Syllabus :: Module -3 (Statistics with R- Apply Level)

Analyzing Data, Summary statistics, Statistical Tests- Continuous Data, Discrete Data, Power tests, Common distributions- type arguments. Probability distributions, Normal distributions

---

Overview of Textbook Contents Aligned with the Syllabus [Module 3]

The textbook dedicates **Part V to "Statistics with R"**, which directly addresses the module's title.

- **Analyzing Data** is covered in **Chapter 16**. This chapter introduces various techniques for analyzing data in R and highlights that many of the functions discussed are useful for preparing data for further analysis or serve as building blocks for other analyses. It discusses **summary statistics**, including functions like `mean`, `min`, and `max`. The chapter also covers other analytical techniques such as correlation and covariance, principal components analysis, factor analysis, and bootstrap resampling.

- **Summary statistics** are explicitly addressed in **Chapter 16** under the section "Summary Statistics". The book details how to calculate various descriptive statistics using built-in R functions.

- **Statistical Tests** are discussed in **Chapter 18, "Statistical Tests"**. This chapter differentiates between tests for **continuous data** and **discrete data**. For continuous data, it covers topics such as t-tests, variance tests (like F-tests and Bartlett's test), and non-parametric tests (like Wilcoxon tests and Kruskal-Wallis). For discrete data, it discusses tests for proportions and contingency tables (like chi-squared tests and Fisher's exact test). While the syllabus mentions "Power tests", the textbook does not explicitly have a section dedicated to 'power tests' within this chapter, although the concept of power is related to the design and interpretation of statistical tests generally discussed.

- **Common distributions** and **Probability distributions** are covered in **Chapter 17, "Probability Distributions"**. This chapter explains the use of R for working with various statistical distributions. It details common distribution-type arguments and discusses families of distribution functions. The chapter provides information on different distribution families, including the **normal distribution**, and lists the R functions associated with each for calculating probabilities, quantiles, densities, and for random number generation (with prefixes like `d`, `p`, `q`, and `r`).

- The **Normal distribution** is specifically discussed as the first example in **Chapter 17**, detailing the associated R functions (`dnorm`, `pnorm`, `qnorm`, `rnorm`) and their usage .

In summary, Module 3's topics are thoroughly covered within Part V of the "R in a Nutshell" textbook, providing explanations and R functions for analyzing data, calculating summary statistics, performing various statistical tests for both continuous and discrete data, and working with common and probability distributions, with a specific focus on the normal distribution.

Analyzing Data

Based on the "R in a Nutshell 2nd edition" textbook, **analyzing data in R** is a central theme of **Chapter 16, titled "Analyzing Data,"** which is part of **Part V, "Statistics with R"**. This chapter introduces a variety of techniques that are fundamental for understanding and preparing data for more advanced statistical modeling or visualization.

Here's a detailed overview of the topics covered within "Analyzing Data" in R, according to the textbook:

- **Summary Statistics**: This section highlights R's capabilities for calculating descriptive statistics.
    - It covers basic functions like **`mean()`**, **`min()`**, and **`max()`** to calculate the average, minimum, and maximum values of a vector, respectively.
    - The importance of the **`na.rm`** argument in these functions for handling missing (`NA`) values is explained; by default, the presence of any `NA` value will result in an `NA` return unless `na.rm=TRUE` is specified.
    - The **`trim`** argument in the `mean()` function allows for the removal of a specified fraction of observations from both ends of the sorted vector, useful for mitigating the impact of outliers.
    - The **`range()`** function is presented as a way to simultaneously obtain both the minimum and maximum values.
    - Other crucial summary statistics functions like **`median()`**, **`quantile()`** (for calculating percentiles), **`IQR()`** (for the interquartile range), **`var()`** (for variance), and **`sd()`** (for standard deviation) are also discussed.
    - The **`summary()`** function is emphasized as a versatile generic function that provides a comprehensive overview of the data depending on its type. For numeric data, it typically shows the minimum, first quartile, median, mean, third quartile, and maximum. For factors, it displays the counts of each level. However, it might not provide meaningful information for character values.
    - As an alternative to `summary()`, the **`str()`** function is introduced for displaying the structure of an R object.

- **Correlation and Covariance**: This section focuses on assessing the linear relationships between numeric variables.
    - **Correlation** is measured using the `cor()` function, which can calculate Pearson, Spearman, or Kendall correlation coefficients based on the `method` argument .
    - **Covariance**, which measures the joint variability of two numeric variables, is calculated using the `cov()` function .
- **Principal Components Analysis (PCA)**: PCA is presented as a technique for reducing the dimensionality of a dataset by identifying principal components, which are uncorrelated linear combinations of the original variables that capture the most variance.
    - The `princomp()` function from the `stats` package is used to perform PCA. It can take a formula (e.g., `~ variable1 + variable2`) or a matrix of numeric data as input.
    - Key arguments like `data`, `subset`, `na.action`, `center` (to mean-center the data), and `scale` (to scale variables to have unit variance) are explained.
    - The output of `princomp()` provides information about the principal components, their variances, and the loadings of the original variables on these components.
- **Factor Analysis**: This technique is introduced as a way to model underlying (latent) factors that might be driving the observed correlations between a set of variables.
    - The `factanal()` function in the `stats` package is used for conducting factor analysis.
    - Important arguments include `x` (the data matrix or data frame), `factors` (the number of factors to extract), `rotation` (to aid in interpretation), and `scores` (to obtain factor scores for each observation).
- **Bootstrap Resampling**: The bootstrap is described as a powerful resampling technique used to estimate the sampling distribution of a statistic by repeatedly drawing random samples with replacement from the original dataset.
    - The `boot()` function from the `boot` package is used to perform bootstrap resampling.
    - The crucial arguments are `data` (the original data), `statistic` (a function that calculates the statistic of interest from a resampled dataset), and `R` (the number of bootstrap replicates).
    - The output of `boot()` provides estimates of the bias and standard error of the statistic.

In summary, Chapter 16 of "R in a Nutshell" provides a foundational understanding of various essential techniques for analyzing data in R, ranging from basic descriptive statistics to more advanced methods like PCA, factor analysis, and bootstrapping. These methods are crucial for gaining initial insights into datasets and preparing them for further statistical exploration .

Summary statistics [Note : It is a part of Analyzing Data /* Important*/]

According to the "AIT 362 Programming in R Syllabus", **"Summary statistics" is explicitly listed as a topic within Module 3, "Statistics with R"**. Course Outcome 3 also mentions the ability to "Explain summary function in R".

The "R in a Nutshell 2nd edition" provides a dedicated section on **"Summary Statistics" within Chapter 16, "Analyzing Data," which is part of Part V, "Statistics with R"**. This chapter states that R includes a variety of functions for calculating these statistics.

- **Basic Functions**: R offers several fundamental functions to calculate individual summary statistics.
    - **`mean()`**: This function calculates the arithmetic mean of a vector. It has an important argument, **`na.rm`**, which specifies how missing (`NA`) values should be treated. By default, if any value is `NA`, the result will be `NA`. Setting `na.rm=TRUE` will ignore the `NA` values in the calculation. The `mean()` function also has a **`trim`** argument to remove a specified fraction of observations from both ends of the sorted vector, which can be useful for mitigating the impact of outliers.
    - **`min()`**: This function returns the minimum value in a vector. Similar to `mean()`, it also has the `na.rm` argument for handling missing values.
    - **`max()`**: This function returns the maximum value in a vector. It also includes the `na.rm` argument.
    - **`range()`**: This function returns a vector containing both the minimum and maximum values of a given vector.
    - **`median()`**: This function calculates the median (the middle value) of a vector. It also has the `na.rm` argument.
    - **`quantile()`**: This function calculates the specified quantiles (percentiles) of a vector. You can specify which quantiles you want (e.g., the 25th, 50th, and 75th percentiles). It also has the `na.rm` argument.
    - **`IQR()`**: This function calculates the interquartile range, which is the difference between the 75th and 25th percentiles, providing a measure of statistical dispersion.
    - **`var()`**: This function calculates the sample variance of a vector, which measures how spread out the data is around the mean. It has the `na.rm` argument.
    - **`sd()`**: This function calculates the standard deviation, which is the square root of the variance and another common measure of data dispersion. It also has the `na.rm` argument.
- **Aggregation and Application**: These individual summary statistic functions can be used with other R functions like `apply()`, `tapply()`, or `aggregate()` to calculate statistics for entire

data frames or for subsets of data within a data frame. The `aggregate()` function is specifically mentioned as another option for summarization.

- **The `summary()` Function**: The **`summary()`** function is highlighted as the most convenient function for getting a comprehensive overview of data. It is a **generic function**, meaning its output depends on the type of object it is applied to.
    - For **numeric values**, `summary()` typically displays the minimum, first quartile, median, mean, third quartile, and maximum values.
    - For **factors** (categorical variables), `summary()` shows the count of each of the most frequent values. Less frequent values are often grouped into an "Other" category.
    - The `summary()` function **does not provide meaningful information for character values**.
    - The syllabus also mentions "Explain aggregate function in R", and the "R in a Nutshell" book details the `aggregate` function in "Summarizing Functions" in Chapter 12.
- **The `str()` Function as an Alternative**: The `str()` function is presented as a popular alternative to `summary()`. It displays the **structure of an R object**, providing information about its type and content, which can be helpful for understanding the data.

In essence, R provides a rich set of tools for calculating various summary statistics, allowing users to quickly understand the central tendency and dispersion of their data. The `summary()` function offers a quick and informative overview for different data types, while individual functions provide more specific calculations that can be applied flexibly across datasets and subsets.

Statistical Tests- Continuous Data, Discrete Data

"Statistical Tests- Continuous Data, Discrete Data" is a key topic covered in **Module 3, "Statistics with R,"** according to the "AIT 362 Programming in R Syllabus". The syllabus also lists **"Power tests"** as part of the same module.

The "R in a Nutshell 2nd edition" provides a detailed explanation of statistical tests in **Chapter 18, titled "Statistical Tests"**. This chapter is divided into sections focusing on tests for **"Continuous Data"** and **"Discrete Data"**.

Statistical Tests Overview:

- Many data problems involve statistical tests to answer questions such as the efficacy of a new drug or the impact of a website design on sales.
- To answer these questions, one formulates a hypothesis, designs an experiment, collects data, and uses R to analyze the data.

- Chapter 18 focuses on the tools available in R for answering these questions, divided into tools for continuous and categorical (discrete) random variables.
- It's important to remember that while the book describes how to find these methods in R and when to use them, it doesn't provide the statistical theory behind them. A good statistics course or book is recommended for a deeper understanding.

Continuous Data:

- This section covers tests applicable to continuous random variables, such as times, dollar amounts, and chemical concentrations.
- **Normal Distribution-Based Tests**: These tests assume the underlying data is normally distributed.
    - **Comparing means**: The `t.test()` function is used to test if the mean of a sample is different from a hypothesized mean (null hypothesis) or to compare the means of two groups. Arguments include `alternative` (to specify one- or two-sided tests), `mu` (the null hypothesis mean), `paired` (for paired data), and `var.equal` (to assume equal variances for two-sample tests). An example compares the mean time to failure of tires of type H to a hypothetical mean. Another example compares the mean yards for field goals kicked with and without wind. Paired t-tests can be performed by setting `paired=TRUE`, as demonstrated with computer performance data.
    - **Comparing variances**: The `var.test()` function can be used to compare the variances of two populations. The `bartlett.test()` is used to test if the variances in multiple groups are the same, as shown with the field goal data. The `fligner.test()` provides a non-parametric alternative for testing homogeneity of variances. The `ansari.test()` and `mood.test()` are for testing differences in scale parameters.
    - **Comparing means across more than two groups (ANOVA)**: Analysis of Variance (ANOVA) is used to compare means across multiple groups. The `aov()` function is a simple way to perform these tests, often used with a formula. An example looks at differences in age at death by cause of death using `aov()` and `anova()`. The `oneway.test()` can be used when the assumption of equal variance across groups in ANOVA is violated.
    - **Pairwise t-tests between multiple groups**: The `pairwise.t.test()` function performs t-tests between every pair of groups with corrections for multiple testing. An example compares the time to failure for different tire types.
    - **Correlation tests**: The `cor.test()` function calculates the correlation between two vectors (Pearson, Spearman, or Kendall) and performs a test of significance. Examples show the correlation between highly and less correlated vectors and between airborne toxins and lung cancer deaths.
    - **Testing for normality**: The `shapiro.test()` function performs the Shapiro-Wilk test for normality.

    - **Testing if a data vector came from an arbitrary distribution**: The `ks.test()` (Kolmogorov-Smirnov test) can be used to see if a vector comes from a specified probability distribution.
    - **Testing if two data vectors came from the same distribution**: The `ks.test()` can also compare two data vectors to see if they likely come from the same distribution.
- **Non-Parametric Tests**: These tests are useful when the data is not normally distributed or the distribution is unknown.
    - **Comparing two means**: The `wilcox.test()` (Wilcoxon rank-sum test or Mann-Whitney U test) compares two independent samples or a paired sample without assuming normality. Examples compare time to failure for different tire types and the difference in yards for field goals.
    - **Comparing more than two means**: The `kruskal.test()` (Kruskal-Wallis rank-sum test) is a non-parametric alternative to one-way ANOVA for comparing means across more than two groups.
    - **Comparing variances**: The `fligner.test()` (Fligner-Killeen test) can be used to test for homogeneity of variances without assuming normality. The `ansari.test()` and `mood.test()` are also mentioned for testing differences in scale parameters.

Discrete Data:

- This section covers tests for discrete random variables, such as counts or proportions.
- **Proportion Tests**: The `prop.test()` function is used to test if the proportions of success in several groups are the same or equal to certain given values. An example tests if David Ortiz was a "true" .300 hitter based on his season's batting average.
- **Binomial Tests** : The `binom.test()` performs an exact test of a simple null hypothesis about the probability of success in a Bernoulli experiment. The syllabus also mentions "Binomial distributions".
- **Tabular Data Tests**: These tests examine the relationship between two categorical variables in a contingency table, testing the null hypothesis that the variables are independent.
    - **Fisher's exact test**: The `fisher.test()` calculates the exact probability of the observed contingency table (or a more extreme one) under the assumption of independence, especially useful for small tables. An example examines the independence of delivery method and sex.
    - **Chi-squared tests**: The `chisq.test()` calculates chi-squared contingency table tests and goodness-of-fit tests. It can be used to test the independence of categorical variables. An example also looks at the relationship between delivery method and sex using `chisq.test()`.
    - **McNemar's chi-squared test**: The `mcnemar.test()` tests for symmetry in a two-dimensional contingency table.
    - **Mantel-Haenszel test**: The `mantelhaen.test()` tests for an association between two categorical variables, controlling for one or more other stratifying variables.
- **Non-Parametric Tabular Data Tests**:

- **Friedman rank-sum test**: The `friedman.test()` is a non-parametric counterpart to two-way ANOVA for unreplicated blocked data.
- **Cochran-Mantel-Haenszel test** : The syllabus mentions this test, and the `mantelhaen.test()` function in R can perform this.
- **Quade test**: The `quade.test()` performs a Quade test with unreplicated blocked data.

In summary, R provides a comprehensive suite of functions for performing a wide array of statistical tests for both continuous and discrete data. The choice of test depends on the type of data, the research question, and the assumptions that can be made about the underlying distributions. The syllabus confirms that these topics are fundamental to the "Statistics with R" module.

Power Tests

Power tests in R are used in the context of **experimental design** to help determine the amount of data needed to collect to obtain statistically significant results, or to find the maximum significance of results that can be calculated from a given amount of data. R provides a set of functions to assist in these calculations.

According to Chapter 19, "Power Tests," in "R in a Nutshell 2nd edition":

- Power tests are helpful when designing an experiment to understand how much data is required for a statistically significant sample. They can also help determine the maximum significance of results possible with a specific amount of data.

- **`power.t.test()`**: This function is used when you plan to use a t-test to check the significance of the results, typically in experiments comparing the mean value of a random variable for a "test" population and a "control" population.

    - The function can calculate any one of the following parameters if the other four are provided:
        - `n`: The number of observations per group.
        - `delta`: The true difference in means between the groups.
        - `sd`: The true standard deviation of the underlying distribution (defaults to 1).
        - `sig.level`: The significance level (Type I error probability, defaults to 0.05).
        - `power`: The power of the test (1 - Type II error probability).
        - `type`: Specifies whether the test is "two.sample", "one.sample", or "paired" (defaults to "two.sample").
        - `alternative`: Specifies whether the test is "two.sided" or "one.sided" (defaults to "two.sided").

        - `strict`: Specifies whether to use a strict interpretation in the two-sided case (defaults to `FALSE`).
    - An example shows how to use `power.t.test()` to calculate the minimum statistically significant difference (`delta`) in Hamilton Rating Scale for Depression (HAMD) scores needed in a study with two groups of 25 subjects each, given a power of 0.95 and a standard deviation of 8.9, at a significance level of 0.05.
- **`power.prop.test()`**: If your experiment involves measuring a proportion and you will be using `prop.test`, you can use this function.

    - Similar to `power.t.test()`, this function can calculate one of the following parameters if the others are given:
        - `n`: The number of observations per group.
        - `p1`: The probability of success in one group.
        - `p2`: The probability of success in the other group.
        - `sig.level`: The significance level (Type I error probability, defaults to 0.05).
        - `power`: The power of the test (1 - Type II error probability).
        - `alternative`: Specifies whether the test is "two.sided" or "one.sided" (defaults to "two.sided").
        - `strict`: Specifies whether to use a strict interpretation in the two-sided case (defaults to `FALSE`).
    - Examples illustrate using `power.prop.test()` to find the power of a test with a sample size of 10 per group and probabilities of success of 0.26 and 0.30 in the two groups, with a significance level of 0.05 and a one-sided alternative.
- **`power.anova.test()`**: This function is mentioned in the index and within the discussion of ANOVA test design. It is used for power calculations in the context of Analysis of Variance (ANOVA) tests, which compare means across more than two groups .

In conclusion, power tests in R are essential for **planning experiments** by allowing you to estimate the required sample size or the expected power of your statistical tests. The functions `power.t.test()`, `power.prop.test()`, and `power.anova.test()` are key tools in R for performing these calculations for t-tests, proportion tests, and ANOVA, respectively.

Common distributions- type arguments [Note : Read Probability Distributions also]

Module 3 of the "AIT 362 Programming in R Syllabus" mentions "**Common distributions- type arguments**" as a topic. This refers to the consistent way that R handles arguments for different types of functions related to probability distributions. These function types include probability density functions (or probability mass functions for discrete distributions), distribution functions, quantile functions, and random number generators.

According to "R in a Nutshell 2nd edition", almost all R functions that generate values for probability distributions follow a similar naming convention:

- **Probability Density Functions (PDFs)** or Probability Mass Functions (PMFs) for discrete distributions begin with "**d**". Examples include `dnorm` for the normal distribution, `dbinom` for the binomial distribution, and `dpois` for the Poisson distribution.
- **Distribution Functions** begin with "**p**". These functions give the cumulative probability of a random variable being less than or equal to a certain value. Examples include `pnorm` for the normal distribution, `pbinom` for the binomial distribution, and `ppois` for the Poisson distribution. The `lower.tail` argument (defaulting to `TRUE`) allows you to choose between $P(X \le q)$ and $P(X > q)$. The `log.p` argument allows you to get the logarithm of the probability.
- **Quantile Functions** begin with "**q**". These are the inverse of the distribution functions. They return the value q such that $P(X \le q) = p$ (by default). Examples include `qnorm` for the normal distribution, `qbinom` for the binomial distribution, and `qpois` for the Poisson distribution. They also have `lower.tail` and `log.p` arguments similar to the distribution functions.
- **Random Number Generators** begin with "**r**". These functions generate random values from the specified distribution. They typically take an argument "**n**" which specifies the number of random values to generate. An exception is the hypergeometric distribution, where the number of random values to generate is renamed to "**nn**". Examples include `rnorm` for the normal distribution, `rbinom` for the binomial distribution, and `rpois` for the Poisson distribution.

Furthermore, most of these types of functions share certain common arguments:

- For **density functions** (`d` functions): they often have an argument **"x"** which specifies the value(s) at which to evaluate the density, and a **"log"** argument (defaulting to `FALSE`) to return either the raw density or its logarithm.
- For **distribution functions** (`p` functions): they typically have an argument **"q"** which specifies the quantile(s) at which to evaluate the distribution, "**lower.tail**" (defaulting to `TRUE` for $P(X \le q)$), and "**log.p**" (defaulting to `FALSE` for the raw probability).

- For **quantile functions** (`q` functions): they generally have an argument "**p**" which specifies the probability (or probabilities), "**lower.tail**" (defaulting to `TRUE` for $P(X \le q)$), and "**log.p**" (defaulting to `FALSE` for the raw probability).
- For **random number generators** (`r` functions): they usually have an argument "**n**" to specify the number of random values to generate (with the exception of `rhyper` where it is "**nn**").

In addition to these type-specific arguments, each **distribution family** (like normal, binomial, gamma, etc.) has its own set of **family arguments** that specify the parameters of that particular distribution. For example, for the normal distribution, these are "**mean**" and "**sd**" (standard deviation). For the binomial distribution, they are "**size**" (number of trials) and "**prob**" (probability of success in each trial).

Understanding these common "type arguments" and the naming conventions makes it easier to work with different probability distributions in R, as you can often predict the names and purposes of the key arguments for a new distribution you encounter. You can always use R's help system (e.g., `?dnorm`) to get specific details about the arguments for any particular distribution function.

Probability distributions

R provides a comprehensive set of tools for working with **probability distributions**. These tools allow you to calculate densities, distribution functions, quantile functions, and to generate random values for a wide range of common statistical distributions. Module 3 of your syllabus specifically mentions "Probability distributions" and "Normal distributions," indicating their importance in the course. Course Outcome 3 also highlights the ability to "Illustrate the use of Probability distributions and basic statistical functions" in R.

Key Concepts and Function Types:

R employs a consistent naming convention for functions associated with probability distributions, making it easier to remember and use them. For almost every probability distribution, you will find functions with the following prefixes:

- **`d`** for **probability density functions (PDFs)** (for continuous distributions) or **probability mass functions (PMFs)** (for discrete distributions). These functions calculate the probability density or mass at a specific value. For example, `dnorm()` for the normal distribution gives the density at a given point.
- **`p`** for **distribution functions**. These functions return the **cumulative probability** of a random variable being less than or equal to a given value q, i.e., $P(X \le q)$ by default. For instance, `pnorm()` gives the cumulative probability for the normal distribution. They often have

arguments like `lower.tail` to compute $P(X > q)$ if set to `FALSE`, and `log.p` to return the logarithm of the probability.

- **`q`** for **quantile functions**. These are the inverse of the distribution functions. Given a probability p, the quantile function returns the value q such that $P(X \le q) = p$ (by default). For example, `qnorm()` finds the value below which a certain proportion of the normal distribution lies. Similar to `p` functions, they also have `lower.tail` and `log.p` arguments.
- **`r`** for **random number generators**. These functions generate random values from the specified distribution. They typically take an argument `n` specifying the number of random values to generate. For example, `rnorm(n)` produces `n` random numbers from a normal distribution. Note that for the hypergeometric distribution, the number of random values is specified by `nn`.

Common Arguments:

Besides the distribution-specific parameters, these four types of functions often share common arguments:

- For **density functions** (`d` functions):
  - `x`: The value(s) at which to evaluate the density/mass.
  - `log`: A logical value indicating whether to return the logarithm of the density/mass (default is `FALSE`).
- For **distribution functions** (`p` functions):
  - `q`: The quantile(s) at which to evaluate the cumulative probability.
  - `lower.tail`: A logical value indicating whether to return $P(X \le q)$ (default is `TRUE`) or $P(X > q)$.
  - `log.p`: A logical value indicating whether to return $\log(P(X \le q))$ (or $\log(P(X > q))$ if `lower.tail` is `FALSE`) (default is `FALSE`).
- For **quantile functions** (`q` functions):
  - `p`: The probability(ies).
  - `lower.tail`: A logical value indicating whether `p` refers to $P(X \le q)$ (default is `TRUE`) or $P(X > q$).
  - `log.p`: A logical value indicating whether the input `p` is given as a logarithm (default is `FALSE`).
- For **random number generators** (`r` functions):
  - `n`: The number of random values to generate (except for `rhyper`, where it is `nn`).

Distribution Families and Their Arguments:

R's `stats` package includes a wide array of probability distribution families. Each family has its own set of **family arguments** that define the parameters of that specific distribution. For example:

- **Normal distribution:** Functions are `dnorm`, `pnorm`, `qnorm`, `rnorm`. Family arguments include `mean` (default 0) and `sd` (standard deviation, default 1).

- **Binomial distribution:** Functions are `dbinom`, `pbinom`, `qbinom`, `rbinom`. Family arguments are `size` (number of trials) and `prob` (probability of success on each trial). Question 16a of your exam requires generating a probability distribution table for this distribution.
- **Poisson distribution:** Functions are `dpois`, `ppois`, `qpois`, `rpois`. The main family argument is `lambda` (rate parameter). Question 16b involves fitting a Poisson distribution.
- Other distributions include beta (`dbeta`, `pbeta`, `qbeta`, `rbeta`), Cauchy (`dcauchy`, `pcauchy`, `qcauchy`, `rcauchy`), chi-squared (`dchisq`, `pchisq`, `qchisq`, `rchisq`), exponential (`dexp`, `pexp`, `qexp`, `rexp`), F (`df`, `pf`, `qf`, `rf`), gamma (`dgamma`, `pgamma`, `qgamma`, `rgamma`), geometric (`dgeom`, `pgeom`, `qgeom`, `rgeom`), hypergeometric (`dhyper`, `phyper`, `qhyper`, `rhyper` with argument `nn`), log-normal (`dlnorm`, `plnorm`, `qlnorm`, `rlnorm`), logistic (`dlogis`, `plogis`, `qlogis`, `rlogis`), negative binomial (`dnbinom`, `pnbinom`, `qnbinom`, `rnbinom`), uniform (`dunif`, `punif`, `qunif`, `runif`), and Weibull (`dweibull`, `pweibull`, `qweibull`, `rweibull`). The specific family arguments for each can be found in the respective function's help documentation (e.g., `?dbeta`).

Visualizing Distributions:

R also provides functions for visualizing probability distributions. For example:

- `hist()` can be used to create histograms to visualize the distribution of a dataset or random samples from a distribution.
- `density()` and `densityplot()` (in the `lattice` package) can be used to estimate and plot the probability density of a dataset.
- `qqnorm()` and `qqplot()` (in the `stats` package) and `qqmath()` (in the `lattice` package) create quantile-quantile plots to compare the distribution of a dataset to a theoretical distribution (often normal) or to another dataset.

Understanding these naming conventions, common arguments, and the availability of various distribution functions is crucial for performing statistical analysis and modeling in R. Chapter 17 of "R in a Nutshell" serves as a valuable resource for further exploration of these concepts. Remember that you can always use R's help system (e.g., `?dnorm`) to get detailed information about the arguments and usage of any specific distribution function.

Normal Distributions **[Note : Important Probability Distribution]**

The "AIT 362 Programming in R Syllabus" explicitly lists "**Normal distributions**" as a topic within **Module 3 (Statistics with R)**. This module also covers "**Common distributions- type arguments**" and "**Probability distributions**" in general, indicating the importance of understanding different distributions, including the normal distribution, and the consistent way R handles them. Course Outcome 3 aims to

enable students to "Illustrate the use of Probability distributions and basic statistical functions", which directly includes working with normal distributions. Sample assessment question 3 under CO3 asks students to "Describe about probability distributions", and questions 16a and 16b in the model question paper involve working with binomial and Poisson distributions, suggesting a broader context for understanding the normal distribution within the framework of probability distributions in R.

The textbook "R in a Nutshell 2nd edition" dedicates **Chapter 17 to "Probability Distributions"**, and it uses the **Normal Distribution** as a primary example to illustrate how R handles various distribution-related tasks.

Key R Functions for the Normal Distribution:

R provides four main functions for working with the normal distribution, following a consistent naming convention:

- **`dnorm(x, mean = 0, sd = 1, log = FALSE)`**: This is the **probability density function (PDF)** for the normal distribution.
    - `x`: Specifies the value or vector of values at which to evaluate the density.
    - `mean`: Specifies the mean of the normal distribution (default is 0).
    - `sd`: Specifies the standard deviation of the normal distribution (default is 1).
    - `log`: A logical value indicating whether to return the raw density (`FALSE`, default) or the logarithm of the density (`TRUE`). The textbook demonstrates plotting the normal distribution using `plot(dnorm, -3, 3, main = "Normal Distribution")`.
- **`pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)`**: This is the **distribution function (cumulative probability function)** for the normal distribution.
    - `q`: Specifies the quantile or vector of quantiles at which to evaluate the cumulative probability $P(X \le q)$.
    - `mean`: Specifies the mean of the normal distribution (default is 0).
    - `sd`: Specifies the standard deviation of the normal distribution (default is 1).
    - `lower.tail`: A logical value. If `TRUE` (default), probabilities are $P(X \le q)$. If `FALSE`, probabilities are $P(X > q)$.
    - `log.p`: A logical value indicating whether to return the raw probability (`FALSE`, default) or the logarithm of the probability (`TRUE`). The textbook provides examples of calculating probabilities using `pnorm`, such as `pnorm(0)` (probability of being less than or equal to the mean in a standard normal distribution) and `pnorm(-1)` (probability of being less than one standard deviation below the mean). It also shows how to plot the cumulative normal distribution with `plot(pnorm, -3, 3, main = "Cumulative Normal Distribution")`.

- **`qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)`**: This is the **quantile function** for the normal distribution, the inverse of `pnorm`.
    - `p`: Specifies the probability or vector of probabilities (between 0 and 1).
    - `mean`: Specifies the mean of the normal distribution (default is 0).
    - `sd`: Specifies the standard deviation of the normal distribution (default is 1).
    - `lower.tail`: A logical value. If `TRUE` (default), `p` corresponds to $P(X \le q)$, and the function returns q. If `FALSE`, `p` corresponds to $P(X > q)$, and the function returns q.
    - `log.p`: A logical value indicating whether the input `p` is the raw probability (`FALSE`, default) or the logarithm of the probability (`TRUE`). Examples in the textbook include finding the median (`qnorm(0.5)`) and the values for a 95% confidence interval (`c(qnorm(.025), qnorm(.975))`). It also demonstrates the inverse relationship with `pnorm` by showing `qnorm(pnorm(-1))` returns -1.
- **`rnorm(n, mean = 0, sd = 1)`**: This is the **random number generator** for the normal distribution.
    - `n`: Specifies the number of random values to generate.
    - `mean`: Specifies the mean of the normal distribution (default is 0).
    - `sd`: Specifies the standard deviation of the normal distribution (default is 1). The textbook shows an example of generating 10,000 random values and visualizing them with a histogram using `hist(rnorm(10000), breaks=50)`.

Common Distribution-Type Arguments:

Chapter 17 highlights that these `dnorm`, `pnorm`, `qnorm`, and `rnorm` functions follow a general pattern for probability distributions in R. They have a prefix indicating the type of function (`d`, `p`, `q`, `r`) and share common arguments like `mean` and `sd` specific to the normal distribution family. Additionally, they include the "**common distribution-type arguments**":

- For density functions (`d`): `x`, `log`
- For distribution functions (`p`): `q`, `lower.tail`, `log.p`
- For quantile functions (`q`): `p`, `lower.tail`, `log.p`
- For random numbers (`r`): `n` (with an exception for the hypergeometric distribution)

This consistency makes it easier to work with other probability distributions in R as well.

Normal Distribution in Statistical Tests:

The normal distribution is fundamental in many statistical tests covered in **Chapter 18 ("Statistical Tests")**. For instance, the **t-test** (implemented by the `t.test` function) often assumes that the underlying

data is normally distributed. Chapter 18 discusses using t-tests to compare means based on this assumption.

Assessing Normality:

R also provides tools to check if data follows a normal distribution. The `qqnorm` function (in the `stats` package) generates a **quantile-quantile plot** to visually compare the distribution of a dataset against a theoretical normal distribution. The `qqline` function can add a reference line to this plot. The `rfs` function in the `lattice` package can also produce quantile-quantile plots of residuals to assess model fit against a specified distribution, including the normal distribution using `distribution=qnorm`. Chapter 18 also mentions `ks.test` for testing if a data vector came from an arbitrary distribution and if two data vectors came from the same distribution, which can be used to assess normality .

Multivariate Normal Distribution:

The `MASS` package includes the `mvrnorm` function which can generate random samples from a **multivariate normal distribution**.

In summary, R provides a comprehensive set of functions for working with normal distributions, covering density, cumulative probability, quantiles, and random number generation. These functions follow a consistent argument structure common to other probability distributions in R. Furthermore, the normal distribution plays a crucial role in various statistical tests, and R offers tools for assessing whether data conforms to a normal distribution. Module 3 of your syllabus highlights the importance of understanding normal distributions within the broader context of statistics with R.

Extra : Basic Statistical Functions in R [A revision on important statistical functions]

- **Summary Statistics**: Chapter 16 discusses functions for calculating summary statistics. These include:
    - **`mean(x, na.rm = FALSE, trim = 0)`**: Calculates the mean of a vector `x`. The `na.rm` argument specifies how `NA` values are treated, and `trim` allows for removing a fraction of observations (outliers).
    - **`min(x, na.rm = FALSE)`**: Finds the minimum value in a vector `x`.
    - **`max(x, na.rm = FALSE)`**: Finds the maximum value in a vector `x`.
    - **`range(x, na.rm = FALSE)`**: Returns a vector containing the minimum and maximum values of `x`.
    - **`median(x, na.rm = FALSE)`**: Computes the sample median.
    - **`sd(x, na.rm = FALSE)`**: Computes the standard deviation of the values in `x`.

    - `quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE, ...)`: Produces sample quantiles corresponding to the given probabilities .
    - `summary(object, ...)`: This is a generic function used to produce summaries of the results of various model fitting. It can provide different summary statistics depending on the type of object passed to it .
- **Functions for Probability Distributions**: Chapter 17 describes functions for working with common statistical distributions, including the normal distribution. For each distribution, there are typically four functions with prefixes `d`, `p`, `q`, and `r` for density, cumulative distribution function, quantile function, and random number generation, respectively. For the normal distribution, these are `dnorm`, `pnorm`, `qnorm`, and `rnorm`. These functions are fundamental for many statistical calculations and tests.

- **Correlation and Covariance**: Chapter 16 also covers functions for calculating these measures of relationship between variables :

    - `cor(x, y = NULL, method = c("pearson", "kendall", "spearman"))`: Computes the correlation between two vectors or the correlation matrix of a matrix or data frame.
    - `cov(x, y = NULL, method = c("pearson", "kendall", "spearman"), use = "everything")`: Computes the covariance between two vectors or the covariance matrix of a matrix or data frame.
- **Basic Statistical Tests**: Chapter 18 introduces statistical tests. Some fundamental tests include:

    - `t.test(x, y = NULL, ...)`: Performs one- and two-sample t-tests.
    - `var.test(x, y, ratio = 1, alternative = c("two.sided", "less", "greater"), ...)`: Performs an F test to compare the variances of two samples from normal populations.
    - `wilcox.test(x, y = NULL, alternative = c("two.sided", "less", "greater"), ...)`: Performs one- and two-sample Wilcoxon tests (also known as Mann-Whitney test).
    - `chisq.test(x, y = NULL, correct = TRUE, p = rep(1/length(x), length(x)), ...)`: Performs chi-squared contingency table tests and goodness-of-fit tests.
- **Linear Regression**: While more advanced, the function for building linear models, `lm(formula, data, ...)`, is a foundational statistical tool in R used for modeling relationships between variables.

These are some of the basic statistical functions available in the `stats` package, which is one of the **base packages** that come with R and is loaded by default.

Other packages like `MASS` and `boot` extend the statistical capabilities of R with functions like `mvrnorm` for generating multivariate normal random variables and `boot` for bootstrap resampling. The `lattice` and `graphics` packages provide functions for visualizing distributions and statistical results.

The `aggregate` function is also useful for computing summary statistics for subsets of data.

This overview provides a starting point for understanding the basic statistical functions available in R, as covered in your syllabus and the "R in a Nutshell" textbook. Remember that R has a vast array of functions for various statistical tasks, and exploring the `stats` package and others will reveal more specialized tools.

Extra : Parametric and Non parametric tests in R [A revision]

R provides a variety of **parametric** and **non-parametric statistical tests**, as detailed in Chapter 18, "Statistical Tests," within the "Statistics with R" section of "R in a Nutshell 2nd edition".

Parametric Tests:

These tests often assume that the underlying data follows a specific distribution, commonly the **normal distribution**. Chapter 18 starts by describing "Normal Distribution-Based Tests". Some examples of parametric tests and the corresponding R functions include:

- **t-tests** (`t.test`): Used for comparing means of one or two samples. The function can perform one-sample, two-sample, and paired t-tests. Examples in the text demonstrate comparing a sample mean to a hypothesized mean, comparing means of two independent groups, and comparing paired data.
- **F-test for comparing variances** (`var.test`): Used to compare the variances of two samples from normal populations.
- **ANOVA (Analysis of Variance)** (`aov`): Used to compare means across more than two groups. The `anova` function can be used to get analysis of variance statistics for linear models created with `lm`.
- **Correlation tests** (`cor.test`): Used to test the correlation between two vectors. The `method` argument allows for Pearson, Kendall, or Spearman correlation coefficients.

Non-Parametric Tests:

These tests do not rely on strong assumptions about the distribution of the data. Chapter 18 also covers "Non-Parametric Tests". Some examples and R functions include:

- **Wilcoxon rank-sum test** (`wilcox.test`): A non-parametric alternative to the two-sample t-test, used for comparing two means without assuming normality. The test works by looking at the ranks of elements. Examples comparing tire failure times are provided.
- **Kruskal-Wallis rank-sum test** (`kruskal.test`): A non-parametric alternative to ANOVA, used for comparing more than two means.
- **Fligner-Killeen test** (`fligner.test`): A non-parametric test for comparing variances among different groups.
- **Ansari-Bradley test** (`ansari.test`) and **Mood's test** (`mood.test`): Used for testing for differences in scale parameters between two samples.
- **Friedman rank-sum test** (`friedman.test`): A non-parametric counterpart to two-way ANOVA tests for tabular data.

The choice between parametric and non-parametric tests depends on the characteristics of the data and the assumptions that can be reasonably made. The textbook suggests that non-parametric tests might be helpful when the distribution of the data is not normally distributed. You can also test for normality using the **Shapiro-Wilk test** (`shapiro.test`) or the **Kolmogorov-Smirnov test** (`ks.test`) to see if a data vector came from an arbitrary distribution.

Furthermore, the syllabus for "AIT 362 Programming in R" mentions "**power tests**", which are related to the design of statistical experiments and are covered in Chapter 19 of "R in a Nutshell". Power tests can be used in the context of both parametric (e.g., `power.t.test`, `power.anova.test`) and non-parametric tests (though the textbook primarily focuses on parametric examples in this chapter, such as `power.prop.test` for proportion tests).

Extra : T Tests in R [A Revision]

T-tests are statistical tests used in R (via the function `t.test`) to answer questions about the **means of data**. They are often used to determine if the mean of a sample is significantly different from a hypothesized value or to compare the means of two different groups.

Here's a more detailed breakdown of t-tests in R:

- **Purpose**:
    - To check the statistical significance of experimental results.
    - To determine if the mean of experimental data is close to a value expected under a **null hypothesis**.
    - To calculate the probability that an **alternative hypothesis** (e.g., the mean is different from the null hypothesis) is true.

- To determine if there is a statistically significant difference between the **means of two groups of observations**.

- **Assumptions**:

  - Many common t-tests assume that the underlying **data is normally distributed**. The sources mention that normal distributions occur frequently in nature, making this assumption often reasonable. You can test for normality in R using functions like `shapiro.test` and `ks.test`.

- **Types of t-tests and the `t.test()` function**:

  - **One-Sample t-test:** This test is used when you have a single group of observations and want to compare its mean to a **hypothesized mean (`mu`)**. You would use `t.test(x, mu=...)`, where `x` is a numeric vector of your data. An example in the text compares the mean time to failure of type H tires to a hypothetical mean of 9 hours.
  - **Two-Sample t-test:** This test is used to compare the **means of two independent groups**. You can provide two numeric vectors as arguments: `t.test(x, y)`. The `var.equal` argument specifies whether to assume equal variances in the two groups. The text provides examples comparing the failure times of different tire types (e.g., E and D).
    - You can also use a **formula** interface for two-sample t-tests when your data is in a data frame, using `t.test(response_variable ~ grouping_factor, data=your_data)`. An example compares field goal distances based on whether the stadium was indoor or outdoor.
  - **Paired t-test:** This test is used when you have **paired data**, such as measurements taken on the same subjects before and after an intervention. You perform this by setting the argument `paired=TRUE` in the `t.test` function: `t.test(x, y, paired=TRUE)`. The text gives an example comparing baseline and result values for computer performance.

- **Key Arguments of `t.test()`**:

  - `x`: A **numeric vector** containing your data values.
  - `y`: An optional second **numeric vector** for two-sample and paired tests. Use `y = NULL` for a one-sample test.
  - `alternative`: A character string specifying the **alternative hypothesis**. Options include `"two.sided"` (mean is different), `"less"` (mean is less than), and `"greater"` (mean is greater than). The default is `"two.sided"`.
  - `mu`: A numeric value specifying the **null hypothesis mean** (for a one-sample test) or the hypothesized difference in means (for a two-sample test). The default is `0`.
  - `paired`: A **logical value** indicating whether the two samples are paired. Defaults to `FALSE`.

    - `var.equal`: A **logical value** indicating whether to assume equal variances in a two-sample t-test. Defaults to `FALSE` (Welch method is used). If `TRUE`, the pooled variance is used.
    - `conf.level`: The **confidence level** for the confidence interval of the mean (or difference in means). The default is `0.95`.
- **Output of `t.test()`**:
    - The **test statistic** (the calculated t-value).
    - The **degrees of freedom** (`df`).
    - The **p-value**: This is the probability of observing data as extreme as, or more extreme than, your data if the null hypothesis were true. A small p-value (typically less than your chosen significance level, e.g., 0.05) suggests that you can reject the null hypothesis.
    - A statement of the **alternative hypothesis**.
    - The **confidence interval** for the mean (or the difference in means).
    - The **sample estimates** (the calculated mean(s) of your data).
- **Non-Parametric Alternative**:
    - The **Wilcoxon rank-sum test** (`wilcox.test`) is a non-parametric test that can be used as an alternative to the two-sample t-test when the assumption of normality is not met. It works by looking at the ranks of the data rather than the actual values.
- **Power of t-tests**:
    - The `power.t.test` function in R can be used in **experimental design** to determine things like the required sample size (`n`), the detectable difference in means (`delta`), or the power of the test, given other parameters like the significance level (`sig.level`) and standard deviation (`sd`).

In summary, `t.test` is a fundamental function in R for performing various types of t-tests to compare means of datasets, with options to specify the nature of the comparison (one sample, two independent samples, or paired samples) and the assumptions about the data. The output provides key statistics to assess the significance of the observed differences.

Extra : ANOVA (Analysis of Variance) [A revision]

ANOVA (Analysis of Variance) is a method used in R to **compare the means across more than two groups**. The sources explain that a full explanation of ANOVA requires understanding statistical models in R, which are covered in Chapter 20 of "R in a Nutshell 2nd edition".

Here's how ANOVA is performed in R using the `aov` function:

- The basic function for performing ANOVA is `aov()`.
- It takes a **formula** as its primary argument, which specifies the relationship between the response variable and the grouping factor(s).
- The general form is `aov(formula, data = NULL, ...)`. The `data` argument specifies the data frame containing the variables.

An example provided in the text looks at differences in age at death by cause of death using the 2006 U.S. mortality data set. The `aov` function is used like this:

aov(age~Cause, data=mort06.smpl)

Here, `age` is the response variable and `Cause` is the factor with different levels representing the groups being compared.

The output of the `aov` function provides a summary of the model, including:

- The call to the function.
- Sum of Squares, Degrees of Freedom, and Mean Squares for the factor and residuals.
- Residual standard error.

To get more detailed information about the ANOVA results, you can use the `model.tables` function on the `aov` object:

model.tables(aov(age~Cause, data=mort06.smpl))

This will print tables of effects for each level of the factor.

The sources also mention that often it is better to use the `lm` function to fit a **linear model** and then use the `anova` function to extract information about the analysis of variance. The `anova` function presents results slightly differently than the `aov` function. For example:

mort06.smpl.lm <- lm(age~Cause, data=mort06.smpl)

anova(mort06.smpl.lm)

This will produce an Analysis of Variance Table, including F-values and p-values for each term in the model.

It's important to note that **ANOVA calculations assume that the variance is equal across groups**. If this assumption is not met (or suspected to be untrue), you can use the `oneway.test` function as an alternative to calculate whether two or more samples have the same mean. This function is similar to calling `t.test` with `var.equal=FALSE` (using the Welch method).

There are other functions for printing information about `aov` objects, such as:

- `proj`: Returns a list of matrices giving the projections of data onto the linear model.
- `TukeyHSD`: Returns confidence intervals on the differences between the means of the levels of a factor with a specified family-wise probability of coverage.
- `se.contrast`: Returns the standard errors for one or more contrasts in an `aov` object.

Finally, the **Kruskal-Wallis rank-sum test** (`kruskal.test`) is mentioned as a **non-parametric equivalent to ANOVA** for comparing more than two means. An example is provided using the same mortality data.

Note : "Two-Way ANOVA"

The sources mention the Friedman rank-sum test as a non-parametric counterpart to two-way ANOVA for tabular data. This suggests that "two-way ANOVA" likely involves analyzing the effect of two categorical independent variables on a continuous dependent variable. The term "tabular data" in the description of the Friedman test implies that the data can be organized in a table where rows and columns represent the levels of these two independent variables.

Extra : **Some important statistical tests available in R** [A Revision]

Test Name	Purpose	Data Type	Key Assumptions	R Function(s)
t-tests	Compare means of one or two samples. Determine if a sample mean differs from a hypothesized mean.	Continuous	Often assumes normal distribution. Equal variances (for some types).	`t.test`

F-test for comparing variances	Compare the variances of two samples.	Continuous	Assumes normal populations.	`var.test`
ANOVA (Analysis of Variance)	Compare means across more than two groups.	Continuous	Assumes equal variance across groups.	`aov`, `anova` (with `lm`)
Correlation tests	Test the correlation between two vectors.	Continuous	Pearson assumes linear relationship.	`cor.test`
Wilcoxon rank-sum test	Non-parametric alternative to the two-sample t-test for comparing two means.	Continuous	Does not assume normality. Looks at ranks.	`wilcox.test`
Kruskal-Wallis rank-sum test	Non-parametric alternative to ANOVA for comparing more than two means.	Continuous	Does not assume normality. Based on ranks.	`kruskal.test`
Fligner-Killeen test	Non-parametric test for comparing variances among different groups.	Continuous	Does not assume normality. Based on medians.	`fligner.test`
Ansari-Bradley test	Test for differences in scale parameters between two samples.	Continuous		`ansari.test`

Mood's test	Test for differences in scale parameters between two samples.	Continuous		`mood.test`
Friedman rank-sum test	Non-parametric counterpart to two-way ANOVA for tabular data.	Continuous/Ranked	Used with blocked data.	`friedman.test`
Proportion tests	Test if proportions of success in several groups are the same.	Discrete (counts)	Can use normal approximation (large samples).	`prop.test`
Binomial tests	Test a hypothesis about the probability of success in a Bernoulli experiment.	Discrete (counts)	Assumes independent Bernoulli trials.	`binom.test`
Chi-squared tests	Test for association between categorical variables in contingency tables. Test goodness-of-fit.	Discrete (counts)	Assumes expected frequencies are not too small.	`chisq.test`
Fisher's exact test	Exact test for independence in small contingency tables.	Discrete (counts)	No specific assumptions about large samples.	`fisher.test`
McNemar's test	Test for symmetry in a two-dimensional contingency table (paired data).	Discrete (counts)	Used for paired nominal data.	`mcnemar.test`

This table highlights some of the key parametric and non-parametric tests discussed in Chapter 18 of "R in a Nutshell". The choice of which test to use depends on the type of data you have and the assumptions you can reasonably make about its distribution. The syllabus for "AIT 362 Programming in R" also mentions **power tests**, which are used to determine the required sample size or the power of a test but are not tests of hypotheses on data themselves. R provides functions like `power.t.test`, `power.prop.test`, and `power.anova.test` for power calculations related to these parametric tests.
