R Statistics

[**1. Qualitative Stats**](#_e46lvv154s6u) **3**

[**2. Quantitative Stats**](#_fbs3dboaaeol) **3**

[**3. Association and Correlation**](#_k2t4z1zgebht) **3**

[**4. Combinatorial**](#_4d5qslprxcz4) **3**

[**5. Distribution**](#_2o10923vlbc2) **4**

[Asymmetry Method](#_ues85yyiphd6) 4

[Distributions](#_vn2vqalwn791) 4

[**6. Sampling**](#_118sbhbt9u7f) **8**

[**7. Stats Tests**](#_7mc0h93nnh13) **8**

[**8. Multivariate Analysis**](#_h4wvk9fmar5) **8**

# Qualitative Stats

Frequency Table: table(df$col)

Contingency Table: table(df$col1, df$col2)

Proportions of a Freq Table: prop.table(freqTable)

Proportions of a Cont Table: prop.table(contTable)

Freq Table Margins: margin.table( df|table, 1|2 ) *- the total sum; 1 for row, 2 for cols*

Cross Tabulation: CrossTable() *- can tests factor indep*

# Quantitative Stats

Arithmetic Average: mean(vector)

Median: median(vector)

Mode: mode <- function(data) { vector = table( as.vector(data) )

names(vector)[ vector == max(vector) ] }

Standard Deviation: sd(vector)

Variance: var(vector) Coefficient of variation: CV = ( sd/avg ) \* 100

Quartile: quantile(vector) Interquartile Range: IQR( vector, na.rm=FALSE )

Percentile: quantile( vector, c(.X) ) #percentil X

# Association and Correlation

*Parameters ‘params’:* can be *‘vector1, vector2’* OR ‘*df’* OR *‘table’*

Chi-Squared: chisq.test( params )

Conti Coeff.: ContCoef( params ) *- package ‘DescTools’*

Phi Coeff.: Phi( params ) *- package ‘DescTools’*

Cramer’s Coeff.: CramerV( params ) *- package ‘DescTools’*

Correlation Coef.: cor( params, method=“person” ) *- or ”kendal”,”spearman”*

Covariance: cov( params, method=“person” ) *- or ”kendal”,”spearman”*

# Combinatorial

Factorial: factorial(n)

Simple Combination: choose(k,n)

# Distribution

## 

## Asymmetry Method

both from package *‘e1071 or moments’*

Skewness Coefficient

sk = skewness(vector)

Kurtosis Coefficient

ck = kurtosis(vector)

## 

## 

## Distributions

Can prefix a *value* to the function name, the values are:

**d:** get probability density values *- PDF*

**p:** get cumulative probabilities *- CDF*

**q:** get quantum values

**r:** get random numbers from the distribution

Bernoulli Dist.

dbern(x, prob, log = FALSE)

pbern(q, prob, lower.tail = TRUE, log.p = FALSE)

qbern(p, prob, lower.tail = TRUE, log.p = FALSE)

rbern(n, prob)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*prob:* probability of success in each trial *- 0<prob<=1*

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

Geometric Dist.

dgeom(x, prob, log = FALSE)

pgeom(q, prob, lower.tail = TRUE, log.p = FALSE)

qgeom(p, prob, lower.tail = TRUE, log.p = FALSE)

rgeom(n, prob)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*prob:* probability of success in each trial *- 0<prob<=1*

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

Binomial Dist.

dbinom(x, size, prob, log = FALSE)

pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)

qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)

rbinom(n, size, prob)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*size:* number of trials

*prob:* probability of success on each trial

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

Poisson Dist.

dpois(x, lambda, log = FALSE)

ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)

qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)

rpois(n, lambda)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*lambda:* event rate per unit

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

Uniform Dist.

dunif(x, min = 0, max = 1, log = FALSE)

punif(q, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)

qunif(p, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)

runif(n, min = 0, max = 1)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*min, max:* lower/upper limits of the distribution *- must be finite*

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

Normal Dist.

dnorm(x, mean = 0, sd = 1, log = FALSE)

pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)

qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)

rnorm(n, mean = 0, sd = 1)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*lambda:* event rate per unit

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

Exponential Dist.

dexp(x, rate = 1, log = FALSE)

pexp(q, rate = 1, lower.tail = TRUE, log.p = FALSE)

qexp(p, rate = 1, lower.tail = TRUE, log.p = FALSE)

rexp(n, rate = 1)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*rate:* vector of rates

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

Gamma Dist.

dgamma(x, shape, rate = 1, scale = 1/rate, log = FALSE)

pgamma(q, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)

qgamma(p, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)

rgamma(n, shape, rate = 1, scale = 1/rate)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*shape, scale:* must be positive

*rate:* an alternative way to specify the scale

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

Chi-Square Dist.

dchisq(x, df, ncp = 0, log = FALSE)

pchisq(q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)

qchisq(p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)

rchisq(n, df, ncp = 0)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*df:* degree of freedom

*ncp:* non-centrality parameter *- non-negative*

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

t Student Dist.

dt(x, df, ncp, log = FALSE)

pt(q, df, ncp, lower.tail = TRUE, log.p = FALSE)

qt(p, df, ncp, lower.tail = TRUE, log.p = FALSE)

rt(n, df, ncp)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*df:* degree of freedom *- maybe non-negative and >0*

*ncp:* non-centrality parameter

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

F Dist.

df(x, df1, df2, ncp, log = FALSE)

pf(q, df1, df2, ncp, lower.tail = TRUE, log.p = FALSE)

qf(p, df1, df2, ncp, lower.tail = TRUE, log.p = FALSE)

rf(n, df1, df2, ncp)

*x, q:* vector of values that you’re search for

*p:* vector of probabilities --- *n:* numb of observations to return

*df1, df2:* degrees of freedom *- Inf is allowed*

*ncp:* non-centrality parameter

*log, log.p:* logical; if TRUE, probabilities p are given as log(p)

*lower.tail:* logical; if TRUE, probabilities are $P[X < x]$, otherwise, $P[X > x]$

# Sampling

Random Samples: sample(df, size, replace=FALSE, prob=NULL) *- also for Permutations*

Systematic: df[S.SY( nrow(df), Kth ) *- package “TeachingSamplig”*

Bootstrapping: bootstraps(df, times=n\_samples)

Stratified: strata(df, c(“ColName”), size=c(n1,n2), method=”srswor”) *- package “sampling”*

Cluster: cluster(df, clustername=c(), size=n\_cluster, method=”srswor”)

*srswor - simple random sampling without replacement*

*srswr - simple random sampling with replacement*

*poisson - poisson sampling*

*systematic - systematic sampling*

# Stats Tests

*Parameter ‘params’* can be: *‘vector1, vector2’* OR ‘*df’* OR *‘table’*

*Parameter ‘formula’ is: ‘df$Col1, df$Col2’*

General Linear Hypotheses: glht( test, linfct, alternative=c(‘two.sided’) )

Confidence Interval: CI( x, ci=0.95 ) *- ci of 95%* *(we are 95% sure that the population mean falls in the range)*

Chi-Squared Test: chisq.test( params )

Sign Test: SIGN.test( formula, conf.level=0.95, mu=u/0, alternative=c(‘two.sided’) )

Wilcoxon Test: wilcox.test( df$Col1, conf.level=0.95, mu=u/0, alternative=c(‘two.sided’) )

Man-Whitney: wilcox.test( formula, conf.level=0.95, mu=u/0, alternative=c(‘two.sided’) )

Kruskal-Wallis: kruskal.test( variable [~ factor1 [+ factor2 [+ …]]], df )

Friedman Test: friedman.test( variable [~ factor1 [| factor2 [| …]]], df )

t-Student Test: t.test( df$Col1, [df$Col2,] conf.level=0.95, mu=u/0, alternative=c(‘two.sided’),

paired=FALSE )

Z Test: z.test( df$Col1, [df$Col2,] conf.level=0.95, mu=u, alternative=c(‘two.sided’),

sigma.x, sigma.y ) *- both sigma is the std of x and y*

*two.sided - bilateral test*

*greater - unilateral, left tail*

*less - unilateral, right tail*

ANOVA: aov( variable ~ factor1 [+ factor2 [+ …]], df )

# Multivariate Analysis

PCA: prcomp( df\_var\_num, scale=FALSE ) *- use summary(mod\_pca) to see more*

Scree Test: scree( df\_var\_num )

Bartlett’s Test: cortest.bartlett( df\_var\_num )

KMO Test: KMO( df\_var\_num )

MANOVA: manova( variable ~ factor1 [+ factor2 [+ …]], data )

1. …