Оглавление

8.3. Создание дашбордов с Quarto 2
8.3.1. Начало работы с Quarto 2
8.3.2. Создание дашборда 5
8.3.3. Формирование сетки компоновки элементов 6
8.3.4. Элементы дашборда 9
8.3.5. Публикация дашборда 24
8.3.6. Публикация с помощью Github Actions 26

8.3. Создание дашбордов с Quarto

8.3.1. Начало работы с Quarto

Дальнейшим развитием идей RMarkdown стал проект Quarto (https://quarto.org). Он объединил в себе все возможности RMarkdown, расширив его благодаря использованию новых языков и технологий. В результате получилась полноценная система публикации, позволяющая создавать отчеты, статьи, презентации, информационные панели, веб-сайты, блоги и книги в форматах HTML, PDF, MS Word, ePub и других.

Возможности не ограничиваются R и RStudio: можно использовать как другие языки программирования (R, Python, Julia, Observable JS), так и другие средства разработки (RStudio, Jupiter, VS Code). Система также поддерживает внешние расширения, что позволяет постоянно наращивать функциональные возможности.

Используемый в Quarto язык разметки Markdown включает также дополнительные возможности: поддерживает запись математических формул с использованием синтаксиса TeX, диаграммы Mermaid и Graphviz, примечания, сообщения и сноски. Подробнее о возможностях Markdown в Quarto можно ознакомиться по ссылке https://quarto.org/docs/authoring/markdown-basics.html.

Таким образом, многие возможности R и Markdown, ранее доступные исключительно пользователям R, теперь адаптированы для тех, кто работает с Python и Julia, занимаясь анализом данных, научной работой и преподаванием.

С помощью Quarto можно:

- Сохранять результаты анализа данных в различных форматах: HTML, pdf, ePub, Word, PowerPoint и т.д. для единого источника публикации.
- Использовать разметку, включающую формулы, цитаты, перекрестные ссылки, расширенные макеты, интерактивные элементы и многое другое.
- Публиковать результаты в Интернете с помощью различных сервисов, таких как Posit Cloud, Github Pages и другие.
- Создавать интерактивные книги, которые можно переводить различные форматы, пригодные для печати и электронных носителей.
- Создавать сайты, включая блоги и техническую документацию.

С примерами проектов, созданных с помощью Quarto можно ознакомиться по ссылке https://quarto.org/docs/gallery. Примечательно, что практически любые документы, созданные с помощью RMarkdown, с легкостью могут быть перенесены в Quarto.

Для начала работы систему необходимо установить согласно инструкции https://quarto.org/docs/get-started. Она не является непосредственно пакетом для R,

скорее это набор инструментов командной строки Quarto CLI, который может работать как с RStudio, так и без него. Для использования с некоторыми средами разработки, например VS Code, может потребоваться специальное расширение. В представленном далее примере будет использована среда разработки RStudio, в которой уже есть все необходимое.

Для начала работы в RStudio необходимо создать новый проект Quarto с помощью меню File -> New Project -> Quarto Project.

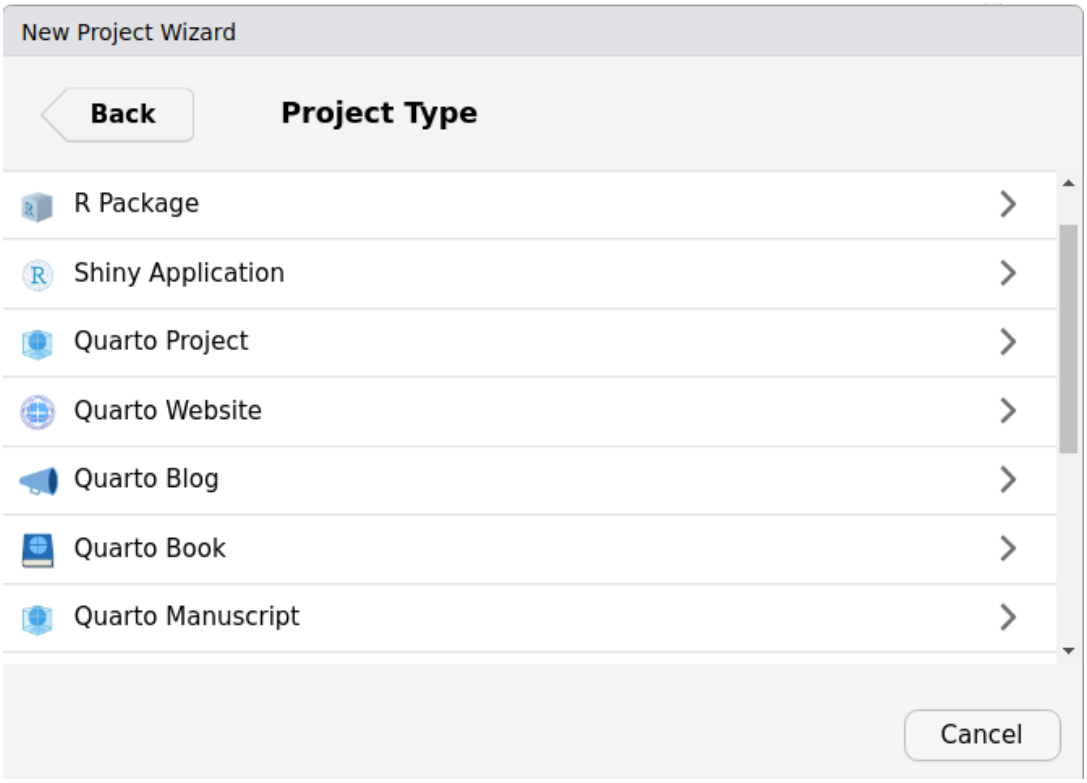


Рисунок 8.140. Меню для создания нового проекта RStudio

Далее необходимо выбрать папку, где будет создан проект и отметить галочкой создание репозитория git (подробнее о git **см. раздел 8.1.**).

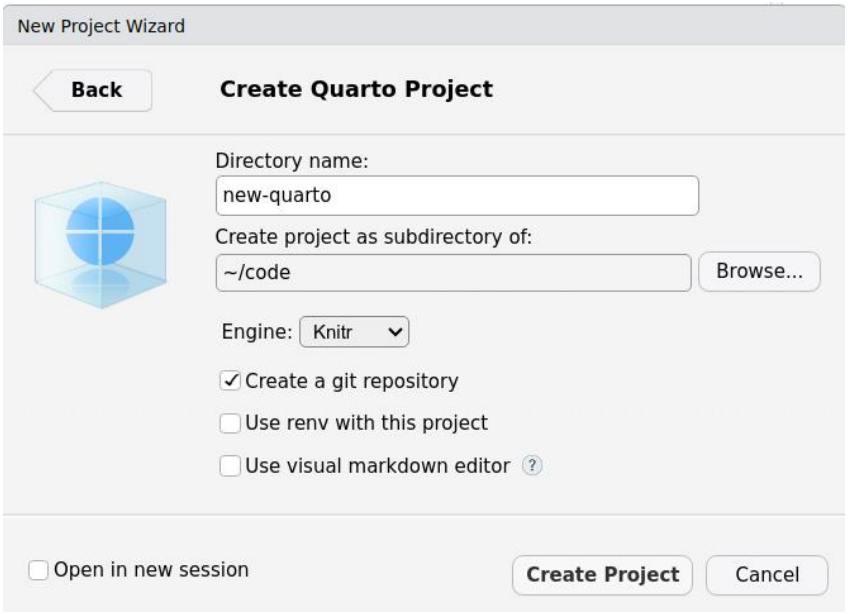


Рисунок 8.141. Создание нового проекта Quarto

В результате будет создана папка проекта со следующим содержимым:

- *_quarto.yml* - файл с настройками документа quarto;
- *new-quarto.qmd* - основной файл, в котором будет содержимое документа.

Если открыть файл new-quarto.qmd, то можно увидеть, что структура данного файла очень похожа на уже знакомый RMarkdown-документ: он состоит из yaml-преамбулы и основного Markdown содержимого с внедренными чанками кода на R.

````
---
title: "new-quarto"
---

## Quarto

Quarto enables you to weave together content and executable code into a finished
document. To learn more about Quarto see .

```{r}
1 + 1
```
````

Данный документ можно скомпилировать или "отрендерить" с помощью кнопки “Render” (**рисунок 8.142.**).

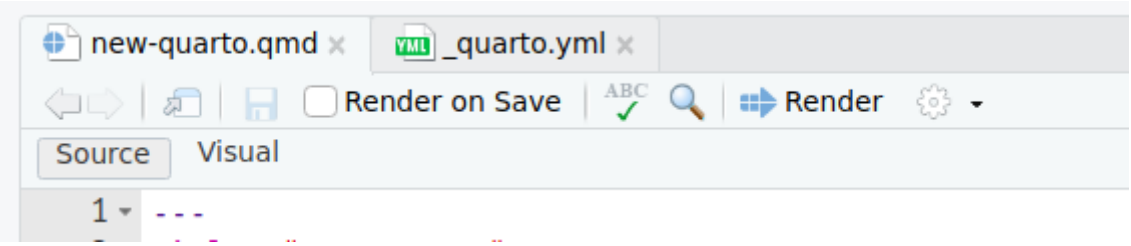


Рисунок 8.142. Основные рабочие инструменты при работе с Quarto в RStudio

В результате компиляции в браузере откроется следующая страница (**рисунок 8.143.**).

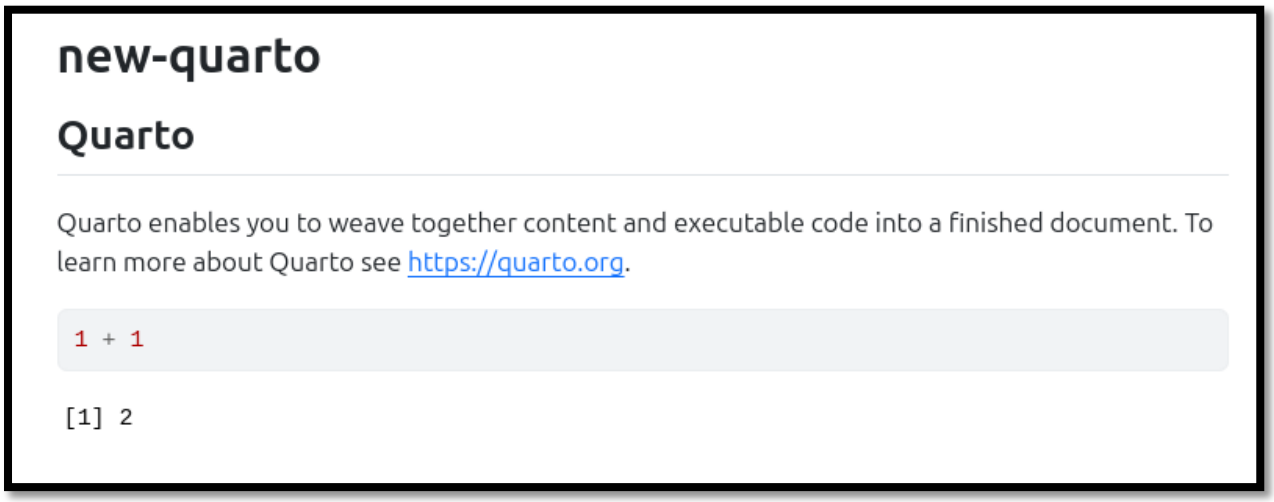


Рисунок 8.143. Результат компиляции (рендеринга) проекта Quarto с тестовым содержимым

После рендеринга в папке проекта появится файл new-quarto.html, который и будет являться результатом работы Quarto. Таким образом, был создан первый документ на Quarto.

8.3.2. Создание дашборда

Для того, чтобы изучить особенности работы с разметкой Quarto и ее отличие от RMarkdown далее будет создан дашборд-страница, которая на одном экране представляет ключевые результаты исследования: заголовок, текстовое описание, таблицы и графики.

Обычно дашборд состоит из следующих компонентов:

- Заголовок — содержит иконку, заголовок, автора дашборда. Может содержать ссылки для перехода на страницы и дополнительные ссылки.
- Страницы — место для размещения содержимого. Для упорядочивания элементов на странице можно использовать виртуальную сетку, которая определяется строками, столбцами, вкладками. Эта компоновка задается уровнями заголовков Markdown.
- Карточки - размещаются в ячейках сетки и содержат в себе текст, графики, таблицы и так далее.

Подробно о создании дашбордов с помощью Quarto можно ознакомиться по ссылке https://quarto.org/docs/dashboards. С примерами созданных дашбордов можно ознакомиться в галерее https://quarto.org/docs/gallery/#dashboards.

Для начала работы с дашбордом прежде всего необходимо заполнить преамбулу: указать, что следует создать именно документ в виде дашборда, заполнить название дашборда и автора. Можно также разместить иконки и дополнительные ссылки, которые будут размещены в правой части заголовка. По умолчанию используются иконки из набора Bootstrap https://icons.getbootstrap.com.

Далее необходимо задать цветовую тему оформления дашборда (в данном примере - *cosmo*). Список тем оформления доступен по ссылке https://quarto.org/docs/dashboards/theming.html.

В итоге преамбула файла new-quarto.qmd выглядит следующим образом.

```
title: "Новый красивый дашборд"
author: "Data Science Course"
format:
  dashboard:
    theme:
    - cosmo
    logo: logo.png
    nav-buttons:
    - icon: github
      href: https://github.com/data-science-course/quarto-dashboard
```

Если после модификации преамбулы нажать кнопку “Render”, то можно увидеть, что в документе добавился заголовок.

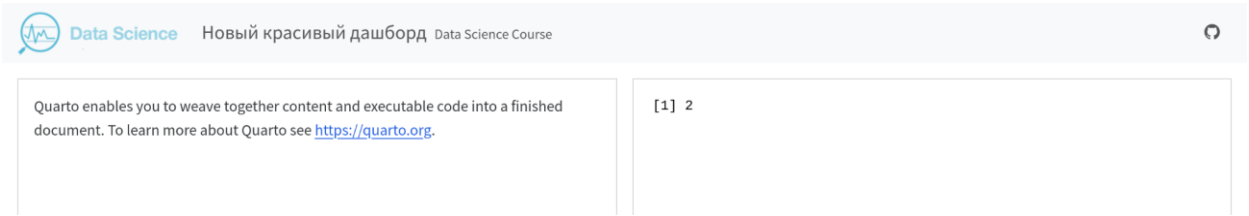


Рисунок 8.144. Результат компиляции (рендеринга) проекта Quarto после модификации преамбулы в файле new-quarto.qmd

8.3.3. Формирование сетки компоновки элементов

Можно заметить, что текст и код на R тоже поменяли свое расположение — раньше они были друг под другом, а теперь занимают левую и правую половины страницы.

Это произошло, потому что был установлен формат документа *dashboard*. Подразумевается, что все условные элементы (информационные части) документа должны помещаться на страницу и размещаться в рамках сетки. Таким образом, блок текста и блок кода заняли два столбца равной ширины. Сетку размещения можно определять с помощью заголовков в формате Markdown:

- Заголовок первого уровня определяет страницу.
- Заголовок второго уровня определяет строку на странице. При размещении нескольких строк их высота по умолчанию будет одинакова, чтобы заполнить все пространство страницы. Элементы, расположенные внутри строки при рендеринге размещаются по горизонтали слева направо.
- Заголовок третьего уровня определяет столбец внутри строки. При этом ширина каждого столбца по умолчанию одинакова. Элементы, размещенные внутри столбца размещаются вертикально сверху вниз.

Например, можно определить страницу с названием Дашборд, которая будет включать в себя сетку из двух строк: в первой будет два столбца, а во второй - три.

```
# Дашборд

## Строка 1

### Столбец 1

Содержимое строки 1 столбца 1

### Столбец 2

Содержимое строки 1 столбца 2

## Строка 2

### Столбец 1

Содержимое строки 2 столбца 1

### Столбец 2

Содержимое строки 2 столбца 2

### Столбец 3

Содержимое строки 2 столбца 3
```

При рендеринге данного документа получается следующий результат.

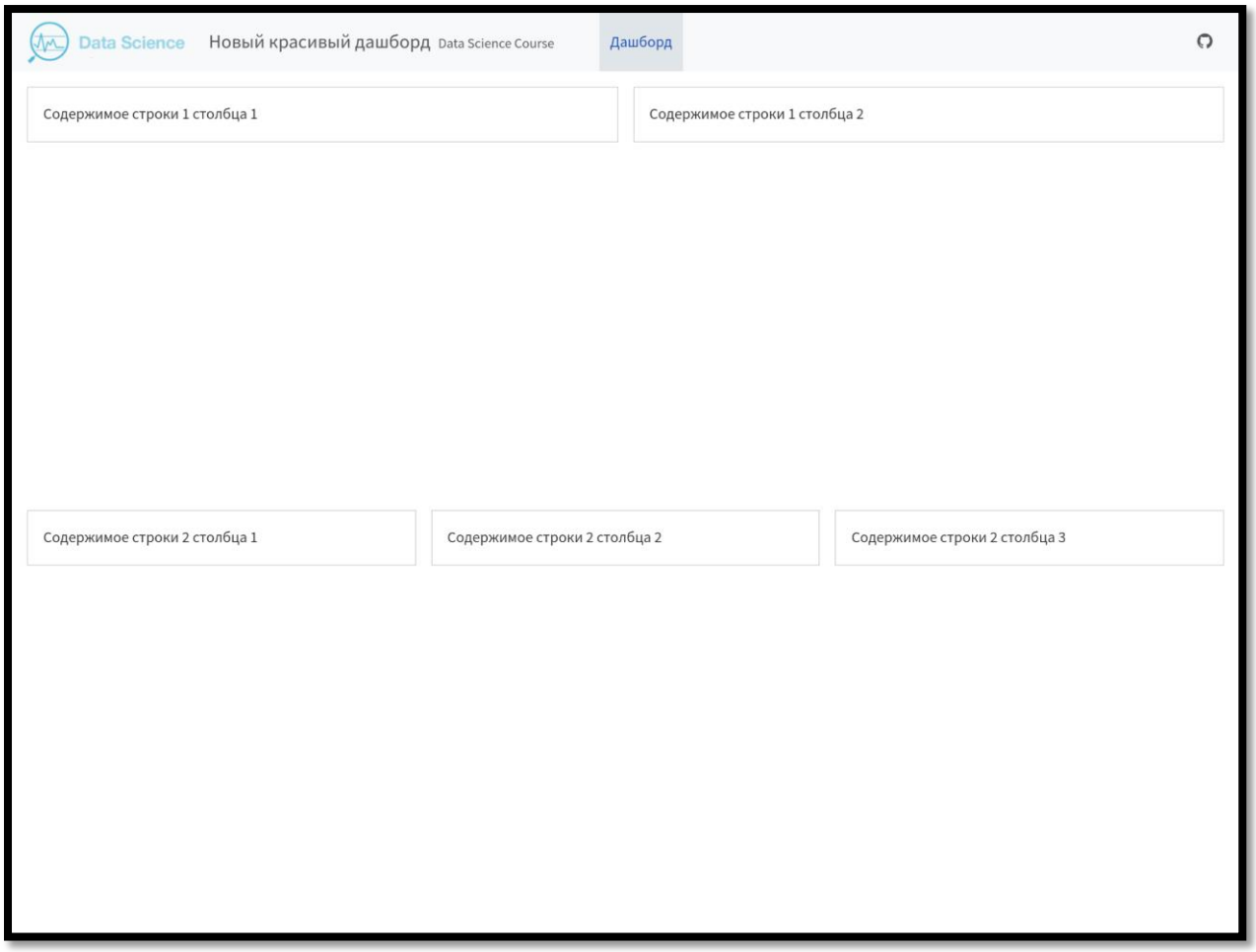


Рисунок 8.145. Результат компиляции (рендеринга) проекта Quarto с настроенной сеткой элементов

При необходимости можно задать необходимую ширину и высоту вручную:

- для столбцов ширина устанавливается
  ### Column {width="50%"}
- для строк высота устанавливается
  ## Row {height="50%"}

Стоит иметь в виду, что уровень вложенности элементов может быть достаточно глубоким: внутри столбцов можно размещать дополнительные строки, внутри которых также будут столбцы и так далее. Таким образом, можно достигать достаточно сложной сетки размещения элементов. При множественной вложенности имеет смысл задавать прямое указание, какой именно контейнер используется - строка (Row) или столбец (Column).

Далее будет подготовлена сетку для дашборда.

```
# Дашборд

## Row
```

```
### Column

#### Row {height="15%"}

##### Column

Количество мужчин

##### Column

Количество женщин

##### Column

Количество детей

#### Row {height="25%"}

Графики структуры организмов для мужчин, женщин и детей

#### Row {.fill}

График со структурой диагнозов по группам пациентов

### Column

#### Row {height="33%"}

Карта

#### Row

Таблица, иллюстрирующая карту
```

В результате получится следующая заготовка.

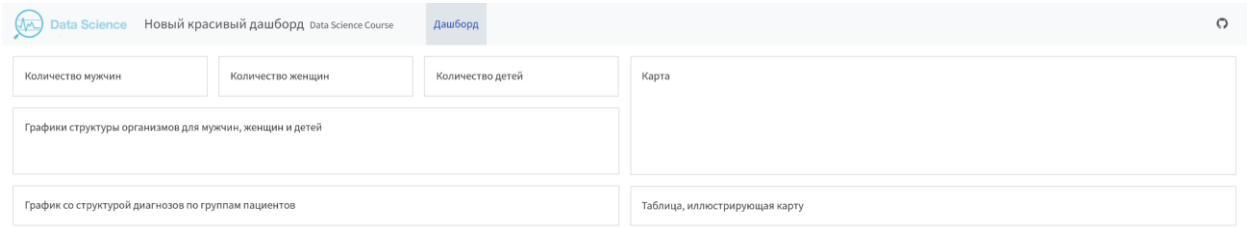


Рисунок 8.146. Дашборд с настроенной сеткой элементов

Теперь необходимо наполнить данную сетку элементами инфографики.

8.3.4. Элементы дашборда

Для наполнения дашборда содержимым можно воспользоваться следующими компонентами.

Блоки кода

Блоки кода или чанки, также как в RMarkdown могут размещаться прямо в тексте документа. Данный код будет выполняться при рендеринге документа, а также его можно запускать с помощью RStudio.

```
```{r}
#| label: example
#| echo: true
#| output: true
1 + 1
```
```

В зависимости от дополнительных параметров можно определять показ самого кода, результатов, информационных сообщений в итоговом документе.

В примере выше указано, что нужно выводить как исходный текст кода (*echo true*) так и результат его выполнения (*output: true*). Параметр *label* позволяет ссылаться на данный фрагмент кода в документе с помощью символа “@”, а также позволяет быстро переходить к нужному блоку кода с помощью RStudio.

Возможны следующие параметры

- *eval* - выполнять код при рендеринге. По умолчанию код выполняется всегда. Если установить значение *false*, то код выполняться не будет, но будет выводиться в итоговый документ.
- *echo* - вывод кода в документ.
- *output* - выводить результат выполнения в документ. По умолчанию выводится всегда.
- *warning* - включить предупреждения в вывод результата.
- *error* - включить сообщения об ошибках в вывод результата.
- *include* - если указать этот параметр как *false*, то любые сообщения и результат не будет выводиться в итоговый документ. Обычно этот параметр используется в случаях, если в коде подключаются различные пакеты, либо выполняются вычисления, которые не относятся напрямую к выводу результата: чтение файлов, подготовка таблиц и так далее.

Пример блока чтения файла выглядит следующим образом.

```
```{r}
#| label: Загрузка данных
#| include: false
patients <- read.csv2("patients.csv", dec = ".")
```
```

Обычно такие блоки размещают в самом начале файла, еще до описания разметки, потому что они носят общий характер и не относятся к какому-либо конкретному элементу.

Карточки значений

Данный компонент позволяет отобразить одно число с подписью и иконкой. Для блока можно задать цвет - он определяется цветами темы дашборда: *primary*, *secondary*, *success*, *info*, *warning*, *danger*, *light*, *dark*, но также можно задавать цвет в шестнадцатеричном формате, например *#ff7f0e*.

Например, можно разместить в дашборде карточки, отражающие количество пациентов. Сначала необходимо рассчитать необходимые значения.

```{r}
#| label: Количество пациентов по полу
#| include: false
mens_count <- patients %>% filter(grepl("Мужчины",PAT_GROUP)) %>% nrow()
womens_count <- patients %>% filter(grepl("Женщины",PAT_GROUP)) %>% nrow()
children_count <- patients %>% filter(grepl("Дети",PAT_GROUP)) %>% nrow()
```

Затем необходимо вывести каждое значение в карточку. Можно заметить, что в данном чанке появилось поле content, которое определяет, что создается элемент *valuebox* - карточка значений. В поле *title* указывается заголовок карточки. Стили оформления и непосредственно значение передается в виде списка.

```{r}
#| label: Кол-во мужчин
#| content: valuebox
#| title: "Мужчины"
list(
  icon = "person-standing",
  color = "primary",
  value = mens_count
)
```

Рекомендуется размещать каждый такой *valuebox* в отдельном столбце в рамках строки. После размещения трех таких блоков, дашборд будет выглядеть следующим образом.

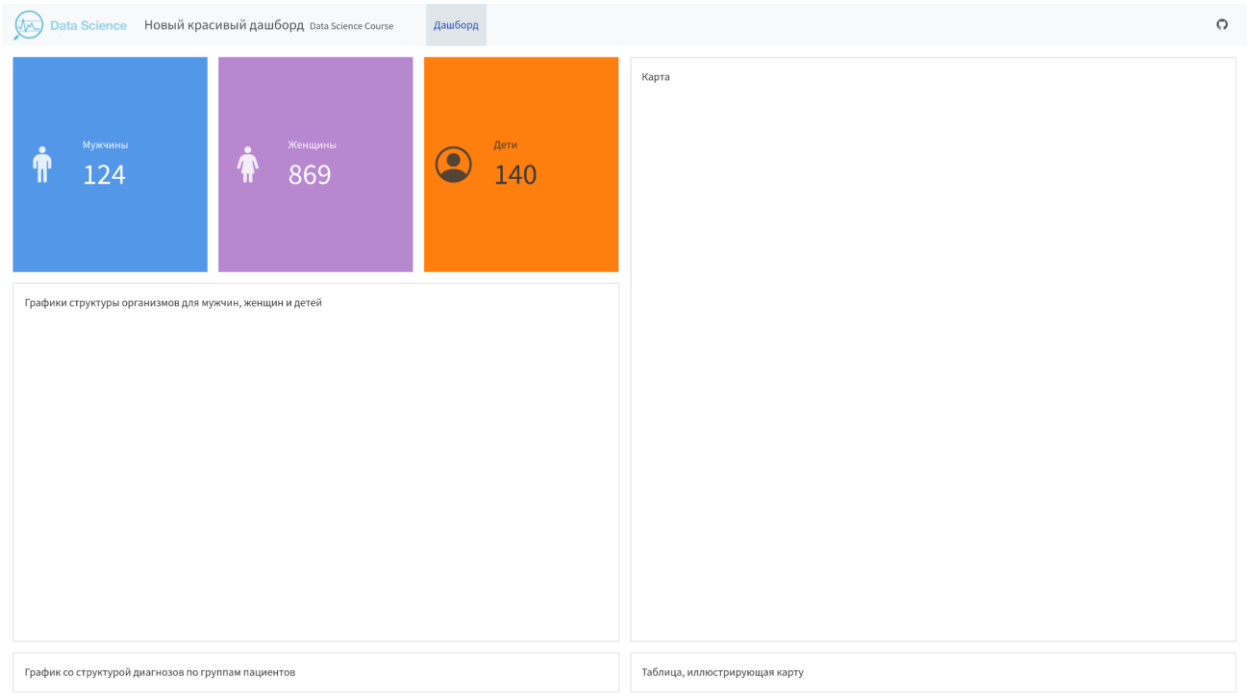


Рисунок 8.147. Дашборд с настроенными карточками значений

Графики ggplot

Для размещения графиков *ggplot* на дашборде необходимо создать график в рамках чанка и отправить его на вывод. Например, можно разместить под каждой карточкой количества пациентов соответствующую структуру выделенных у них микроорганизмов. Сначала необходимо рассчитать значения для каждой группы пациентов.

```
```{r}
#| label: Подсчет количества организмов
#| include: false

org_male <- patients %>%
  filter(grepl("Мужчины",PAT_GROUP)) %>%
  group_by(STRAIN) %>%
  summarise(Count = n()) %>%
  ungroup() %>%
  mutate(Percent = round(100 * Count / sum(Count))) %>%
  arrange(desc(Percent)) %>%
   mutate(csum = rev(cumsum(rev(Count))),
          pos = Count/2 + lead(csum, 1),
          pos = if_else(is.na(pos), Count/2, pos))
```
```

Данная таблица будет использоваться для формирования графика *ggplot* в виде пирога. Сам код графика представлен ниже. При его описании используется новый атрибут *fig-cap* - он позволяет определить подпись к графику, которая будет отображаться на дашборде.

```
```{r}
```
```

```
#| label: Подсчет количества организмов у мужчин
#| fig-cap: "Структура организмов у мужчин"
ggplot(org_male, aes(x = "" , y = Count, fill = fct_inorder(STRAIN))) +
  geom_col(width = 1, color = 1) +
  coord_polar(theta = "y") +
  scale_fill_brewer(palette = "Pastel1") +
  geom_label_repel(data = org_male,
                   aes(y = pos, label = paste0(Count, " (", Percent, "%)")),
                   size = 4.5, nudge_x = 1, show.legend = FALSE) +
  guides(fill = guide_legend(title = "Организм")) +
  theme_void()
```

Графики можно размещать последовательно друг за другом каждый в своем чанке в рамках одной строки. В этом случае они будут располагаться слева направо по горизонтали и каждый будет иметь одинаковую ширину. Таким образом, не потребуется создавать для каждого графика отдельный столбец и они все будут находиться каждый под соответствующей карточкой значений.

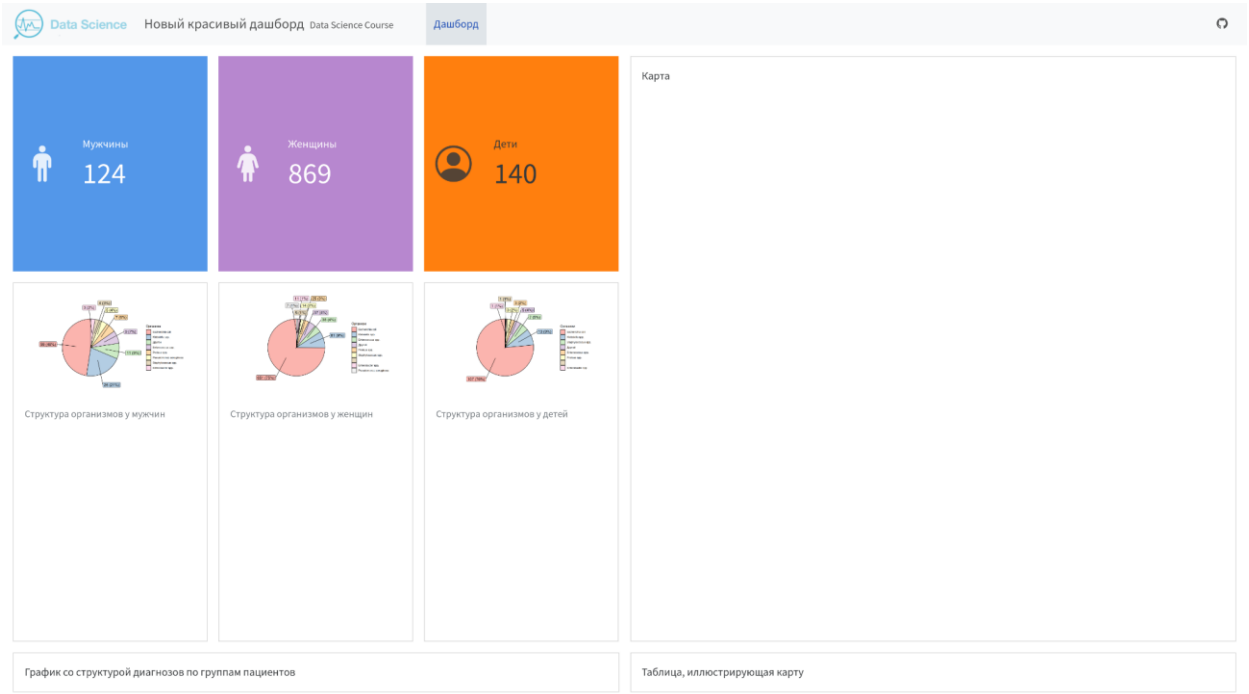


Рисунок 8.148. Дашборд с настроенными карточками значений и графиками

Каждую такую карточку можно развернуть на весь экран, чтобы рассмотреть подробнее. Для этого нужно навести курсором на нижний правый угол карточки и нажать появившуюся кнопку **Expand**.

Графики plotly

С помощью интерактивного графика *plotly* далее будет представлена структура диагнозов для каждой группы пациентов в виде стакбара. Перед этим необходимо сократить названия диагнозов, чтобы они корректно отображались на графике.

Обработка и подготовка данных для графика, а также его отрисовка будет приведена в одном чанке.

```
```{r}
#| label: Диагнозы по группам
#| title: Структура диагнозов по группам пациентов

diag <- patients %>% group_by(PAT_GROUP, mkb_name) %>%
  summarise(Count = n()) %>%
  ungroup() %>%
  pivot_wider(names_from = "PAT_GROUP", values_from = "Count", values_fill = 0)
%>%
  mutate(mkb_name = case_when(
    mkb_name == "Интерстициальный цистит (хронический)" ~ "Хронический цистит",
    mkb_name == "Необструктивный хронический пиелонефрит, связанный с рефлюксом" ~
"Хронический пиелонефрит",
    mkb_name == "Острый тубулоинтерстициальный нефрит" ~ "Острый нефрит",
    mkb_name == "Инфекция мочевыводящих путей без установленной локализации" ~
"Инфекция МВП",
    TRUE ~ mkb_name
  ))

diag$mkb_name_x <- sapply(diag$mkb_name,
                      FUN = function(x) {paste(strwrap(x, width = 24), collapse =
"
")})

plot_ly(diag, x = ~mkb_name_x, y = ~`Дети, неосложненные`, type = 'bar', name =
'Дети, неосложненные') %>%
add_trace(y = ~`Дети, осложненные`, name = 'Дети, осложненные') %>%
add_trace(y = ~`Женщины, неосложненные`, name = 'Женщины, неосложненные') %>%
add_trace(y = ~`Женщины, осложненные`, name = 'Женщины, осложненные') %>%
add_trace(y = ~`Мужчины, неосложненные`, name = 'Мужчины, неосложненные') %>%
add_trace(y = ~`Мужчины, осложненные`, name = 'Мужчины, осложненные') %>%
layout(yaxis = list(title = 'Кол-во'), barmode = 'stack') %>%
layout(xaxis = list(title = '' )) %>%
layout(legend = list(orientation = 'h'))
```
```

По умолчанию такой чанк отображается в дашборде в виде карточки. Для таких карточек можно задать заголовок с помощью атрибута *title*. В результате получится следующий дашборд.

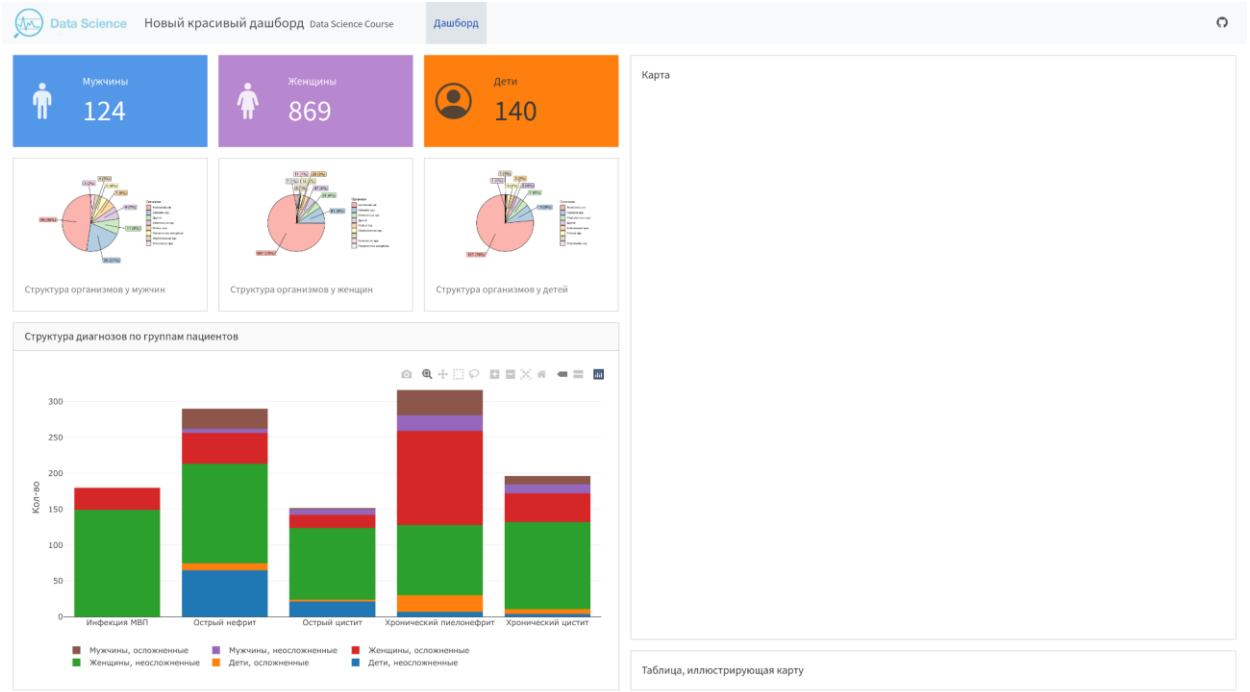


Рисунок 8.149. Дашборд с настроенными карточками значений, графиками *ggplot2* и *plotly*

Карты leaflet

На втором столбце дашборда разместится карта, которая будет иллюстрировать сколько пациентов из каких городов приняло участие в исследовании. Для построения такой интерактивной карты будет использоваться пакет *leaflet*. Для построения такой карты необходимо посчитать количество пациентов в каждой категории с группировкой по каждому городу в соответствующем чанке.

```
```{r}
#| label: Количество пациентов по городам
#| include: false
men <- patients %>%
  filter(grepl("Мужчины",PAT_GROUP)) %>%
  select(CITYNAME, LATITUDE, LONGITUDE) %>%
  group_by(CITYNAME, LATITUDE, LONGITUDE) %>%
  summarise(CountMen = n()) %>%
  ungroup()
```
```

Все эти таблицы объединяются в итоговую, которая будет использована для отрисовки карты.

```
```{r}
#| label: leaflet
#| title: Количество пациентов по городам
mapdata <- all %>% left_join(men) %>% left_join(women) %>% left_join(children) %>%
mutate_all(~replace(., is.na(.), 0))
mapdata %>%
  leaflet() %>%
```
```

```
  addCircleMarkers(lng = ~ LONGITUDE, lat = ~ LATITUDE,
    stroke = FALSE, fillOpacity = 0.5,
    radius = ~ scales::rescale(sqrt(Count), c(1, 10)),
    label = ~ paste("" , CITYNAME, ": ", Count, "", "<br/>",
      "Мужчин:", CountMen, "<br/>",
      "Женщин:",  CountWoman, "<br/>",
      "Дети:",  CountChild
      ) %>% map(html),
    labelOptions = c(textsize = "15px")) %>%

addTiles("http://services.arcgisonline.com/arcgis/rest/services/Canvas/World_Light
_Gray_Base/MapServer/tile/{z}/{y}/{x}")
```

В результате получается карта с маркерами. При клике на маркер появляется всплывающее сообщения с количеством пациентов по категориям в выбранном городе.

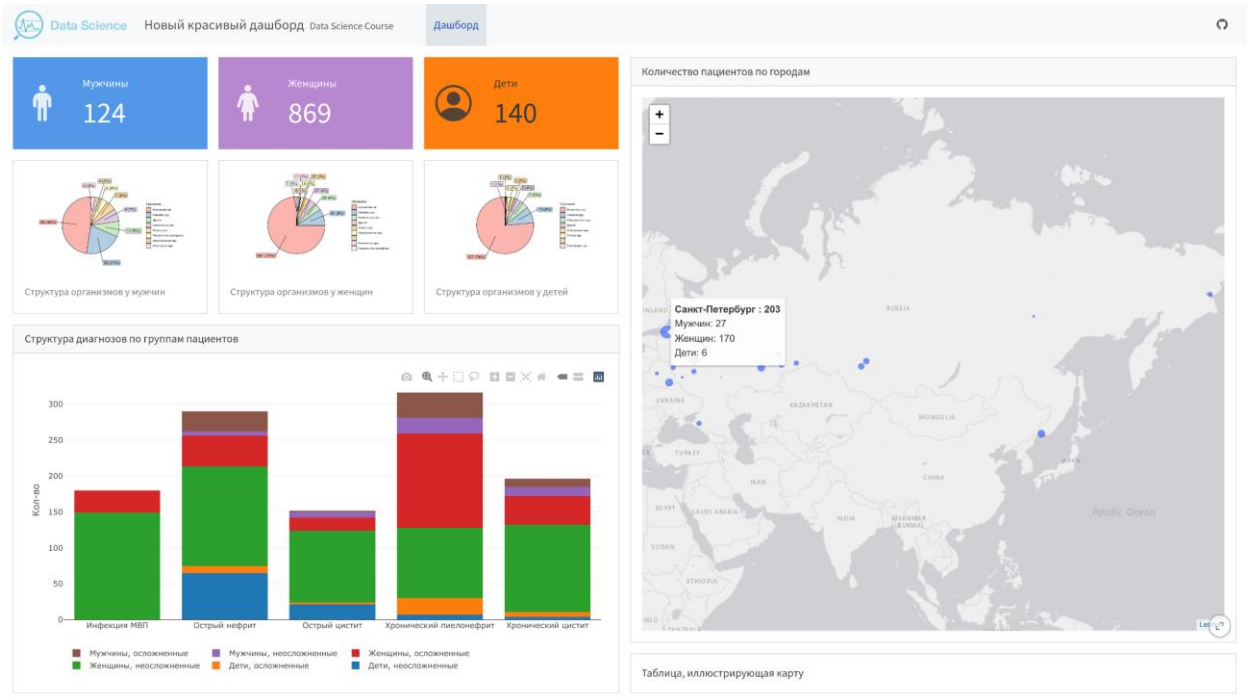


Рисунок 8.150. Дашборд с настроенными карточками значений, графиками *ggplot2*, *plotly* и картой *leaflet*

Таблицы kable

Для отображения количества пациентов по городам с разбивкой по группам необходимо подготовить соответствующую таблицу.

```
```{r}
#| label: Расчет количества пациентов по городам
#| include: false
citypat <- patients %>% group_by(CITYNAME, PAT_GROUP) %>% summarise(Count = n())
%>%
  ungroup() %>%
  pivot_wider(names_from = "PAT_GROUP", values_from = "Count", values_fill = 0)
%>%
```
```

```
  select(order(colnames(.)))
colnames(citypat)[1] <- "Город"
```

Для вывода таблиц на рендер по умолчанию в Quarto используется пакет *kable*, поэтому для вывода потребуется следующий чанк.

```
```{r}
#| label: kable
#| title: kable
citypat %>%
  kbl(caption = "Распределение пациентов по городам") %>%
  kable_styling(bootstrap_options = c("striped"))
```
```

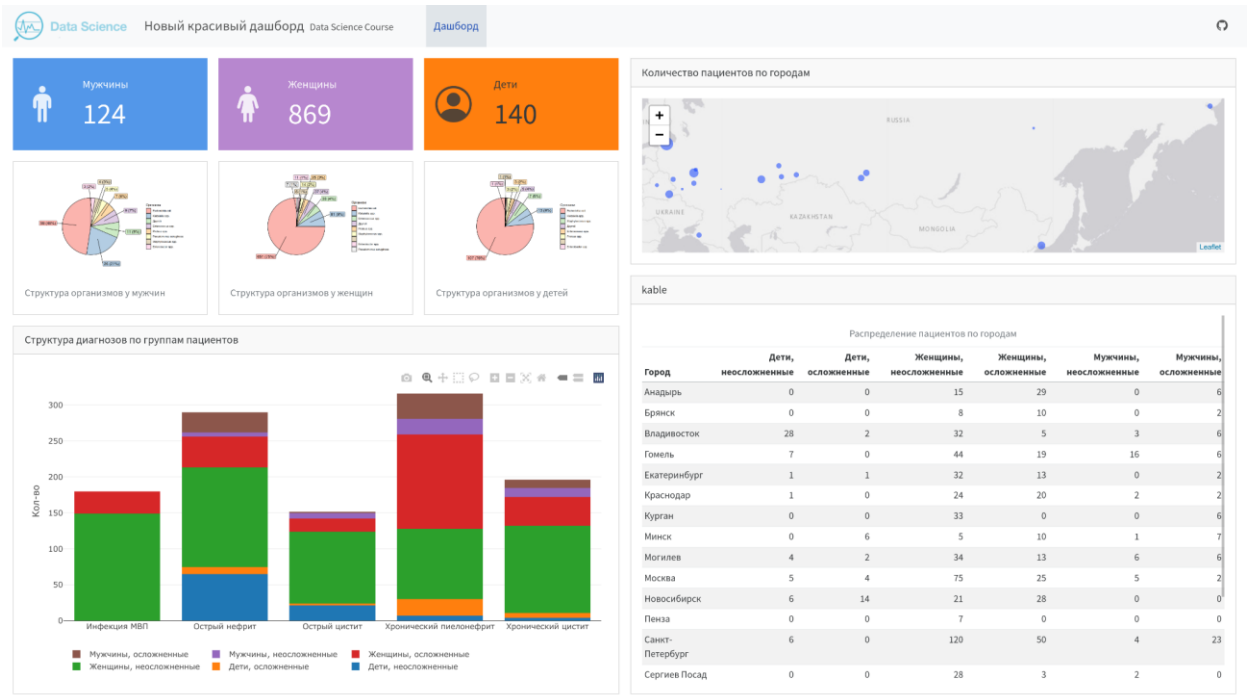


Рисунок 8.151. Дашборд с настроенными карточками значений, графиками *ggplot2*, *plotly*, картой *leaflet*, и таблицей *kable*

Для того, чтобы сравнить возможности оформления и вывода таблиц, далее представлен пример чанков с использованием пакетов представления таблиц: *gt* и *flextable* (**см. разделы 8.2.2. и 8.2.3.**).

Таблицы gt

```
```{r}
#| label: gt
#| title: gt
citypat %>%
  gt() %>%
  tab_header(title = "Распределение пациентов по городам") %>%
  opt_row_striping()
```
```

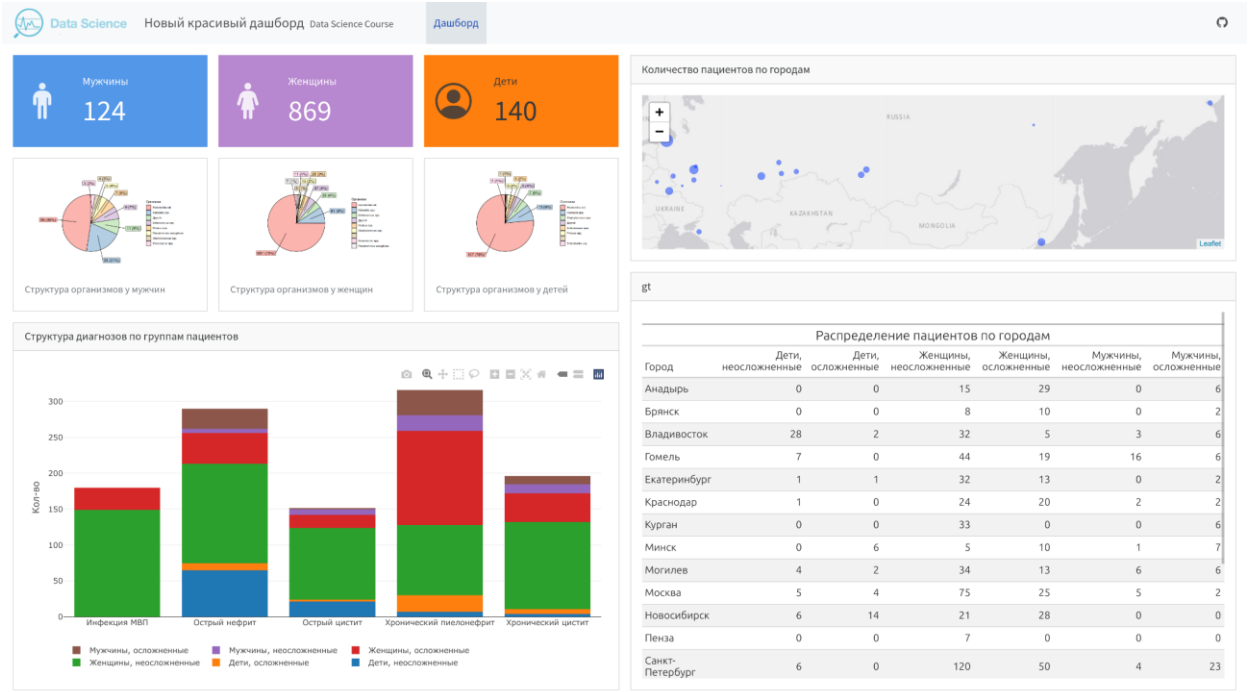
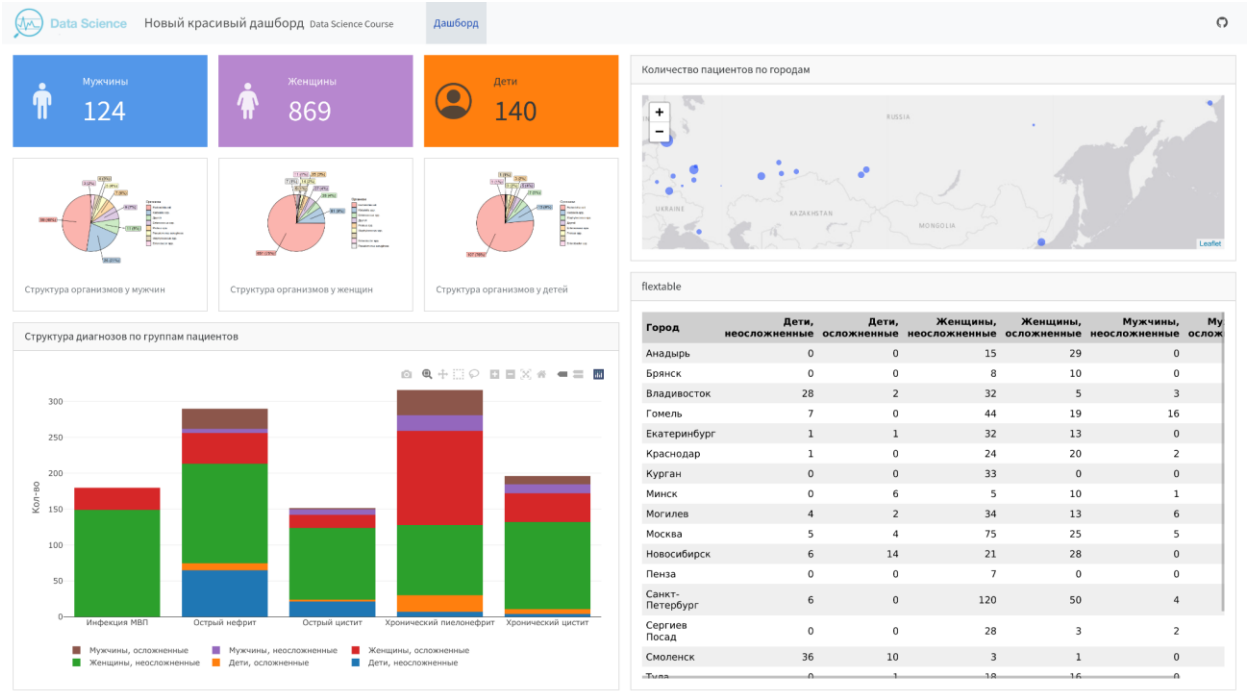


Рисунок 8.152. Дашборд с настроенными карточками значений, графиками *ggplot2*, *plotly*, картой *leaflet*, и таблицей *gt*

Таблицы flextable

```r
#| label: flextable
#| title: flextable
citypat %>%
  flextable() %>%
  set_caption("Распределение пациентов по городам") %>%
  theme_zebra()
```



Город	Дети, неосложненные	Дети, осложненные	Женщины, неосложненные	Женщины, осложненные	Мужчины, неосложненные	Му осложн
Анадырь	0	0	15	29	0	
Брянск	0	0	8	10	0	
Владивосток	28	2	32	5	3	
Гомель	7	0	44	19	16	
Екатеринбург	1	1	32	13	0	
Краснодар	1	0	24	20	2	
Курган	0	0	33	0	0	
Минск	0	6	5	10	1	
Могилев	4	2	34	13	6	
Москва	5	4	75	25	5	
Новосибирск	6	14	21	28	0	
Пенза	0	0	7	0	0	
Санкт-Петербург	6	0	120	50	4	
Сергиев Посад	0	0	28	3	2	
Смоленск	36	10	3	1	0	

Рисунок 8.153. Дашборд с настроенными карточками значений, графиками *ggplot2*, *plotly*, картой *leaflet*, и таблицей *flextable*

Табы в карточках

В стандартных компонентах Quarto есть возможность размещения в одной панели сразу нескольких визуализаций с возможностью переключения между ними с помощью карточек. Для этого нужно создать строку с атрибутом *tabset*.

Например, можно разместить в такой строке сразу три таблицы, чтобы можно было сравнить особенности их оформления.

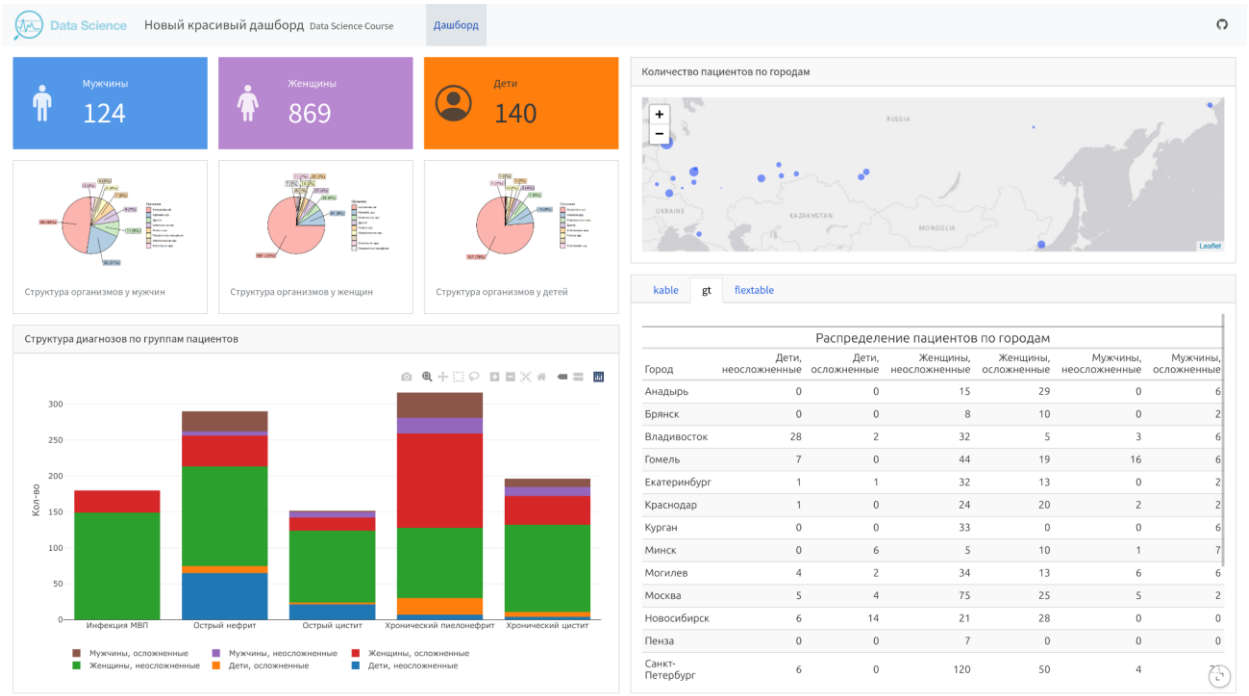
```
#### Row {.tabset}

```{r}
#| label: kable
#| title: kable
citypat %>%
  kbl(caption = "Распределение пациентов по городам") %>%
  kable_styling(bootstrap_options = c("striped"))
```

```{r}
#| label: gt
#| title: gt
citypat %>%
  gt() %>%
  tab_header(title = "Распределение пациентов по городам") %>%
  opt_row_striping()
```

```{r}
#| label: flextable
```
```

```
#| title: flextable
citypat %>%
  flextable() %>%
  set_caption("Распределение пациентов по городам") %>%
  theme_zebra()
```



Город	Дети, неосложненные	Дети, осложненные	Женщины, неосложненные	Женщины, осложненные	Мужчины, неосложненные	Мужчины, осложненные
Анадырь	0	0	15	29	0	6
Брянск	0	0	8	10	0	2
Владивосток	28	2	32	5	3	6
Гомель	7	0	44	19	16	6
Екатеринбург	1	1	32	13	0	2
Краснодар	1	0	24	20	2	2
Курган	0	0	33	0	0	6
Минск	0	6	5	10	1	7
Могилев	4	2	34	13	6	6
Москва	5	4	75	25	5	2
Новосибирск	6	14	21	28	0	0
Пенза	0	0	7	0	0	0
Санкт-Петербург	6	0	120	50	4	

Рисунок 8.154. Дашборд с настроенными карточками значений, графиками *ggplot2*, *plotly*, картой *leaflet*, и дополнительными вкладками для каждого типа таблиц (*kable*, *gt* и *flextable*)

Какие таблицы использовать - на самом деле вопрос визуального восприятия, привычки и типа документов, которые предполагается создавать. Стоит лишь отметить некоторые особенности каждого из пакетов:

- *kable* является стандартным способом вывода таблиц для используемого способа рендеринга документов, поэтому результат его работы может быть более предсказуемым.
- *gt* сейчас наиболее быстро развивается непосредственно авторами Quarto, имеет больше возможностей оформления и выглядит более перспективным именно для создания интерактивных документов в формате HTML.
- *flextable* больше заточен на создание таблиц в офисных документах.

В целом использование пакета *gt* выглядит более перспективным.

Интерактивные таблицы reactable

Нельзя также не упомянуть пакет для создания интерактивных таблиц с фильтрами и поиском - *reactable*. Если предыдущие пакеты удобно использовать для отображения

компактных таблиц, где важна непосредственно визуальная составляющая, то *reactable* позволяет отобразить относительно "большую" и "плоскую" таблицу и дать пользователю просматривать ее самостоятельно. Такой прием удобно использовать для показа непосредственно набора данных, на основании которого производился анализ, чтобы продемонстрировать исходные данные и дать возможность перепроверки результатов.

Для отображения исходного набора данных лучше добавить новую страницу с помощью заголовка первого уровня.

```
# Набор данных
```

Так как по умолчанию все управляющие элементы таблицы описаны на английском языке, необходимо переопределить стандартные подписи. Удобно это сделать глобально для настроек пакета, чтобы не задавать эти параметры для каждой создаваемой таблицы, а определить как параметры по умолчанию.

```
```{r}
#| label: reactable-options
#| include: false

options(reactable.language = reactableLang(
  pageSizeOptions   = "показано {rows} значений",
  pageInfo          = "с {rowStart} по {rowEnd} из {rows} строк",
  pagePrevious      = "назад",
  pageNext          = "вперед",
  searchPlaceholder = "Поиск...",
  noData            = "Значения не найдены"
))
```
```

Удобно, что непосредственно в самой таблице можно переопределить названия столбцов таблицы данных, которые будут использоваться при отображении, не меняя описание самого набора данных (например, сделать так чтобы столбец исходных данных “AGE” отображался как “Возраст”).

```
```{r}
#| label: reacttable
#| title: Список пациентов, включенных в исследование

patients %>%
  select(-c("LATITUDE", "LONGITUDE")) %>%
  select(study_subject_id, PAT_GROUP, SEX, AGE, DATEBIRTH,
         STRAIN, DATESTRAIN,
         CENTER, COUNTRY, CITYNAME, DATEFILL,
         DIAG_ICD, mkb_name, COMPL) %>%
  reactable(filterable = TRUE, searchable = TRUE, striped = TRUE,
            columns = list(
              study_subject_id = colDef(name = "ID", width = 64, defaultSortOrder
= "asc"),
              PAT_GROUP = colDef(name = "Группа", width = 150),
              SEX = colDef(name = "Пол", width = 100),
              AGE = colDef(name = "Возраст", width = 90),
```
```

```
                DATEBIRTH = colDef(name = "Дата рожд.", width = 120),
                STRAIN = colDef(name = "Организм", width = 150),
                DATESTRAIN = colDef(name = "Дата получ.", width = 120),
                CENTER = colDef(name = "Центр", width = 70),
                COUNTRY = colDef(name = "Страна", width = 100),
                CITYNAME = colDef(name = "Город", width = 150),
                DATEFILL = colDef(name = "Дата заполн.", width = 120),
                DIAG_ICD = colDef(name = "МКБ-10", width = 80),
                mkb_name = colDef(name = "Диагноз"),
                COMPL = colDef(name = "Осложнения")
              ))
```

В результате данных манипуляций получится отдельная страница, представляющая таблицу с набором данных, который использовался при построении дашборда.

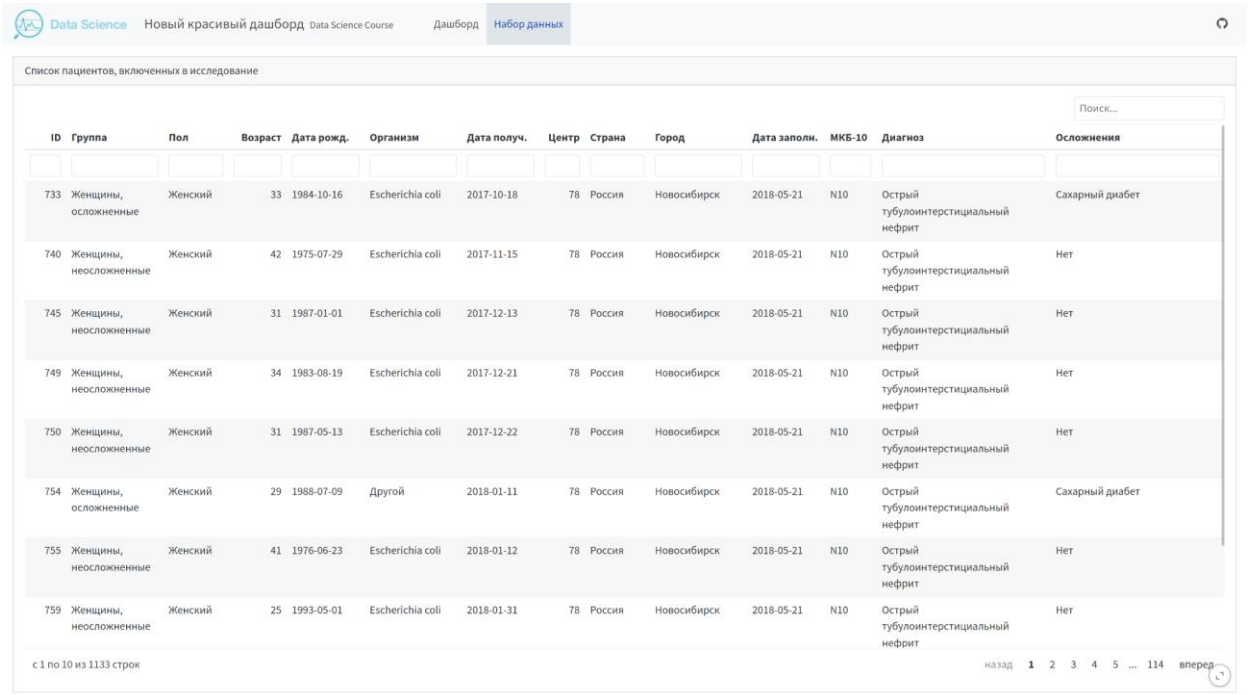


Рисунок 8.155. Дашборд с отдельной страницей, отображающей исходные данные

Сайдбар

Для размещения некоторой справочной информации, не имеющей важного значения для визуализации данных непосредственно, можно использовать элемент сайдбар, который может быть по умолчанию скрыт и не отбирать полезное место у визуализации.

Он размещается на странице и задается заголовком второго уровня.

```
## {.sidebar}
```

Сайдбар является полноценным элементом страницы и в нем можно размещать как статический контент Markdown, так и динамические вычисления R. В качестве

примера далее в сайдбар кратко описана информация, которая представлена на дашборде.

```
Данное исследование рассматривает  `{r} nrow(patients)` пациентов из `{r}
length(unique(patients$CITYNAME))` городов России и Беларуси.

|               |                                          |
|-------------- |------------------------------------------|
| **Центров **  | `{r} length(unique(patients$CENTER))` |
| **Пациентов** | `{r} nrow(patients)`                     |
| **Дата**      | `{r} Sys.Date()`                         |
```

Еще одним скрывающимся элементом, который можно разместить на дашборде является информационное сообщение *callout-note*.

```
::: {.callout-note collapse="true"}
### О проекте

Это демонстрационное исследование, код проекта доступен по [по
ссылке](https://github.com/data-science-course/quarto-dashboard).
:::
```

Такие сообщения могут быть разных типов, которые отличаются цветом заголовка и используемой иконкой:

- *note* - обычное сообщение (синее);
- *warning* - предупреждение (оранжевое);
- *important* - важное сообщение (красное);
- *tip* - подсказка (зеленая);
- *caution* - предупреждение (желтое).

Режим отображения также можно определить:

- *default* - отображение по умолчанию, с цветным заголовком и иконкой;
- *simple* - упрощенное отображение, заливка у заголовка отсутствует;
- *minimal* - сообщение, обозначенное границей.

```
::: {.callout-tip appearance="simple"}
## Подсказка

Пример подсказки, которая не скрывается.
:::
```

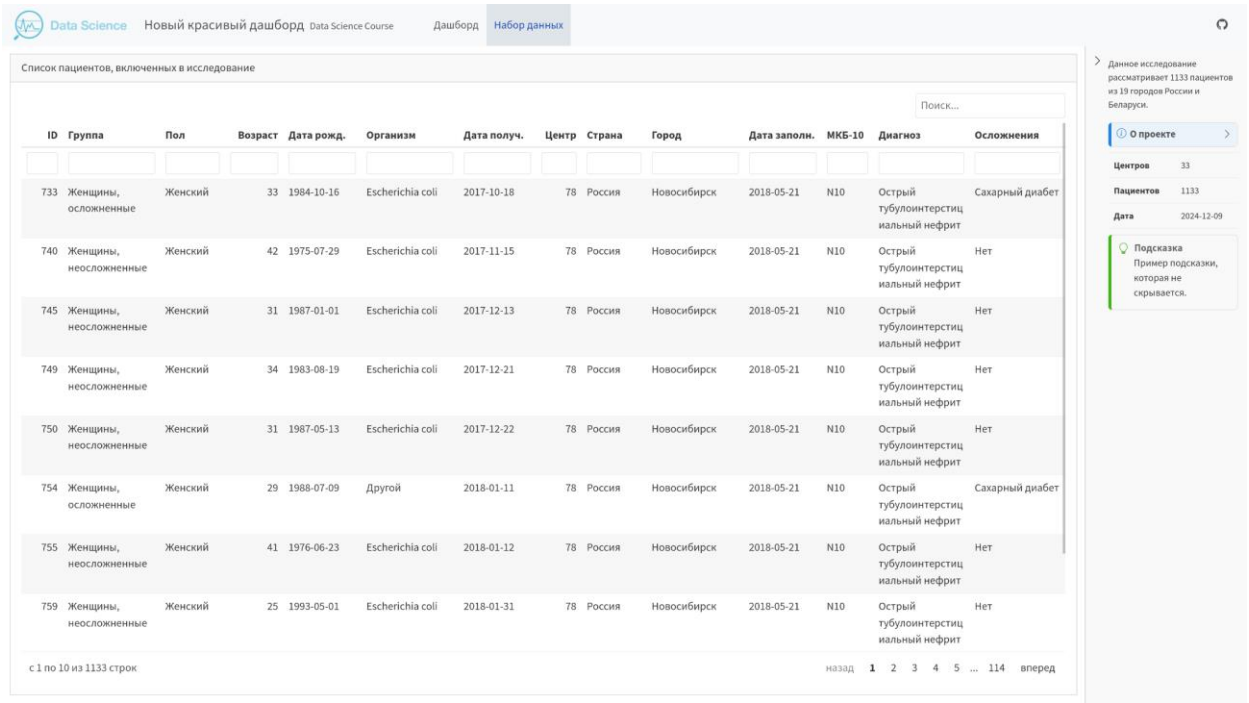


Рисунок 8.156. Дашборд с элементом сайдбар

8.3.5. Публикация дашборда

Когда формирование дашборда завершено, его необходимо скомпилировать в файлы, пригодные для публикации. Это можно сделать как с помощью RStudio при нажатии кнопки Render, так и с помощью команды в терминале.

```
quarto render
```

В результате рядом с файлом *.qmd* появится файл *.html*, а все служебные файлы, необходимые для корректного отображения дашборда в браузере будут расположены в папке, название которой оканчивается на *_files*. Данный HTML-файл можно открыть в браузере и посмотреть результат. Именно эти файлы и будут использоваться для публикации в интернете.

Публикация документа Quarto в интернете доступна напрямую из RStudio с помощью кнопки Publish.

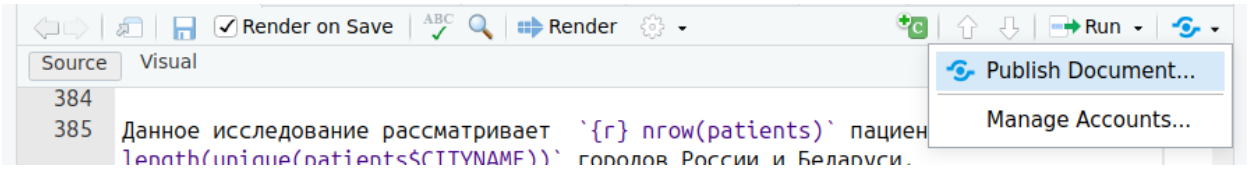


Рисунок 8.157. Публикация дашборда из RStudio

С помощью данной кнопки можно опубликовать дашборд на сервисах: Rpubs, Posit Connect, Posit Cloud.

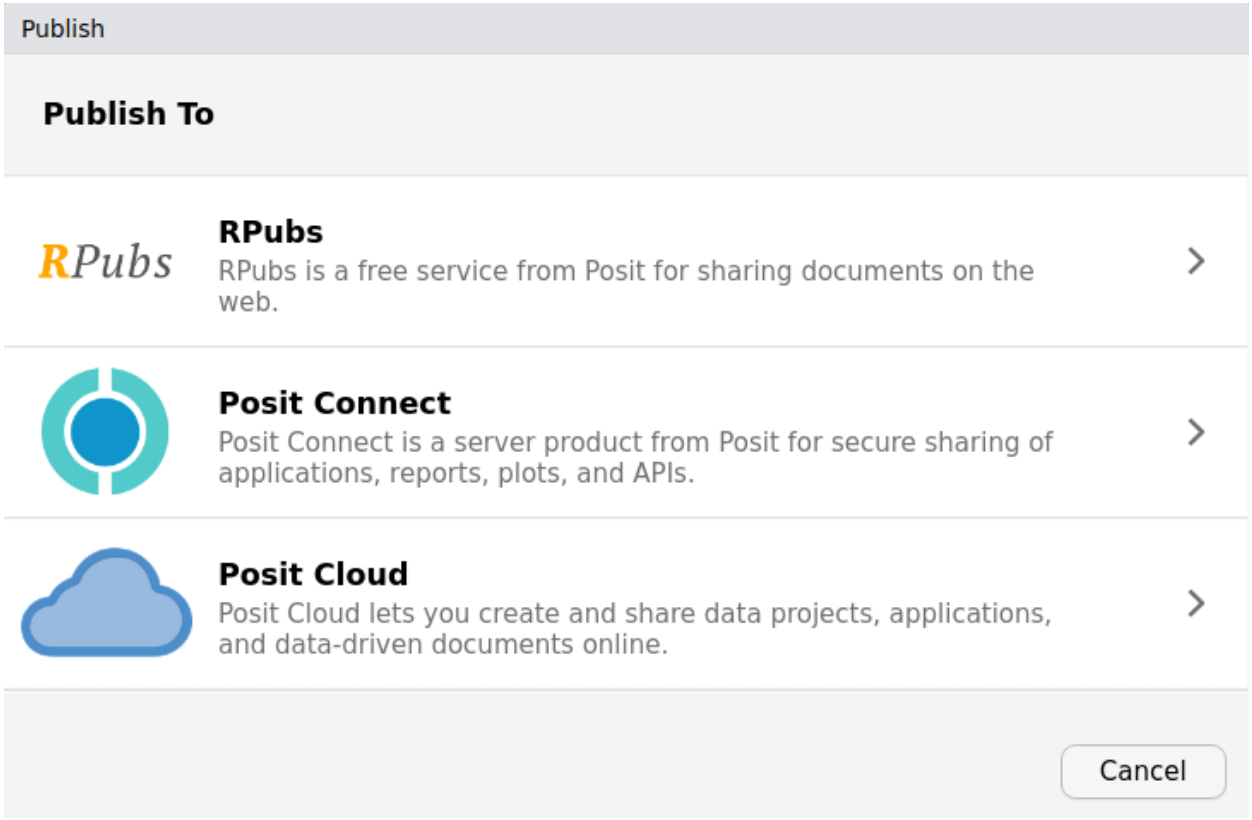


Рисунок 8.158. Выбор способа публикация дашборда из RStudio

Однако сам Quarto предлагает более богатые возможности, с помощью команды:

```
quarto publish
```

По умолчанию доступны сервисы Quarto Pub, GitHub Pages, Posit Connect, Netlify, Confluence, Hugging Face Spaces.

Удобно использовать публикацию на Github Pages, так как этот способ позволяет хранить в одном репозитории и исходный код и скомпилированный дашборд. При этом дашборд будет доступен для просмотра в интернете по прямой ссылке.

Для того, чтобы все получилось, необходимо, чтобы проект содержал в себе Git-репозиторий и был размещен на GitHub. Важное условие для публикации GitHub Pages - необходимо, чтобы репозиторий был публичный, поэтому необходимо внимательно следить за публикуемыми файлами, чтобы они не содержали параметры подключения, логины, пароли подключения к каким-либо закрытым системам и прочие данные, которыми не стоит делиться.

После ввода команды quarto publish в терминале будет предложено выбрать используемый провайдер (Provider) - необходимо выбрать Github Pages.

В терминале отобразится адрес, по которому будет доступен дашборд и необходимо подтвердить публикацию, введя в терминал Y и нажав Enter.

```
? Provider: › GitHub Pages
? Publish site to https://xxxxxx.github.io/new_quarto/ using gh-pages? (Y/n) ›
```

Запустится процесс рендеринга дашборда, в репозитории будет создана новая ветвь gh-pages, которая будет содержать скомпилированные файлы, после того как процесс публикации завершится, выполнится команда push, для отправки изменений на GitHub и откроется сайт с опубликованным дашбордом.

В дальнейшем для быстрой публикации на Github Pages можно использовать команду

```
quarto publish gh-pages
```

В настройках репозитория Github для опубликованного дашборда можно задать собственное доменное имя (в настройках репозитория в разделе Pages).

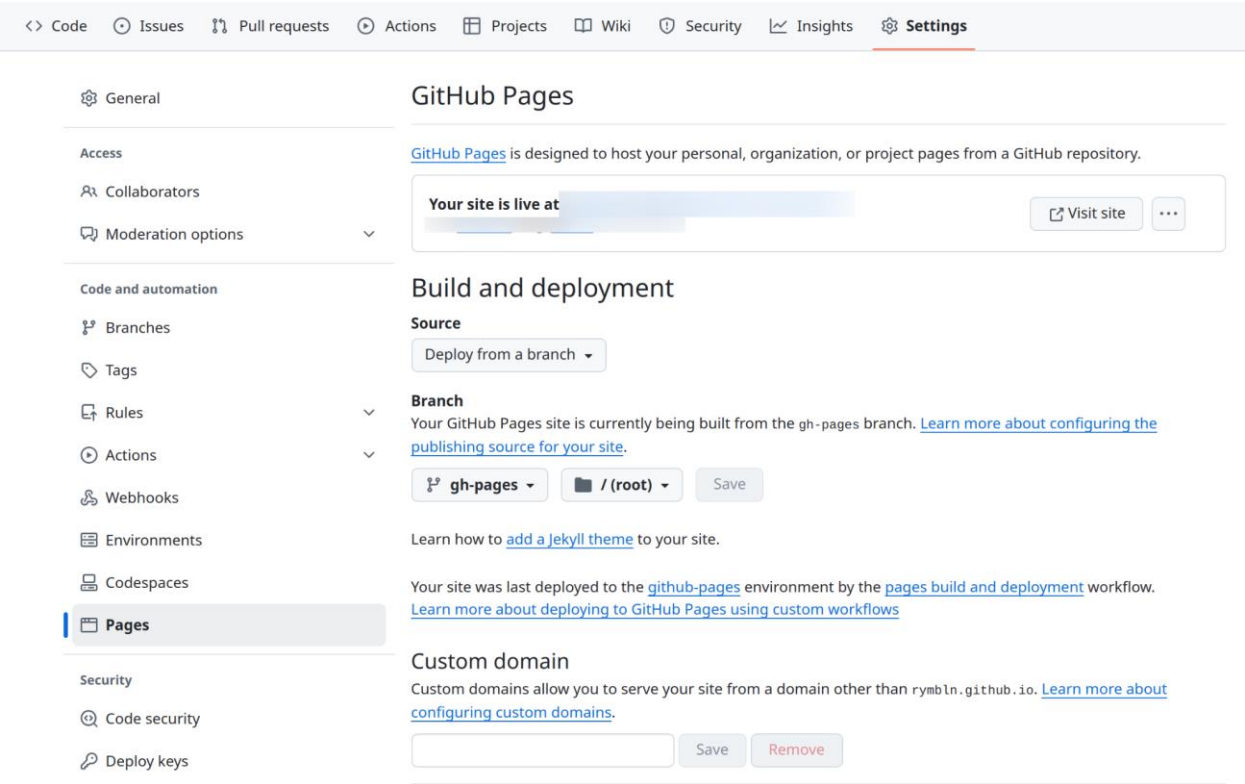


Рисунок 8.159. Настройки репозитория на Github

8.3.6. Публикация с помощью Github Actions

Для автоматизации публикации дашборда можно использовать сервис Github Actions (или аналогичные сервисы), который позволяет автоматизировать рутинные процессы компиляции и публикации. С помощью специальных файлов можно настроить

действия, которые будут выполняться при каждом коммите в репозиторий, или с определенной периодичностью.

Quarto позволяет использовать этот способ для публикации на сервисы Quarto Pub, Github Pages, Netlify. Кроме того, существует отдельный репозиторий с примерами таких файлов для различных действий https://github.com/quarto-dev/quarto-actions.

Перед использованием сервиса необходимо установить пакет *renv* (https://rstudio.github.io/renv/articles/renv.html). Он позволяет обеспечить воспроизводимость выполнения процесса рендеринга и работы любого R-проекта за счет создания специальных файлов, в которых будет хранится информация об используемых версиях R и всех установленных пакетов. С помощью специальных команд можно сохранить перечень всех пакетов в файл и потом восстановить ровно те же пакеты тех же версий на другом компьютере (**см. раздел 6.7.3.**).

После установки пакета *renv* необходимо выполнить в консоли R команду *renv::init()*.

Она создаст файл с описанием конфигурации R в файле renv.lock, файл .Rprofile, который будет запускаться автоматически при каждом открытии проекта в R и папку renv со служебными файлами. Все эти файлы нужно сохранить в репозитории для успешного выполнения процессов Github Actions.

Список всех зависимостей, то есть пакетов, которые нужны для работы проекта R, выводится с помощью команды *renv::dependencies()*.

После инициализации нужно выполнить команду *renv::snapshot()*, чтобы создать снимок всех пакетов, используемых в проекте.

Восстановление пакетов на другом компьютере выполняется с помощью команды *renv::restore()*.

Когда конфигурация *renv* создана, можно задать конфигурацию процесса Github Actions. Она должна определяться в файле *.github/workflows/publish.yml*.

GitHub Actions — это платформа для автоматизации процессов в рамках репозитория GitHub. Она позволяет создавать и запускать рабочие процессы (workflows), которые автоматически выполняются при наступлении определенных событий в репозитории.

Основные компоненты GitHub Actions:

- Workflow (рабочий процесс): YAML-файл, который описывает автоматизируемый процесс.
- Job (задача): определенная часть рабочего процесса, которая выполняется на виртуальной машине.
- Step (шаг): отдельные шаги внутри задачи, которые могут включать команды или действия (actions).

- Action (действие): повторно используемый блок, который выполняет одну конкретную задачу, например, установка зависимостей, запуск тестов и т.д.

Пример содержимого такого файла можно посмотреть в официальной документации по ссылке https://quarto.org/docs/publishing/github-pages.html#github-action.

В самом начале определяются параметры запуска процесса. Он будет запускаться при каждом коммите (команда push) в репозиторий в ветку main.

```
on:
  workflow_dispatch:
  push:
    branches: main
```

Параметры запуска могут отличаться, например можно настроить запуск процесса с заданной периодичностью. В этом случае содержимое блока on, будет таким:

```
on:
  schedule:
    - cron: '0 10 * * *'  # Ежедневно в 10:00 UTC
  workflow_dispatch:
```

Описание периодичности записывается в синтаксисе *cron*, где сначала идут минуты, затем часы, дни, месяцы, годы, определяющие в какой момент времени будет запускаться процесс. Периодический запуск процесса имеет смысл, когда дашборд строится на основании каких-либо внешних данных. В этом случае можно определить файл дашборда, настроить периодичность процесса сборки в Github Actions, отправить изменения на Github и уже не возвращаться к редактированию — он будет сам постоянно обновляться.

В представленном случае набор данных находится в репозитории, поэтому процесс будет запускаться при каждом коммите. Далее в файле указывается название процесса, чтобы его можно было отличать от других в интерфейсе Github.

```
name: Quarto Publish
```

После названия описываются задачи, которые необходимо выполнить в рамках рабочего процесса. Сначала задается тип используемой виртуальной машины, на которой будут выполняться шаги, а затем описываются сами шаги. В представленном примере все будет выполняться на виртуальной машине с установленной Ubuntu 24.04, дополнительно будут даны разрешения на запись данных, потому что в процессе будет генерироваться файлы дашборда.

```
# Задачи, которые необходимо выполнить
jobs:
  # Задача сборки и публикации проекта
  build-deploy:
    # Указание, что запускать нужно на ubuntu последней версии
```

```
    runs-on: ubuntu:24.04
    permissions:
      contents: write
    # Описание необходимых шагов сборки
    steps:
```

Прежде всего нужно установить пакеты непосредственно в самой системе Ubuntu, которые потребуются для дальнейшей работы. Для этого в поле run: описываются команды, которые нужно выполнить.

```
- name: Install libcurl
  run: sudo apt install -y libcurl4-openssl-dev libfontconfig1-dev libcairo2-dev
libharfbuzz-dev libfribidi-dev libfreetype6-dev libpng-dev libtiff5-dev libjpeg-
dev libgdal-dev
```

После необходимо получить актуальное состояние репозитория - склонировать его внутрь созданной виртуальной машины. В данном случае используются стандартные описания процедур, уже определенные в соответствующих Github-репозиториях, поэтому нет необходимости писать все команды скрипта вручную в блоке run, а можно использовать блок uses: с названиями процедур.

```
- name: Check out repository
  uses: actions/checkout@v4
```

Следующим этапом устанавливается Quarto.

```
- name: Set up Quarto
  uses: quarto-dev/quarto-actions/setup@v2.1.6
```

Затем устанавливается среда R. Важно обратить внимание, что указанная версия должна совпадать с версией, указанной в файле renv.lcok.

```
- name: Install R
  uses: r-lib/actions/setup-r@v2
  with:
    r-version: '4.4.1'
```

Когда R установлен, можно установить *renv* и все пакеты, необходимые для сборки проекта.

```
- name: Install R Dependencies
  uses: r-lib/actions/setup-renv@v2
  with:
    cache-version: 1
```

Финальным этапом запускается рендер дашборда и публикация его в ветку gh-pages репозитория. Для доступа к репозиторию обычно используется токен доступа, который передается через переменные окружения.

```
- name: Render and Publish
  uses: quarto-dev/quarto-actions/publish@v2.1.6
  with:
    target: gh-pages
  env:
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

После выполнения последнего этапа скомпилированный дашборд будет опубликован на Github Pages.

Целиком файл .github/workflows/publish.yml выглядит следующим образом:

```
on:
  workflow_dispatch:
  push:
    branches: main
name: Quarto Publish
jobs:
  build-deploy:
    runs-on: ubuntu-24.04
    permissions:
      contents: write
    steps:
      - name: Install libcurl
        run: sudo apt install -y libcurl4-openssl-dev libfontconfig1-dev
libcairo2-dev libharfbuzz-dev libfribidi-dev libfreetype6-dev libpng-dev libtiff5-
dev libjpeg-dev libgdal-dev
      - name: Check out repository
        uses: actions/checkout@v4
      - name: Set up Quarto
        uses: quarto-dev/quarto-actions/setup@v2.1.6
      - name: Install R
        uses: r-lib/actions/setup-r@v2
        with:
          r-version: '4.4.1'
      - name: Install R Dependencies
        uses: r-lib/actions/setup-renv@v2
        with:
          cache-version: 1
      - name: Render and Publish
        uses: quarto-dev/quarto-actions/publish@v2.1.6
        with:
          target: gh-pages
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

За ходом запущенных процессов можно наблюдать на вкладке Action репозитория Github.com.

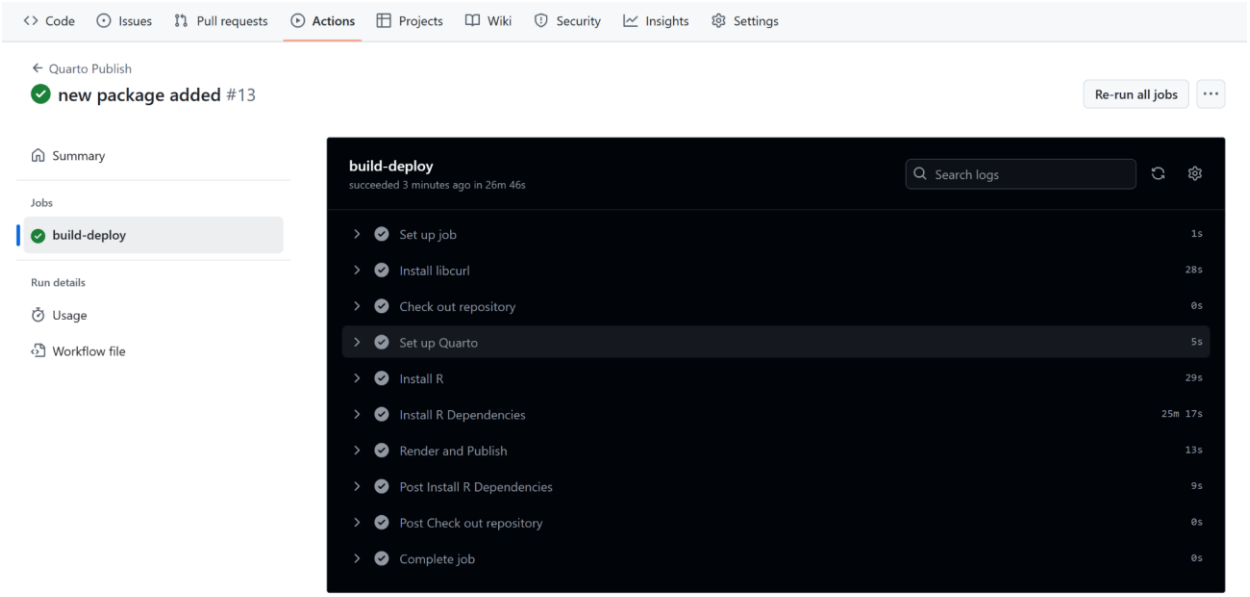


Рисунок 8.160. Отслеживание запущенных процессов обновления и пересборки дашборда на Github.com

Теперь при каждой отправке изменений в репозиторий (например, при изменении данных или добавлении новых компонентов в дашборд) будет выполняться этот процесс и автоматически публиковаться изменения. Таким образом, произойдет автоматическая сборка и публикация дашборда. Благодаря настроенному рабочему процессу изменения можно вносить прямо в репозитории через веб-интерфейс репозитория на Github.com или локально на компьютере без всех установленных пакетов, а компиляция будет осуществляться на серверах Github.