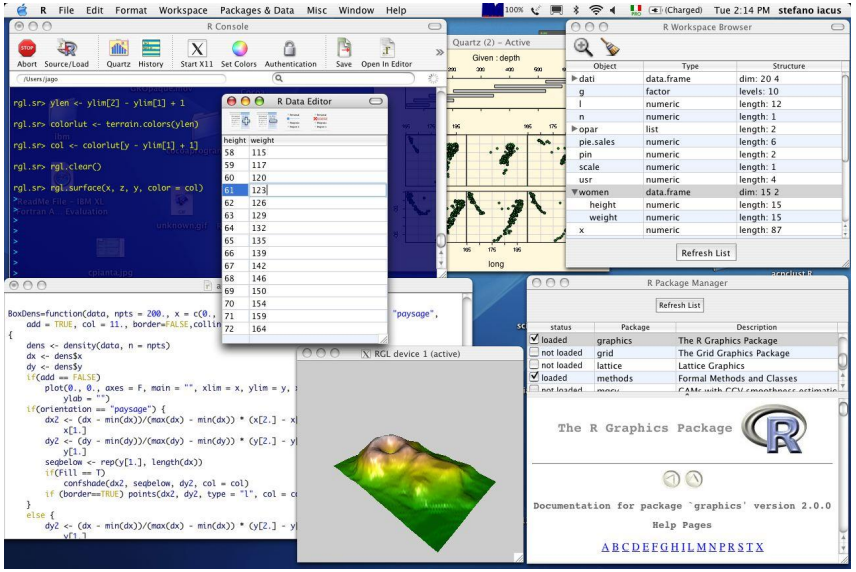
# Введение в R

# Что такое R?

В августе 1993 г. двое молодых новозеландских ученых Ross Ihaka и Robert Gentleman анонсировали свою новую разработку, которую они назвали R. (кафедра статистики в университете Окленда).


Проект вначале развивался довольно медленно, но когда в нем появилось достаточно возможностей, в том числе уникальная по легкости система написания дополнений (пакетов), язык R и его среда стали очень популярны во всём мире среди специалистов разных областей. Сейчас множество людей развивают этот язык.



# Что такое R?

- Программное средство для
  - Чтения и манипулирования данными
  - Вычислений
  - Проведения статистического анализа
  - Отображения результатов
- Среда программирования

  R – это среда программирования для анализа данных и графики..
- Платформа для разработки и внедрения новых алгоритмов.

  R предоставляет платформу для разработки новых алгоритмов и передачи методов. Это может быть достигнуто тремя путями:
  - Функции, которые используют существующие в R алгоритмы
  - Функции, которые вызывают процедуры, написанные на C или Fortran
  - Создание пакетов, содержащих код для обобщения и представления данных, вывода их на печать или в виде графиков

# Применение, преимущества и недостатки R

Коротко говоря, R применяется везде, где нужна работа с данными. Это не только статистика в узком смысле слова, но и «первичный» анализ (графики, таблицы сопряженности) и продвинутое математическое моделирование.

В принципе, R может использоваться и там, где в настоящее время принято использовать специализированные программы математического анализа, такие как MATLAB или Octave. Но, разумеется, более всего его применяют для статистического анализа — от вычисления средних величин до вейвлет-преобразований и временных рядов.

Географически R распространен тоже очень широко. Трудно найти американский или западноевропейский университет, где бы не работали с R. Очень многие серьезные компании (скажем, Boeing) устанавливают R для работы

# Применение, преимущества и недостатки R

У R два главных преимущества:

1. **неимоверная гибкость**
2. **свободный код.**

Гибкость позволяет создавать приложения (пакеты) практически на любой случай жизни.

Свободный код — это не просто бесплатность программы, но и возможность разобраться, как именно происходит анализ, а если в коде встретилась ошибка — самостоятельно исправить ее и сделать исправление доступным для всех.

# Применение, преимущества и недостатки R

У R есть и немало недостатков.

Самый главный из них — это **трудность обучения программе.** Команд много, вводить их надо вручную, запомнить все трудно, а привычной системы меню нет. Поэтому порой очень трудно найти, как именно сделать какой-нибудь анализ. Если функция известна, то узнать, что она делает, очень легко, обычно достаточно набрать команду help(*название функции*). Увидеть код функции тоже легко, для этого надо просто набрать ее название без скобок или (лучше) ввести команду getAnywhere(название функции).

# Команды языка R

- В R все команды записываются в файл .Rhistory. Команды можно вызывать повторно, в отличие от MatLab. Чтобы убедиться в сохранении истории команд, можно явно воспользоваться функцией savedhistory(). История команд, использованных во время предыдущей сессии, может быть вызвана с помощью функции loadhistory(). Предыдущие команды вызываются клавишами ↑ и ↓.

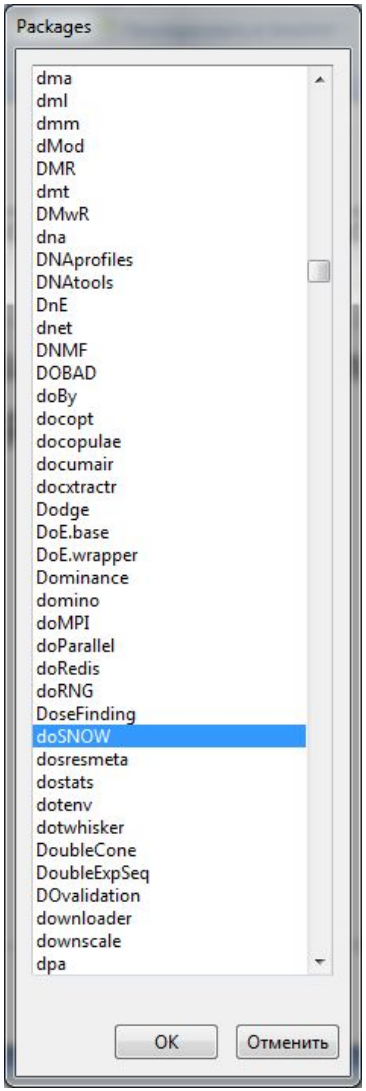
# Объекты R

- По умолчанию R создает объекты в памяти и сохраняет их в единственный файл .Rdata. Объекты R автоматически сохраняются в этот файл. Пакеты загружаются в текущей сессии R.

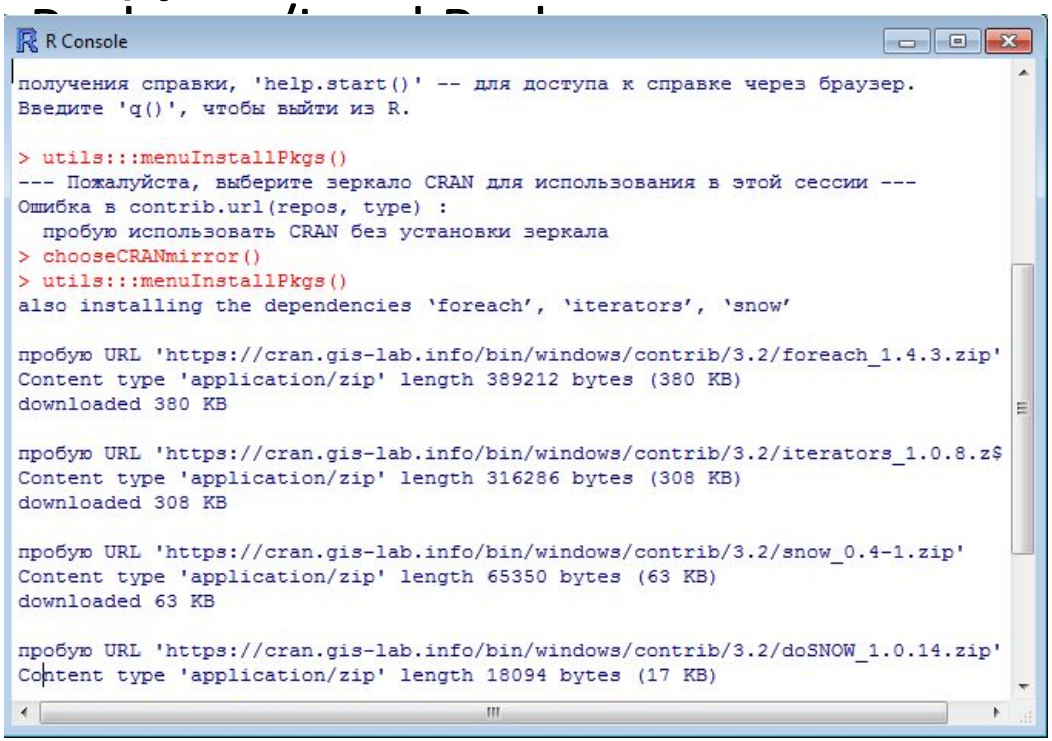
# Выход из R

- Команда q()
- Или просто закрыть окно. При этом будет предложено сохранить сессию.

# Инсталляция пакетов R




- Инсталлировать пакет в R можно с помощью меню Packages/Install Packages. При этом будет предложено выбрать сайт для инсталляции. После инсталляции пакеты можно загружать в R с помощью



# Язык R

Базовый синтаксис

# Ввод команд в R

По умолчанию место для ввода команды в R обозначается знаком >:

```
> 5+2
[1] 7
```

Если команда синтаксически неполная, появляется знак продолжения +:

```
> 8+3*
+ 5
[1] 23
```

Оператор присваивания – левая стрелка <-:

```
> a<-4+5
> a
[1] 9
```

# Ввод команд в R

Последнее выражение можно получить с помощью внутреннего объекта .Last.value:

```
> value<-.Last.value
> value
[1] 9
```

Функции rm() или remove() используются для удаления объектов из рабочей директории:

```
> rm(value)
> value
Ошибка: объект 'value' не найден
```

# Имена в R

- Имена в R могут быть любыми комбинациями букв, цифр и точек, но они не могут начинаться с цифры. R чувствителен к регистру.
- Нужно избегать использования имен встроенных функций в качестве объектов. Для этого желательно проверять содержание объекта, который вы хотите использовать.

```
> value
Ошибка: объект 'value' не найден
> T
[1] TRUE
> t
function (x)
UseMethod("t")
<bytecode: 0x000000001394b0f0>
<environment: namespace:base>
```

# Использование пробелов

- R игнорирует лишние пробелы между именами объектов и операторами:

```
> value <- 2 *    4
> value
[1] 8
```

- Но в операторе присваивания нельзя использовать пробел между < и -.
- Количество пробелов в выражениях, стоящих в кавычках, существенно:

```
> value<-"Hello"
> value1<-" Hello"
> value==value1
[1] FALSE
```

# Справка

- Вызов справки по функции, объекту или оператору осуществляется следующими командами:

  >?function

  >help(function)

  или вызовом меню Help в R.
- Вызов справки по какой-либо теме осуществляется командой >help.search("topic"), например:

> help.search("linear regression")

# Типы данных

- В R есть четыре атомарных типа данных
  - Numeric
    > value <- 605
    > value
    [1] 605
  - Character
    > string <- "Hello World«
    > string
    [1] "Hello World«
  - Logical
    > 2 < 4
    [1] TRUE
  - Complex number
    > cn <- 2 + 3i
    > cn
    [1] 2+3i

# Атрибуты объекта

- Атрибуты важны при манипулировании объектами. У всех объектов есть два атрибута -- mode и length.

```
> mode(value)
[1] "numeric"
> length(value)
[1] 1
> mode(string)
[1] "character"
> length(string)
[1] 1
> mode(2<4)
[1] "logical"
> mode(cn)
[1] "complex"
> length(cn)
[1] 1
> mode(sin)
[1] "function"
```

- Объекты NULL – это пустые объекты без присвоенного mode. Их длина равна нулю.

```
> names(value)
[1] NULL
```

# Пропущенные значения

- Во многих практических примерах некоторые элементы данных могут быть неизвестны, следовательно, им будет присвоено пропущенное значение. Код для пропущенных значений это NA. Он указывает на то, что значение элемента объекта неизвестно. Каждая операция над NA дает результат NA.
- Функция is.na() может быть использована для проверки пропущенных значений в объекте.

```
> value <- c(3,6,23,NA)
> is.na(value)
[1] FALSE FALSE FALSE TRUE
> any(is.na(value))
[1] TRUE
> na.omit(value)
[1] 3 6 23
> attr(,"na.action")
[1] 4
> attr(,"class")
[1] "omit"
```

# Неопределенные и бесконечные значения

- Бесконечные и неопределенные значения (Inf, -Inf and NaN) могут быть протестированы с помощью функций is.finite, is.infinite, is.nan и is.number аналогичным образом.
- Эти значения можно получить, например, при делении на 0 или взятии логарифма от 0.

```
> value1 <- 5/0
> value2 <- log(0)
> value3 <- 0/0
> cat("value1 = ",value1," value2 = ",value2,
" value3 = ",value3,"\n")
value1 = Inf value2 = -Inf value3 = NaN
```

# Арифметические операторы

| оператор | описание | пример |
|---|---|---|
| + | Сложение | >2+5<br>[1] 7 |
| - | Вычитание | >2-5<br>[1] -3 |
| * | Умножение | >2*5<br>[1] 10 |
| / | Деление | >2/5<br>[1] 0.4 |
| ^ | Возведение в степень | >2^5<br>[1] 32 |
| %/% | Целочисленное деление | >5%/%2<br>[1] 2 |
| %% | Деление по модулю (остаток от деления) | >5%%2<br>[1] 1 |

# Операторы сравнения

| Оператор | описание | пример |
|---|---|---|
| == | равно | > value1<br>[1] 3 6 23<br>> value1==23<br>[1] FALSE FALSE TRUE |
| != | Не равно | > value1 != 23<br>[1] TRUE TRUE FALSE |
| < | Меньше | > value1 < 6<br>[1] TRUE FALSE FALSE |
| > | Больше | > value1 > 6<br>[1] FALSE FALSE TRUE |
| <= | Меньше или равно | > value1 <= 6<br>[1] TRUE TRUE FALSE |
| >= | Больше или равно | > value1 >= 6<br>[1] FALSE FALSE TRUE |

# Логические операторы

| Оператор | описание | пример |
|---|---|---|
| & | Поэлементное и | > value2<br>[1] 1 2 3<br>> value1==6 & value2 <= 2<br>[1] FALSE TRUE FALSE |
| \| | Поэлементное или | > value1==6 \| value2 <= 2<br>[1] TRUE TRUE FALSE |
| && | Управляющее и (проверяет только первый элемент вектора) | > value1[1] <- NA<br>> is.na(value1) && value2 == 1<br>[1] TRUE |
| \|\| | Управляющее или (проверяет только первый элемент вектора) | > is.na(value1) \|\| value2 == 4<br>[1] TRUE |
| xor | Поэлементное исключающее или | > xor(is.na(value1), value2 == 2)<br>[1] TRUE TRUE FALSE |
| ! | Логическое отрицание | > !is.na(value1)<br>[1] FALSE TRUE TRUE |

# Распределения и симуляция

- В R есть множество распределений для симуляции данных, нахождения квантилей, вероятностей и функций плотности. Менее распространенные распределения находятся в специальных пакетах.
- Примеры распределений вероятности:

| Функция R | Распределение | Параметры | Пакет |
|---|---|---|---|
| binom | биномиальное | size, prob | stats |
| exp | экспоненциальное | rate | stats |
| gamma | Гамма | shape, rate | stats |
| norm | нормальное | Mean, sd | stats |

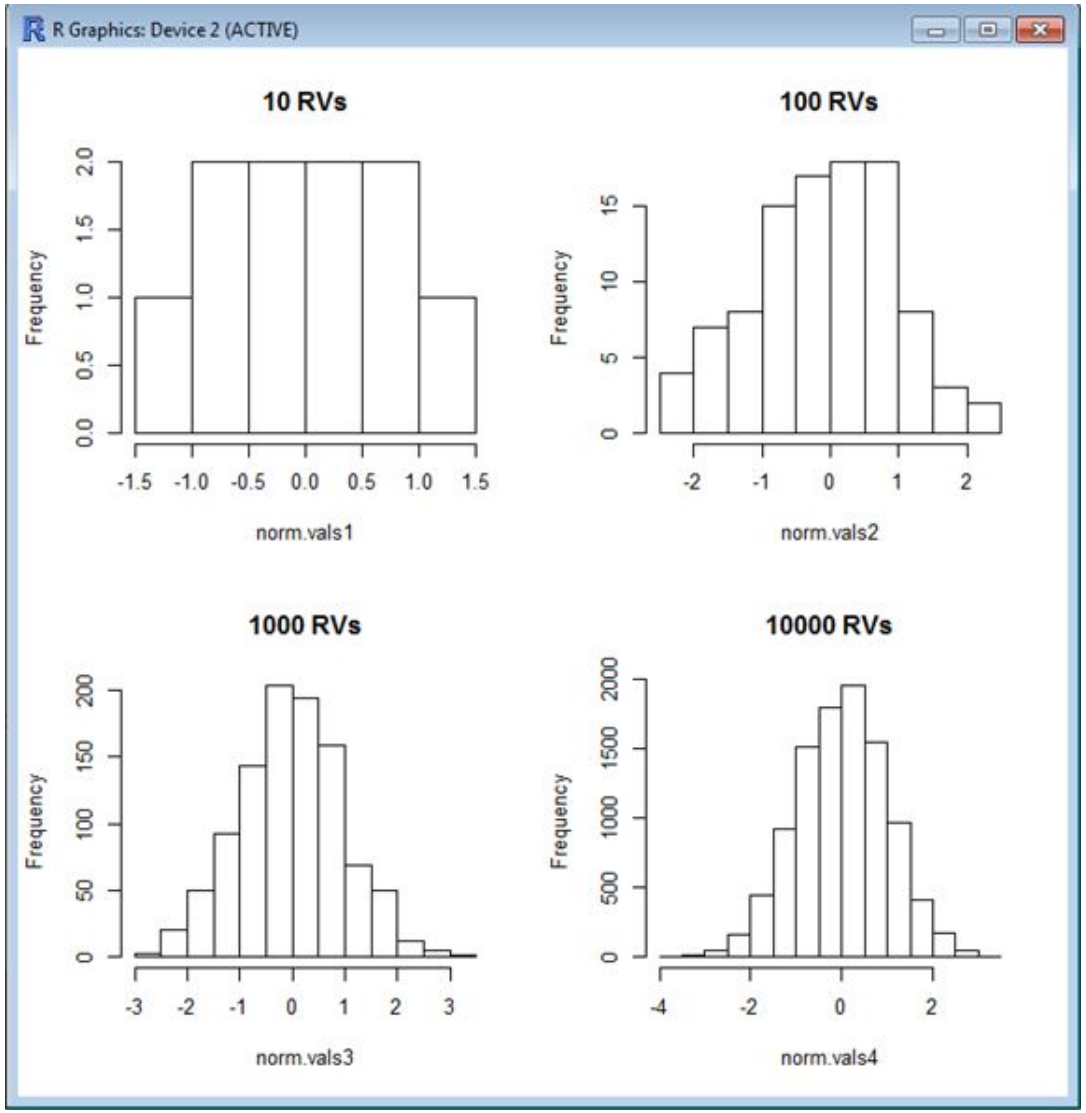
# Распределения и симуляция

- В R каждое в имени каждого распределения используется префикс, обозначающий, нужно ли использовать вероятность, квантиль, функцию плотности или случайное значение. Ниже показаны все возможные префиксы:
  - p: вероятности (функции распределения)
  - q: квантили (процентные точки)
  - d: функции плотности (вероятности для дискретных случайных величин)
  - r: случайные (или симулированные) значения.
- Следующий пример показывает, как можно симулировать данные из нормального распределения, используя функция rnorm.

# Пример

```
norm.vals1 <- rnorm(n=10)
norm.vals2 <- rnorm(n=100)
norm.vals3 <- rnorm(n=1000)
norm.vals4 <- rnorm(n=10000)
# set up plotting region
par(mfrow=c(2,2))
hist(norm.vals1,main="10 RVs")
hist(norm.vals2,main="100 RVs")
hist(norm.vals3,main="1000 RVs")
hist(norm.vals4,main="10000 RVs")
```

# Гистограммы



# Интерпретация результатов

- С ростом размера выборки форма распределения становится больше похожа на нормальное распределение. Про объект norm.vals1 трудно сказать, что он был сгенерирован из нормального распределения с мат.ожиданием 0 и СКО 1. Если посмотреть на суммарную статистику этого объекта, то увидим, что его мат. ожидание и СКО не близки к 0 и 1 соответственно.

```
> c(mean(norm.vals1),sd(norm.vals1))
[1] 0.2461831 0.7978427
```

- Посчитаем МО и СКО объекта norm.vals4, сгенерированного 10,000 случайных значений из распределения N(0, 1):

```
> c(mean(norm.vals4),sd(norm.vals4))
[1] 0.004500385 1.013574485
```

- Для больших симуляций, результат будет еще ближе:

```
> norm.vals5 <- rnorm(n=1000000)
> c(mean(norm.vals5),sd(norm.vals5))
[1] 0.0004690608 0.9994011738
```

# Центральная предельная теорема

- При приближении размера $n$ выборки, взятой из популяции с математическим ожиданием $\mu$ и дисперсией $\sigma^2$, к бесконечности, статистические оценки выборочного распределения будут сходится к рассматриваемым теоретическим распределениям.

# Объекты R

# Объекты данных в R

- Четыре наиболее часто используемых типа объектов данных в R – это векторы, матрицы, блоки данных и списки.
- Вектор – набор элементов одинакового вида (mode), логических, численных (integer или double), комплексных, символьных или списков.
- Матрица это множество элементов, представленных в виде строк и столбцов, где все элементы одного вида (mode), логических, численных (integer или double), комплексных или символьных.
- Блок данных – то же самое, что и матрица, но колонки могут быть разных видов.
- Список – это обобщение вектора, представляющее собой коллекцию объектов данных.

# Создание векторов

- **Функция c**
- Самый простой способ создать вектор – конкатенация с помощью функции c, связывающей вместе символьные, численные или логические элементы.

```
> value.num <- c(3,4,2,6,20)
> value.char <- c("koala","kangaroo","echidna")
> value.logical.1 <- c(F,F,T,T)
# or
> value.logical.2 <- c(FALSE,FALSE,TRUE,TRUE)
```

- Для логических векторов TRUE и FALSE – логические значения, а T и F – переменные с такими значениями.

# Создание векторов

- **Функции rep и seq**
- Функция rep реплицирует элементы векторов. Например,

```
> value <- rep(5,6)
> value
[1] 5 5 5 5 5 5
```

- Функция seq создает регулярную последовательность значений, формирующих вектор.

```
> seq(from=2,to=10,by=2)
[1] 2 4 6 8 10
> seq(from=2,to=10,length=5)
[1] 2 4 6 8 10
> 1:5
[1] 1 2 3 4 5
> seq(along=value)
[1] 1 2 3 4 5 6
```

# Функции, которые дают результат такой же длины

| Function | Description |
|---|---|
| cos, sin, tan | Cosine, Sine, Tangent |
| acos, asin, atan | Inverse functions |
| cosh, sinh, tanh | Hyperbolic functions |
| acosh, asinh, atanh | Inverse hyperbolic functions |
| log | Logarithm (any base, default is natural logarithm) |
| log10 | Logarithm (base 10) |
| exp | Exponential (*e* raised to a power) |
| round | Rounding |
| abs | Absolute value |
| ceiling, floor, trunc | Truncating to integer values |
| gamma | Gamma function |
| lgamma | Log of gamma function |
| sqrt | Square root |

# Функции, результатом которых является число

| Function | Description |
|---|---|
| sum | Sum elements of a vector |
| mean | arithmetic mean |
| max, min | Maximum and minimum |
| prod | Product of elements of a vector |
| sd | standard deviation |
| var | variance |
| median | 50th percentile |

# Создание матриц

- **Функции dim и matrix**
- Функция dim может использоваться для конвертации вектора в матрицу

```
> value <- rnorm(6)
> dim(value) <- c(2,3)
> value
[,1] [,2] [,3]
[1,] 0.7093460 -0.8643547 -0.1093764
[2,] -0.3461981 -1.7348805 1.8176161
```

- Чтобы конвертировать назад в вектор, надо опять применить функцию dim.

```
dim(value) <- NULL
```

- Или можно использовать функцию matrix

```
> matrix(value,2,3)
[,1] [,2] [,3]
[1,] 0.7093460 -0.8643547 -0.1093764
[2,] -0.3461981 -1.7348805 1.8176161
```

- Если хотим заполнять по строкам

```
> matrix(value,2,3,byrow=T)
[,1] [,2] [,3]
[1,] 0.709346 -0.3461981 -0.8643547
[2,] -1.734881 -0.1093764 1.8176161
```

# Создание списков

- Списки создаются с помощью функции list. Могут включать элементы различных видов, длины и размера

```
> L1 <- list(x = sample(1:5, 20, rep=T),
y = rep(letters[1:5], 4), z = rpois(20, 1))
> L1
$x
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
$y
[1] "a" "b" "c" "d" "e" "a" "b" "c" "d" "e" "a" "b“
[13] "c" "d" "e" "a" "b" "c" "d" "e"
$z
[1] 1 3 0 0 3 1 3 1 0 1 2 2 0 3 1 1 0 1 2 0
```

# Доступ к элементам списка

- Через имя или по номеру позиции, на которой находится элемент, с использованием операции взятия подмножества [[]] или извлечения $.

```
> L1[["x"]]
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
> L1$x
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
> L1[[1]]
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
```

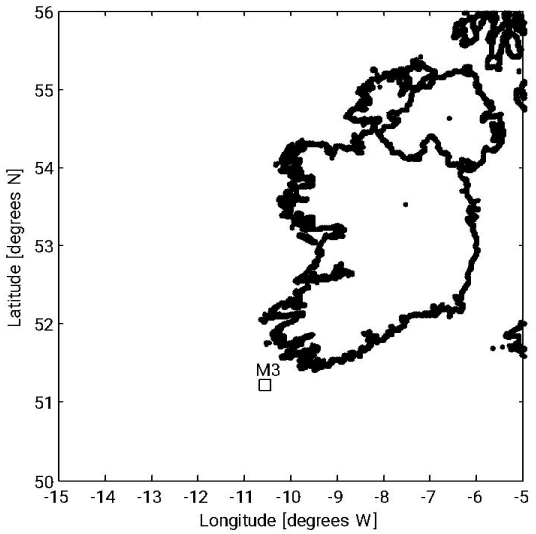
- Чтобы извлечь подсписок, используются одинарные скобки:

```
> L1[1]
$x
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
```

# Примеры – Прогноз скорости ветра для ветропарка в Валенсии (Ирландия)

Данные были получены с погодного буя M3, расположенного у юго-западного побережья Ирландии (рис. 1). Следующие параметры регистрировались ежечасно: ретроспе-ктива скорости и направление ветра, атмосферное давление, максимальная скорость порывов ветра, относите-льная влажность воздуха и пр.

Данные содержали различные пропуски. Для заполнения пропусков используется **<u>метод многомерного восстановления данных при помощи связанных уравнений</u>** (от англ. Multivariate imputation by chained equations, MICE).




| | x1 | x2 | x3 | x4 | y1 | y2 | y3 | y4 |
|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 10 | 10 | 8 | NA | 9.14 | 7.46 | 6.58 |
| 2 | 8 | 8 | 8 | 8 | NA | 8.14 | 6.77 | 5.76 |
| 3 | 13 | 13 | 13 | 8 | NA | 8.74 | 12.74 | NA |
| 4 | 9 | 9 | 9 | 8 | 8.81 | 8.77 | 7.11 | NA |
| 5 | 11 | 11 | 11 | 8 | 8.33 | 9.26 | 7.81 | NA |
| 6 | 14 | 14 | 14 | 8 | 9.96 | 8.10 | 8.84 | 7.04 |
| 7 | 6 | 6 | 6 | 8 | 7.24 | 6.13 | 6.08 | 5.25 |
| 8 | 4 | 4 | 4 | 19 | 4.26 | 3.10 | 5.39 | 12.50 |
| 9 | 12 | 12 | 12 | 8 | 10.84 | 9.13 | 8.15 | 5.56 |
| 10 | 7 | 7 | 7 | 8 | 4.82 | 7.26 | 6.42 | 7.91 |
| 11 | 5 | 5 | 5 | 8 | 5.68 | 4.74 | 5.73 | 6.89 |

# Примеры – Прогноз скорости ветра для ветропарка в Валенсии (Ирландия)

Примера программы на R для восстановления пропуска данных с помощью метода MICE

```
1  library("mice")
2  library("VIM")
3  library("graphics")
4
5  # Load and switch to new env.
6  #wdata <-load("WindData.RData")
7  #head(wdata)
8
9  # Let's visualize a number and structure of missing rows
10 # md.pattern(bouyData)
11 # data = bouyData;
12
13 # Use small build-in testing dataset for debugging
14 anscombe <- within(anscombe, {
15   y1[1:3] <- NA
16   y4[3:5] <- NA
17 });
18 data = anscombe;
19
20
21 imp <- mice(data, print = TRUE, seed = 23109, maxit = 5)
22 fulldata <- complete(imp)
23 print(imp)
24
25 # Save imputation distribution plot for all variables
26 #pdf('impdist.pdf')
27 #stripplot(imp, pch = 20, cex = 1.2)
28 #dev.off()
29
```

| | x1 | x2 | x3 | x4 | y1 | y2 | y3 | y4 |
|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 10 | 10 | 8 | NA | 9.14 | 7.46 | 6.58 |
| 2 | 8 | 8 | 8 | 8 | NA | 8.14 | 6.77 | 5.76 |
| 3 | 13 | 13 | 13 | 8 | NA | 8.74 | 12.74 | NA |
| 4 | 9 | 9 | 9 | 8 | 8.81 | 8.77 | 7.11 | NA |
| 5 | 11 | 11 | 11 | 8 | 8.33 | 9.26 | 7.81 | NA |
| 6 | 14 | 14 | 14 | 8 | 9.96 | 8.10 | 8.84 | 7.04 |
| 7 | 6 | 6 | 6 | 8 | 7.24 | 6.13 | 6.08 | 5.25 |
| 8 | 4 | 4 | 4 | 19 | 4.26 | 3.10 | 5.39 | 12.50 |
| 9 | 12 | 12 | 12 | 8 | 10.84 | 9.13 | 8.15 | 5.56 |
| 10 | 7 | 7 | 7 | 8 | 4.82 | 7.26 | 6.42 | 7.91 |
| 11 | 5 | 5 | 5 | 8 | 5.68 | 4.74 | 5.73 | 6.89 |

| | x1 | x2 | x3 | x4 | y1 | y2 | y3 | y4 |
|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 10 | 10 | 8 | 7.24 | 9.14 | 7.46 | 6.58 |
| 2 | 8 | 8 | 8 | 8 | 5.68 | 8.14 | 6.77 | 5.76 |
| 3 | 13 | 13 | 13 | 8 | 7.24 | 8.74 | 12.74 | 7.91 |
| 4 | 9 | 9 | 9 | 8 | 8.81 | 8.77 | 7.11 | 5.56 |
| 5 | 11 | 11 | 11 | 8 | 8.33 | 9.26 | 7.81 | 5.76 |
| 6 | 14 | 14 | 14 | 8 | 9.96 | 8.10 | 8.84 | 7.04 |
| 7 | 6 | 6 | 6 | 8 | 7.24 | 6.13 | 6.08 | 5.25 |
| 8 | 4 | 4 | 4 | 19 | 4.26 | 3.10 | 5.39 | 12.50 |
| 9 | 12 | 12 | 12 | 8 | 10.84 | 9.13 | 8.15 | 5.56 |
| 10 | 7 | 7 | 7 | 8 | 4.82 | 7.26 | 6.42 | 7.91 |
| 11 | 5 | 5 | 5 | 8 | 5.68 | 4.74 | 5.73 | 6.89 |

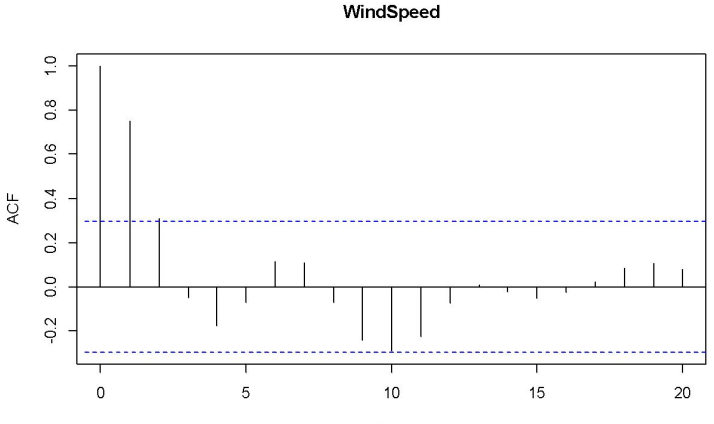
# Примеры – Прогноз выработки мощности для ветроустановок Апшеренского полуострова

Пример программы на R для прогноза выработки мощности ВЭУ на базе авторегрессионной модели и искусственной нейронной сети

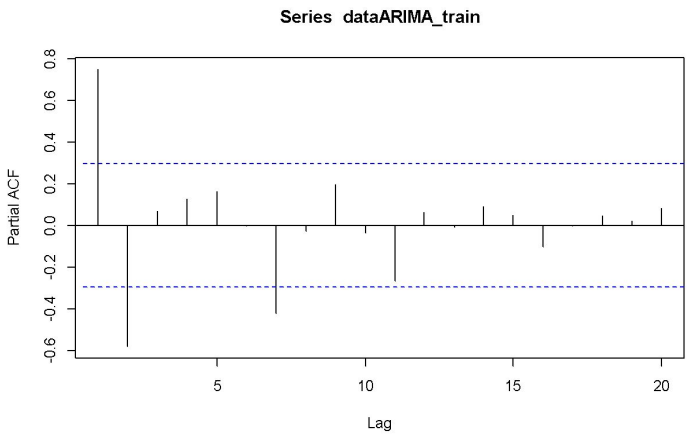
```r
# Загрузка необходимых пакетов
library(forecast)
library("hydroGOF")
library(caret)

#Загрузка исходных данных для расчёта
alldata <-read.csv("alldata.csv", sep = ";", na.strings=c("", "NA", "NULL"), header=TRUE)
dataARIMA_train <-read.csv("data_train_ARIMA.csv", sep = ";", na.strings=c("", "NA", "NULL"), header=TRUE)
dataARIMA_test <-read.csv("data_test_ARIMA.csv", sep = ";", na.strings=c("", "NA", "NULL"), header=TRUE)
dataANN_train <-read.csv("data_train_AI.csv", sep = ";", na.strings=c("", "NA", "NULL"), header=TRUE)
dataANN_test <-read.csv("data_test_AI.csv", sep = ";", na.strings=c("", "NA", "NULL"), header=TRUE)

#Построение графика исходных данных скорости ветра
plot.ts(alldata)

# Прогнозирование на базе модели АРПСС
acf(dataARIMA_train, lag.max=20) #построение автокорреляционной функции
pacf(dataARIMA_train, lag.max=20) #построение частной автокорреляционной функции
fit <- auto.arima(dataARIMA_train) #построение прогнозной модели АРПСС
fit_test <-Arima(dataARIMA_test, model = fit) # тестирование модели АРПСС на тестовой выборке
fit #вывод данных о полученной модели АРПСС
accuracy(fit_test) # расчёт ошибок прогноза модели АРПСС
plot(forecast(fit, 16)) #построение графика результатов тестирования АРПСС
lines(alldata, pch=22, lty=2, col="black") #построение графика результатов тестирования АРПСС
title(xlab="Время", col.lab=rgb(0,0.5,0)) #построение графика результатов тестирования АРПСС
title(ylab="Скорость ветра", col.lab=rgb(0,0.5,0)) #построение графика результатов тестирования АРПСС

# Прогнозирование на базе интеллектуальной модели
control <- trainControl(method="repeatedcv", number=10, repeats=3) # задание метода настройки и тестирования интеллектуальной модели
model <- train(windSpeed ~ ., data=dataANN_train, method="elm", trControl=control, tuneLength=5) # задание интеллектуальнйо модели по
plot(model) # построение графика с результатми настройки интеллектуальной модели
model_test <- predict(model, dataANN_test) # выполнение прогноза на 16 значений вперёд на базе интеллектуальной модели
RMSE <- rmse(model_test, dataANN_test$WindSpeed)  # расчёт ошибок прогноза интеллектуальной модели
MAE <- mae(model_test, dataANN_test$WindSpeed) # расчёт ошибок прогноза интеллектуальной модели
RMSE # вывод ошибки RMSE
MAE # вывод ошибки MAE
plot(dataANN_test$WindSpeed, type="o", col="blue", ann=FALSE) #построение графика результатов тестирования интеллектуальной модели
lines(model_test, type="o", pch=22, lty=2, col="red") #построение графика результатов тестирования интеллектуальной модели
title(xlab="Время", col.lab=rgb(0,0.5,0)) #построение графика результатов тестирования интеллектуальной модели
title(ylab="Скорость ветра", col.lab=rgb(0,0.5,0)) #построение графика результатов тестирования интеллектуальной модели
```

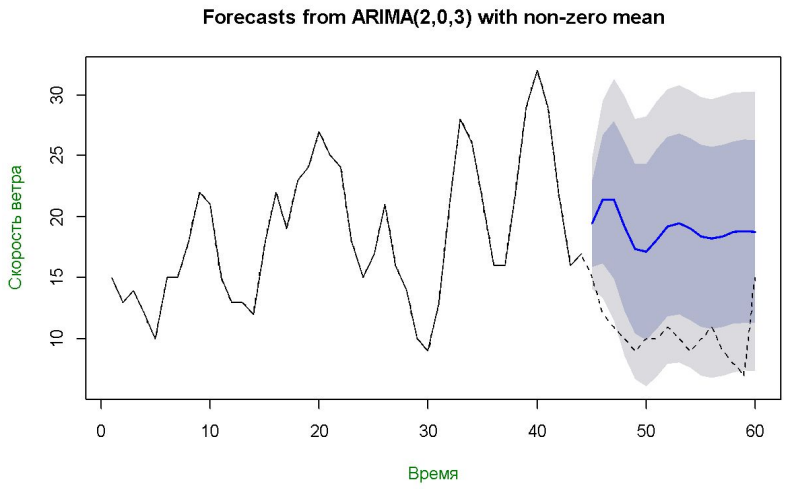
# Примеры – Прогноз выработки мощности для ветроустановок Апшеренского полуострова

Понимание скрытой периодичности и тенденций временного ряда позволяет прогнозировать его поведение в ближайшем будущем. Простой и общий подход к описанию эффектов памяти реализуют модели ARIMA. Задача прогнозирования на базе моделей ARIMA состоит в определении коэффициентов полиномов авторегрессии AR(p) и скользящего среднего MA(q), по данным выборки стационарного процесса выработки мощности ветротурбинами



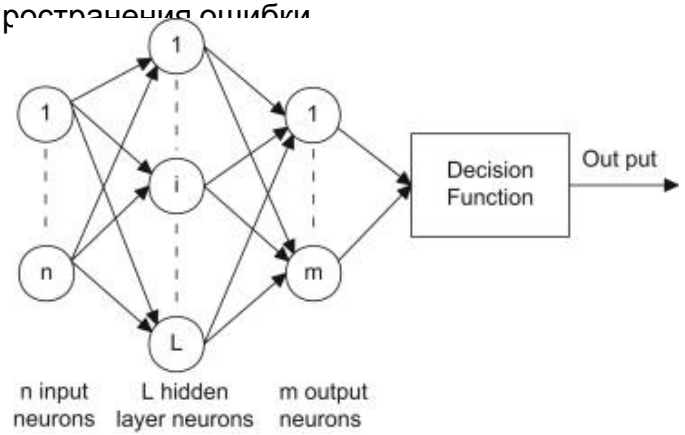

```
> fit #вывод данных о полученной модели АРПСС
Series: dataARIMA_train
ARIMA(2,0,3) with non-zero mean

Coefficients:
         ar1      ar2     ma1     ma2     ma3  intercept
      0.8732  -0.6470  0.2272  0.7488  0.5074    18.6312
s.e.  0.1955   0.1704  0.2025  0.1745  0.1692     1.2315

sigma^2 estimated as 7.642:  log likelihood=-106.18
AIC=226.35   AICc=229.46   BIC=238.84
> accuracy(fit_test) # расчёт ошибок прогноза модели АРПСС
                     ME     RMSE      MAE       MPE     MAPE     MASE        ACF1
Training set -2.262717 3.218292 2.962307 -25.03743 29.70136 1.851442 -0.0042517
```

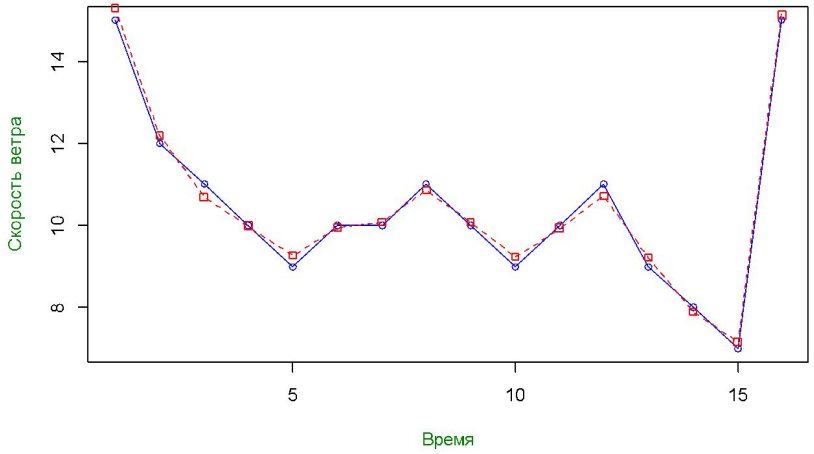


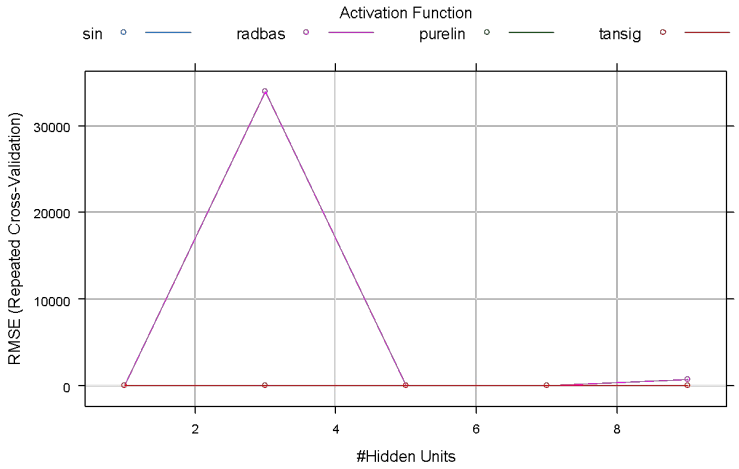



# Примеры – Прогноз выработки мощности для ветроустановок Апшеренского полуострова

**Extreme Learning Machine** представляет собой особый алгоритм обучения, который по структуре представляет из себя нейронную сеть с одним скрытым слоем. Веса для скрытого слоя назначаются случайным образом, а для открытого — взятием псевдообратной матрицы. Такой принцип даёт очень высокую скорость обучения, по сравнению с популярными алгоритмами обучения ИНС, к примеру методом обратного распространения ошибки.




```
RMSE was used to select the optimal model using  the smallest value.
The final values used for the model were nhid = 7 and actfun = purelin.
```

```
> RMSE # вывод ошибки RMSE
[1] 0.1873559
> MAE # вывод ошибки MAE
[1] 0.1642089
```