**psych** - много функций для статистики. **describe** - одна из функций. **corr.test(df)** - коэффициент корреляции и p-значения.

**Название пакета::название функции(...) - применит функцию без подгрузки пакета.**

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**ggplot2**

**ggplot2 -** полезности для построения графиков. Существует сайт разработчиков с таким же названием, где есть все рекомендации, как с ним работать.

**"Hmisc"** -сюда же.

**ggplot(df, aes(x = mpg))+**

**geom\_histogram(fill = 'white', col = 'black')** - создаст гистограмму и цвета.

**geom\_dotplot()** - создаст дотплот (график из точек)

**geom\_density()** - создаст график плотности.

**facet\_grid(~subject)** - разобьёт по этому показателю. **Либо facet\_grid(subject ~ .). facet\_grid(x ~ y, margins = T)** - сделает разбивку для каждой и для всех. Используется для факторов.

**facet\_wrap(~ x + y)** - разобьёт на не связанные между собой графики. Можно указать кол-во строчек или столбцов, прописав внутри nrow = ... или ncol = ...

**geom\_smooth()** - создаст нелинейное сглаживание.

**geom\_smooth(method = 'lm')** - создаст прямую линию регрессии.

**x(y)lab('название')** - поменяет название на оси x(y).

**x(y)lim(c(число, число2))** - задаст границы меток на оси

**scale\_x(y)\_continuous(name = '...', breaks = seq(..., ..., ...), limits = c(число, число2) - name** - переименует ось, **breaks** - отобразит указанное количесто меток на оси, **limits =** задаст границы меток на оси. Также есть другие функции,

которые можно узнать по справке.

**scale\_color\_discrete(name = '...', labels = c('...', '...'))** - изменит название для переменной, изображённой цветом.

**scale\_color\_manual(values = c('цвет', 'цвет2'), name, labels)** - как discrete, только можно самому выбирать цвета.

**scale\_fill\_discrete(manual)(...)** - тоже самое, что и с цветом, но для заливки. Так можно для любого вида отображения (size(scale\_size\_continuous), shape и т.д). discrete - для факторной переменной, continuous - для числовой.

**scale\_fill(color)\_brewer(type = '', palette = число)** - позволяет выбрать палетту цветов. В справке есть ссылка на сайте, где указано, какой номер палетты отвечает за какие цвета. https://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3

**theme** - очень много различных настроек по справке.

внутри **scale(labels = scales::percent)** - переведёт десятичные дроби в проценты.

**library(grid)** - позволяет написать текст, не имеющий отношения к графику. Нужно, например, для указания источника данных.

**library(plotly) - ggplotly(переменная, в которой сохранён график, построенный в ggplot)** - сделает график интерактивным.

**plo\_ly(...)** - можно использовать по аналогии с qplot(...).

**library(ggthemes)** - добавляет дополнительные темы и скейлы. В справке есть ссылка на GitHub с подробным описанием функций пакета.

Также есть куча других расширений для этого пакета. Список https://exts.ggplot2.tidyverse.org/gallery/.

Всё, что указываем в верхнем aes-е, распростаняется на все geom-ы. Можно отдельно указать переменную под geom-ом **(geom\_point(aes(color = z)))**, тогда она будет отображаться только в нём.

Если графики накладываются друг на друга, то внутри geom-а нужно прописать **geom\_...(position = position\_dodge(значение))**

**ggplot(...)+**

**stat\_summary(fun.data = mean\_cl\_normal, geom = 'errorbar')** - построит графики доверительных интервалов для среднего значения.

В библиотеке **gvlma - функция gvlma(...)** - позволяет получить оценку выполнения основных допущений линейной регрессии. Например оценку на гомокседантичность.

**summary(gvlma(...)).**

**ggplot2 - этапы: 1. Что рисуем? (какой датафрейм, какие переменные) 2. Как рисуем? (столбцы, точки, линии и т. д.) 3. Статистическая трансформация (например сглаживание или регрессионная прямая) 4. "Масштаб" (какие именно значения**

**переменных и как они будут изображены на графике (какой "шаг" и т.д.) 5. Координаты 6. Группировка (разбивка на несколько графиков по дополнительным переменным).**

**qplot**

**qplot(x, y, data = df)** - выдаст простейший самый очевидный график, который скорее всего подойдёт для этих переменных. Можно обратиться qplot(data$x). qplot, в отличие от базовой графики, позволяет сохранят себя в переменную, через которую

можно легко этот самый график вызвать. Внутри можно указывать **(color = z, shape = v, size = I(...), alpha = I(прозрачность)) size** - размер точек (или другого), если указать без I, то цифра размера будет указана сбоку, что не несёт в

себе никакой информации по теме.

Если написать **(color = I('цвет'))** - то все точки станут этого цвета.

**(color, fill, shape, size, alpha)**

В qplot можно использовать любой geom. **qplot(x, data = df, geom = '...').**

**ggplot(df, aes(x, y)+**

**stat\_summary(fun.data = mean\_cl\_boot, geom = 'errorbar')** - нарисует график по трансформированным значениям (среднее значение y, а также минимальная и максимальная доверительная зона, это делает функция mean\_cl\_boot)

Очень много деталей в R.часть 2 "подробнее о визуализации"

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**dplyr**

**library(dplyr)** - одна из особенностей - позволяет использовать переменные, которые ещё не созданы (но объявлены внутри df), для создания других переменных. Например data\_frame(x = rnorm(10), y = abs(x)). В базовом R-овском датафрейме мы

бы не смогли сразу обратиться к иксу.

**select(df, ...(те столбцы, которые хотим отобрать)).** Например **select(diamonds, cut, price).** Если хотим все от cut до price, то cut:price. Если хотим убрать колонку (напрмер cut), то **select(df, -cut).** Также можно обращаться по номерам.

Много полезных особенностей можно узнать по справке. Внутри select можно использовать аргумент **contains('символы')** - выдаст только те результаты, где есть необходимые символы.

Внутри функции можно использовать аргумент **starts\_with('буква, на которую должна начинаться переменная')** - выдаст те переменные, которые начинаются на нужную букву.

**slice(df, номер строчки)** - выберает определённую строчку.

**filter(df, условие(например такая-то переменная больше стольки-то или Name %in% c(x, y))** - выдаст данные, соответствующие условию. Например **filter(mtcars, am == 'manual').**

**arrange(diamonds, price, depth)** - отсортирует df по цене (от меньшей к большей) и будет пытаться также отсортировать столбец depth. **arrange(df, desc(x))** - от больших к меньшим.

**rename(df, 'new X' = X)** - переименовывает переменную. Если в новом названии есть пробел, тогда его нужно заключить в кавычки.

**mutate(df, sqrt\_X = sqrt(X))** - создаст в df новую переменную, в которой будут квадратные корни из X. Также можно не только создавать новые переменные, но и менять старые. Например **mutate(mtcars, am = factor(am), vs = factor(vs))** - превратит

am и vs - в факторы.

**mutate\_each(df, funs(действие))** - совершит действие над каждой колонкой. Если хотим под funs написать свою функцию (например каждый столбик умножить на 2), то **mutate\_each(df, funs(. \* 2)). mutate\_each(df, funs(ifelse(. < 0, 0, .)))** -

если значение меньше нуля, превратит его в ноль, если не меньше, то вернёт первоначальное значение.

**group\_by(df, x(столбик))** - создаёт датафрейм такой же, как исходный, но со сгруппированными данными. Например: **gr\_diamonds <- group\_by(diamonds, cut)** в diamonds переменная cut имеет несколько градаций (fair, good, very good, premium,

ideal). После **group\_by** любое действие будет применяться к каждой из этих градаций. Группировать можно сразу по нескольким столбцам. Результаты применений функций будут показаны для каждой градации каждого столбца.

**sumarise(df, функция(напрмер mean))** - получает на вход df, применяет действие ко всему df и выдаёт только одно значение. **summarise** очень часто используется вместе с **group\_by,** т.к. позволяет узнать какой-либо показатель для всех градаций

переменной. Например: **summarise(gr\_diamonds, mean(price))** - выдаст средние цены для каждой градации переменной cut. Если в **summarise** использовать какие-либо функции, определяющие **p-уровень** значимости и другие параметры, то в функции **необходимо** указать, какой именно из параметров мы хотим увидеть. Например **summarise(shapiro.test(x)$p)** – выдаст **p-уровень** значимости при проверке на нормальность.

**n()** - (именно в таком виде, с пустыми скобками) внутри функции summarise выдаст количество наблюдений для каждой градации.

**do({})** - сделает функцию. Подробности по справке.

**summarise\_all(df, funs(...))** - выполнит функцию каждого столбца.

**mutate\_if(df, условие, действие)** - сделает mutate с каждой ячейчкой, только если для ячейки выполняется условие.

**%>%** - передест предыдущие данные в следующее действие.

Если мы используем вектор с именами, то у каждой функции есть аналоги, которые отличаются нижним подчёркиванием, например **summarise\_(.dots = list(...)).** В .dots вписываем наши имена или переменную, содержащую их список.

**transmute(df, ...)** - То же, что и mutate, но с удалением исходных колонок

**sample\_n(df, число)** - Дата фрейм, состоящий из случайных рядов исходного

**inner\_join(x, y, by = ...)** - Объединение двух дата фреймов по ключу; берутся только значения по ключам, встречающимся в обоих дата фреймах

**near(x, y)** – безопасный способ сравнить две переменных без вероятнсти ошибки из-за плавающей точки.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**data.table**

**library(data.table)** - особенности в том, что функции этой библиотеки очень быстро работают с большими объёмами данных.

**fread('product.csv')** - создаст datatable из данных product. Считывает данные намного быстрее, чем read.table.

В data.table **products[price > 10000]** выполнится сразу, не нужно излишнего обращения, которое было в стандартном R **(products[products$price > 10000])**. Если хотим отобрать по нескольким показателям, то используем знак &. **products[price**

**> 10000 & brand %in% c('Epson', 'Apple')]**

Если нам нужны определённые названия фирм то используем оператор **%in%** и название фирмы. Например: **products[brand %in% c('Epson', 'Apple')]** - выдаст только бренды Epson и Apple.

Если хотим отфильтровать показатель, который имеет логическую форму (ответы TRUE или FALSE), для обращения обязательно нужно указать, что нам нужно T или F. **products(available == T).**

**data.table[3]** - вернёт третью строку, а если бы это был data.frame, он бы вернул 3 столбец.

Если нам нужно всё, кроме определённых значений, нужно поставить !. Например **products[!(1:10)]** - выдаст всё, кроме первых десяти строчек.

**dt[, list(x, y и, например, действие с z (например z / 1000)]** - выдаст все строки для столбцов x и y, а также преобразует z и выдаст значения. Слово **list** можно заменить на **. (точку).**

**order** - расставляет значения в порядке возрастания или убывания. **products[order(price, decreasing = T)]** - выставит цены в порядке убывания.

**products[order(price, decreasing = T), list(name, price.1k = paste0(price / 1000, 'тыс. руб.'))] -** выдаст наименования и цены (в тысячах рублей) в порядке убывания. Пункт **paste0(price / 1000, 'тыс. руб.)** - паст исчезнет, а к строкам в

столбце price добавится тыс. руб.

**products[, .(price = sum(price))] -** вернёт сумму цен всех товаров в каталоге.

**products[, .(name.with.brand = paste0(brand, '-', name))][order(name.with.brand)] -** выдаст бренды с названием товара через тире, бренды будут расположены по порядку.

products[, .(price = { - выдаст минимальную и максимальную цену, а также отношения средней цены к медиане.

a <- mean(price)

b <- median(price)

c(min(price), max(price), a/b)

})]

products[, .(mean.price = mean(price)), by = brand] - выдаст среднюю цену для всех брендов. (агрегация).

products[order(-price), .(name = head(name, 3), price = head(price, 3)), by = brand] - выдаст три самых дорогих товара для каждого бренда.

filter.expensive.available <- function(products, brands) {

products[price >= 5000 & brand %in% brands & available[available = T]]

} - принимает на вход products (объект типа data.table) и вектор названий брендов, и возвращает только те строчки, которые соответствуют товарам, цена

которых больше или равна 5000 рублей, доступны на складе, и принадлежат одному из переданных брендов.

purchases.median.order.price <- function(purchases) {

as.numeric(purchases[quantity >= 0][, .(sum = sum(price \* quantity)), by = ordernumber][, .(median(sum))])

} - возвращает медианную стоимость заказа (число).Группировку стоит проводить с помощью data.table. Записи с неположительным количеством купленных товаров (возвраты)

игнорировать. Обратите внимание, что одному заказу может соответствовать несколько записей – «позиций» с одинаковым ordernumber, и что при расчете стоимости заказа надо учитывать ситуации, когда пользователь купил несколько товаров

одного типа (их количество указано в quantity).

dt[действие, head(.SD, 3), by = ...] - три верхних пункта для каждого столбца.

**dt(..., .(name = .N)] - .N** - количество элементов в группе.

**Оператор := с**оздаёт/изменяет столбец без создания копии объекта (если делать без него, может не хватить оперативной памяти). **x[, new.column := expr]. x[i, new.column := expr, by = ...]. x[, c('col1', 'col2') := list(expr1, expr2)]**

**merge(dt1, dt2, by = 'название колонки'**) - объединит два дататейбла в один по совпадениям в столбце. Если названия стобцов в дататейблах разные, тогда нужно использовать **merge(dt1, dt2, by.x = 'название1' , by.y = 'название2')**

Также можно это сделать через квадратные скобки **dt1[dt2, on = 'название общего столбца']**

**merge(dt1, dt2, all = T)** - объединит два в один, значения, которые не совпадают заменятся на NA.

**merge(dt1, dt2)** - объединит по совпадающим колонкам и значениям.

Для подсчёта уникальных значений (например число покупателей | пользователей) в data.table можно использовать **uniqueN().**

purchases.with.brands <- merge(purchases, products[, list(product\_id, brand)], by = 'product\_id')

top.20.brands <- head(purchases.with.brands[, .(total.brand.users = uniqueN(externalsessionid)), by = brand][order(-total.brand.users)], 20) - выдаст двадцать самых популярных брендов и количество уникальных пользователей этих брендов.

purchases.with.brands <- merge(purchases, products[, list(product\_id, brand)], by = 'product\_id')

top.20.brands <- head(purchases.with.brands[, .(total.brand.users = uniqueN(externalsessionid)), by = brand][order(-total.brand.users)], 20)

users <- purchases.with.brands[, .(unique.brands = uniqueN(brand),

items = .N,

brand = brand[1]),

by = externalsessionid]

brand.loyal.users <- users[items > 1][unique.brands == 1][, .(total.loyal.orders = .N), by = brand]

brand.stats <- merge(top.20.brands, brand.loyal.users, by = 'brand')

brand.stats[order(-total.brand.users)] - выдаст топ 20 популярных брендов в порядке убывания количества пользователей бренда.

setkeys(dt, название столбца) - задаст ключ, который позволяет быстро совершать merge с большими объёмами данных.

mark.position.portion <- function(purchases) {

q <- purchases[quantity > 0]

q[, price.portion := format(round(price / sum(price \* quantity) \* quantity \* 100, 2), nsmall = 2), by = ordernumber][]

} - функцию, которая будет с помощью := добавлять столбец «price.portion», содержащий процент стоимости товара в заказе, с двумя знаками после запятой (нули после запятой

не опускать). Проверяться будет возвращаемая из функции таблица. Тип нового столбца - character (строка). Записи с неположительным количеством товаров убрать перед расчётом.Например: если в заказе есть три позиции: 1 ед. по 100р,

1 ед. по 300 руб. и 2 ед. по 50 р., в столбце должно быть «20.00, 60.00, 20.00

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**rmarkdown -** средство для создания динамических документов, отчётов и презентаций. Интсрумент для встраивания результатов исполнения кода на R в текст документа. Это нужно для: 1) полной воспроизводимости результатов, 2) для конвертирования

исходного .Rmd файла в разные форматы (HTML, PDF, MS Word, Markdown, различные презентации, шаблоны для вебсайтов), 3) для удобства написания, разметки и форматиррования документа, 4) для распростарнения открытого кода, идей, алгоритмов

и результатов исследования.

**File -> new file -> R markdown** - создаст маркдаун. В нём прописываем код текст, код, дальше жмём '**knit'**, после чего увидим, как всё будет выглядеть в конечном формате. В конечном формате будет отображён результат работы кода.

Кусочки кода называются чанками.

текст `r код R` текст. - вставит результат кода прямо в текст.

```{r} - синтаксис чанков. В конечно файле будет написана функция и результат. Если внутри фигурных скобок прописать {r. echo = F} - то выдаст только результат кода, но не сам код. Справой стороны знак шестерёнки позволит выбрать, что

войдёт в конечный файл.

код R

```

Markdown: HTML - набор инструкций для браузера. Написание HTML-страниц в ручном режиме не очень удобно. Markdown - текстовый формат, удобный для написания и чтения, который легко конвертировать в HTML.

Основные элементы Markdown:

\*\*текст\*\* -> жирный шрифт. \*текст\* -> курсивный шрифт. `sum(x)` -> sum(x) - моноширинный шрифт. A^2^~i~ -> верхние и нижние индексы. ~~текст~~ - зачёркнутый текст. \\* \\ \$ -> \* \ $ - экранирование символов. -- --- - автозамена тире и

длинного тире, они станут без пропусков, без штрихов.

# Заголовок 1 - ###### Заголовок 6.

<https//...> -> https://... - гиперссылка.

[Текст](https://...) - 'Текст', который превратился в гиперссылку.

http://www.rsudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf - ссылка на все основные элементы.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**library(tidyr)**

**separate(df, колонка, c('название одной из новых колонок', 'название одной из новых колонок'), sep = ' x ')** - разделит одну колонку на несколько. x - тот знак, который будет разделять.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**library(stringr)**

**str\_extract(текст(вектор, df), 'символ.')** - вернёт символ и следующий за ним символ. Но только один первый.

**str\_extract\_all(текст(вектор, df), 'символ.')** - вернёт все, а не только первый.

**str\_replace(текст, 'символ', 'символ2')** - первый символ, который **первым** попадётся в тексте, будет заменён на второй символ.

**str\_replace\_all(текст, 'символ', 'символ2')** - заменит **все**, а не первый.

**str\_length(...)** - количество символов в тексте. Пробел считается за символ.

**str\_replace(df$x, '[:digit:]+', '')** - удалит цифры из переменной, оставит только буквы.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**library(stringi)**

**stri\_reverse(xyz)** - развернёт напечатанное. Выдаст zyx.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**library(microbenchmark)**

**microbenchmark(x, y)** - как system.time, но лучше. Выдаст статистику по затраченному времени на выполнение функций.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**library(tidyr)**

**gather(df, 'Название', 'Название2', -'Название3')** - из 'широкого' в 'длинный' формат **(wide to long)**. Разобьёт на другие переменные.

**spread(...)** - обратная функция для **gather**.

**separate(df, 'Название переменной, которую делим', c('название1', 'название2'), 'знак, по которому происходит разделение')** - разобьёт переменную на две других.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**library(tibble)**

**add\_column(df, название колонки (например можно paste0(‘название’, номера))** – добавит новые колонки в датафрейм.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Shiny – для работы с R в браузере.**

[**https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/index.html**](https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/index.html)

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Функция bptest, используемая в видео для тестирования данных на предмет **гетероскедастичности**, находится в пакете [lmtest](https://cran.r-project.org/package=lmtest" \t "_blank).

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

library(car)

vif(df) – расчитает показатель VIF.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

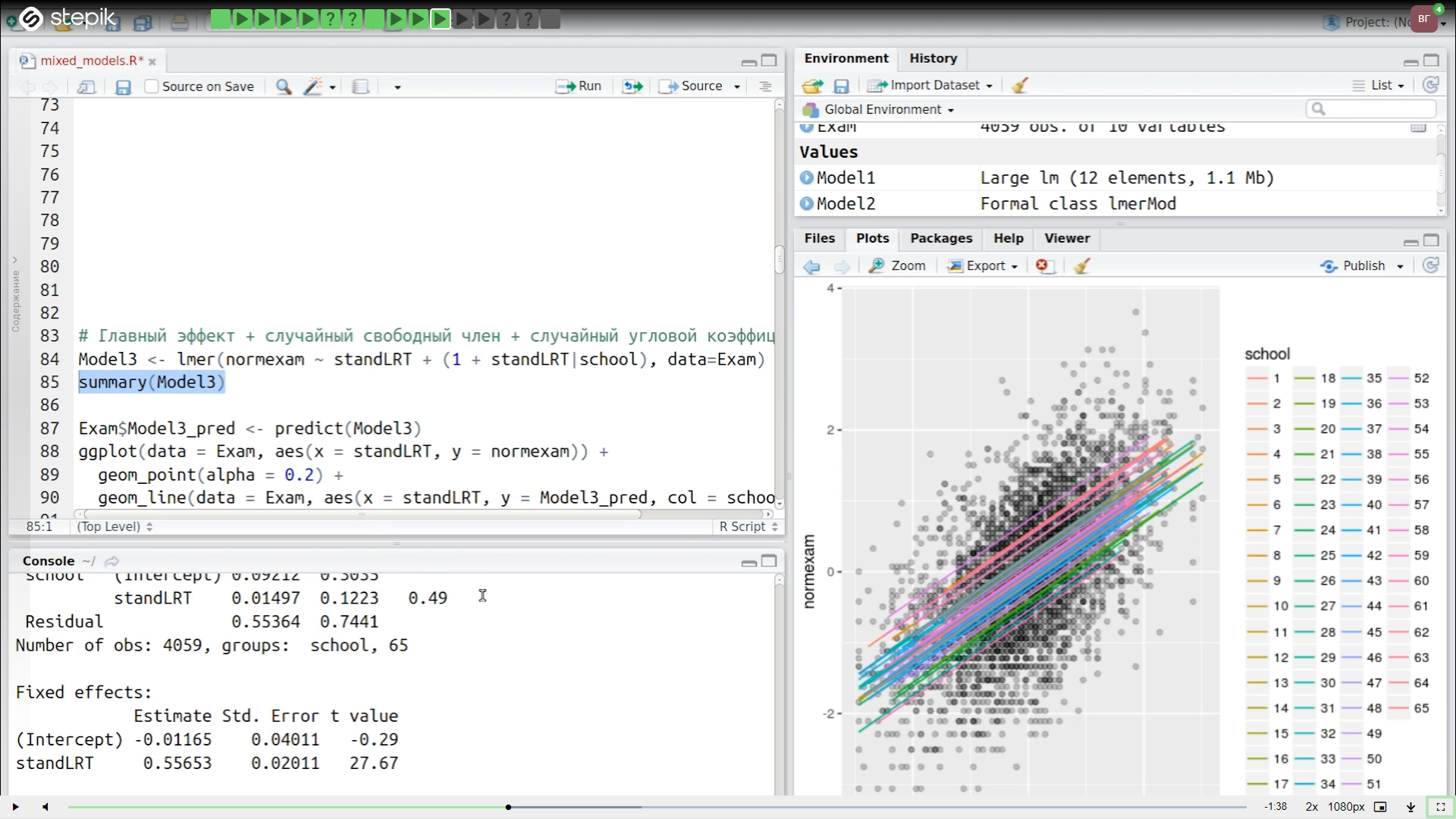
library(lme4) – пакет для создания смешанных регрессионных моделей.

lmer(зависимая переменная ~ проверяемая независимая переменная + (свободный член(в примере была 1) + проверяемая независимая переменная | рандомный эффект), df). Можно указывать без повторного применения проверяемой независимой переменной (+ (1 | рандомный эффект)), но тогда не будут учитываться угловые коээфициенты и на графике это будет выглядеть, как параллельные линии на разной высоте. А если всё-же указать, то угол наклона линий будет отличаться.

Если нам не важен свободный член в модели, но важен угол наклона, тогда вместо единицы нужно указать 0.

summary к этой модели не показывает p-уровень, но показывает t-значение. Обычно если t значение больше 2-х, то эффект будет значимым (как при p < 0.05), но это не всегда так.

График:



Если мы точно знаем, что корреляции между случайными эффектами нет, тогда модель должна принять вид:

lmer(x ~ y, (1|рандомный эффект) + (0 + y|рандомный эффект), df).

Чтобы понять, какая модель лучше предскажет: с независимой переменной или без неё, нужно создать две модели, в каждой внутри прописать lmer(…, REML = FALSE,…), вторая модель будет intercept only model lmer(x ~ вместо независимой переменной просто единица,…). Далее anova(модель один, модель два) – там уже будет показано по p, есть ли статистически значимая разница между моделями, а по AIC и BIC можно судить, какая модель лучше предсказывает, обычно чем эти показатели меньше, тем модель точнее, тем выше её относительное качество.

Для логистической регрессии команда glmer(…, family = ‘binomial’,…). Там в summary уже есть p-уровень значимости.

Predict(model, new\_df, allow.new.levels = T) – для предсказания на новой модели, а не существующей.

fixef(model) – коэффициенты для углового коэффициента (intercept) и для фиксированного.

ranef(model) – коэффициенты для случайного эффекта (intercept) и для углового коэффициента (фиксированного)???

Среднее от случайных эффектов равно нулю.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

library(mlmRev)

data(‘Exam’) – данные, в которых содержится различная информация об учениках 65 лондонских школ.