In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('sample_prices.csv')
```

In [3]:
```
df
```

Out[3]:

	BLUE	ORANGE
0	8.7000	10.6600
1	8.9055	11.0828
2	8.7113	10.7100
3	8.4346	11.5907
4	8.7254	12.1070
5	9.0551	11.7876
6	8.9514	11.2078
7	9.2439	12.5192
8	9.1276	13.3624
9	9.3976	14.4080
10	9.4554	11.9837
11	9.5704	12.2718
12	9.7728	11.5892

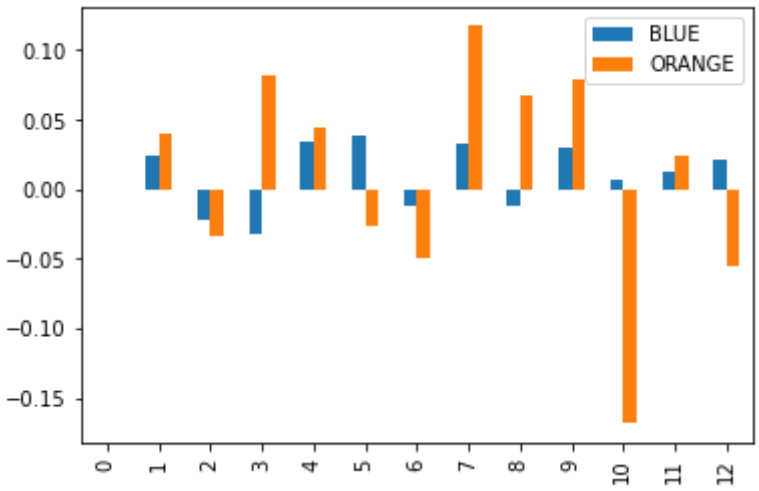
In [4]:
```
df.pct_change().dropna
```

Out[4]:
```
<bound method DataFrame.dropna of         BLUE    ORANGE
0        NaN       NaN
1   0.023621  0.039662
2  -0.021807 -0.033638
3  -0.031763  0.082232
4   0.034477  0.044544
5   0.037786 -0.026381
6  -0.011452 -0.049187
7   0.032676  0.117008
8  -0.012581  0.067353
9   0.029581  0.078249
10  0.006151 -0.168261
11  0.012162  0.024041
12  0.021149 -0.055623>
```

In [5]:
```
returns = df.pct_change()
```

In [6]:
```
returns.plot.bar()
```

Out[6]: <AxesSubplot:>



In [7]:
```
returns.std()
```

Out[7]:
```
BLUE      0.023977
ORANGE    0.079601
dtype: float64
```

In [8]:
```
returns.mean()
```

Out[8]:
```
BLUE      0.01
ORANGE    0.01
dtype: float64
```

In [9]:
```
returns
```

Out[9]:

	BLUE	ORANGE
0	NaN	NaN
1	0.023621	0.039662
2	-0.021807	-0.033638
3	-0.031763	0.082232
4	0.034477	0.044544
5	0.037786	-0.026381
6	-0.011452	-0.049187
7	0.032676	0.117008
8	-0.012581	0.067353
9	0.029581	0.078249
10	0.006151	-0.168261
11	0.012162	0.024041
12	0.021149	-0.055623

In [10]:
```
# multiple each columns: for compound values
import pandas as pd
import numpy as np
np.prod(returns+1)
```

Out[10]:
```
BLUE      1.123310
ORANGE    1.087167
dtype: float64
```

In [11]:
```
# annualization
rm = 0.01
(1+rm)**12
```

Out[11]:
```
1.1268250301319698
```

In [12]:
```
# sharp ratio
# return minues risk free rate / volatity
returns.dropna()
```

Out[12]:

	BLUE	ORANGE
1	0.023621	0.039662
2	-0.021807	-0.033638
3	-0.031763	0.082232
4	0.034477	0.044544
5	0.037786	-0.026381
6	-0.011452	-0.049187
7	0.032676	0.117008
8	-0.012581	0.067353
9	0.029581	0.078249
10	0.006151	-0.168261
11	0.012162	0.024041
12	0.021149	-0.055623

In [13]:
```
# calculate variance
returns.std()
```

Out[13]:
```
BLUE      0.023977
ORANGE    0.079601
dtype: float64
```

In [14]:
```
deviations = ((returns - returns.mean())**2)/(returns.shape[0])
```

In [15]:
```
deviations
```

Out[15]:

	BLUE	ORANGE
0	NaN	NaN
1	1.427120e-05	0.000068
2	7.782029e-05	0.000146
3	1.341669e-04	0.000401
4	4.608688e-05	0.000092
5	5.939074e-05	0.000102
6	3.539918e-05	0.000269
7	3.955580e-05	0.000881
8	3.922383e-05	0.000253
9	2.949260e-05	0.000358
10	1.139841e-06	0.002444
11	3.597065e-07	0.000015
12	9.560915e-06	0.000331

In [16]:
```
data = pd.read_csv('Portfolios_Formed_on_ME_monthly_EW.csv',header=0)
```

In [17]:
```
data
```

Out[17]:

	Unnamed: 0	<= 0	Lo 30	Med 40	Hi 30	Lo 20	Qnt 2	Qnt 3	Qnt 4	Hi 20	Lo 10	Dec 2
0	192607	-99.99	-0.43	1.52	2.68	-0.57	0.59	1.60	1.47	3.33	-1.45	0.29
1	192608	-99.99	3.90	3.04	2.09	3.84	3.59	3.71	1.61	2.33	5.12	2.59
2	192609	-99.99	-1.08	-0.54	0.16	-0.48	-1.40	0.00	-0.50	-0.09	0.93	-1.87
3	192610	-99.99	-3.32	-3.52	-3.06	-3.29	-4.10	-2.89	-3.36	-2.95	-4.84	-1.77
4	192611	-99.99	-0.46	3.82	3.09	-0.55	2.18	3.41	3.39	3.16	-0.78	-0.32
...	...	...	...	...	...	...	...	...	...	...	...	...
1105	201808	-99.99	3.47	4.04	2.87	3.09	5.05	3.90	3.54	2.49	2.41	5.07
1106	201809	-99.99	-2.24	-1.85	0.08	-2.04	-2.38	-2.48	-0.74	0.19	-1.68	-3.08
1107	201810	-99.99	-10.76	-10.88	-7.63	-10.52	-11.74	-10.55	-9.45	-7.41	-10.02	-11.98
1108	201811	-99.99	-2.08	2.18	2.19	-2.78	1.69	1.46	2.62	2.49	-3.65	-0.23
1109	201812	-99.99	-14.28	-12.41	-9.76	-14.77	-12.44	-12.22	-11.34	-9.21	-15.31	-13.19

1110 rows × 20 columns

In [18]:
```
pofio = pd.read_csv('Portfolios_Formed_on_ME_monthly_EW.csv',header=0,index
```

In [19]:
```
pofio
```

Out[19]:

	<= 0	Lo 30	Med 40	Hi 30	Lo 20	Qnt 2	Qnt 3	Qnt 4	Hi 20	Lo 10	Dec 2	Dec 3	Dec
192607	NaN	-0.43	1.52	2.68	-0.57	0.59	1.60	1.47	3.33	-1.45	0.29	-0.15	1.
192608	NaN	3.90	3.04	2.09	3.84	3.59	3.71	1.61	2.33	5.12	2.59	4.03	3.
192609	NaN	-1.08	-0.54	0.16	-0.48	-1.40	0.00	-0.50	-0.09	0.93	-1.87	-2.27	-0.
192610	NaN	-3.32	-3.52	-3.06	-3.29	-4.10	-2.89	-3.36	-2.95	-4.84	-1.77	-3.36	-4.
192611	NaN	-0.46	3.82	3.09	-0.55	2.18	3.41	3.39	3.16	-0.78	-0.32	-0.29	4.
...	...	...	...	...	...	...	...	...	...	...	...	...	...
201808	NaN	3.47	4.04	2.87	3.09	5.05	3.90	3.54	2.49	2.41	5.07	5.30	4.
201809	NaN	-2.24	-1.85	0.08	-2.04	-2.38	-2.48	-0.74	0.19	-1.68	-3.08	-3.22	-1.
201810	NaN	-10.76	-10.88	-7.63	-10.52	-11.74	-10.55	-9.45	-7.41	-10.02	-11.98	-11.89	-11.
201811	NaN	-2.08	2.18	2.19	-2.78	1.69	1.46	2.62	2.49	-3.65	-0.23	1.23	2.
201812	NaN	-14.28	-12.41	-9.76	-14.77	-12.44	-12.22	-11.34	-9.21	-15.31	-13.19	-11.94	-13.

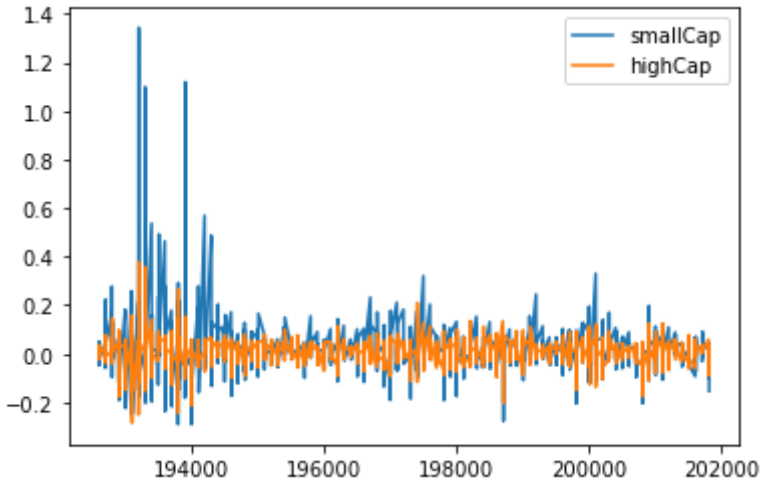
1110 rows × 19 columns

In [20]:
```
selection = pofio[['Lo 10','Hi 10']]
```

In [21]:
```
selection.columns = ['smallCap','highCap']
selection = selection/100
```

In [22]:
```
selection.plot()
```

Out[22]: <AxesSubplot:>



In [23]:
```
selection
```

Out[23]:

	smallCap	highCap
192607	-0.0145	0.0329
192608	0.0512	0.0370
192609	0.0093	0.0067
192610	-0.0484	-0.0243
192611	-0.0078	0.0270
...	...	...
201808	0.0241	0.0234
201809	-0.0168	0.0087
201810	-0.1002	-0.0657
201811	-0.0365	0.0253
201812	-0.1531	-0.0890

1110 rows × 2 columns

In [24]:
```
# Calculate ann
ann_value = selection.std()*np.sqrt(12)
```

In [25]:
```
# risk/return ratio
ann_value
```

Out[25]:
```
smallCap    0.368193
highCap     0.186716
dtype: float64
```

In [26]:
```
# return
# return per month times months
number_of_month = selection.shape[0]
ann_return = ((selection+1).prod()**(1/number_of_month))**12-1
```

In [27]:
```
selection.shape
```

Out[27]:
```
(1110, 2)
```

In [28]:
```
ann_return
```

Out[28]:
```
smallCap    0.167463
highCap     0.092810
dtype: float64
```

In [29]:
```
# return/risk ratio
risk_return_ratio = ann_return/ann_value
risk_return_ratio
```

Out[29]:
```
smallCap    0.454825
highCap     0.497063
dtype: float64
```

In [30]:
```
# sharp ratio: compared to risk free rate
risk_free = 0.03
sharp_ratio = (ann_return-risk_free)/ann_value
```

In [31]:
```
sharp_ratio
```

Out[31]:
```
smallCap    0.373346
highCap     0.336392
dtype: float64
```

In [32]:
```
# calculate the max drawdown
# maximum loss from buying at the peak and sell at the lowest price
```

Capture Drawdowns

In [33]:
```
import pandas as pd
```

In [34]:
```
data1 = pd.read_csv('Portfolios_Formed_on_ME_monthly_EW.csv',header=0,index
```

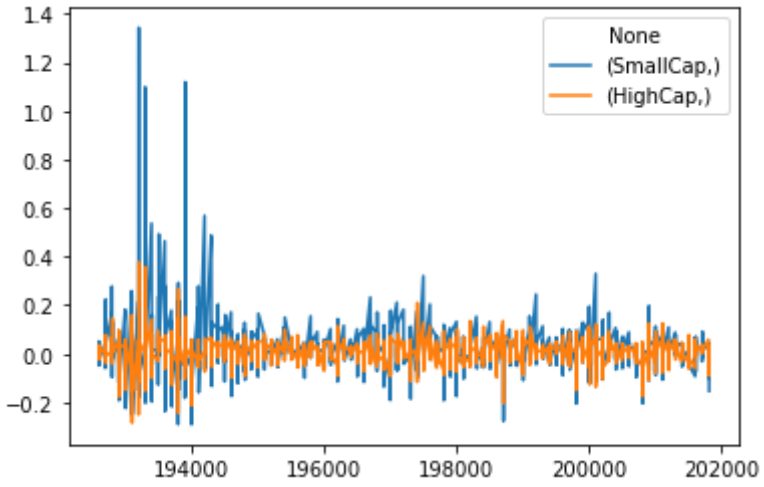
In [35]:
```
draw = data1[['Lo 10','Hi 10']]
```

In [36]:
```
draw.columns = [['SmallCap','HighCap']]
```

In [37]:
```
draw = draw/100
```

In [38]:
```
draw.plot.line()
```

Out[38]: <AxesSubplot:>



In [39]:
```
draw.index
```

Out[39]:
```
Int64Index([192607, 192608, 192609, 192610, 192611, 192612, 192701, 192702,
            192703, 192704,
            ...
            201803, 201804, 201805, 201806, 201807, 201808, 201809, 201810,
            201811, 201812],
           dtype='int64', length=1110)
```

In [40]:
```
draw.index = pd.to_datetime(draw.index, format='%Y%M')
```

In [41]:
```
draw.index = draw.index.to_period('M')
draw.head()
```

Out[41]:

	SmallCap	HighCap
1926-01	-0.0145	0.0329
1926-01	0.0512	0.0370
1926-01	0.0093	0.0067
1926-01	-0.0484	-0.0243
1926-01	-0.0078	0.0270

In [42]:
```
# compute weath index
# compute previous peaks
# compute drawdown which is the percentgae of previous peak
wealth_index = 1000*(1+draw['HighCap']).cumprod()
```

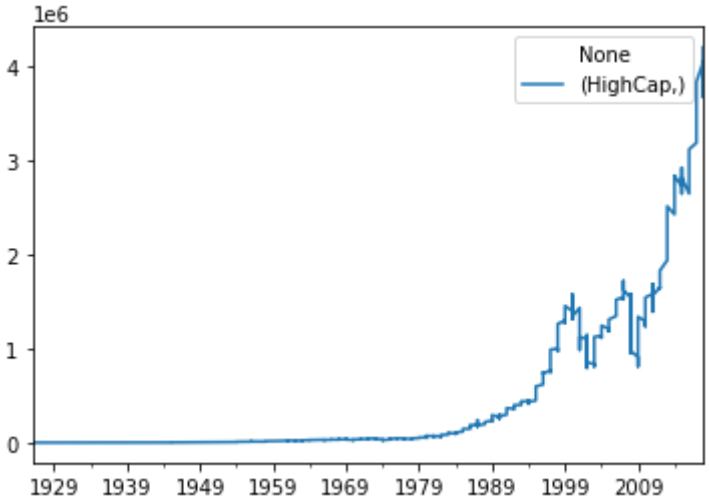
In [43]:
```
wealth_index.head()
```

Out[43]:

	HighCap
1926-01	1032.900000
1926-01	1071.117300
1926-01	1078.293786
1926-01	1052.091247
1926-01	1080.497711

In [44]:
```
wealth_index.plot.line()
```

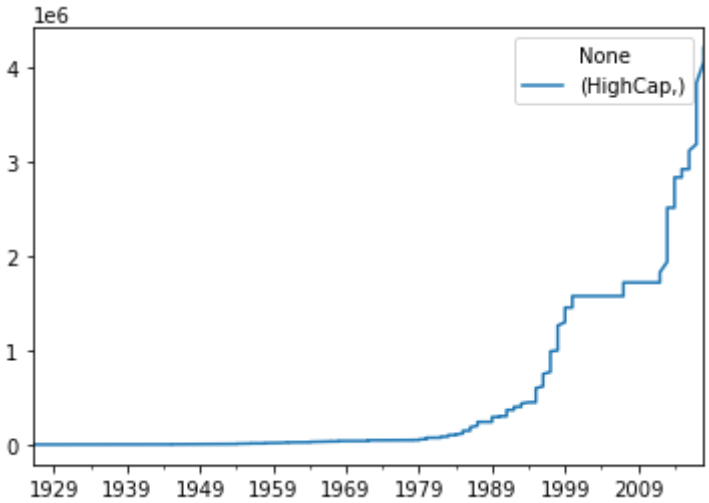
Out[44]: <AxesSubplot:>



In [45]:
```
Previous_peaks = wealth_index.cummax()
```

In [46]:
```
Previous_peaks.plot()
```

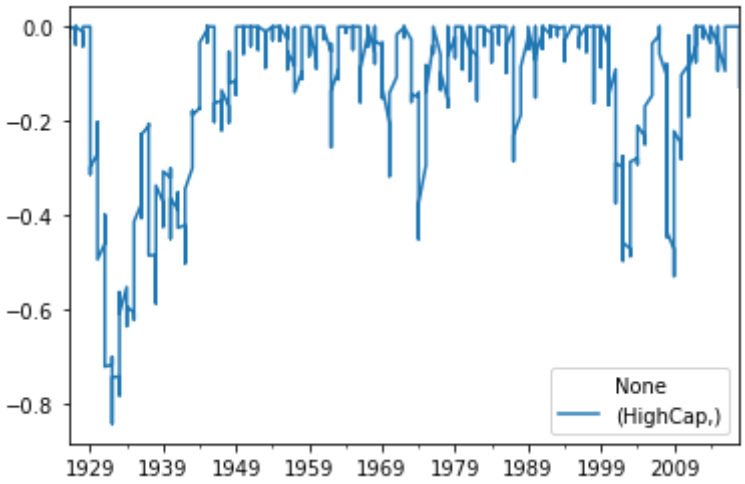
Out[46]: <AxesSubplot:>



In [47]:
```
drawdown = (wealth_index - Previous_peaks)/Previous_peaks
```

In [48]:
```
drawdown.plot.line()
```

Out[48]: <AxesSubplot:>



In [49]:
```
drawdown.head()
```

Out[49]:

	HighCap
1926-01	0.0000
1926-01	0.0000
1926-01	0.0000
1926-01	-0.0243
1926-01	0.0000

In [50]:
```
drawdown.min()
```

Out[50]:
```
HighCap   -0.840038
dtype: float64
```

In [51]:
```
drawdown['1975':].idxmin()
```

Out[51]:
```
HighCap    2009-01
dtype: period[M]
```

In [52]:
```
drawdown.idxmin()
```

Out[52]:
```
HighCap    1932-01
dtype: period[M]
```

In [53]:
```
def calculate_wealth(return_series: pd.Series):
    '''
    Takes a time series of asset returns
    Computes and returns a DataFrame that contains:
    the wealth index
    the previous peaks
    percentage drawdowns

    '''
    wealth_index = 1000*(1+return_series).cumprod()
    previous_peaks = wealth_index.cummax()
    drawdown_pet = (wealth_index - previous_peaks)/previous_peaks


    return pd.DataFrame({
        'Wealth': wealth_index,
        'Peaks': previous_peaks,
        'Drawdown':drawdown_pet
    })
```

In [56]:
```
(draw['HighCap']).info
```

Out[56]:
```
<bound method DataFrame.info of         HighCap
1926-01  0.0329
1926-01  0.0370
1926-01  0.0067
1926-01 -0.0243
1926-01  0.0270
...          ...
2018-01  0.0234
2018-01  0.0087
2018-01 -0.0657
2018-01  0.0253
2018-01 -0.0890

[1110 rows x 1 columns]>
```

In [57]:
```
calculate_wealth(draw['SmallCap'])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/cf/s1wshv2j2bz4cbfxfg5qgrgm0000gn/T/ipykernel_56633/429810598.py in <module>
----> 1 calculate_wealth(draw['SmallCap'])

/var/folders/cf/s1wshv2j2bz4cbfxfg5qgrgm0000gn/T/ipykernel_56633/1661038896.py in calculate_wealth(return_series)
     13
     14
---> 15     return pd.DataFrame({
     16         'Wealth': wealth_index,
     17         'Peaks': previous_peaks,

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/frame.py in __init__(self, data, index, columns, dtype, copy)
    612         elif isinstance(data, dict):
    613             # GH#38939 de facto copy defaults to False only in non-dict cases
--> 614             mgr = dict_to_mgr(data, index, columns, dtype=dtype, copy=copy, typ=manager)
    615         elif isinstance(data, ma.MaskedArray):
    616             import numpy.ma.mrecords as mrecords

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/internals/construction.py in dict_to_mgr(data, index, columns, dtype, typ, copy)
    462         # TODO: can we get rid of the dt64tz special case above?
    463
--> 464     return arrays_to_mgr(
    465         arrays, data_names, index, columns, dtype=dtype, typ=typ, consolidate=copy
    466     )

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/internals/construction.py in arrays_to_mgr(arrays, arr_names, index, columns, dtype, verify_integrity, typ, consolidate)
    117         # figure out the index, if necessary
    118         if index is None:
--> 119             index = _extract_index(arrays)
    120         else:
    121             index = ensure_index(index)

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/internals/construction.py in _extract_index(data)
    623
    624         if not indexes and not raw_lengths:
--> 625             raise ValueError("If using all scalar values, you must pass an index")
    626
    627         if have_series:

ValueError: If using all scalar values, you must pass an index
```

In [103]:
```
draw
```

Out[103]:

	SmallCap	HighCap
1926-01	-0.0145	0.0329
1926-01	0.0512	0.0370
1926-01	0.0093	0.0067
1926-01	-0.0484	-0.0243
1926-01	-0.0078	0.0270
...	...	...
2018-01	0.0241	0.0234
2018-01	-0.0168	0.0087
2018-01	-0.1002	-0.0657
2018-01	-0.0365	0.0253
2018-01	-0.1531	-0.0890

1110 rows × 2 columns

Building modules

In [69]:
```
import pandas as pd
```

In [86]:
```
import Hello as h
```

In [125]:
```
%load_ext autoreload
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload

[autoreload of edhec_risk_kit failed: Traceback (most recent call last):
  File "/Users/mac/opt/anaconda3/lib/python3.9/site-packages/IPython/exte
nsions/autoreload.py", line 245, in check
    superreload(m, reload, self.old_objects)
  File "/Users/mac/opt/anaconda3/lib/python3.9/site-packages/IPython/exte
nsions/autoreload.py", line 394, in superreload
    module = reload(module)
  File "/Users/mac/opt/anaconda3/lib/python3.9/imp.py", line 314, in relo
ad
    return importlib.reload(module)
  File "/Users/mac/opt/anaconda3/lib/python3.9/importlib/__init__.py", li
ne 169, in reload
    _bootstrap._exec(spec, module)
  File "<frozen importlib._bootstrap>", line 613, in _exec
  File "<frozen importlib._bootstrap_external>", line 846, in exec_module
  File "<frozen importlib._bootstrap_external>", line 983, in get_code
  File "<frozen importlib._bootstrap_external>", line 913, in source_to_c
ode
  File "<frozen importlib._bootstrap>", line 228, in _call_with_frames_re
moved
  File "/Users/mac/Coursera(python investment)/edhec_risk_kit.py", line 1
9
    'Drawdown':drawdown_pet}, index = 1)
                              ^
SyntaxError: invalid character ', ' (U+FF0C)
]
```

In [126]:
```
%autoreload 2
```

In [127]:
```
h.message
```

Out[127]:
```
'Hello Jane and Jis'
```

In [128]:
```
import edhec_risk_kit as erk
```

In [129]:
```
returnS = erk.get_ffme_returns()
```

In [130]:
```
returnS
```

Out[130]:

	SmallCap	LargeCap
1926-01	-0.0145	0.0329
1926-01	0.0512	0.0370
1926-01	0.0093	0.0067
1926-01	-0.0484	-0.0243
1926-01	-0.0078	0.0270
...	...	...
2018-01	0.0241	0.0234
2018-01	-0.0168	0.0087
2018-01	-0.1002	-0.0657
2018-01	-0.0365	0.0253
2018-01	-0.1531	-0.0890

1110 rows × 2 columns

In [131]:
```
type(draw['SmallCap'])
```

Out[131]:
```
pandas.core.frame.DataFrame
```

In [134]:
```
x = draw['SmallCap']
```

In [135]:
```
erk.calculate_wealth(x)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/cf/s1wshv2j2bz4cbfxfg5qgrgm0000gn/T/ipykernel_56633/1749938188.py in <module>
----> 1 erk.calculate_wealth(x)

~/Coursera(python investment)/edhec_risk_kit.py in calculate_wealth(x)
     15
     16
---> 17     return pd.DataFrame({'Wealth': wealth_index,
     18                          'Peaks': previous_peaks,
     19                          'Drawdown':drawdown_pet}, index = 1)

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/frame.py in __init__(self, data, index, columns, dtype, copy)
    612         elif isinstance(data, dict):
    613             # GH#38939 de facto copy defaults to False only in non-dict cases
--> 614             mgr = dict_to_mgr(data, index, columns, dtype=dtype, copy=copy, typ=manager)
    615         elif isinstance(data, ma.MaskedArray):
    616             import numpy.ma.mrecords as mrecords

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/internals/construction.py in dict_to_mgr(data, index, columns, dtype, typ, copy)
    462         # TODO: can we get rid of the dt64tz special case above?
    463
--> 464     return arrays_to_mgr(
    465         arrays, data_names, index, columns, dtype=dtype, typ=typ, consolidate=copy
    466     )

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/internals/construction.py in arrays_to_mgr(arrays, arr_names, index, columns, dtype, verify_integrity, typ, consolidate)
    117         # figure out the index, if necessary
    118         if index is None:
--> 119             index = _extract_index(arrays)
    120         else:
    121             index = ensure_index(index)

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/internals/construction.py in _extract_index(data)
    623
    624         if not indexes and not raw_lengths:
--> 625             raise ValueError("If using all scalar values, you must pass an index")
    626
    627         if have_series:

ValueError: If using all scalar values, you must pass an index
```

In [114]:
```
draw['SmallCap']
```

Out[114]:

	SmallCap
1926-01	-0.0145
1926-01	0.0512
1926-01	0.0093
1926-01	-0.0484
1926-01	-0.0078
...	...
2018-01	0.0241
2018-01	-0.0168
2018-01	-0.1002
2018-01	-0.0365
2018-01	-0.1531

1110 rows × 1 columns

Deviations from Normality

In [47]:
```
%load_ext autoreload
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [57]:
```
%autoreload 2
```

In [7]:
```
import pandas as pd
import edhec_risk_kit as erk
```

In [9]:
```
hfi = erk.get_hfi_returns()
```

In [10]:
```
hfi.head()
```

Out[10]:

date	Convertible Arbitrage	CTA Global	Distressed Securities	Emerging Markets	Equity Market Neutral	Event Driven	Fixed Income Arbitrage	Global Macro	Long/Short Equity
1997-01	0.0119	0.0393	0.0178	0.0791	0.0189	0.0213	0.0191	0.0573	0.0281
1997-02	0.0123	0.0298	0.0122	0.0525	0.0101	0.0084	0.0122	0.0175	-0.0006
1997-03	0.0078	-0.0021	-0.0012	-0.0120	0.0016	-0.0023	0.0109	-0.0119	-0.0084
1997-04	0.0086	-0.0170	0.0030	0.0119	0.0119	-0.0005	0.0130	0.0172	0.0084
1997-05	0.0156	-0.0015	0.0233	0.0315	0.0189	0.0346	0.0118	0.0108	0.0394

In [13]:
```
x = pd.concat([hfi.mean(),hfi.median(),hfi.mean()>hfi.median()],axis= "colu
```

In [16]:
```
x
```

Out[16]:

	0	1	2
Convertible Arbitrage	0.005508	0.0065	False
CTA Global	0.004074	0.0014	True
Distressed Securities	0.006946	0.0089	False
Emerging Markets	0.006253	0.0096	False
Equity Market Neutral	0.004498	0.0051	False
Event Driven	0.006344	0.0084	False
Fixed Income Arbitrage	0.004365	0.0055	False
Global Macro	0.005403	0.0038	True
Long/Short Equity	0.006331	0.0079	False
Merger Arbitrage	0.005356	0.0060	False
Relative Value	0.005792	0.0067	False
Short Selling	-0.001701	-0.0053	True
Funds Of Funds	0.004262	0.0052	False

In [18]:
```
x.columns = ['mean','median','mean is greater than median']
```

In [19]:
```
x[['mean','median']].plot()
```

Out[19]: <AxesSubplot:>

In [24]:
```
erk.skewness(hfi).sort_values()
```

Out[24]:
```
Fixed Income Arbitrage   -3.940320
Convertible Arbitrage    -2.639592
Equity Market Neutral    -2.124435
Relative Value           -1.815470
Event Driven             -1.409154
Merger Arbitrage         -1.320083
Distressed Securities    -1.300842
Emerging Markets         -1.167067
Long/Short Equity        -0.390227
Funds Of Funds           -0.361783
CTA Global                0.173699
Short Selling             0.767975
Global Macro              0.982922
dtype: float64
```

In [25]:
```
# use built-in to calculate skewness
import scipy.stats
scipy.stats.skew(hfi)
```

Out[25]:
```
array([-2.63959223,  0.17369864, -1.30084204, -1.16706749, -2.12443538,
       -1.40915356, -3.94032029,  0.98292188, -0.39022677, -1.32008333,
       -1.81546975,  0.76797484, -0.36178308])
```

In [27]:
```
hfi.shape
```

Out[27]: (263, 13)

In [39]:
```
import numpy as np
normal_rets = np.random.normal(0,.15,size=(263,1))
```

In [40]:
```
erk.skewness(normal_rets)
```

Out[40]:
```
0.011545384845060541
```

Kurtosis

In [41]:
```
erk.kurtosis(hfi)
```

Out[41]:
```
Convertible Arbitrage     23.280834
CTA Global                 2.952960
Distressed Securities      7.889983
Emerging Markets           9.250788
Equity Market Neutral     17.218555
Event Driven               8.035828
Fixed Income Arbitrage    29.842199
Global Macro               5.741679
Long/Short Equity          4.523893
Merger Arbitrage           8.738950
Relative Value            12.121208
Short Selling              6.117772
Funds Of Funds             7.070153
dtype: float64
```

In [42]:
```
erk.skewness(normal_rets)
```

Out[42]:
```
0.011545384845060541
```

In [43]:
```
scipy.stats.kurtosis(normal_rets)
```

Out[43]:
```
array([-0.17653776])
```

In [45]:
```
scipy.stats.jarque_bera(normal_rets) # JB 检验样本是否正态分布
```

Out[45]:
```
Jarque_beraResult(statistic=0.34736561024183066, pvalue=0.840563477807973
5)
```

In [43]:
```
scipy.stats.kurtosis(normal_rets)
```

Out[43]:
```
array([-0.17653776])
```

In [46]:
```
scipy.stats.jarque_bera(hfi) # JB 检验样本是否正态分布
```

Out[46]:
```
Jarque_beraResult(statistic=25656.585999171326, pvalue=0.0)
```

In [49]:
```
erk.is_normal(normal_rets)
```

Out[49]:
```
True
```

In [50]:
```
erk.is_normal(hfi)
```

Out[50]:
```
False
```

In [51]:
```
hfi.aggregate(erk.is_normal) # aggregate function is to apply function to e
```

Out[51]:
```
Convertible Arbitrage     False
CTA Global                 True
Distressed Securities     False
Emerging Markets          False
Equity Market Neutral     False
Event Driven              False
Fixed Income Arbitrage    False
Global Macro              False
Long/Short Equity         False
Merger Arbitrage          False
Relative Value            False
Short Selling             False
Funds Of Funds            False
dtype: bool
```

In [52]:
```
ffme = erk.get_ffme_returns()
erk.skewness(ffme)
```

Out[52]:
```
SmallCap    4.410739
LargeCap    0.233445
dtype: float64
```

In [53]:
```
erk.kurtosis(ffme)
```

Out[53]:
```
SmallCap    46.845008
LargeCap    10.694654
dtype: float64
```

In [54]:
```
erk.is_normal(ffme)
```

Out[54]:
```
False
```

In [55]:
```
ffme.aggregate(erk.is_normal)
```

Out[55]:
```
SmallCap    False
LargeCap    False
dtype: bool
```

In [ ]:
```
# There are at least four standard methods for calculating Var
# Method 1: Historical (non parametric)
# 2: Variance-covariance(Parametric Gaussian)
# 3: Parametric non gaussian
# 4: Cornish-fisher(semi parametric)
```

In [108]:
```
import pandas as pd
import edhec_risk_kit as erk
%load_ext autoreload
%autoreload 2
%matplotlib inline
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Semideviation

In [63]:
```
hfi.std(ddof=0)
```

Out[63]:
```
Convertible Arbitrage     0.016536
CTA Global                0.023290
Distressed Securities     0.017009
Emerging Markets          0.032476
Equity Market Neutral     0.008115
Event Driven              0.016712
Fixed Income Arbitrage    0.011517
Global Macro              0.014694
Long/Short Equity         0.019897
Merger Arbitrage          0.009600
Relative Value            0.011462
Short Selling             0.047655
Funds Of Funds            0.015536
dtype: float64
```

In [64]:
```
hfi
```

Out[64]:

date	Convertible Arbitrage	CTA Global	Distressed Securities	Emerging Markets	Equity Market Neutral	Event Driven	Fixed Income Arbitrage	Global Macro	Long/Short Equity
1997-01	0.0119	0.0393	0.0178	0.0791	0.0189	0.0213	0.0191	0.0573	0.0281
1997-02	0.0123	0.0298	0.0122	0.0525	0.0101	0.0084	0.0122	0.0175	-0.0006
1997-03	0.0078	-0.0021	-0.0012	-0.0120	0.0016	-0.0023	0.0109	-0.0119	-0.0084
1997-04	0.0086	-0.0170	0.0030	0.0119	0.0119	-0.0005	0.0130	0.0172	0.0084
1997-05	0.0156	-0.0015	0.0233	0.0315	0.0189	0.0346	0.0118	0.0108	0.0394
...	...	...	...	...	...	...	...	...	...
2018-07	0.0021	-0.0058	0.0093	0.0040	-0.0010	0.0055	0.0022	-0.0014	0.0067
2018-08	0.0024	0.0166	0.0002	-0.0277	0.0004	0.0011	0.0017	-0.0007	0.0035
2018-09	0.0034	-0.0054	0.0050	-0.0110	-0.0016	0.0032	0.0036	0.0006	-0.0023
2018-10	-0.0073	-0.0314	-0.0158	-0.0315	-0.0129	-0.0257	-0.0023	-0.0096	-0.0402
2018-11	-0.0068	-0.0053	-0.0193	0.0120	-0.0211	-0.0034	-0.0067	-0.0087	-0.0044

263 rows × 13 columns

In [65]:
```
hfi[hfi<0].std(ddof=0)
```

Out[65]:
```
Convertible Arbitrage     0.019540
CTA Global                0.012443
Distressed Securities     0.015185
Emerging Markets          0.028039
Equity Market Neutral     0.009566
Event Driven              0.015429
Fixed Income Arbitrage    0.017763
Global Macro              0.006579
Long/Short Equity         0.014051
Merger Arbitrage          0.008875
Relative Value            0.012244
Short Selling             0.027283
Funds Of Funds            0.012122
dtype: float64
```

In [66]:
```
erk.semideviation(hfi)
```

Out[66]:
```
Convertible Arbitrage     0.019540
CTA Global                0.012443
Distressed Securities     0.015185
Emerging Markets          0.028039
Equity Market Neutral     0.009566
Event Driven              0.015429
Fixed Income Arbitrage    0.017763
Global Macro              0.006579
Long/Short Equity         0.014051
Merger Arbitrage          0.008875
Relative Value            0.012244
Short Selling             0.027283
Funds Of Funds            0.012122
dtype: float64
```

VaR and CVaR Value At Risk

In [ ]:
```
- Historic VaR
- Parametric VaR - Gussian
- Modified COrnish-Fisher VaR
```

In [67]:
```
import numpy as np
```

In [69]:
```
np.percentile(hfi,5,axis = 0)
```

Out[69]:
```
array([-0.01576, -0.03169, -0.01966, -0.04247, -0.00814, -0.02535,
       -0.00787, -0.01499, -0.02598, -0.01047, -0.01174, -0.06783,
       -0.02047])
```

In [73]:
```
def var_historic(r,level=5):
    '''
    VaR Historic
    '''
    if isinstance(r,pd.DataFrame):
        return r.aggregate(var_historic,level=level)
    elif isinstance(r, pd.Series):
        return -np.percentile(r,level)
    else:
        raise TypeError("Expected r to be Series or DataFrame")
```

In [74]:
```
var_historic(hfi)
```

Out[74]:
```
Convertible Arbitrage     0.01576
CTA Global                0.03169
Distressed Securities     0.01966
Emerging Markets          0.04247
Equity Market Neutral     0.00814
Event Driven              0.02535
Fixed Income Arbitrage    0.00787
Global Macro              0.01499
Long/Short Equity         0.02598
Merger Arbitrage          0.01047
Relative Value            0.01174
Short Selling             0.06783
Funds Of Funds            0.02047
dtype: float64
```

In [109]:
```
type(hfi)
```

Out[109]:
```
pandas.core.frame.DataFrame
```

In [110]:
```
erk.var_historic(hfi)
```

Out[110]:
```
Convertible Arbitrage     0.01576
CTA Global                0.03169
Distressed Securities     0.01966
Emerging Markets          0.04247
Equity Market Neutral     0.00814
Event Driven              0.02535
Fixed Income Arbitrage    0.00787
Global Macro              0.01499
Long/Short Equity         0.02598
Merger Arbitrage          0.01047
Relative Value            0.01174
Short Selling             0.06783
Funds Of Funds            0.02047
dtype: float64
```

In [111]:
```
from scipy.stats import norm
z = norm.ppf(.05) # return Z-score
```

In [112]:
```
z
```

Out[112]:
```
-1.6448536269514729
```

In [113]:
```
-(hfi.mean() + z*hfi.std(ddof=0))
```

Out[113]:
```
Convertible Arbitrage     0.021691
CTA Global                0.034235
Distressed Securities     0.021032
Emerging Markets          0.047164
Equity Market Neutral     0.008850
Event Driven              0.021144
Fixed Income Arbitrage    0.014579
Global Macro              0.018766
Long/Short Equity         0.026397
Merger Arbitrage          0.010435
Relative Value            0.013061
Short Selling             0.080086
Funds Of Funds            0.021292
dtype: float64
```

In [114]:
```
hfi
```

Out[114]:

date	Convertible Arbitrage	CTA Global	Distressed Securities	Emerging Markets	Equity Market Neutral	Event Driven	Fixed Income Arbitrage	Global Macro	Long/Short Equity
1997-01	0.0119	0.0393	0.0178	0.0791	0.0189	0.0213	0.0191	0.0573	0.0281
1997-02	0.0123	0.0298	0.0122	0.0525	0.0101	0.0084	0.0122	0.0175	-0.0006
1997-03	0.0078	-0.0021	-0.0012	-0.0120	0.0016	-0.0023	0.0109	-0.0119	-0.0084
1997-04	0.0086	-0.0170	0.0030	0.0119	0.0119	-0.0005	0.0130	0.0172	0.0084
1997-05	0.0156	-0.0015	0.0233	0.0315	0.0189	0.0346	0.0118	0.0108	0.0394
...	...	...	...	...	...	...	...	...	...
2018-07	0.0021	-0.0058	0.0093	0.0040	-0.0010	0.0055	0.0022	-0.0014	0.0067
2018-08	0.0024	0.0166	0.0002	-0.0277	0.0004	0.0011	0.0017	-0.0007	0.0035
2018-09	0.0034	-0.0054	0.0050	-0.0110	-0.0016	0.0032	0.0036	0.0006	-0.0023
2018-10	-0.0073	-0.0314	-0.0158	-0.0315	-0.0129	-0.0257	-0.0023	-0.0096	-0.0402
2018-11	-0.0068	-0.0053	-0.0193	0.0120	-0.0211	-0.0034	-0.0067	-0.0087	-0.0044

263 rows × 13 columns

In [115]:
```
list = [erk.var_gaussian(hfi), erk.var_gaussian(hfi,modified = True), erk.v
comparison = pd.concat(list,axis = 1)
comparison.columns = ['Gaussion','Cornish-Fisher','Historic']
```

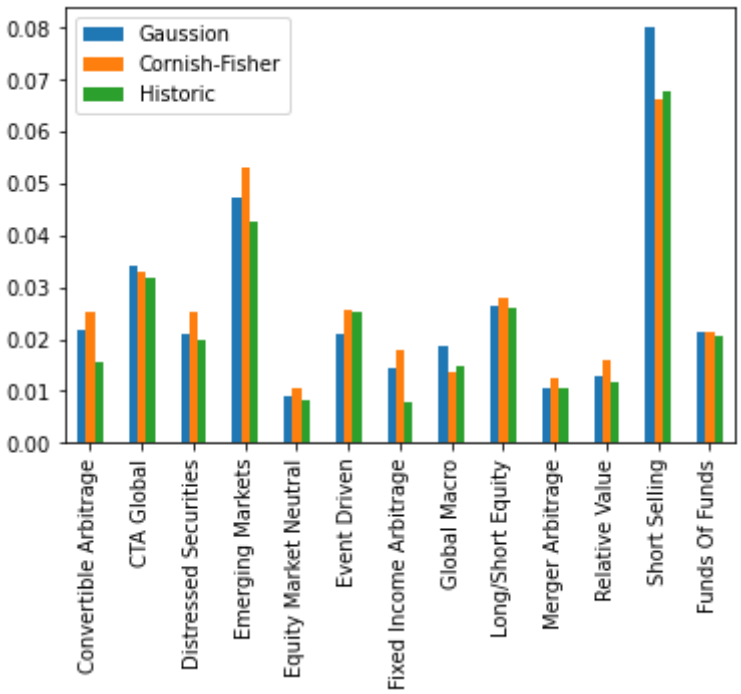
In [116]:
```
comparison
```

Out[116]:

	Gaussion	Cornish-Fisher	Historic
Convertible Arbitrage	0.021691	0.025166	0.01576
CTA Global	0.034235	0.033094	0.03169
Distressed Securities	0.021032	0.025102	0.01966
Emerging Markets	0.047164	0.053011	0.04247
Equity Market Neutral	0.008850	0.010734	0.00814
Event Driven	0.021144	0.025516	0.02535
Fixed Income Arbitrage	0.014579	0.017881	0.00787
Global Macro	0.018766	0.013581	0.01499
Long/Short Equity	0.026397	0.027935	0.02598
Merger Arbitrage	0.010435	0.012612	0.01047
Relative Value	0.013061	0.016157	0.01174
Short Selling	0.080086	0.066157	0.06783
Funds Of Funds	0.021292	0.021576	0.02047

In [117]:
```
comparison.plot.bar()
```

Out[117]: <AxesSubplot:>



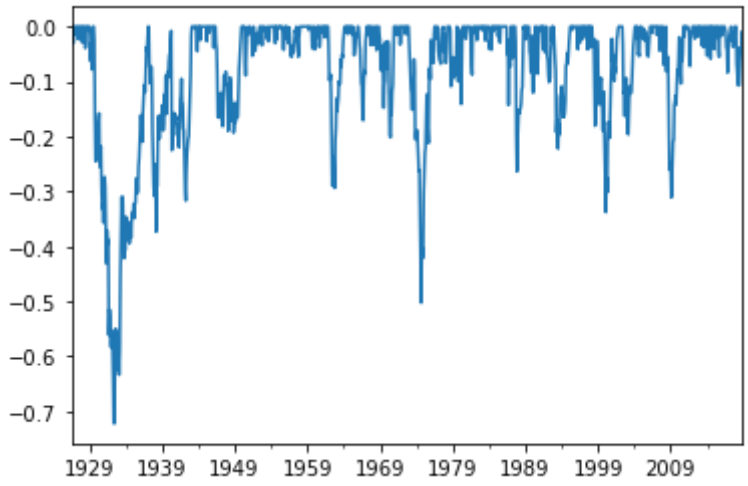
In [2]:
```
import pandas as pd

data = pd.read_csv('ind30_m_vw_rets.csv',header=0,index_col=0,parse_dates=T

data.index = pd.to_datetime(data.index, format = '%Y%m').to_period('M')

data.columns

data.columns = data.columns.str.strip()
# series all has str method

data.columns
```

Out[2]:
```
Index(['Food', 'Beer', 'Smoke', 'Games', 'Books', 'Hshld', 'Clths', 'Hlt
h',
       'Chems', 'Txtls', 'Cnstr', 'Steel', 'FabPr', 'ElcEq', 'Autos', 'Ca
rry',
       'Mines', 'Coal', 'Oil', 'Util', 'Telcm', 'Servs', 'BusEq', 'Pape
r',
       'Trans', 'Whlsl', 'Rtail', 'Meals', 'Fin', 'Other'],
      dtype='object')
```

In [3]:
```
import edhec_risk_kit as erk
```

In [6]:
```
erk.calculate_wealth(data['Food'])['Drawdown'].plot.line()
```

Out[6]: <AxesSubplot:>



In [8]:
```
erk.var_gaussian(data[['Food','Smoke','Coal','Beer','Fin']],modified=True)
```

Out[8]:
```
Food     0.061207
Smoke    0.080292
Coal     0.047359
Beer     0.033881
Fin      0.075199
dtype: float64
```

In [12]:
```
erk.var_gaussian(data,modified=True).sort_values().tail()
```

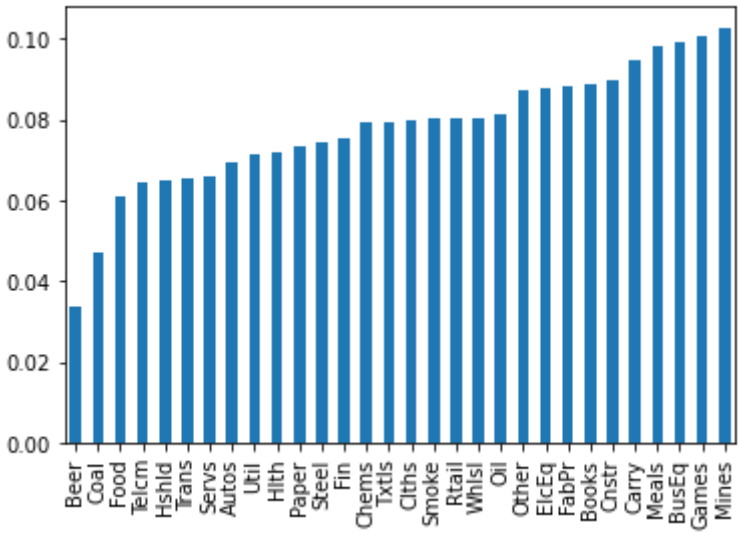
Out[12]:
```
Carry    0.094527
Meals    0.098403
BusEq    0.099377
Games    0.100701
Mines    0.102782
dtype: float64
```

In [13]:
```
erk.var_gaussian(data,modified=True).sort_values().head()
```

Out[13]:
```
Beer     0.033881
Coal     0.047359
Food     0.061207
Telcm    0.064719
Hshld    0.064886
dtype: float64
```

In [15]:
```
erk.var_gaussian(data,modified=True).sort_values().plot.bar()
```

Out[15]: <AxesSubplot:>

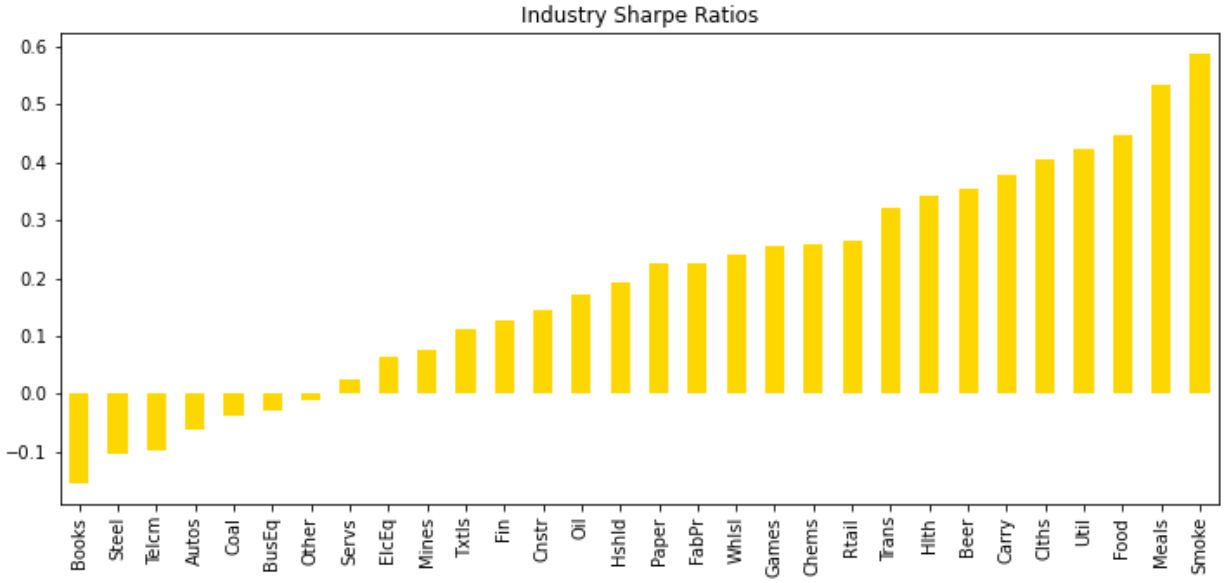


In [19]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [24]:
```
erk.sharpe_ratio(data['2000':],0.03,12).sort_values().plot.bar(title = 'Ind
```

Out[24]: <AxesSubplot:title={'center':'Industry Sharpe Ratios'}>



In [29]:
```
data.loc['2000']
```

Out[29]:

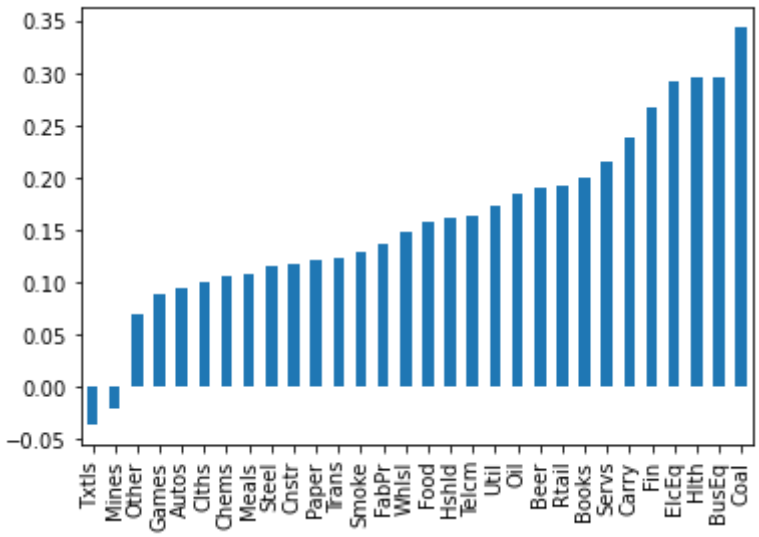
	Food	Beer	Smoke	Games	Books	Hshld	Clths	Hlth	Chems	Txtls	...	
2000-01	-0.0829	-0.0228	-0.0862	0.0229	-0.0092	-0.0651	-0.1138	0.0756	-0.0933	-0.0764	...	-0
2000-02	-0.0689	-0.1164	-0.0401	-0.0177	-0.0071	-0.1157	-0.1278	-0.0288	-0.0729	-0.0584	...	-0
2000-03	0.0969	0.0013	0.0511	0.1052	0.1293	-0.1426	0.2506	0.0028	0.1217	0.0535	...	0
2000-04	-0.0390	0.0368	0.0379	0.0220	-0.0734	0.0429	0.0430	0.0525	-0.0373	0.0648	...	-0
2000-05	0.1565	0.1187	0.1951	0.0119	-0.0593	0.0279	-0.0504	0.0394	-0.0010	-0.0328	...	-0
2000-06	0.0234	0.0605	0.0296	0.0018	-0.0010	-0.0253	-0.0677	0.1152	-0.0648	-0.0988	...	0
2000-07	-0.0088	0.0657	-0.0439	-0.0114	-0.0119	-0.0264	0.0598	-0.0616	-0.0004	0.0485	...	-0
2000-08	-0.0266	-0.1083	0.1903	0.0571	0.0282	0.0350	0.0253	0.0358	0.0187	-0.0365	...	-0
2000-09	0.0490	0.0546	0.0087	-0.0827	-0.0067	-0.0173	-0.0123	0.0372	-0.0477	0.0555	...	-0
2000-10	0.0607	0.0899	0.2280	-0.0498	-0.0178	0.1000	0.0427	0.0260	0.0489	-0.0303	...	-0
2000-11	0.0444	0.0420	0.0407	-0.1427	-0.0527	0.0017	0.0407	0.0079	-0.0084	-0.0047	...	-0
2000-12	0.0551	-0.0257	0.1696	0.0447	0.1041	0.0315	0.1585	0.0353	0.1220	0.0655	...	-0

12 rows × 30 columns

In [32]:
```
er = erk.annualize_rets(data.loc['1995':'2000'],12).sort_values()
```

In [33]:
```
er.plot.bar()
```

Out[33]: <AxesSubplot:>



In [34]:
```
cov = data.loc['1995':'2000'].cov()
```

In [36]:
```
cov.shape
```

Out[36]: (30, 30)

In [ ]:
```
# 协方差矩阵
```