appendable,nrows->1023,ncols->2,indexers->[index],dc->[])
/stooq/us/nyse/stocks/prices frame_table (typ->appendable_multi,
nrows->7983429,ncols->7,indexers->[index],dc->[date,ticker])
/stooq/us/nyse/stocks/tickers frame_table
(typ->appendable,nrows->3969,ncols->2,indexers->[index],dc->[])
/stooq/us/nysemkt/stocks/prices frame_table (typ->appendable_multi,

```



```
nrows->744452,ncols->7,indexers->[index],dc->[date,ticker])
/stooq/us/nysemkt/stocks/tickers          frame_table
(typ->appendable,nrows->298,ncols->2,indexers->[index],dc->[])
/us_equities/stocks                        frame      (shape->[6834,6])
```

1.14 Create Dummy variables

For most models, we need to encode categorical variables as ‘dummies’ (one-hot encoding):

```
[42]: dummy_data = pd.get_dummies(data,
                                   columns=['year','month','msize','age','sector'],
                                   prefix=['year','month','msize','age',''],
                                   prefix_sep=['_','_','_','_','_'])
dummy_data = dummy_data.rename(columns={c:c.replace('.0','') for c in
    ↪ dummy_data.columns})
dummy_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
MultiIndex: 360752 entries, ('A', Timestamp('2001-01-31 00:00:00', freq='M')) to
('ZUMZ', Timestamp('2018-03-31 00:00:00', freq='M'))
```

```
Data columns (total 88 columns):
```

#	Column	Non-Null Count	Dtype
0	return_1m	360752 non-null	float64
1	return_2m	360752 non-null	float64
2	return_3m	360752 non-null	float64
3	return_6m	360752 non-null	float64
4	return_9m	360752 non-null	float64
5	return_12m	360752 non-null	float64
6	Mkt-RF	360752 non-null	float64
7	SMB	360752 non-null	float64
8	HML	360752 non-null	float64
9	RMW	360752 non-null	float64
10	CMA	360752 non-null	float64
11	momentum_2	360752 non-null	float64
12	momentum_3	360752 non-null	float64
13	momentum_6	360752 non-null	float64
14	momentum_9	360752 non-null	float64
15	momentum_12	360752 non-null	float64
16	momentum_3_12	360752 non-null	float64
17	return_1m_t-1	358914 non-null	float64
18	return_1m_t-2	357076 non-null	float64
19	return_1m_t-3	355238 non-null	float64
20	return_1m_t-4	353400 non-null	float64
21	return_1m_t-5	351562 non-null	float64
22	return_1m_t-6	349724 non-null	float64
23	target_1m	358914 non-null	float64
24	target_2m	357076 non-null	float64

25	target_3m	355238	non-null	float64
26	target_6m	349724	non-null	float64
27	target_12m	338696	non-null	float64
28	year_2001	360752	non-null	uint8
29	year_2002	360752	non-null	uint8
30	year_2003	360752	non-null	uint8
31	year_2004	360752	non-null	uint8
32	year_2005	360752	non-null	uint8
33	year_2006	360752	non-null	uint8
34	year_2007	360752	non-null	uint8
35	year_2008	360752	non-null	uint8
36	year_2009	360752	non-null	uint8
37	year_2010	360752	non-null	uint8
38	year_2011	360752	non-null	uint8
39	year_2012	360752	non-null	uint8
40	year_2013	360752	non-null	uint8
41	year_2014	360752	non-null	uint8
42	year_2015	360752	non-null	uint8
43	year_2016	360752	non-null	uint8
44	year_2017	360752	non-null	uint8
45	year_2018	360752	non-null	uint8
46	month_1	360752	non-null	uint8
47	month_2	360752	non-null	uint8
48	month_3	360752	non-null	uint8
49	month_4	360752	non-null	uint8
50	month_5	360752	non-null	uint8
51	month_6	360752	non-null	uint8
52	month_7	360752	non-null	uint8
53	month_8	360752	non-null	uint8
54	month_9	360752	non-null	uint8
55	month_10	360752	non-null	uint8
56	month_11	360752	non-null	uint8
57	month_12	360752	non-null	uint8
58	msize_-1	360752	non-null	uint8
59	msize_1	360752	non-null	uint8
60	msize_2	360752	non-null	uint8
61	msize_3	360752	non-null	uint8
62	msize_4	360752	non-null	uint8
63	msize_5	360752	non-null	uint8
64	msize_6	360752	non-null	uint8
65	msize_7	360752	non-null	uint8
66	msize_8	360752	non-null	uint8
67	msize_9	360752	non-null	uint8
68	msize_10	360752	non-null	uint8
69	age_0	360752	non-null	uint8
70	age_1	360752	non-null	uint8
71	age_2	360752	non-null	uint8
72	age_3	360752	non-null	uint8

73	age_4	360752	non-null	uint8
74	age_5	360752	non-null	uint8
75	Basic Industries	360752	non-null	uint8
76	Capital Goods	360752	non-null	uint8
77	Consumer Durables	360752	non-null	uint8
78	Consumer Non-Durables	360752	non-null	uint8
79	Consumer Services	360752	non-null	uint8
80	Energy	360752	non-null	uint8
81	Finance	360752	non-null	uint8
82	Health Care	360752	non-null	uint8
83	Miscellaneous	360752	non-null	uint8
84	Public Utilities	360752	non-null	uint8
85	Technology	360752	non-null	uint8
86	Transportation	360752	non-null	uint8
87	Unknown	360752	non-null	uint8

dtypes: float64(28), uint8(60)

memory usage: 107.2+ MB