VAR - GARCH

Kannan Singaravelu, CQF

1 Value-at-Risk

There are several methods to evaluate risk for an individual stock or a portfolio, such as variance, standard deviation of returns, et al. But, those measures do not consider a probability distribution. However, many risk managers prefer a simple measure called Value at Risk (VaR). **VaR** is one of the most important metrics that is used to measure the risk associated with a financial position or a portfolio of financial instruments and can be defined as `the maximum loss with a confidence level over a predetermined period`.

Let's say that the 1-day 95% VaR of our portfolio is $100. This means that 95% of the time, it is expected that - under normal market conditions - we will not lose more than $100 by holding our portfolio over one day.

Some of the approaches that are commonly used in the industry are

- **Parametric**
- **Historical**
- **Monte Carlo**
- **Modified**

Import Libraries

We'll import the required libraries that we'll use in this example.

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

# Import data manipulation libraries
import pandas as pd
import numpy as np
from pprint import pprint
from collections import OrderedDict
from numpy.linalg import multi_dot
from scipy import stats
from tabulate import tabulate
```

```
# Import plotly express
import plotly.express as px
px.defaults.width, px.defaults.height = 1000, 600

# Set precision
pd.set_option('display.precision', 4)
```

Retrieve Data

We will retrieve stock prices to build for VaR calculation.

[ ]:
```
# Import & Initialize database
import sqlalchemy
engine = sqlalchemy.create_engine('sqlite:///India')
```

[ ]:
```
# Specify assets / stocks
assets = sorted(['ICICIBANK', 'ITC', 'RELIANCE', 'TCS', 'ASIANPAINT'])
print(assets)
```

[ ]:
```
# Query close price from database
# Refer lab 3 and 1 for further details
df = pd.DataFrame()
for asset in assets:
    df1 = pd.read_sql_query(f'SELECT Date, Close FROM {asset}', engine,␣
 ↪index_col='Date')
    df1.columns = [asset]
    df = pd.concat([df, df1], axis=1)

# View dataframe
df
```

[ ]:
```
# Calculate daily returns
returns = df.pct_change().dropna()
returns.head()
```

1.1 Parametric VaR

The Variance-covariance is a parametric method which assumes (almost always) that the returns are normally distributed. In this method, we first calculate the mean and standard deviation of the returns to derive the risk metric. Based on the assumption of normality, we can generalise, $ VaR = position * (μ- z * σ) $

Confidence Level	Value At Risk
90%	μ - 1.29 * σ
95%	μ - 1.64 * σ
99%	μ - 2.33 * σ

where, μ is the return, σ is the volatility and z is the number of standard deviation from the mean.

[ ]:
```
# Stock returns
stockreturn = returns['ICICIBANK']

# Calculate mean and standard deviation
mean = np.mean(stockreturn)
stdev = np.std(stockreturn)

# Calculate VaR at difference confidence level
VaR_90 = stats.norm.ppf(1-0.90,mean,stdev)
VaR_95 = stats.norm.ppf(1-0.95,mean,stdev)
VaR_99 = stats.norm.ppf(1-0.99,mean,stdev)
```

[ ]:
```
# number of stdev from the mean
stats.norm.ppf(0.01)
```

[ ]:
```
# Ouput results in tabular format
table = [['90%', VaR_90],['95%', VaR_95],['99%', VaR_99] ]
header = ['Confidence Level', 'Value At Risk']
print(tabulate(table,headers=header))
```

1.2 Historical VaR

Asset returns do not necessarily follow a normal distribution. An alternative is to use sorted returns to evaluate a VaR. This method uses historical data where returns are sorted in ascending order to calculate maximum possible loss for a given confidence level.

[ ]:
```
# Use quantile function for Historical VaR
hVaR_90 = returns['ICICIBANK'].quantile(0.10)
hVaR_95 = returns['ICICIBANK'].quantile(0.05)
hVaR_99 = returns['ICICIBANK'].quantile(0.01)
```

[ ]:
```
# Ouput results in tabular format
htable = [['90%', hVaR_90],['95%', hVaR_95],['99%', hVaR_99]]
print(tabulate(htable,headers=header))
```

1.3 MonteCarlo VaR

The Monte Carlo simulation approach has a number of similarities to historical simulation. It allows us to use actual historical distributions rather than having to assume normal returns. As returns are assumed to follow a normal distribution, we could generate n simulated returns with the same mean and standard deviation (derived from the daily returns) and then sorted in ascending order to calculate maximum possible loss for a given confidence level.

[ ]:
```
# Set seed for reproducibility
np.random.seed(42)

# Number of simulations
n_sims = 5000
```

```
# Simulate returns and sort
sim_returns = np.random.normal(mean, stdev, n_sims)

# Use percentile function for MCVaR
MCVaR_90 = np.percentile(sim_returns,10)
MCVaR_95 = np.percentile(sim_returns, 5)
MCVaR_99 = np.percentile(sim_returns,1)
```

[ ]:
```
# Ouput results in tabular format
mctable = [['90%', MCVaR_90],['95%', MCVaR_95],['99%', MCVaR_99]]
print(tabulate(mctable,headers=header))
```

1.4 Normality Test

In the Parametric VaR, we assumed that the returns are normally distributed. However, in the real world, we know that stock / portfolio returns do not necessarily follow a normal distribution. Let's perform a quick check to determine the normality of the underlying returns and see whether we need to modify our approach in deriving the VaR numbers.

Shapiro The Shapiro-Wilk test is a test of normality and is used to determine whether or not a sample comes from a normal distribution.

[ ]:
```
# normality test
stats.shapiro(stockreturn)
```

Our null hypothesis is that the stock's daily returns follows a normal distribution. Since the p-value is less than 0.05, we reject the null hypothesis. We have sufficient evidence to say that the sample data does not come from a normal distribution. This result shouldn't be surprising as the data comes from an empirical distribution.

Anderson-Darling Alternatively, we can perform an Anderson-Darling Test. It is a goodness of fit test that measures how well the data fit a specified distribution. This test is most commonly used to determine whether or not the data follow a normal distribution.

[ ]:
```
# normality test
stats.anderson(stockreturn)
```

Based on the above result, the null hypothesis is rejected since the test statistic value is much higher than the critical value of 1.09 even at 1% significance level.

Plot Histogram We will now plot histogram to visualize the returns distribution

[ ]:
```
# Plot histogram
px.histogram(returns,
             histnorm='probability density',
             title='Histogram of Returns',
             barmode='relative')
```

1.5 Modified VaR

Standard normal distribution have a zero mean, unit variance, zero skewness, and its kurtosis of 3. However, we now know the distribution is not normal and in such scenario, the skewness and excess kurtosis of many stock returns are not zero. As a consequence, the modified VaR was developed to utilize those four moments instead of the first two moments.

$$mVaR = position * (\mu - t * \sigma)$$

where,

$$t = z + \frac{1}{6}(z^2 - 1)s + \frac{1}{24}(z^3 - 3z)k - \frac{1}{36}(2z^3 - 5z)s^2$$

μ is the return, σ is the volatility, s is the skewness, k is the kurtosis and z is the absolute number of standard deviation from the mean.

[ ]:
```
# First four moments
dist = OrderedDict({
    'Mean': np.mean(returns['ICICIBANK']),
    'Variance': np.std(returns['ICICIBANK']),
    'Skew': stats.skew(returns['ICICIBANK']),
    'Kurtosis': stats.kurtosis(returns['ICICIBANK'])
    })

pprint(dist)
```

[ ]:
```
# Specify params for modified VaR
z = abs(stats.norm.ppf(0.01))
s = stats.skew(stockreturn)
k = stats.kurtosis(stockreturn)
t = z+1/6*(z**2-1)*s+1/24*(z**3-3*z)*k-1/36*(2*z**3-5*z)*s**2

# Calculate VaR at difference confidence level
mVaR_99 = (mean-t*stdev)
mVaR_99
```

1.6 Scaling VaR

Now, let's calculate VaR over a 5-day period. To scale it, multiply by square root of time.

$$VaR = position * (\mu - z * \sigma) * \sqrt{T}$$

where, T is the horizon or forecast period.

[ ]:
```
# VaR Scaling
forecast_days = 5
f_VaR_90 = VaR_90*np.sqrt(forecast_days)
f_VaR_95 = VaR_95*np.sqrt(forecast_days)
f_VaR_99 = VaR_99*np.sqrt(forecast_days)
```

```
[ ]: # Ouput results in tabular format
ftable = [['90%', f_VaR_90],['95%', f_VaR_95],['99%', f_VaR_99] ]
fheader = ['Confidence Level', '5-Day Forecast Value At Risk']
print(tabulate(ftable,headers=fheader))
```

```
[ ]: # Plot Scaled VaR
sVaR = pd.DataFrame([-100*VaR_99*np.sqrt(x) for x in range(100)],␣
 ↪columns=['ScaledVaR'])
px.scatter(sVaR, sVaR.index,'ScaledVaR',title='Scaled VaR', labels={'index':
 ↪'Horizon'})
```

1.7 Expected Short Fall

VaR is a reasonable measure of risk if assumption of normality holds. Else, we might underestimate the risk if we observe a fat tail or overestimate the risk if tail is thinner. Expected shortfall or Conditional Value at Risk - **CVaR** - is an estimate of expected shortfall sustained in the worst 1 - x% of scenarios. It is defined as the average loss based on the returns that are lower than the VaR threshold. Assume that we have n return observations, then the expected shortfall is

$$CVaR = \frac{1}{n} * \sum_{i=1}^{n} R_i[R \leq hVaR_{cl}]$$

where, R is returns, $hVaR$ is historical VaR and cl is the confidence level.

```
[ ]: # Calculate CVar
CVaR_90 = returns['ICICIBANK'][returns['ICICIBANK']<=hVaR_90].mean()
CVaR_95 = returns['ICICIBANK'][returns['ICICIBANK']<=hVaR_95].mean()
CVaR_99 = returns['ICICIBANK'][returns['ICICIBANK']<=hVaR_99].mean()
```

```
[ ]: # Ouput results in tabular format
ctable = [['90%', CVaR_90],['95%', CVaR_95],['99%', CVaR_99] ]
cheader = ['Confidence Level', 'Conditional Value At Risk']
print(tabulate(ctable,headers=cheader))
```

1.8 Portfolio VaR

If we know the returns and volatilities of all the assets in the portfolio, we can derive portfolio VaR. We will now derive VaR of minimum variance portfolio consisting of Indian stocks.

```
[ ]: # Weights from Minimum Variance Portfolio
wts = np.array([2.553e-01,  4.434e-02,  2.944e-01,  8.664e-02,  3.193e-01])

# Portfolio mean returns and volatility
port_mean = wts.T @ returns.mean()
port_stdev = np.sqrt(multi_dot([wts.T, returns.cov(), wts]))
pVaR = stats.norm.ppf(1-0.99, port_mean, port_stdev)

print(f"Mean: {port_mean}, Stdev: {port_stdev}, pVaR: {pVaR}")
```

2 GARCH

Asset price volatility is central to derivatives pricing. It is defined as measure of price variability over certain period of time. In essence, it describes standard deviation of returns. There are different types of volatility: Historical, Implied, Forward. In most cases, we assume volatility to be constant, which is clearly not true and numerous studies have been dedicated to estimate this variable, both in academia and industry.

Volatility

Volatility estimation by statistical means assume equal weights to all returns measured over the period. We know that over 1-day, the mean return is small as compared to standard deviation. If we consider a simple m-period moving average, where σ_n is the volatility of return on day n, then with $\overline{u} \approx 0$, we have

$$\sigma_n^2 = \frac{1}{m} \sum_{i=1}^{m} u_{n-i}^2$$

where, u is return and σ^2 is the variance.

ARCH

However, any large return within this n period will elevate the volatility until it drops out of the sample. Further, we observe volatility is mean reverting and tends to vary about a long term mean. To address this effect, we adopt to the weighting schemes.

$$\sigma_n^2 = \gamma \overline{\sigma}^2 + \sum_{i=1}^{m} \alpha_i u_{n-i}^2$$

$$\sigma_n^2 = \omega + \sum_{i=1}^{m} \alpha_i u_{n-i}^2$$

where, $\omega = \gamma \overline{\sigma}^2$ and weights must sum to 1.

This is known as Autoregressive Conditional Heteroscedastic model. Autoregressive models are a statistical technique involving a regression of lagged values where the model suggests that past values can help forecast future values of the same variable. Within the model, a time series is the dependent variable and lagged values are the independent variables.

The ARCH model, was originally developed by Robert Engle in 1982 to measure the dynamics of inflation uncertainty. Conditional heteroskedasticity refers to the notion that the next period's volatility is conditional on the volatility in the current period as well as to the time varying nature of volatility. However, given the volatility dynamics, this model fail to fully capture the persistence of volatility.

GARCH

To address the shortcoming, ARCH has been extended to a generalised framework where we add volatility as a forecasting feature by adding previous variance. This method is popularly known as Generalized ARCH or GARCH model.

$$\sigma_n^2 = \omega + \sum_{i=1}^{p} \alpha_i u_{n-i}^2 + \sum_{i=1}^{q} \beta_i \sigma_{n-i}^2$$

where, p and q are lag length.

GARCH(1,1) is then represented as,

$$\sigma_n^2 = \omega + \alpha u_{n-1}^2 + \beta \sigma_{n-1}^2$$

where, $\alpha + \beta < 1$ and $\gamma + \alpha + \beta = 1$ as weight applied to long term variance cannot be negative and $\frac{\omega}{(1-\alpha-\beta)}$ is the long-run variance.

The GARCH model is a way of specifying the dependence of the time varying nature of volatility. The model incorporates changes in the fluctuations in volatility and tracks the persistence of volatility as it fluctuates around its long-term average and are exponentially weighted. To model GARCH or the conditional volatility, we need to derive ω, α, β by maximizing the likelihood function.

2.1 ARCH Toolbox

ARCH is one of the popular tools used for financial econometrics, written in Python - with Cython and/or Numba used to improve performance. We will now use `arch_model` to fit our GARCH model using this package.

[ ]:
```
# Import arch library
from arch import arch_model
```

[ ]:
```
# Mean zero
g1 = arch_model(stockreturn, vol='GARCH', mean='Zero', p=1, q=1, dist='Normal')
model = g1.fit()
```

[ ]:
```
# Model output
print(model)
```

[ ]:
```
model.summary()
```

[ ]:
```
# Model params
model.params
```

[ ]:
```
# Model Confidence Interval
model.conf_int(alpha=.01)
```

[ ]:
```
# Plot annualised vol
fig = model.plot(annualize='D')
```

[ ]:
```
# Forecast for next 5 days
model_forecast = model.forecast(horizon=60)

# Plot forecasted volatility
fdf = pd.DataFrame(np.sqrt(model_forecast.variance.dropna().T *252)*100)
fdf.columns = ['Cond_Vol']
px.scatter(fdf, fdf.index, 'Cond_Vol', labels={'index':'Horizon'}, title='GARCH␣
↪Volatility Forecast')
```

3 References

- Arch
- Scipy
- Tabulate

Python Labs by Kannan Singaravelu.