```
%load_ext autoreload
%autoreload 2
%matplotlib inline

from google.colab import drive
drive.mount('/content/drive')
```

```
import os, sys

DATAPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio/data'
print(f"DATAPATH:{DATAPATH} contents:{os.listdir(DATAPATH)}")

MODULEPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio/nb'
print(f"MODULEPATH:{MODULEPATH} contents:{os.listdir(MODULEPATH)}")

sys.path.append(MODULEPATH)
print(f"sys.path:{sys.path}")
```

```
import numpy as np
import pandas as pd

import edhec_risk_kit_106_BBI as erk
```

Q1-4

```
ffme = erk.get_ffme_returns(DATAPATH)
#ffme.describe()
ffme.head()
```

```
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
```

```python
def returns_annualized(returns_by_month):
  """
  Computes annualized returns from month by month returns (assumes pd.DataFrame
  """
  returns_overall = (1 + returns_by_month).prod() - 1.0
  print(f"returns_overall:"); print(returns_overall); print(f"\n")
  returns_monthly = (1.0 + returns_overall) ** (1.0 / ffme.shape[0]) - 1.0
  print(f"returns_monthly:"); print(returns_monthly); print(f"\n")
  returns_annually = (1.0 + returns_monthly) ** 12 - 1.0
  print(f"returns_annually:"); print(returns_annually); print(f"\n")
  return returns_annually

returns_annually = returns_annualized(ffme)
```

```
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
```

```python
# #Annualized Return of the portfolios over the entire period
# returns_overall = (1 + ffme).prod() - 1.0
# print(f"returns_overall:"); print(returns_overall); print(f"\n")
```

```
# returns_monthly = (1.0 + returns_overall) ** (1.0 / ffme.shape[0]) - 1.0
# print(f"returns_monthly:"); print(returns_monthly); print(f"\n")
# returns_annually = (1.0 + returns_monthly) ** 12 - 1.0
# print(f"returns_annually:"); print(returns_annually); print(f"\n")
```

```
SmallCap    0.051772
LargeCap    0.040245
dtype: float64
```

```
def volatility_annualized(returns_by_month):
  """
  Computes annualized volatility from month by month returns (assumes pd.DataFra
```

```
    """

    volatility_monthly = returns_by_month.std()
    print(f"volatility_monthly:"); print(volatility_monthly); print(f"\n")
    volatility_annually = volatility_monthly * np.sqrt(12)
    print(f"volatility_annually:"); print(volatility_annually); print(f"\n")

    return volatility_annually

volatility_annually = volatility_annualized(ffme)
```

```
Convertible Arbitrage     0.031776
CTA Global                0.049542
Distressed Securities     0.046654
Emerging Markets          0.088466
Equity Market Neutral     0.018000
Event Driven              0.048612
Fixed Income Arbitrage    0.041672
Global Macro              0.024316
Long/Short Equity         0.049558
Merger Arbitrage          0.025336
Relative Value            0.026660
Short Selling             0.113576
Funds Of Funds            0.039664
```

```
# volatility_monthly = ffme.std()
# print(f"volatility_monthly:"); print(volatility_monthly); print(f"\n")
# volatility_annually = volatility_monthly * np.sqrt(12)
```

```
4 # print(f"volatility_annually:"); print(volatility_annually); print(f"\n")
```

```
SmallCap    0.162609
LargeCap    0.121277
dtype: float64
```

Q5-8

```
1 ffme_1995_2015 = ffme['1995':'2015']
2 print(ffme_1995_2015.head())
3 print(ffme_1995_2015.tail())
```

```
-0.994457883209753
```

```
1 returns_annually_1995_2015 = returns_annualized(ffme_1995_2015)
2 volatility_annually_1995_2015 = volatility_annualized(ffme_1995_2015)
```

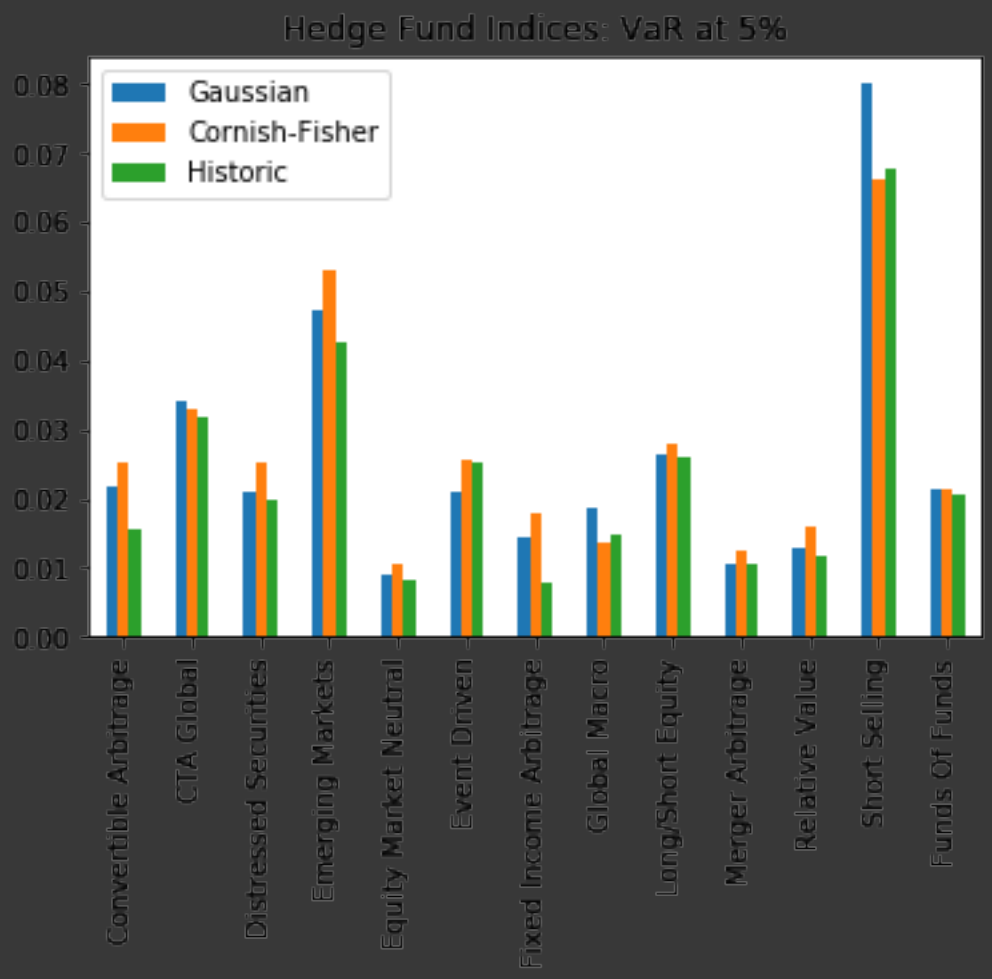
```
Convertible Arbitrage     0.01576
CTA Global                0.03169
Distressed Securities     0.01966
Emerging Markets          0.04247
Equity Market Neutral     0.00814
Event Driven              0.02535
Fixed Income Arbitrage    0.00787
Global Macro              0.01499
Long/Short Equity       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

Q9-12

```
for clm in ffme_1995_2015.columns:
  print(f"\nportfolio:{clm}")
  drawdown_clm = erk.drawdown(ffme_1995_2015[clm])
```

```
print(drawdown_clm)
print(f"Drawdown.max: value:{-drawdown_clm['Drawdown'].min()}; period:{drawdow
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10db59ac8>
```




```
# return_series = ffme_1995_2015['SmallCap']
# print(return_series)

# wealth_index = 1000*(1+return_series).cumprod()
# previous_peaks = wealth_index.cummax()
# drawdowns = (wealth_index - previous_peaks)/previous_peaks
# print(drawdowns)
```

```
# pd.DataFrame({"Wealth": wealth_index,
#                           "Previous Peak": previous_peaks,
#                           "Drawdown": drawdowns})
#erk.drawdown(ffme_1995_2015.iloc[:,:1])
```

```
Global Macro                0.982922
Short Selling               0.767975
CTA Global                  0.173699
```

```
Funds Of Funds             -0.361783
Long/Short Equity          -0.390227
Emerging Markets           -1.167067
Distressed Securities      -1.300842
Merger Arbitrage           -1.320083
Event Driven               -1.409154
Relative Value             -1.815470
Equity Market Neutral      -2.124435
Convertible Arbitrage      -2.639592
Fixed Income Arbitrage     -3.940320
dtype: float64
```

Old Q1-2

```
#Annualized Return of the Lo 20 portfolio over the entire period
returns_overall = (1 + ffme['LargeCap']).prod() - 1.0
returns_monthly = (1.0 + returns_overall) ** (1.0 / ffme.shape[0]) - 1.0
```

```
returns_annually = (1.0 + returns_monthly) ** 12 - 1.0
print(f"returns_LargeCap: overall:{returns_overall:0.4f}; monthly:{returns_month
print(f"  annually:{returns_annually}")
print(f"\nAnswer:{returns_annually * 100.0:0.2f}")
```

```
print(f"Answer: annualized volatility: {ffme['LargeCap'].std() * np.sqrt(12):0.2
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client
```

Q13-16

```
hfi = erk.get_hfi_returns(DATAPATH)
hfi
```

```
1 hfi_2009_end = hfi['2009':]
2 hfi_2009_end
```

```
1 hfi_2009_end_semi_deviation = erk.semideviation(hfi_2009_end)
2 hfi_2009_end_semi_deviation.sort_values()
```

```
hfi_2009_end_skewness = erk.skewness(hfi_2009_end)
hfi_2009_end_skewness.sort_values()
```

```
hfi_2009_end_kurtosis = erk.kurtosis(hfi_2009_end)
hfi_2009_end_kurtosis.sort_values()
```

Downside Measures: SemiDeviation, VaR and CVaR

We're going to look at a few measures of downside risk. We've already seen how to compute drawd
popular measures, and we are going to develop code to compute these and add them to our toolbo

The first measure is the simplest, which is the semideviation, which is nothing more than the volati
negative.

The code is very simple:

```
def semideviation(r):
    """
    Returns the semideviation aka negative semideviation of r
    r must be a Series or a DataFrame, else raises a TypeError
    """
    is_negative = r < 0
    return r[is_negative].std(ddof=0)
```

```
1 hfi = erk.get_hfi_returns(DATAPATH)
```

```
1 hfi.describe()
```

volatility_monthly:

```
SmallCap    0.106288
LargeCap    0.053900
dtype: float64


volatility_annually:
SmallCap    0.368193
LargeCap    0.186716
dtype: float64
```

```
def semideviation(r):
    """
    Returns the semideviation aka negative semideviation of r
    r must be a Series or a DataFrame, else raises a TypeError
    """
    is_negative = r < 0
    return r[is_negative].std(ddof=0)

```

```
erk.semideviation(hfi)
```

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

```
1 erk.semideviation(hfi).sort_values()
```

```
            SmallCap  LargeCap
1995-01       0.0543    0.0312
1995-02       0.0200    0.0355
1995-03       0.0155    0.0221
1995-04       0.0260    0.0228
1995-05       0.0172    0.0455
            SmallCap  LargeCap
2015-08      -0.0299   -0.0620
2015-09      -0.0558   -0.0338
2015-10       0.0271    0.0780
2015-11       0.0109    0.0044
2015-12      -0.0475   -0.0171
```

```
1 ffme = erk.get_ffme_returns(DATAPATH)
2 erk.semideviation(ffme)
```

```
returns_overall:
SmallCap    11.038982
LargeCap     5.478394
dtype: float64
```

```
1 # This will not work: erk.semideviation([1,2,3,4])
```

VaR and CVaR

We'll look at three different ways to compute Value At Risk

1. Historic VaR
2. Parametric Gaussian VaR
3. Modified (Cornish-Fisher) VaR

To compute the historic VaR at a certain level, say 5%, all we have to do is to find the number such
number and 95% of the returns fall above that number. In other words, we want the 5 percentile ret

Fortunately, numpy has a `np.percentile` function that computes exactly that.

Add the following code to the `edhec_risk_kit.py` file:

```
def var_historic(r, level=5):
    """
    Returns the historic Value at Risk at a specified level
    i.e. returns the number such that "level" percent of the returns
    fall below that number, and the (100-level) percent are above
    """
    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 a Series or DataFrame")
```

```
1 import numpy as np
2 np.percentile(hfi, 5, axis=0)
```

```
1 erk.var_historic(hfi, level=1)
```

Note that for reporting purposes, it is common to invert the sign so we report a positive number to that is at risk.

Conditional VaR aka Beyond VaR

Now that we have the VaR, the CVaR is very easy. All we need is to find the mean of the numbers th

```
1 erk.cvar_historic(hfi, level=1).sort_values()
```

```
1 erk.cvar_historic(ffme)
```

Equity Fixed

Convertible Arbitrage	CTA Global	Distressed Securities	Emerging Markets	Equity Market	Event Driven	Fixed Income	Glo Ma

Parametric Gaussian VaR

The idea behind this is very simple. If a set of returns is normally distributed, we know, for instance the mean and 50% are above.

We also know that approx two thirds of the returns lie within 1 standard deviation. That means one deviation from the mean. Since the normal distribution is symmetric, approximately one sixth (app deviation away from the mean. Therefore, if we know the mean and standard deviation and if we as distributed, the 16% VaR would be the mean minus one standard deviation.

In general we can always convert a percentile point to a z-score (which is the number of standard d a number is). Therefore, if we can convert the VaR level (such as 1% or 5%) to a z-score, we can cal percent of returns lie below it.

`scipy.stat.norm` contains a function `ppf()` which does exactly that. It takes a percentile such as score corresponding to that in the normal distribution.

```
1 from scipy.stats import norm
2 norm.ppf(.5)
```

```
1 norm.ppf(.16)
```

Therefore, all we need to do to estimate the VaR using this method is to find the z-score correspon add that many standard deviations to the mean, to obtain the VaR.

```python
from scipy.stats import norm
def var_gaussian(r, level=5):
    """
    Returns the Parametric Gauusian VaR of a Series or DataFrame
    """
    # compute the Z score assuming it was Gaussian
    z = norm.ppf(level/100)
    return -(r.mean() + z*r.std(ddof=0))
```

```
1 erk.var_gaussian(hfi)
```

```
1 erk.var_historic(hfi)
```

```
Equity Market Neutral    -0.896327
```

```
Funds Of Funds            -0.646908
Merger Arbitrage          -0.551065
Event Driven              -0.488821
Long/Short Equity         -0.463703
Distressed Securities     -0.254944
Emerging Markets           0.033123
CTA Global                 0.052062
Relative Value             0.159953
Global Macro               0.348184
Short Selling              0.456518
Fixed Income Arbitrage     1.121453
Convertible Arbitrage      1.305911
dtype: float64
```

Cornish-Fisher Modification

The Cornish-Fisher modification is an elegant and simple adjustment.

The z-score tells us how many standard deviations away from the mean we need to go to find the V
know that z-score will give us an inaccurate number. The basic idea is that since we can observe th
data, we can adjust the z-score up or down to come up with a modifed z-score. e.g. intuitively, all o
skewness is negative, we'll decrease the z-score further down, and if the skewness is positive, we'l

The adjusted z-score which we'll call $z_{cornishfisher}$ given by:

$$z_{cornishfisher} = z + \frac{1}{6}(z^2 - 1)S + \frac{1}{24}(z^3 - 3z)(K - 3) - \frac{1}{36}(2z^3 - 5$$

We can modify the previous function by adding a "modified" parameter with a default value of `True` following piece of code is executed, which modifes `z`:

```
if modified:
    # modify the Z score based on observed skewness and kurtosis
    s = skewness(r)
    k = kurtosis(r)
    z = (z +
            (z**2 - 1)*s/6 +
            (z**3 -3*z)*(k-3)/24 -
            (2*z**3 - 5*z)*(s**2)/36
        )
```

The rewritten function is:

```
from scipy.stats import norm
def var_gaussian(r, level=5, modified=False):
```

```
    """
    Returns the Parametric Gauusian VaR of a Series or DataFrame
    If "modified" is True, then the modified VaR is returned,
    using the Cornish-Fisher modification
    """
    # compute the Z score assuming it was Gaussian
    z = norm.ppf(level/100)
    if modified:
        # modify the Z score based on observed skewness and kurtosis
        s = skewness(r)
        k = kurtosis(r)
        z = (z +
                (z**2 - 1)*s/6 +
                (z**3 -3*z)*(k-3)/24 -
                (2*z**3 - 5*z)*(s**2)/36
            )

    return -(r.mean() + z*r.std(ddof=0))
```

We can now compare the different methods:

```
1 var_table = [erk.var_gaussian(hfi),
2               erk.var_gaussian(hfi, modified=True),
3               erk.var_historic(hfi)]
```

```
comparison = pd.concat(var_table, axis=1)
comparison.columns=['Gaussian', 'Cornish-Fisher', 'Historic']
comparison.plot.bar(title="Hedge Fund Indices: VaR at 5%")
```

Note that in some cases, the cornish-fisher VaR is lower i.e. estimates a smaller loss than you wou
assumption. That can happen if the observed skewness is positive, as is the case for "Short Selling

```
erk.skewness(hfi).sort_values(ascending=False)
```

```
CTA Global                 2.591087
```

```
Merger Arbitrage          2.715238
Global Macro              3.164362
Distressed Securities     3.319725
Event Driven              3.620617
Funds Of Funds            3.816132
Long/Short Equity         4.115713
Short Selling             4.175832
Emerging Markets          4.401636
Relative Value            4.512482
Equity Market Neutral     5.071677
Fixed Income Arbitrage    6.406941
Convertible Arbitrage     6.775731
dtype: float64
```

```
%load_ext autoreload
%autoreload 2
%matplotlib inline

from google.colab import drive
drive.mount('/content/drive')
```

```
import os, sys

DATAPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio
print(f"DATAPATH:{DATAPATH} contents:{os.listdir(DATAPATH)}")

MODULEPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfol
print(f"MODULEPATH:{MODULEPATH} contents:{os.listdir(MODULEPATH)}")

sys.path.append(MODULEPATH)
print(f"sys.path:{sys.path}")
```

```
import numpy as np
import pandas as pd

import edhec_risk_kit_106_BBI as erk
```

Q1-4

```
ffme = erk.get_ffme_returns(DATAPATH)
#ffme.describe()
ffme.head()
```

```
def returns_annualized(returns_by_month):
  """
  Computes annualized returns from month by month returns (assumes pd.D
```

```
    """
    returns_overall = (1 + returns_by_month).prod() - 1.0
    print(f"returns_overall:"); print(returns_overall); print(f"\n")
    returns_monthly = (1.0 + returns_overall) ** (1.0 / ffme.shape[0]) -
    print(f"returns_monthly:"); print(returns_monthly); print(f"\n")
    returns_annually = (1.0 + returns_monthly) ** 12 - 1.0
    print(f"returns_annually:"); print(returns_annually); print(f"\n")
    return returns_annually

returns_annually = returns_annualized(ffme)
```

```
# #Annualized Return of the portfolios over the entire period
# returns_overall = (1 + ffme).prod() - 1.0
# print(f"returns_overall:"); print(returns_overall); print(f"\n")
```

```
# returns_monthly = (1.0 + returns_overall) ** (1.0 / ffme.shape[0]) -
# print(f"returns_monthly:"); print(returns_monthly); print(f"\n")
# returns_annually = (1.0 + returns_monthly) ** 12 - 1.0
# print(f"returns_annually:"); print(returns_annually); print(f"\n")
```

```
def volatility_annualized(returns_by_month):
  """
  Computes annualized volatility from month by month returns (assumes p
```

```
    """

    volatility_monthly = returns_by_month.std()
    print(f"volatility_monthly:"); print(volatility_monthly); print(f"\n"
    volatility_annually = volatility_monthly * np.sqrt(12)
    print(f"volatility_annually:"); print(volatility_annually); print(f"\

    return volatility_annually

volatility_annually = volatility_annualized(ffme)
```

```
# volatility_monthly = ffme.std()
# print(f"volatility_monthly:"); print(volatility_monthly); print(f"\n"
# volatility_annually = volatility_monthly * np.sqrt(12)
```

```
# print(f"volatility_annually:"); print(volatility_annually); print(f"\
```

Q5-8

```
ffme_1995_2015 = ffme['1995':'2015']
print(ffme_1995_2015.head())
print(ffme_1995_2015.tail())
```

```
returns_annually_1995_2015 = returns_annualized(ffme_1995_2015)
volatility_annually_1995_2015 = volatility_annualized(ffme_1995_2015)
```

Q9-12

```
for clm in ffme_1995_2015.columns:
  print(f"\nportfolio:{clm}")
  drawdown_clm = erk.drawdown(ffme_1995_2015[clm])
```

```
print(drawdown_clm)
print(f"Drawdown.max: value:{-drawdown_clm['Drawdown'].min()}; period
```

```
# return_series = ffme_1995_2015['SmallCap']
# print(return_series)

# wealth_index = 1000*(1+return_series).cumprod()
# previous_peaks = wealth_index.cummax()
# drawdowns = (wealth_index - previous_peaks)/previous_peaks
# print(drawdowns)
```

```
# pd.DataFrame({"Wealth": wealth_index,
#                           "Previous Peak": previous_peaks,
#                           "Drawdown": drawdowns})
#erk.drawdown(ffme_1995_2015.iloc[:,:1])
```

Old Q1-2

```
#Annualized Return of the Lo 20 portfolio over the entire period
returns_overall = (1 + ffme['LargeCap']).prod() - 1.0
returns_monthly = (1.0 + returns_overall) ** (1.0 / ffme.shape[0]) - 1.
```

```
returns_annually = (1.0 + returns_monthly) ** 12 - 1.0
print(f"returns_LargeCap: overall:{returns_overall:0.4f}; monthly:{retu
print(f"  annually:{returns_annually}")
print(f"\nAnswer:{returns_annually * 100.0:0.2f}")
```

```
print(f"Answer: annualized volatility: {ffme['LargeCap'].std() * np.sqr
```

Q13-16

```
hfi = erk.get_hfi_returns(DATAPATH)
hfi
```

```
hfi_2009_end = hfi['2009':]
hfi_2009_end
```

```
hfi_2009_end_semi_deviation = erk.semideviation(hfi_2009_end)
hfi_2009_end_semi_deviation.sort_values()
```

```
hfi_2009_end_skewness = erk.skewness(hfi_2009_end)
hfi_2009_end_skewness.sort_values()
```

```
hfi_2009_end_kurtosis = erk.kurtosis(hfi_2009_end)
hfi_2009_end_kurtosis.sort_values()
```

```
1 hfi = erk.get_hfi_returns(DATAPATH)
```

```
1 hfi.describe()
```

```
def semideviation(r):
    """
    Returns the semideviation aka negative semideviation of r
    r must be a Series or a DataFrame, else raises a TypeError
    """
    is_negative = r < 0
    return r[is_negative].std(ddof=0)

```

```
erk.semideviation(hfi)
```

```
1 hfi[hfi<0].std(ddof=0)
```

```
1 erk.semideviation(hfi).sort_values()
```

```
ffme = erk.get_ffme_returns(DATAPATH)
erk.semideviation(ffme)
```

```
# This will not work: erk.semideviation([1,2,3,4])
```

```
import numpy as np
np.percentile(hfi, 5, axis=0)
```

```
erk.var_historic(hfi, level=1)
```

Note that for reporting purposes, it is common to invert the sign so we report a positive number to represent the *loss* i.e. the amount that is at risk.

Conditional VaR aka Beyond VaR

Now that we have the VaR, the CVaR is very easy. All we need is to find the mean of the numbers that fell below the VaR!

```
1 erk.cvar_historic(hfi, level=1).sort_values()
```

```
1 erk.cvar_historic(ffme)
```

Parametric Gaussian VaR

The idea behind this is very simple. If a set of returns is normally distributed, we know, for instance, that 50% of the returns are below the mean and 50% are above.

We also know that approx two thirds of the returns lie within 1 standard deviation. That means one third lie beyond one standard deviation from the mean. Since the normal distribution is symmetric, approximately one sixth (approx 16%) lie below one standard deviation away from the mean. Therefore, if we know the mean and standard deviation and if we assume that the returns are normally distributed, the 16% VaR would be the mean minus one standard deviation.

In general we can always convert a percentile point to a z-score (which is the number of standard deviations away from the mean that a number is). Therefore, if we can convert the VaR level (such as 1% or 5%) to a z-score, we can calculate the return level where that percent of returns lie below it.

`scipy.stat.norm` contains a function `ppf()` which does exactly that. It takes a percentile such as 0.05 or 0.01 and gives you the z-score corresponding to that in the normal distribution.

```
1 from scipy.stats import norm
2 norm.ppf(.5)
```

```
1 norm.ppf(.16)
```

```
1 erk.var_gaussian(hfi)
```

```
erk.var_historic(hfi)
```

```
var_table = [erk.var_gaussian(hfi),
             erk.var_gaussian(hfi, modified=True),
             erk.var_historic(hfi)]
comparison = pd.concat(var_table, axis=1)
comparison.columns=['Gaussian', 'Cornish-Fisher', 'Historic']
comparison.plot.bar(title="Hedge Fund Indices: VaR at 5%")
```

Note that in some cases, the cornish-fisher VaR is lower i.e. estimates a smaller loss than you would get from a pure gaussian assumption. That can happen if the observed skewness is

positive, as is the case for "Short Selling" and "Global Macro"

```
erk.skewness(hfi).sort_values(ascending=False)
```