Parametric Estimation

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Jamsheed Shorish
Computational Economist

A class of distributions

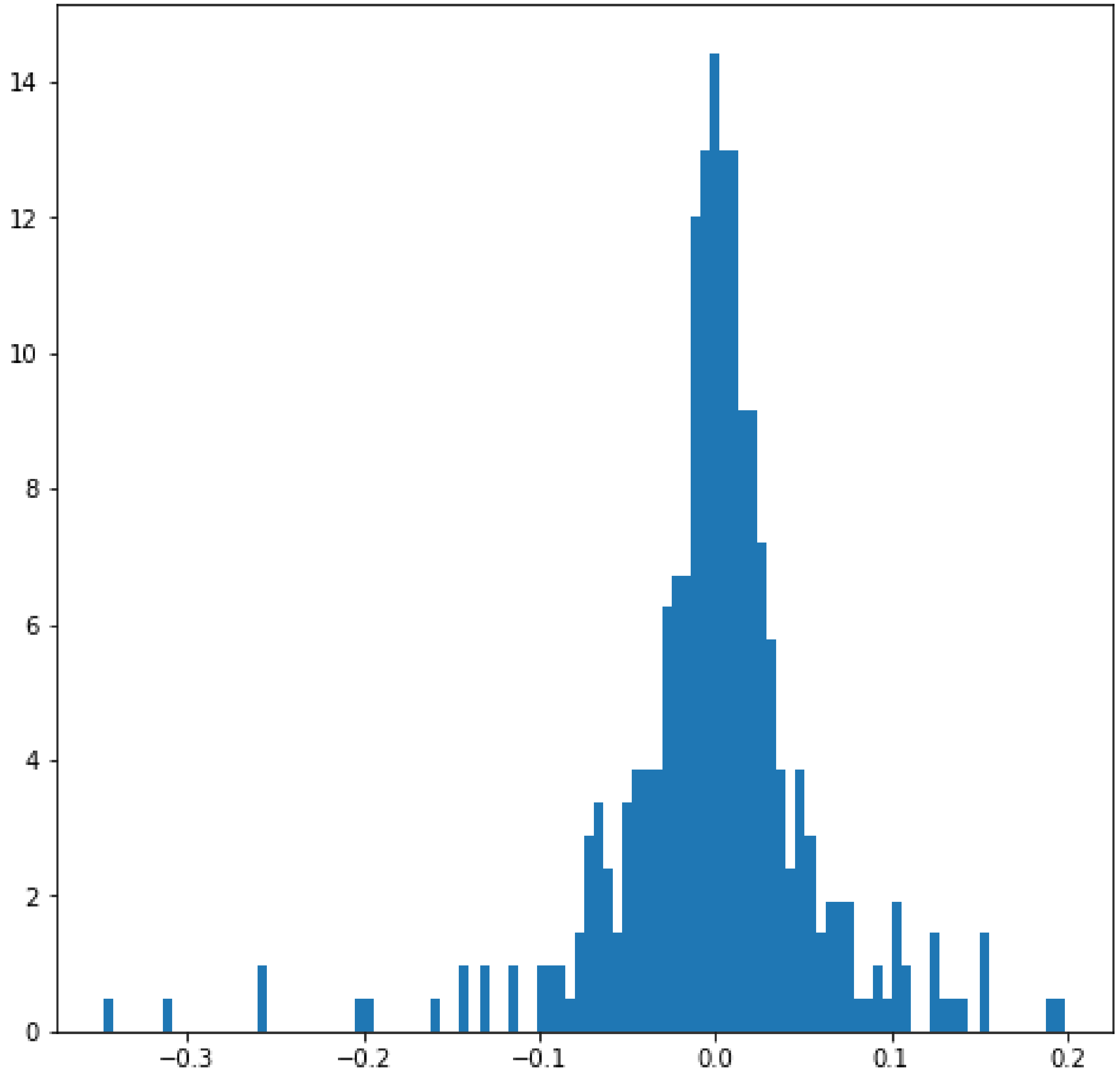
- **Loss distribution:** not known with certainty
- *Class* of possible distributions?
  - Suppose class of distributions $f(x;\theta)$
  - x is loss (random variable)
  - θ is vector of unknown **parameters**
- **Example:** Normal distribution
  - Parameters: $\theta = (\mu, \sigma)$, mean μ and standard deviation σ
- **Parametric estimation:** find 'best' $\theta^\star$ given data
- **Loss distribution:** $f(x, \theta^\star)$

Fitting a distribution

- Fit distribution according to error-minimizing criteria
  - **Example:** `scipy.stats.norm.fit()` , fitting Normal distribution to data
    - **Result:** optimally fitted mean and standard deviation
- **Advantages:**
  - Can *visualize* difference between data and estimate using histogram
  - Can provide *goodness-of-fit* tests

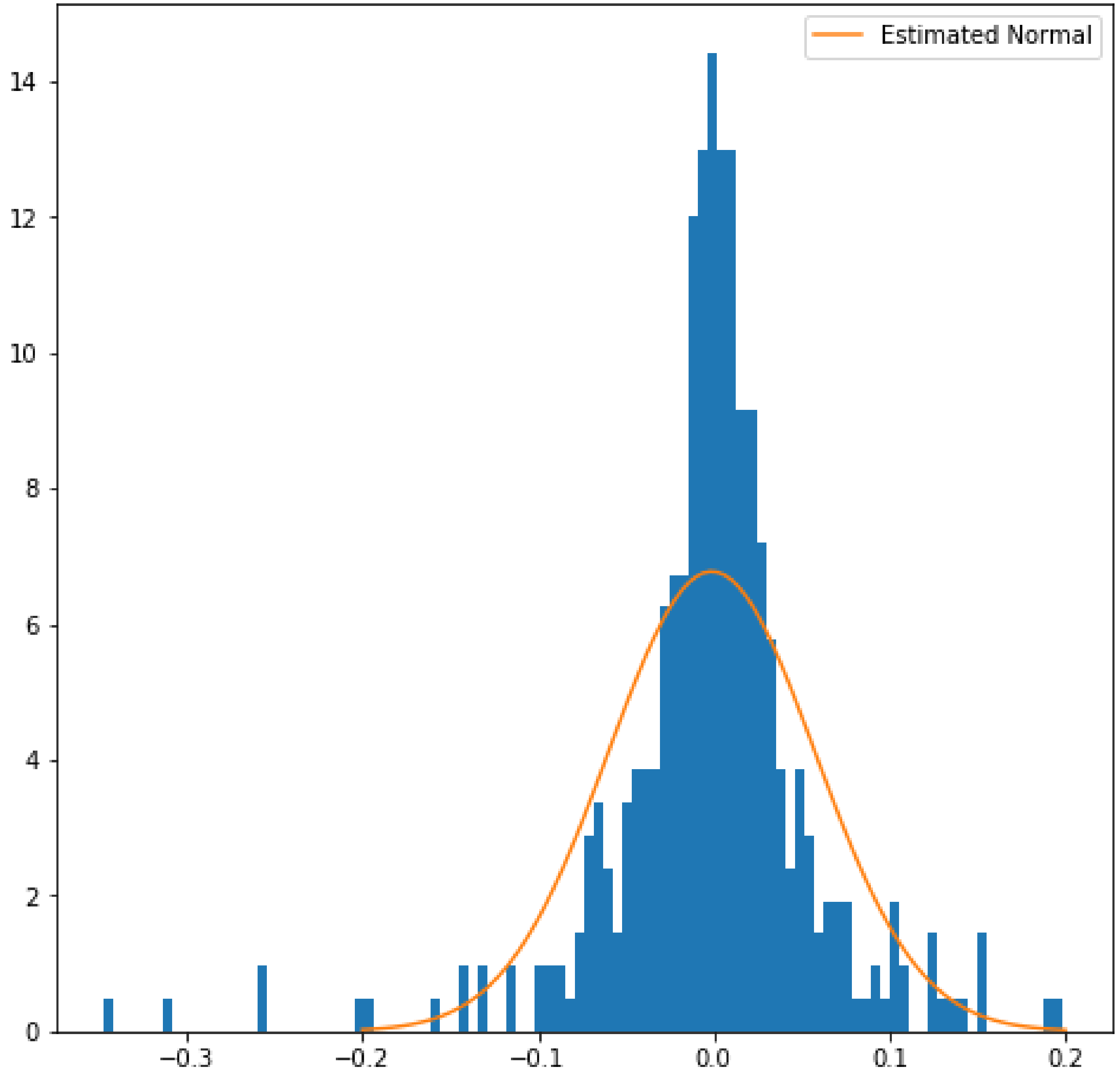
Goodness of fit

- How well does an estimated distribution fit the data?
- **Visualize:** plot histogram of portfolio losses



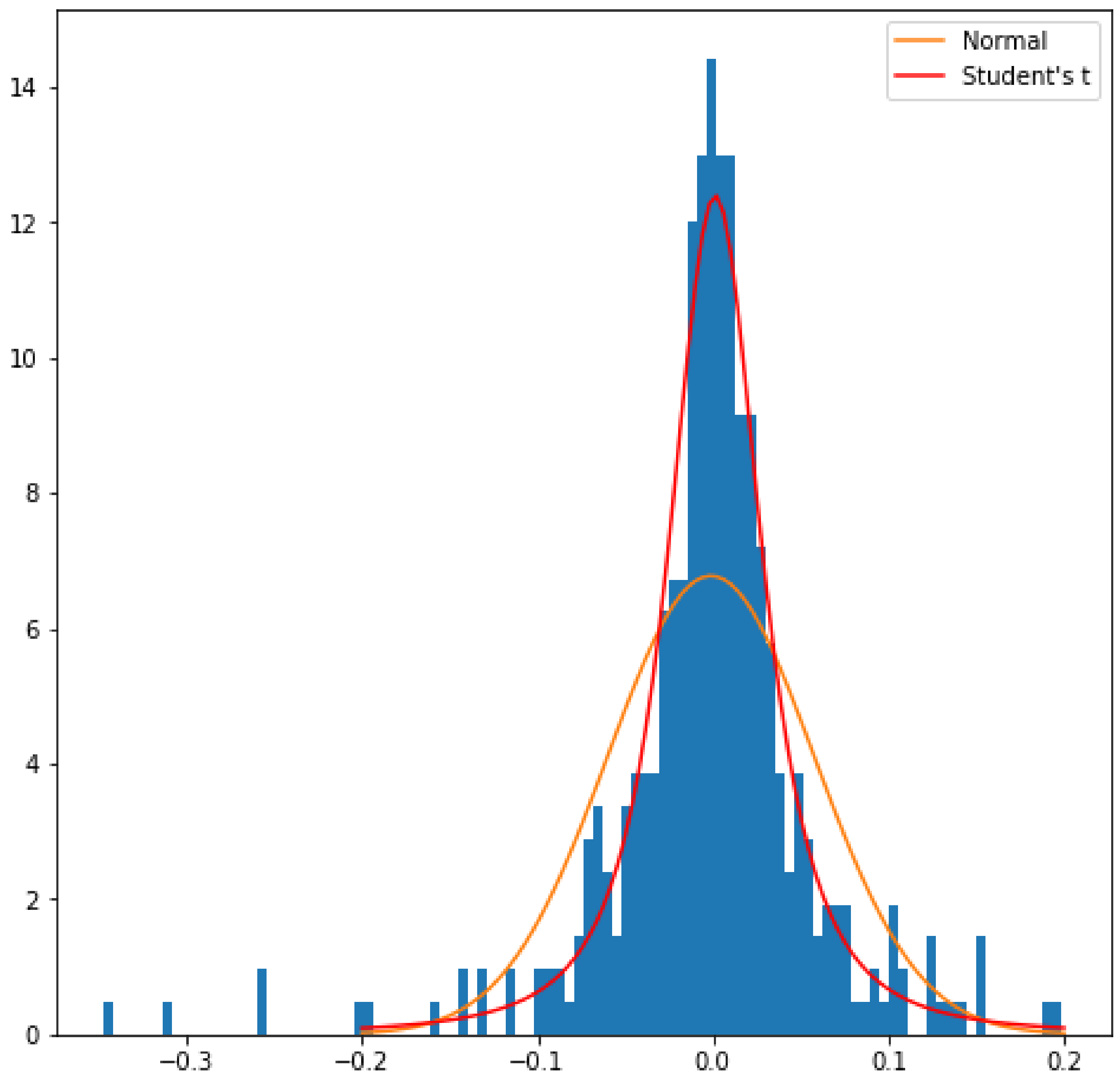
Goodness of fit

- How well does an estimated distribution fit the data?
- **Visualize:** plot histogram of portfolio losses
- Normal distribution with `norm.fit()`



Goodness of fit

- How well does an estimated distribution fit the data?
- **Visualize:** plot histogram of portfolio losses
- **Example:**
  - Normal distribution with `norm.fit()`
  - Student's t-distribution with `t.fit()`
  - Asymmetrical histogram?



Anderson-Darling test

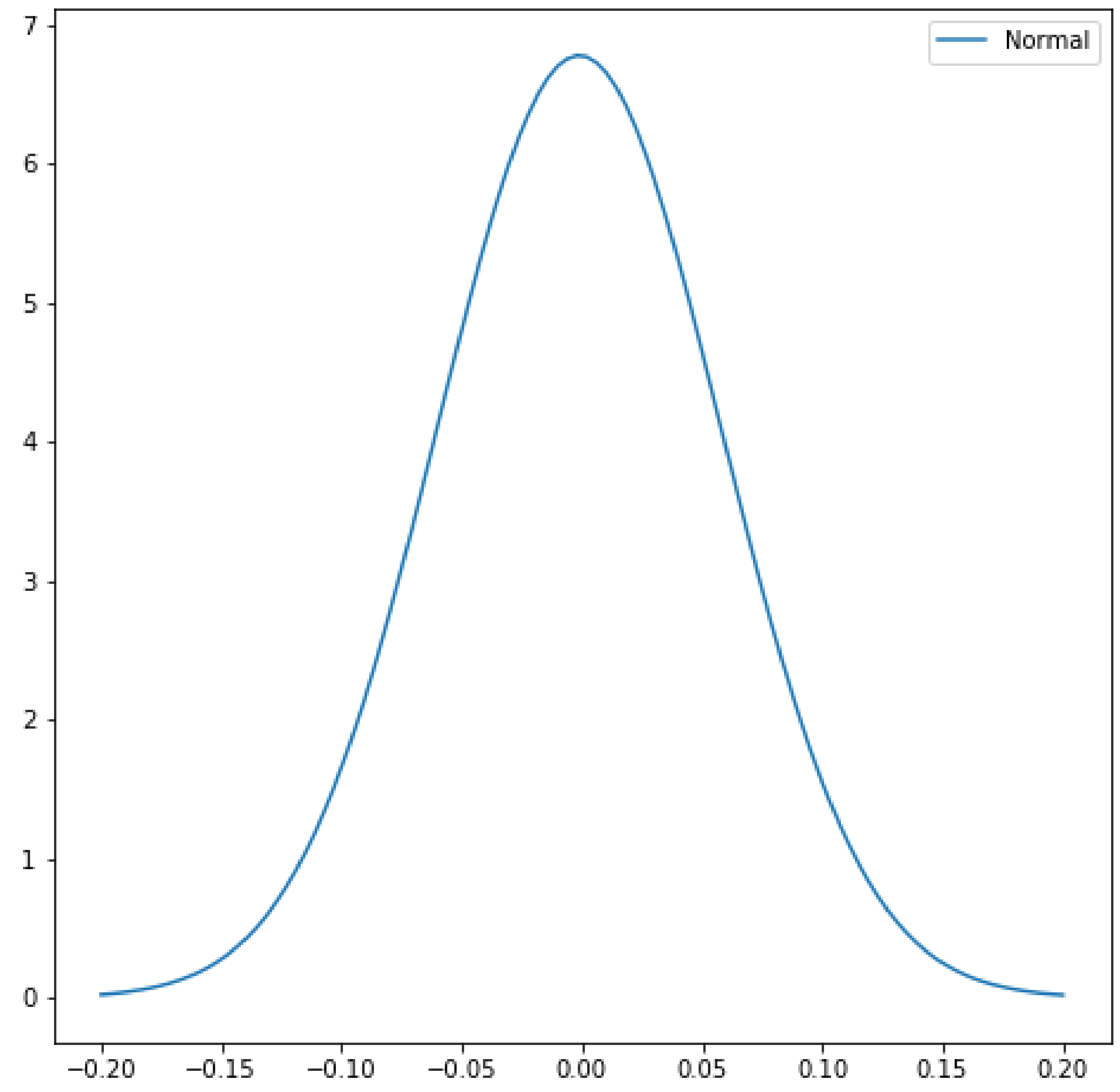
- Statistical test of goodness of fit
  - Test null hypothesis: data are Normally distributed
  - Test statistic rejects Normal distribution if larger than `critical_values`
- Import `scipy.stats.anderson`
- Compute test result using `loss` data

```
from scipy.stats import anderson
anderson(loss)
```

```
AndersonResult(statistic=11.048641503898523,
critical_values=array([0.57 , 0.649, 0.779, 0.909, 1.081]),
significance_level=array([15. , 10. ,  5. ,  2.5,  1. ]))
```

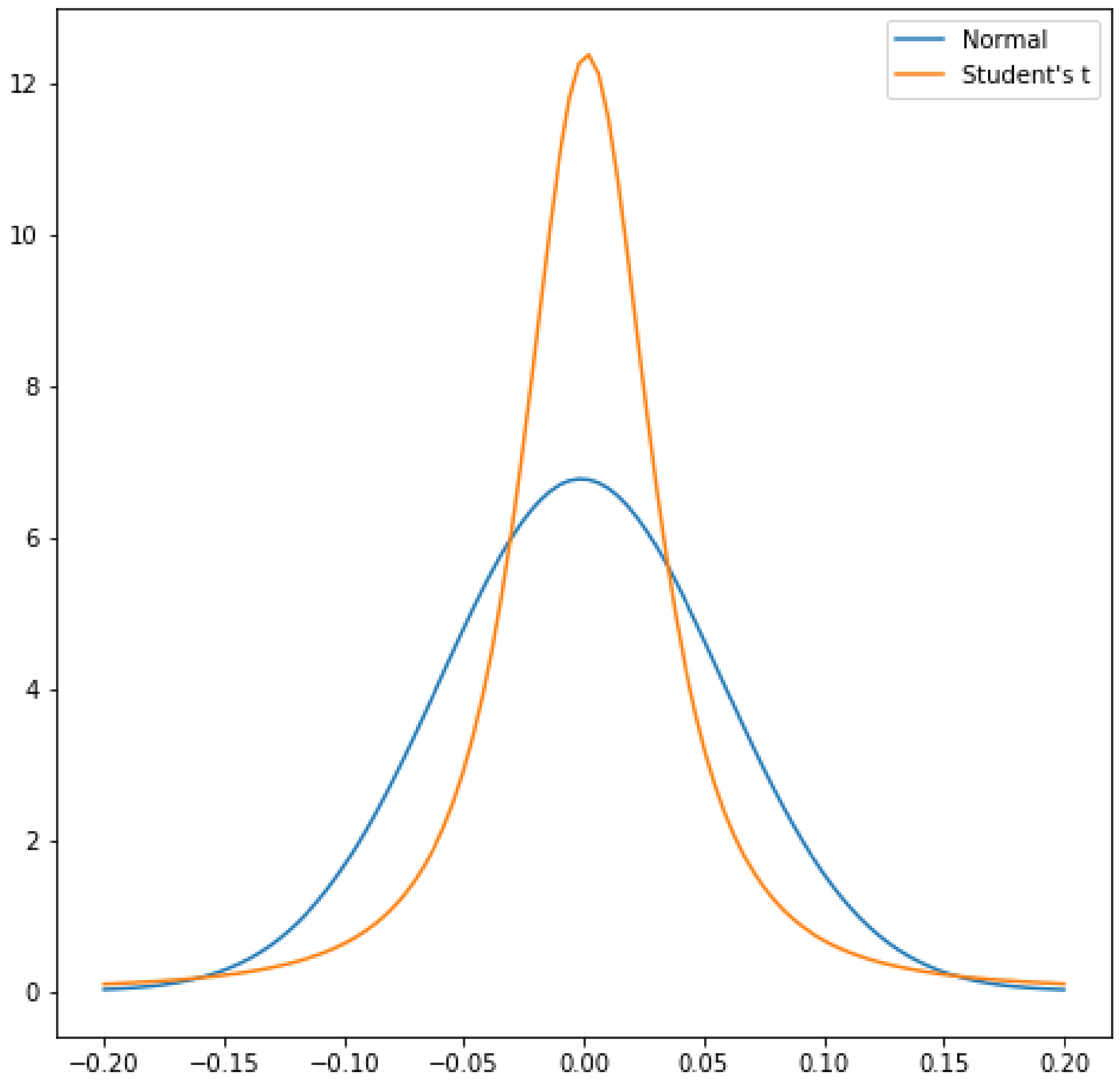
Skewness

- Skewness: degree to which data is non-symmetrically distributed
    - Normal distribution: symmetric



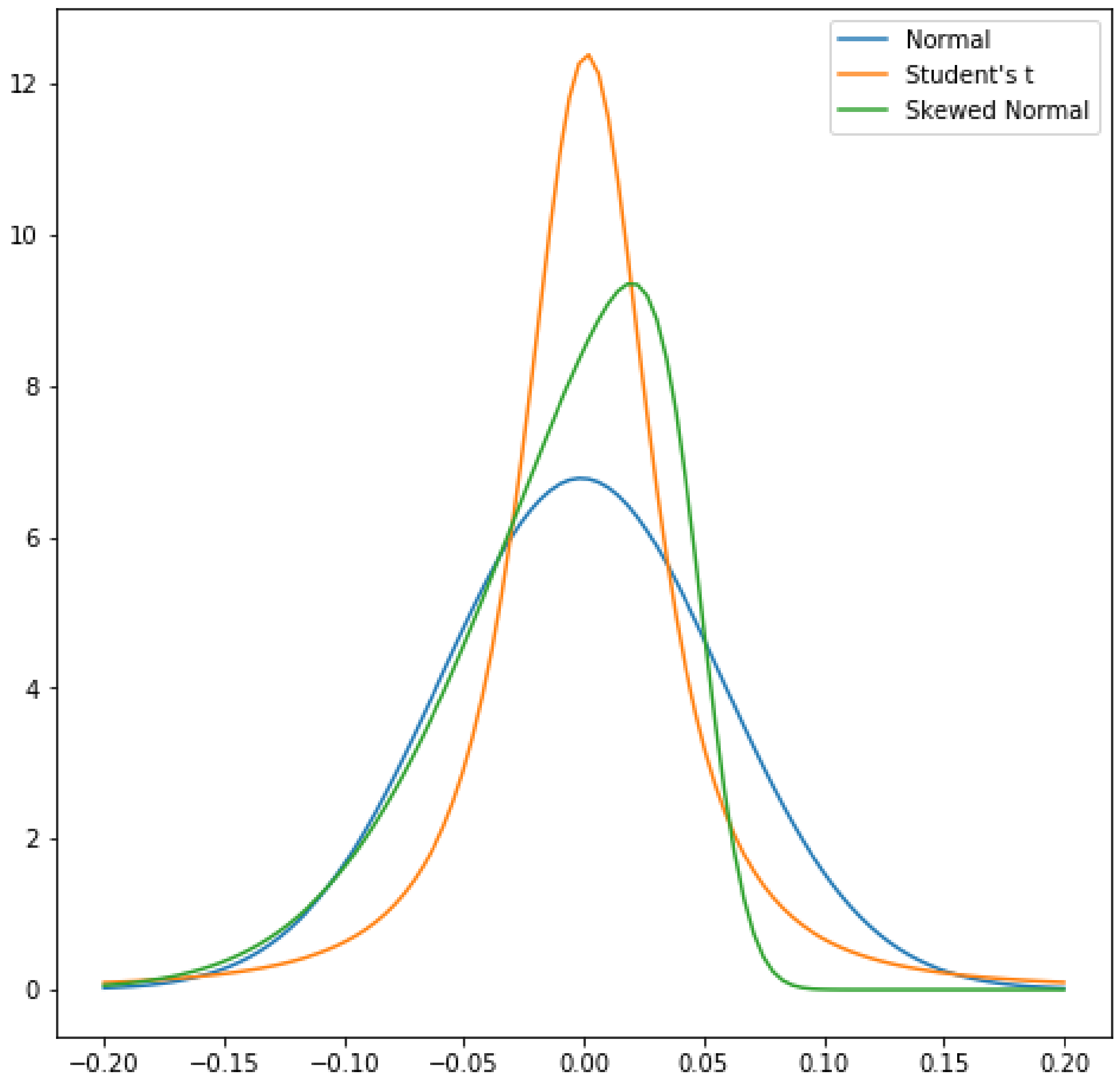
Skewness

- Skewness: degree to which data is non-symmetrically distributed
  - Normal distribution: symmetric
  - Student's t-distribution: symmetric



Skewness

- Skewness: degree to which data is non-symmetrically distributed
  - Normal distribution: symmetric
  - Student's t-distribution: symmetric
- **Skewed Normal** distribution: asymmetric
  - Contains Normal as special case
  - Useful for portfolio data, where e.g. losses more frequent than gains
  - Available in `scipy.stats` as `skewnorm`



Testing for skewness

- Test how far data is from symmetric distribution: `scipy.stats.skewtest`
- *Null hypothesis*: no skewness
- Import `skewtest` from `scipy.stats`
- Compute test result on `loss` data
  - Statistically significant => use distribution class with skewness

```
from scipy.stats import skewtest
skewtest(loss)
```

```
SkewtestResult(statistic=-7.786120875514511,
pvalue=6.90978472959861e-15)
```

Let's practice!

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Historical and Monte Carlo Simulation

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Jamsheed Shorish
Computational Economist

Historical simulation

- No appropriate class of distributions?
- **Historical simulation:** use past to predict future
  - No distributional assumption required
  - Data about previous losses become *simulated* losses for tomorrow

Historical simulation in Python

- **VaR**: start with returns in `asset_returns`
- Compute `portfolio_returns` using portfolio `weights`
- Convert `portfolio_returns` into `losses`
- VaR: compute `np.quantile()` for `losses` at e.g. 95% confidence level
- Assumes future distribution of losses is *exactly* the same as past

```
weights = [0.25, 0.25, 0.25, 0.25]
portfolio_returns = asset_returns.dot(weights)
losses = - portfolio_returns
VaR_95 = np.quantile(losses, 0.95)
```

Monte Carlo simulation

- **Monte Carlo simulation**: powerful combination of parametric estimation and simulation
  - Assumes distribution(s) for portfolio loss and/or risk factors
  - Relies upon random draws from distribution(s) to create random *path*, called a *run*
  - Repeat random draws $\Rightarrow$ creates **set** of simulation runs
- Compute simulated portfolio loss over *each* run up to desired time
- Find VaR estimate as quantile of simulated losses

Monte Carlo simulation in Python

- **Step One:**
  - Import Normal distribution `norm` from `scipy.stats`
  - Define `total_steps` (1 day = 1440 minutes)
  - Define number of runs `N`
  - Compute mean `mu` and standard deviation `sigma` of `portfolio_losses` data

```
from scipy.stats import norm
total_steps = 1440
N = 10000
mu = portfolio_losses.mean()
sigma = portfolio_losses.std()
```

Monte Carlo simulation in Python

- **Step Two:**
  - Initialize `daily_loss` vector for `N` runs
  - Loop over `N` runs
    - Compute Monte Carlo simulated `loss` vector
      - Uses `norm.rvs()` to draw repeatedly from standard Normal distribution
      - Draws match data using `mu` and `sigma` scaled by 1/ `total_steps`

```
daily_loss = np.zeros(N)
for n in range(N):
    loss = ( mu * (1/total_steps) +
      norm.rvs(size=total_steps) * sigma * np.sqrt(1/total_steps) )
```

Monte Carlo simulation in Python

- **Step Three:**
  - Generate cumulative `daily_loss` , for each run `n`
  - Use `np.quantile()` to find the VaR at e.g. 95% confidence level, over `daily_loss`

```
daily_loss = np.zeros(N)
for n in range(N):
    loss = mu * (1/total_steps) + ...
      norm.rvs(size=total_steps) * sigma * np.sqrt(1/total_steps)
    daily_loss[n] = sum(loss)
VaR_95 = np.quantile(daily_loss, 0.95)
```

Simulating asset returns

- **Refinement:** generate random sample paths of *asset returns* in portfolio
  - Allows more realism: asset returns can be individually simulated
  - Asset returns can be *correlated*
    - Recall: efficient covariance matrix `e_cov`
    - Used in Step 2 to compute asset returns
- **Exercises:** Monte Carlo simulation with asset return simulation

Let's practice!

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Structural breaks

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Jamsheed Shorish
Computational Economist

Risk and distribution

- **Risk management toolkit**
  - Risk mitigation: MPT
  - Risk measurement: VaR, CVaR
- **Risk**: dispersion, volatility
  - Variance (standard deviation) as risk definition
- Connection between risk and **distribution** of risk factors as random variables

Stationarity

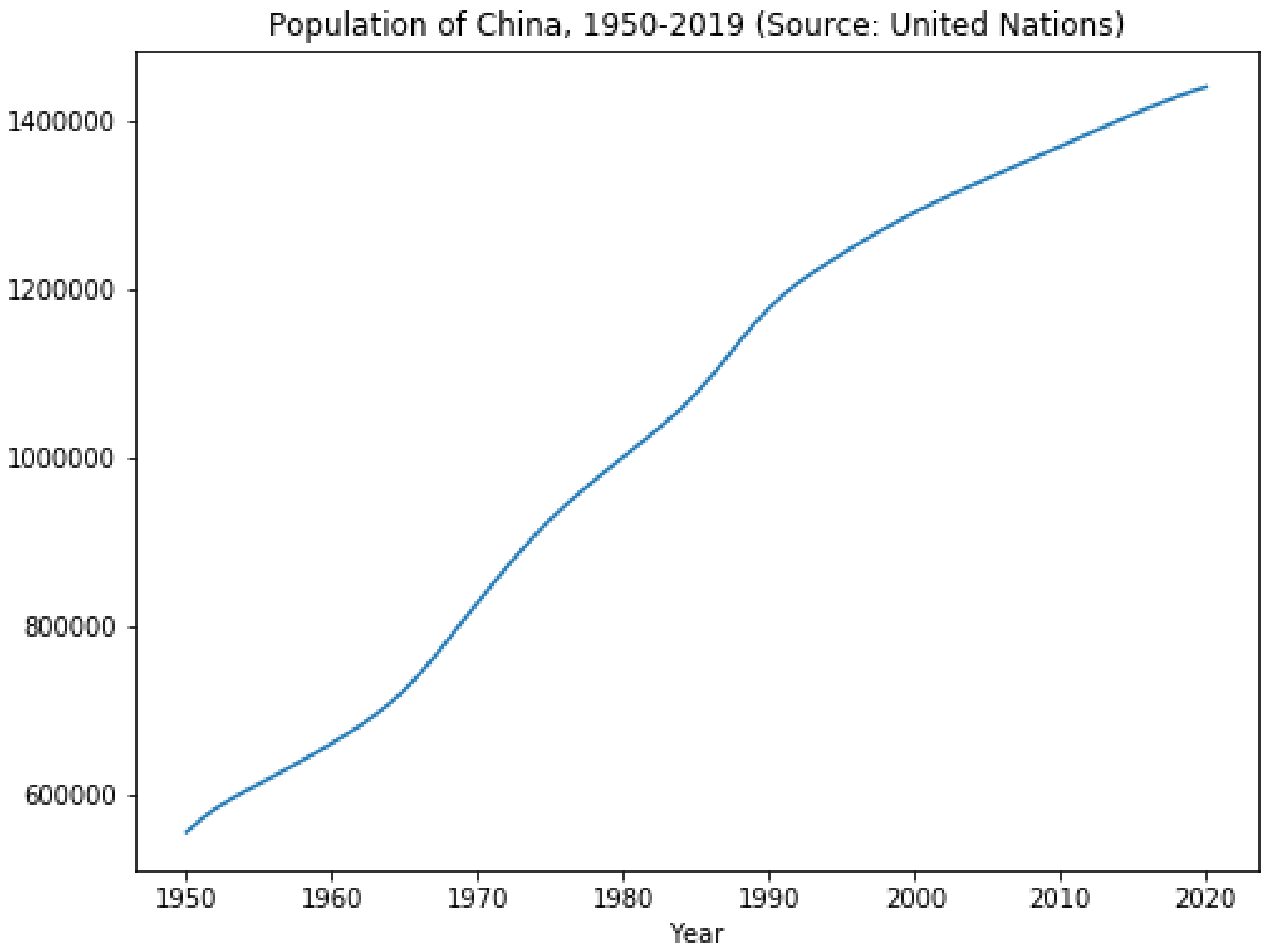
- **Assumption**: distribution is same over time
- Unchanging distribution = **stationary**
- Global financial crisis period efficient frontier
  - *Not* stationary
- Estimation techniques **require** stationarity
  - **Historical**: unknown *stationary* distribution from past data
  - **Parametric**: assumed *stationary* distribution class
  - **Monte Carlo**: assumed *stationary* distribution for random draws

Structural breaks

- **Non-stationary** => perhaps distribution *changes* over time
- Assume specific points in time for change
  - Break up data into *sub-periods*
  - **Within each sub-period, assume stationarity**
- **Structural break(s)**: point(s) of change
  - Change in 'trend' of average and/or volatility of data

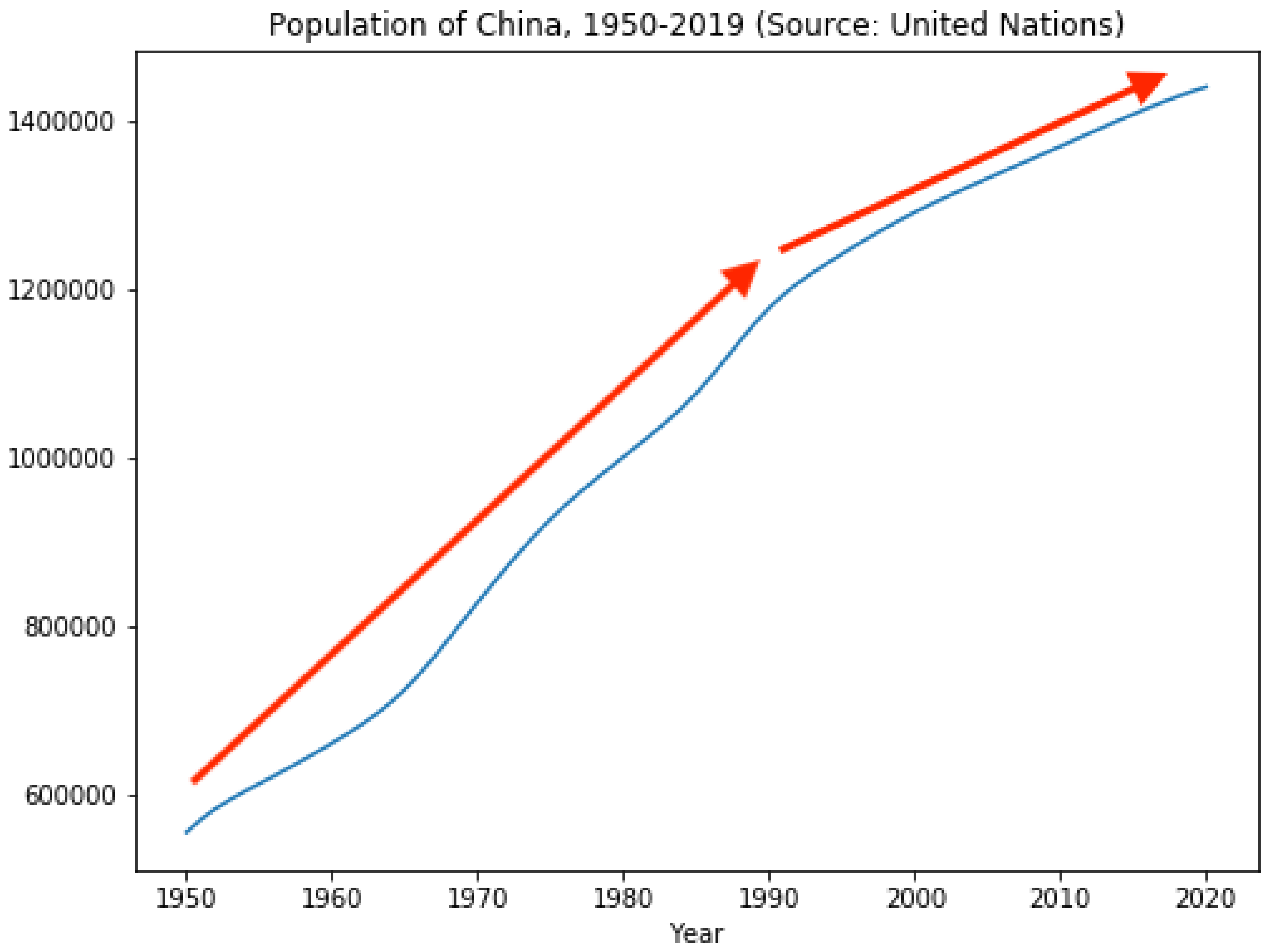
Example: China's population growth

- Examine period **1950 - 2019**
- Trend is roughly *linear*...



Example: China's population growth

- Examine period **1950 - 2019**
- Trend is roughly *linear...*
- ...but seems to slow down from around 1990
- Possible **structural break** near 1990.
- Implies distribution of net population (births - deaths) *changed*
- **Possible reasons:** government policy, standard of living, etc.



The Chow test

- **Previous example:** *visual evidence* for structural break
- **Quantification:** statistical measure
- **Chow Test:**
  - Test for existence of structural break given *linear* model
  - *Null hypothesis*: no break
  - Requires three OLS regressions
    - Regression for *entire* period
    - Two regressions, *before* and *after* break
  - Collect **sum-of-squared residuals**
  - Test statistic is distributed according to **"F" distribution**

The Chow test in Python

- **Hypothesis:** structural break in 1990 for China population
- Assume linear "factor model":

$$\log(\text{Population}_t) = \alpha + \beta * \text{Year}_t + u_t$$

- OLS regression using `statsmodels` 's `OLS` object over full period 1950 - 2019
  - Retrieve sum-of-squared residual `res.ssr`

```
import statsmodels.api as sm
res = sm.OLS(log_pop, year).fit()
print('SSR 1950-2019: ', res.ssr)
```

```
SSR 1950-2019: 0.29240576138055463
```

The Chow test in Python

- Break 1950 - 2019 into **1950 - 1989** and **1990 - 2019** sub-periods
- Perform OLS regressions on each sub-period
  - Retrieve `res_before.ssr` and `res_after.ssr`

```
pop_before = log_pop.loc['1950':'1989']; year_before = year.loc['1950':'1989'];
pop_after  = log_pop.loc['1990':'2019']; year_after =  year.loc['1990':'2019'];
res_before = sm.OLS(pop_before, year_before).fit()
res_after  = sm.OLS(pop_after, year_after).fit()
print('SSR 1950-1989: ', res_before.ssr)
print('SSR 1990-2019: ', res_after.ssr)
```

```
SSR 1950-1989: 0.011741113017411783
SSR 1990-2019: 0.0013717593339608077
```

The Chow test in Python

- Compute the F-distributed **Chow test statistic**
  - Compute the numerator
    - k = 2 degrees of freedom = 2 OLS coefficients α, β
  - Compute the denominator
    - 66 degrees of freedom = total number of data points (70) - 2*k

```
numerator = (ssr_total - (ssr_before + ssr_after)) / 2
denominator = (ssr_before + ssr_after) / 66
chow_test = numerator / denominator
print("Chow test statistic: ", chow_test, "; Critical value, 99.9%: ", 7.7)
```

```
Chow test statistic: 702.8715822890057; Critical value, 99.9%: 7.7
```

Let's practice!

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Volatility and extreme values

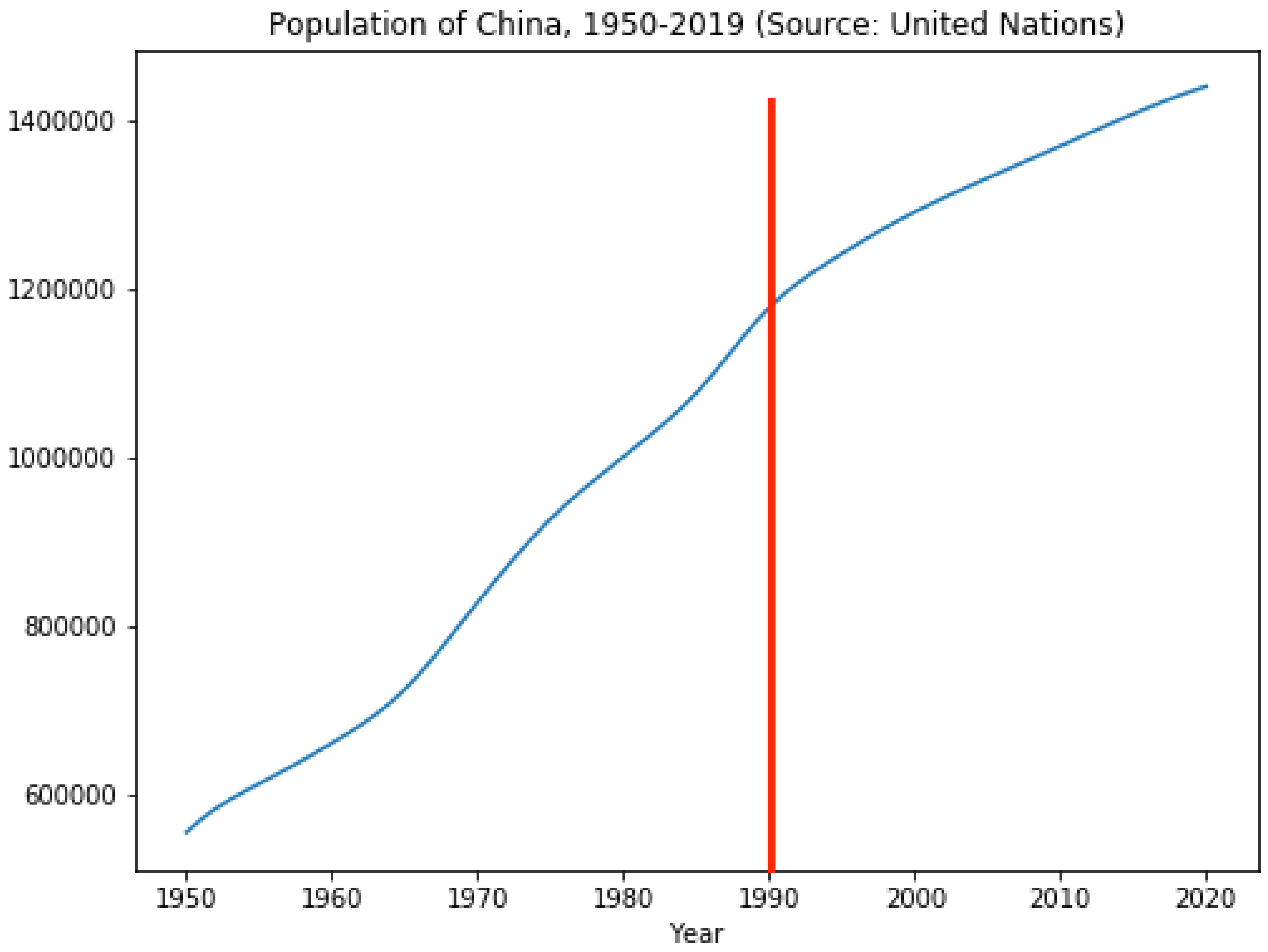
QUANTITATIVE RISK MANAGEMENT IN PYTHON

Jamsheed Shorish
Computational Economist

Chow test assumptions

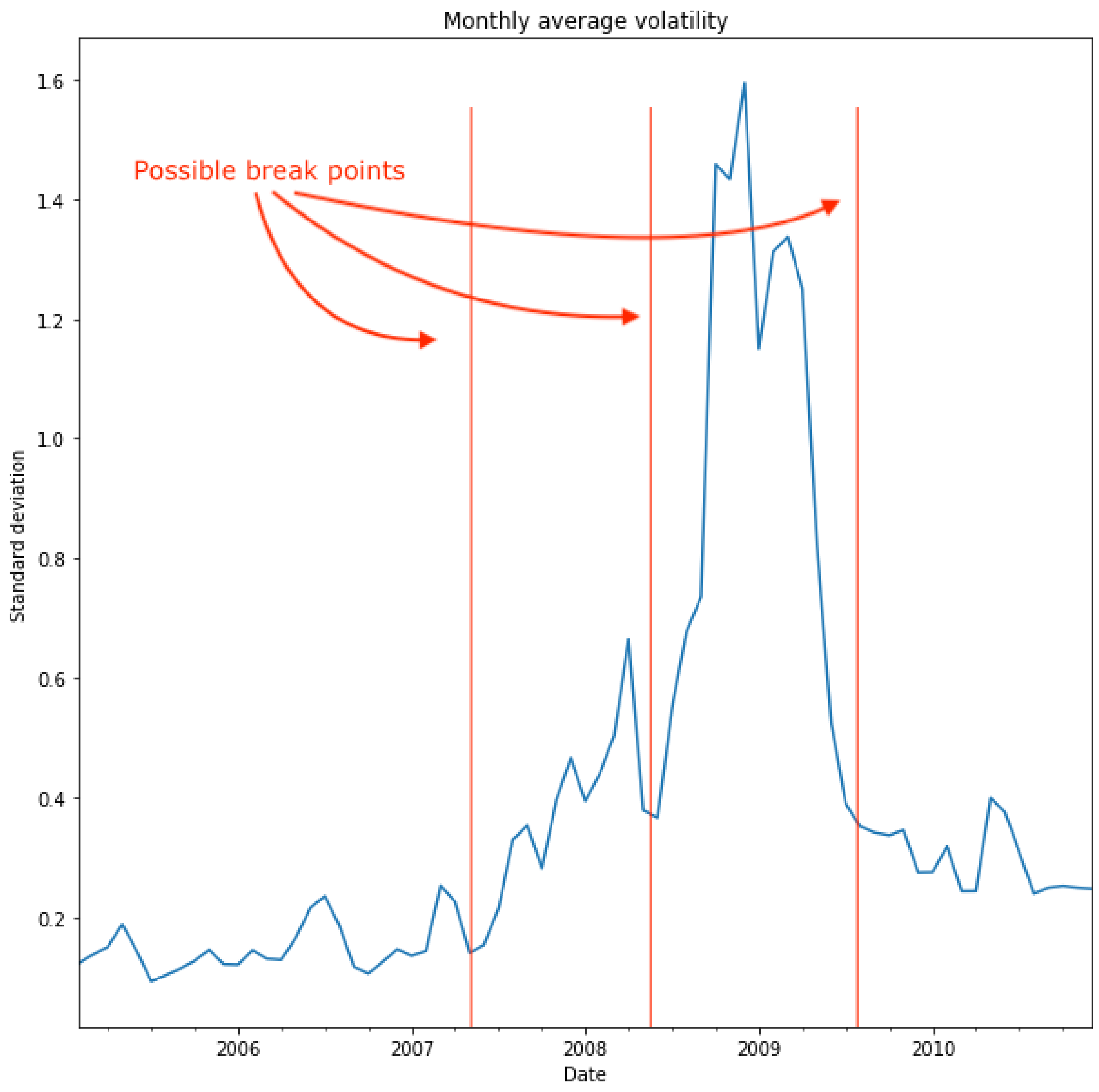
- **Chow test:** identify statistical significance of possible structural break
- **Requires:** *pre-specified* point of structural break
- **Requires:** *linear* relation (e.g. factor model)

$$\log(\text{Population}_t) = \alpha + \beta * \text{Year}_t + u_t$$



Structural break indications

- Visualization of trend may not indicate break point
- Alternative: examine **volatility** rather than trend
  - Structural change often accompanied by greater uncertainty => volatility
  - Allows richer models to be considered (e.g. *stochastic volatility* models)



Rolling window volatility

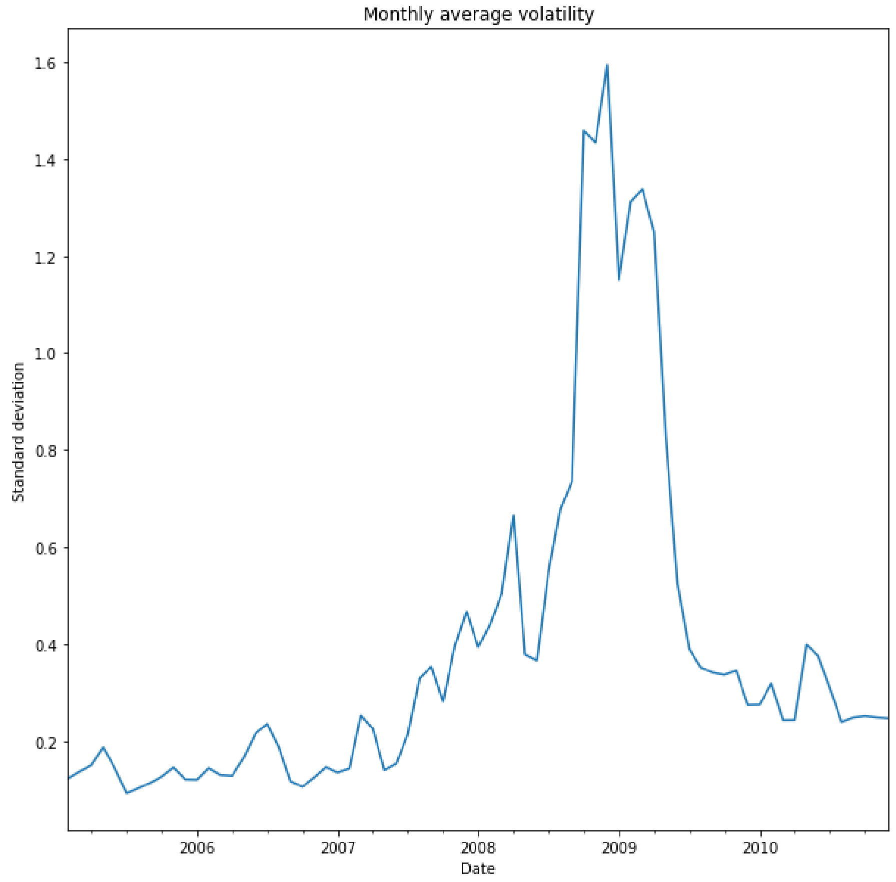
- **Rolling window:** compute volatility over time and detect changes
- **Recall:** 30-day rolling window
  - Create rolling window from ".rolling()" method
  - Compute the volatility of the rolling window (drop unavailable dates)
  - Compute summary statistic of interest, e.g. `.mean()` , `.min()` , etc.

```
rolling = portfolio_returns.rolling(30)
volatility = rolling.std().dropna()
vol_mean = volatility.resample("M").mean()
```

Rolling window volatility

- Visualize resulting volatility (variance or standard deviation)

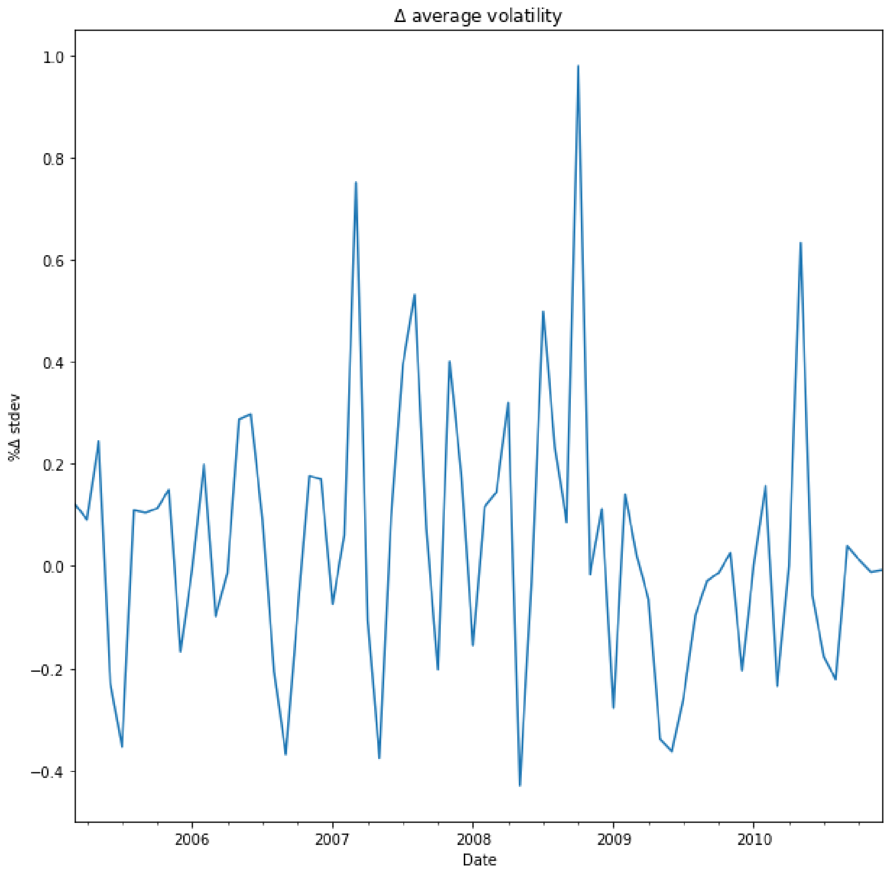
```
import matplotlib.pyplot as plt
vol_mean.plot(
  title="Monthly average volatility"
).set_ylabel("Standard deviation")
plt.show()
```



Rolling window volatility

- Visualize resulting volatility (variance or standard deviation)
- Large *changes in volatility* => possible structural break point(s)
- Use proposed break points in linear model of volatility
  - Variant of Chow Test
- Guidance for applying e.g. ARCH, stochastic volatility models

```
vol_mean.pct_change().plot(
  title="$\Delta$ average volatility"
).set_ylabel("% $\Delta$ stdev")
plt.show()
```

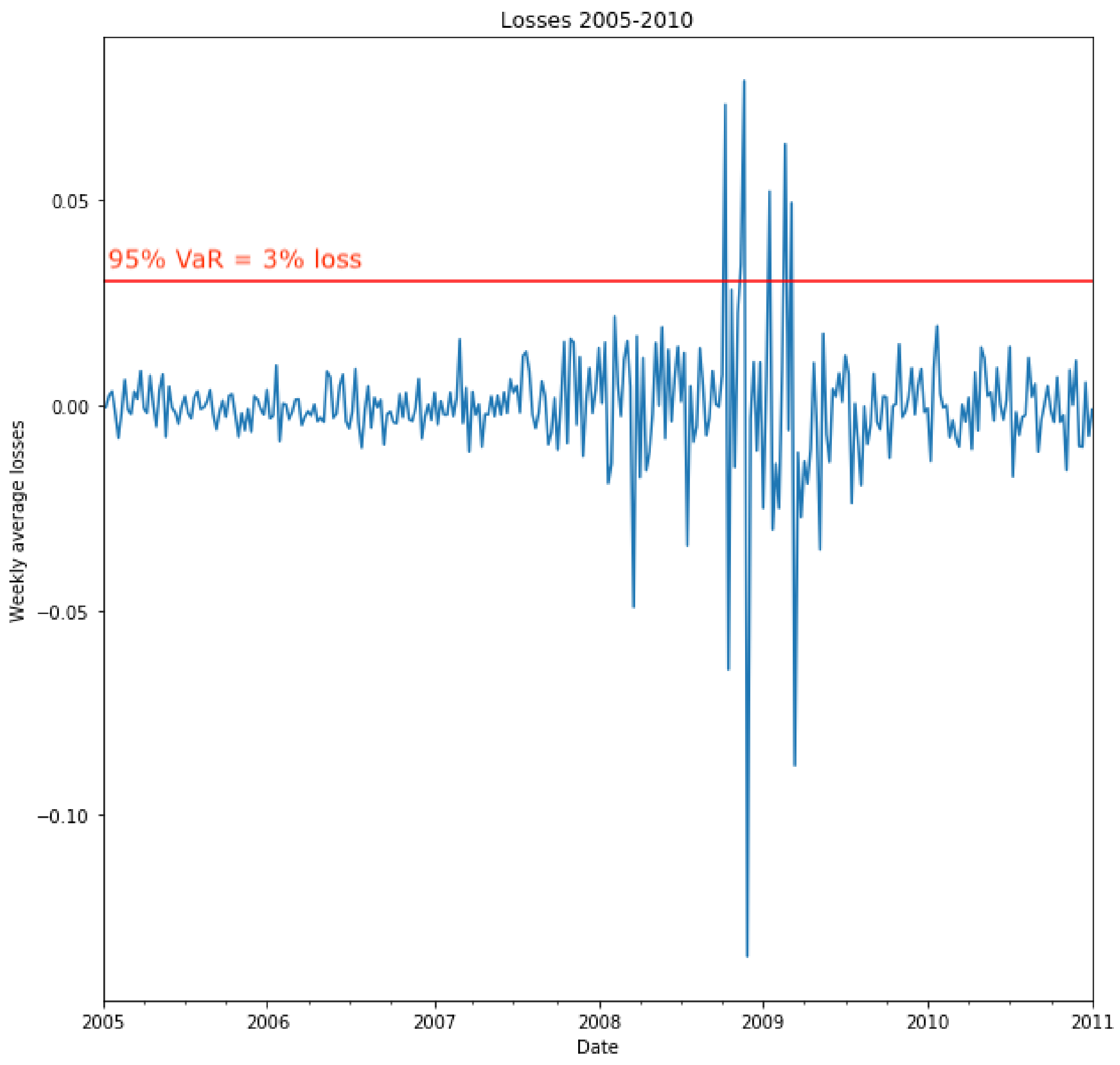


Extreme values

- **VaR, CVaR**: maximum loss, expected shortfall at particular confidence level
- Visualize changes in maximum loss by plotting VaR?
  - Useful for large data sets
  - Small data sets: *not enough information*
- **Alternative**: find losses *exceeding* some threshold
- **Example**: VaR_{95} is maximum loss 95% of the time
  - So 5% of the time, losses can be expected to *exceed* VaR_{95}
- **Backtesting**: use previous data *ex-post* to see how risk estimate performs
  - Used extensively in enterprise risk management

Backtesting

- Suppose $\mathrm{VaR}_{95} = 0.03$
- Losses exceeding 3% are then *extreme values*
- **Backtesting**: around 5% (100% - 95%) of previous losses should *exceed* 3%
  - **More** than 5%: distribution with wider ("fatter") tails
  - **Less** than 5%: distribution with narrower tails
- **CVaR** for backtesting: accounts for tail better than VaR



Let's practice!

QUANTITATIVE RISK MANAGEMENT IN PYTHON