محاسبات آماری و ریاضی با نرم افزار R

مؤلفین:

امید چترآبگون

محسن اسماعیل بیگی

مهدی راسخی

دانشگاه ملایر

مقدمه مؤلفین:

حمد و سپاس مخصوص خداوندی است که به انسان قدرت اندیشیدن عطا فرمود. خداوندی که اندیشه را سبب رجحان انسان بر سایر مخلوقات قرار داد.

تجزیه و تحلیل داده‌های حاصل از نتایج آزمایش‌ها و داده‌های خام همواره یکی از دغدغه‌های استفاده کنندگان در خیلی از زمینه‌ها و رشته‌های علمی می‌باشد. در خیلی از رشته‌ها کاربران آن از جمله اقتصاد، روانشناسی، کشاورزی، زیست شناسی و رشته‌های پزشکی و پیراپزشکی و ... نیازمند این تجزیه و تحلیل می‌باشند. یکی از نرم افزارهای رایج در این زمینه نرم افزار SPSS است که مورد علاقه بسیاری از کاربران می‌باشد اما متاسفانه این نرم افزار قابلیت برنامه نویسی و استفاده از مطالب جدید را کمتر دارا می‌باشد. به همین دلیل ما بر آن شدیم که در این کتاب و مجموعه به معرفی نرم افزاری بپردازیم که علاوه بر دارا بودن قابلیت‌های نرم افزار SPSS قابلیت و توان برنامه نویسی و استفاده بهینه از مطالب جدید و بروز را داشته باشد. نرم افزار R با داشتن این ویژگی‌های منحصر بفرد قابلیت استفاده از مطالب بروز را به صورت آنلاین و آفلاین فراهم می‌کند. این نرم افزار، یک نرم افزار رایگان است که براحتی قابل دانلود می باشد و حجم آن نیز بسیار کم است. با توجه به مزیت‌های بسیار خوب این نرم افزار در تجزیه و تحلیل داده‌ها همواره فقدان یک مجموعه به عنوان راهنما و کمکی برای این نرم افزار به چشم می آمد لذا برآن شدیم تا با خلق این اثر هم به معرفی این نرم افزار پرداخته و هم اینکه آن را به عنوان یک مجموعه کامل و بی نقص به دانشجویان کارشناسی و تحصیلات تکمیلی در رشته ها و زمینه‌های مختلف معرفی کنیم. در پایان لازم می‌دانیم که از زحمات اساتید گرامی جناب آقای دکتر علیرضا دانش خواه عضو هیئت دانشگاه کرانفیلد انگلستان، جناب آقایان دکتر غلامعلی پرهام و دکتر بهزاد منصوری اعضای هیئت علمی دانشگاه شهید چمران اهواز که اینجانبان را در تهیه و تنظیم این کتاب راهنمایی نمودند کمال قدردانی و سپاس گزاری را به عمل آوریم.

مؤلفین

مرداد ماه ۱۳۹۴

تقدیم به تمام کسانی که

علم و روشنی را سرلوحه و چراغ راه خویش قرار داده اند

نه ثروت نه سیاست نه پست و مقام و نه هیچ چیز دیگر!

فهرست مطالب

فصل اول

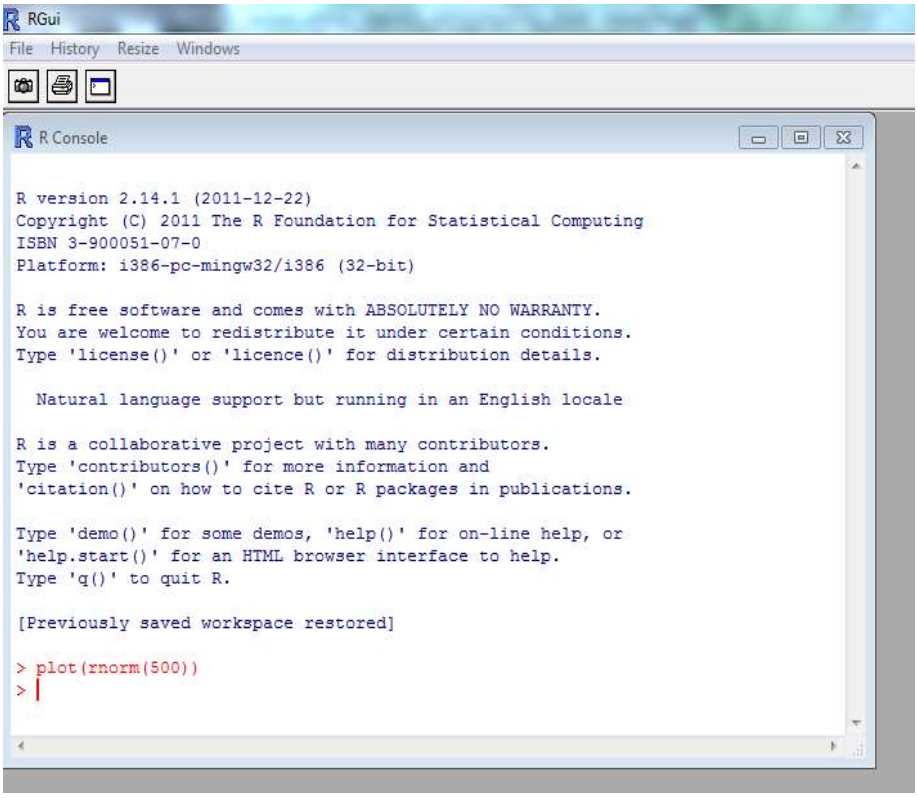
کلیات

۱-۱ گام نخست

نرم افزار R را به عنوان یک نرم افزار رایگان آماری، می‌توان از طریق مراجعه به وبسایت شبکه جامع بایگانی R به نشانی

http://cran.r-project.org

دریافت و بر روی کامپیوتر خود نصب کرد. پس از طی مراحل نصب این نرم افزار را می‌توان از طریق شمایل موجود بر روی میزکار رایانه یا از طریق منوی Start ویندوز خود اجرا کنید.پس از اجرای این برنامه تصویری مانند شکل زیر را مشاهده خواهید کرد.



نرم افزار R به صورت یک محیط نرم افزاری پرسش-پاسخ عمل می‌کند، در

محیط این برنامه و در پنجره‌ای که در تصویر فوق با عنوان **میزفرمان**[1] مشاهده می‌کنید، برنامه آمادگی دریافت پرسش شما را دارد، حال در این پنجره و در مقابل علامت > شما باید فرمان خود را تایپ کرده و با زدن دکمه‌ی Enter منتظر دریافت پاسخ و یا خروجی مورد نظرخود باشید. در صورتی که خواسته‌ی شما صحیح وارد شده باشد، در زیر همان خط فرمان، شما پاسخ درخواست خود را مشاهده خواهید کرد. (شکل زیر)

```
> (4+5)*2
[1] 18
>
```

در محیط نرم افزار R درصورتی که بخواهیم به طور همزمان و در یک خط دستور، چند پرسش را مطرح کنیم می‌توانیم از نماد "؛" استفاده کنیم

```
> 2+5; sin(pi/4); abs(-4)
[1] 7
[1] 0.7071068
[1] 4
>
```

در محیط این نرم افزار می‌توان دستورات نوشته شده و نتایج موجود در میزفرمان را پاک کرد؛ برای پاکسازی میزفرمان می‌توان از کلید ترکیبی Ctrl+L یا زیرمنوی Clear Console در منوی Edit استفاده کرد.

در صورتی‌که در هنگام نوشتن دستور و برنامه و زدن دکمه Enter ، دستور مورد نظر کامل نباشد، مثلا یک پرانتز باز شده را نبسته باشیم، یکی از آرگومان‌های تابع مورد استفاده را ننوشته باشیم، برنامه بدون دادن هیچ گونه پاسخی سطر پایینی را با یک علامت + شروع کرده و منتظر کامل کردن سئوال می‌ماند.

---

[1] Console

```
> (5.45-31)*23 - (5.03 + 3
+
```

در مثال داده شده پرانتز دوم بسته نشده است. ما اکنون سطر دوم را بـه صـورت زیر ادامه می‌دهیم

```
> (5.45-31)*23 - (5.03 + 3
+ -4)*
+
```

سطر دوم را دوباره با یک عملگر ضرب به پایان رسانده‌ایم، بنابراین برنامه وارد سطر سوم می‌شود.

```
> (5.45-31)*23 - (5.03 + 3
+ -4)*
+ 2
[1] -595.71
>
```

همانگونه که ملاحظه می‌شود بلاخره در پایان سـطر سـوم دسـتور کامـل شـده و برنامه پاسخ مورد نظر را چاپ کرده است.

همچنین در صورتی که تعدادی پرانتزهای باز و بسته شده برابر نباشـد بـا پیغـام خطایی مواجه می‌شویم. بعنوان مثال مجموع عدد دو و سه با یـک پرانتـز بیشـتر بسته شده است.

```
> (2+3))
Error: unexpected ')' in "(2+3))"
```

دستور خطا بیان می‌کند که یک پرانتز بسته غیر مورد انتظار در رابطه وجود دارد و با حذف آن می‌توان محاسبه را انجام داد.

```
> (2+3)
[1] 5
```

۱-۱-۱ یک ماشین حساب پیشرفته

نرم افزار R در مواقع نیاز به عنوان یک ماشین حساب پیشرفته نیز عملکرده و محاسابات شما را انجام خواهد داد. در شکل زیر چند مثال ساده از انجام عملیات ریاضی در محیط این نرم افزار و خروجی‌های داده شده مشاهده می‌شوند.

```
> 8+2
[1] 10
> 6-5
[1] 1
> 4*3
[1] 12
> 50/2
[1] 25
> 2^5
[1] 32
```

در جدول زیر نیز برخی از توابع موجود در نرم افزار معرفی شده‌اند.

تابع و دستور	خروجی
$log(x)$	لگاریتم x در مبنای طبیعی
$exp(x)$	مقدار e^x
$log(x,n)$	لگاریتم x در مبنای n
$factorial(x)$	محاسبه‌ی مقدار $x!$
$sqrt(x)$	ریشه (جذر) x
$abs(x)$	قدرمطلق x
$choose(n,x)$	ترکیب x از n ، مقدار $\frac{n!}{x!(n-x)!}$
$\cos(x), \sin(x), \tan(x)$	توابع مثلثاتی x بر حسب رادیان
$floor(x)$	جزء صحیح عدد x
$ceiling(x)$	گرد کردن عدد اعشاری x رو به بالا

در محیط نرم افزار R معمولا جهت نمایش اعداد خیلی کوچک یا بزرگ از نمادگذاری e و به صورت زیر استفاده می‌شود:

نماد	مقدار واقعی
$1.2e-5$	0/000012
$3e11$	3×10^{11}

همچنین در برنامه‌ی R داده‌های اعشاری را می‌توان روند کرد برای این کار از دستور round() و به صورت زیر استفاده می‌شود.

```
> x<-5.6734
> round(x)
[1] 6
> round(x,2)
[1] 5.67
> round(x,1)
[1] 5.7
```

در دستور round() ، آرگومان دوم تعداد ارقام اعشاری عدد روندشده را نشان می‌دهد، به طوری که مشخص می‌کند عدد مورد نظر تا چند رقم اعشار روند شود. در صورت وارد نکردن این مقدار عدد داده شده به یک عدد صحیح روند می‌شود.

۱-۱-۱-۱ خارج قسمت و باقیمانده

نرم افزار R امکان محاسبه‌ی خارج قسمت و باقیمانده یک تقسیم را نیز دارد، با استفاده از عملگر %/% مقدار خارج قسمت یک تقسیم و با استفاده از عملگر %% مقدار باقیمانده آن را می‌توان محاسبه کرد.

در شکل زیر مقدار خارج قسمت تقسیم ۱۲۳ بر ۱۴ و سپس مقدار باقیمانده حاصل از این تقسیم گزارش شده است.

```
> 123%/%14
[1] 8
> 123%%4
[1] 3
> 
```

۱-۱-۲ مقدار تعریف نشده ، بی‌نهایت

در محیط نرم‌افزار مقدار مثبت بی‌نهایت توسط inf و منفی بی‌نهایت توسط inf- نمایش داده می‌شوند. همچنین مقادیر تعریف نشده توسط NaN (مخفف Not a Number) و مقادیر بدون جواب (و مقادیر گمشده) توسط Na (مخفف Not availabe) نمایش داده می‌شوند.

۱-۲ بسته‌ها

برنامه R به صورت پیش فرض بسیاری از توابع مورد نیاز ما را دربر داشته و کارهای آماری مورد نظر ما را انجام میدهد؛ با این حال برای انجام محاسبات در بسیاری از مواقع نیاز داریم که به تعریف تابع و دستور جدید برای R بپردازیم؛ با توجه به متن باز بودن برنامه افراد زیادی پس از تعریف توابع و دستورهای جدید آن‌ها را در وبسایت برنامه منتشر می‌کنند؛ به همین دلیل تعداد زیادی از این دستورات که تحت عنوان بسته (Package) شناخته میشوند در وبسایت برنامه در دسترس می‌باشند، ما می‌توانیم بسته مورد نظر را در صورت وجود از وبسایت برنامه دانلود و نصب کنیم؛ این کار به دو صورت امکان پذیر است، یکی به صورت

آنلاین و مستقیم و روش دوم دانلود فایل بسته موردنظر و نصب آفلاین آن، این کار را میتوان از طریق منوی Package و زیرمنوهای آن انجام داد.

۱-۲-۱ نصب آفلاین بسته

با استفاده از مرورگر اینترنتی خود به وبسایت www.r-project.org مراجعه کنید؛ صفحه‌ای باز می‌شود؛ در آن میتوانید لینک‌ها متعددی را مشاهده کنید؛ در قسمت Download and Install R می‌توانید فایل نصب R را نیز بیابید؛ برای دسترسی به لیست بسته‌ها می‌توان به ۲ طریق عمل کرد؛ استفاده از زیرمنوی Task views در سمت چپ صفحه و در قسمت CRAN برای دسترسی به فهرست موضوعی بسته‌ها و یا استفاده از لینک Packages برای دستیابی به فهرست الفبایی بسته‌ها.

پس از استفاده از هر یک از فهرستهای موجود و یافتن بسته مورد نظر با کلیک کردن بر روی نام ان به صفحه بسته مورد نظر منتقل می‌شوید؛ که در این صفحه اطلاعاتی در مورد بسته؛ از قبیل سازنده، نسخه، تاریخ ساخت، زمینه موضوعی و لینکهای دانلود آن مشاهده می‌کنید. همچنین ممکن است بسته موردنظر خود برای اجرا وابسته به بسته‌های دیگر باشد که نام آنها را می‌توان در قسمت Depends مشاهده کرد؛ که این بسته‌ها را نیز باید دانلود و قبل از نصب بسته مورد نظر بر روی برنامه نصب کرد. با توجه به سیستم عامل خود (معمولا ویندوز) فایل مربوط به بسته را دانلود کنید؛ همچنین در صورت وجود در قسمت Reference Manual می‌توان فایل PDF راهنمای بسته و کارکردن با آن را نیز دانلود کرد. پس از دانلود بسته از طریق زیر منوی Install package(s) from local zip files در منوی Packages فایل مورد نظر را انتخاب کنید. به این صورت بسته مورد نظر به برنامه R افزوده می‌شود.

۱-۲-۲ نصب آنلاین بسته

برای نصب آنلاین بسته ابتدا باید از اتصال رایانه‌ی خود به اینترنت مطمئن باشیم،

پس از اطمینان از دسترسی به اینترنت برای نصب انلاین بسته مورد نظر می‌توان از طریق منوی Packages و زیر منوی Install package استفاه کرد؛ برای اولین بار و در صورت عدم انتخاب سرور پیش‌فرض یک پنجره تحت عنوان CRAN mirror را مشاهده خواهید کرد، که در آن باید گزینه‌ی Iran را انتخاب کنید. سپس در پنجره‌ی دوم لیست همه‌ی بسته‌های موجود در وبسایت جهت دانلود را مشاهده خواهید کرد. با انتخاب بسته‌ی مورد نظر، نرم افزار به صورت خودکار بسته را دانلود کرده و آن را نصب می‌کند، با استفاده از این روش چنانچه بسته مورد نظر، برای اجرا بسته‌های پیش نیاز دیگری را لازم داشته باشد، برنامه آن بسته‌ها را نیز به صورت خودکار دریافت و نصب خواهد کرد.

۱-۲-۳ فراخوانی بسته

پس از پایان نصب، بسته موردنظر دیگر به برنامه شما افزوده می‌شود؛ اما با هر بار بستن برنامه و بازکردن مجدد آن در صورت نیاز به بسته نصب شده یا فراخوانی توابع موجود در آن، باید آن بسته را برای برنامه فراخوانی کنید؛ این کار را می‌توانید با ارسال فرمان زیر و زدن دکمه Enter در میزفرمان انجام دهید. به جای Package Name نام بسته‌ی مورد نظر خود را وارد نمائید.

```
> library(Package Name)
```

مثلا جهت فراخوانی یک بسته با نام spatial از دستور زیر استفاده می‌کنیم

```
> library(spatial)
>
```

همچنین این کار با استفاده از منوی Package و زیر منوی Load Package… و انتخاب نام بسته‌ی مورد نظر در پنجره باز شده امکان پذیر می‌باشد.

۱-۲-۴ به روزرسانی بسته

بسته‌های نوشته شده برای برنامه R پیوسته در حال به‌روز شدن بوده و هر چند وقت یکبار یک ویرایش جدید از بسته‌های موجود منتشر می‌شود. شما می‌توانید بسته‌های نصب شده خود را از طریق منوی Package و زیرمنوی Update packages.. با آخرین تغییرت اعمال شده در آنها از طریق توسعه دهندگان و یا سازندگان به روزرسانی کنید.

استفاده از داده ها در نرم افزارR :

در نرم افزار R داده هایی وجود دارند که در بسته های خاص موجود و قبلاً به کار برده شده اند. برای فراخوانی این داده ها ابتدا باید بسته نرم افزاری را که این داده ها در آن در دسترس هستند فراخوانی کرد. قابل توجه است که با استفاده از دستور ()data می توان تمام داده های موجود دریک بسته را نیز مشاهده کرد. این دستور به صورت زیر است که در آن از تمام داده های موجود در بسته MASS داده های phones فراخوانی شده است.

```
> library(MASS)
> data()
> data(phones)
```

```
Data sets in package 'datasets':
AirPassengers           Monthly Airline Passenger Numbers 1949-1960
BJsales                 Sales Data with Leading Indicator
BJsales.lead (BJsales)
                        Sales Data with Leading Indicator
BOD                     Biochemical Oxygen Demand
CO2                     Carbon Dioxide uptake in grass plants
ChickWeight             Weight versus age of chicks on different diets
DNase                   Elisa assay of DNase
EuStockMarkets          Daily Closing Prices of Major European Stock
                        Indices, 1991-1998
........ (more data sets in between are omitted.)
uspop                   Populations Recorded by the US Census
volcano                 Topographic Information on Auckland's Maunga
                        Whau Volcano
warpbreaks              The Number of Breaks in Yarn during Weaving
women                   Average Heights and Weights for American Women
Data sets in package 'MASS':
Aids2                   Australian AIDS Survival Data
Animals                 Brain and Body Weights for 28 Species
... (more data sets are omitted here)
oats                    Data from an Oats Field Trial
painters                The Painter's Data of de Piles
petrol                  N. L. Prater's Petrol Refinery Data
phones                  Belgium Phone Calls 1950-1973
quine                   Absenteeism from School in Rural New South
                        Wales
... (more data sets are omitted here)

Use 'data(package = .packages(all.available = TRUE))'
to list the data sets in all *available* packages.
```

```
> library(MASS)
> data()
> data(phones)
> phones
$year
 [1] 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73

$calls
 [1]   4.4   4.7   4.7   5.9   6.6   7.3   8.1   8.8  10.6  12.0  13.5  14.9
[13]  16.1  21.2 119.0 124.0 142.0 159.0 182.0 212.0  43.0  24.0  27.0  29.0
```

۱-۳ اشیاء؛ تخصیص، فراخوانی و حذف

در برنامه R می‌توان اعداد و اطلاعات را در شئ[۲] ها ذخیره کرد، برای اینکه نتیجه یک محاسبه و یا یک عدد را در یک شئ قرار داده و بعدها آن‌را مجددا فراخوانی کنیم؛ از عملگر تخصیص[۳] که به صورت -> می‌باشد، استفاده می‌کنیم؛ مثلا در پائین حاصل جمع ۶ و ۵ را در شئ n ذخیره کرده و سپس آن‌را فراخوانی می‌کنیم. برای فراخوانی و یا مشاهده مقدار یک شئ تنها کافیست نام آن‌را در میزفرمان تایپ کنیم.

```
> n<-5+6
> n
[1] 11
> 
```

در برنامه R نیز مانند بسیاری از زبانهای برنامه‌نویسی محدودیت‌هایی در نامگذاری اشیاء (در واقع همان متغیرها) وجود دارند.

۱- اسم شئ نباید عدد باشد، یا اینکه با عدد شروع شود.

۲- در برنامه R میتوان اسم شئ را از نام توابع نیز انتخاب کرد، که این برخلاف محدودیت‌های رایج در زبانهای برنامه‌نویسی است.

۳-اسامی شئ(و متغیر)نسبت به حروف کوچک و بزرگ حساس می‌باشد.

۴-در اسم شئ (و متغیر) نمی‌توان از فاصله (یا Space) استفاده کرد.

۱-۳-۱ لیست تمام اشیاء موجود

گاهی مواقع در یک برنامه، ممکن است تعداد اشیاء بسیار زیاد شود؛ به طوری که

---

[۲] Object

[۳] Assign Operator

حتی اسم خیلی از آنها را نیز فراموش کنیم؛ در برنامه *R* می‌توان لیست تمام اشیاء موجود در فضای کار را با استفاده از گزینه‌ی List objects از منوی Misc مشاهده کرد، یا اینکه در میزفرمان با استفاده از دستور ls() و به صورت زیر

```
> ls()
  [1] "a"         "build1"    "build2"    "build3"    "build4"
  [6] "CAD"       "CHF"       "conf"      "ctl"       "dlmFilt1"
 [11] "dlmFilt2"  "dlmFilt3"  "dlmFilt4"  "dlmS1"     "dow"
 [16] "e"         "et"        "et1"       "et2"       "et3"
```

اسامی همه‌ی اشیاء را مشاهده کرد.

گزینه‌ی دیگری که در منوی Misc وجود دارد و در به کارکردن با اشیاء مرتبط می‌باشد؛ زیر منوی Remove all objects می‌باشد؛ به کمک این گزینه تمام اشیاء تعریف شده در برنامه را می‌توان از حافظه آن حذف کرد.

۱-۳-۲ شی و بردار

در برنامه R ما می‌توانیم به شئ بیش از یک مقدار، یا به طور دقیق‌تر یک بردار تخصیص بدهیم؛ بدین منظور از دستور c و مطابق شکل زیر عمل می‌کنیم.

```
> bordar<-c(2,3,5)
> bordar
[1] 2 3 5
>
```

در دستور فوق، مقادیر ۲،۳ و ۵ را به شیء bordar تخصیص داده که با فراخوانی bordar مقدار آن‌ها در خط پاسخ چاپ شده است.

درصورتی که شیء یا برداری را قبلا تعریف ننموده و آن را فرا بخوانیم با خطای زیر مواجه می‌شویم.

```
> H
Error: object 'H' not found
```

کارگاه ۱ : آشنایی با اشیاء

```
name <-"kambiz"; age <- 24;  m1 <-10;  m2 <-15; a <-13
```

// مقادیری را به اشیاء تخصیص داده ایم، توجه کنید که برای تخصیص مقادیر رشته‌ای به اشیاء از نماد "" استفاده کرده‌ایم.

```
> name
[1] "kambiz"
> age
[1] 24
>
```

// اشیاء a و name را فراخوانی کرده‌ایم.

```
> mode(name)
[1] "character"
> mode(a)
[1] "numeric"
```

// با استفاده از دستور mode ، نوع شیءهای name و a چاپ شده است.

```
> ls()
[1] "a"    "age"  "m1"   "m2"   "name"
```

// لیست اسامی تمام اشیاء را درخواست و مشاهده کرده‌ایم.

```
> ls.str()
a :  num 10
age :  num 24
m1 :  num 10
m2 :  num 15
name :  chr "kambiz"
```

// به کمک دستور ls.str() لیست تمام اشیاء و همچنین نوع آنان و مقادیر آنها را را نمایش داده‌ایم.

```
> ls(pat="m")
[1] "m1"   "m2"   "name"
```

// لیست تمامی اشیائی که نام آنان شامل m میباشد.

```
> ls(pat="^m")
[1] "m1" "m2"
>
```

// لیست تمامی اشیائی که نام آنان با m شروع می‌شود.

نکته:

در صورتی که برداری را تعریف کنید که شامل کاراکتر و عدد باشد، خروجی بردار چه عدد و چه کاراکتر را به صورت کاراکتر شناخته می‌شود.

```
> X<-c(2,3,"Ali")
> X
[1] "2"   "3"   "Ali"
```

برای تشخیص نوع یک شی (کاراکتر یا عدد) کافی است یکی از دستورات زیر را به همراه اسم بردار درون پرانتز اجرا نمایید که با TRUE یا FALSE نوع بردار می‌تواند تعین شود. برای مثال برای X فوق داریم:

```
> is.character(X)
[1] TRUE
> is.numeric(X)
[1] FALSE
```

کارگاه ۲ : آشنایی با بردارها و دنباله‌ای از داده‌ها

```
> a<-c(2,3.5,4)
```

// برداری شامل درایه‌های ۲ و ۳,۵ و ۴ را به a تخصیص داده‌ایم.

```
> b<-c(5:12)
> b
[1]  5  6  7  8  9 10 11 12
```

// b اعداد صحیح از ۵ تا ۱۲. با استفاده از عملگر تولید دنباله‌ای از اعداد صحیح :

.

```
> d<-seq(1,6,by=.5)
> d
 [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
```

// d دنباله‌ای از اعداد با فاصله‌ی .5 و با شروع ۱ و پایان ۶.

```
> e<-c(a,b)
> e
 [1]  2.0  3.5  4.0  5.0  6.0  7.0  8.0  9.0 10.0 11.0 12.0
```

// e دو بردار a و b را در برمی‌گیرد، ابتدا مقادیر بردار a و سپس مقادیر بردار b.

```
> f<-c(a,1,a)
> f
[1] 2.0 3.5 4.0 1.0 2.0 3.5 4.0
```

// f بردار a و ۱ و دوباره a را دربرمی‌گیرد.

```
> g<-rep(a,2)
> g
[1] 2.0 3.5 4.0 2.0 3.5 4.0
```

// g بردار a را دو بار پیاپی دربرمی‌گیرد. rep تابع تکرار می‌باشد.

```
> j<-rep(a,each=2)
> j
[1] 2.0 2.0 3.5 3.5 4.0 4.0
```

// j هر درایه بردار a را دو بار پیاپی دربرمی‌گیرد

// با دقت به تفاوت g و j توجه کنید.

```
> h<-seq(length=10, from=0, to=4)
> h
 [1] 0.0000000 0.4444444 0.8888889 1.3333333 1.7777778 2.2222222
 [7] 2.6666667 3.1111111 3.5555556 4.0000000
>
```

// بازه‌ی (۰،۴) را به ۱۰ قسمت برابر می‌شکند یا ایجاد دنباله‌ای به طول ۱۰ از فاصله‌ی 0 تا ۴.

```
> length(j)
[1] 6
```

// با استفاده از دستور length طول بردار j محاسبه شده است.

```
> g[-1]
[1] 3.5 4.0 2.0 3.5 4.0
```

// دستور داده شده بردار g را با حذف کردن درایه اول آن برمی‌گرداند. به‌طور کلی دستور g[-k] بردار g را با حذف درایه‌ی k ام آن برمی‌گرداند.

```
> g[-c(2,3)]
[1] 2.0 2.0 3.5 4.0
```

// دستور داده شده بردار g را با حذف کردن درایه‌های دوم و سوم آن برمی‌گرداند. به‌طور کلی دستور g[-c(k,w)] بردار g را با حذف درایه‌ی kو w ام آن برمی‌گرداند.

```
> g<-c(2,3.5,4,2,3.5,4)
> g[4]
[1] 2
```

// درایه‌ی چهارم بردار g فراخوانی می‌شود.

```
> g[c(1,4,5)]
[1] 2.0 2.0 3.5
```

// درایه‌های اول، چهارم و پنجم بردار g را فرامی‌خواند.

کارگاه ۳: عملگر منطقی

ابتدا بردار vec را به صورت زیر تعریف می‌کنیم

```
> vec<-c(4,6,-3,7,9,0)
> vec
[1]  4  6 -3  7  9  0
> 
```

حال با استفاده از عملگر مقایسه < نتایج منطقی زیر را خواهیم داشت، در واقع می‌خواهیم بدانیم کدام یک از عناصر بردار داده شده از ۱ بزرگتر هستند.

```
> vec > 1
[1]  TRUE  TRUE FALSE  TRUE  TRUE FALSE
> 
```

که برنامه به صورت TRUE و FALSE پاسخ مقایسه مورد نظر را نشان می‌دهد.

نکته: توجه کنید که نرم افزار R عبارت TRUE را مانند عدد ۱ و False را مانند عدد ۰ می‌شناسد (مانند یک تابع نشانگر).

```
> 1*(vec>6)
[1] 0 0 0 1 1 0
> 1+(vec>6)
[1] 1 1 1 2 2 1
```

حال با استفاده از عملگرهای منطقی & (و)، | (یا) و ! (نه) به صورت زیر نتایج منطقی زیر را خواهیم داشت

```
> vec > 1 | vec<6
[1] TRUE TRUE TRUE TRUE TRUE TRUE
> vec > 1 & vec<6
[1]  TRUE FALSE FALSE FALSE FALSE FALSE
> !vec > 1
[1] FALSE FALSE  TRUE FALSE FALSE  TRUE
```

۱-۴ مسیرکار و خواندن داده‌ها

برای کارکردن و خواندن فایلها برنامه‌ی R ازیک مسیر تحت عنوان مسیرکار[۴] استفاده می‌کند؛ در واقع R فایلها را در این مسیر ذخیره می‌کند. با استفاده از دستور getwd() می‌توان آدرس مسیرکار را پیدا کرد؛ همچنین در صورت نیازمی‌توان از طریق دستور setwd("address") آدرس مسیرکار را تغییر داد. مثلا ما می‌خواهیم مسیرکار را به آدرس D:/statistics تغییر بدهیم؛ از دستور زیر استفاده می‌کنیم:

```
> setwd("D:/statistics")
```

لازم به ذکر است که به هنگام تغییر مسیرکار باید ابتدا پوشه‌ی مورد نظر در آدرس داده شده را ایجاد کرده باشیم. مثلا در این حالت ما باید در درایو D یک پوشه با اسم statistics قبلا ایجاد کرده باشیم.

پس از این به بعد تمام فایل‌هایی را که قرار است توسط برنامه بخوانیم و یا ذخیره کنیم، به طور پیش فرض در آدرس فوق قرار می‌گیرند.

R توانایی خواندن داده‌های ذخیره‌شده در فایلهای متنی با فرمت ASCII را

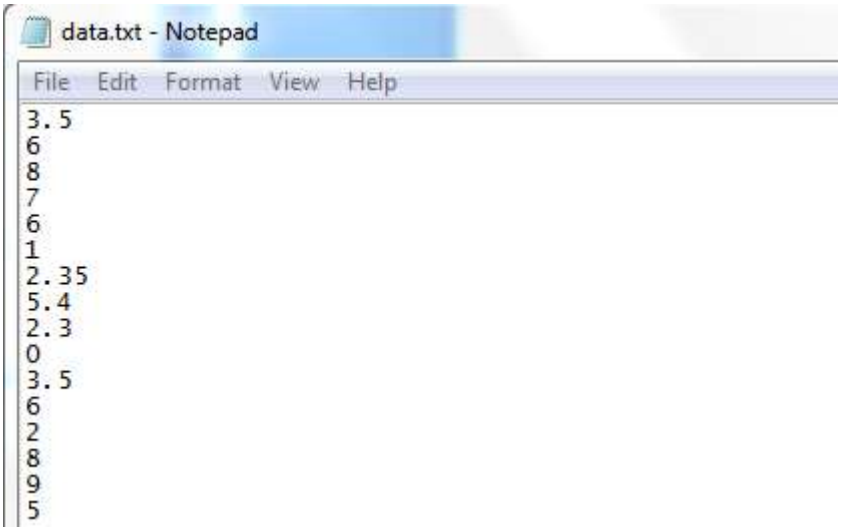
---

[۴] Working directory

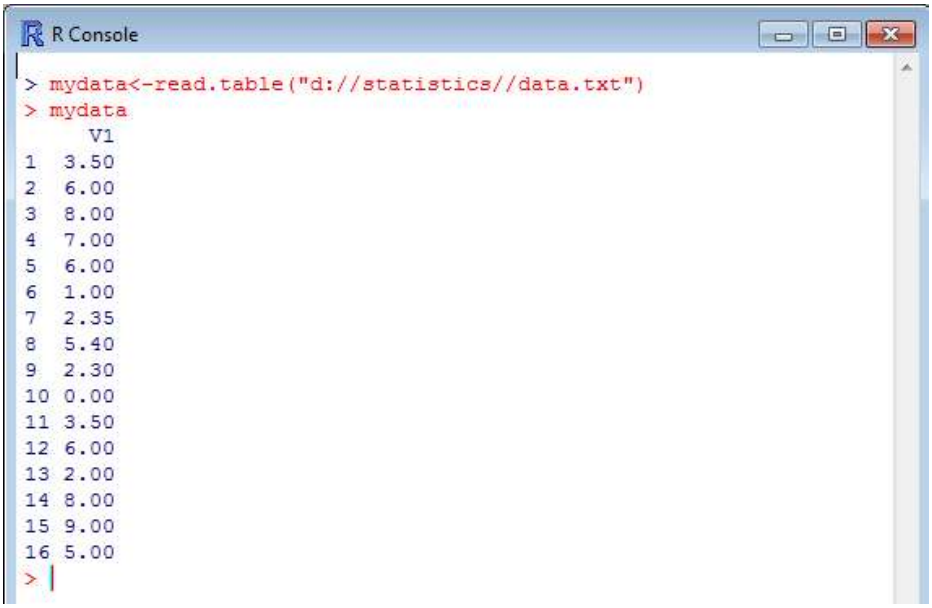
داراست، برای این‌کار فایل مورد نظر را در مسیرکار قرار داده و سپس از طریق دستورات read.table و scan و read.fwf آن‌را در برنامه R فرامی‌خوانیم. البته R توانایی فراخوانی داده‌های ذخیره‌شده توسط Excel, Spss, Access و ... را داراست؛ که در این موارد باید بسته مورد نظر را برای برنامه R نصب کرد. ما در اینجا ابتدا خود را به خواندن داده‌های متنی محدود می‌کنیم.

۱-۴-۱ خواندن داده‌ها از روی فایل

برای وارد کردن داده‌ها در نرم افزار R ، ابتدا با استفاده از ویرایشگر متن notepad در محیط ویندوز، داده‌های خود را در یک فایل txt.* به گونه‌ای که در هر سطر آن یک عدد وارد شده باشد ذخیره می‌کنیم.



سپس با استفاده از دستور read.table و به صورت زیر داده‌ها را فراخوانی می‌کنیم.



داده‌ها را با استفاده از دستور read.table فراخوانی کرده و آنها را با استفاده از عملگر تخصیص <- ، در یک شیء ، در اینجا شیء mydata ، قرارمی‌دهیم.

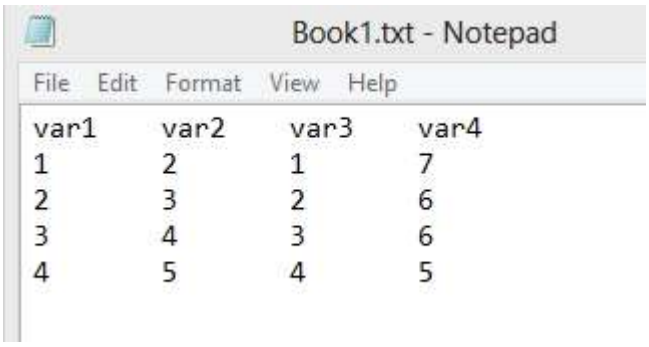
۱-۴-۲ خواندن داده ها از برنامه‌ی اکسل

فرض کنید ما یک سری مجموعه داده در برنامه اکسل داریم، داده ها به صورت ستونی و مانند شکل زیر در محیط این برنامه وارد شده‌اند.

	A	B	C	D	E
1	var1	var2	var3	var4	
2	1	2	1	7	
3	2	3	2	6	
4	3	4	3	6	
5	4	5	4	5	

برای فراخوانی این داده‌ها در محیط R ابتدا باید آنها را در یک فایل txt ذخیره کرد. به گونه‌ای که در فایل ذخیره شده داده‌های هر متغیر در زیر هم و به صورت ستونی واقع شوند، برای این کار از منوی ... Save as گزینه‌ی Other Formats

را انتخاب کرده و سپس در پنجره باز شده فایل مورد نظر را با انتخاب فرمت text(tab delimited) در قسمت Save as type ذخیره می‌کنیم. با باز کردن فایل موجود در برنامه ویرایشگر متن Notepad خواهیم دید که داده‌ها به صورت زیر ذخیره شده‌اند



حال برای انتقال داده ها به محیط R از دستور read.table استفاده می‌کنیم.

```
> data<-read.table("d:\\data\\book1.txt",header=T)
> data
  var1 var2 var3 var4
1    1    2    1    7
2    2    3    2    6
3    3    4    3    6
4    4    5    4    5
>
```

در دستور داده شده برای فراخوانی داده‌ها گزینه‌ی header=T در صورتی که در داده‌های موجود ردیف اول حاوی عنوان و نام متغیرها باشد وارد می‌شود. داده‌ها در این حالت و به اصطلاح در یک چارچوب داده‌ای[۵] فراخوانی شده‌اند. حال برای کارکردن با این داده‌ها مایل هستیم که داده‌های مربوط به هر متغیر را در یک شیء در محیط برنامه R نسبت دهیم. برای اینکار از نماد $ و به صورت زیر استفاده می‌کنیم.

---

۵ Data Frame

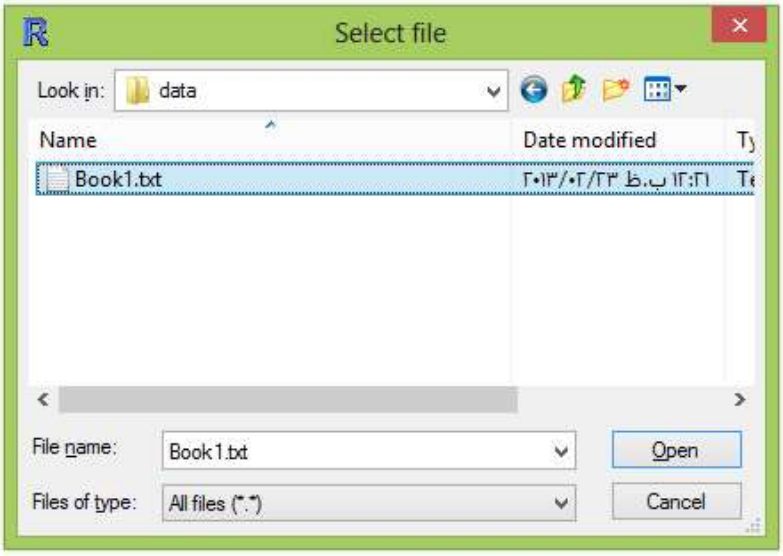
```
> x4<-data$var4
> x4
[1] 7 6 6 5
```

همانگونه که مشاهده می‌شود داده‌های مربوط به ستون چهارم یا var4 در متغیر x4 ذخیره شده‌اند. در برنامه‌ی داده شده برای تخصیص داده‌ها به یک شیء پس از انتخاب نام متغیر و استفاده از عملگر تخصیص ابتدا نام چارچوب داده‌ای، در این مثال data ، سپس نماد $ و پس از آن نام متغیر مورد نظر در چارچوب داده‌ای را (در این مثال var4) وارد می‌کنیم.

در صورتی که داده‌ها در محیط اکسل دارای ردیف نام متغیر نباشند، در دستور read.table دیگر نیازی به استفاده از آرگومان header=T نبوده و پس از فراخوانی برنامه‌ی R به صورت خودکار در چارچوب داده‌ای داده شده به متغیرها عناوین V1، V2 و ... را اختصاص می‌دهد.

همچنین در محیط برنامه R این امکان وجود دارد که داده‌ها را بتوان بدون وارد کردن آدرس و فایل مورد نظر و با استفاده از دستور زیر از طریق پنجره‌ی انتخاب فایل انتخاب کرد.

```
> data<-read.table(file.choose(),header=T)
```



۱-۴-۳ وارد کردن داده به طور مستقیم

یکی از روش‌های دیگر واردکردن داده‌ها به محیط نرم افزار R وارد کردن داده‌ها به طور مستقیم و با استفاده از صفحه کلید می‌باشد. در این حالت از دستور scan و تایپ داده مورد نظر و زدن دکمه‌ی Enter پس از هر داده استفاده می‌شود. پس از وارد کردن آخرین داده با دوبار زدن متوالی دکمه‌ی Enter کار خواندن داده‌ها به پایان خواهد رسید.

```
> y<-scan()
1: 34
2: 28
3: 31
4: 29.5
5: 30.2
6: 56
7:
Read 6 items
>
```

همچنین دستور scan را می‌توان به مانند دستور read.table برای خواندن داده‌ها از فایل‌های متنی با فرمت‌های txt و dat استفاده کرد.

```
> mydata<-scan("d://data//data.txt")
Read 17 items
> mydata
 [1] 23.00 14.20 21.00 10.30 12.00 13.00 16.50 14.20 10.60 15.00
[11] 14.00 19.00 20.00 11.30 12.00 18.00 10.25
>
```

در این حالت تفاوت دستور scan و read.table در این است که دستور scan داده‌ها را به صورت یک بردار و پشت سر هم فرخوانی می کند. اما دستور read.table برای خواندن داده‌های چندمتغیره (غیربرداری یا ماتریسی) مناسب می‌باشد. همچنین skip یک آرگومان تابع scan می‌باشد، که یک مقدار عددی گرفته و از ابتدای داده‌های موجود در فایل داده ها به تعداد مقدار داده شده را نادیده می‌گیرد.

در نرم افزار R اعدادی که در داخل [] و در سمت چپ و ابتدای هر سطر از داده‌ها قرار گرفته‌اند، شمارنده‌ی داده‌ها هستند. در مواردی که با تعداد داده‌های بسیار روبه‌رو هستیم می‌توانند ما را در یافتن داده‌ها کمک کنند، مثلا در شکل فوق عدد ۱۱ در ابتدای سطر دوم بیانگر این است که اولین داده‌ی وارد شده در این سطر داده‌ی ۱۱ام می‌باشد.

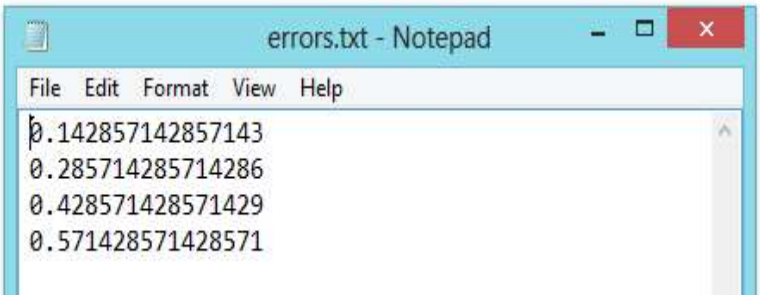
۱-۴-۴ ذخیره کردن داده‌ها

در بسیاری از موارد ممکن است پس از بررسی یک سری داده و یا اعمال تغییرات مورد نظر در آنها، آن داده‌ها و یا باقیمانده‌های حاصل از برازش یک مدل را در یک فایل متنی ذخیره کنیم و یا اینکه گاهی اوقات لازم می‌شود این خروجی‌ها را در یک نرم افزار آماری دیگر مورد تجزیه و تحلیل قرار دهیم. در نرم افزار R با استفاده از دستور write.table این کار امکان پذیر است. به عنوان مثال باقیمانده‌های حاصل از برازش یک مدل آماری را در یک شیء با نام errers ذخیره کرده‌ایم، حال به صورت زیر آنها را در یک دایل متنی با نام errors.txt و در مسیر /D:/dataanalysis ذخیره می‌کنیم.

```
> errors
[1] 0.1428571 0.2857143 0.4285714 0.5714286
> write.table(errors, file = "D:\\dataanalysis\\errors.txt",
+ row.names=FALSE, col.names=FALSE)
```

به نحوه‌ی نوشتن آدرس فایل و استفاده از \\ توجه نمائید. هم چنین در دستور فوق آرگومان‌های row.names و col.names مشخص کننده‌ی نحوه‌ی ذخیره داده‌ها هستند که آیا اسم یا شماره‌ی ردیف داده‌ها نیز ذخیره شود یا نه؟ در دستور دادن مقدار FALSE (با حروف بزرگ) اسم سطر یا ستون ذخیره نشده و در صورت دادن مقدار TRUE اسم و عنوان سطر یا ستون‌ها نیز با داده‌ها ذخیره می‌شوند.گفتنی است که تابع write.table قابلت ذخیره کردن داده‌ها را در یک فایل با پسوندهای dat و data نیز دارد.

```
> errors
[1] 0.1428571 0.2857143 0.4285714 0.5714286
> write.table(errors, file = "D:\\dataanalysis\\errors.txt",
+ row.names=FALSE, col.names=FALSE)
>
```

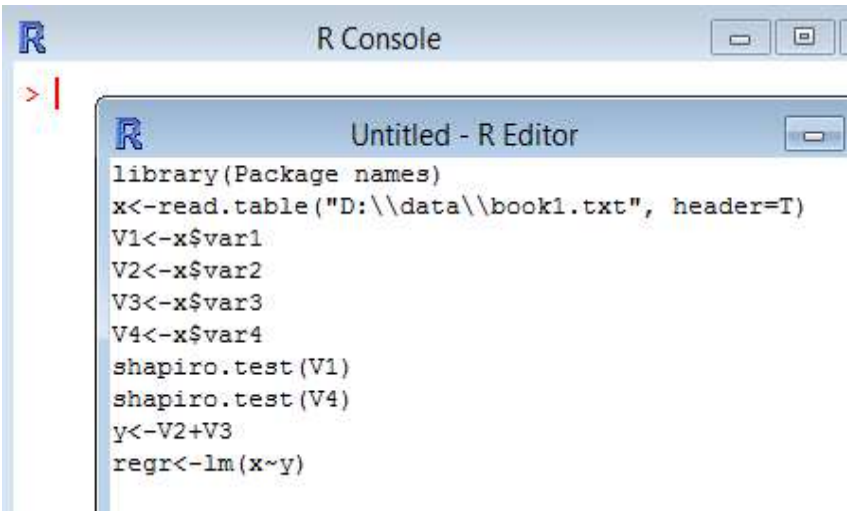


۱-۴-۵ ویرایشگر R

همانگونه که تاکنون مشاهده کرده‌اید ما در محیط برنامه‌ی R دستورهای خود را در قسمت میزکار برنامه و R Consule وارد کرده و در همان جا و پس از پایان دستور، پاسخ برنامه را دریافت کرده‌ایم. این روش استفاده از برنامه R در حالاتی که با یک مثال و یا کاربرد ساده مواجهه هستیم مناسب و کارا است. در بسیاری از موارد نیاز داریم در یک زمان مجموعه‌ای از دستورها و تابع‌ها را در برنامه اجرا کرده، در صورت بروز خطا آنها را ویرایش و مهمتر از همه آنها را ذخیره کرده و در استفاده‌های بعدی از آنها استفاده کنیم. بدین منظور ما می‌توانیم از ویرایشگر برنامه (R Editor) استفاده کنیم که از منوی File گزینه‌ی New Script را انتخاب می‌کنیم. پس از آن یک پنجره‌ی ویرایشگر با عنوان Untitled – R Editor باز می‌شود. حال در این قسمت می‌توانیم دستورهای خود را نوشته، آنها را اجرا، ذخیره و بازخوانی کرده و در قسمت R Console خروجی برنامه را مشاهده کنیم.

پس از نوشتن دستورهای مورد نظر تنها کافی است از Ctrl+A برای انتخاب متن

دستورها استفاده کرده و سپس با زدن Ctrl+R خروجی مورد نظر را در قسمت R Console مشاهده کنید. در صورتی که نیاز باشد تنها بخشی از دستورات مورد نظر در ویرایشگر را اجرا کنیم ابتدا آن دستورات را از طریق نشان‌گر ماوس انتخاب کرده و سپس با زدن دکمه Ctrl+R و یا راست کلیک کردن و انتخاب گزینه‌ی Run line or Selection قسمت مورد نظر خود را اجرا کنیم. همچنین برای اجرای تنها یک سطر از برنامه کافی است بر روی سطر مورد نظر کلیک کرده و از طریق صفحه کلید Ctrl+R را فشار دهیم.



از طریق منوی File و گزینه‌ی Save می‌توان متن دستورات موجود در ویرایشگر را ذخیره کرده و سپس در اجراهای بعدی برنامه از همان منوی File و گزینه‌ی Open Scripts دوباره دستورات و کدهای نوشته شده را به محیط برنامه‌ی R فراخوانی کرد.

۱-۵ چارچوب داده‌ای

یک چارچوب داده‌ای عبارت است از مجموعه‌ای از داده‌ها و یا عوامل که به صورت سطری و ستونی در کنار هم قرار می‌گیرند. به صورتی‌که در یک مجموعه داده‌ای ممکن است با بیش از یک ستون مواجه شده که هر ستون نمایانگر مقادیر یک

متغیر می‌باشد. در چارچوب داده‌ای طول بردارهای ستونی با هم یکسان می‌باشد و در هر ردیف نیز داده‌های موجود مربوط به یک تیمار یا واحد آزمایشگاهی وارد شده‌اند.

در جدول زیر یک چارچوب داده‌ای از وزن و قد چند نفر نشان داده شده است.

نام	قد	وزن
آرش	۱۷۴	۷۲
امین	۱۸۷	۷۳
مهرداد	۱۷۹	۷۱
مهران	۱۸۰	۷۰
کامیار	۱۷۹	۶۸
زانیار	۱۸۱	۶۶
شهروز	۱۷۶	۷۰

در محیط نرم افزار R جدول فوق را می‌توان به صورت زیر و به کمک تابع data.frame در یک چارچوب داده‌ای تعریف کرد.

```
> weight<-c(72,73,71,70,68,66,70)
> length<-c(174,187,179,180,179,181,176)
> name<-c("arash","amin","mehrdad","mehran",
+ "kamyar","zanyar","Shahrooz")
> df<-data.frame(name,length,weight)
> df
      name length weight
1    arash    174     72
2     amin    187     73
3  mehrdad    179     71
4   mehran    180     70
5   kamyar    179     68
6   zanyar    181     66
7 Shahrooz    176     70
>
```

همانگونه که پیشتر گفتیم تـابع read.table داده‌هـا را در یـک چـارچوب داده‌ای فرامی‌خوند. در صورتی که تنها خواستار استفاده از متغییر قـد باشـیم، داده‌هـای مربوط به آن‌را می‌توانیم با استفاده از نماد $ و به صورت زیر از چـارچوب داده‌ای فرا بخوانیم.

```
> leng<-df$length
> leng
[1] 174 187 179 180 179 181 176
> 
```

همچنین این کار را با استفاده از دستور attach می‌توان انجام داد، بـه صـورتی‌که نام چارچوب داده‌ای را ضمیمه‌ی برنامه کرده و سپس برای فراخوانی هـر یـک از متغیرهای موجود در چارچوب تنها لازم است نام آن را فراخواند و دیگـر نیـاز بـه استفاده از نماد $ و نام چارچو ب نیست:

```
> attach(df)
The following object(s) are masked _by_ '.GlobalEnv':

    length
> length
[1] 174 187 179 180 179 181 176
```

با استفاده از دستور detach نیز می‌توان یک چارچوب داده‌ای را از حالـت attach خارج کرد.

```
> detach(df)
```

در برنامه‌ی R برای یک چارچوب داده‌ای (و همچنین برای هر ماتریس) می‌تـوان به انتخاب یک سطر یا ستون و یا یک مولفه‌ی خاص پرداخت. به عنوان مثـال در چارچوب داده‌ای داده شده سطر ۵ ام را که داده‌های نـام، قـد و وزن مربـوط بـه "کامیار" را در خود دارد به صورت زیر انتخاب می‌کنیم

```
> df[5,]
    name length weight
5 kamyar    179     68
```

انتخاب ستون دوم، داده‌های مربوط به قد را به صورت زیر

```
> df[,2]
[1] 174 187 179 180 179 181 176
```

و داده مربوط به ردیف ۵ و ستون ۲ (قد کامیار) را به صورت زیر انتخاب می‌کنیم

```
> df[5,2]
[1] 179
```

همچنین در یک چارچوب داده‌ای مقادیر موجود در آن را می‌توان با استفاده از عملگر تخصیص تغییر داد، مثلا به صورت زیر درایه‌ی مربوط به سطر ۵ و ستون ۲ را به مقدار ۱۷۴ تغییر می‌دهیم

```
> df[5,2]<-174
> df[5,2]
[1] 174
```

در چارچوب داده‌ای داده شده انتخاب مشخصات افرادی که قد آنها از ۱۸۰ بلندتر بوده است به صورت زیر می‌باشد

```
> df[df$length>180,]
    name length weight
2   amin    187     73
6 zanyar    181     66
```

به نحوه‌ی تعریف شرط موردنظر توجه کنید، ما قصد انتخاب سطرهایی از چارچوب داده‌ای را داشته‌ایم که قد آنها بزرگتر از ۱۸۰ باشد پس شرط را در قسمت اول داخل [,] تعریف می‌کنیم.

انتخاب ستونی با قد بزرگتر از ۱۸۰ و وزن کمتر و مساوی با ۷۰ به صورت زیر

می‌باشد.

```
> df[df$length>180 & df$weight<=70,]
    name length weight
6 zanyar    181     66
```

راهنمای R

راهنمای موجود در برنامه‌ی R می‌تواند یکی از مراجع بسیار خوب و کمک کننده‌ی ما در استفاده از برنامه و بسته‌های آن باشد. که از طریق منوی Help در دسترس می‌باشد، زیرمنوهای موجود در این منو به صورت زیر می‌باشند:

Console: این زیر منو تمام کلیدهای میانبر فعال در میز فرمان را به ما نشان می‌دهد.

FAQ on R for Windows / FAQ on R: بیشترین سئوالات پرسیده شده در مورد برنامه و پاسخهای اولیه به آنها.

Manuals in PDF: راهنمای برنامه به صورت PDF

R functions(texts) : راهنمای توابع R، با کلیک کردن بر روی این گزینه یک پنجره باز شده که شما باید در کادر موجود نام تابع را نوشته؛ تا برنامه راهنمای آن‌را به شما نشان دهد. البته برای این کار می‌توانید در میزفرمان دستور زیر را نیز اجرا کنید:

```
> help("function name")
```

Html help: راهنمای جامع نرم افزار به صورت آفلاین. این راهنما همچنین از طریق میز فرمان و با استفاده از دستور زیر نیز در دسترس است:

```
> help.start()
```

Search help : جستجو در راهنمای نرم‌افزار.

Search.r-project.org : جستجوی آنلاین در سایت برنامه.

Apropos : جستجو در توابع؛ یافتن توابعی که نام آنها شامل کلمه مورد جستجو می‌باشد؛ همچنین به صورت زیر و در قسمت R Console نیز قابل اجراست:

```
> apropos("کلمه مورد جستجو")
```

مثلا در زیر فهرست تمام تابع‌های در دسترس برنامه که نام آنها شامل line می‌باشد نمایش داده شده است.

```
> apropos("line")
 [1] ".__C__LinearMethodsList" "abline"
 [3] "contourLines"            "findLineNum"
 [5] "getSrcLines"             "line"
 [7] "linearizeMlist"          "lines"
 [9] "lines.default"           "lines.ts"
[11] "matlines"                "qqline"
[13] "readline"                "readLines"
[15] "smooth.spline"           "spline"
[17] "splinefun"               "splinefunH"
[19] "writeLines"              "xspline"
>
```

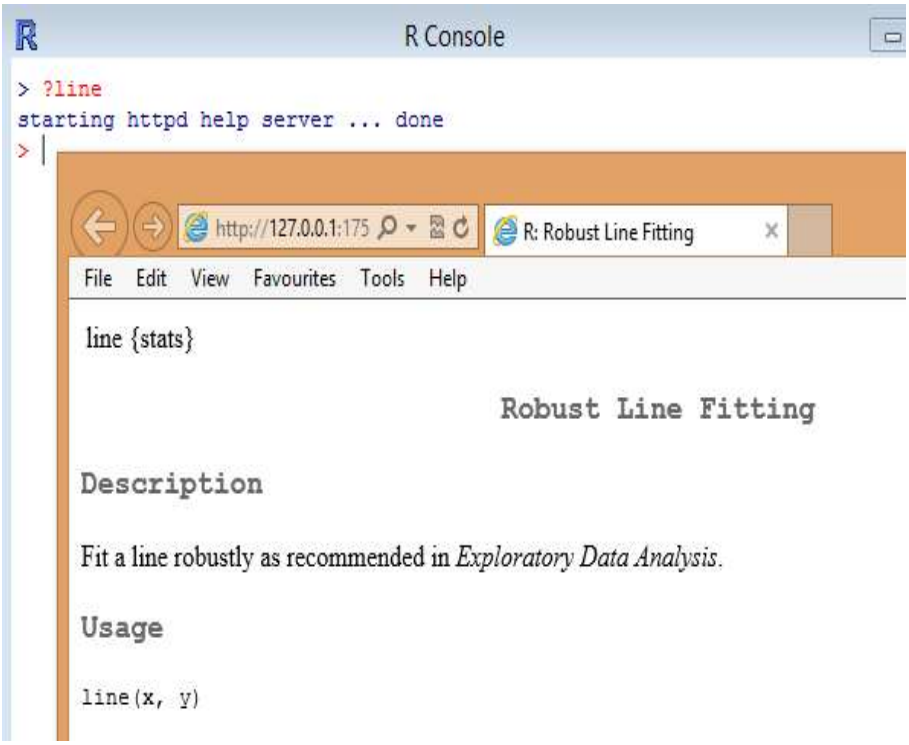
R Project home page: ورود به سایت برنامه.

CRAN home page: ورود به سایت آرشیو جامع برنامه.

About: نمایش نسخه برنامه.

در محیط برنامه‌ی R و یکی از رایج‌ترین راه‌های استفاده از راهنمای برنامه استفاده از عملگرهای کمک ? و ?? می‌باشد. در صورت نیاز به راهنمایی در مورد یکی از دستورها و یا تابع‌های برنامه و یا بسته‌های افزوده شده به برنامه، می‌توانیم

از عملگر function name? استفاده کنیم. با نوشتن این دستور در R comsole و در صورتی‌که تابع مورد نظر وجود داشته باشد، پنجره‌ی راهنمای تابع مورد نظر در مرورگر وب شما باز می‌شود (اتصال به اینترنت لازم نیست).



در راهنمای داده شده و در سطر اول نام تابع و بسته‌ای که تابع در آن گنجانده شده است در داخل {} آمده است. در ادامه معمولا توصیفی از تابع مورد نظر، نحوه‌ی استفاده از آن، آرگومان‌های ورودی و خروجی پس از اجرای آن و در بیشتر موارد یک یا چند مثال از آن تابع نمایش داده شده‌اند.

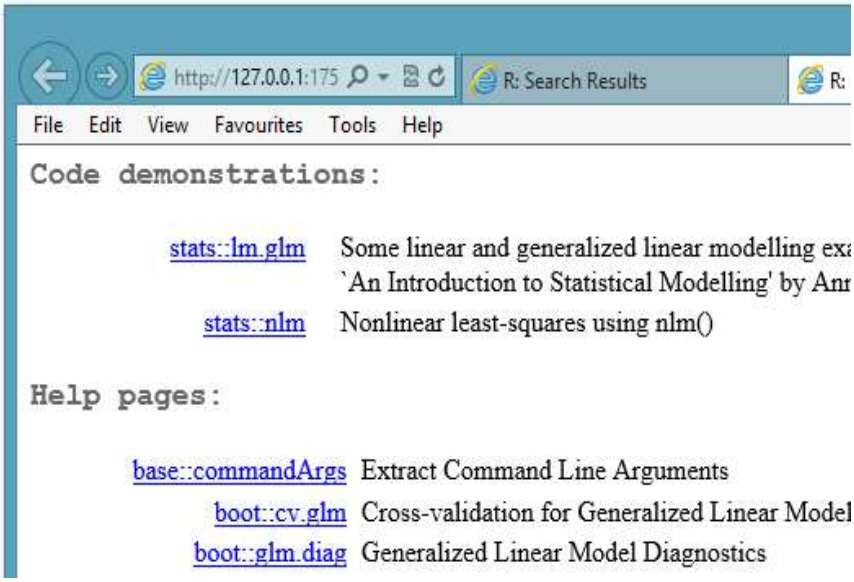
عملگر ?? در مقایسه با عملگر ? جستجوی عمیق‌تری در راهنمای برنامه انجام می‌دهد. در هنگام استفاه از عملگر ? در صورتی‌که نام تابع مورد نظر صحیح وارد نشده باشد و یا اینکه تابعی با آن عنوان در برنامه و بسته‌های فراخوانی شده

موجود نباشد، برنامه پیام خطای زیر را نمایش می‌دهد.

```
> ?liner
No documentation for 'liner' in specified packages and libraries:
you could try '??liner'
>
```

در چنین حالتی خود برنامه ما را به استفاده از عملگر ?? ترغیب می‌کند. این عملگر برخلاف ? که تنها در نام تابع و دستورهای موجود به جستجو می‌پردازد، به جستجوی عمیق‌تری در فایل راهنمای موجود در برنامه می‌پردازد.

```
> ??linear
>
```



در پنجره‌ی باز شده، ابتدا فهرست مثال‌های موجود در برنامه که عنوان آنها شامل linear می‌باشد آمده است که با کلیک بر روی آنها می‌توان به کد و دستورات مورد نظر دسترسی پیدا کرد. در قسمت دوم و در زیر Help pages تمام توابع موجود که در قسمت توصیف و یا تشریح آنها از کلمه linear استفاده شده است و در راهنمای آنها آمده است نمایش داده شده اند که با کلیک بر روی هر یک از

آنها صفحه‌ی راهنمای تابع موردنظر باز می‌شود. نحوه‌ی نمایش نام تابع‌ها به این صورت است که ابتدا نام بسته‌ی حاوی تابع و سپس :: و پس از آن در سمت راست نام تابع مورد نظر آمده است.

به عنوان یک مثال دیگر فرض کنید می‌خواهیم برای یک سری داده آزمون نرمال بودن آنها را انجام دهیم اما از وجود یا عدم وجود یک تابع خاص برای انجام چنین آزمونی در برنامه و بسته‌های نصب شده آگهی نداریم. در این حالت با جستجوی normality نتیجه زیر حاصل می‌شود.

که راهنمای برنامه به ما می‌گوید در بسته‌ی stats یک تابع با عنوان shapiro.test برای انجام آزمون نرمال بودن وجود دارد. با کلیک بر روی نام تابع می‌توان به جزئیات بیشتری در مورد این تابع و آزمون مورد نظر و نحوه‌ی انجام و خروجی آن دست یافت.

تمرین

۱) دستور مناسب برای اجرای محاسبات زیر را بنویسید.

الف) $\frac{3+4^2-12}{\sqrt{7-2}-3}$

ب) $\frac{1}{\sqrt{2\pi}} e^{\frac{-(3-2)^2}{2}}$

ج) $0.5^{3.5} \times e^{\frac{-0.5}{3}} \times \frac{3^{4.5}}{\Gamma(4.5)}$

د) $\frac{\sin(12)}{\cos(3)+\tan(0.7)}$

ماتریس‌ها و بردارها

نرم افزار R به عنوان یک برنامه آماری امکان انجام محاسبات و تجزیه و تحلیل آماری داده‌های برداری، ماتریسی و آرایه‌های چندبعدی را به کاربران می‌دهد. در این بخش به معرفی توابع مربوط به بردارها و ماتریس‌ها در R پرداخته و نحوه کارکردن با بردار و ماتریس‌ها را مورد بررسی قرار می‌دهیم.

۲-۱ بردارها

پیش‌تر و در بخش معرفی اشیاء اندک توضیحاتی در مورد بردارها ارائه داده شد. قبل از ورود به مبحث بردارها باید به این نکته اشاره کرد که R متغیرهای عددی (اسکالرها) را به عنوان بردارهای یک بعدی می‌شناسد.

۲-۱-۱ تعریف بردار

در برنامه‌ی R برای تعریف و ایجاد یک بردار می‌توان از تابع‌های c ، seq و rep استفاده کرد. هر چند برداری شامل دنباله‌ای از اعداد صحیح را با استفاده از عملگر : و به صورت زیر می‌توان ایجاد کرد.

```
> y<-4:12
> y
[1]  4  5  6  7  8  9 10 11 12
> x<-8:2
> x
[1] 8 7 6 5 4 3 2
```

بردار y را به صورت زیر نیز می‌توان تعریف کرد

```
> y<-c(4,5,6,7,8,9,10,11,12)
> y
[1]  4  5  6  7  8  9 10 11 12
> y<-seq(4,12)
> y
[1]  4  5  6  7  8  9 10 11 12
```

همچنین بردار z را می‌توان از پشت سر هم قرار دادن دو بردار x و y به صورت زیر تعریف کرد

```
> z<-c(x,y)
> z
 [1]  8  7  6  5  4  3  2  4  5  6  7  8  9 10 11 12
```

دستور seq را نیز می توان برای تولید دنباله‌ای از اعداد صحیح به صورت زیر مورد استفاده قرار داد که در این حالت یک دنباله از اعداد صحیح را با آغاز از عدد اول خواهیم داشت

```
> seq(3,10)
[1]  3  4  5  6  7  8  9 10
> seq(3,-4)
[1]  3  2  1  0 -1 -2 -3 -4
>
```

در دستور زیر یک دنباله از اعداد را با آغاز از ۲ و پایان ۱۶ و با گام ۴ ایجاد کرده‌ایم

```
> seq(2,16,4)
[1]  2  6 10 14
```

در زیر نیز با آغاز از ۱ و با گام ۰,۵ و پایان ۵ بردار t را ایجاد کرده‌ایم

```
> t<-seq(1,5,0.5)
> t
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

در زیر نیز دنباله‌ای به طول ۸ از آغاز ۲ و پایان ۴ با فاصله‌ی یکسان از هم ایجاد کرده‌ایم

```
> d<-seq(length=8, from=2, to=4)
> d
[1] 2.000000 2.285714 2.571429 2.857143 3.142857 3.428571
[7] 3.714286 4.000000
```

در محیط برنامه R می‌توان از دستور rep نیز در ایجاد و تعریف بردارها با کمک تکرار استفاده کرد. در زیر ابتدا بردار a را تعریف کرده و سپس در ادامه بردار b را

با دوبار تکرار a پشت سر هم ایجاد کرده‌ایم

```
> a<-c(2,5,8)
> b<-rep(a,2)
> b
[1] 2 5 8 2 5 8
```

بردار f را با استفاده از دستور rep و از بردار a به گونه‌ای ایجاد می‌کنیم که هر درایه‌ی بردار a دوبار تکرار شود. به نحوه‌ی تعریف بردارهای f و b توجه نمائید.

```
> f<-rep(a,each=2)
> f
[1] 2 2 5 5 8 8
```

بردار c را با ۳ بار تکرار عدد ۵ و با استفاده از دستور rep به صورت زیر ایجاد کرده‌ایم

```
> c<-rep(5,3)
> c
[1] 5 5 5
```

با استفاده از دستور rep می‌توان به تکرار یک بردار یا عدد، به مقدار آرگومان دوم و یا تکرار عناصر یک بردار به تعداد عناصر یک بردار هم اندازه پرداخت، در مثال زیر ۲ را با یک تکرار و ۴ را با ۳ تکرار در کنار هم آورده‌ایم

```
> rep(c(2,4),c(1,3))
[1] 2 4 4 4
>
```

تکرار عناصر بردار a با طول سه، به طوری‌که عنصر اول یکبار، عنصر دوم دوبار و عنصر سوم سه بار تکرار شود، به صورت زیر می‌باشد

```
> rep(a,1:3)
[1] 2 5 5 8 8 8
>
```

گاهی در برنامه نویسی نیازمند وجود روشی برای ایجاد یک بردار با تعداد اعضای صفر هستیم که پس از تعریف بتوان مقادیری را در آن بردار بریزیم. (دستور length(x) طول بردار x یا به عبارت دیگر تعداد اعضای x را نمایش می‌دهد.)

```
> x<-rep(0,0)
> length(x)
[1] 0
> x<-c()
> length(x)
[1] 0
```

۲-۱-۲ توابع برداری

در برنامه‌ی R تعدادی از توابع تعریف شده‌اند که یک بردار را به عنوان ورودی می‌پذیرند. یکی از این توابع تابع sum(y) می‌باشد که با استفاده از آن می‌توان مجموع درایه‌های بردار y را محاسبه کرد. فرض کنید بردار y به صورت زیر تعریف شده باشد، بنابراین

```
> y
 [1]  4  5  6  7  8  9 10 11 12
> sum(y)
[1] 72
>
```

mean(y) میانگین درایه‌های بردار y را بازمی‌گرداند

```
> mean(y)
[1] 8
```

mean(y, trim=p) میانگین اصلاح شده درایه‌های بردار y را بازمی‌گرداند (p عددی بین ۰ تا ۰/۵ است که نشان دهنده نسبت حذف داده‌ها از ابتدا و انتهای بردار است) (کاربرد در زمان حضور داده پرت)

```
> mean(y,trim=.2)
[1] 8
```

median(y) میانه‌ی بردار y را محاسبه می‌کند

```
> median(y)
[1] 8
```

var(y) مقدار واریانس نمونه‌ای y را به ما می‌دهد

```
> var(y)
[1] 7.5
```

همچنین با استفاده از تابع cumsum(y) فراوانی تجمعی بردار y را می‌توان محاسبه کرد.

```
> y
[1]  4  5  6  7  8  9 10 11 12
> cumsum(y)
[1]  4  9 15 22 30 39 49 60 72
>
```

با استفاده از تابع prod(y) حاصلضرب درایه‌های بردار y را می‌توان محاسبه کرد

```
> prod(y)
[1] 79833600
```

توابع max(y) و min(y) مقادیر بیشینه و کمینه بردار y را پیدا می‌کنند

```
> min(y)
[1] 4
> max(y)
[1] 12
```

تابع range(y) دامنه‌ی بردار y را به صورت دو مولفه شامل کمینه و بیشینه y نمایش می‌دهد

```
> range(y)
[1]  4 12
```

اگر بردار y2 را به صورت زیر تعریف کنیم،

```
> y2<-c(4.5,5,8,5.6,9,11,6,8,4)
> y2
[1]  4.5  5.0  8.0  5.6  9.0 11.0  6.0  8.0  4.0
```

آنگاه به کمک تابع sort می‌توان عناصر بردار y2 را به صورت صعودی مرتب کرد

```
> sort(y2)
[1]  4.0  4.5  5.0  5.6  6.0  8.0  8.0  9.0 11.0
```

با آوردن آرگومان decreasing و قرار دادن مقدار TRUE به آن می‌توان بـردار را به صورت نزولی مرتب کرد

```
> sort(y2, decreasing=TRUE)
[1] 11.0  9.0  8.1  8.0  6.0  5.6  5.0  4.5  4.0
```

همچنین می‌توان از تابع زیر برای مرتب کردن نزولی عناصر بردار y2 استفاده کرد

```
> rev(sort(y2))
[1] 11.0  9.0  8.1  8.0  6.0  5.6  5.0  4.5  4.0
```

علاوه بر این می‌توان با روش زیر عناصر بردار y2 را به صورت نزولی مرتب کرد.

```
> -sort(-y2)
[1] 11.0  9.0  8.1  8.0  6.0  5.6  5.0  4.5  4.0
```

تابع cor(y,y2) همسبتگی بین دو بردار y و y2 را محاسبه می‌کند

```
> cor(y,y2)
[1] 0.163841
```

تابع rank(y2) رتبه‌ی عناصر موجود در بردار y2 را به ما می‌دهد

```
> y2
[1]  4.5  5.0  8.1  5.6  9.0 11.0  6.0  8.0  4.0
> rank(y2)
[1] 2 3 7 4 8 9 5 6 1
```

length(y) طول و تعداد عناصر بردار y را محاسبه می‌کند

```
> length(y)
[1] 9
> length(y2)
[1] 9
```

۲-۱-۳ ایجاد نمونه‌ی تصادفی

در نرم‌افزار R با استفاده از تابع sample می‌توان یک نمونه تصادفی از درایه‌های یک بردار ایجاد کرد. به عنوان مثال در زیر از اعداد طبیعی ۱ تا ۱۸ یک نمونه تصادفی به اندازه‌ی ۶ ایجاد می‌کنیم.

```
> t<-1:18
> sample(t,5)
[1] 12 18  4  7  8
```

همچنین این تابع یک آرگومان با نام replace را می‌پذیرد که مقدار آن مشخص کننده‌ی ایجاد نمونه با جایگذاری (T) و یا بدون جایگذاری (F) می‌باشد.

```
> sample(t,8, replace=T)
[1] 17 11  6  7  3  8  8  2
> sample(t,8, replace=F)
[1]  5 11 17  1 10  7 12 16
```

همچنین این تابع یک آرگومان با نام prob را می‌پذیرد که یک بردار از احتمالات را که طول آن با طول بردار برابر می‌باشد را می‌پذیرد و برای نمونه‌گیری با احتمالات نابرابر به کار می‌رود به عنوان مثال فرض کنید از بردار X زیر می‌خواهیم یک نمونه باجایگذاری ۱۰ تایی ولی با احتمالات نابرابر برای اعضای X اخذ کنیم. بردار احتمال متناسب با هر عضو بردار X را با بردار p نمایش داده‌ایم. همانطور که مشاهده می‌نمایید ۳ دارای احتمال بیشتری است و در نمونه هم به وضوح مشهود است.

```
> X<-c(2,3,1,5)
> p<-c(.1,.7,.1,.1)
> sample(X,10,prob=p,replace=T)
 [1] 3 3 3 3 3 3 1 5 3 3
```

۲-۱-۴ کارکردن با بردارها

y[j] عنصر j ام بردار y را به ما می‌دهد و به کمک y[c(i,j,k)] عناصر i ، j و k ام بردار استخراج می‌شوند.

```
> y
[1]  4  5  6  7  8  9 10 11 12
> y[5]
[1] 8
> y[c(2,5)]
[1] 5 8
```

با استفاده از عملگر تخصیص می‌توان مقدار عنصر i ام بردار y را تغییر داد. در زیر درایه‌ی دوم بردار y به مقدار ۴- تغییر داده شده است.

```
> y
[1]  4  5  6  7  8  9 10 11 12
> y[2]<- -4
> y
[1]  4 -4  6  7  8  9 10 11 12
```

انتخاب شرطی

می‌توان درایه‌های یک بردار را به صورت شرطی انتخاب کرد، مثلا همه‌ی عناصر بردار y را که بزرگتر یا مساوی با ۸ هستند انتخاب و آنها را در بردار q قرار داد. برای این کار از روش زیر استفاده می‌کنیم

```
> y
[1]  4 -4  6  7  8  9 10 11 12
> q<-y[y>=8]
> q
[1]  8  9 10 11 12
```

حذف کردن درایه‌هایی که مقدار آنها برابر با ۷ می‌باشد:

```
> y[y!=7]
[1]  4 -4  6  8  9 10 11 12
```

درایه‌هایی که بزرگتر از ۶/۵ و کوچکتر از ۸/۱ می‌باشند

```
> y[y>6.5 & y<8.1]
[1] 7 8
```

مجموع شرطی

فرض کنید می‌خواهیم برای بردار y مجموع عناصری را که بزرگتر یا مساوی با ۸ هستند محاسبه کنیم، یکی از روش‌ها انتخاب عناصر مورد نظر و تخصیص آنها به یک بردار مانند q و سپس استفاده از تابع sum(q) می‌باشد.

```
> y
[1]  4 -4  6  7  8  9 10 11 12
> q<-y[y>=8]
> q
[1]  8  9 10 11 12
```

یا اینکه می‌توان شرط مورد نظر را به عنوان آرگومان تابع sum آورد

```
> sum(y[y>=8])
[1] 50
```

شرط‌های منطقی و یک کاربرد

برای بردار دلخواه y2 ما می‌خواهیم به صورت منطقی عناصر آن را با عدد ۸/۳ مقایسه کنیم و ببینم درایه‌های آن بزرگتر هستند یا کوچکتر. برای این کار از روش زیر استفاده می‌کنیم

```
> y2>=8.3
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
```

مقادیر TRUE متناظر با درایه‌هایی هستند که در شرط ما (در اینجا بزرگتر بودن از ۸/۳) صدق می‌کنند. یکی از کاربردهای جالب این روش در حالتی که با بردارهای با اندازه‌ی بزرگ روبه‌رو هستیم شمارش تعداد درایه‌هایی است که در شرط ما صدق می‌کنند، به عبارتی چه تعداد از درایه‌ها از عدد z بزرگتر هستند،

چه تعداد از آنها داده‌ی گمشده هستند و یا مقدار آنها برابر NA می‌باشد و ...

به این منظور از تابع sum و به صورت زیر استفاده می‌کنیم.

```
> sum(y2>=8.3)
[1] 2
```

توجه داشته باشید که تابع sum(y2>=8.3) تنها تعداد درایه‌های بزرگتر از ۸/۳ را می‌شمارد، اما تابع sum(y2[y2>=8.3]) مجموع درایه‌های بزرگتر از ۸/۳ را محاسبه می‌کند.

حال فرض کنید برای مجموعه‌ای از داده‌ها می‌خواهیم آنها را به داده‌های صفر و یک تبدیل کنیم، مثلا داده‌های بزرگتر از عدد k را مقدار ۱ و مقادیر کمتر از k را صفر قرار دهیم. بدین منظور به صورت زیر عمل می‌کنیم. برای یک بردار دلخواه y با استفاده از دستور مقایسه منطقی y>7 با توجه به صدق کردن این شرط برای هر درایه‌ی آن بردار، برنامه R مقادیر FALSE یا TRUE را برمی‌گرداند.

```
> y
 [1]  4 -4  9  7  8  9 10  3 12  5
> y>7
 [1] FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
```

یکی از ویژگی‌های جالب برنامه‌ی R تبدیل مقادیر TRUE و FALSE به مقادیر ۱ و ۰ است به هنگامی‌که آنها را در یک رابطه‌ی ریاضی و یا به عنوان ورودی یک تابع آماری مانند sum() مورد استفاده قرار می‌دهیم. اکنون تنها با یک ضرب ساده می‌توان مقادیر بازگردانده شده را به عناصر ۱ و ۰ تبدیل کرد

```
> 1*(y>7)
 [1] 0 0 1 0 1 1 1 0 1 0
```

برای هر بردار دلخواه و هر نوع شرط منطقی دیگری به سادگی می‌توان این کار را انجام داد.

چک کردن یک شرط برای همه‌ی عناصر یک بردار

اگر بخواهیم بررسی کنیم همه‌ی عناصر یک بردار در یک شرط خاص صدق می‌کنند از دستور all(condition) استفاده می‌کنیم و برای بررسی اینکه حداقل یکی از عناصر بردار مورد نظر در یک شرط خاص پیروی می‌کنند از دستور any(condition) استفاده می‌کنیم.

برای بردار دلخواه y ابتدا چک می‌کنیم که آیا همه‌ی عناصر آن از ۱ بزرگتر هستند و سپس با استفاده از any بررسی می‌کنیم آیا در بردار y درایه‌ای با مقدار کوچکتر یا مساوی با ۱ وجود دارد؟ نتایج به صورت پاسخ‌های TRUE و FALSE برگردانده می‌شوند.

```
> y
[1]  4 -4  6  7  8  9 10 11 12
> all(y>1)
[1] FALSE
> any(y<=1)
[1] TRUE
```

یک دستور پرکاربرد در شروط منطقی، دستور which است که نشان دهنده شماره عضوی است که در شرط صدق می‌کند. مثلا میخواهیم ببینیم کدامیک از اعضای بردار y فوق در شرط کوچکتر یا مساوی یک صدق می‌کنند.

```
> which(y<=1)
[1] 2
```

این دستور نشان داد که عضو دوم بردار y در این شرط صدق می‌کند. در صورتی که بخواهیم آن عضو را نشان دهیم می‌توان به صورت زیر عمل کرد

```
> y[which(y<=1)]
[1] -4
```

کمینه و بیشینه‌ی موازی چندبردار هم اندازه

در برنامه‌ی R این امکان وجود دارد که برای چند بردار با اندازه و تعداد درایه‌های برابر، برای هر جایگاه درایه‌ای مقدار کمینه و بیشینه را پیدا کرد. این کار با استفاده از تابع‌های pmax وpmin امکان پذیر می‌باشد. این توابع یک بردار هم اندازه با بردارهای اولیه را به عنوان پاسخ برمی‌گردانند. که درایه‌ی اول این بردار، برابر است با بیشینه (یا کمینه)ی درایه‌های اول بردارهای ورودی، درایه‌ی دوم آن برابر است با بیشینه (یا کمینه)ی درایه‌های دوم بردارهای ورودی و ... . به بردار پاسخ، بردار بیشینه (یا کمینه)ی موازی گفته می‌شود.

به عنوان مثال در زیر برای سه بردار دلخواه a ، b و c با اندازه‌ی ۴ که به صورت زیر تعریف شده‌اند، بردارهای بیشینه موازی و کمینه موازی را محاسبه کرده‌ایم.

```
> a<- c(-2,3,0,2)
> b<- 0:3
> c<-c(2,-1,3,3)
> pmin(a,b,c)
[1] -2 -1  0  2
> pmax(a,b,c)
[1] 2 3 3 3
```

۲-۲ ماتریس‌ها

در محیط برنامه‌ی R برای تعریف و ایجاد ماتریس از دستور matrix استفاه می‌شود. یکی از روش‌های ایجاد یک ماتریس،تبدیل یک بردار به ماتریس است. توجه داشته باشید که همواره یک بردار با اندازه‌ی k ، یک ماتریس با ۱ سطر و k ستون می‌باشد.

استفاده از دستور matrix

در این روش ابتدا یک بردار را تعریف کرده و سپس با استفاده از دستور matrix آنرا به یک ماتریس تبدیل می‌کنیم.

```
> vect<-c(2,1,0,0,0,1,0,2,2)
> matrix(vect,nrow=3)
     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    1    0    2
[3,]    0    1    2
```

در این حالت R داده‌ها را به صورت ستونی در ماتریس قرار می‌دهد، ابتدا ستون اول و سپس ستون‌های بعدی را ایجاد می‌کند. چنانچه بخواهیم درایه‌های ماتریس ایجاد شده را سطری پر کنیم، از دستور زیر استفاده می‌کنیم

```
> matrix(vect,nrow=3, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    2    1    0
[2,]    0    0    1
[3,]    0    2    2
```

به هنگام تعریف یک ماتریس، چنانچه اندازه‌ی بردار داده‌ها مضربی از مقدار nrow که تعیین کننده‌ی تعداد سطرهای ماتریس است، نباشد، پس از پر کردن همه‌ی درایه‌های ماتریس با عناصر بردار در صورت وجود درایه‌ی خالی در ماتریس، دوباره از عنصر اول بردار شروع کرده این درایه‌های خالی را پر می‌کند. در این حالت برنامه ماتریس مورد نظر ما را ایجاد می‌کند ولی یک پیام نیز با این مضمون که تعداد درایه‌های بردار مضربی از تعداد سطرهای ماتریس نیست، به ما می‌دهد

```
> matrix(vect,nrow=4, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    2    1    0
[2,]    0    0    1
[3,]    0    2    2
[4,]    2    1    0
Warning message:
In matrix(vect, nrow = 4, byrow = TRUE) :
  data length [9] is not a sub-multiple or multiple
```

در تعریف یک ماتریس به جای nrow می‌توان از ncol که مشخص کننده تعداد ستون‌های ماتریس است، استفاده کرد.

```
> matrix(vect,ncol=3, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    2    1    0
[2,]    0    0    1
[3,]    0    2    2
```

با استفاده از دستور dim می‌توان تعداد سطر و ستون‌های یک ماتریس را استخراج کرد.

```
> vector<-c(1,2,3,4,4,3,2,1)
> X<-matrix(vector,nrow=2, byrow=TRUE)
> X
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    4    3    2    1
> dim(X)
[1] 2 4
```

ماتریس داده شده یک ماتریس با ۲ سطر و ۴ ستون می‌باشد، حال می‌خواهیم ماتریس X را به یک ماتریس با ۴ سطر و ۲ ستون استفاده کنیم، بدین منظور به صورت زیر عمل می‌کنیم

```
> dim(X)<-c(4,2)
> X
     [,1] [,2]
[1,]    1    3
[2,]    4    2
[3,]    2    4
[4,]    3    1
```

این روش ابتدا ماتریس اولیه‌ی X را با خواندن داده‌های آن به صورت ستونی به یک بردار تبدیل می‌کند سپس بردار ایجاد شده را به یک ماتریس با ۴ سطر و ۲ ستون تبدیل می‌کند. در ایجاد یک ماتریس، برنامه R به صورت پیش فرض ابتدا ستون اول و سپس ستون‌های بعدی را ایجاد می‌کند، مگر این‌که در دستور ایجاد ماتریس از آرگومان byrow=TRUE استفاده شده باشد.

روش تعریف بعد

یکی دیگر از روش‌های ایجاد ماتریس استفاده از روش تعریف بعد برای یک بـردار می‌باشد. در این حالت از روش تخصیص بعد به صورت زیر استفاده می‌کنیم. کـه در آن مقدار اول تعداد سطرها و مقدار دوم تعداد ستون‌ها را نمایش می‌دهد.

```
> data<-c(0,1,2,3,4,4,3,2,1,0)
> dim(data)<-c(2,5)
> data
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    2    4    3    1
[2,]    1    3    4    2    0
```

تعریف ماتریس قطری

در نرم افزار R با استفاده از دسـتور diag(p) می‌تـوان یـک مـاتریس همـانی بـا مرتبه‌ی p ایجاد کرد.

```
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

چنانچه بخواهیم یک ماتریس قطری با درایه‌های یکسـان بـا مرتبـه‌ی p اسـتفاده کنیم از یکی از روش‌های

```
> diag(k,p)
```

```
> k*diag(p)
```

استفاده می‌کنیم.

```
> diag(3,4)
     [,1] [,2] [,3] [,4]
[1,]    3    0    0    0
[2,]    0    3    0    0
[3,]    0    0    3    0
[4,]    0    0    0    3
> 2*diag(3)
     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    2    0
[3,]    0    0    2
```

اگر بخواهیم یک ماتریس قطری با عناصر روی قطر اصلی غیر یکسان ایجاد کنیم، به روش زیر عمل می‌نماییم.

```
> diag(3:-2)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    3    0    0    0    0    0
[2,]    0    2    0    0    0    0
[3,]    0    0    1    0    0    0
[4,]    0    0    0    0    0    0
[5,]    0    0    0    0   -1    0
[6,]    0    0    0    0    0   -2
> diag(c(2,3,0,6))
     [,1] [,2] [,3] [,4]
[1,]    2    0    0    0
[2,]    0    3    0    0
[3,]    0    0    0    0
[4,]    0    0    0    6
```

در واقع از همان دستور diag استفاده کرده‌ایم که آرگومان ورودی آن یک بردار شامل درایه‌های روی قطر اصلی می‌باشد.

۲-۲-۱ کار کردن با ماتریس‌ها

برای یک ماتریس داده شده، می‌توان به انتخاب یک ستون و یا سطر خاص آن پرداخت. درایه‌ی مربوط به سطر i و ستون jام یک ماتریس را به صورت زیر می‌توان انتخاب کرد.

```
> matrixname[i,j]
```

به عنوان مثال درایه‌ای سطر دوم و ستون چهارم ماتریس data را به صورت زیر انتخاب می‌کنیم.

```
> data[2,4]
[1] 2
```

همچنین مقدار این درایه را به صورت زیر می‌توان تغییر داد:

```
> data[2,4]<-100
> data
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    2    4    3    1
[2,]    1    3    4  100    0
```

چنانچه بخواهیم تنها سطر i ام یا ستون j ام یک ماتریس را انتخاب کنیم، از روش زیر استفاده می‌کنیم

```
> matrixname[i,]
```

```
> matrixname[,j]
```

به عنوان مثال برای ماتریس data سطر دوم و ستون سوم آن‌را به صورت زیر انتخاب می‌کنیم

```
> data[2,]
[1]   1   3   4 100   0
> data[,3]
[1] 4 4
```

چنانچه بخواهیم، چند سطر یا ستون را با هم انتخاب کنیم، به صورت زیر عمل می‌کنیم. در واقع یک بردار شامل شماره‌ی سطرها (یا ستون‌ها)ی مورد نظر را به جای شماره‌ی سطر در دستور فوق قرار می‌دهیم. در بردار data ، ستون‌های اول، دوم و چهارم را به صورت زیر انتخاب می‌کنیم

```
> data
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    2    4    3    1
[2,]    1    3    4  100    0
> data[,c(1,2,5)]
     [,1] [,2] [,3]
[1,]    0    2    1
[2,]    1    3    0
```

با استفاده از روش زیر می‌توانیم عناصر روی قطر اصلی یک ماتریس مربعی را انتخاب و یا تغییر دهیم.

```
> B
     [,1] [,2] [,3]
[1,]    3    0    1
[2,]    2   -1    1
[3,]    1   -2    1
> diag(B)
[1]  3 -1  1
> diag(B)<-c(10,10,10)
> B
     [,1] [,2] [,3]
[1,]   10    0    1
[2,]    2   10    1
[3,]    1   -2   10
```

نامگذاری سطرها و ستون‌های یک ماتریس

با استفاده از تابع‌های rownames و colnames می‌توان سطرها و ستون‌های یک ماتریس را نام گذاری کرد.

```
> rownames(matrix name)<-c("names")
```

```
> colnames(matrix name)<-c("names")
```

به عنوان مثال سطرهای ماتریس data را به صورت زیر نامگذاری می‌کنیم.

```
> rownames(data)<-c("aval", "dovom")
> data
      [,1] [,2] [,3] [,4] [,5]
aval     0    2    4    3    1
dovom    1    3    4  100    0
```

ترانهاده‌ی یک ماتریس

برای تعریف ترانهاده‌ی یک ماتریس از دستور t(matrix name) استفاده می‌کنیم.

```
> data
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    2    4    3    1
[2,]    1    3    4    2    0
> t(data)
     [,1] [,2]
[1,]    0    1
[2,]    2    3
[3,]    4    4
[4,]    3    2
[5,]    1    0
```

عملیات سطری و ستونی

برای یک ماتریس، با استفاده از تابع‌های rowSums و colSums می‌توان به محاسبه‌ی مجموع سطری و ستونی درایه‌های بردار پرداخت. هم چنین تابع‌های rowMeans و colMeans میانگین سطری و ستونی ماتریس را محاسبه می‌کنند.

```
> data
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    2    4    3    1
[2,]    1    3    4    2    0
> rowSums(data)
[1] 10 10
> colSums(data)
[1] 1 5 8 5 1
```

همچنین میانگین درایه‌های روی سطرها و ستون‌ها به صورت زیر محاسبه می‌شوند

```
> rowMeans(data)
[1] 2 2
> colMeans(data)
[1] 0.5 2.5 4.0 2.5 0.5
```

اضافه کردن سطر و ستون به ماتریس

برای اضافه کردن سطر جدید به یک ماتریس از دستور rbind و به منظور اضـافه کردن ستون جدید از دستور cbind و به صورت زیر استفاده می‌کنیم.

```
> rbind(matrix name,new row)
```

به عنوان مثال به ماتریس data یک ستون جدید با درایه‌های ۰ اضافه می‌کنیم

```
> newdata<-cbind(data, rep(0,2))
> newdata
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    2    4    3    1    0
[2,]    1    3    4    2    0    0
```

و سپس به ماتریس newdata یک سطر جدید که مجموع ستونی داده‌ها می‌باشد اضافه می‌کنیم

```
> data2<-rbind(newdata, colSums(newdata))
> data2
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    2    4    3    1    0
[2,]    1    3    4    2    0    0
[3,]    1    5    8    5    1    0
```

به ماتریس داده‌شده یک ستون که مجموع سطری داده‌هاست به صورت زیر اضافه می‌کنیم

```
> datat
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    0    2    4    3    1    0   10
[2,]    1    3    4    2    0    0   10
[3,]    1    5    8    5    1    0   20
```

حال سطر سوم و ستون هفتم ماتریس فوق را بـا عنـوان colsums و rowsums نامگذاری می‌کنیم

```
> colnames(datat)<-c(1:6,"rowsums")
> rownames(datat)<-c(1,2,"colsume")
> datat
        1 2 3 4 5 6 rowsums
1       0 2 4 3 1 0      10
2       1 3 4 2 0 0      10
colsume 1 5 8 5 1 0      20
```

۲-۲-۲ عملیات ریاضی ماتریس‌ها

بـرای جمـع و تفریـق دو مـاتریس در محـیط R از عملگرهـای + و - اسـتفاده می‌کنیم که در این حالـت ماتریس‌هـا بایـد دارای بعـد یکسـان باشـند. بـرای دو ماتریس دلخواه A و B داریم

```
> A<-diag(2,3)
> A
     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    2    0
[3,]    0    0    2
> B<-matrix(c(3:-2,1,1,1), nrow=3)
> B
     [,1] [,2] [,3]
[1,]    3    0    1
[2,]    2   -1    1
[3,]    1   -2    1
> A+B
     [,1] [,2] [,3]
[1,]    5    0    1
[2,]    2    1    1
[3,]    1   -2    3
```

در محیط R برای ضرب دو ماتریس از عملگر %*% استفاده می‌شـود. همچنـین با استفاده از عملگر * می‌توان حاصلضرب درایه‌های متناظر در دو ماتریس با بعـد

یکسان را محاسبه کرد. برای دو ماتریس A و B با توجه به دو عملگر داده شـده نتایج زیر را خواهیم داشت.

```
> A*B
     [,1] [,2] [,3]
[1,]    6    0    0
[2,]    0   -2    0
[3,]    0    0    2
> A%*%B
     [,1] [,2] [,3]
[1,]    6    0    2
[2,]    4   -2    2
[3,]    2   -4    2
```

اگر ماتریس C را به این صورت تعریف کنیم که با ماتریس A دارای بعـد یکسـان نباشد و

```
> C<-matrix(c(1,2,0,3,4,1), nrow=2)
> C
     [,1] [,2] [,3]
[1,]    1    0    4
[2,]    2    3    1
```

آنگاه با استفاده از عملگرهای فوق نتایج زیر را در مورد حاصلضرب دو ماتریس A و C نتایج زیر را خواهیم داشت

```
> A*C
Error in A * C : non-conformable arrays
> C*A
Error in C * A : non-conformable arrays
> A%*%C
Error in A %*% C : non-conformable arguments
> C%*%A
     [,1] [,2] [,3]
[1,]    2    0    8
[2,]    4    6    2
```

از آنجایی کـه ماتریس‌هـای A و C بعـدهای یکسـانی ندارنـد، تنهـا حاصلضـرب ماتریسی CA قابل محاسبه می‌باشد.

محاسبه‌ی دترمینان

در محیط برنامه R دترمینان یک ماتریس مربعی را با استفاده از دستور det می‌توان محاسبه کرد

```
> B
     [,1] [,2] [,3]
[1,]   10    0    1
[2,]    2   10    1
[3,]    1   -2   10
> det(B)
[1] 1006
```

حال برای ماتریس B سطر سوم آن‌را دو برابر کرده و دترمینان آن را محاسبه می‌کنیم

```
> B[,3]<-2*B[,3]
> B
     [,1] [,2] [,3]
[1,]   10    0    2
[2,]    2   10    2
[3,]    1   -2   20
> det(B)
[1] 2012
```

می‌دانیم که اگر یک سطر یا ستون یک ماتریس را در عدد k ضرب کنیم، آنگاه دترمینان آن نیز k برابر می‌شود.

معکوس یک ماتریس

در محیط برنامه‌ی R با استفاده از تابع ginv می‌توان معکوس یک ماتریس معکوس‌پذیر را محاسبه کرد. این تابع در بسته‌ی MASS گنجانده شده است که این بسته معمولا به صورت پیش فرض بر روی برنامه‌ی R نصب شده است، اما برای استفاده از تابع‌های آن باید ابتدا آن‌را فراخوانی کرد.

```
> B
     [,1] [,2] [,3]
[1,]   10    0    2
[2,]    2   10    2
[3,]    1   -2   20
> library(MASS)
> ginv(B)
             [,1]         [,2]         [,3]
[1,]  0.10139165 -0.001988072 -0.009940358
[2,] -0.01888668  0.098409543 -0.007952286
[3,] -0.00695825  0.009940358  0.049701789
```

معکوس معکوس یک ماتریس برابر با همان ماتریس خواهد بود:

```
> D<-ginv(B)
> ginv(D)
     [,1]          [,2] [,3]
[1,]   10  3.015437e-16    2
[2,]    2  1.000000e+01    2
[3,]    1 -2.000000e+00   20
```

مقادیر ویژه و بردارهای ویژه یک ماتریس

برای ماتریس L مقادیر و بردارهای ویژه با استفاده از توسط دستور eigen(L) محاسبه می‌شوند.

```
> x<-matrix(3:11, nrow=3)
> eigen(x)
$values
[1]  2.182475e+01 -8.247517e-01 -5.549124e-16

$vectors
           [,1]       [,2]       [,3]
[1,] -0.4948814 -0.8543727  0.4082483
[2,] -0.5737479 -0.1842330 -0.8164966
[3,] -0.6526145  0.4859068  0.4082483
```

۲-۳ آرایه‌ها

هر آرایه[۶] عبارت است از یک شیء که دارای ویژگی بعدپذیری می‌باشد، در واقع با دادن ویژگی بعد به یک شیء می‌توان آن را به یک آرایه تبدیل کرد. بـه آرایـه‌ی یک بعدی، بردار و به آرایه‌ی دو بعدی ماتریس گفته می‌شود. برنامـه‌ی R بـه مـا اجازه می‌دهد که آرایه‌هایی با بعدهای بیشتر از دو نیز داشته باشیم.

در زیر ابتدا یک بردار (آرایه‌ی یک بعدی) شامل دنباله‌ی اعداد طبیعی از ۱ تا ۲۴ را تشکیل می‌دهیم

```
> a<-1:24
> a
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
[16] 16 17 18 19 20 21 22 23 24
```

حال با تعریف بعد این بردار را به یک ماتریس ۴*۶ تبدیل می‌کنیم

```
> dim(a)<-c(6,4)
> a
     [,1] [,2] [,3] [,4]
[1,]    1    7   13   19
[2,]    2    8   14   20
[3,]    3    9   15   21
[4,]    4   10   16   22
[5,]    5   11   17   23
[6,]    6   12   18   24
```

اکنون a یک آرایه با دو بعد می‌باشد. حال a را به یک آرایه‌ی سـه بعـدی تبـدیل می‌کنیم، برای این‌کار از تابع dim استفاده می‌کنیم، به گونه‌ای که بعـد اول آن ۳ سطح، بعد دوم ۴ سطح و بعد سوم آن دارای ۲ سطح باشد

[۶] Array

```
> dim(a)<-c(3,4,2)
> a
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

به مانند آنچه که در مورد ماتریس‌ها داشتم، انتخاب یک درایه، سطر و یـا سـتون خاص در آرایه‌های با بعد بیشتر از ۲ نیز می‌توان به اننتخـاب یـک درایـه خـاص پرداخت.

در آرایه‌ی a جدول دوم، که درایه‌های مربوط به سطح دوم از بعد سوم را در خود جای داده است را به شکل زیر انتخاب می‌کنیم

```
> a[,,2]
     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

اگر در درایه‌ی a بخواهیم اعداد ۴،۵،۶،۱۶،۱۷ و۱۸ را انتخاب کنیم، همانگونه که مشاهده می‌شود، این اعداد بر روی ستون دوم (در واقع سطح دو از بعد دوم) واقع شده‌اند

```
> a[,2,]
     [,1] [,2]
[1,]    4   16
[2,]    5   17
[3,]    6   18
```

همانگونه که مشاهده می‌شود دو ستون انتخاب شده در کنار هم قرار گرفته‌اند، به کمک آرگومان drop=F می‌توان درایه‌های انتخاب شده را در همان نمایش بعدی آنها نشان داد

```
> a[,2,, drop=F]
, , 1

     [,1]
[1,]    4
[2,]    5
[3,]    6

, , 2

     [,1]
[1,]   16
[2,]   17
[3,]   18
```

برای انتخاب یک درایه‌ی خاص به عنوان مثال در اینجا عدد ۱۰ به طریق زیر عمل می‌کنیم

```
> a[1,4,1]
[1] 10
```

که در a[1,4,1] عدد اول نشان‌دهنده‌ی سطر جایگاه عدد ۱۰، ۴ بیانگر ستون آن و سپس ۱ نشانگر شماره سطح آن در بعد سوم است.

همچنین یک آرایه را می‌توان با استفاده از دستور array و به صورت زیر نیز ایجاد کرد

```
> array(data,dim, dim names)
```

به عنوان مثال اعداد ۱ تا ۳۶ را در یک آرایه‌ی چهار بعدی که بعدهای آن ۲،۳،۳،۲ سطح دارند، قرار می‌دهیم

```
> array(rep(1:36),c(2,3,3,2))
, , 1, 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2, 1

     [,1] [,2] [,3]
[1,]    7    9   11
```

محاسبات حاشیه‌ای در آرایه‌ها

در برنامه‌ی R یک تابع با نام apply امکان انجام محاسبات حاشیه‌ای در آرایه‌های را به ما می‌دهد، مثلا بخواهیم برای بعد اول (همان سطرها) میانگین را محاسبه کنیم، برای بعد دوم (یا همان ستون‌ها) جمع روی ستون را به دست آوریم و یا اینکه تابع یا دستور f را بر روی بعد j ام اعمال کنیم.

فرم کلی استفاده از این دستور به صورت زیر می‌باشد.

```
> apply(name,j,f)
```

حال برای آرایه‌ی a میانگین همه‌ی بعدها را به صورت زیر محاسبه می‌کنیم

```
> apply(a,2,mean)
[1]  8 11 14 17
> apply(a,3,mean)
[1]  6.5 18.5
> apply(a,1,mean)
[1] 11.5 12.5 13.5
```

در قسمت دوم مقدار گزارش شده‌ی ۱۸/۵ میانگین حاشیه‌ای جدول دوم می‌باشد. اعداد گزارش شده در پاسخ به apply(a,1,mean) میانگین اعداد بر روی سطرهای اول، دوم و سوم را نمایش می‌دهد، به طوری‌که مقدار ۱۱/۵ میانگین عددهای ۱،۴،۷،۱۰،۱۳،۱۶،۱۹،۲۲ می‌باشد.

با استفاده از دستور apply برای ماتریس دلخواه V، واریانس ستون‌ها را می‌توان به صورت زیر محاسبه کرد

```
> V=matrix(c(1,3,4,2,1,7,2,3,5),ncol=3, byrow=T)
> V
     [,1] [,2] [,3]
[1,]    1    3    4
[2,]    2    1    7
[3,]    2    3    5
> apply(V,2,var)
[1] 0.3333333 1.3333333 2.3333333
```

در یک ماتریس (آرایه) ستون‌ها بیانگر بعد دوم می‌باشند و همچنین تابع var واریانس نمونه‌ای را محاسبه می‌کند.

لازم به ذکر است که برای ضرب خارجی دو آرایه از عملگر %o% استفاده می‌شود.

برنامه نویسی در R

یکی از مزیت‌های استفاده از نرم افزار R در انجام محاسبات آماری این است که کاربر در صورتی که تابع یا بسته‌ای مناسب جهت انجام دادن کار مورد نظر خود نیافت، می‌توان خود دست به کار شده و برنامه یا تابع مورد نظر خود را کدنویسی کند. در برنامه R شکل کلی یک تابع به صورت زیر می‌باشد

```
> functionname<-function(arguments){
+ function body }
```

در دستورالعمل فوق، نام تابع مورد نظر خود را به جای functionname ، شیء حاوی ورودی‌های برنامه را به جای arguments و بدنه‌ی تابع، یعنی دستورات مربوط به تابع و محاسباتی که باید انجام دهد را در قسمت function body می‌نویسیم. سپس تابع را با استفاده از دستور

```
> functionname(arguments.value)
```

فراخوانی می‌کنیم.

مثلا در زیر یک تابع با اسم myfunc ایجاد کرد و متغیر x را به عنوان آرگومان ورودی آن اختیار کرده‌ایم، در قسمت function body از برنامه خواسته‌ایم مقدار x-4 و x+4 را برای ما در یک بردار چاپ کند.

```
> myfunc<-function(x) {
+ print(c(x-4,x+4)) }
> myfunc(5)
[1] 1 9
> myfunc(7)
[1]  3 11
```

برنامه R این امکان را به ما می‌دهد که در قسمت function body از تابع‌های موجود در برنامه یا سایر بسته‌ها (به شرطی که بسته مورد نظر را فراخوانی کرده باشیم) استفاده کنیم، همچنین لازم به ذکر است که اگر یک تابع در برنامه R با نام تابعی که ما ایجاد کرده‌ایم وجود داشته باشد، تابع نوشته شده توسط ما

جایگزین تابع پیشین می‌شود. با توجه به این نکته بهتر است همـواره در انتخـاب نام تابع خود کمی دقت کرده و آن را به گونه‌ای تعریف کنیم که پیشتر یک تـابع با این اسم در محیط برنامه موجود نباشد.

به عنوان مثالی دیگر فرض کنید بخواهیم ریشه های معادلـه درجـه دوم $ax^2 + bx + c = 0$ را با استفاده از یک تابع به نـام myroot بـه صـورت زیـر محاسـبه کنیم:

```
> myroot<-function(a,b,c){
+ delta<-b^2-4*a*c
+ if (delta>0){
+ x1=(-b+sqrt(delta))/(2*a)
+ x2=(-b-sqrt(delta))/(2*a)}
+ return(c(x1,x2))
+ }
```

به عنوان کاربردی از تابع ساخته شده می توان به صورت زیر عمل کرد

```
> myroot(2,3,-1)
[1]  0.2807764 -1.7807764
```

این برنامه رای می توان به صورت زیر ویرایش و کامل نمود طوریکه برای دلتـای منفی و صفر نیز قابل اجرا باشد

```
> myroot<-function(a,b,c){
+ delta<-b^2-4*a*c
+ if (delta>0){
+ x1=(-b+sqrt(delta))/(2*a)
+ x2=(-b-sqrt(delta))/(2*a)}
+ else if (delta==0){
+ x1=-b/(2*a)
+ x2=-b/(2*a)}
+ else{ print("no root")}
+ return(c(x1,x2))
+ }
```

به هنگام نوشتن برنامه و یا ایجاد یک تابع همواره سعی شود که از اسامی معنادار

و مرتبط برای نام تابع ایجاد شده استفاده شود، مثلا استفاده از نام myprod برای تابعی که قرار است یک مجموع را برای ما محاسبه کند چندان مناسب نیست.

در صورتی‌که در حال نوشتن یک برنامه‌ی طولانی هستید، می‌توانید توضیحاتی را در لابه‌لای کدهای خود بگنجانید، در محیط برنامه R در صورتی‌که یک متن یا نوشته پس از # بیاید، برنامه R بی‌توجه به محتوای آن به سطر بعدی مراجعه می‌کند. این توضیحات در آینده می‌توانند شما را در درک سریع و بهتر از کدهای نوشته شده یاری کنند و یا اگر شخص دیگری کدهای شما را مورد استفاده قرار داد وی را در درک برنامه نوشته شده یاری دهند.

```
> myfunc(4)  #x-4 va x+4 ra mohasebe mikonad
[1] 0 8
```

یک تابع ممکن است در بدنه‌ی خود برای ما یک مقایسه انجام دهد و یا اینکه از یک عملگر یا دستور استفاده کند. همچنین بدنه‌ی تابع می‌تواند بسیار ساده و در حقیقت یک خط برنامه باشد و یا اینکه بسیار پیچیده باشد.

۴-۲ عملگرهای مقایسه‌ای و منطقی

در نوشتن بدنه‌ی یک تابع در بسیاری از موارد، از برنامه R خواسته می‌شود یک یا چند مقدار را با هم یا یک مقدار انتخابی مقایسه کند و در صورت برقرار بودن یا برقرار نبودن یک شرط، یک عبارت یا نتیجه را برگرداند. در برنامه‌ی R چندین عملگر منطقی و مقایسه‌ای وجود دارد که در ادامه به معرفی مختصر آنها می‌پردازیم

عملگرهای مقایسه‌ای

==	برابری
!=	نابرابری

<, >	کوچکتر و بزرگتر
=<, >=	کوچکتر و مساوی، بزرگتر و مساوی

عملگرهای منطقی

!	نه، برقرار نباشد
&	و
|	یا

۲-۵ ساختارهای کنترلی

در برنامه نویسی درمحیط R ، مانند همه‌ی زبان‌های برنامه نویسی دیگر، تعدادی دستور و قاعده به منظور کنترل جریان اجرای برنامه و محاسبات تعریف شده‌اند. این ساختارها را در حالت کلی می‌توان به دو دسته‌ی شرطی و تکرار تقسیم بندی کرد. ساختارهای شرطی در یک برنامه، جهت جریان آن را با بررسی برقراری یک یا چند شرط انتخاب می‌کنند. در ساختارهای کنترلی تکرار نیز در جریان یک برنامه بخشی از آن چندین مرتبه بازخوانی و اجرا خواهد شد. در صورت آشنایی با زبان‌های برنامه نویسی دیگر به ویژه زبان برنامه نویسی C درک و فهم ادامه این فصل بسیار ساده خواهد بود.

۲-۵-۱ ساختارهای کنترلی شرطی

یکی از رایج‌ترین روش‌های کنترل شرطی یک جریان استفاده از if و else می‌باشد. یک ساختار کنترل شرطی در محیط R و با استفاده از if به صورت زیر تعریف می‌شود.

if (شرط مورد نظر) {فرمان یکم} else {فرمان دوم}

که در صورت برقراری شرط مورد نظر، فرمان یکم و در صورت عدم برقراری شرط مورد نظر فرمان دوم اجرا می‌شود.

```
> x<-4
> if(x<=5) {
+  print(x)
+ }else { print(0)}
[1] 4
```

در ساختار کنترلی فوق، عبارت کنترل عبارت است از، برقراری شـرط کـوچکتر و مساوی ۵ بودن x ، که در صورت برقراری شرط فوق، مقدار x چاپ شود و در غیر این صورت مقدار ۰ را چاپ نماید.

به عنوان مثال برای مقدار x=۴ ، شرط فوق برقرار بوده، پس برنامه فرمان یکم را اجرا می‌کند.

یک ساختار کنترل شرطی دیگر در محیط R ، دستور ifelse می‌باشد و به صورت زیر تعریف می‌شود.

ifelse (عبارت شرط، فرمان یکم، فرمان دوم)

این ساختار نیز، با بررسی عبارت شرطی و در صورت برقرار بودن آن، فرمان یکـم و در غیر این صورت فرمان دوم را اجرا می‌کند.

```
> x<-3
> ifelse(x<5 | x>8, x, 0)
[1] 3
> x<-6
> ifelse(x<5 | x>8, x, 0)
[1] 0
```

تفاوت اساسی دستور ifelse با ساختار کنترلی if در این است کـه دسـتور ifelse شرط مورد نظر را برای دنباله‌ای از مقادیر، در حقیقت یک بردار، بررسی کرده و با توجه به شرط فوق فرمان یکم یا فرمان دوم را اجرا می‌کند

```
> x<-c(2,3,7,8,6,4)
> ifelse(x<5 | x>8, x, 0)
[1] 2 3 0 0 0 4
```

دستور if/else می تواند به صورت آشیانه ای نیز بکار گرفته شود کـه سـاختار آن به صورت زیر است

if (عبارت شرط){ فرمان مورد نظر }

else if (عبارت شرط){فرمان مورد نظر}

else if (عبارت شرط){فرمان مورد نظر}

else {فرمان مورد نظر}

مثالی از کاربرد این دستور می تواند به صورت زیر باشد که در آن از توزیع گسسته زیر یک نمونه به صورت تصادفی انتخاب شده است

x	1	2	3	4
P(X=x)	0.2	0.2	0.3	0.3

```
> u<-runif(1)
> if ( u<=0.2 ) {
+ x<-1
+ } else if ( u>0.2 & u<=0.4 ) {
+ x<-2
+ } else if ( u>0.4 & u<=0.7 ) {
+ x<-3
+ } else {
+ x<-4
+ }
```

۲-۵-۲ ساختارهای کنترلی تکرار

ساختارهای کنترلی تکرار به گونه‌ای تعریف شده‌اند، که در صورت برقـراری یـک شرط برای یک مقدار که به عنوان مقدار شمارنده شناخته میشود، یک فرمـان یـا مجموعه فرمان را چندین بار تکرار کند.

یکی از دستورهای کنترلی for می‌باشد که به صورت

for (متغیر شمارنده و عبارت کنترل)عبارت دستوری{

تعریف می‌شود. در اینگونه ساختارهای دستوری به ازای مقادیر متغیـر شـمارنده، عبارت دستوری اجرا می‌شود.

```
> for (i in c(2,4)) {print(rnorm(i))}
[1]  0.5445633 -0.4653242
[1]  0.1631263 -0.4694707  0.8371224 -0.7830546
```

در دستور فوق برای مقادیر ۲ و ۴ متغیر شمارنده‌ی i ، دنباله‌ای از اعداد با توزیـع نرمال استاندارد به اندازه‌ی i تولید کرده‌ایم.

در مثال زیر برای مقادیر i از ۱ تا ۷ ، به مقادیر i دو واحد اضافه کـرده و آن‌هـا را چاپ کرده‌ایم.

```
> for(i in 1:7)
+ { k<-i+2 ; print(k)}
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

با توجه به مثال‌های فوق، عبارت کنترل به صورت

بردار i in

تعریف می‌شود.

به عنوان مثالی دیگر دنباله اعداد فیبوناتچی را به صورت زیر در نظر بگیرید

0,1,1,2,3,5,8,13,21, ….

برنامه زیر ۱۰۰ جمله اول این دنباله را با داشتن دو جمله اول آن تولید می کند

```
> f<-rep(0,10)
> f[1]<-0
> f[2]<-1
> for(i in 3:10){
+ f[i]<-f[i-2]+f[i-1]
+ }
> f
```

همچنین حلقه for می تواند به صورت تودرتو نیز به کار گرفته شود به عنوان مثال برنامه زیر را در نظر بگیرید.

برنامه زیر حالت های مختلف تبدیل یک اسکناس هزار تومانی را به اسکناس های پنجاه، صد و دویست تومانی تولید می نماید بطوریکه از هر اسکناس حداقل یک مورد موجود باشد.

```
> for (i in seq(2,14,2)){
+ for( j in 1:7){
+ for (k in 1:4){
+ if (i*50+j*100+k*200==1000)
+ p<-c(i,j,k)
+ print(p)
+ }
+ }
+ }
+ }
```

خروجی این برنامه به صورت زیر است

```
[1] 2 1 4
[1] 2 3 3
[1] 2 5 2
[1] 2 7 1
[1] 4 2 3
[1] 4 4 2
[1] 4 6 1
[1] 6 1 3
[1] 6 3 2
[1] 6 5 1
[1] 8 2 2
[1] 8 4 1
[1] 10  1  2
[1] 10  3  1
[1] 12  2  1
[1] 14  1  1
```

این برنامه با برداشتن این شرط که از هر اسکناس یک مورد موجود باشد به صورت زیر می باشد

```
> for (i in 0:20){
+ for( j in 0:10){
+ for (k in 1:5){
+ if (i*50+j*100+k*200==1000){
+ p<-c(i,j,k)
+ print(p)
+ }
+ }
+ }
+ }
```

یکی دیگر از ساختارهای کنترلی توسط while ایجاد می‌شود، این ساختار بر خلاف ساختار for که با توجه به یک متغیره شمارنده تعداد تکرارها را تعیین می‌کند، یک حلقه (عبارت دستوری) را تا زمان عدم برقراری یک شرط تکرار می‌کند و به صورت زیر تعریف می‌شود:

while (عبارت شرطی){عبارت دستوری}

به عنوان مثال، در زیر با شروع از ۰ تا زمانی‌که j<۱۰ باشد، برنامه مقدار j+۰,۵ را

برگردانده و دو واحد به j اضافه کرده و سپس برقراری شـرط را چـک کـرده و در صورت عدم برقراری آن از حلقه خارج می‌شود.

```
> j<-0
> while(j<10){
+   k<-j+0.5
+ print(k)
+ j<-j+2 }
[1] 0.5
[1] 2.5
[1] 4.5
[1] 6.5
[1] 8.5
```

به عنوان مثالی دیگر فرض کنید بخواهیم جملات دنباله فیبوناتچی را تا زمانی که کمتر از ۱۰۰ است بنویسیم، برنامه این کار به صورت زیر است

```
> n<-2
> f<-c(0,1)
> while (f[n]<100){
+ s<-f[n]+f[n-1]
+ f[n+1]<-s
+ n<-n+1}
> f
```

ساختار کنترلی دیگری که به منظور تکرار یک حلقه در R بـه‌کار بـرده می‌شـود، عبارت است از repeat که به صورت زیر تعریف می‌شود.

repeat {عبارت دستوری if(شرط) break()}

```
> x<-0
> repeat { print (sin(x))
+          x<-x+pi/4
+          if (x >2*pi)break()}
[1] 0
[1] 0.7071068
[1] 1
[1] 0.7071068
[1] 1.224606e-16
[1] -0.7071068
[1] -1
[1] -0.7071068
[1] -2.449213e-16
```

در مثال فوق، مقدار sin(x) را برای زاویه‌های نیمساز کوچکتر از ۲π محاسبه شده است و در سطر آخر برای (شرط) x بزرگتر از ۲π ، توسط دستور break() از حلقه خارج می‌شود.لازم به ذکر است که در محیط برنامه‌ی R می‌توان در عبارت دستوری یک ساختار، از ساختار دستوری دیگری به صورت تو در تو استفاده کرد.

نکته: یک دستور دیگر برای چاپ خروجی دستور cat است. به مثال زیر توجه کنید(عبارت "\n" در انتهای دستور cat نشان دهنده این است که خروجی بعدی را در سطر بعد نمایش دهد):

```
> for(i in 1:10){
+ d<-prod(1:i)
+ cat("The Value of",i,"! is=",d,"\n")
+ }
The Value of 1 ! is= 1
The Value of 2 ! is= 2
The Value of 3 ! is= 6
The Value of 4 ! is= 24
The Value of 5 ! is= 120
The Value of 6 ! is= 720
The Value of 7 ! is= 5040
The Value of 8 ! is= 40320
The Value of 9 ! is= 362880
The Value of 10 ! is= 3628800
```

ساختارهای کنترلی دیگری نیز وجود دارند که عبارتند از sapply، tapply، apply و lapply.

۱. sapply: در این دستور که به صورت

```
> sapply(vec, f)
```

می باشد ورودی آن یک بردار است و عملگر f روی عناصر آن عمل می کند.
نمونه ای از کاربرد این دستور به صورت زیر است

```
> sapply(c(-2,3,-4,5,-6),abs)
[1] 2 3 4 5 6
```

به عنوان مثالی دیگر

```
> sapply(3:5,seq)
[[1]]
[1] 1 2 3

[[2]]
[1] 1 2 3 4

[[3]]
[1] 1 2 3 4 5
```

تابع sapply برای محاسبات تکراری پیچیده می تواند بسیار مفید باشد.

2. lapply: شکل کلی تابع lapply مانند sapply می باشد و به صورت زیر است

```
> lapply(vec,f)
```

تنها تفاوت این دستور با sapply به این صورت است که خروجی آن به صورت لیست می باشد

3. apply: در این دستور ورودی آن یک ماتریس به صورت

```
> apply(matrix, 1/2, f)
```

می باشد و خروجی آن یک بردار است. در این دستور عملگر f روی سطر (عدد ۱) یا ستون (عدد ۲) ماتریس عمل می کند. نمونه ای از این دستور به صورت زیر است:

```
> x<-matrix(1:24,nrow=4)
> apply(x, 1, sum)
[1] 66 72 78 84
```

4. tapply: در این دستور که به صورت زیر می باشد

```
> tapply(vec, grouping, f)
```

خروجی این دستور می تواند به صورت آرایه یا ماتریس باشد.

در برنامه نویسی و طراحی الگوریتم های مورد نظر گاهی اوقات محاسبه مدت زمان اجرای برنامه از اهمیت خاصی برخوردار است برای این کار ابتدا زمان سیستم ثبت می شود و در پایان اجرای برنامه زمان فعلی از زمان ثبت شده قبلی کسر می گردد. این دستور به صورت زیر است:

```
> p1 <- proc.time()
> proc.time() - p1
```

به عنوان مثال زمان اجرای برنامه زیر یعنی ۱,۴۷ ثانیه در خروجی نشان داده شده است

```
> p1 <- proc.time()
> w<-rnorm(10000000)
> proc.time() - p1
   user  system elapsed
   1.42    0.05    1.47
```

همچنین برای ایجاد وقفه در اجرای یک برنامه از دستور Sys.sleep(k) استفاده می شود طوریکه این دستور مدت زمانی به اندازه k ثانیه اجرای دستورات بعدی را به تعویق می اندازد. به عنوان مثال دستور زیر ابتدا تابع sin(x) را برای یک x مشخص محاسبه می کند و بعد از ۵ ثانیه مقدار cos(x) را برای همان x محاسبه می نماید.

```
x<-pi/2
y1<-sin(x)
y1
Sys.sleep(5)
y2<-cos(x)
y2
```

تمرینات:

۱. برنامه ای بنویسید که دوعدد را از کاربر بگیرد و چهار عمل اصلی حساب را روی آنها انجام دهد.

جواب:

۲. برنامه بنویسید که بردار x را به عنوان متغیر مستقل و بردار y را به عنوان متغیر وابسته انتخاب کند و ضرایب رگرسیون خطی ساده را برای آن محاسبه نماید.

جواب:

۳. برنامه ای بنویسید که از یک فرایند مارکف با ماتریس احتمال انتقال و بردار احتمال آغازی معین شبیه سازی کند

جواب:

```
> gms<-function(tran, ini, seq){
+ nul<-c(1, 2, 3)
+ myseq<-vector()
+ first<-sample(nul, 1, rep=TRUE, prob=ini)
+ myseq[1]<-first
+ for(i in 2:seq) {
+ pre<-myseq[i-1]
+ probab<-tran[pre, ]
+ myseq[i]<-sample(nul, 1, rep=TRUE, prob=probab)
+ }
+ return(myseq)
+ }
```

۴. برنامه ای بنویسید که دنباله ای از صفر و یک ها تولید کند و تعداد یک های پشت سر هم را یادداشت نمایند. قابل توجه است که تعداد یک های پشت سر هم به دوره گردش معروف است.

جواب:

۵. برنامه ای بنویسید که ریشه های یک معادله را با استفاده از روش نیوتن محاسبه نماید.

جواب:

۶. برنامه ای بنویسید که یک جدول توزیع احتمال را دریافت کرده و ضریب همبستگی را برای آن محاسبه نماید.

جواب:

۷. برنامه ای بنویسید که جدول z یعنی جدول توزیع نرمال را چاپ کند.

جواب:

۸. برنامه ای بنویسید که ضریب همبستگی رتبه ای تای کندال را برای دو بردار دلخواه از متغیرها محاسبه کند.

جواب:

۹. برنامه ای بنویسید که یک بردار دلخواه را دریافت کند و آن را مرتب نماید.

جواب:

۱۰.

محاسبات ریاضی

نرم افزار R امکان انجام محاسبات متنوع ریاضیاتی را به کاربران خود می‌دهد. در این بخش به برخی از موارد آن اشاره‌ای گذرا خواهیم داشت.

۴-۱ حل دستگاه معادلات خطی

با استفاده از برنامه‌ی R می‌توان به حل دستگاه معادلات خطی پرداخت، برای حل دستگاه معادلات خطی

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$$

با تعریف ماتریس ضرائب a، b، d و e ، و ماتریس مقادیر معلوم c و f به صورت زیر

$$A = \begin{bmatrix} a & b \\ d & e \end{bmatrix}, \qquad \mathrm{B} = \begin{bmatrix} \mathrm{c} \\ \mathrm{f} \end{bmatrix}$$

و از دستور solve(A,B) استفاده می‌شود.

به عنوان مثال برای حل دستگاه معادلات

$$\begin{cases} ۴x + ۲y = ۸ \\ ۱x + ۳y = ۹ \end{cases}$$

به صورت زیر عمل می‌کنیم

```
> A<-matrix(c(4,1,2,3),nrow=2)
> B<-matrix(c(8,9), nrow=2)
> solve(A,B)
     [,1]
[1,]  0.6
[2,]  2.8
```

به همین روش می‌توان به حل دستگاه‌ی خطی با بیش از دو معادله نیز پرداخت، به عنوان مثال یک دستگاه سه معادله و سه مجهول به صورت زیر حل شده است

```
> A<-matrix(c(4,1,2,3,0,1,0,2,1),nrow=3)
> B<-matrix(c(8,9,0), nrow=3)
> solve(A,B)
     [,1]
[1,]  -43
[2,]   60
[3,]   26
```

در این حالت و به هنگام تعریف ماتریس ضرائب باید توجه کرد، کـه در برنامـه R دستور matrix به صورت ستونی ماتریس‌ها رو تعریف می‌کند. در تعریف ماتریس ضرائب باید ضرائب متغیر اول در ستون اول، ضرائب متغیر دوم در ستون دوم و ... قرار بگیرند.

محاسبه حد:

۲-۴ محاسبه‌ی مشتق و انتگرال

برای محاسبه‌ی مشتق f(x) نسبت به x از دسـتور D و بـه صـورت زیـر اسـتفاده می‌کنیم.

```
> D(expression(f(x)),"x")
```

در زیر چند مثال ساده ارائه شده‌اند

```
> D(expression(exp(x)),"x")
exp(x)
> D(expression(3*x^2+2*x),"x")
3 * (2 * x) + 2
> D(expression(sin(x)),"x")
cos(x)
```

همچنین برای محاسبه‌ی انتگرال یگانـه، در محـیط R از دسـتور integrate و بـه صورت

```
> integrate(f(x),a,b)
```

استفاده می‌شود که در آن f(x) یک تابع می‌باشـد. در مثـال زیـر مقـدار انتگـرال

sin(x) از فاصله‌ی $\frac{\pi}{۴}$ تا ۳π را محاسبه کرده‌ایم

```
> int <- function(x) {sin(x)}
> integrate(int,pi/4,3*pi)
1.707107 with absolute error < 1.1e-13
```

محاسبه انتگرال به صورت عددی:

ابتدا به محاسبه انتگرال ها یک بعدی روی بازه های متناهی و غیر متناهی پرداخته و سپس آنها را در حالت چندگانه مورد بررسی قرار می دهیم. برای این کار فرض کنید بخواهیم انتگرال زیر را حل کنیم

$$\int_0^{\infty} \frac{1}{(x+1)\sqrt{x}} dx$$

این انتگرال به صورت زیر حل میگردد

```
> ## define the integrated function
> integrand <- function(x) {1/((x+1)*sqrt(x))}
> ## integrate the function from 0 to infinity
> integrate(integrand, lower = 0, upper = Inf)
3.141593 with absolute error < 2.7e-05
```

به عنوان مثال دیگر انتگرال زیر را درنظر بگیرید

$$\int_{-1.96}^{1.96} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$$

جواب عددی این انتگرال به صورت زیر است

```
> f <- function(x) {1/sqrt(2*pi)*exp(-x^2/2)}
> integrate(f, lower = -1.96, upper = 1.96)
0.9500042 with absolute error < 1e-11
```

گام اول برای محاسبه انتگرال های چندگانه عددی، نصب بسته نرم افزاری cubature است. به عنوان مثال برای حل انتگرال زیر

$$\int_0^{\frac{1}{2}}\int_0^{\frac{1}{2}}\int_0^{\frac{1}{2}}\frac{2}{3}(x_1+x_2+x_3)dx_1dx_2dx_3$$

با استفاده از دستور adaptIntegrate به صورت زیر عمل می کنیم

```
> library(cubature) # load the package "cubature"
Warning message:
package 'cubature' was built under R version 3.2.2
> f <- function(x) { 2/3 * (x[1] + x[2] + x[3]) } # "x" is vector
> adaptIntegrate(f, lowerLimit = c(0, 0, 0), upperLimit = c(0.5, 0.5, 0.5))
$integral
[1] 0.0625

$error
[1] 1.126893e-17

$functionEvaluations
[1] 33

$returnCode
[1] 0
```

ادامه

۳-۴ یافتن ریشه‌های یک معادله‌ی چندجمله‌ای

در محیط نرم‌افزار R برای یافتن جـواب معادلـه‌ی $ax^k + bx^{k-1} + \cdots + cx + d = 0$ از دستور polyroot و به صورت

```
> polyroot(c(d,c,...,b,a))
```

استفاده می‌شود. به عنوان مثال برای یافتن ریشـه‌های معادلـه‌ی $4x^3 + 2 = 1$ ابتدا معادله را به $4x^3 + 1 = 0$ تبدیل کرده و داریم

```
> polyroot(c(1,0,0,4))
[1]  0.3149803+0.5455618i -0.6299605+0.0000000i
[3]  0.3149803-0.5455618i
```

همچنین از دستور uniroot(f, c(a,b)) برای پیدا کردن ریشه تابع f(x) بین مقادیر a و b استفاده می شود. به عنوان مثال

```
> f<-function(x){x^2-1}
> uniroot(f,c(0,2))
$root
[1] 0.9999997

$f.root
[1] -5.356504e-07

$iter
[1] 6

$init.it
[1] NA

$estim.prec
[1] 6.535148e-05
```

برای تغییر در محاسبه خطای ریشه می توان از آرگومان Tlorance به صورت زیر استفاده کرد.

```
> uniroot(f,
+ c(0,2), tol=2^-52)
$root
[1] 1

$f.root
[1] 0

$iter
[1] 8

$init.it
[1] NA

$estim.prec
[1] 6.508366e-05
```

توابع بهینه سازی

برای یافتن مینیمم مقدار یک تابع در بازه a تـا b از دسـتور optimize(f,c(a,b)) استفاده می کنیم به عنوان مثال

```
> f<-function(x){x^2-1}
> m = optimize(f,c(-5,5))
> m
$minimum
[1] 0

$objective
[1] -1
```

همچنین دستور optimize(f, c(a,b), p1=p1,p2=p2) مینیم مقدار تـابع f را در بازه a و b با وجود پارامترهای اضافی p1 و p2 محاسبه می کند.

```
> f<-function(x,p1,p2){x^p1-p2}
> optimize(f,c(-5,5), p1=2, p2=1)
$minimum
[1] 0

$objective
[1] -1
```

دستور optim(c(a,b,c),f) مقادیر x و y و z را می یابد که تابع f(x,y,z) را در ازای مقادیر اولیه a، b و c مینیمم می کند. به عنوان مثال داریم

```
> f<-function(x,y,z){x^2+y^2+z^-1}
> optim(c(1,2.2,3.4),f)$par
```

همچنین دستور optim(c(1,2.2,3.4), f, p1 = p1, p2 = p2)$par این کار را با وجود پارامترهای اضافی p1 و p2 انجام می دهد.

دستور دیگری که مشابه دستور optim کار می‌کند دستور nlminb است که تابع f را در ازای مقادیر اولیه مینیمم می کند. به عنوان مثال داریم

```
> f<-function(par){
+ x<-par[1]
+ y<-par[2]
+ z<-par[3]
+ x^2+y^2+z^-1
+ }
> nlminb(c(1,1,1),f)$par
[1] -1.777920e-17  1.017759e-18  1.525306e+10
```

توجه کنید که ورودی تابع در دستور nlminb باید به فرم بردار باشد. همچنین در دستورات optim و nlminb میتوان برای متغیرها با استفاده از دستورات lower=c() یا upper=c() کران تعیین نمود.

معادلات دیفرانسیل

نرم افزار R قابلیت حل معدلات دیفرانسیل را دارا می باشد به عنوان مثـال فـرض کنید بخواهیم معادله دیفرانسیل

$$\frac{dx}{dt} = 5x$$

را از $t = 3$ تا $t = 12$ را برای مقدار آغازی $x(3) = 7$ بـه صـورت عـددی حـل کنیم. برای این کار ابتدا باید بسته نرم افزاری *deSolve* را نصب و فراخوانی کـرد، سپس تابع f را به صورت زیر تعریف می کنیم.

```
> f = function(t,x,parms) {
+ return(list(5*x))
+ }
> y=lsoda(7, seq(3,8,
+ 0.5), f,NA)
```

جواب

```
> y
   time            1
1   3.0 7.000000e+00
2   3.5 8.527776e+01
3   4.0 1.038898e+03
4   4.5 1.265640e+04
5   5.0 1.541868e+05
6   5.5 1.878384e+06
7   6.0 2.288345e+07
8   6.5 2.787781e+08
9   7.0 3.396219e+09
10  7.5 4.137451e+10
11  8.0 5.040458e+11
```

سیستم معادلات دیفرانسیل

برای حل سیستمی از معادلات دیفرانسیل

$$\frac{dw}{dt} = 5w, \qquad \frac{dz}{dt} = 3w + 7t$$

از t=3 تا t=12 با مقادیر اولیه w(3)=7 و z(3)=8.2 بـه صـورت زیـر عمـل مـی کنیم. قابل ذکر است که به دلیل طولانی بودن محاسبات فقط قسمتی از خروجی نمایش داده شده است. ستون اول خروجی مقادیر زمان، ستون دوم مقـادیر w(t) و ستون سوم مقادیر z(t) می باشد.

```
> myfunc = function(t,x,parms) {
+ w = x[1]; z = x[2];
+ return(list(c(5*w, 3*w+7*z)))
+ }
> y=lsoda(c(7,8.2),
+ seq(3,12, 0.1), myfunc,NA)
> y
   time             1            2
1   3.0 7.000000e+00 8.200000e+00
2   3.1 1.154105e+01 2.034564e+01
3   3.2 1.902798e+01 4.729037e+01
4   3.3 3.137184e+01 1.056499e+02
5   3.4 5.172342e+01 2.299305e+02
6   3.5 8.527751e+01 4.913444e+02
7   3.6 1.405988e+02 1.036140e+03
8   3.7 2.318083e+02 2.163516e+03
9   3.8 3.821873e+02 4.483715e+03
10  3.9 6.301204e+02 9.238362e+03
```

همچنین برای حل عددی معادلات دیفرانسیل که وابسته به پارامترهای خاص می باشد به صورت زیر عمل می کنیم. به عنوان مثال برای حل معادله

$$\frac{dx}{dt} = rx(1 - x/K),$$

از t=0 تا t=20 با مقدار اولیه x(0)=2.5 به صورت زیر عمل می کنیم.

```
> func2=function(t,x,parms) {
+ r=parms[1]; K=parms[2]
+ return(list(r*x*(1-x/K)))
+ }
> y=lsoda(2.5,seq(0,20,0.1),
+ func2,c(1.3,50))
> y
    time         1
1    0.0  2.500000
2    0.1  2.827444
3    0.2  3.194893
4    0.3  3.606442
5    0.4  4.066398
6    0.5  4.579226
7    0.6  5.149477
8    0.7  5.781703
9    0.8  6.480337
10   0.9  7.249550
```

آمار توصیفی

۵-۱ آمارتوصیفی

در محیط نرم افزار R مشاهدات مربوط به یک متغیر به صورت یک بردار، یا یک ستون از یک ماتریس مورد بررسی قرار می‌گیرند.در این بخش به نحوه‌ی محاسبه‌ی شاخص‌های توصیفی برای داده‌های مربوط به یک متغیر، با استفاده از یک مثال می‌پردازیم. همواره آگاهی از ویژگی‌های کلی و توصیفی یک سری از مشاهدات قبل از پرداختن به مبحث مدل‌بندی و روش‌های آماری اهمیت ویژه‌ای دارد.

ابتدا با استفاده از تابع runif که تابع ایجاد نمونه‌ی تصادفی از توزیع یکنواخت می‌باشد، یک نمونه به اندازه‌ی ۱۰۰ ازتوزیع یکنواخت (۲،۴-) ایجاد کرده و نمونه مورد نظر را در شیء x قرار می‌دهیم

```
> x<-runif(100,-2,4)
```

اکنون برای نمونه‌ی ایجاد شده به محاسبه‌ی شاخص‌های آمار توصیفی می‌پردازیم. میانگین داده‌ها با استفاده از تابع mean محاسبه می‌شود

```
> mean(x)
[1] 0.9502419
```

مقدار انحراف استاندارد توسط تابع sd محاسبه می‌شود

```
> sd(x)
[1] 1.818655
```

برای محاسبه‌ی واریانس یک متغیر از تابع var استفاده می‌شود

```
> var(x)
[1] 3.307507
```

همچین مقدار میانه را تابع meadian برمی‌گرداند

```
> median(x)
[1] 0.9988701
```

با استفاده از دستور quantile می‌توان چارک‌ها را محاسبه کرد

```
> quantile(x)
        0%        25%        50%        75%       100%
-1.9312374 -0.6330585  0.9988701  2.4610529  3.9241489
```

دهک‌ها و صدک‌ها نیز توسط همین دستور quantile و با اضافه کردن یک بردار حاوی مقدار دهک یا صدک مورد نیاز به عنوان آرگومان دوم این تابع محاسبه می‌شوند، به عنوان مثال برای محاسبه‌ی دهک هفتم و صدک ۴۷م به صورت زیر عمل می‌کنیم

```
> quantile(x, c(0.47,0.7))
       47%        70%
0.8961001 2.3225543
```

یکی از تابع‌های موجود در محیط نرم‌افزار R ، تابع summary می‌باشد. این تابع به صورت همزمان کمترین مقدار، چارک اول، میانه، میانگین، چارک سوم و مقدار بیشینه را محاسبه می‌کند

```
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.9310 -0.6331  0.9989  0.9502  2.4610  3.9240
```

محاسبه چولگی و کشیدگی:

برای محاسبه چولگی و کشیدگی ابتدا باید بسته نرم افزاری moments را نصب و سپس این بسته را به صورت زیر فراخوانی کنید

```
> install.packages('moments')
```

```
> library(moments)
```

سپس به کمک دستورات زیر می توان چولگی و کشیدگی را برای یک بردار از داده ها محاسبه نمود

```
> skewness(x)
[1] -0.03559101
> kurtosis(x)
[1] 3.07084
```

نمودارها

از آنجایی که نمودارها در تجزیه و تحلیل‌های آماری از اهمیت خاصی برخوردار هستند، مانند هر برنامه آماری دیگر، نرم‌افزار R نیز امکان ترسیم نمودارهای آماری را به کاربران می‌دهد. رسم یک نمودار آماری به تعداد متغیرها، اینکه بخواهیم یک متغیر را در طول زمان رسم کرده و یا اینکه دو متغیر را در مقابل هم و ... بستگی دارد.

تابع مورد استفاده در رسم نمودارها استفاده از دستور plot می‌باشد، این تابع آرگومان‌های مختلفی دارد که به کمک آنها می‌توان ویژگی‌های نمودار رسم شده را تغییر داد. در اینجا به معرفی این تابع و برخی از آرگومان‌های آن می‌پردازیم.

فرم کلی تابع plot و شناسه هایش به صورت زیر می باشد

```
plot(x, y, pch ="یک کاراکتر", lty = یک عدد, main ="maint itle", sub ="subtitle", xlab ="xAxislable",
ylab ="yAxislable", xlim = یک بازه, ylim = یک بازه, type ="p")
```

در اینجا دستور plot نمودار نقطه‌ای ساده‌ی x را در مقابل y رسم می کند. شکل زیر را ببینید.

```
> x<--5:5
> y<-x^2
> plot(x,y)
```