1. **random**
2. **sorting**
3. **vectorization**
4. **Stats related**

3a_basic_tools

Code

1. **random**

1.1. random sample

1. seed: `set.seed(13579)`
2. `sample()`
   1. syntax
      `sample(x, size, replace = FALSE, prob = NULL)`
      `sample.int(n, size = n, replace = FALSE, prob = NULL, useHash = (n > 1e+07 && !replace && is.null(prob) && size <= n/2))`
   2. usage

```
# basics
sample(c(1, 2, 3), size=2)
sample(52, size = 5, replace=F)   # means sample from 1:52

# without size param: means sample same num of (permutation)
sample(10)  # permutation of 1:10 (since replace=F)

# sample with probability
sample(c(-1, 0, 1), size=2, prob=c(0.25, 0.5, 0.25))
```

1.2 Random distributions

1. Supported distributions

Distribution	R name	Additional arguments
Beta	beta	shape1, shape2, ncp
Binomial	binom	size, prob
Cauchy	cauchy	location, scale
Chi-square	chisq	df, ncp
Exponential	exp	rate
F	f	df1, df2, ncp
Gamma	gamma	shape, scale
Geometric	geom	prob
Hypergeometric	hyper	m, n, k
Log-normal	lnorm	meanlog, sdlog
Logistic	logis	location, scale
Negative binomial	nbinom	size, prob
Normal	norm	mean, sd
Poisson	pois	lambda
Student's t	t	df, ncp
Uniform	unif	min, max
Weibull	weibull	shape, scale
Wilcoxon	wilcox	m, n

   1. Notes
      1. For uniform distribution, default min, max is 0, 1
      2. For normal distribution, default mean = 0, sd (sqrt(var)) = 1.
         Also, when writing `N(1, 4)`, mean is 1, sd is sqrt(4) = 2!
2. Supported functions for these distributions
   When using, just `<R_function_name><R_dist_name>`
   Math def: Lec2 P22
   1. sample random number from the distribution:
      R_function_name: `r` (random)
      eg. `runif(n, min = 0, max = 1)`
   2. Probability density function (pdf) for discrete or Probability mass function (pmf) for continuous R_function_name: `d` (density) eg.
      `dnorm(seq(-4, 4, 0.1))` ; `dbinom(0:20,size=20,prob=1/4)`
   3. Cumulative probability distribution function (cdf) R_function_name: `p` (probability) eg. `pnorm(1.96)` : P(x<1.96)
   4. Quantiles R_function_name: `q`
      eg. `qnorm(0.25)` : x such that P(X<x) = 0.25
      eg. `qnorm(0.5) = 0.00`

1.3 Examples

1. eg of random walk and plotting as time series
   Lec2 P12-14
2. eg of Mont Carlo:
   Intro Lec2 P16-17, 31-

1. Critical value approach for testing if is the same distribution, at 95% significant level

```
s1 <- sd(x)
s2 <- sd(y)
n1 <- length(x)
n2 <- length(y)
PooledSD <- sqrt(((n1-1)*s1^2+(n2-1)*s2^2)/(n1+n2-2))

(t <- ((mean(x)-mean(y)))/(PooledSD*sqrt(1/n1+1/n2))) # t-statistic
qt(.975,n1+n2-2)  # critical value (t_{\alpha/2})
(abs(t) > qt(.975,n1+n2-2)) # if TRUE, we should reject H0
```

2. **sorting**

1. sort `sort(x, decreasing = FALSE, ...)`

```
> sort(c(3, 12, 2, 1))
[1]  1  2  3 12
```

   1. give sort order

```
# 2 methods both OK
> order(c(3, 12, 2, 1))
[1] 4 3 1 2

> sort.list(c(3, 12, 2, 1))
[1] 4 3 1 2
```

3. **vectorization**

`<CCLD>` : **If** is vectorization to vector, use `ifelse` or `vapply()` ; **If** is matrix use `outer` to do mask first**.**

1. vector
2. use access expressions to modify

```
v[v>0] <- v[v>0] * 2

women$height[women$weight > 140]
```

3. `ifelse()` is a vectorization tool
4. generally `vapply()`
   1. idea: can use the vapply() function to vectorize the operation of a function that does not support vectorization**.**
   2. syntax

```
vapply(X, FUN, FUN.VALUE, ...)
```

      1. X is vector, cannot be matrix
      2. FUN.VALUE specifies the output format numeric(1) indicating floating point format**.** Other possible output format includes: logical(1), integer(1), character(1)
   3. eg

```
is.positive <- function(x) {
    if (x > 0) 1 else 0
}

vapply(x, is.positive, FUN.VALUE=numeric(1))
```

      1. note:
         numeric(1) indicating floating point format**.** Other possible output format includes: logical(1), integer(1), character(1)
5. matrix, dataframe and other 2D f
   1. `outer` : Use it to generate masks**.** See matrix section
   2. `apply` : see section 2b 2.4

4. **Stats related**

1. basic stats values calculation: see section 2a 3.1**.**
2. Egs