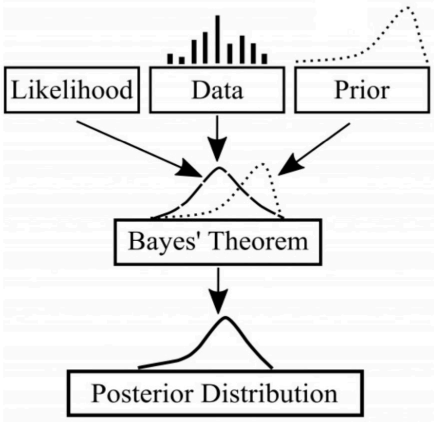
Topic5: Rethinking Statistics with Bayesian Method

1. Establish a belief about the data, including prior and likelihood, We also check if the data is possible given the initial model assumption (Prior Predictive Checks)
2. Update the model with data. Check the updated model agree with the original data (Posterior Predictive Checks)
3. Apply our models, credible interval, forecasting etc.



Package:
PyMC3, ArviZ

Bayesian Theorem: Starting with a belief (distribution, or prior) about parameter, we update belief (posterior) with the arrival of data

$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{\sum P(x|\theta_i)P(\theta_i)}$ $P(\theta)$: prior prob.; $P(x)$: model evidence / marginal likelihood
$P(\theta|x)$ posterior prob.; $P(x|\theta)$: the likelihood of evidence

continuous prior: $P(x) = \int P(x|\theta)P(\theta)d\theta$. the marginal likelihood $\xrightarrow{hard}$ Markov chain Monte Carlo Appox (MCMC)

others like MCMC with Metropolis - Hasting algorithm / Hamiltonian Monte Carlo (HMC) / No-U-Turn Sampler (NUTS)

Bayesian Model of Risk and Reward: 举了个股票的例子. 对Return来说, t-dist with low df $I = \mu + 6t$

p = t.pdf (x, df=2, loc=1, scale=2) 有3个参数: 在没有条件情况下 $\mu \sim$ Normal, sd ~ Uniform.

df ~ exponential (确保df>0) $P(x|\lambda) = \lambda e^{-\lambda x}, x \geq 0$. ($\mu_x = \sigma_x = \frac{1}{\lambda}$, λ: rate parameter) expon.pdf (x, scale = 1/λ)

posterior distribution of Sharpe ratio: annual $SR = \sqrt{252} \times \frac{\text{average daily return}}{\text{sd. of daily return}}$ (just calculation)

Then need to train the model: tune=1000 (1000 iterations) draws: # of sample draw from model's distribution

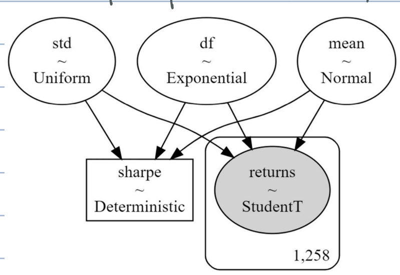
chains: multiple chains and compare the distribution. trace = pm.sample (tune=tune, draws=draws, chains=4)

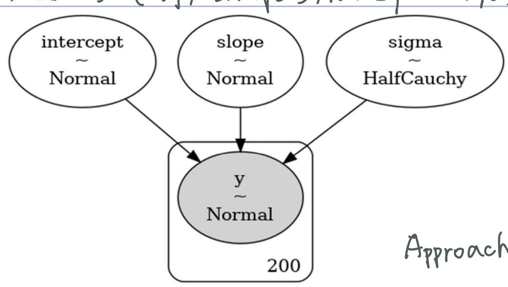
每一个draw是一行. 出现 mean, std, df, sharpe, chain. 总行数是 draws × chains.

to DF: pm.trace_to_dataframe (trace).assign (chain = lambda x: x.index // draws) az.plot_trace (trace, var_name = ("mean", "std"))

Credible Interval: the range containing a particular percentage of probable values.

az.plot_posterior (trace, var_names = ("df", "sharpe"), hdi_prob = .95)





Bayesian Regression:

Approach I: x = np.linspace (0, 1, 200),

μ = beta0 + beta1 × x.

y = μ + np.random.normal (scale=.05, size)

std: pm.HalfCauchy(), Y, β: Normal()

Approach II: pm.glm.GLM.from_formula ('y ~ x', data)

Prior and Posterior Prediction Checks, PPCs

```
size = 200
beta0 = 0.5
beta1 = 3
test=pd.DataFrame()
test["x"]=np.linspace(-10, 10, size)
test["mu"]=beta0 + beta1 * data["x"]
test["y"]=data["mu"]+np.random.normal(scale=6, size=size)
```

Simulating

Prior predictive checks is to generate data according to the prior in order to assess whether a prior is appropriate.

```
with pm.Model() as test_model:
    beta0 = pm.Normal("b0", 0.0, 10.0)
    beta1 = pm.Normal("b1", 0.0, 10.0)

    mu = beta0 + beta1 *test["x"]
    std = pm.Exponential("sd", 1.0)

    output = pm.Normal("obs", mu=mu, sigma=std, observed=test["y"])
```

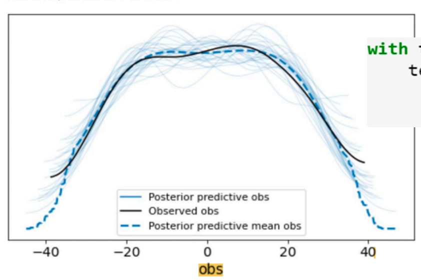
random variable can only take positive number (or half Cauchy distribution)

```
with test_model:
    prior_checks = pm.sample_prior_predictive(samples=100, random_seed=89)
for intercept, slope in zip(prior_checks["b0"], prior_checks["b1"]):
    y =intercept+ slope *test["x"]
    plt.plot(test["x"], y, c="c", alpha=0.1)
```

Posterior predictive checks analyzes the degree to which data generated from the model deviate from data generated from the true distribution, that is, whether the approximated posterior distribution is close to the real one.

```
visual_data = az.from_pymc3(test_trace, posterior_predictive=ppc)
az.plot_ppc(visual_data,num_pp_samples=30)
```

<AxesSubplot:xlabel='obs'>



MCMC

```
with test_model:
    test_trace = pm.sample(draws=1000,
                           tune=2000,
                           random_seed=80)
```

Sampling from Posterior Dist.

the posterior(blue) is same with empirical(black)
我们已知b0，b1，std的分布，则生成很多组，input x生成y和y的density curve（浅蓝色曲线）
将许多条density curve取平均成后验分布（蓝色虚线），将虚线与黑线对比

Robust Regression with Fat tails   like lots of outliers (fat tail)

例子: 比如有些回归在 0 附近 residual var ↑. / noise not normal but t-dist.

```
with pm.Model() as model:
    pm.GLM.from_formula('y ~ x', data2)
    trace3 = pm.sample(tune=1000)
```

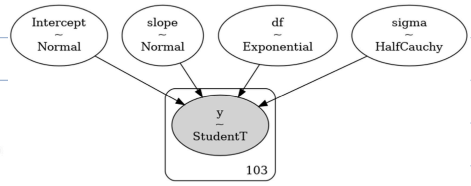
normal ×

```
with pm.Model() as  robust_model:
    f = pm.glm.families.StudentT()
    pm.GLM.from_formula('y ~ x', data2, family=f)
    robust_trace = pm.sample(tune=2000)
```

✓

```
with pm.Model() as my_model:

    std = pm.HalfCauchy('sigma', beta=10, testval=1)
    beta0 = pm.Normal('Intercept', 0, sd=20)
    beta1 = pm.Normal('slope', 0, sd=20)
    df = pm.Exponential('df', 1 / 29,testval=5) + 2.# make the df is around 2
    # testval is the initial value to start sample.
    # likelihood
    likelihood = pm.StudentT('y',nu=df, mu=beta0 + beta1 *data2["x"],
    sd=std, observed=data2["y"])
```



Topic6: Time Series Models

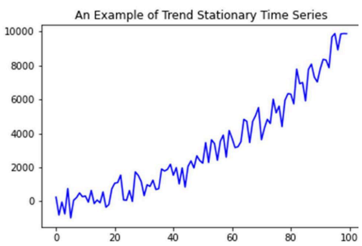
Definition:

- Stationarity means that the statistical properties of a time series y_t, i.e. mean $\mu(y_t)$, standard deviation $\sigma(y_t)$ and correlation $\rho(y_t, y_{t-k})$, $k = 1,2, \ldots$ do not change over time.

Trend Stationary

Definition:

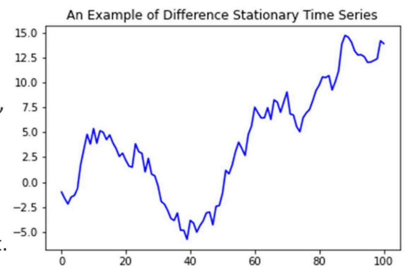
- $y_t = g(t) + x_t$ in which the trend $g(t)$ is deterministic function of time t and x_t is stationary. y_t is stationary around the deterministic trend. After the trend is removed, the series will be stationary. KPSS test can tell whether the series is stationary/trend stationary.



Difference Stationary

Definition:

- $y_t = y_{t-1} + x_t$ in which x_t is stationary. Difference Stationary series (Also called unit root process) can be made strict stationary by differencing. ADF test be applied to test



Stationary process (time series):

- i.i.d process: $\varepsilon_t, t = 1,2, \ldots$
- Auto-correlated stationary process: $y_t = 0.5 * y_{t-1} + \varepsilon_t$, AR, MA, ARMA, etc.

Nonstationary process:

- Difference stationary(Unit root process): $y_t = y_{t-1} + x_t$, $y_t = 2y_{t-1} - y_{t-2} + x_t$, x_t is stationary process.
  - Random walk: $y_t = y_{t-1} + \varepsilon_t$
- Trend stationary: $y_t = g(t) + x_t$
- Mixed non-stationary: $y_t = g(t) + r_t + x_t$, where r_t is a random walk.

How to test stationarity of time series? unit root test
Augmented Dickey Fuller ("ADF") test and
Kwiatkowski-Phillips-Schmidt-Shin ("KPSS") test.

KPSS Test: based on $y_t = r_t + g(t) + x_t$ (underlined assumption)   test whether var of random walk is not 0.

$g(t)$: deterministic trend; r_t: random walk; x_t: stationary component.

{ H_0: trend-stationary
{ H_1: has unit root

but! the default option of KPSS test is $g(t) = c$. test the stationary.

{ kpss_test(A) p=0.01 reject stationary
{ kpss_test(A, null="ct") p=0.10. can not reject trend stationary

ADF Test: based on $y_t = g(t) + a y_{t-1} + \varepsilon_t$ test whether $a=1$ or not (like KPSS alternative)

{ H_0: the process has a unit-root   But default option of ADF test is $g(t) = c$. test the stationary.
{ H_1: no unit root.

{ adf_test(A) p=0.86 can not reject
{ adf_test(A, reg="ct") p=3e-16 reject have unit-root.

ADF test: you can't reject H_0; KPSS test: reject H_0. Both imply that series has unit root.

ADF test: Reject H_0. KPSS test: don't reject H_0. Both imply that series is stationary or trend stationary depending on the model you use.

Can't reject H_0 for both ADF and KPSS tests: data give not enough observations.

Reject H_0 for both ADF and KPSS tests: You need other tests to verify the assumptions about the model structure.

Decomposition:

- trend stationary $y_t = \overbrace{T_t + S_t}^{\mu_t} + x_t$ T_t: long-term trend S_t: seasonal component

  μ_t: deterministic mean trend. x_t: stationary process with mean 0

- difference stationary $\Delta y_t = y_t - y_{t-1}$ / $\Delta^2 y_t = \Delta y_t - \Delta y_{t-1} = y_t - 2y_{t-1} + y_{t-2}$ / $\Delta^3 y_t$

Accuracy Measures:

$$RMSE = \sqrt{\frac{\sum (y_{n+i} - \hat{y}_{n+i})^2}{n}}$$ ($\hat{y}_{n+i}$ or y_{n+i-1})

$$\text{Normalized RMSE (NRMSE)} = \frac{RMSE}{y_{max} - y_{min}}$$

```
# Naive Benchmark
apple1["Change"]=apple1["Close"]-apple1["Close"].shift(1)
apple1["Y"]=apple1["Change"].shift(-1)
apple1["Yhat"]=apple1["Change"]
```

Modeling Stationary Time Series

Autoregressive (AR) model: AR(p) capture the linear dependence

$y_t = \phi_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$, $\varepsilon_t \sim i.i.d.$

所有的根都要小于等于1，则AR(p)是Stationary： $1 - \phi_1 x^{-1} - \phi_2 x^{-2} - \cdots - \phi_p x^{-p} = 0$

如果有r个根为1，要做d次差才能平稳 (the process is intergrated denoted I(r))

Moving Average (MA) Model: MA(q) based on past noises. always stationary: weighted sum of noise

$y_t = \theta_0 + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$ $\varepsilon_t \sim i.i.d$

ARMA Model $y_t = AR(p) + MA(q)$ $y_t = \phi_0 + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$ $\varepsilon_t \sim i.i.d$

- The autocorrelation coefficient function (ACF) $\rho_K = Corr(y_t, y_{t-K})$
- PACF: direct corr between observation and indirect influence of time $\rho_K^p = Corr(y_t, y_{t-K} \mid y_{t-1}, \ldots, y_{t-K+1})$

MA(q): ACF截尾，$\rho_K = 0$ for $K > q$；PACF不截，ρ_K^p decays exponentially.

AR(p): PACF截尾，$\rho_K^p = 0$ for $K > q$；ACF不截，ρ_K decays exponentially.

ARMA: ACF/PACF都不截尾

如何选择ARMA(p,q)？ comparing $MSE = \frac{\sum (y_t - \hat{y}_t)^2}{n}$ ↗ smaller & simpler

param= BestARMA(apple["Return"], trainsize = 30, comparesize = 50) { how many days/minute to train; want to roll forward and make predictions

param["RMSE"].sort_values()