EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103006/I/2023/421000214391

POKROČILÉ RIADENIE VÝSTUPOV V JAZYKU R

Diplomová práca

2023 **Bc. Alena Stracenská**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

POKROČILÉ RIADENIE VÝSTUPOV V JAZYKU R

Diplomová práca

Študijný program:	Informačný manažment
Študijný odbor:	Ekonómia a manažment
Školiace pracovisko:	Katedra matematiky a aktuárstva
Vedúci záverečnej práce:	doc. Ing. Michal Páleš, PhD.

Bratislava 2023 **Bc. Alena Stracenská**

Ekonomická univerzita v Bratislave
Fakulta hospodárskej informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Alena Stracenská
Študijný program:	informačný manažment (Jednoodborové štúdium, inžiniersky II. st., denná forma)
Študijný odbor:	ekonómia a manažment
Typ záverečnej práce:	Inžinierska záverečná práca
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: Pokročilé riadenie výstupov v jazyku R

Anotácia: Diplomová práca sa bude zaoberať rozhraním R Markdown a R Shiny pre pokročilé spracovávanie a prezentovanie výstupov v jazyku R. Tieto techniky môžu byť využité v oblasti manažérskeho rozhodovania a takisto ako podpora dátovej vedy (data science), ak údaje spracovávame v jazyku R. Od študenta sa vyžadujú pokročilé znalosti v oblasti funkcionality jazyka R a rovnako jeho IDE, ktorým je R Studio.

Vedúci:	doc. Ing. Michal Páleš, PhD.
Katedra:	KMA FHI - Katedra matematiky a aktuárstva
Vedúci katedry:	doc. Ing. Michal Páleš, PhD.
	doc. Ing. Michal Páleš, PhD.

Dátum zadania: 14.10.2021

Dátum schválenia: 12.04.2023

prof. Ing. Ivan Brezina, CSc.
osoba zodpovedná za realizáciu študijného programu

Čestné vyhlásenie

Čestne vyhlasujem, že som diplomovú prácu Pokročilé riadenie výstupov v jazyku R vypracovala samostatne a uviedla som všetku použitú literatúru.

Bratislava, dňa 24.4.2023

.....

podpis autora

Poďakovanie

Chcem sa poďakovať vedúcemu záverečnej práce doc. Ing. Michalovi Pálešovi PhD. za odborné vedenie, rady a pripomienky, ktoré mi pomohli pri vypracovaní tejto diplomovej práce.

Abstrakt

STRACENSKÁ, Alena: *Pokročilé riadenie výstupov v jazyku R.* - Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky: Katedra matematiky a aktuárstva - Vedúci záverečnej práce: doc. Ing. Michal Páleš, PhD. - Bratislava, FHI EU, 2023, 131 strán

Práca sa zaoberá rozhraniami R Markdown a R Shiny pre pokročilé spracovanie výstupov v jazyku R. Súčasťou práce sú detailné technické a praktické opisy oboch rozhraní, vrátane ich prepojenia. Prvá kapitola obsahuje opis jazyka R a oboch rozhraní. Druhú kapitolu tvoria ciele práce. V tretej kapitole sa zameriame na technické špecifikácie oboch rozhraní a opis jednotlivých funkcií. V poslednej štvrtej kapitole sme vytvorili praktický návod, ktorý používateľovi krok po kroku na jednotlivých príkladoch vysvetlí použitie funkcií z tretej kapitoly a umožní mu tvorbu plne funkčných R Markdown dokumentov a R Shiny aplikácií vrátane ich nasadenia na web.

Kľúčové slová: jazyk R, R Markdown, R Shiny

Abstract

STRACENSKÁ, Alena: *Advanced managing of outputs in R language.* - University of Economics in Bratislava. Faculty of Economic Informatics: Department of Mathematics and Actuarial Science - Thesis Supervisor: doc. Ing. Michal Páleš, PhD. - Bratislava, FHI EU, 2023, 131 pages

This diploma thesis evaluates R Markdown and R Shiny interfaces for the use of advanced program outputs processing in R programming language. This includes detailed technical and practical descriptions of both interfaces, including theirs interconnections. In the first section is includes description of R programming language and it´s interfaces. Second section is made up of goals for this thesis. Third section is dedicated for technical parameters, specifications and aspects of both interfaces and description of common functions. Practical use of functions from both interfaces are listed in final, forth section, as step-by-step outline on examples of use, including aplication development and deployment process.

Keywords: R language, R Markdown, R Shiny

Obsah

Úvod **9**

1 Súčasný stav riešenej problematiky doma a v zahraničí **10**

1.1 Jazyk R . . . 10

1.1.1 Výhody jazyka R . . . 11

1.1.2 Nevýhody jazyka R . . . 11

1.1.3 RStudio . . . 12

1.2 R Markdown . . . 13

1.3 R Shiny . . . 14

1.3.1 HTML . . . 15

1.3.2 CSS . . . 15

1.3.3 JavaScript . . . 16

1.4 LaTeX . . . 17

1.5 NGINX . . . 17

1.6 HTTP/HTTPS protokol . . . 18

1.7 WebSocket . . . 18

1.8 SSL a TLS . . . 18

2 Cieľ práce **20**

3 Metodika práce **21**

3.1 Technická špecifikácia R Markdown . . . 21

3.1.1 Nástroj Pandoc . . . 21

3.1.2 Balíčky a značkovacie jazyky . . . 21

3.1.3 Balíček `knitr` . . . 23

3.1.4 R Markdown workflow . . . 23

3.2 Typy výstupných formátov . . . 25

3.3 Štruktúra R Markdown dokumentu . . . 27

3.3.1 YAML hlavička . . . 27

3.3.2 Telo dokumentu . . . 28

3.3.3 Code chunks . . . 33
3.4 Nastavenia výstupných formátov . . . 35
3.4.1 PDF dokument . . . 35
3.4.2 HTML dokument . . . 40
3.4.3 Dashboard . . . 43
3.5 Technická špecifikácia R Shiny . . . 46
3.6 Proces tvorby R Shiny aplikácie . . . 46
3.6.1 Typy používateľských vstupov . . . 47
3.6.2 Typy výstupov . . . 52
3.6.3 Reaktivita . . . 54
3.6.4 Rozloženie a témy . . . 57
3.7 Shiny dashboard . . . 65
3.7.1 Hlavička . . . 66
3.7.2 Navigačný panel . . . 68
3.7.3 Telo . . . 69
3.7.4 Témy . . . 74
3.8 Prepojenie R Markdown a R Shiny . . . 75

4 Výsledky práce **78**
4.1 Praktická ukážka č. 1 . . . 78
4.2 Praktická ukážka č. 2 . . . 92
4.3 Praktická ukážka č. 2 v jazyku Python . . . 98
4.4 Praktická ukážka č. 3 . . . 103
4.5 Praktická ukážka č. 4 . . . 106
4.6 Možnosti nasadenia na web . . . 112
4.6.1 Nasadenie aplikácie na Shiny server . . . 112
4.6.2 Zabezpečenie Shiny servera . . . 119
4.7 Komplexný prehľad . . . 123

Záver **124**

Zoznam použitej literatúry **126**

Úvod

Rapídny nárast dát v rôznych odvetviach núti firmy začať rozmýšlať, akými spôsobmi a nástrojmi môžu jednotlivé dáta spracovať. V súčasnosti existuje veľké množstvo nástrojov, ktoré môžu použiť. Závisí ako veľmi kvalitnými dátami disponujú a v akom množstve sú generované.

Jedným z obvyklých riešení je z dát vytvoriť automatické reporty, prehľadné dashboardy alebo webové aplikácie. Uvedené riešenia poskytujú ucelené pohľady na spracované dáta a môžu vo veľkej miere uľahčiť rozhodovanie. Najčastejšie sa na tvorbu reportov a aplikácií používajú komplexné riešenia. Veľmi často sa v praxi používa jazyk R, ktorý obsahuje mnohé rozhrania a balíčky, ktoré nám ponúkajú veľa možností na spracovanie a vizualizáciu dát.

Konkrétne sa jedná o rozhrania R Markdown a R Shiny, ktoré sú natívne a pomocou nich dokážeme v priebehu pár hodín jednoducho vytvoriť prehľadné dokumenty alebo webové aplikácie.

V tejto práci sa budeme podrobne zaoberať dvomi rozhraniami v jazyku R, a to R Markdown a R Shiny. Opíšeme ich technickú špecifikáciu a zároveň si vysvetlíme, ako ich prakticky použiť a aké funkcie poskytujú. Okrem toho si ukážeme konkrétne príklady výstupov z oboch rozhraní, ktoré používateľa prevedú krok po kroku tvorbou R Markdown výstupných formátov a R Shiny webovej aplikácie, ktorú si takisto aj nasadíme. Podobným spôsobom si nasadíme aj ostatné praktické príklady, ktoré budú následne dostupné na webovej stránke. Výsledkom práce bude praktický návod k tvorbe reportov a webových aplikácií už k spomínaným rozhraniam.

Výsledky uvedené v práci môžu byť užitočné aj pre používateľov, ktorí už skúsenosti s jazykom R majú, no chcú si rozšíriť vedomosti o možnosti, ktoré jazyk R ponúka.

1 Súčasný stav riešenej problematiky doma a v zahraničí

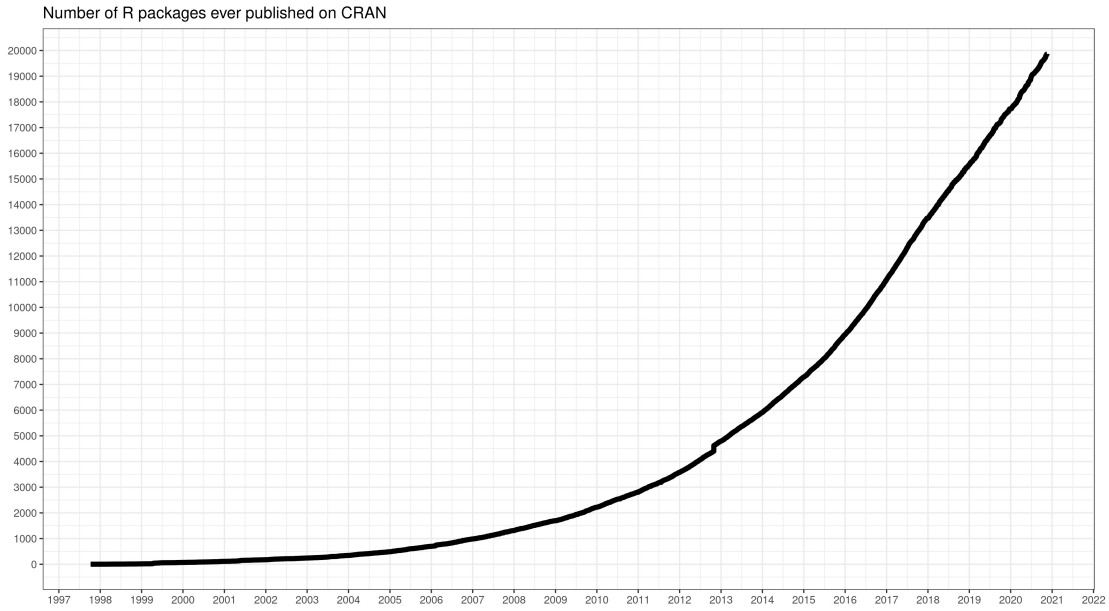
1.1 Jazyk R

Jazyk R bol od roku 1991 vyvíjaný dvojicou autorov Rossom Ihakom a Robertom Gentlemanom. Pochádza z jeho predchodcu jazyka S, ktorý mal len komerčnú implementáciu S-PLUS, t.j. neexistovala open-source verzia. Jazyk S má základy v dátovej analýze a nie je tradičným programovacím jazykom. Filozofiou ktorou sa autori jazyka S - John Chambers a jeho spolupracovníci v Bell Laboratories riadili je, že chceli vytvoriť jazyk, ktorý bude zrozumiteľný nielen pre programátorov, ale aj pre širšiu odbornú verejnosť v oblasti štatistiky a spracovania dát. V roku 1993 Ross Ihaka a Robert Gentleman odbornej verejnosti oznámili, že vyvíjajú jazyk R, jazyk s rovnakou filozofiou ako jazyk S.

V roku 1995 Martin Mächler presvedčil autorov, aby použitím GNU (*General Public License*), sprístupnili jazyk R verejnosti vo forme open-source. Ešte v tom istom roku vydali prvú oficiálnu verziu jazyka R. Medzi rokmi 1996 až 1997 sa formovala skupina, ktorá pomáhala vyvíjať jazyk R. V roku 2000 bola sprístupnená verejnosti finálna *stable-beta* 1.0.0 verzia jazyka R. [10],[22],[31]

Medzi rokmi 2000 až 2015 bola založená nezisková organizácia *R Foundation*, korporátne konzorcium *R Consortium* a takisto sa uskutočnila prvá medzinárodná konferencia jazyka R vo Viedni. Do roku 2023 bolo vydaných viac ako 93 verzií jazyka R. [10],[14]

Programovací jazyk R sa najčastejšie používa v štatistických výpočtoch, dátových analýzach, dátovej vede a v aktuárstve. R disponuje veľkým množstvom balíčkov, ktoré je možné využiť v projektoch. Balíčky sa nachádzajú v centrálnom softvérovom repozitári CRAN (*Comprehensive R Archive Network*), ktorý obsahuje viac ako 18 700 rôznych balíčkov, ktoré sa môžu použiť napríklad pre spracovanie dát a zobrazenie výstupov. [4],[5],[10]



Obrázok 1: Počet balíčkov publikovaných v CRANe, zdroj: [6]

1.1.1 Výhody jazyka R

- Voľne dostupný a šíriteľný t.j. open-souce.
- Spustiteľný na všetkých počítačových platformách a operačných systémoch napr. Windows, Mac OS, Linux.
- Obsahuje veľké množstvo štatistických a analytických balíčkov.
- Komunita R nadšencov, ktorá radí používateľom a odpovedá na rôzne otázky na online fórach.
- Pravidelné aktualizácie už existujúcich balíčkov a vydávanie nových balíčkov. [22]

1.1.2 Nevýhody jazyka R

- Založený na 50 ročnej technológií jazyka S s malou podporou dynamickej a 3D grafiky, no neustálym vývojom sa táto nevýhoda prakticky eliminuje.
- Objekty, ktoré musia byť uložené vo fyzickej pamäti. Je to z jednej časti kvôli *scoping rules*, t.j. kde jazyk priradí hodnotu danej premennej a z druhej časti kvôli tomu,

že R zaberá viac pamäte ako podobne zamerané jazyky.

- Tvorba špecializovaných balíčkov založená na dobrovoľných príspevkoch R nadšencov. [22]

1.1.3 RStudio

Najpoužívanejším IDE pre jazyk R je RStudio, ktoré vzniklo v roku 2009 a jeho autorom je J.J. Allaire. RStudio existuje v niekoľkých adaptáciách, zahrňajúcich open-source vydania, cloudové riešenia ako aj profesionálne enterprise produkty. Základnou open-source adaptáciou je RStudio Desktop, jedná sa o klasickú aplikáciu ktorú si používateľ nainštaluje a spúšťa lokálne. RStudio Desktop Pro je platená verzia RStudia Desktop, zameraná najmä pre biznis používateľov.

Pre pokročilých používateľov, ktorí využívajú zdieľaný hardvér na výpočtovo náročnejšie úlohy je k dispozícii open-source verzia RStudio Server, ktorú si používateľ prevádzkuje sám. Na rovnakej funkcionalite je založená služba RStudio Cloud, ktorú si môžeme vybrať ak nemáme vlastný server. RStudio Team a Rstudio Workbench sú enterprise edície. RStudio Workbench je pokračovaním už vyradeného produktu RStudio Server Pro, jedná sa teda o biznis implementáciu serverového RStudia. RStudio Team je komplexné enterprise riešenie, ktoré zahŕňa RStudio Workbench, ale aj RStudio Connect - nástroj na pokročilú kolaboráciu na projektoch a RStudio Package Manager - nástroj na pokročilú správu balíčkov v enterprise prostredí.

V októbri 2022 sa firma vlastniaca RStudio premenovala na Posit, kvôli tomu, že sa už nechce priamo zameriavať na jazyk R, ale chce vytvárať open-source riešenia, ktoré budú integrovať R a Python súčasne. V súčasnosti ostáva názov RStudio zachovaný. Zmena rebrandingu firmy sa dotkla týchto adaptácií RStudia:

- RStudio Connect = Posit Connect
- RStudio Package Manager = Posit Package Manager
- RStudio Workbench = Posit Workbench [35],[36]

1.2 R Markdown

Dokumentový formát **R Markdown** bol prvýkrát predstavený začiatkom roka 2012, ako súčasť balíčka `knitr`. Za jeho vznikom bol nápad spojiť code chunks rôznych programovacích jazykov do jedného **R Markdown** dokumentu. Už spomínaný balíček `knitr` podporoval od svojho vzniku viacero autorských jazykov (*sofvér, ktorý sa používa na tvorbu rôznych dokumentov/programov, bez nutnosti mať skúsenosti s programovaním)*, napríklad LaTeX, HTML, AsciiDoc, reStructuredText a Textile. Práve vďaka svojej jednoduchosti a relatívne rýchlo pochopiteľnej syntaxi sa stal **R Markdown** najpoužívanejším dokumentovým formátom.

Originálna verzia **Markdown** vytvorená autorom Johnom Gruberom v roku 2004 bola zo začiatku príliš jednoduchá a prakticky nevhodná pre písanie technických resp. odborných dokumentov. V podstate sa autor riadil myšlienkou, aby sa syntax pužívala len na formátovanie textu na web. Nenachádzala sa v nej syntax, ktorou by sa dali vytvárať tabuľky, matematické výrazy, poznámky alebo citácie. Tento problém vyriešil v roku 2014 John MacFarlane vytvorením nástroja **Pandoc**, ktorý doplnil syntax **Markdownu** a zároveň umožnil konvertovať nielen **Markdown** dokumenty do veľkého množstva výstupných formátov.

Balíček `rmarkdown` bol prvýkrát vytvorený na začiatku roka 2014. Počas posledných ôsmich rokov sa naďalej rozvíjal a v súčasnosti sa dá použiť na:

- Tvorbu R Markdown dokumentov, ktoré môžeme konvertovať do výstupných formátov ako sú PDF, HTML alebo Word.
- Tvorbu slajdov pre prezentácie (HTML5, LaTeX Beamer, PowerPoint).
- Tvorbu dashboardov s flexibilno-interaktívnym rozložením grafických prvkov.
- Tvorbu reaktívnych webových aplikácií pomocou balíčka `shiny`.
- Písanie článkov do časopisov resp. žurnálov.
- Písanie dlhších literárnych diel.
- Generovanie webových stránok a blogov. [12],[44]

Posledné dve verzie balíčka **rmarkdown** 2.20 a 2.21 vydané v januári tohto roka obsahujú pár zmien. Konkrétne ide o úpravy a vylepšenia `ioslides_presentation()` prezentácií, nahradenie funkcie `tufte_handout()` za `tufte::tufte_handout()` a zmena balíčka pre ikony HTML výstupných formátov. V poslednom roku firma Posit súčasne popri **R Markdown** vyvíja dokumentový formát Quarto, ktorý podporuje nielen R, ale natívne aj Python a ďalšie programovacie jazyky ako JavaScript a Julia. Do budúcnosti sa predpokladá, že Quarto bude viacjazyčný a využívaný pre ešte neexistujúce jazyky. [34],[38]

1.3 R Shiny

shiny je open-source balíček jazyka R, ktorý implementuje framework na tvorbu reaktívnych webových aplikácií. Prvýkrát bol predstavený na konferencii Joint Statistical Meeting v roku 2012 jeho autorom, ktorý bol v tom období aj technickým riaditeľom RStudia, Joem Chengom. Bol vytvorený za účelom uľahčenia tvorby reaktívnych webových aplikácii, reportov, analýz R programátorom, bez toho, aby museli ovládať ďalšie programovacie jazyky napr. HTML, CSS a JavaScript, ktoré sa používajú na tvorbu webových aplikácii.

Pod pojmom reaktívna webová aplikácia si môžeme predstaviť, že ak používateľ klikne na políčko a zvolí inú vstupnú hodnotu prípadne pohne sliderom, tak naprogramovaný kód na serveri načíta napríklad `.csv súbor`, vyberie podmnožínu dát alebo natrénuje model, čo následne vedie k výstupu v podobe opätovného vykreslenia grafov, tabuliek, máp atď. Čiže v jednoduchosti, pri zmene vstupu sa automaticky aktualizuje výstup.

Najčastejšie sa zvykne balíček **shiny** používať okrem tvorby webových aplikácií, aj na tvorbu dashboardov pre sledovanie vývoja výsledkov, tvorbu komplexných modelov na prezentovanie vizualizácií netechnicky zameranému publiku, vytváranie interaktívnych demo stránok pre učenie data-science/štatistiky na školách.

R Shiny aj **R Markdown** sa dajú prepojiť. To znamená, že **R Shiny** môžeme použiť v **R Markdown**. [15],[42]

1.3.1 HTML

HTML (*Hyper Text Markup Language*) je značkovací jazyk, ktorý slúži na vytváranie webových stránok. Opisuje štruktúru webovej stránky t.j. ako má stránka vyzerať (odseky, nadpis, menu). Používa sa spolu s CSS a JavaScriptom na dizajnovanie webových stránok. HTML kód pozostáva z viacerých elementov, pomocou ktorých prehliadač vykresľuje obsah danej stránky. Elementy môžu byť rôzneho typu. Súbory majú príponu `.html` respektíve `.htm`. Základná štruktúra HTML kódu pozostávajúca z hlavičky a tela vyzerá nasledovne [40]:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>My First Heading</h1>
    <p>My first paragraph.</p>
  </body>
</html>
```

1.3.2 CSS

CSS (*Cascading Style Sheets*) je jazyk, ktorý opisuje, ako sa majú elementy jazykov HTML, XML (vrátane SVG, MathML, XHTML) na stránke zobraziť. Používa sa napríklad na zmenu farby pozadia, zmenu veľkosti, typu a fontu písma. Súbory majú príponu `.css`. CSS môžeme vytvoriť buď v jednom súbore a neskôr ho nalinkovať do HTML súboru viacerými spôsobmi, to znamená, že všetky podstránky danej webovej stránky budú mať rovnaký vzhľad alebo môžeme samotný CSS kód písať do HTML súboru priamo, čo vyzerá nasledovne [11],[17]:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background-color: red;
      }
```

```
    </style>
  </head>
  <body>
    <h1>CSS language</h1>
    <p>This page has red background color.</p>
  </body>
</html>
```

1.3.3 JavaScript

JavaScript je skriptovací programovací jazyk, ktorý sa primárne využíva na implementovanie funkcionality na strane klienta. V praxi to znamená že zdrojový kód, vložený do webovej stránky, je vykonávaný priamo v prehliadači klienta. Umožňuje tak pridať do webovej stránky rôzne, zväčša dynamické funkcionality, ako napríklad interaktívne mapy, animovanú grafiku, automatické obnovenie obsahu, rolovacie menu, ktoré samotné HTML a CSS neumožňujú. Keďže sa skript vykonáva priamo u klienta, je možné implementovať dynamické funkcionality bez toho, aby musel klient obnovovať webovú stránku. Súbory majú príponu `.js`. Taktiež je JavaScript možné pridať priamo do HTML súboru do hlavičky alebo tela, alebo ho je možné nalinkovať. Pri pridaní do hlavičky HTML súboru vyzerá kód nasledovne [18],[41]:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function myFunction() {
        document.getElementById("demo").innerHTML = "Paragraph changed.";
      }
    </script>
  </head>
  <body>
    <h2>Demo JavaScript in Head</h2>
    <p id="demo">A Paragraph</p>
    <button type="button" onclick="myFunction()">Try it</button>
  </body>
</html>
```

1.4 LaTeX

LaTeX je typografický systém určený pre profesionálne a poloprofesionálne použitie. Je založený na sádzacom jazyku TeX, ktorý je doplnený o makrá. Jeho filozofiou je, že autor textu sa má zameriavať len na samotný text a dizajn výsledného dokumentu nastavuje dizajnér separátne od obsahu. Je vydaný pod LPPL licenciou (*LaTeX Project Public Licence*) a v zásade sa jedná o voľne šíriteľný program.

Okrem príkazov, ktoré ponúka samotný LaTeX sú k dispozícii aj rozšírenia v podobe balíkov. Tieto balíky dopĺňajú funkcionalitu LaTeXu a zároveň zjednodušujú prácu autorovi pri tvorbe dokumentu napríklad tým, že namiesto zložitého nastavovania funkcií a štýlov použije autor prednastavené makro z balíka. LaTeX má k dispozícii centrálny repozitár balíkov Comprehensive TeX Archive Network (*CTAN*). Rôzne distribúcie LaTeXu už obvykle obsahujú početnú sadu balíkov (fonty a makrá), ktoré umožňujú plnohodnotne pracovať s dokumentom bez nutnosti inštalácie špecializovanejších balíkov z CTANu.

Zo zdrojového kódu je kompilovaný výsledný výstupný formát pomocou LaTeXového nástroja. Zvolený nástroj číta syntax zdrojového kódu, fonty a makrá ktoré spracováva a na základe zdrojového kódu vytvorí výstup. Výstupný formát ako aj špecifické aspekty zdrojového kódu určuje typ LaTeXového nástroja. Medzi ne patrí `latex`, ktorý je pôvodným nástrojom a generuje výstup vo formáte DVI, `pdflatex` generuje PDF dokumenty, `xelatex` podporuje Unicode vstup a fonty vo formátoch `.ttf` a `.otf` a `lualatex`, ktorý okrem Unicode a moderných fontov zahŕňa aj podporu Lua skriptovania.

V súčasnosti predstavuje LaTeX najobľúbenejšiu platformu pre tvorbu akademických a vedeckých publikácií. [16]

1.5 NGINX

NGINX je open-source HTTP server, reverzné proxy, generické UDP/TCP proxy a POP3/IMAP proxy server. Vyznačuje sa jednoduchosťou, modulárnosťou, škálovateľnosťou, nízkymi nárokmi na výkon a univerzálnosťou. Medzi najčastejšie využitia, okrem použitia ako HTTP web server, patria HTTP reverzné proxy, HTTP load balancer a kešo-

vanie HTTP obsahu. V súčasnosti využíva NGINX, ako HTTP web server alebo reverzné proxy, vyše 20% webových stránok na internete. [19],[24]

1.6 HTTP/HTTPS protokol

Hypertextový prenosový protokol (*Hypertext Transfer Protocol*) je protokol aplikačnej vrstvy určený najmä na prenos hypermédií. HTTP je využívaný primárne na prenos dát medzi webovými servermi a klientmi (prehliadačmi). Najpoužívanejšou verziou je protokol HTTP/1.1 a opisuje ho štandard RFC 2616.

HTTP protokol obohatený o kryptografické opatrenia, pomocou SSL alebo TLS protokolu, je HTTPS (*Hypertext Transfer Protocol over SSL*). HTTPS opisuje štandard RFC2817. [25],[26]

1.7 WebSocket

WebSocket je komunikačný protokol umožňujúci obojsmernú komunikáciu medzi klientom a serverom. Umožňuje aplikáciám na strane klienta obojsmernú komunikáciu so serverom, bez nutnosti vytvárania viacerých HTTP spojení. WebSocket protokol opisuje štandard RFC 6455. V súvislosti s touto prácou používa protokol WebSocket rozhranie **R Shiny**. [28],[43]

1.8 SSL a TLS

Secure Sockets Layer (*SSL*) je kryptografický protokol určený na zabezpečenie komunikácie medzi dvomi zariadeniami v počítačovej sieti. Jeho najnovšiu verziu, SSL 3.0 opisuje štandard RDC 6101. V súčasnosti SSL nahrádza modernejší protokol TLS, ktorý je bezpečnejší, napriek tomu je protokol SSL na internete stále vo veľkom využívaný. V technickej praxi dochádza častokrát ku zámene pojmov SSL a TLS, keďže oba protokoly majú rovnaký účel. [27]

Transport Layer Security (*TLS*) je kryptografický protokol určený na zabezpečenie

komunikácie medzi dvomi zariadeniami v počítačovej sieti. Jedná sa o modernejšiu alternatívu ku SSL, pričom rozdiel je najmä v implementácií kryptografických funkcií. Účel a použitie, najmä v prípade HTTPS protokolu zostáva rovnaký. Aktuálnu verziu TLS 1.3 opisuje štandard RFC 8446. [29]

2 Cieľ práce

Hlavným cieľom diplomovej práce je vytvoriť návod pre rozhrania jazyka R, a to **R Markdown** a **R Shiny**, ktorý prevedie čitateľa od technického fungovania oboch rozhraní, cez základné príkazy a funkcie, ktorými sa naučí tvorbu základných dokumentov a aplikácií až po reálne nasadenie webovej aplikácie. Aby sme naplnili hlavný cieľ, zadefinovali sme si nasledovné čiastkové ciele:

- Opis jazyka R, použitého vývojového prostredia a ďalších náležitých pojmov k rozhraniam.
- Opis technickej špecifikácie oboch rozhraní a analýza jednotlivých funkcií a ich detailný opis a vysvetlenie.
- Implementácia niektorých opísaných funkcií na výstupných formátoch vo forme PDF dokumentu, HTML dokumentu, **R Markdown** dashboardu a takisto **R Shiny** webovej aplikácie.
- Analýza možností nasadenia webovej aplikácie a opis nasadenia.

Získané vedomosti bude čitateľ schopný použiť prakticky ihneď, vďaka natívnosti oboch rozhraní, vrátane ich prepojenia.

Po prečítaní tejto práce bude čitateľ poznať pokročilé nástroje na tvorbu dokumentov a aplikácií v jazyku R, bude ich vedieť prakticky použiť a v prípade potreby aj nasadiť.

3 Metodika práce

3.1 Technická špecifikácia R Markdown

Z technického hľadiska využíva **R Markdown** viacero procesov, ktoré vzájomnou spoluprácou vytvárajú rôzne typy formátov. Nižšie si opíšeme jednotlivé nástroje a vzťahy medzi nimi, na ktorých je založená tvorba výstupných formátov.

3.1.1 Nástroj Pandoc

Pandoc je konvertor, ktorý umožňuje konvertovať veľké množstvo markup formátov do iných formátov, vhodnejších na grafickú prezentáciu, ako sú `.doc`, `.pdf`. Pod pojmom markup si môžeme predstaviť, že daný formát resp. jazyk kombinuje text a informáciu o texte tzv. metainformáciu, ktorá hovorí o logickej štruktúre alebo spôsobe prezentácie textu.

Nástroj **Pandoc** je samostatný program nezávislý od R a nemá vlastné grafické používateľské rozhranie. Keďže balíček **`rmarkdown`** je priamo závislý na konverzii dokumentov pomocou **Pandocu**, je **Pandoc** priamo zahrnutý v každej novej verzii jazyka R.

3.1.2 Balíčky a značkovacie jazyky

R Markdown je značkovací jazyk, ktorý je založený na pôvodnom značkovacom jazyku **Markdown**. Keďže sa zvykne zamieňať s balíčkami jazyka R, rozdiely medzi nimi si opíšeme nižšie.

Značkovací jazyk Markdown

Markdown je odľahčený značkovací jazyk s jednoduchým formátovaním syntaxe, ktorý môže byť konvertovaný do HTML a ďalších formátov. Samotný **Markdown** súbor je jednoduchý textový súbor a má príponu `.md`. Rovnako ako ostatné značkovacie jazyky ako napríklad HTML, nemá žiadnu spojitosť s jazykom R.

Dodnes nebol špecifikovaný všeobecný **Markdown** štandard. Kvôli neexistujúcemu štandardu píšu autori rôzne varianty tohto jazyka, aby buď opravili nedostatky alebo doplnili funkcie.

Balíček `markdown`

`markdown` je balíček jazyka R, ktorý konvertuje `.Rmd` súbory do HTML. Je to predchodca balíčka **`rmarkdown`**. Už spomenutý balíček **`rmarkdown`** ponúka viacero funkcionalít a vylepšení, preto sa odporúča pôvodný balíček **`markdown`** už nepoužívať.

Značkovací jazyk R Markdown

R Markdown je rozšírenie pôvodnej **Markdown** syntaxe. **R Markdown** súbory sú jednoduché textové súbory, ktoré majú príponu `.Rmd`. Súbory **R Markdown** píšeme pomocou rozšírenej **Markdown** syntaxe, ktorá umožňuje vkladanie R zdrojových kódov tak, aby mohli byť aj spúšťané a vo finálnom dokumente zobrazované aj ich výstupy. Pri **R Markdown** súboroch, ktoré budú spracovávané pomocou balíčka **`rmarkdown`**, je možné takisto používať aj Pandoc markdown syntax, ktorá rozširuje **Markdown** o ďalšie funkcionality.

Balíček `rmarkdown`

`rmarkdown` je balíček jazyka R, ktorý spracúva a konvertuje `.Rmd` súbory do veľkého množstva výstupných formátov. Jednou z najdôležitejších funkcií je `rmarkdown:render`, ktorá je založená na nástroji **Pandoc**. Táto funkcia vyrenderuje vstupný súbor pomocou **Pandocu** na špecifikovaný výstupný súbor. Ak vstupný súbor obsahuje vložený kód, tak na jeho spracovanie sa zavolá funkcia `knitr::knit`, ktorá je volaná prioritne pred **Pandocom**.

Na začiatku **R Markdown** súboru sa nachádza YAML hlavička, ktorá obsahuje metadáta, na základe ktorých môže funkcia `rmarkdown::render` meniť rôzne nastavenia renderovania.

3.1.3 Balíček `knitr`

`knitr` je balíček jazyka R, ktorý spracováva jednoduchý textový dokument s vloženým zdrojovým kódom, vykoná ho a zapíše výsledok späť do dokumentu. Konvertuje:

- **R Markdown** (`.Rmd`) súbor na štandardný **Markdown** (`.md`) súbor.
- Sweave (`.Rnnw`) súbor do `.tex` formátu.
- (`.Rhtml`) súbor do html.

Najdôležitejšia funkcia tohto balíčka, `knitr::knit`, plní tri úlohy. Prvou úlohou je parsovanie R zdrojového kódu, vloženého do spracovávaného súboru. Funkcia `knit()` sa pozrie na vstupný súbor, ktorý dostane od používateľa, zhodnotí, v ktorých častiach dokumentu sa nachádza kód, ktorý chce používateľ vykonať a ten posunie do R. Druhú nemenej dôležitú úlohu zastáva vykresľovač výsledkov, ktorý výsledok exekúcie kódu zapíše do dokumentu vo formáte, ktorý je interpretovaný ako surový výsledný typ. Napríklad, ak knitrom spracovávame `.Rmd` súbor, výstup zdrojových kódov je vo výslednom `.md` súbore označený Markup značkami.

Dôležitou vecou, ktorú treba spomenúť je, že samotný balíček **`knitr`** nekonvertuje medzi dokumentovými formátmi ako napríklad `.md` na `.html`. Aj keď neumožňuje priamo konvertovať súbory do výstupných formátov, disponuje skôr podpornými funkciami, ktoré pomáhajú s konverziou dokumentov iným balíčkom. Napríklad funkcia `knitr::knit2html` nevykonáva sama konverziu, ale volá funkciu `markdown::markdownToHTML`. Táto funkcia a jej správanie je pozostatkom z čias, než balíček **`rmarkdown`** predbehol balíček **`markdown`**. [1],[30]

3.1.4 R Markdown workflow

V kapitolách 3.1.1, 3.1.2, a 3.1.3 sme si definovali jednotlivé súčasti **R Markdown**, od ktorých závisí finálna tvorba dokumentu. Teraz si krok po kroku uvedieme, čo sa deje počas spracovania dokumentu.

Súbor do ktorého píšeme text a zároveň kód vo forme code chunks, a v ktorom je zahrnutá YAML hlavička s metadátami má príponu `.Rmd`. Ako prvú vykoná svoju úlohu funkcia `knit()` z balíčka **`knitr`**, ktorá parsuje celý zdrojový kód vložený v `.Rmd` súbore, zabezpečí jeho exekúciu v R a pripraví výsledky na zobrazenie vo výslednom dokumente. Všetky výsledky kódov sú konvertované do správneho značkovacieho jazyka, ktorý bude obsiahnutý dočasne v súbore s príponou `.md`. Potom je `.md` súbor sprocesovaný nástrojom **Pandoc**, ktorý vezme základné parametre YAML hlavičky `title`, `author`, a `date`, prípadne iné parametre a skonvertuje dokument na želaný výstupný formát, ktorý je špecifikovaný v parametri `output` v YAML hlavičke.

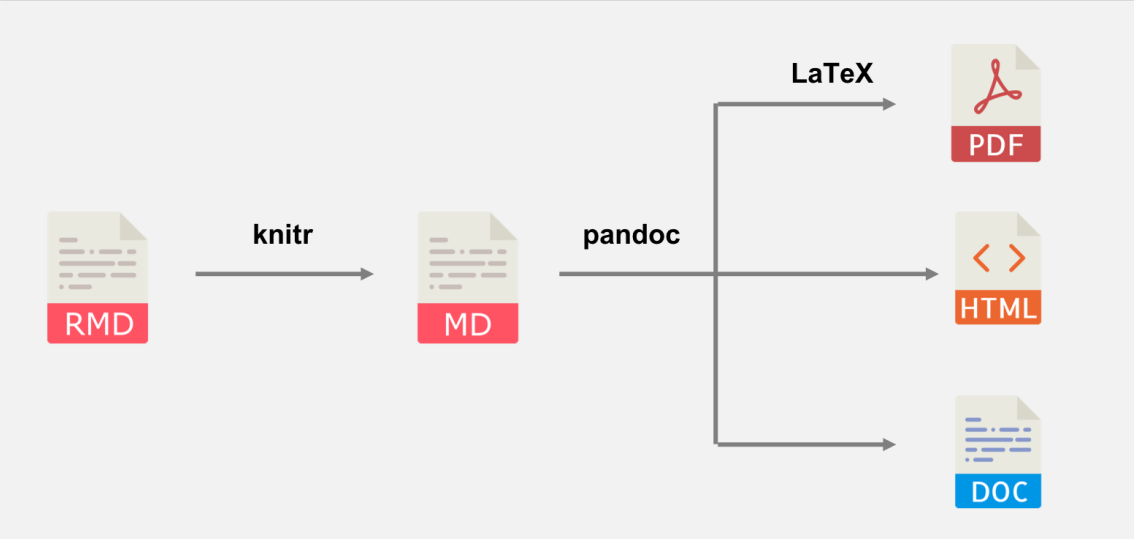
Špecifický prípad nastáva, keď je výstupným formátom PDF dokument. Na jeho spracovanie sa vyžaduje dodatočná vrstva. **Pandoc** skonvertuje dočasný `.md` súbor, do dočasného `.tex` súboru. Následne je `.tex` súbor sprocesovaný LaTeXom do finálneho PDF dokumentu. To zabezpečuje balíček **`rmarkdown`**, ktorý zavolá funkciu `latexmk()` z balíčka **`tinytex`**, ktorá pomocou LaTeXu skompiluje `.tex` súbor do `.pdf` súboru a vznikne tak PDF dokument.

Zjednodušene sa vieme nato pozrieť takto:

`rmarkdown:render` = `knitr::knit` + Pandoc (+ LaTeX pre PDF dokument)

Nie všetky **R Markdown** dokumenty je nutné konvertovať pomocou **Pandocu**. Dočasný `.md` môže byť skompilovaný aj prostredníctvom ďalších **Markdown** kompilátorov, ktoré sú vo forme R balíčkov:

- Balíček **`xaringan`** umožňuje sprocesovať `.md` výstup do JavaScriptovej knižnice, ktorá vykreslí obsah **Markdownu** do webového prehliadača.
- Balíček **`blogdown`** podporuje `.Rmarkdown` dokumentový formát, ktorý je balíčkom **`knitr`** spracovaný do súboru `.markdown` a tento **Markdown** dokument sa zvyčajne vyrenderuje do HTML. [30],[45]



Obrázok 2: Diagram ilustrujúci priebeh konvertovania **R Markdown** dokumentu do finálneho výstupného dokumentu, zdroj: [45]

3.2 Typy výstupných formátov

Ako sme si už uviedli v kapitole 1.2 **R Markdown** umožňuje tvoriť viacero výsledných formátov. Najčastejšie používaným výstupným formátom sú dokumenty, ich jednotlivé typy sú uvedené nižšie.

- `pdf_document` je najpoužívanejší výstupný formát z dokumentov. S LaTeXom, ktorý je open-source dokumentový formát, vieme vytvoriť výsledný PDF dokument.
- `word_document` je formát pre Microsoft Word dokumenty *(.docx)*.
- `odt_document` je formát pre OpenDocument Text dokumenty *(.odt)*.
- `rtf_document` je formát pre Rich Text Format dokumenty *(.rtf)*.
- `md_document` formát pre **Markdown** dokument. Sám o sebe sa používa len v prípade, keď spolupracovníci používajú **Markdown**.
- `github_document` je formát, ktorý je vytvorený na zdieľanie na GitHube.

Jedným z ďalších výstupných formátov sú tzv. Notebooks `html_notebook`, ktoré sú variáciou klasického `html_document`. Rozdiel je len v tom, že `html_document` sa pou-

žíva na prezentovanie výsledkov, zatiaľ čo `html_notebook` na spoluprácu medzi viacerými spolupracovníkmi. `html_notebook` takisto obsahuje po vyrenderovaní celý zdrojový kód narozdiel od HTML dokumentu.

Prezentácie patria taktiež k jedným z výstupných dokumentov. Ovládanie síce nie je až tak intuitívne ako pri klasickom PowerPointe, ale ak chceme do slajdov vkladať výsledky R kódov, môže nám to ušetriť veľa času. Typy jednotlivých prezentácií sú uvedené nižšie.

- `ioslides_presentation` je HTML prezentácia s ioslides.
- `slidy_presentation` je HTML prezentácia s W3C Slidy.
- `beamer_presentation` je PDF prezentácia s Latex Beamer.
- `powerpoint_presentation` je PowerPoint prezentácia.
- `revealjs::revealjs_presentation` je HTML prezentácia založená na `reveal.js` frameworku.

Dashboardy sú ďalším typom výstupného formátu, ktorý sa používa na zobrazenie veľkého množstva informácií prehľadne a rýchlo tak, aby používateľovi bola podaná jasná informácia vo forme grafov. K tomuto účelu môžeme použiť balíček **`flexdashboard`**.

Ďalším možným výstupným formátom sú webové stránky, v tomto prípade sa YAML hlavička bude dosť líšiť a bude potrebné do nej pridať rôzne HTML, prípadne CSS súbory. Následne ju vieme cez príkaz `render_site` nasadiť v statickej forme.

Všetky vyššie uvedené HTML výstupné formáty môžeme zinteraktívniť pomocou HTML widgets resp. `htmlwidgets` vo forme napríklad interaktívnej mapy. Medzi balíčky `htmlwidgets` podporujúce interaktivitu zaraďujeme:

- **`dygraphs`** balíček pre vizualizácie časových radov.
- **`DT`** balíček pre interaktívne tabuľky.
- **`rthreejs`** balíček pre interaktívne 3D grafy.
- **`DiagrammeR`** balíček pre diagramy ako sú flowcharty a sieťové grafy.

Ich výhoda spočíva v tom, že nepotrebujeme vedieť HTML alebo JavaScript, aby sme ich mohli použiť. K ďalším balíčkom, ktoré podporujú interaktivitu patrí aj **shiny**. o tomto balíčku si viac povieme v kapitole 3.5. [13],[44]

3.3 Štruktúra R Markdown dokumentu

R Markdown dokument pozostáva z troch častí, ktoré si detailne rozoberieme. Ide konktétne o YAML hlavičku, code chunks (*bloky kódov*) a telo dokumentu. Pred samotným rozobraním jednotlivých častí si budeme musieť najprv nainštalovať balíček a to príkazom `install.packages("rmarkdown")`. Do projektu ho explicitne pripájať netreba. Ak nemáme vytvorený projekt, tak najprv si ho vytvoríme cez **File** → **New Project** → **New Directory** → **New Project** → **Directory Name** = zadáme názov projektu → **Create Project**. Ak už máme projekt, do ktorého chceme pridať `.rmd` súbor, vytvoríme ho cez **File** → **New File** → **R Markdown**, následne si zvolíme výstup a vyplníme parametre `title`, `author` a `date`.

3.3.1 YAML hlavička

YAML hlavička musí byť súčasťou každého dokumentu. Je umiestnená na začiatku dokumentu. Špecifikuje sa v nej, ako bude dokument skonvertovaný a čo bude v hlavičke dokumentu napísané. Jej hranice sú zhora a zdola označené symbolom troch pomlčiek `---`. YAML hlavička napríklad pri HTML dokumente vyzerá takto:

```
---
title: "R Markdown dokument"
author: "Alena Stracenska"
date: "2022-10-29"
output: html_document
---
```

V hlavičke taktiež môžeme deklarovať parametre pre parametrizované reporty s názvom `params`, ku ktorým môžeme následne pristupovať takto `params$year`. Viac nastavení pre parametre v YAML hlavičke môžeme vidieť na obrázku 3, kde následne napravo vidíme, pre ktorý výstupný formát sú dané parametre dostupné a v opise majú taktiež uvedené aké hodnoty môžu nadobúdať. [44],[45]

IMPORTANT OPTIONS	DESCRIPTION	HTML	PDF	MS Word	MS PPT
anchor_sections	Show section anchors on mouse hover (TRUE or FALSE)	X			
citation_package	The LaTeX package to process citations ("default", "natbib", "biblatex")		X		
code_download	Give readers an option to download the .Rmd source code (TRUE or FALSE)	X			
code_folding	Let readers to toggle the display of R code ("none", "hide", or "show")	X			
css	CSS or SCSS file to use to style document (e.g. "style.css")	X			
dev	Graphics device to use for figure output (e.g. "png", "pdf")	X	X		
df_print	Method for printing data frames ("default", "kable", "tibble", "paged")	X	X	X	X
fig_caption	Should figures be rendered with captions (TRUE or FALSE)	X	X	X	X
highlight	Syntax highlighting ("tango", "pygments", "kate", "zenburn", "textmate")	X	X	X	
includes	File of content to place in doc ("in_header", "before_body", "after_body")	X	X		
keep_md	Keep the Markdown .md file generated by knitting (TRUE or FALSE)	X	X	X	X
keep_tex	Keep the intermediate TEX file used to convert to PDF (TRUE or FALSE)		X		
latex_engine	LaTeX engine for producing PDF output ("pdflatex", "xelatex", or "lualatex")		X		
reference_docx/_doc	docx/pptx file containing styles to copy in the output (e.g. "file.docx", "file.pptx")			X	X
theme	Theme options (see Bootswatch and Custom Themes below)	X			
toc	Add a table of contents at start of document (TRUE or FALSE)	X	X	X	X
toc_depth	The lowest level of headings to add to table of contents (e.g. 2, 3)	X	X	X	X
toc_float	Float the table of contents to the left of the main document content (TRUE or FALSE)	X			

Obrázok 3: Parametre, ktoré môže nadobúdať YAML hlavička pre výstupné formáty, zdroj: [8]

3.3.2 Telo dokumentu

Telo dokumentu pozostáva z **Markdown** syntaxe, na ktorej je založený **R Markdown**. Jednotlivé zložky syntaxe si opíšeme nižšie.

Inline formátovanie

Samotné inline formátovanie nám umožňuje nastaviť časť textu odlišne od zvyšku textu. Typ písma *italic* nastavíme prostredníctvom hviezdičiek alebo podčiarkovníkov napríklad `_text_` alebo `*text*`. Typ písma **Bold** nastavíme pomocou hviezdičiek takto `**text**`. Pomocou páru tíld ~ nastavíme dolné indexy `H~3~PO~4~`, čo vypíše H_3PO_4. Naopak horné indexy nastavíme cez znak mocniny ^ takto `Cu^2+^`, čo nám zobrazí Cu^{2+}.

Ak chceme vyznačiť inline kód, čiže zvýrazniť kód v texte použijeme spätné apostrofy ``. Napríklad `code`, čo nám vypíše `code`.

Pre vytváranie odkazov môžeme použiť syntax `[text](link na stránku)` napríklad `[RStudio](https://www.rstudio.com)`. Podobná syntax sa používa aj ku vkladaniu obrázkov `![názov alebo opis obrázka](cesta k obrázku)`, napríklad pomocou nasledujúceho príkazu `![Logo jazyka R](/Users/strac/Desktop/DP/obrazky/logor.jpg)` by sme vložili logo jazyka R. Poznámku pod čiarou môžeme pridať pomocou mocniny a hranatých zátvoriek `^[]`, napríklad `^[Toto je poznámka pod čiarou]`.

Ak chceme vložiť bibliografiu do PDF dokumentu, odporúča sa používať LaTeXové balíky `natbib` alebo `biblatex`. Ich použitie je uvedené v kapitole 3.4.1. [44]

Tvorba nadpisov, odrážiek a zoznamov

Úrovne nadpisov môžeme písať prostredníctvom mriežky `#`. Mriežka sa ale používa aj na pridávanie komentárov, no pri nadpisoch musí byť oddelená medzerou. V PDF dokumente sa dá nastaviť až 6 úrovní nadpisov. Štandardne sa ale využívajú maximálne 3 úrovne nadpisov.

```
# Prvá úroveň
## Druhá úroveň
### Tretia úroveň
```

Ak nechceme, aby nám očíslovalo jednotlivé úrovne nadpisov, môžeme za ne vložiť buď príkaz `{-}` alebo `{.unnumbered}`. Oba príkazy treba písať v množinových zátvorkách (*tzv. curly braces*).

```
# Prvá úroveň {-}
## Druhá úroveň {.unnumbered}
```

Neočíslovaný zoznam odrážok s vnorenými pododrážkami môžeme vytvoriť takto:

```
- prva odrazka
- druha odrazka
- tretia odrazka
    - prva pododrazka
    - druha pododrazka
    - tretia pododrazka
```

Dôležité je mať nechať jeden riadok voľný, aby sa odrážky vypísali správne. Naopak očíslovaný zoznam odrážok aj s vnorenými pododrážkami môžeme vytvoriť takto:

```
1. prva odrazka
2. druha odrazka
3. tretia odrazka
    - prva pododrazka
    - druha pododrazka
```

Citáty píšeme prostredníctvom znaku väčší, ktorý vyzerá takto `>`. Znak je potrebné napísať v každom riadku daného citátu. Ak chceme pridať autora vpravo dole, treba si doinštalovať balíček **`tufte`** a z neho použiť funkciu `quote_footer()`.

```
> "First, solve the problem. Then, write the code."
>
> `r tufte::quote_footer('--- John Johnson')`
```

Zvýrazniť kód môžeme ako sme už spomínali pomocou spätného apostrofu `` `code` `` alebo pomocou troch spätných apostrofov s dodatkom `markdown`, ktorý nám zvýrazní pozadie kódu nasivo. Ďalším spôsobom je napísanie troch spätných apostrofov pred a za daný blok kódu, ktorý chceme zvýrazniť. ```` ```code``` ````. Taktiež môžeme zvýrazniť kód pridaním štyroch medzier pred daný kód.

````
``` markdown
Tento kusok kodu je zvyrazneny na sivom pozadi.
```
    Zvyrazneny kusok kodu po styroch medzerach.
```Zvyrazneny kusok kodu.```
````

Matematické výrazy

Matematické LaTeXové inline rovnice/výrazy môžeme písať prostredníctvom LaTeXovej syntaxe dvoch dolárov `$matematicky_vyraz$`, napríklad `$f(k) = {n \choose k} p^{k} (1-p)^{n-k}$` nám zobrazí Bernoulliho vetu $f(k) = \binom{n}{k} p^k (1-p)^{n-k}$. Ak chceme zobraziť výraz mimo inline módu použijeme dva páry dolárov `$$matematicky_vyraz$$`, do ktorých výraz uzavrieme `$$f(k) = {n \choose k} p^{k} (1-p)^{n-k}$$`, výstup následne môžeme vidieť nižšie v strede riadku.

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k}$$
````

Takisto môžeme použiť rôzne matematické módy uprostred syntaxe jedného alebo dvojitého páru dolárov. [20],[44],[45]

```
$$\begin{array}{ccc}
x_{11} & x_{12} & x_{13}\\
x_{21} & x_{22} & x_{23}
\end{array}$$
```

Čo nám následne vypíše toto:

$$\begin{array}{ccc} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{array}$$

Matice môžeme napísať takto:

```
$$X = \begin{bmatrix}1 & x_{1}\\
1 & x_{2}\\
1 & x_{3}
\end{bmatrix}$$
```

Čo nám vypíše danú maticu:

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \end{bmatrix}$$

Nižšie uvedieme zopár základných matematických výrazov zapísaných LaTeXovou syntaxou.

```
#zakladna syntax rovnic
$$
E = mc^2
$$
```

Výsledok:

$$E = mc^2$$

```
#zlomky
$$
\frac{1}{2}
$$
```

Výsledok:

$$\frac{1}{2}$$

```
#dolne indexy
$$
Y = X_1 + X_2
$$
#horne indexy
$$
a^2 + b^2 = c^2
$$
```

Výsledok:

$$Y = X_1 + X_2$$

Výsledok:

$$a^2 + b^2 = c^2$$

```
#odmocniny
$$
\sqrt{p}
$$
#mocniny
$$
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
$$
```

Výsledok:

$$\sqrt{p}$$

Výsledok:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
#sumy
$$
\sum_{i = 1}^{n}{(\bar{x} - x_i)^2}
$$
```

Výsledok:

$$\sum_{i=1}^{n}(\bar{x} - x_i)^2$$

```
#Bayesova veta
$$
Pr(\theta | y) = \frac{Pr(y | \theta) Pr(\theta)}{Pr(y)}
$$

$$
Pr(\theta | y) \propto Pr(y | \theta) Pr(\theta)
$$
```

Výsledok:

$$Pr(\theta|y) = \frac{Pr(y|\theta)Pr(\theta)}{Pr(y)}$$
$$Pr(\theta|y) \propto Pr(y|\theta)Pr(\theta)$$

```
#Linearny model
$$
Y \sim X\beta_0 + X\beta_1 + \epsilon
$$

$$
\epsilon \sim N(0,\sigma^2)
$$
```

Výsledok:

$$Y \sim X\beta_0 + X\beta_1 + \epsilon$$

$$\epsilon \sim N(0, \sigma^2)$$

3.3.3 Code chunks

Do tela dokumentu môžeme vkladať aj code chunks, ktoré sa následne exekuujú do výsledného formátu. Výsledkom kódu môže byť napríklad graf, tabuľka, resp. výstup nejakej funkcie. Priestor, v ktorom sa nachádzajú code chunks je vymedzený tromi apostrofmi ``` zhora aj zdola.

````
```{r}
data_vstupne <- read.csv("dataLA.csv", sep = ",")
premenna_a~<- lm(y~x, data = data_vstupne)
summary(premenna_a)
```
````

Vyššie uvedený code chunk by nám následne vo výslednom dokumente vypísal tabuľku k lineárnej regresii. Do samotných množinových zátvoriek môžeme vypísať viacero parametrov napríklad `echo`, ktoré nám hovorí o tom, či do výsledného formátu chceme, alebo nechceme zahrnúť vypísanie zdrojového kódu, potom `warning`, `message`, `error`, ktoré nám hovoria, či chceme alebo nechceme zahrnúť warningy, erory a správy do výsledného kódu. Užitočným parametrom je aj `out.width` a `out.height`, ktoré nám umožňujú nastaviť šírku a výšku obrázkov v percentách vo výstupných formátoch a `fig.cap` nám umožní pridať názov obrázku.

Výhoda code chunks je, že nie sú dedikované len na jazyk R, do množinových zátvoriek si môžeme napísať, aký jazyk chceme použiť. Napísaním nižšie uvedeného príkazu do konzoly `names(knitr::knit_engines$get())` môžeme zistiť, aké všetky jazyky sa dajú použiť. [44]

```
> names(knitr::knit_engines$get())
 [1] "awk"       "bash"      "coffee"    "gawk"      "groovy"    "haskell"   "lein"      "mysql"
 [9] "node"      "octave"    "perl"      "php"       "psql"      "Rscript"   "ruby"      "sas"
[17] "scala"     "sed"       "sh"        "stata"     "zsh"       "asis"      "asy"       "block"
[25] "block2"    "bslib"     "c"         "cat"       "cc"        "comment"   "css"       "ditaa"
[33] "dot"       "embed"     "eviews"    "exec"      "fortran"   "fortran95" "go"        "highlight"
[41] "js"        "julia"     "python"    "R"         "Rcpp"      "sass"      "scss"      "sql"
[49] "stan"      "targets"   "tikz"      "verbatim"
```

Obrázok 4: Všetky dostupné jazyky pre code chunks, zdroj: [44]

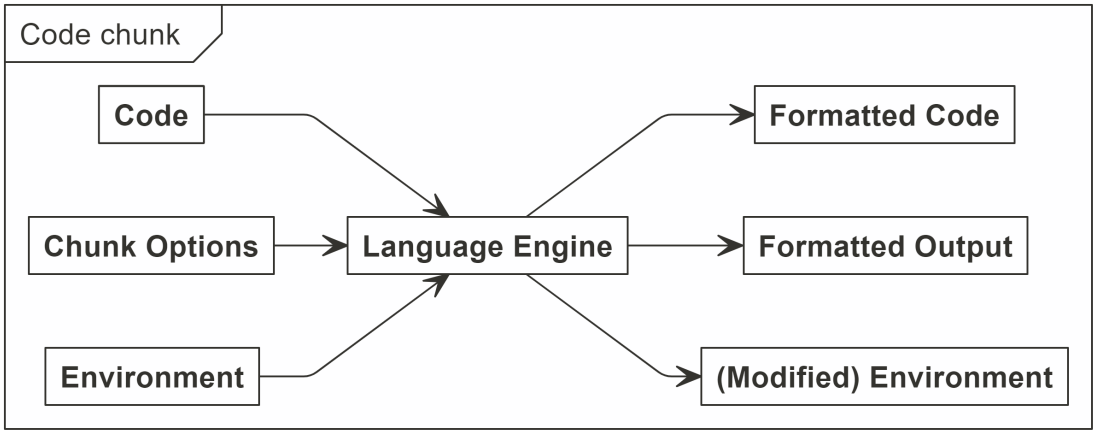
Vyberme si napríklad jazyk SQL, ktorým chceme pristupovať k databáze. Do prvého code chunku si cez jazyk R spravme pripojenie na lokálnu databázu.

````
```{r}
#pripojenie balicka do projektu
library(RMySQL)
mysqlconnection = dbConnect(RMySQL::MySQL(),
                            dbname='pekaren',
                            host='localhost',
                            port=3306,
                            user='root',
                            password='')
```
````

Následne napíšme do chunku SQL dopyt nad databázou a skúsme code chunk označiť prívlastkom sql v množinových zátvorkách.

````
```{sql, connection = mysqlconnection, max.print = 1000}
SELECT hodina, COUNT(hodina) AS `pocet_vyskytov`, dp.nazov_ulice, dp.mesto
FROM predaj_platba pp
left join d_cas dc on dc.id_cas = pp.id_cas
left join d_predajna dp on dp.id_predajna = pp.id_predajna
GROUP BY hodina
ORDER BY `pocet_vyskytov` DESC
```
````

Výsledkom bude vypísaná tabuľka dopytu nad databázou. Takto môžeme napríklad využiť jednotlivé jazyky uvedené na Obrázku 4.



Obrázok 5: Proces znázorňujúci vstupy a výstupy pri exekúcii code chunks, zdroj: [45]

3.4 Nastavenia výstupných formátov

Keďže výstupných formátov je veľké množstvo, vybrali sme si tri najčastejšie používané formáty, ktoré si detailne opíšeme.

3.4.1 PDF dokument

YAML hlavička PDF dokumentu vyzerá na začiatku takto [45]:

```
---
title: "R Markdown dokument"
author: "Alena Stracenska"
date: "2022-10-29"
output: pdf_document
---
```

Parameter `title` nám hovorí o názve, `author` o mene autora prípadne autorov, `date` o dátume a parameter `output` o dokumente, ktorý na konci vznikne. Do hlavičky môžeme okrem základných parametrov spomenutých pridať aj veľké množstvo ďalších parametrov. Jedným z nich je parameter `toc`, ktorý umožňuje pridať obsah a pomocou parametra `toc_depth` môžeme oddeliť jednotlivé úrovne obsahu. V `pdf_document` je `toc_depth` defaultne nastavený na úroveň 2, kým v `html_document` je nastavený na úroveň 3. Ďalší podstatný parameter `number_sections` nám očísluje jednotlivé úrovne obsahu a zároveň parameter `toc-title` nám umožňuje zmeniť názov obsahu napríklad z anglického Contents na slovenský Obsah. Ďalšími parametrami, ktoré môžeme pridať sú

`subtitle` a už spomínaný systémový dátum `date`. Parametrov, ktorých môžeme pridať je samozrejme viac. V PDF dokumente sa najčastejšie využívajú LaTeXové parametre, ktoré dopĺňajú dokument o viacero funkcionalít. Prehľadne si ich vysvetlíme a rozoberieme v ďalšej časti tejto kapitoly. [44]

```
---
title: "R Markdown dokument"
subtitle: "Toto je dokument pre účely diplomovej práce"
author: "Alena Stracenska"
date: "`r format(Sys.time(), '%d %B %Y')`"
output:
  pdf_document:
    toc: true
    toc_depth: 3
    number_sections: true
toc-title: "Obsah"
---
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
Prvá úroveň
Druhá úroveň
Tretia úroveň
```

Následne bude výstup vyzerať takto:

R Markdown dokument

Toto je dokument pre účely diplomovej práce

Alena Stracenska

30 október 2022

Obsah

1 Prvá úroveň **1**

1.1 Druhá úroveň 1

1.1.1 Tretia úroveň 1

1 Prvá úroveň

1.1 Druhá úroveň

1.1.1 Tretia úroveň

Obrázok 6: Nastavenie obsahu v YAML hlavičke, zdroj: [45]

K ďalším parametrom, ktoré môžeme pridať do YAML hlavičky patrí napríklad `df_print`, ktorý umožní zobraziť dátové rámce (*tzv. dataframes*). Zoznam všetkých možných hôdnôt, ktoré môže parameter nadobúdať v `pdf_document` sú uvedené nižšie [44]:

Tabuľka 1: Hodnoty, ktoré môže nadobúdať parameter `df_print`

Hodnota	Opis
default	volá generickú metódu print.data.frame
kable	používa funkciu knitr::kable()
tibble	používa funkciu tibble::print.tbl_df()
vlastná funkcia	použitie vlastnej vytvorenej funkcie

Zdroj: [44]

Pomocou parametra `highlight` vieme zvýrazniť želanú syntax kódu. Disponuje niekoľkými štýlmi ako sú `default`, `tango`, `pygments`, `kate`, `monochrome`, `espresso`, `zenburn`, `haddock`, `breezedark`, `arrow` a `rstudio`. [44]

```
---
title: "R Markdown dokument"
author: "Alena Stracenska"
date: "`r format(Sys.time(), '%d %B %Y')`"
output:
  pdf_document:
    highlight: kate
    df_print: kable
---
## Zobrazenie dataframe & zvyraznenie syntaxe
```{r}
head(iris,10)
```
```

Hlavičku sme pri nasledujúcom výstupe nezobrazili, výstup oboch parametrov je uvedený na obrázku 7. Pri štandardnom zobrazení datasetu by nám ho vykreslilo vo forme, ktorá nebude až tak prepracovaná ako pri použití parametra `df_print`.

Je potrebné ešte podotknúť, že pre renderovanie PDF dokumentu je potrebné mať nainštalovaný balíček **`tinytex`**, v prípade ak ho nainštalovaný nemáme, RStudio nám ponúkne možnosť jeho inštalácie.

Zobrazenie dataframe & zvyraznenie syntaxe

```
head(iris,10)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

Obrázok 7: Pokročilé možnosti nastavenia pre PDF dokument v YAML hlavičke, zdroj: [45]

Ak už spomenuté parametre `df_print` a `highlight` nastavíme, tak ich môžeme použiť v celom kóde a platia pre celý kód. To znamená, že rovnakým štýlom sa vykreslia všetky dátové rámce a aj syntax bude rovnako zvýraznená.

Už skôr sme si uviedli, že k vytvoreniu PDF dokumentu potrebujeme LaTeX. Veľká časť z jeho parametrov môže byť taktiež použitá v YAML hlavičke. Napríklad parameter `fontsize: 12pt` nám umožňuje nastaviť veľkosť písma v celom dokumente (akceptované veľkosti sú 10, 11 a 12), ak zadáme inú hodnotu, tak nám RStudio vypíše chybovú hlášku. Parameter `geometry: margin=1in` nám zase hovorí o veľkosti okrajov.

```
---
title: "R Markdown dokument"
author: "Alena Stracenska"
output:
  pdf_document:
    latex_engine: xelatex
fontsize: 12pt
geometry: margin=1in
---
```

Tabuľka 2: Hodnoty, ktoré nadobúdajú LaTeXové parametre v YAML hlavičke

Parameter	Opis
lang	Kód daného jazyka (Document language code).
fontsize	Veľkosť písma (10pt, 11pt alebo 12pt).
documentclass	LaTeXový typ dokumentu (napr. article, report, beamer).
classoption	Nastavenia pre documentclass (napr. oneside).
geometry	Nastavenie pre geometry class (napr. margin=1in).
mainfont, sansfont, monofont, mathfont	Fonty na použitie v LaTeX dokumentoch (funguje len s xelatexom alebo lualatexom).
linkcolor, urlcolor, citecolor	Farby k odkazom a citáciám.

Zdroj: [44]

Defaultne sú PDF dokumenty renderované pomocou nástroja `pdflatex`. V prípade, ak by nám z nejakého dôvodu nefungoval `pdflatex`, môžeme alternatívne použiť `xelatex` alebo `lualatex`. Obe spomínané alternatívy lepšie podporujú Unicode a systémové fonty.

```
---
title: "R Markdown dokument"
output:
  pdf_document:
    latex_engine: xelatex
---
```

Bibliografiu je možné vytvoriť pre všetky výstupné formáty prostredníctvom **Pandoc** nástroja `pandoc-citeproc`. Pre výstup vo forme PDF dokumentu je lepšie použiť LaTeXové balíky `natbib` alebo `biblatex`. Nastavíme ich použitím parametra `citation_package`.

```
---
title: "R Markdown dokument"
output:
  pdf_document:
    citation_package: natbib
---
```

V YAML hlavičke môžeme globálne nastaviť aj výšku a šírku výstupov, prípadne či chceme k nim aj opis. a to prostredníctvom parametrov `fig_width`, `fig_height` a `fig_caption`

pre opis obrázkov. Treba však dávať pozor na to, že jednotné nastavenie sa týka len grafov resp. výstupov z naprogramovaných kódov a nie z vložených `.jpg`, `.png` a rôznych iných obrázkových formátov.

```
---
title: "R Markdown dokument"
output:
  pdf_document:
    fig_width: 7
    fig_height: 6
    fig_caption: true
---
```

Do YAML hlavičky môžeme taktiež vložiť parameter `header-includes` do ktorého môžeme pridávať LaTeXový kód, prípadne balíky, ktoré môžeme následne použiť priamo v dokumente. [44],[45]

```
header-includes:
  - \usepackage{amsmath}
```

Obrázok z externého zdroja vo formáte `.jpg`, `.png` alebo iných by sme do dokumentu mohli pridať cez nižšie uvedený LaTeXový kód a následne v množinových zátvorkách v R code chunku nastavíme jeho šírku, názov a vložíme ho cez `knitr::include_graphics()`.

```
header-includes:
  - \renewcommand{\figurename}{Obrázok č.}
  - \makeatletter
  - \def\fnum@figure{\textbf{\figurename\nobreakspace\thefigure}}
  - \makeatother
---
```{r echo=FALSE, out.width='100%', fig.cap = "Zdrojová stránka dát", echo=FALSE}
knitr::include_graphics('./data.png')
```
```

3.4.2 HTML dokument

Keďže bol **Markdown** na začiatku dizajnovaný pre výstupný formát v podobe HTML dokumentu, tomu zodpovedá aj počet parametrov, ktoré má zo všetkých formátov najviac. Niektoré parametre má s ostatnými formátmi spoločné a niektoré unikátne.

V parametri `output` nastavíme ako výstupný formát `html_document`. Obsah pridávame prostredníctvom parametra `toc`. Parameter `toc_depth` je defaultne nastavený na úroveň obsahu 3, ak ho nenastavíme inak. Veľmi praktickým parametrom je `toc_float`, ktorý umožní zobrazenie obsahu naľavo aj pri scrollovaní. Má 2 nastavenia a to `collapsed` a `smooth_scroll`, ktoré sú defaultne nastavené na `true` a pri scrollovaní sa pomocou nich rozbaľuje menu.

Ďalším užitočným parametrom je `.tabset`, ktorý umožňuje zobraziť jednotlivé položky obsahu vo forme horizontálnych buttonov/tlačidiel. Dva parametre s názvami `.tabset-fade` a `.tabset-pills` umožňujú nastaviť dodatočné funkcie napríklad permanentné zvýraznenie zvoleného buttonu.

```
## Štvrťročné výsledky {.tabset .tabset-fade .tabset-pills}
### Produkty
(obsah buttonu)
### Mestá
(obsah buttonu)
```

Do HTML dokumentu je možné pridať `.css` súbor použitím parametra `css` v hlavičke. Ak použijeme vlastný CSS súbor tak parametre `highlight` a `theme` musíme nastaviť na `null`.

V prípade ak nemáme vlastný `.css` súbor, tak na nastavenie témy/pozadia dokumentu nám slúži parameter `theme`, ktorý má nasledovné témy a to `default`, `bootstrap`, `cerulean`, `cosmo`, `darkly`, `flatly`, `journal`, `lumen`, `paper`,`readable`, `sandstone`, `simplex`, `spacelab`, `united` a `yeti`. S parametrom `highlight` môžeme zvýrazniť želanú syntax, vyberáme z nasledujúcich tém: `default`, `tango`, `pygments`, `kate`, `monochrome`, `espresso`, `zenburn`, `haddock`, `breezedark` a `textmate`.

Nastavenia obrázkov sú podobné ako pri PDF dokumente, takisto aj to, aké hodnoty môže nadobúdať parameter `df_print`. Jednu hodnotu `df_print` nadobúda naviac a ňou je `paged`, alebo `rmarkdown::paged_table`. Ďalšie možnosti pre nastavenie parametra `df_print: paged` v HTML sú uvedené v tabuľke 3. [44]

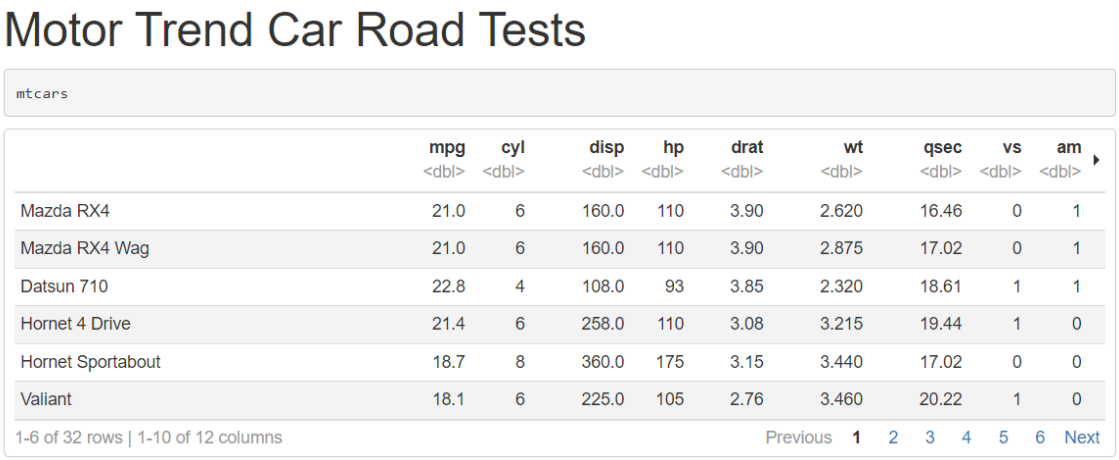
Tabuľka 3: Možnosti nastavenia pre tabuľku v HTML dokumente

Parameter	Opis
max.print	Vypíše počet zadaných riadkov.
rows.print	Vypíše počet zadaných riadkov s možnosťou prepínania na ďalšie riadky.
cols.print	Vypíše počet zadaných stĺpcov s možnosťou prepínania na ďalšie stĺpce.
cols.min.print	Vypíše minimálny počet stĺpcov na zobrazenie k šírke obrazovky.
pages.print	Vypíše možnosť prepínania pre zadaný počet.
paged.print	Pri nastavení FALSE sa vypne možnosť prepípania.
rownames.print	Pri nastavení FALSE sa odstránia názvy riadkov.

Zdroj: [44]

V nasledujúcom kóde môžeme vidieť použitie parametra `df_print: paged` v YAML hlavičke a potom v chunku aj zobrazenie maximálne šiestich riadkov cez parameter `rows.print = 6`. Výslednú tabuľku môžeme vidieť na obrázku 8, na ktorom vidíme následne možnosť prepínania na ďalšie strany tabuľky.

```
---
title: "Motor Trend Car Road Tests"
output:
  html_document:
    df_print: paged
---
```{r, rows.print = 6}
mtcars
```
```



Obrázok 8: Vizualizácia výstupu použitím parametra `df_print` a `rows.print`, zdroj: [44]

Pomocou parametra `code_folding` a hodnoty `hide`, čiže `code_folding: hide` vieme skryť celý kód. V podstate je to button, na ktorý kliknutím skryjeme a ďalším kliknutím zobrazíme kód. Parameter môže nadobúdať aj hodnotu `show`. [44]

3.4.3 Dashboard

Ako sme si spomínali v kapitole 3.2. dashboardy zobrazujú aktuálne dianie. Sú jedným z pokročilých výstupných formátov **R Markdown** a na ich tvorbu sa využíva balíček **`flexdashboard`**. Hlavné pravidlo pri tvorbe tohto výstupného formátu je určiť si rozloženie celého dashboardu. V našom prípade si vieme v hlavičke nastaviť parameter `orientation` na `columns` alebo `rows`. Parameter `vertical_layout` môže nadobúdať 2 hodnoty a to `fill` a `scroll`. `fill` sa zvyčajne používa, keď máme 1 alebo 2 grafy vo vertikálnej polohe, naopak `scroll`, keď máme 3 a viac grafov vertikálne uložených. [2]

```
---
title: "Untitled"
output:
  flexdashboard::flex_dashboard:
    orientation: columns
    vertical_layout: fill
---
```{r setup, include=FALSE}
library(flexdashboard)
```
Column {data-width=650}
```

````
-----------------------------------------------------------------------
### Chart A

```{r}
```
Column {data-width=350}
-----------------------------------------------------------------------
### Chart B

```{r}
```
### Chart C

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

V RStudio sa k vyššie uvedenému kódu dostaneme cez **File** → **New File** → **R Markdown** → **From Template** → **Flex Dashboard**. Rozloženie dashboardu podľa kódu bude teda stĺpec o šírke 650 teda Chart a (Graf A), potom dva stĺpce o šírke 350 Chart B a Chart C (Graf B a Graf C), ktoré budú pod sebou vedľa Grafu A. Do R code chunks môžeme následne vložiť kód obohatený o rôzne funkcie z HTML widgets balíčkov podporujúcich interaktivitu, ktorého výstupom bude zobrazený graf, prípadne tabuľka v danom stĺpci.

Takisto môžeme nastaviť dashboard tak, aby mal viacero stránok, cez ktoré sa dá preklikávať cez buttony. Spravíme to prostredníctvom viacerých znakov = za sebou. Nižšie je uvedený kód.

````
---
title: "Multiple Pages"
output: flexdashboard::flex_dashboard
---
Visualizations {data-icon="fa-signal"}
=====================================
### Chart 1
```{r}
```
### Chart 2
```{r}
```
Tables {data-icon="fa-table"}
=====================================
````

```
### Table 1

```{r}
```
Table 2
```{r}
```
```

Existuje veľa rôznych typov rozložení, ktoré si môžeme prispôsobiť. Teraz sa pozrime nato, akými komponentami je možné vyplniť jednotlivé stĺpce alebo riadky dashboardu.

Jedným z často používaných komponentov je valueBox, prislúcha mu funkcia `valueBox()`, ktorá zobrazuje ikonu a názvom a počtom daných položiek danej ikony. Napríklad mali by sme ikonu koša, názov Počet spamov za deň a číslo 15.

Ďalším typom komponentov sú Gauges, u nás známe ako metrika výkonu niečoho v nejakom intervale. Tomuto komponentu slúži na vykreslenie funkcia `gauge()`, ktorá má povinné 3 parametre a to `value`, `min` a `max`. Takisto môžeme `gauge()` špecifikovať parametrom `symbol`, ktorý bude zobrazovať napríklad `%`. Funkcia `gauge_sectors()` nám umožňuje špecifikovať množinu troch stavov, a to `success`, `warning` alebo `danger`. Označenie ako sme zvyknutí býva v poradí zelená, oranžová a červená, ale je možné si ho prispôsobiť podľa seba.

V prípade, ak potrebujeme do dashboardu dopísať dodatočné informácie k niečomu vieme tak urobiť buď dopísaním textu za YAML hlavičku, predtým, než rozdelíme dashboard na boxy, alebo ho jednoducho môžeme napísať do boxov pomocou symbolu väčší `>`, kde nevložíme graf, ale len text.

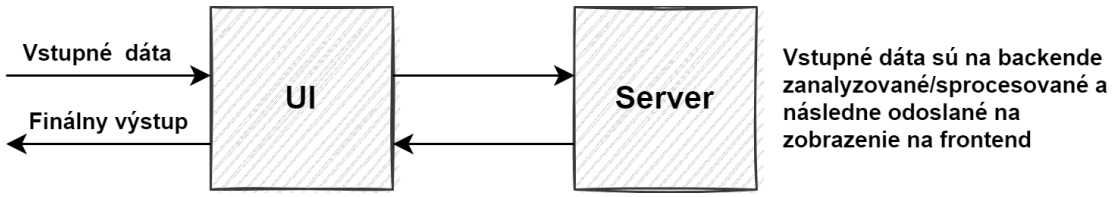
Zinteraktívniť naše grafy a metriky môžeme dvoma spôsobmi a to pomocou balíčka **`shiny`** alebo prostredníctvom HTML widgets. Pri HTML widgets sa najčastejšie využívajú balíčky **`Leaflet`**, **`dygraphs`**, **`Plotly`**, **`rbokeh`**, **`Highcharter`**, **`visNetwork`**, **`networkD3`**, **`DataTables`**, **`threejs`**, **`rglwidget`**, **`DiagrammeR`** a **`MetricsGraphics`**. [23],[44]

3.5 Technická špecifikácia R Shiny

Ako sme si už spomenuli v kapitole 1.3 balíček **`shiny`** slúži na tvorbu reaktívnych webových aplikácií bez nutnosti vedieť programovacie jazyky dedikované na tvorbu takýchto aplikácií. Samotná štruktúra Shiny webovej aplikácie pozostáva z 3 častí:

- Prvou z nich je User Interface (UI), čiže používateľské rozhranie (frontend aplikácie), ktoré zaisťuje vstup požiadaviek od používateľa. Takisto sa pomocou neho zobrazí výstup po spracovaní v serverovej časti. Zaužívané označenie je `ui.R`.
- Druhou z nich je Server function, je to backend aplikácie, zabezpečuje, aby sa používateľské vstupy spracovali respektíve premenili na výstupy a následne správne zobrazili na webovej stránke. Zaužívané označenie je `server.R`.
- Treťou časťou je shinyApp function, ktorá dáva dokopy frontend a backend aplikácie.

Proces fungovania Shiny aplikácie je pomerne jednoduchý. Na webovej stránke (UI, frontend), ktorú vidíme zadáme vstupné dáta, môžeme si pod nimi predstaviť výber dátumov, výber iných hodnôt, potiahnutie sliderom. Následne sa tieto vstupné dáta posunú do serverovej časti, ktorá ich zanalyzuje resp. sprocesuje a vyprodukovaný finálny výstup je následne odoslaný naspäť na frontend, kde sa zobrazí. [3],[35]



Obrázok 9: Workflow Shiny aplikácie, zdroj: [3]

3.6 Proces tvorby R Shiny aplikácie

Shiny aplikáciu môžeme najjednoduchšie vytvoriť v RStudio nasledovným spôsobom **File** → **New Project** → **Shiny Application** → **Directory name** = zadáme názov

aplikácie → **Create Project**. Následne nám zobrazí R skript s názvom `app.R`, v ktorom sú zahrnuté všetky tri časti Shiny aplikácie. Tento postup je vhodný pre vytváranie jednoduchých Shiny aplikácii. Takto vyzerá základná štruktúra Shiny aplikácie [42]:

```
library(shiny)
ui <- fluidPage(
)
server <- function(input, output) {
}
shinyApp(ui, server)
```

Pre vytváranie pokročilejších Shiny aplikácií je kvôli prehľadnosti lepšie rozdeliť si tieto 3 spomenuté komponenty do troch R skriptov s názvami napríklad `global.R` - tu budú pripojené balíčky a načítané dáta. V `ui.R` - budú definované používateľské vstupy a výstupy a v `server.R` bude celé analyzovanie/spracovanie dát. [3],[35]

3.6.1 Typy používateľských vstupov

Vstupy od používateľa píšeme do funkcie `fluidPage()`, ktorá nám v podstate vytvorí grafickú vizualizáciu stránky v závislosti od použitých elementov/komponentov a používateľských vstupov.

Textový vstup

Textový vstup ako už z názvu vyplýva umožňuje používateľovi napísať text do okna. Použijeme nato funkciu `textInput()`, ktorá nadobúda argumenty `inputID`, ktorým bude daný vstup volaný v celej aplikácii, `label` je názov, ktorý leží nad priestorom, kde používateľ vloží text, `value` je defaultný text, ktorý chceme mať predpísaný v priestore a ktorý musí používateľ zmazať, pred vložením textu. `placeholder` je podobný ako `value`, avšak pred vložením textu nemusíme nič mazať a zostane ako pozadie, ktoré po vložení textu zmizne a po jeho vymazaní sa zase objaví. Syntax funkcie je nasledovná:

```
textInput(inputId = "textInput", label = "Vlož text", value = "",
placeholder = "text")
```

Číselný vstup

Číselný vstup je v podstate rovnaký ako textový vstup s tým rozdielom, že miesto textu vkladáme číslo a funkcia sa nazýva `numericInput()`. Navyše sú argumenty `min` a `max`, ktoré nedovolia používateľovi zvoliť väčšiu alebo menšiu hodnotu mimo ich intervalu. Argument `step` nám umožňuje posúvať hodnotu čísel o zadanú hodnotu buď napríklad z 9 na 10, potom z 10 na 11. Syntax funkcie je nasledovná:

```
numericInput(inputId = "numericInput", label = "Vlož číslo",
value = 5, min = 1, max = 10, step = 1)
```

Označovacie políčko

Označovacie políčko známe aj ako checkbox umožňuje používateľovi označiť alebo odznačiť svoj výber. Funkcia `checkboxInput()` má rovnaké argumenty ako textový vstup, čo sa ale líši je binárna logika argumentu `value`. Ak bude `value = T`, políčka budú zobrazené ako označené. Naopak, ak bude `value = F` políčka sa zobrazia ako neoznačené. Syntax funkcie je nasledovná:

```
checkboxInput(inputId = "checkbox", label = "Označ alebo Odznač", value = T)
```

Existuje ešte jeden typ funkcie na označovanie políčok s názvom `checkboxGroupInput()`, ako už z názvu vyplýva umožňuje označiť viacero políčok naraz a nie iba jedno ako funkcia `checkboxInput()`. Rozdielnym argumentom oproti funkcii `checkboxInput()` je `choices`, kde do vektora uvedieme názvy daných políčiek.

```
checkboxGroupInput(inputId = "multi_checkbox", label = "Označ alebo Odznač",
choices = c("Nazov1", "Nazov2", "Nazov3"))
```

Rozbaľovací zoznam

Drop-down menu alebo rozbaľovací zoznam, z ktorého si následne môžeme zvoliť položky, ktoré chceme. Slúži nám na to funkcia `selectInput()`, ktorá sa od funkcie `checkboxGroupInput()` odlišuje argumentom `multiple`, ktorý umožňuje používateľovi zvoliť jednu alebo viacero položiek zo zoznamu. `multiple = T` pre viacero položiek a `multiple = F` pre jednu položku. Syntax funkcie je nasledovná:

```
selectInput(inputId = "select", label = "Vyber položku",
choices = c("Položka1", "Položka2", "Položka3"), multiple = T)
```

Posuvník

Posuvník alebo tiež známy ako slider umôžňuje používateľovi zvoliť maximálnu a minimálnu hodnotu z nejakej množiny hodnôt a používateľ následne vidí rozmedzie, napríklad na množine 10 čísel si zvolíme interval od 1 po 7. Zabezpečuje nám to funkcia `sliderInput()`, ktorá má všetky argumenty rovnaké ako funkcia `numericInput()`, ale líši sa logikou fungovania. Syntax funkcie je nasledovná:

```
sliderInput(inputId = "slider", label = "Zvoľ si rozmedzie",
min = 1, max = 10, value = 5, step = 1)

sliderInput(inputId = "range", label = "Zvoľ si rozmedzie",
value = c(10, 30), min = 0, max = 100)
```

Výber z viacerých označovacích políčok

Takzvané radiobuttons sú podobné ako označovacie a odznačovacie políčka, ale líšia sa v tom, že pri radiobuttons môžeme zvoliť len jedno z viacerých políčok. Slúži nám na to funkcia `radioButtons()`, ktorá má rovnaké argumenty ako funkcia `selectInput()`, okrem teda argumentu `multiple`. Syntax funkcie je nasledovná:

```
radioButtons(inputId = "radio", label = "Zvoľ možnosť",
choices = c("Možnosť1", "Možnosť2", "Možnosť3"))
```

Výber dátumov

Rozbaľovací zoznam, z ktorého si vie používateľ vybrať dátum umožňuje funkcia s názvom `dateInput()`, ktorá nadobúda okrem klasických argumentov `inputId` a `label` aj argument `value`, kde napíšeme dátum vo formáte `yyyy-mm-dd`, ak nenapíšeme nič, tak nám automaticky nastaví aktuálny dátum. Takisto pomocou argumentov `min` a `max` môžeme zvoliť dátumy, mimo ktorých rozmedzia nebude môcť používateľ voliť iný dátum. Syntax funkcie je nasledovná:

```
dateInput(inputId = "date", label = "Zvoľ dátum", value = "1993-06-30")
```

Pre dátumy existuje ešte jedna funkcia, ktorá umožňuje výber časového intervalu medzi dvoma dátumami, ktorá sa nazýva `dateRangeInput()`. Má extra argumenty `start` a `end` pre začiatočný a konečný dátum. Taktiež môžeme zvoliť `min` a `max` argumenty, ktoré sme spomenuli vo Výbere dátumov. Syntax funkcie je nasledovná:

```
dateRangeInput(inputId = "dateRange", label = "Vyber dátumy",
start = "2021-02-15", end = "2022-11-11")
```

Tlačidlá akcie (Action buttons)

Action buttons sú tlačidlá po ktorých stlačení sa refreshne výstup na stránke. Existuje pre ne funkcia `actionButton()` a `fluidRow()`, ktorá nám slúži na definovanie riadka, v ktorom budú uložené. Trochu sa odlišujú od klasických vstupov, ich argumenty sú z časti rovnaké, ale argument `class` je naviac a hovorí o stave a veľkosti buttonu. Môžeme si vybrať z týchto typov stavov buttonu: `"btn-primary"` `"btn-success"`, `"btn-info"`, `"btn-warning"`, `"btn-danger"` a z veľkostí: `"btn-lg"`, `"btn-sm"`, `"btn-xs"`. Cez stav `"btn-block"` nastavíme buttony na celú šírku obrazovky.

```
fluidRow(
    actionButton(inputId = "click",
    label =  "Click me!", class = "btn-danger"),
    actionButton(inputId = "clicktoo",
    label = "Click me too!", class = "btn-lg btn-success")
  ),
  fluidRow(
    actionButton(inputId = "confirm",
    label = "confirm!", class = "btn-block")
  )
```

Vloženie súboru

Vloženie súboru do Shiny aplikácie je taktiež možné, ale funkcia ktorá to zabezpečuje `fileInput()` vyžaduje špeciálne ošetrenie na strane servera. Daná funkcia môže nadobúdať argumenty `name`, `size` a `datapath`. Prvý argument nám hovorí o názve súboru, druhý o veľkosti a tretí o systémovej ceste k súboru. Ale ako sme si už spomenuli, keďže použitie funkcie `fileInput()` je mierne komplikované, tak sa ňou nebudeme zaoberať. Ak by ju bolo potrebné použiť, jej detailný opis je uvedený v dokumentácii. [3],[35],[42]

Zobrazenie vstupov v aplikácii

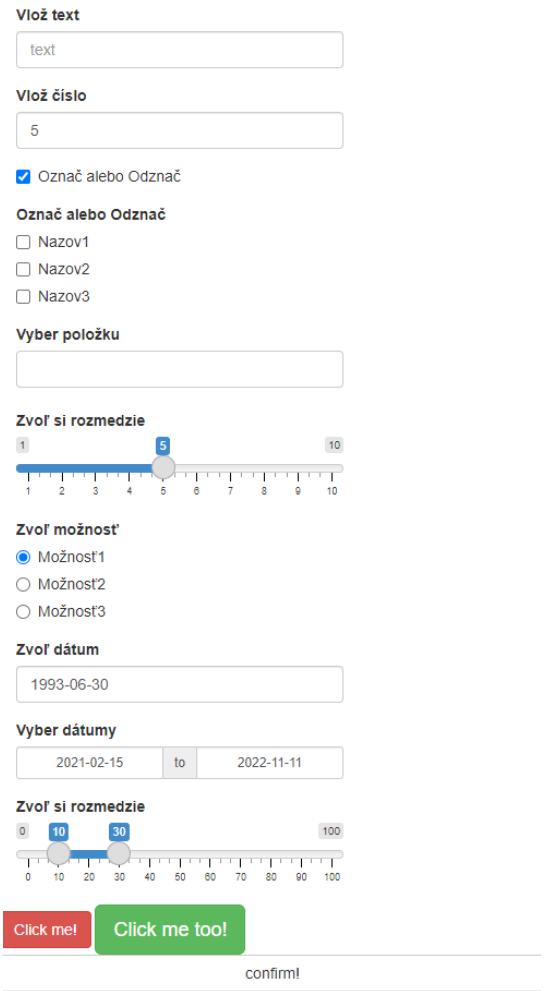
```
ui <- fluidPage(
  textInput(inputId = "textInput", label = "Vlož text", value = "", placeholder = "text"),
  numericInput(inputId = "numericInput", label = "Vlož číslo", value = 5, min = 1, max = 10, step = 1),
  checkboxInput(inputId = "checkbox", label = "Označ alebo Odznač", value = T),
  checkboxGroupInput(inputId = "multi_checkbox", label = "Označ alebo Odznač", choices = c("Nazov1", "Nazov2","Nazov3")),
  selectInput(inputId = "select", label = "Vyber položku", choices = c("Položka1", "Položka2", "Položka3"), multiple = T),
  sliderInput(inputId = "slider", label = "Zvoľ si rozmedzie", min = 1, max = 10, value = 5, step = 1),
  radioButtons(inputId = "radio", label = "Zvoľ možnosť", choices = c("Možnosť1", "Možnosť2", "Možnosť3")),
  dateInput(inputId = "date", label = "Zvoľ dátum", value = "1993-06-30"),
  dateRangeInput(inputId = "dateRange", label = "Vyber dátumy", start = "2021-02-15", end = "2022-11-11"),
  sliderInput(inputId = "range", label = "Zvoľ si rozmedzie", value = c(10, 30), min = 0, max = 100),
  fluidRow(
    actionButton(inputId = "click", label =  "Click me!", class = "btn-danger"),
    actionButton(inputId = "clicktoo", label = "Click me too!", class = "btn-lg btn-success")
  ),
  fluidRow(
    actionButton(inputId = "confirm", label = "confirm!", class = "btn-block")
  )

)

server <- function(input, output) {
}

shinyApp(ui = ui, server = server)
```

Obrázok 10: Kód znázorňujúci možné vstupy od používateľa, zdroj: [42]



Obrázok 11: Vizualizácia vstupov od používateľa, zdroj: [42]

3.6.2 Typy výstupov

Pri výstupoch budeme vkladať kód do funkcie `fluidpage()` a serverovej funkcie `function(input, output){}`.

Textový výstup

Pri textovom výstupe si potrebujeme v serverovej funkcii `function(input, output){}` zadefinovať unikátny identifikátor cez objekt `output` a operand dolára `$`, na ktorý sa následne budeme vedieť odvolať vo `fluidpage()`. Taktiež použijeme funkciu s názvom `renderText()`, do ktorej napíšeme text, ktorý chceme zobraziť. Následne do `fluidpage()` zadáme funkciu `verbatimTextOutput()`, v ktorej bude zobrazený unikátny identifikátor alebo funkciu `textOutput()`, prvá z nich má pevnú šírku fontu a je skôr vhodná pre text alebo kód, ktorý si môže používateľ nakopírovať. [7],[35],[42]

Do `function(input, output){}` napíšeme:

```
output$text1 <- renderText("Ukazka vypisania textu ")
```

Do `fluidpage()` napíšeme:

```
 verbatimTextOutput("text1")
```

Tabuľkový výstup

Logika pri tabuľkovom výstupe je rovnaká ako pri textovom výstupe s tým rozdielom, že potrebujeme dataset. Funkcia, ktorú napíšeme do serverovej funkcie `function(input, output){}` má názov `renderTable()` a ako parameter jej určíme zabudovaný dataset `iris`. Do `fluidpage()` napíšeme funkciu `tableOutput()`. [7],[35],[42]

Do `function(input, output){}` napíšeme:

```
data("iris")
output$tabulka~<- renderTable(iris)
```

Do `fluidpage()` napíšeme:

```
tableOutput("tabulka")
```

Výstup vo forme grafu

Tento typ výstupu má základ spoločný s ostatnými, ale líši sa funkciami. V serverovej funkcii `function(input, output){}` použijeme funkciu `renderPlot()` a vo `fluidpage()` použijeme funkciu `plotOutput()`. Pre grafy si musíme doinštalovať balíček **ggplot2** príkazom `install.packages("ggplot2")`. Následne ho musíme pripojiť do projektu príkazom `library(ggplot2)`. [7],[35],[42]

Do `function(input, output){}` napíšeme:

```
data("iris")
output$graf = renderPlot({
    ggplot(data=iris, aes(x = Petal.Length, y = Petal.Width)) + geom_point()
  })
```

Do `fluidpage()` napíšeme:

```
plotOutput("graf")
```

DT tabuľkový výstup

Tento výstup sa od klasického tabuľkového výstupu líši tým, že tabuľky sú krajšie vizualizované a sú interaktívne. Netreba zabudnúť na doinštalovanie balíčka **DT** a jeho pripojenie do projektu, rovnako ako sme to spravili s balíčkom **ggplot2**. V serverovej funkcii `function(input, output){}` používame funkciu `renderDataTable()`, v ktorej zvolíme dataset a dodatočné nastavenia cez `options`. `pageLength = 5` nám hovorí, že defaultne sa zobrazí tabuľka s piatimi riadkami a `lengthMenu = c(5, 10, 15, 20)` umožňuje si zvoliť z menu počet riadkov. Vo `fluidpage()` použijeme funkciu `dataTableOutput()`. [7],[35],[42]

Do `function(input, output){}` napíšeme:

```
data("iris")
output$dt <- renderDataTable(
iris, options = list(pageLength = 5,
                     lengthMenu = c(5, 10, 15, 20)))
```

Do `fluidpage()` napíšeme:

```
dataTableOutput("dt")
```

Dynamické UI

Dynamické vykresľovanie používateľského rozhrania UI (User Interface - vzhľad webovej aplikácie, stránky) pomocou funkcie `renderUI()` možno v **R Shiny** tiež považovať ako formu výstupu. Príslušný názov výstupu zadefinujeme vo `fluidPage()` pomocou funkcie `uiOutput("nazov")`. V tomto prípade by sme v serverovej funkcii `function(input, output){}` použili volanie funkcie `output$nazov<-renderUI()`, podobným spôsobom ako iný výstup. Do tejto funkcie potom zabalíme akýkoľvek UI element, ktorý potrebujeme vykresliť.

Jedným z možných použití tejto funkcie je dynamické vykresľovanie rôzneho počtu rovnakých elementov (napríklad viacero grafov), pričom konečný počet závisí od používateľovho vstupu alebo inej reakčnej premennej. Takáto implementácia je ale náročnejšia a vyžaduje si predprípravu zoznamu vykresľovaných elementov. [7],[35],[42]

3.6.3 Reaktivita

Reaktivita umožňuje používateľovi webstránky, meniť výstupy (napríklad vizualizácie), na základe jeho vstupu a bez nutnosti obnovenia celej stránky. V Shiny aplikácii sa celá logika reaktívneho programovania vykonáva v serverovej funkcii `function(input, output){}`. Táto funkcia určí, ktorý zdrojový kód sa má vykonať a ktorú používateľskú reláciu (session) ovplyvní. Pre každého aktívneho používateľa je vytvorená vlastná session s vlastnými objektami. Ak by používateľ 1 zmenil vstupnú hodnotu na stránke, tak používateľ 2 by zmenu nevidel a naopak, zmeny vstupu od používateľa 2 sa neprejavia používateľovi 1, lebo pre oboch používateľov je vytvorená vlastná session.

So vstupmi a výstupmi manipulujeme pomocou objektov `input` a `output`, ktoré sú parametrami serverovej funkcie (v našom prípade `function`). Tieto objekty sa správajú podobne ako zoznam, môžu mať viacero premenných - podľa vstupov a výstupov našej Shiny aplikácie.

Príklad reaktívneho kódu:

```
library(shiny)
ui <- fluidPage(
  textInput(inputId = "meno", label = "Ako sa voláš?"),
  textOutput("pozdrav")
)
server <- function(input, output) {
  output$pozdrav <- renderText({
  paste0("Ahoj ", input$meno, "!")
  })
}
shinyApp(ui = ui, server = server)
```

Z daného kódu by nám mohlo vyplývať že serverová funkcia len jednoducho a jednorázovo pošle reťazec do UI, ktoré následne zabezpečí jeho vypísanie na webovej stránke. No pri reaktívnom programovaní tento kód povie Shiny aplikácii, ako by mohla v prípade potreby, vytvoriť string. Kód sa potom môže vykonávať opakovane, no len vtedy, keď dôjde ku zmene reaktívneho vstupu. Cez objekt `input` a jeho premennú `meno`, pomocou operandu dolár `$`, čiže `input$meno`, pristupujeme v serverovej funkcii ku hodnote zadanej používateľom (`meno`), jeho hodnotu pomocou funkcie `renderText()` priradíme do premennej `output$pozdrav`, čo v podstate zabezpečuje, že sa nám vypíše text `Ahoj nami_zadane_meno !`. Pomocou uvedenej syntaxe môžeme takto pristupovať ku rôznym vstupom od používateľa. Je ale nutné podotknúť, že hodnoty uložené v `input` a `output` objektoch nie je možné meniť v zdrojovom kóde, len vstupom od používateľa.

Pri reaktívnom programovaní využívame aj niekoľko ďalších funkcií, okrem už spomenutých `renderText()`, `renderPlot()`, `renderDataTable()` a `renderUI()`, ktoré sa vykonajú stále, keď sa im zmení vstup od používateľa a slúžia na respektívne formy výstupu.

V prípade, ak by sme vstup od používateľa pred vypísaním alebo vykreslením do grafu chceli nejako spracovať (napríklad matematicky), je nutné použiť funkciu `reactive()`. Ako argument tejto funkcie môžeme použiť inú funkciu (alebo viac funkcií), ktorý chceme vykonať. Keď používateľ zmení vstup. funkcia zabezpečí vykonanie tohto kódu pri každej zmene. Takýmto systémom môžeme do reaktívneho procesu aplikácie vložiť blok, ktorý spojí vstup s výstupom.

```
library(shiny)
ui <- fluidPage(
  textInput(inputId = "meno", label = "Ako sa voláš?"),
  textOutput("pozdrav")
)
server <- function(input, output) {
  string <- reactive({
    paste0("Ahoj ", toupper(input$meno), "!")
  })
  output$pozdrav <- renderText(string())
}
shinyApp(ui = ui, server = server)
```

Funkciami `observe()` a `observeEvent()` docielime vykonanie konkrétneho zdrojového kódu, podobne ako u funkcie `reactive()`, no na rozdiel od nej tieto dve funkcie nevracajú žiadnu hodnotu. Rozdielom medzi `observe()` a `observeEvent()` je princíp vykonania. V prípade `observe()`, ak dôjde ku zmene akejkoľvek reakčnej premennej zabalenej v tejto funkcii, dôjde k jej vykonaniu. U funkcie `observeEvent()` je princíp trochu iný, ku vykonaniu zdrojového kódu dôjde, keď nastane zmena prvého argumentu tejto funkcie, zároveň však nebude zdrojový kód vykonaný ak príde ku zmene akejkoľvek inej reakčnej premennej. Toto môžeme využiť napríklad, ak chceme aby aplikácia počkala, kým používateľ zadá viacero vstupov a následne stlačí tlačítko.

`reactiveVal()` slúži na vytvorenie reaktívneho objektu ktorý ukladá jednu premennú. Z bežnej premennej tak vytvoríme premennú ktorá má reaktívne vlastnosti, čo môžeme využiť napríklad ak chceme vykonať určitú funkciu alebo zdrojový kód ak nastane zmena takejto premennej. Podobne funguje aj funkcia `reactiveValue()` s tým rozdielom, že na rozdiel od `reactiveVal()` nevytvárame len 1 premennú ale viacero premenných, ku ktorým pristupujeme rovnakým spôsobom ako ku bežnému zoznamu premenných.

Funkcia `eventReactive()` je aktivovaná rovnakým spôsobom ako funkcia s názvom `observeEvent()` - ak príde ku zmene jej prvého parametra, napríklad nejakou udalosťou. Na rozdiel od funkcie `observeEvent()` vracia funkcia `eventReactive()` výstup v podobe reaktívneho objektu.

Ako už názov funkcie `isolate()` napovedá, slúži na izolovanie reaktívnych premenných. Túto funkciu vieme použiť v iných reaktívnych funkciách, napríklad aby zmena vstup-

nej premennej neaktivovala funkciu, kde je daná premenná izolovaná, ale iba funkciu B. Iným možným využitím funkcie `isolate()` môže byť prerušenie nadväznosti viacerých reaktívnych premenných v zložitejších aplikáciách. [7],[35],[42]

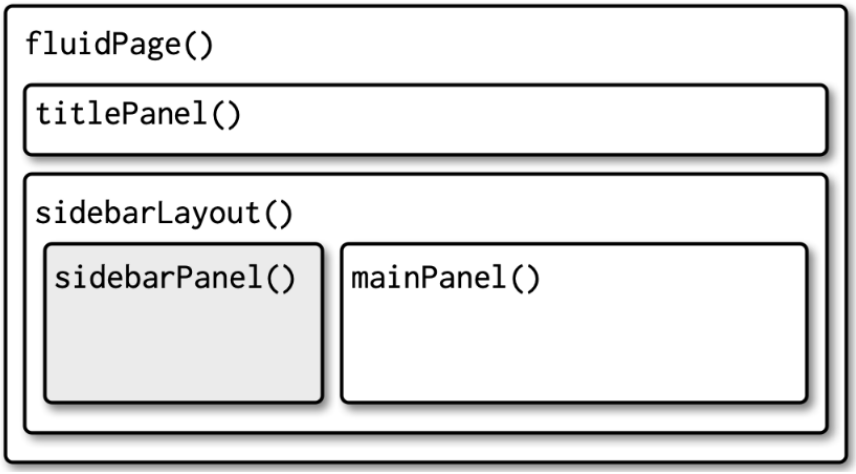
3.6.4 Rozloženie a témy

Shiny aplikácia ponúka veľa možnosti rozloženia komponentov/elementov na stránke a taktiež tvorby jednej alebo viacerých obrazoviek resp. stránok a podstránok pomocou rôznych funkcií.

Rozloženie komponentov na jednej obrazovke

Najdôležitejšou funkciou, do ktorej vkladáme všetky ostatné funkcie pre rozloženie komponentov je `fluidPage()`. Funkcia `sidebarLayout()` spolu s funkciami `titlePanel()`, `sidebarPanel()` a `mainPanel()` nám umožňujú vytvoriť rozloženie s dvoma komponentami v podobe stĺpcov, z ktorých jeden stĺpec bude naľavo, v ktorom budú vstupy od používateľa a stĺpec vpravo bude zobrazovať výstup vo forme napríklad grafu alebo inej vizualizácie. Stĺpce budú rozložené vo funkcii `sidebarLayout()`, `sidebarPanel()` nám následne zabezpečí stĺpec so vstupmi a v `mainPanel()` stĺpec s výstupom. Funkcia s názvom `titlePanel()` nám následne vytvorí priestor pre názov/opis aplikácie. [42]

```
fluidPage(
  titlePanel(
    # nazov/opis aplikacie pripadne menu
  ),
  sidebarLayout(
    sidebarPanel(
      # vstup/y
    ),
    mainPanel(
      # vystup/y
    )
  )
)
```



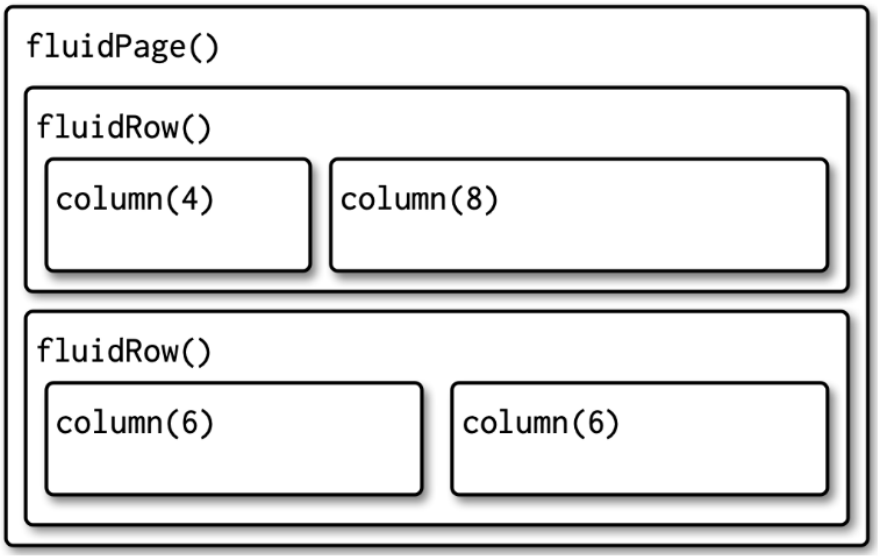
Obrázok 12: Štruktúra rozloženia komponentov na jednej obrazovke, zdroj: [42]

Rozloženie komponentov vo forme riadkov nám zabezpečí funkcia `fluidRow()`, pomocou `column()` vytvoríme v daných riadkoch stĺpce.

```
fluidPage(
  fluidRow(
    column(4,
      ...
    ),
    column(8,
      ...
    )
  ),
  fluidRow(
    column(6,
      ...
    ),
    column(6,
      ...
    )
  )
)
```

Aby sme danému kódu pochopili presnejšie, je nutné podotknúť, že každý riadok sa skladá z 12 stĺpcov a prvý argument funkcie `column()` nám hovorí, koľko daných stĺpcov chceme obsadiť. Pomocou tohto princípu môžeme vytvárať hocijaké iné rozloženia, ktoré budeme potrebovať pre aplikáciu. Príklad štruktúry rozloženia vyššie uvedeného kódu je

na obrázku 13. [42]



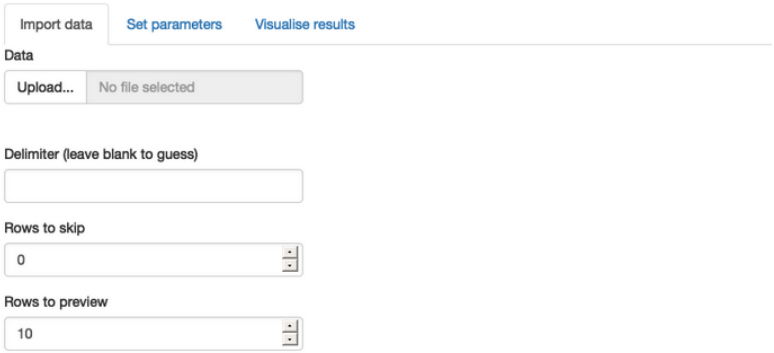
Obrázok 13: Štruktúra rozloženia riadkových komponentov na jednej obrazovke, zdroj: [42]

Tvorba viacerých obrazoviek

Pri komplexnejších Shiny aplikáciách je lepšie rozdeliť webovú stránku na viacero obrazoviek. To nám umožňuje funkcia `tabsetPanel()`, ktorá vytvára kontajner resp. miesto pre viacero funkcií `tabPanel()`, ktoré môžu následne obsahovať rôzne vstupy od používateľa. Funkciu `tabsetPanel()` je možné použiť hocikde v kóde prípadne ju vnoriť do viacerých `tabsetPanel()` funkcií. [42]

```
ui <- fluidPage(
  tabsetPanel(
    tabPanel("Import data",
      fileInput("file", "Data", buttonLabel = "Upload..."),
      textInput("delim", "Delimiter (leave blank to guess)", ""),
      numericInput("skip", "Rows to skip", 0, min = 0),
      numericInput("rows", "Rows to preview", 10, min = 1)
    ),
    tabPanel("Set parameters"),
    tabPanel("Visualise results")
  )
)
```

Vo vyššie uvedenom kóde už nemusíme písať argumenty ako `inputId` alebo `label`, ich nadobúdajúce hodnoty stačí uviesť do úvodzoviek. Na základe postupnosti argumentov im už funkcia vie priradiť správne hodnoty.



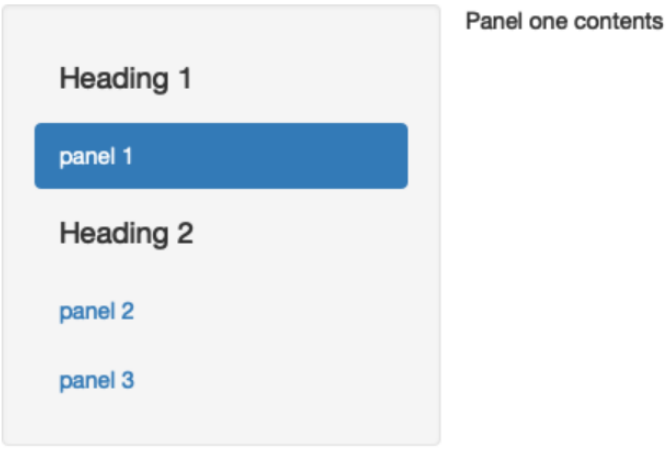
Obrázok 14: Vizualizácia výstupu viacerých obrazoviek, zdroj: [42]

Tvorba navigačného panela

Navigačný panel môže byť buď horizontálny alebo vertikálny. Umožňujú to funkcie `navbarPage()` a `navbarMenu()` a `navlistPanel()`. Funkcia `navlistPanel()` je podobná funkcii `tabsetPanel()`, ale narozdiel od nej vykreslí názvy položiek navigačného panelu vertikálne. [42]

```
ui <- fluidPage(
  navlistPanel(
    id = "tabset",
    "Heading 1",
    tabPanel("panel 1", "Panel one contents"),
    "Heading 2",
    tabPanel("panel 2", "Panel two contents"),
    tabPanel("panel 3", "Panel three contents")
  )
)
```

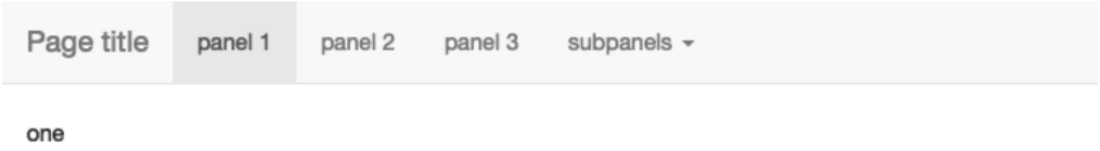
Na obrázku 15 môžeme vidieť vertikálny navigačný panel, obdobne na obrázku 16 môžeme vidieť horizontálny navigačný panel.



Obrázok 15: Vizualizácia vertikálneho navigačného panela, zdroj [42]

Na inom princípe sa používa funkcia `navbarPage()`, ktorá vykreslí názvy položiek navigačného panelu horizontálne, ale pomocou funkcie `navbarMenu()`, môžeme pridať drop-down rozbaľovacie menu pre danú položku navigačného panela. [42]

```
ui <- navbarPage(
  "Page title",
  tabPanel("panel 1", "one"),
  tabPanel("panel 2", "two"),
  tabPanel("panel 3", "three"),
  navbarMenu("subpanels",
    tabPanel("panel 4a", "four a"),
    tabPanel("panel 4b", "four b"),
    tabPanel("panel 4c", "four c")
  )
)
```



Obrázok 16: Vizualizácia horizontálneho navigačného panela, zdroj: [42]

Témy

Väčšina tém je nielen v Shiny ale v celom jazyku R založená na Bootstrape. Bootstrap je framework jazyka CSS, ktorý umožňuje zobraziť elementy danej webovej stránky v jednom štýle. My sa ale nebudeme zaoberať samotným Bootstrap frameworkom, keďže Shiny aplikácia už má v sebe zabudované funkcie, ktoré nám ho automaticky implementujú. Okrem Bootstrapu môžeme použiť aj iné CSS frameworky, ktoré sú súčasťou nasledujúcich balíčkov:

- Balíček **`shiny.semantic`** vytvorený Appsilonom, vznikol ako nadstavba formantic UI.
- Balíček **`shiny.Mobile`** vytvorený RInterface, vznikol ako nadstavba pre framework 7 a bol nadizajnovaný pre mobilné aplikácie.
- Balíček **`shinymaterial`** vytvorený Ericom Andersonom vznikol ako nadstavba nad Google Material design frameworkom.
- Balíček **`shinydashboard`** vytvorený RStudiom, poskytuje rozloženie pre tvorbu dashboardov. [42]

Užitočným balíčkom pre použitie rôznych Bootstrap tém je balíček **`bslib`**. Môžeme použiť už vytvorenú, alebo si vytvoriť vlastnú tému pomocou argumentu `theme`. Aby sme si ale zjednodušili tvorbu aplikácie, použijeme už vopred pripravenú **bootswatch** tému použitím argumentu `bootswatch`. Možné témy, z ktorých môžeme vyberať sú `Cerulean`, `Cosmo`, `Cyborg`, `Darkly`, `Flatly`, `Journal`, `Litera`, `Lumen`, `Lux`, `Materia`, `Minty`, `Morph`, `Pulse`, `Quartz`, `Sandstone`, `Simplex`, `Sketchy`, `Slate`, `Solar`, `Spacelab`, `Superhero`, `United`, `Vapor`, `Yeti` a `Zephyr`. [37],[42]

```
ui <- fluidPage(
  theme = bslib::bs_theme(bootswatch = "darkly"),
  sidebarLayout(
    sidebarPanel(
      textInput("txt", "Text input:", "text here"),
      sliderInput("slider", "Slider input:", 1, 100, 30)
    ),
    mainPanel(
```

```
      h1(paste0("Theme: darkly")),
      h2("Header 2"),
      p("Some text")
    )
  )
)
```

Obrázok 17: Vizualizácia niektorých bootswatch tém, zdroj: [42]

Vlastnú tému vo funkcii `bs_theme` si môžeme vytvoriť pomocou argumetov `bg` pre farbu pozadia, `fg` pre farbu elementov na stránke a `base_font` pre nastavenie typu písma. Viacero argumentov je prehľadne opísaných v dokumentácii. [42]

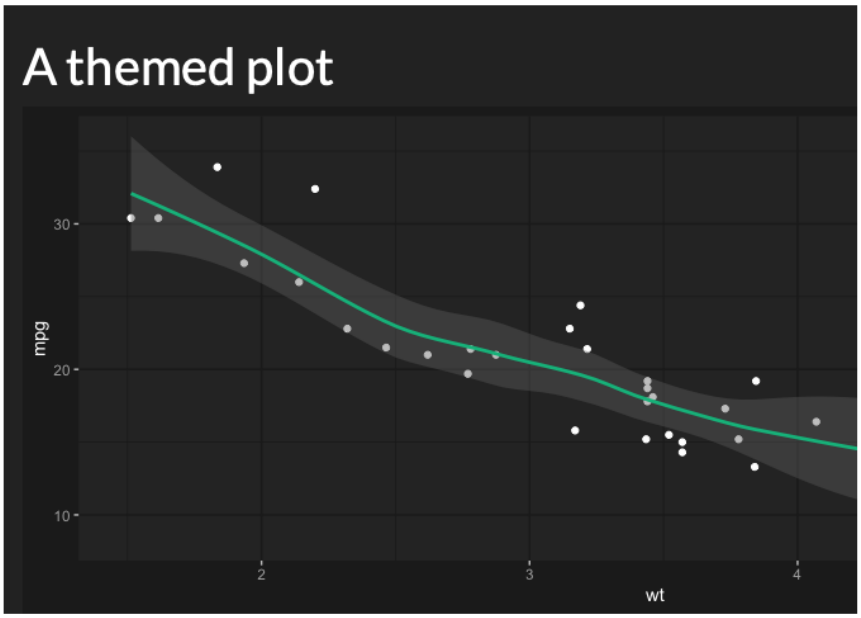
```
custom_theme <- bslib::bs_theme(
  bg = "#FFFFFF",
  fg = "green",
  base_font = "Maven Pro"
)
ui <- fluidPage(
  theme = custom_theme,
  sidebarLayout(
    sidebarPanel(
      textInput("txt", "Text input:", "text here"),
      sliderInput("slider", "Slider input:", 1, 100, 30)
    ),
    mainPanel(
      h1(paste0("Theme: darkly")),
      h2("Header 2"),
      p("Some text")
    )
  )
)
```

Farby k pozadiam vieme napísať v hexadecimálnom tvare, ale aj vo forme stringu. [42]

Okrem nastavenia tém si môžeme prispôsobiť taktiež témy pre vykresľovanie grafov, tak aby sa zhodovali s našou témou pomocou balíčka **thematic**, ktorý automaticky nastaví všetky grafy na nami vytvorenú alebo vybranú preddefinovanú tému v argumente `theme`. Stačí zavolať v serverovej časti funkciu `thematic_shiny()`. [42]

```
library(ggplot2)
library(shiny)
library(bslib)
library(thematic)

ui <- fluidPage(
  theme = bslib::bs_theme(bootswatch = "darkly"),
  titlePanel("a themed plot"),
  plotOutput("plot"),
)
server <- function(input, output, session) {
  thematic::thematic_shiny()
  output$plot <- renderPlot({
    ggplot(mtcars, aes(wt, mpg)) +
      geom_point() +
      geom_smooth()
  }, res = 96)
}
shinyApp(ui, server)
```



Obrázok 18: Vizualizácia grafu automatickým nastavením rovnakej témy, zdroj: [42]

Ak by nám z nejakého dôvodu nefungoval balíček **bslib**, alternatívne môžeme použiť balíček **shinythemes**, ktorý si doinštalujeme a pripojíme do projektu. Následne vo funkcii `fluidpage()`, použijeme argument `theme` a do neho vložíme príkaz `shinytheme("cerulean")`. [33]

```
library(shiny)
library(shinythemes)
ui <- fluidPage(theme = shinytheme("cerulean"),
  sidebarLayout(
    sidebarPanel(
      textInput("txt", "Text input:", "text here"),
      sliderInput("slider", "Slider input:", 1, 100, 30)
    ),
    mainPanel(
      h1(paste0("Theme: cerulean")),
      h2("Header 2"),
      p("Some text")
    )
  )
)
server <- function(input, output) {
}
shinyApp(ui, server)
```

Jednotlivé témy môžu byť `cerulean`, `cosmo`, `cyborg`, `darkly`, `flatly`, `journal`, `lumen`, `paper`, `readable`, `sandstone`, `simplex`, `slate`, `spacelab`, `superhero`, `united` a `yeti`.

Ak by sme chceli použiť nami vytvorenú tému, alebo stiahnutú vo forme `.css` súboru musíme daný súbor pripojiť do priečinka s projektom a následne ho priradiť takto `theme = "mytheme.css"`. [33]

3.7 Shiny dashboard

Samotný dashboard je rozšírením Shiny aplikácie, všetky 3 časti majú rovnaké, ale v UI časti používame trochu iné funkcie. Dashboard sa nachádza v balíčku **shinydashboard**, nainštalujeme si ho cez `install.packages("shinydashboard")` pripojíme do aplikácie pomocou príkazu `library(shinydashboard)`. Narozdiel od Shiny aplikácie nahradíme

funkciu `fluidPage()` funkciou `dashboardPage()`. `dashboardPage()` sa skladá z hlavičky reprezentovanej funciou `dashboardHeader()`, bočného panelu reprezentovaného funkciou `dashboardSidebar()` a tela, ktorý reprezentuje funkcia `dashboardBody()`. Samotná štruktúra dashboardu je uvedená nižšie. [32],[35]

```
library(shiny)
library(shinydashboard)
ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)
server <- function(input, output) {
}
shinyApp(ui, server)
```

3.7.1 Hlavička

Názov hlavičky si nastavíme jednoducho a to pomocou argumentu `title`.

```
dashboardHeader(title = "My Dashboard")
```

Obrázok 19: Vizualizácia názvu hlavičky, zdroj: [32]

Funkcia, ktorá nám umožnuje vytvoriť rozbaľovacie alebo drop-down menu sa nazýva `dropdownMenu()`. Menu sa rozdeľuje na tri typy, a to na menu správ, notifikácií a úloh. Pre rôzne typy menu nadobúda stavy v argumente `type` a to `messages`, `notifications` a `tasks`. Pri úlohách nadobúda ešte argument `badgeStatus`, ktorý môže nadobúdať hodnoty `success`, `warning`, `primary`, `info` a `danger`. Pre správu máme funkciu `messageItem()` s argumentami `from`, `message`, `icon` a `time`, čiže názov správy a od koho je, ikonu a čas, ktorý je vo forme stringu.

Na podobnom princípe funguje aj notifikačné menu, kde ale používame funkciu `notificationItem()`. Taktiež obsahuje argument `text` s notifikačným textom, `icon` pre ikonu a `status` pre určenie stavu daného tasku.

Obdobný princíp fungovania má taktiež menu úloh, kde funkcia `taskItem()` disponuje argumentami `value`, `color` a textový názov úlohy uvedený v úvodzovkách.

V kóde na obrázku 20 sme napriamo napísali hodnoty jednotlivých argumentov v rozbaľovacích menu. Dynamicky by sa nám mohli meniť tak, že by sme do zoznamu `.list` ukladali hodnoty, ktoré sa následne v serverovej funkcii `function(input, output){}` cez funkciu `renderMenu()` aktualizujú a pomocou príkazu vloženého do `dashboardHeader()` v tvare `dashboardHeader(dropdownMenuOutput("messageMenu"))` by sa následne vypísali. Samotný príkaz nie je na obrázku uvedený. [32]

```
library(shiny)
library(shinydashboard)
ui <- dashboardPage(
  dashboardHeader(title = "My Dashboard",
                  dropdownMenu(type = "messages",
                               messageItem(
                                 from = "Sales Dept",
                                 message = "Sales are steady this month."),
                               messageItem(
                                 from = "Support",
                                 message = "The new server is ready.",
                                 icon = icon("life-ring"),
                                 time = "2022-12-01")),
                  dropdownMenu(type = "notifications",
                               notificationItem(
                                  text = "5 new users today",
                                  icon("users")),
                               notificationItem(
                                 text = "Server load at 86%",
                                 icon = icon("exclamation-triangle"),
                                 status = "warning")),
                  dropdownMenu(type = "tasks", badgeStatus = "success",
                               taskItem(value = 90, color = "green",
                                        "Documentation"),
                               taskItem(value = 75, color = "yellow",
                                        "Server deployment")
                                      )),
  dashboardSidebar(),
  dashboardBody()
)
server <- function(input, output) {
  output$messageMenu <- renderMenu({
    msgs <- apply(messageData, 1, function(row) {
      messageItem(from = row[["from"]], message = row[["message"]])
    })
    dropdownMenu(type = "messages", .list = msgs)
  })
}
shinyApp(ui = ui, server = server)
```

Obrázok 20: Kód znázorňujúci tvorbu rozbaľovacích menu, zdroj: [32]

Obrázok 21: Vizualizácia rozbaľovacích menu, zdroj: [32]

Skrytie hlavičky umožňuje v `dashboadHeader()` argument `disable` nastavený na hodnotu `TRUE`.

3.7.2 Navigačný panel

Navigačný panel, v ktorom sa budeme vedieť preklikávať na rôzne stránky dashboardu vieme vytvoriť pomocou funkcie `sidebarMenu()`, ktorá do seba vnára funkcie `menuItem()` a `tabItem()`. Tu si však musíme dávať pozor nato, že argument `tabName` sa musí zhodovať vo funkciách `menuItem()` a `tabItem()`. Argumenty `badgeLabel` a `badgeColor` sa môžu použiť na štítok jednotlivých položiek panela ako názov a farba pozadia. [32]

```
library(shiny)
library(shibydashboard)
sidebar <- dashboardSidebar(
  sidebarMenu(
    menuItem("Dashboard", tabName = "dashboard", icon = icon("dashboard")),
    menuItem("Widgets", icon = icon("th"), tabName = "widgets",
             badgeLabel = "new", badgeColor = "green")
  )
)
body <- dashboardBody(
  tabItems(
    tabItem(tabName = "dashboard",
            h2("Dashboard tab content")
    ),

    tabItem(tabName = "widgets",
            h2("Widgets tab content")
    )
  )
)

ui <- dashboardPage(
  dashboardHeader(title = "Simple tabs"),
  sidebar,
  body
)
server <- function(input, output) {
}

shinyApp(ui, server)
```

Obrázok 22: Kód znázorňujúci tvorbu navigačného panela, zdroj: [32]

Obrázok 23: Vizualizácia navigačného panela, zdroj: [32]

Dynamicky je možné generovať navigačný panel pomocou funkcií `renderMenu()` a `sidebarMenuOutput()`. Ukážku kódu dynamického generovania navigačného panelu môžeme vidieť na obrázku 24.

```r
library(shiny)
library(shinydashboard)

ui <- dashboardPage(
  dashboardHeader(title = "Dynamic sidebar"),
  dashboardSidebar(
    sidebarMenuOutput("menu")
  ),
  dashboardBody()
)

server <- function(input, output) {
  output$menu <- renderMenu({
    sidebarMenu(
      menuItem("Menu item", icon = icon("calendar"))
    )
  })
}

shinyApp(ui, server)
```

Obrázok 24: Kód znázorňujúci tvorbu dynamického navigačného panela, zdroj: [32]

V navigačnom paneli môžeme mať okrem klasických vstupov uvedených v kapitole 3.6.1 aj jeden špecifický vstup, v ktorom môžeme vyhľadávať na stránke. Funkcia, ktorá to zabezpečuje sa nazýva `sidebarSearchForm()`. Jej použitie je uvedené nižšie. [32]

```r
sidebarSearchForm(textId = "searchText", buttonId = "searchButton",
                  label = "Search...")
```

Obrázok 25: Vizualizácia `sidebarSearchForm()` používateľského vstupu, zdroj: [32]

Ak by sme chceli skryť navigačný panel môžeme to urobiť obdobne ako pri hlavičke, ale s tým rozdielom, že funkcia sa bude volať `dashboardSidebar()`, argument a aj jeho hodnota vo vnútri ostane rovnaká ako pri hlavičke. [32]

3.7.3 Telo

V tele dashboardu sa môže vyskytovať všeličo z klasickej Shiny aplikácie. Ideálne je použiť elementy, ktoré budú štruktúrované, na ich vytváranie nám poslúži funkcia `box()`. V jej vnútri sa môže nachádzať hocijaký obsah klasickej Shiny aplikácie. V klasickom dashboarde ju používame vo vnútri funkcie `fluidRow()`. [32]

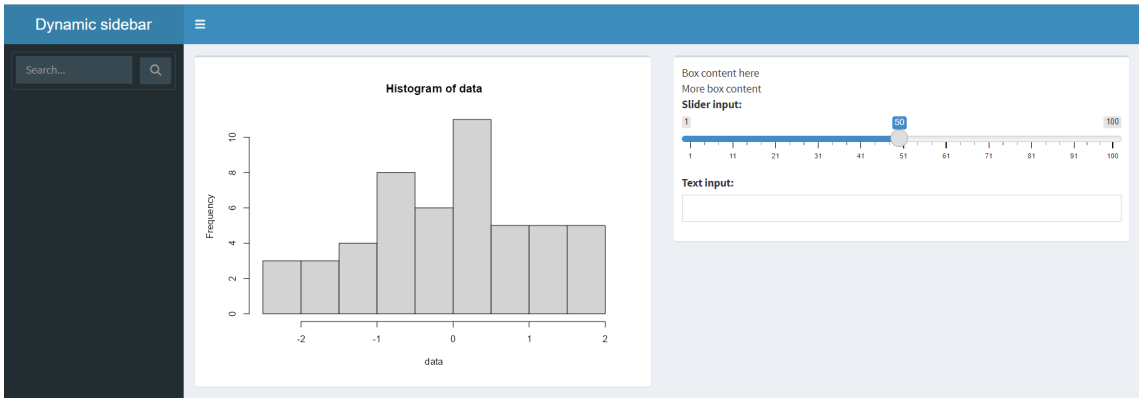
```r
library(shiny)
library(shinydashboard)
ui <- dashboardPage(
  dashboardHeader(title = "Dynamic sidebar"),

  dashboardSidebar(
    sidebarSearchForm(textId = "searchText", buttonId = "searchButton",
                      label = "Search...")),
  dashboardBody(
    fluidRow(
      box(plotOutput("plot1")),
      box(
        "Box content here", br(), "More box content",
        sliderInput("slider", "Slider input:", 1, 100, 50),
        textInput("text", "Text input:")
      )
    )
  )
)
server <- function(input, output) {
  set.seed(122)
  histdata <- rnorm(500)

  output$plot1 <- renderPlot({
    data <- histdata[seq_len(input$slider)]
    hist(data)
  })
}

shinyApp(ui, server)
```

Obrázok 26: Kód znázorňujúci tvorbu `box()` elementov, zdroj: [32]



Obrázok 27: Vizualizácia `box()` elementov, zdroj: [32]

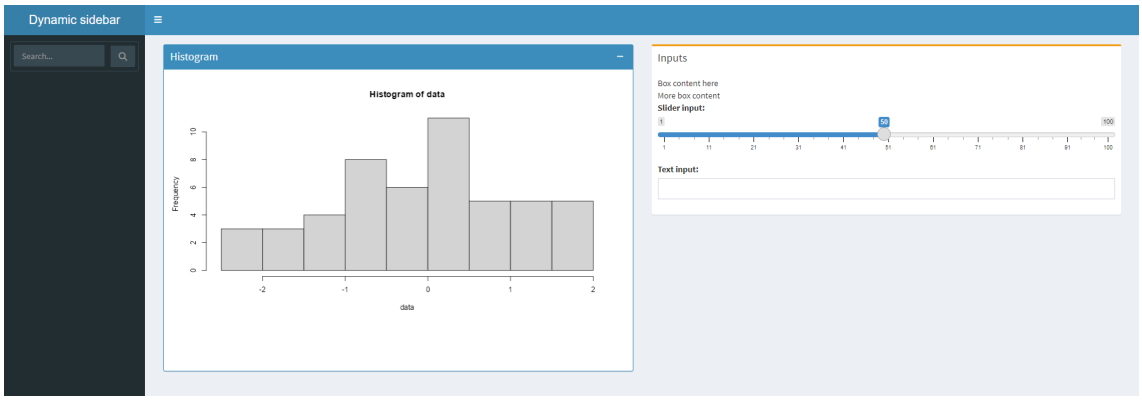
Elementy vytvorené funkciou `box()` môžu tiež nadobúdať argumenty `title` a `status`. Prvý z nich hovorí o názve a druhý o stave nad názvom. Jednotlivé stavy môžu byť `primary`, `success`, `info`, `warning` a `danger`.

K ďalším používaným argumentom patria `solidHeader` a `collapsible`. Ak argument `solidHeader` nadobúda hodnotu `TRUE`, tak hrúbka podfarbenej hlavičky je väčšia, ak nadobúda `FALSE`, tak opačne. Ak `collapsible` má hodnotu `TRUE`, tak element môže byť minimalizovaný znakom vpravo hore. [32]

```r
library(shiny)
library(shinydashboard)
ui <- dashboardPage(
  dashboardHeader(title = "Dynamic sidebar"),
  dashboardSidebar(
    sidebarSearchForm(textId = "searchText", buttonId = "searchButton",
                      label = "Search...")),
  dashboardBody(
    box(plotOutput("plot1"),
      title = "Histogram", status = "primary", solidHeader = TRUE,
      collapsible = TRUE,
      plotOutput("plot3", height = 70)
    ),
    box(
      title = "Inputs", status = "warning", solidHeader = FALSE,
      "Box content here", br(), "More box content",
      sliderInput("slider", "Slider input:", 1, 100, 50),
      textInput("text", "Text input:")
    )))
server <- function(input, output) {
  set.seed(122)
  histdata <- rnorm(500)

  output$plot1 <- renderPlot({
    data <- histdata[seq_len(input$slider)]
    hist(data)
  })}
shinyApp(ui, server)
```

Obrázok 28: Kód znázorňujúci použitie argumentov `solidHeader` a `collapsible`, zdroj: [32]



Obrázok 29: Vizualizácia použitia argumentov `solidHeader` a `collapsible`, zdroj: [32]

Ak by sme chceli mať jednofarebné pozadie v elemente, môžeme na to použiť argument `background`. Dostupné farby sú: `red`, `yellow`, `aqua`, `blue`, `light-blue`, `green`, `navy`, `teal`, `olive`, `lime`, `orange`, `fuchsia`, `purple`, `maroon` a `black`. [32]

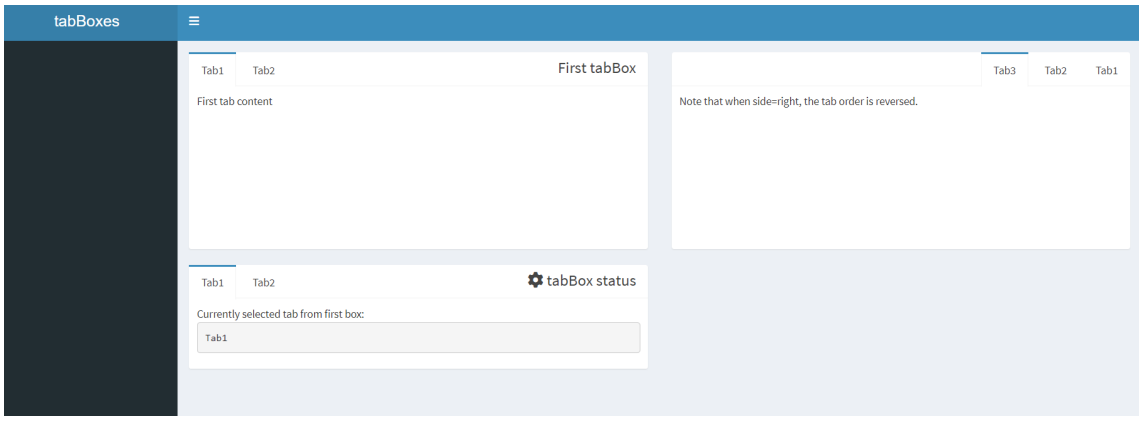
Ak by sme chceli zobraziť v jednotlivých elementoch rôzne grafy, vizualizácie, texty alebo hocičo iné, môžeme tak spraviť pomocou funkcie `tabBox()`, ktorá je podobná funkcii `tabsetPanel()` z balíčka **`shiny`**. Fungujú na rovnakom princípe, má ako vstup jednu

alebo viacero funkcií `tabPanel()`, v ktorej si môžeme vybrať, ktorú záložku vyberieme s daným obsahom. V serverovej časti následne použijeme `input$tabset1`.

Už spomínaná funkcia `tabBox()` má rovnaké argumenty `height`, `width` a `title` ako funkcia `box()`. Pomocou argumentu `side` si môžeme zvoliť, na ktorej strane elementu budú zobrazené záložky.

```
library(shiny)
library(shinydashboard)
ui <-  dashboardPage(
  dashboardHeader(title = "tabBoxes"),
  dashboardSidebar(),
  dashboardBody(
    fluidRow(
      tabBox(
        title = "First tabBox",
        id = "tabset1", height = "250px",
        tabPanel("Tab1", "First tab content"),
        tabPanel("Tab2", "Tab content 2")
      ),
      tabBox(
        side = "right", height = "250px",
        selected = "Tab3",
        tabPanel("Tab1", "Tab content 1"),
        tabPanel("Tab2", "Tab content 2"),
        tabPanel("Tab3", "Note that when side=right, the tab order is reversed.")
      )
    ),
    fluidRow(
      tabBox(
        # Title can include an icon
        title = tagList(shiny::icon("gear"), "tabBox status"),
        tabPanel("Tab1",
                 "Currently selected tab from first box:",
                 verbatimTextOutput("tabset1Selected")
        ),
        tabPanel("Tab2", "Tab content 2")
      )
    )
  )
)
server = function(input, output) {
  output$tabset1Selected <- renderText({
    input$tabset1
  })
}
shinyApp(ui, server)
```

Obrázok 30: Kód znázorňujúci použitie funkcie `tabBox()`, zdroj: [32]



Obrázok 31: Vizualizácia elementov so záložkami, zdroj: [32]

Ďalším typom elementu sú infoBoxy, ktoré znázorňujú číselné hodnoty, text a ikonu. Umožňuje to funkcia `infoBox()`. Ak chceme dať infoBoxy do prvého riadka nastavíme argument `fill` na hodnotu `FALSE`, pri druhom riadku sa hodnota zmení na `TRUE`.

Podobným typom elementov ako infoBoxy sú aj valueBoxy, ktoré sme si spomínali v kapitole 3.4.3. [32]

Rozloženie

Rozloženie je podobné ako pri klasickej Shiny aplikácii. Máme dva typy rozložení a to riadkové a stĺpcové. Oba typy rozloženia píšeme do funkcie `dashboardBody()`.

Pri riadkovom rozložení použijeme známu funkciu `fluidRow()`, do ktorej vložíme jednu alebo viacero funkcií `box()`, ak napríklad jeho argument bude mať `width = 6`, tak daný element bude zaberať polovicu šírky. Horné elementy bývajú zvyčajne zarovnané, narozdiel od spodných. Vo veľkej miere to závisí od obsahu a veľkosti daných elementov. [32]

```
box(title = "Box title", height = 300, "Box content"),
```

Pri stĺpcovom rozložení používame vo funkcii `fluidRow()` funkciu `column()`.

Okrem riadkového a stĺpcového rozloženia existuje aj zmiešané rozloženie, kde môžeme kombinovať funkcie `fluidRow()`, `box()` a `column()`. [32]

3.7.4 Témy

Rovnako ako pri klasickej Shiny aplikácii môžeme používať rôzne farebné schémy dashboardov a to pomocou príkazu uvedeného nižšie.

```
dashboardPage(skin = "blue")
```

Jednotlivé témy majú nasledovné farby `blue`, `black`, `purple`, `green`, `red` a `yellow`.

Vlastnú tému si môžeme vytvoriť pomocou balíčka **`fresh`**, ktorý si treba nainštalovať a pripojiť do projektu. Následne si pomocou jednotlivých funkcií `create_theme()`, `adminlte_sidebar()` a `adminlte_global()` nastavíme nielen farbu, ale aj šírku navigačného panela. Farby môžeme uvádzať v hexadecimálnom tvare ale aj v tvare stringu. Nami vytvorenú tému následne použijeme vo funkcii `use_theme()`. [39]

```
library(shiny)
library(shinydashboard)
library(fresh)
mytheme <- create_theme(
  adminlte_color(
    light_blue = "#434C5E"
  ),
  adminlte_sidebar(
    width = "200px",
    dark_bg = "#D8DEE9",
    dark_hover_bg = "#81A1C1",
    dark_color = "#2E3440"
  ),
  adminlte_global(
    content_bg = "#FFF",
    box_bg = "#D8DEE9",
    info_box_bg = "#D8DEE9"
  )
)
ui <- dashboardPage(
  dashboardHeader(title = "My dashboard"),
  dashboardSidebar(),
   dashboardBody(use_theme(mytheme))
  )
server <- function(input,output){}
shinyApp(ui, server)
```

Obrázok 32: Vizualizácia vlastného nastavenia témy pomocou balíčka **fresh**, zdroj: [39]

Balíček **dashboardthemes** disponuje taktiež niekoľkými preddefinovanými témami, konkrétne sa jedná o `theme_blue_gradient`, `theme_flat_red`, `theme_grey_dark`, `theme_grey_light`, `theme_onenote`, `theme_poor_mans_flatly` a `theme_purple_gradient`. [21]

3.8 Prepojenie R Markdown a R Shiny

Interaktívne HTML dokumenty môžeme vytvárať pridaním balíčka **shiny** do **R Markdown** kódu. Môžeme tak urobiť v dvoch krokoch:

- Pridaním parametra `runtime: shiny` do YAML hlavičky.
- Pridaním Shiny používateľských vstupov do R code chunks.

Pridanie parametra do YAML hlavičky

```
---
runtime: shiny
output: html_document
---
```

Pri uložení kódu s vyššie uvedenou YAML hlavičkou sa ikona `Knit`, zmení na `Run Document`. Samotná zmena ikony znamená, že RStudio už nebude kompilovať statický HTML dokument, ale dynamický dokument vo forme Shiny aplikácie.

Pridanie Shiny používateľských vstupov

Do dokumentu po pripravení YAML hlavičky následne pridáme používateľské vstupy do R code chunks. Ich kompletný zoznam aj s argumentami je uvedený v kapitole 3.6.1. Výstupy je možné vykonať pomocou `render` funkcií `renderImage()`, `renderPlot()`, `renderPrint()`, `renderTable()`, `renderText()` a `renderUI()`.

````
---
runtime: shiny
output: html_document
---

```{r echo = FALSE}

selectInput("n_breaks", label = "Number of bins:",
              choices = c(10, 20, 35, 50), selected = 20)
sliderInput("bw_adjust", label = "Bandwidth adjustment:",
              min = 0.2, max = 2, value = 1, step = 0.2)


```

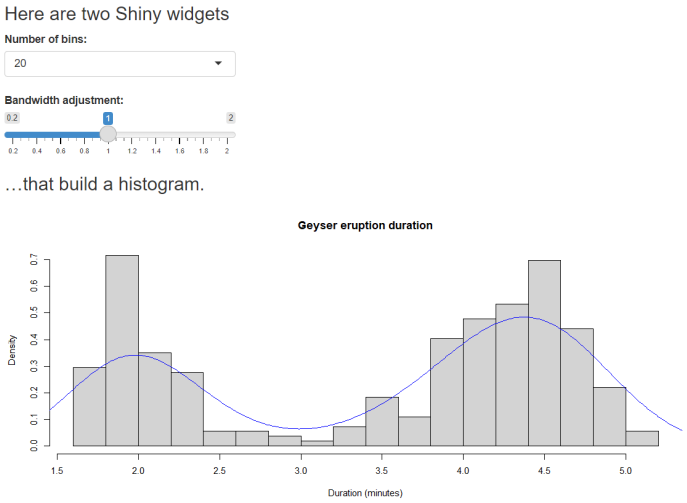
```{r echo = FALSE}

renderPlot({
  hist(faithful$eruptions, probability = TRUE,
       breaks = as.numeric(input$n_breaks),
       xlab = "Duration (minutes)",
       main = "Geyser eruption duration")

  dens <- density(faithful$eruptions, adjust = input$bw_adjust)
  lines(dens, col = "blue")
})

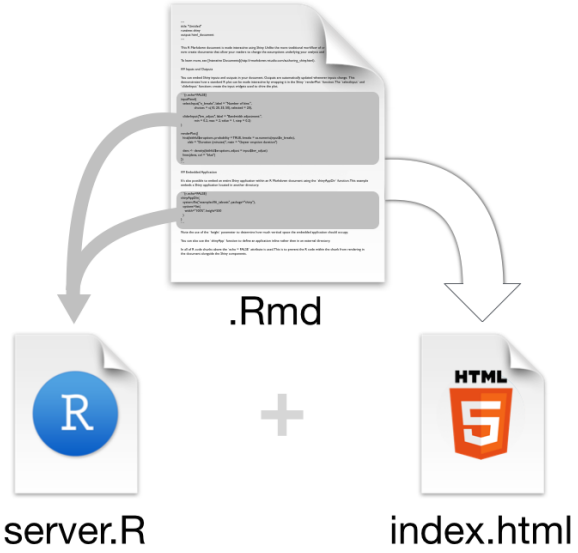
```
````

Vyššie uvedený kód obsahuje v prvom R chunku dva používateľské vstupy s názvom `selectInput()` a `sliderInput()`. V druhom R chunku obsahuje už funkciu `renderPlot()`, ktorá na základe zmeny používateľských vstupov vykreslí histogram. Na obrázku 33 môžeme následne vidieť výstup interaktívneho dokumentu, ktorý obsahuje histogram zobrazujúci distribúciu dát o dĺžke erupcií gejzíru a graf hustoty pravdepodobnosti nám ukazuje, ako sú dáta pravdepodobne rozdelené. [9]



Obrázok 33: Vizualizácia výstupu interaktívneho dokumentu, zdroj: [9]

Po kliknutí na `Run Document` balíček **`rmarkdown`** extrahuje kód z code chunks do pseudo súboru `server.R`. Následne **R Markdown** použije html výstup z `.md` súboru ako `index.html`, do ktorého umiestni reaktívne prvky. Vďaka tomu môžu výstupy v ľubovoľnom code chunku používať vstupy a reaktívne výrazy z ostatných code chunks. Samotný frontend (UI) `ui.R` už netreba písať, keďže samotné rozloženie poskytuje **R Markdown**.



Obrázok 34: Štruktúra interaktívneho dokumentu, zdroj: [9]

4 Výsledky práce

V tejto kapitole si ukážeme použitie balíčkov **rmarkdown** a **shiny** na praktických ukážkach. Taktiež si ukážeme možnosti nasadenia vytvorenej aplikácie na web.

4.1 Praktická ukážka č. 1

V prvej praktickej ukážke sme sa zamerali na tvorbu Shiny dashboard webových aplikácii pre analýzu akcií na trhoch, obe aplikácie sa líšia statickým a dynamickým generovaním elementov, ktoré si porovnáme.

Základným balíčkom, z ktorého sme čerpali dáta pre aplikáciu bol **tidyquant**. Ďalšími nemenej dôležitými balíčkami bez ktorých by aplikácia nefungovala boli **shinydashboard** a **shiny**. Medzi doplnkové balíčky, ktoré sme použili patria **fresh**, **ggplot2**, **plotly** a **DT**.

Elementárna aplikácia

Pred frontendovou a backendovou časťou aplikácie sme si zadefinovali tému pre vzhľad dashboardu pomocou balíčka **fresh**, z ktorého sme využili funkcie `create_theme()`, `adminlte_color()`, `adminlte_sidebar()` a `adminlte_global()`.

Následne sme prešli k tvorbe frontedovej (UI) časti, kde sme si vytvorili cez funkcie `dashboardPage()`, `dashboardHeader()`, `dashboardSidebar()` a `dashboardBody()` rozloženie dashboardu. V `dashboardPage()` sme použili funkciu `dashboardHeader()`, kde sme uviedli názov a definovali rozbaľovacie menu cez funkciu `dropdownMenu()`. Vo funkcii `dashboardSidebar()` sme si vytvorili navigačný panel cez `sidebarMenu()` a v ňom položky cez funkciu `menuItem()`. Následne sme cez funkcie `selectizeInput()` a `dateInput()` nastavili používateľské vstupy pre typ akcie, ktorú chceme zvoliť a dátumové rozmedzie, v ktorom chceme zobraziť graf danej akcie, s tým, že používateľ môže zvoliť maximálne 4 akcie. Následne sme v `dashboardBody()` nastavili nami nadefinovanú tému cez funkciu `use_theme(mytheme)` a následne sme použili funkciu `tabItems()`, navigačného panela `Stocks` po zvolení akcií a dátumov vykreslí 4 elementy cez funkciu `box()` na obra-

zovku vo forme grafov.

Serverovú funkciu aplikácie môžeme rozdeliť na 2 funkčné časti. V prvej časti, pomocou shiny funkcie `reactive()` načítame v cykle dáta o používateľom zvolených akciách. Zdrojový kód zabalený v tejto funkcii sa vykoná, ak dôjde ku zmene akejkoľvek reaktívnej premennej, pomocou používateľského vstupu. Jedná sa o premenné `input$stocks`, `input$date_start` a `input$date_end`. Tieto reaktívne premenné posúvame do funkcie `tq_get()` pomocou ktorej čerpáme dáta o akciách a ktorej výstup je výstup pre funkciu `reactive()`.

Druhá časť serverovej funkcie už vykresľuje samotné grafy. Samotné UI elementy pre pre tieto grafy sú definované na frontende (UI) vo funkcii `dashboardPage()` a sú stále viditeľné, aj keď nie sú vykreslené žiadne dáta. Pre väčšiu prehľadnosť a skrátenie zdrojového kódu sú grafy vykresľované v cykle, mierne iným spôsobom ako klasickým, reaktívnym volaním funkcie `renderPlotly()`. Použijeme preto funkciu `observe()`, ktorá zabezpečí vykonanie zabaleného zdrojového kódu ak dôjde ku zmene reaktívnej premennej, v našom prípade výstupu z vyššie uvedenej `reactive()` funkcie - premennej `data_stocks`. V samotnom cykle potom samotný `renderPlotly()` zabalíme do ďalšej funkcie, `local()`. Tá nám zabalený zdrojový kód vykoná v lokálnom prostredí s dočasnými premennými a zabalené reaktívne funkcie v nej voláme v sekvencii, v akej sú v zdrojovom kóde. Pre jej správne fungovanie ale musíme uložiť lokálnu hodnotu kontrolnej premennej (`i`) cyklu. Než vstúpime do `renderPlotly()` funkcie, musíme si ešte prichystať ID (názov) UI elementu, do ktorého chceme konkrétny graf vykresliť. Keďže máme naše `plotlyOutput()` pomenované ako `plot_stocks1` až `plot_stocks4`, stačí nám spojiť string `plot_stocks` s hodnotou uloženou v (lokálnej kópii) kontrolnej premennej cyklu, pomocou volania `plot_name<- paste0("plot_stocks",j)`. Do samotného, j-teho UI elementu pre graf potom vykresľujeme pomocou volania (`output[[plot_name]] <- renderPlotly()`. V tejto funkcii máme potom už len zabalené nastavenie vstupných premenných pre `ggplot()`, na Y-os grafu vykresľujeme fixne hodnotu (`adjusted`) ceny akcie. Funkcia `tq_get()` nám vracia viacero stĺpcov, s minimálnou a maximálnou dennou cenou atď. Nakoniec vykreslíme graf volaním funkcie `ggplotly()` z balíčka **`plotly`**.

Veľkou nevýhodou tohto prístupu ku vykresľovaniu grafov je, že ak používateľ nezvolí 4 akcie tak zvyšné elementy ostanú prázdne. Ak už bola akcia vykreslená a používateľ zruší výber danej akcie, graf nezmizne a prekreslí sa, až keď používateľ znovu vyberie inú akciu. Jedným z možných riešení je použitie funkcií `insertUI()` a `removeUI()`, no ešte lepším riešením je postavenie plne dynamického UI.

```
library(shiny)
library(shinydashboard)
library(fresh)
library(ggplot2)
library(plotly)
library(tidyquant)

mytheme <- create_theme(
adminlte_color(
light_blue = "#2F2E2F"
),
adminlte_sidebar(
width = "200px",
dark_bg = "#2F2E2F",
dark_hover_bg = "#2F2E2F",
dark_color = "#2F2E2F"
),
adminlte_global(
content_bg = "#2E3440",
box_bg = "#ffffff",
info_box_bg = "#ffffff"
)
)

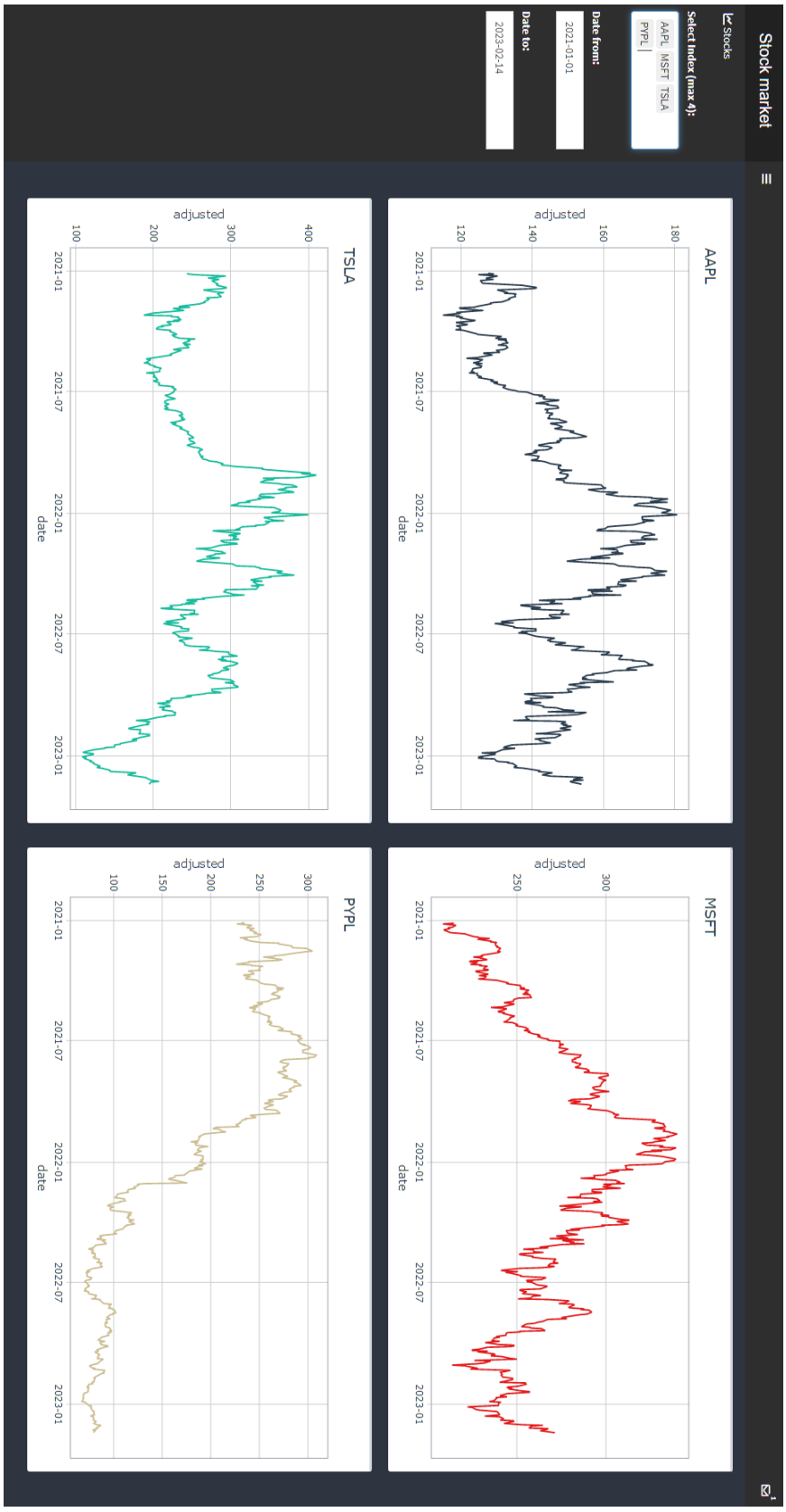
ui <- function(req) {
dashboardPage(
dashboardHeader(title = "Stock market",
dropdownMenu(
notificationItem(
text = "Content Settings",
icon = icon("cog",lib="glyphicon"),
status = "success"))),
dashboardSidebar(
sidebarMenu(
menuItem("Stocks", tabName = "stocks", icon = icon("chart-line")),
selectizeInput(inputId = "stocks", label = "Select Index (max 4):",
 choices = c("AAPL", "MSFT",
"GOOG", "AMZN", "TSLA", "NVDA", "META",
"BABA", "CRM", "AMD", "INTC", "PYPL",
"ATVI", "EA", "JNJ", "PFE", "MRNA", "BNTX",
```

```
"KO", "DTE", "ORAN",
"TEF"), multiple = T, selected = "AAPL",options = list(maxItems = 4)),
dateInput(inputId = "date_start", label = "Date from: ", value = "2021-01-01"),
dateInput(inputId = "date_end", label = "Date to: ", value = NULL)
)
),

dashboardBody(use_theme(mytheme),
tabItems(
tabItem(tabName = "stocks",
column(width = 12,
 box(plotlyOutput("plot_stocks1",width = "100%")),
 box(plotlyOutput("plot_stocks2",width = "100%")),
 box(plotlyOutput("plot_stocks3",width = "100%")),
 box(plotlyOutput("plot_stocks4",width = "100%")),
)
)))
)}

server <- function(input, output, session) {
data_stocks <- reactive({
req(input$stocks)
temp_data_stocks <- list()
for (i in 1:length(input$stocks))
temp_data_stocks[[i]] <- tq_get(input$stocks[i], get = "stock.prices",
from = input$date_start, to = input$date_end)
temp_data_stocks
})

observe({
data=data_stocks()
for (i in 1:length(input$stocks)){
local({
j<-i
plot_name<- paste0("plot_stocks",j)
output[[plot_name]] <- renderPlotly({
date <- data[[j]]$date
adjusted <- data[[j]]$adjusted
p <- ggplot(data[[j]] ,aes(x = date, y = adjusted)) +
geom_line(color = palette_light()[[j]]) +
scale_y_continuous() +
labs(title = isolate(input$stocks[j]),
 y = "adjusted ", x = "date") +
theme_tq()
ggplotly()
})})}})}
shinyApp(ui, server)
```



Obrázok 35: Vizualizácia obsahu položky navigačného panela `Stocks` v elementárnej aplikácii pre analýzu akcií, zdroj: [vlastné spracovanie]

Pokročilá aplikácia

Pri pokročilej aplikácii sme používali rovnaké balíčky a nami definovanú tému.

Frontendová (UI) časť je rozdelená zase do 4 funkcií rovnako ako pri elementárnej verzii tejto aplikácie. Líšia sa v tom, že funkcia `dashboardSidebar()` má až 3 položky, ktoré sme vytvorili cez funkciu `menuItem()`. Narozdiel od elementárnej verzie tejto aplikácie, je táto verzia obohatená v položke `Stocks` o výber trhového indexu, z ktorého si následne môže používateľ vybrať akcie. Takisto je používateľovi umožnené vybrať dátumové rozmedzie a typ ceny akcie. V položke `Data` nájdeme prehľadne zobrazené akcie jednotlivých trhových indexov. Doteraz ešte nespomenutou funkciou je `conditionalPanel()`, ktorá nám umožňuje zrolovať jednotlivé vstupy v navigačnom paneli, ak zvolíme inú položku. Vo funkcii `dashboardBody()` používame funkciu `uiOutput()`, v ktorej je v serverovej časti implementovaná tvorba dynamických elementov a taktiež funkcie `dataTableOutput()` a `textOutput()` pre vykreslenie interaktívnej DT tabuľky a vypísanie textu. Tieto elementy sú rozdelené do svojich vlastných položiek v navigačnom paneli, ktoré si volíme, zobrazená je vždy len 1 položka navigačného panela.

Serverová časť je od predchádzajúcej verzie rozšírená nielen o dynamické generovanie UI, ale aj o ďalšie ošetrenia používateľského vstupu, aby nedochádzalo ku chybám pri vykresľovaní obsahu.

Prvou zmenou pre používateľa je výber trhového indexu. Keďže je tento vstupný UI element duplicitný, nachádza sa v položke `Stocks` aj `Data`, no s rôznym ID (názvom), ako prvú vec robíme synchronizáciu týchto 2 vstupov. Keďže nám Shiny neumožňuje prepisovať hodnoty reakčných premenných, aj napriek tomu že sa jedná o neefektívne riešenie, vytvorili sme kód, ktorý synchronizuje tieto 2 vstupy do jednej premennej. Vytvorili sme si preto novú reakčnú premennú `set_index` a manuálne nastavili jej základnú hodnotu korešpondujúc so základným výberom trhového indexu. Následne, v dvoch `observe()` blokoch sledujeme zmenu premenných `input$set_indext1` a `input$set_indext2` pre jednotlivé položky. V oboch prípadoch uložíme zmenenú hodnotu. V prvom prípade do premennej `set_index` a pomocou funkcie `updateSelectInput()` upravíme perspektívny výber pre druhú položku, a naopak.

Po zmene zvoleného trhového indexu, pomocou funkcie `tq_get()` zabalenej v reaktívnej funkcii stiahneme príslušný zoznam akcií. Po jeho uložení do reakčnej premennej sa daný zoznam vykreslí v príslušnej položke a pomocou funkcie `updateSelectizeInput()` sa aktualizuje možnosť výberu akcií v `selectizeInput()`. Následné načítanie dát ku jednotlivým zvoleným akciám prebieha rovnako ako v prvej ukážke.

Narozdiel od prvej ukážky ošetrujeme ešte ďalší používateľský vstup, a to začiatočný a konečný dátum, pre ktorý sú dáta sťahované. Podobne ako u synchronizácie 2 vstupov, ktoré ovládajú v podstate tú istú premennú, náš spôsob zadania časového intervalu nie je najefektívnejší, keďže nepoužívame `dateRangeInput()`, ale dva `dateInput()`. Nastáva kolízia, keď môže používateľ zadať začiatočný dátum neskorší ako konečný, a naopak. Toto riešime čiastočne pomocou funkcie `updateDateInput()` zabalenej v reakčnej funkcii `observeEvent()`, a to pre oba dátumy. Tieto funkcie sledujú, či začiatočný (konečný) dátum nebol zadaný ako väčšia (menšia) hodnota ako konečný (začiatočný) dátum, respektívne. Ak takáto situácia nastane, vstup od používateľa sa invaliduje, používateľ musí zadať dátum znovu, tentokrát s ohraničujúcou podmienkou.

Pred samotným vykreslením grafu musí nastať ešte jedna akcia, a to vykreslenie UI elementu pre tento graf. Maximálny počet vykresliteľných UI elementov, ako aj zvolených akcií, je obmedzený statickou hodnotou uloženou v premennej `max`, v prípade tejto ukážky, 10. V cykle, horne ohraničenom premennou `max` a lokalizovanom prostredí, si pomocou funkcie `renderUI()` nachystáme tieto výstupné elementy. Vytvoríme im názvy $\texttt{box}_1$ až $\texttt{box}_j$, do ktorých vkladáme `plotlyOutput()` s názvom $\texttt{plot}_1$ až $\texttt{plot}_j$, ich číselný index korešponduje s indexom boxu, v ktorom sú umiestnené. Pre tieto UI elementy je znovu použitá funkcia `renderUI()`, v reakčnom kontexte, kde pomocou funkcie `lapply()` pripravíme zoznam vykresľovaných `htmlOutput()` elementov, pre počet vybraných akcií, a vytvoríme im názvy $\texttt{box}_1$ až $\texttt{box}_n$.

Vykresľovanie grafov funguje rovnako, ako pri predchádzajúcej ukážke. Jediným pridaným kódom, je validácia vykresľovaných dát. V prípade ak si používateľ zvolí akciu, ktorú služba Yahoo finance nepozná, alebo dôjde k neočakávanej chybe pri načítaní dát o akciách, funkcia `validate()` pomocou zabalenej funkcie `need()` overí, či sa na danom

indexe premennej `data` nachádzajú dáta. Ak je táto podmienka nesplnená (`NA`, popr. `NaN`), používateľovi sa vypíše vlastná chybová hláška.

```
library(shiny)
library(shinydashboard)
library(tidyquant)
library(ggplot2)
library(plotly)
library(fresh)
library(DT)

max <- 10
mytheme <- create_theme(
adminlte_color(
light_blue = "#2F2E2F"
),
adminlte_sidebar(
width = "200px",
dark_bg = "#2F2E2F",
dark_hover_bg = "#2F2E2F",
dark_color = "#FFFFFF"
),
adminlte_global(
content_bg = "#2E3440",
box_bg = "#ffffff",
info_box_bg = "#ffffff"
))

ui <- function(req) {
dashboardPage(
dashboardHeader(title = "Stock Market",
dropdownMenu(
notificationItem(
text = "Content settings",
icon = icon("cog", lib = "glyphicon"),
status = "success"))
),

dashboardSidebar(
sidebarMenu(id = "tabs",
menuItem("Stocks", tabName = "stocks", icon = icon("chart-line")),
conditionalPanel("input.tabs === 'stocks'",

 selectInput("set_indext1", label = "Select Index: ",
 c("DOW","DOWGLOBAL","SP400","SP500","SP600"), multiple = F, selected = "DOW"),
 selectizeInput("set_stocks", label = "Select Stock Symbols: ",
NULL, multiple = T, selected = "AAPL",
options = list(maxItems = max)),
```

```
 dateInput(inputId = "datum_start", label = "Date from: ",
 value = "2021-01-01"),
 dateInput(inputId = "datum_end", label = "Date to: ",
 value = Sys.Date(), max=Sys.Date()),
 selectInput(inputId = "price", label = "Select Type of Price: ",
 multiple = F,
 choices = c("open", "high", "low", "close", "adjusted"))
),

menuItem("Data", tabName = "Data", icon = icon("table")),
conditionalPanel("input.tabs === 'Data'",
 selectInput("set_indext2", label = "Select Index: ",
 c("DOW","DOWGLOBAL","SP400","SP500","SP600"), multiple = F,
 selected = "DOW")),

menuItem("About", tabName = "info", icon = icon("info"))
)),

dashboardBody(use_theme(mytheme),
tabItems(
tabItem("stocks",
uiOutput("plots")),
tabItem("Data",
column(box(dataTableOutput("DataDT"), width = "100%"), width = 12)),
tabItem("info",
column(box(textOutput("sources"), width = "100%"), width = 12))
)))
}

server <- function(input, output,session) {
set_index <- reactiveVal("DOW")

observe({
set_index(input$set_indext1)
updateSelectInput(session = getDefaultReactiveDomain(),
"set_indext2",
selected = input$set_indext1)
})
observe({
set_index(input$set_indext2)
updateSelectInput(session = getDefaultReactiveDomain(),
"set_indext1",
selected = input$set_indext2)
})

symbol_list <- reactive(
tq_index(set_index())
)
```

```
observe(
updateSelectizeInput(
session = getDefaultReactiveDomain(),
"set_stocks",
choices = symbol_list()[-1,1],
selected = symbol_list()[2,1],
options = list(),
server = FALSE
))

output$plots <- renderUI({
plot_output_list <- lapply(1:length(input$set_stocks), function(i) {
boxname <- paste0("box", i)
htmlOutput(boxname)
})
tagList(plot_output_list)
})

for (i in 1:max) {
local({
j <- i
boxname <- paste0("box",j)
output[[boxname]] <- renderUI({
plotname <- paste0("plot", j)
column(width=6,
 box(plotlyOutput(plotname,width='100%'),
 width='100%'))})
})
}

observeEvent(input$datum_start,{
req(input$datum_start)
my_datum_start<-isolate(input$datum_start)
my_datum_end<-isolate(input$datum_end)
if (my_datum_start>=my_datum_end)
updateDateInput(session,"datum_start", max = input$datum_end-1)
})
observeEvent(input$datum_end,{
req(input$datum_end)
my_datum_start<-isolate(input$datum_start)
my_datum_end<-isolate(input$datum_end)
if (my_datum_start>=my_datum_end)
updateDateInput(session,"datum_end", min = input$datum_start+1,max=Sys.Date())
})

data_stocks <- reactive({
temp_data_stocks <- list()
```

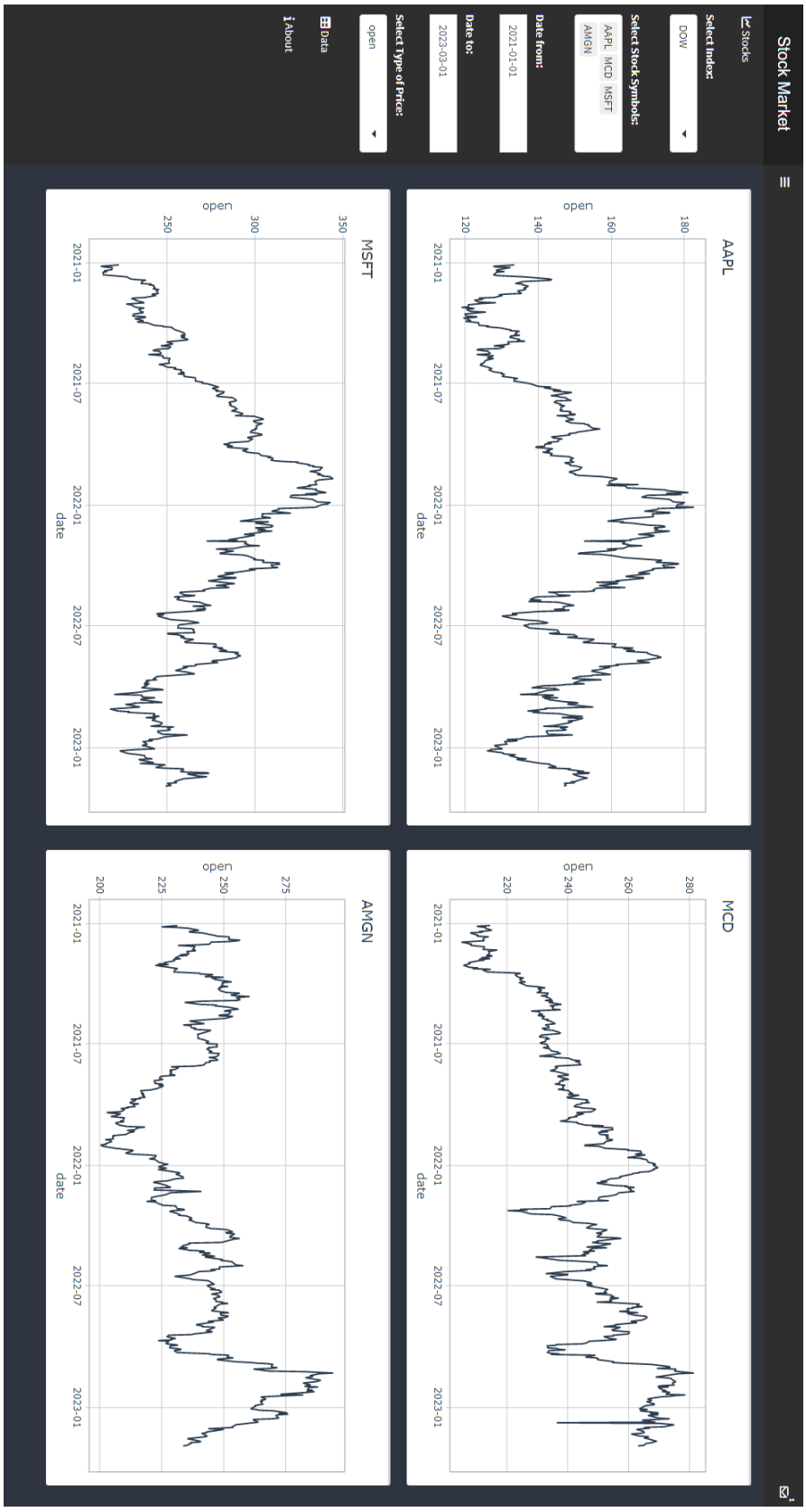
```
for (i in 1:length(input$set_stocks)){
req(input$set_stocks)
temp_data_stocks[[i]] <- tq_get(input$set_stocks[i], get = "stock.prices",
from = input$datum_start, to = input$datum_end)
}
temp_data_stocks
})

observe({
data <- data_stocks()
for (i in 1:length(isolate(input$set_stocks))){
local({
j<-i
plotname <- paste0("plot", j)
output[[plotname]] <- renderPlotly({
validate(need(data[[j]],paste0("Graph can not be plotted, maybe symbol ",
isolate(input$set_stocks[j])," is not available on Yahoo finance.")))
price <- data[[j]][[input$price]]
g <- ggplot(data[[j]], aes(x = date, y = price)) +
geom_line(color = palette_light()[[1]]) +
scale_y_continuous() +
labs(title = isolate(input$set_stocks[j]),
 y = input$price, x = "date") +
theme_tq()
ggplotly(g)
})})}
})
output$res <- renderText({})

output$DataDT <- renderDataTable(
(as.data.frame(symbol_list())), top = "bottom", options = list(
pageLength = 30, lengthMenu = c(10, 20, 50, 100)))

output$sources <- renderText(
"Shiny dashboard application for displaying stock prices.
Stocks data used from Yahoo finance, downloaded through tidyquant package.
Other used packages: ggplot2, plotly, fresh, shiny and shinydashboard.
In tabItem \"Stocks\" user can choose between stock indexes
and choose up to 10 available symbol from chosen index.
Stock prices are visualised in user choosed date range.
At tabItem \"Data\" can user choose from available stock indexes
and is presented with list od stock symbols for set index
, with aditional informations.
The aim of this application is to show
dynamically generated UI in Shiny Dashboard application."
)}

shinyApp(ui = ui, server = server)
```



Obrázok 36: Vizualizácia obsahu položky navigačného panela `Stocks` v pokročilej aplikácii pre analýzu akcií, zdroj: [vlastné spracovanie]

Obrázok 37: Vizualizácia obsahu položky navigačného panela `Data` v pokročilej aplikácii pre analýzu akcií, zdroj: [vlastné spracovanie]

Obrázok 38: Vizualizácia obsahu položky navigačného panela `About` v pokročilej aplikácii pre analýzu akcií, zdroj: [vlastné spracovanie]

4.2 Praktická ukážka č. 2

V druhej praktickej ukážke sme sa zamerali na tvorbu PDF dokumentu. Použili sme na to balíčky **rmarkdown**, **tinytex** a následne pri spracovaní a vizualizácií dát **dplyr**,**imputeTS**, **tidyr** a **ggplot2**. V YAML hlavičke sme použili parametre `title`, `author`, `date`, `output`, `df_print`, `fig_width`, `fig_height`, `fig.caption`, `latex_engine`, `number_sections` a `header_includes`. V tele dokumentu sme využili symbol `|` s tromi medzerami, ktoré nám určujú veľkosť odseku, takisto odrážky, prelink na stránku a tiež `\texttt{}`, s ktorým vieme označiť text typom písma `Courier`. Následne sme vložili obrázok, ktorého nastavenie je v YAML hlavičke, v tele sme uviedli jeho šírku, názov a cestu k nemu. Potom sme v jednotlivých R code chunks napísali kód, ktorý sa po vyexportovaní zobrazil v bunkách.

Prvá časť dokumentu s názvom **1 Informácie o dátach** poskytuje informácie o dátach a zdrojovú stránku, z ktorej sme dáta použili. V druhej časti dokumentu s názvom **2 Exploratívna analýza** sme sa snažili získať nové informácie z dát a upraviť ich tak, aby nám poskytli zmyslupné informácie. Použili sme napríklad lineárnu interpoláciu, výpočet ročných priemerov a následne sme zobrazili zmeny teplôt počas posledných 70 rokov.

```
---
title: Historic temperatures - Slovakia
author: Alena Stracenská
date: "`r format(Sys.time(), '%d. %B %Y')`"
output:
  pdf_document:
    df_print: kable
    fig_width: 7
    fig_height: 3.7
    fig_caption: true
    number_sections: true
latex_engine: xelatex
header-includes:
  - \renewcommand{\figurename}{Obrázok č.}
  - \makeatletter
  - \def\fnum@figure{\textbf{\figurename\nobreakspace\thefigure}}
  - \makeatother
---

# Informácie o dátach
|   Dáta sme vzali v surovom stave zo stránky [ecad.eu](https://www.ecad.eu/).
```

Sú v nich uvedené minimálne, maximálne, priemerné teploty a rôzne ďalšie
meteorologické údaje pre európske meteorologické stanice.
Keďže sme chceli pracovať len s minimálnou, maximálnou a priemernou teplotou
za posledných 70 rokov na Slovensku, tieto údaje nám spĺňali nasledovné
meteorologické stanice:

- Hurbanovo (skratka: HUR)
- Košice (skratka: KE)
- Poprad (skratka: PP)
- Oravská Lesná (skratka: ORL)

| Pre každú stanicu sme mali samostatný \texttt{.txt} súbor.
Keďže úprava priamo v R by bola mierne komplikovaná, rozhodli sme
sa dáta do základného datasetu spracovať v Exceli.

```{r echo=FALSE, out.width='100%', fig.cap = "Zdrojová stránka dát", echo=FALSE}
knitr::include_graphics('./ecad_edu.png')
```

| Výsledný dataset tvorí 13 stĺpcov, z ktorých jeden tvorí dátum
 a ďalších dvanásť obsahuje teploty.
V súbore sa takisto nachádzajú dni bez nameraných teplôt.
Celý dataset si najprv očistíme a následne sa v grafickej forme pozrieme,
ako sa vyvíjali teploty za posledných 70 rokov na Slovensku.
\pagebreak

Exploratívna analýza

```{r setup, include=FALSE}
options(warn=-1)
library(dplyr, warn.conflicts=FALSE)
library(imputeTS)
library(tidyr)
library(ggplot2)
```

```{r}
df <- read.csv("vsetky_udaje_sprocesovane_nofix.csv", sep = ";", encoding = "UTF-8")
columns_to_select <- c("DATE","HUR_MEAN_TMP","KE_MEAN_TMP","ORL_MEAN_TEMP",
"PP_MEAN_TEMP")
#zobrazenie 5/13 stlpcov datasetu
df[1:5, ] %>% select(all_of(columns_to_select))
```

```{r}
```

```
#nahradenie zlych hodnot (-9999) prazdnou hodnotou (Na)
df[df == "-9999"] <- "NA"
df[] <- lapply(df, as.numeric)
# uprava datumu (int) na objekt (date) a vynasobenie teplot
df$DATE <- as.Date(as.character(df$DATE),"%Y%m%d")
df[,2:13]<-df[,2:13]*0.1
#linearna interpolacia neplatnych hodnot (max 5 za sebou)
df[ ,2:13] <- na_interpolation(df[ ,2:13], maxgap = 5)
#odstranenie hodnot Na od 1.1.2021
df <- df[df$DATE<as.Date("2021-01-01"),]
df[, 2:13] <- round(df[, 2:13], digits = 2)
df <- df %>% drop_na()
sum(is.na(df))
#rocne priemery
df$Year <- format(df$DATE,format="%Y")
rocne <- aggregate(cbind(HUR_MEAN_TMP,KE_MEAN_TMP,ORL_MEAN_TEMP,PP_MEAN_TEMP,
HUR_MIN_TEMP,KE_MIN_TEMP,ORL_MIN_TEMP,PP_MIN_TEMP,
HUR_MAX_TEMP,KE_MAX_TEMP,ORL_MAX_TEMP,
PP_MAX_TEMP) ~ Year , df , mean )
hurbanovo <- subset(rocne,select=c(Year,HUR_MEAN_TMP,HUR_MIN_TEMP,
HUR_MAX_TEMP))
kosice <- subset(rocne,select=c(Year,KE_MEAN_TMP, KE_MAX_TEMP,
KE_MIN_TEMP))
poprad <- subset(rocne,select=c(Year,PP_MEAN_TEMP, PP_MAX_TEMP,
PP_MIN_TEMP))
oravska_lesna <- subset(rocne,select=c(Year,ORL_MEAN_TEMP,
ORL_MAX_TEMP, ORL_MIN_TEMP))
hurbanovo$Year <- as.numeric(hurbanovo$Year)
kosice$Year <- as.numeric(kosice$Year)
poprad$Year <- as.numeric(poprad$Year)
oravska_lesna$Year <- as.numeric(oravska_lesna$Year)

graf <- ggplot() +
geom_line(data=hurbanovo, aes(x = Year, y = HUR_MEAN_TMP, color = "Hurbanovo"))+
geom_line(data=kosice, aes(x = Year, y = KE_MEAN_TMP, color = "Košice")) +
geom_line(data=poprad, aes(x = Year, y = PP_MEAN_TEMP, color = "Poprad"))+
geom_line(data=oravska_lesna, aes(x = Year, y = ORL_MEAN_TEMP,
color = "Oravská Lesná"))+
labs(x = "Roky", y = "Stupne", title = paste0("Priemerné teploty za
roky 1951-2021"))+
scale_color_manual(name = "Meteostanice", values = c("Hurbanovo" = "red",
 "Košice" = "yellow",
 "Poprad" = "green",
 "Oravská Lesná" = "blue")) +
theme(legend.position = "bottom", plot.title = element_text(hjust = 0.5))
graf


```

Historic temperatures - Slovakia

Alena Stracenská

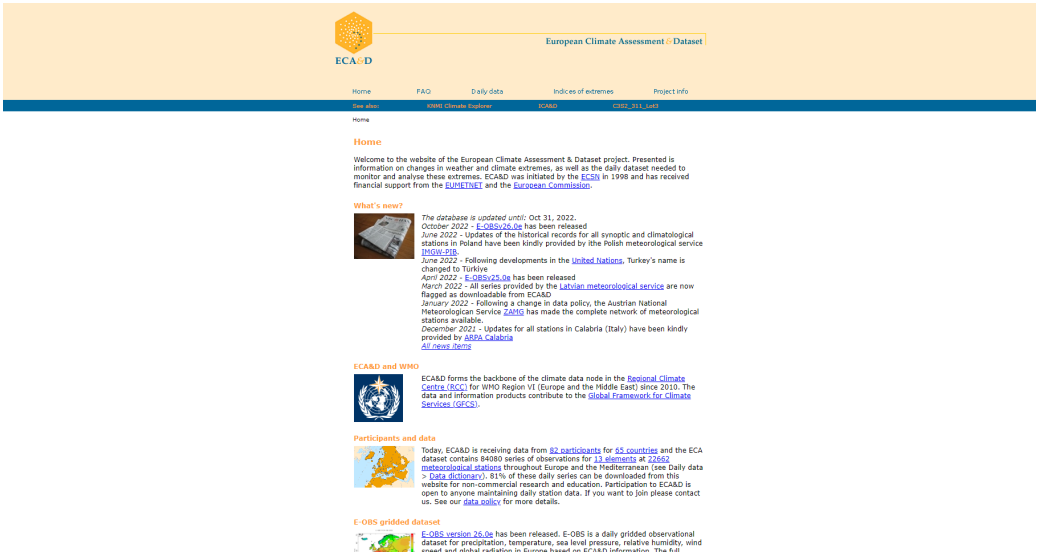
22. apríl 2023

1 Informácie o dátach

Dáta sme vzali v surovom stave zo stránky ecad.eu. Sú v nich uvedené minimálne, maximálne, priemerné teploty a rôzne ďalšie meteorologické údaje pre európske meteorologické stanice. Keďže sme chceli pracovať len s minimálnou, maximálnou a priemernou teplotou za posledných 70 rokov na Slovensku, tieto údaje nám spĺňali nasledovné meteorologické stanice:

- Hurbanovo (skratka: HUR)
- Košice (skratka: KE)
- Poprad (skratka: PP)
- Oravská Lesná (skratka: ORL)

Pre každú stanicu sme mali samostatný `.txt` súbor. Keďže úprava priamo v R by bola mierne komplikovaná, rozhodli sme sa dáta do základného datasetu spracovať v Exceli.



Obrázok č. 1: Zdrojová stránka dát

Výsledný dataset tvorí 13 stĺpcov, z ktorých jeden tvorí dátum a ďalších dvanásť obsahuje teploty. V súbore sa takisto nachádzajú dni bez nameraných teplôt. Celý dataset si najprv očistíme a následne sa v grafickej forme pozrieme, ako sa vyvíjali teploty za posledných 70 rokov na Slovensku.

2 Exploratívna analýza

```
df <- read.csv("vsetky_udaje_sprocesovane_nofix.csv", sep = ";", encoding = "UTF-8")
columns_to_select <- c("DATE","HUR_MEAN_TMP","KE_MEAN_TMP","ORL_MEAN_TEMP","PP_MEAN_TEMP")
#zobrazenie  5/13 stlpcov datasetu
df[1:5, ] %>% select(all_of(columns_to_select))
```

DATE	HUR_MEAN_TMP	KE_MEAN_TMP	ORL_MEAN_TEMP	PP_MEAN_TEMP
19510101	0	-9999	-9999	-9999
19510102	33	-9999	-9999	-9999
19510103	47	13	28	-8
19510104	23	23	-6	0
19510105	15	10	1	-20

```
#nahradenie zlych hodnot (-9999) prazdnou hodnotou (Na)
df[df == "-9999"] <- "NA"
df[] <- lapply(df, as.numeric)
# uprava datumu (int) na objekt (date) a vynasobenie teplot
df$DATE <- as.Date(as.character(df$DATE),"%Y%m%d")
df[,2:13]<-df[,2:13]*0.1
#linearna interpolacia neplatnych hodnot (max 5 za sebou)
df[ ,2:13] <- na_interpolation(df[ ,2:13], maxgap = 5)
#odstranenie hodnot Na od 1.1.2021
df <- df[df$DATE<as.Date("2021-01-01"),]
df[, 2:13] <- round(df[, 2:13], digits = 2)
df <- df %>% drop_na()
sum(is.na(df))
```

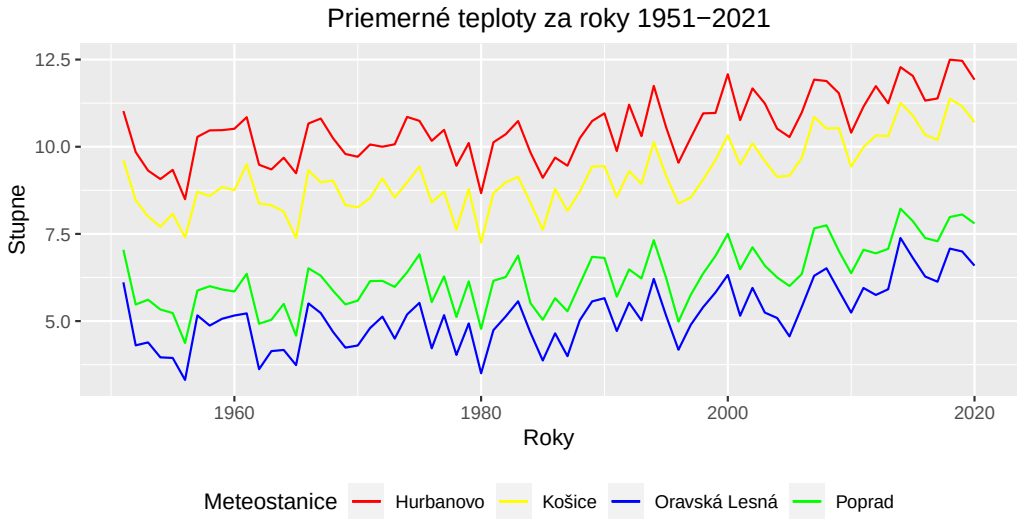
```
## [1] 0
```

```
#rocne priemery
df$Year <- format(df$DATE,format="%Y")
rocne <- aggregate( cbind(HUR_MEAN_TMP,KE_MEAN_TMP,ORL_MEAN_TEMP,PP_MEAN_TEMP,
                          HUR_MIN_TEMP,KE_MIN_TEMP,ORL_MIN_TEMP,PP_MIN_TEMP,
                          HUR_MAX_TEMP,KE_MAX_TEMP,ORL_MAX_TEMP,
                          PP_MAX_TEMP) ~ Year , df , mean )
hurbanovo <- subset(rocne,select=c(Year,HUR_MEAN_TMP,HUR_MIN_TEMP,HUR_MAX_TEMP))
kosice <- subset(rocne,select=c(Year,KE_MEAN_TMP, KE_MAX_TEMP, KE_MIN_TEMP))
poprad <- subset(rocne,select=c(Year,PP_MEAN_TEMP, PP_MAX_TEMP, PP_MIN_TEMP))
oravska_lesna <- subset(rocne,select=c(Year,ORL_MEAN_TEMP, ORL_MAX_TEMP, ORL_MIN_TEMP))
hurbanovo$Year <- as.numeric(hurbanovo$Year)
kosice$Year <- as.numeric(kosice$Year)
poprad$Year <- as.numeric(poprad$Year)
oravska_lesna$Year <- as.numeric(oravska_lesna$Year)

graf <- ggplot() +
  geom_line(data=hurbanovo, aes(x = Year, y = HUR_MEAN_TMP, color = "Hurbanovo")) +
  geom_line(data=kosice, aes(x = Year, y = KE_MEAN_TMP, color = "Košice")) +
  geom_line(data=poprad, aes(x = Year, y = PP_MEAN_TEMP, color = "Poprad"))+
  geom_line(data=oravska_lesna, aes(x = Year, y = ORL_MEAN_TEMP,
                                    color = "Oravská Lesná")) +
  labs(x = "Roky", y = "Stupne", title = paste0("Priemerné teploty za roky 1951-2021")) +
  scale_color_manual(name = "Meteostanice", values = c("Hurbanovo" = "red",
```

```
                                                    "Košice" = "yellow",
                                                    "Poprad" = "green",
                                                    "Oravská Lesná" = "blue"))+
  theme(
  legend.position = "bottom",
  plot.title = element_text(hjust = 0.5)

)
graf
```



4.3 Praktická ukážka č. 2 v jazyku Python

Podobne ako v jazyku R, môžeme **Markdown** použiť aj v Pythone. Využívame tú istú syntax, ktorá sa po exporte líši v zopár veciach. Vezmime si napríklad Jupyter Notebook, ktorý predstavuje interaktívne vývojové prostredie pre Python a použime ho na znázornenie rovnakého výstupu ako v predošlej kapitole v jazyku R.

Na nasledujúcich stranách môžeme vidieť výstupy z Jupyter Notebook vo forme PDF dokumentu. Narozdiel od **Markdownu** v jazyku R má **Markdown** v Jupyter Notebooku krajšie vizualizované bunky s kódom. Aj keď v samotnom R vieme meniť parameter `theme` v YAML hlavičke a tým pádom aj dizajn buniek, oproti Jupyter Notebooku neponúkajú krajší vzhľad.

Čo však v samotnom Jupyter Notebooku je možné použiť, ale len obmedzene sú rôzne LaTeXové príkazy/balíky na doplnenie dokumentu. Napríklad matematické výrazy môžeme napísať prostredníctvom JavaScriptovej knižnice `MathJax`, ktorú nainštalujeme cez `> Pip3 install mathjax`. Ak by sme chceli predsa len použiť LaTeX, je možné si vyexportovať výstup vo forme `.tex` súboru z Jupyter Notebooku a následne ho upraviť v rôznom online/offline LaTeXovom nástroji pomocou LaTeXovej syntaxe.

Obsah samotného dokumentu je rovnaký a takisto sú použité rovnaké postupy exploratívnej analýzy ako pri **R Markdown** dokumente.

V **R Markdown** je možné použiť skoro všetky LaTeXové príkazy/balíky a obohatiť tak dokument o prvky, ktoré samotný **Markdown** v Jupyter Notebooku neponúka.

Historic temperatures - Slovakia

Alena Stracenská[1] and Ondrej Šima[2]

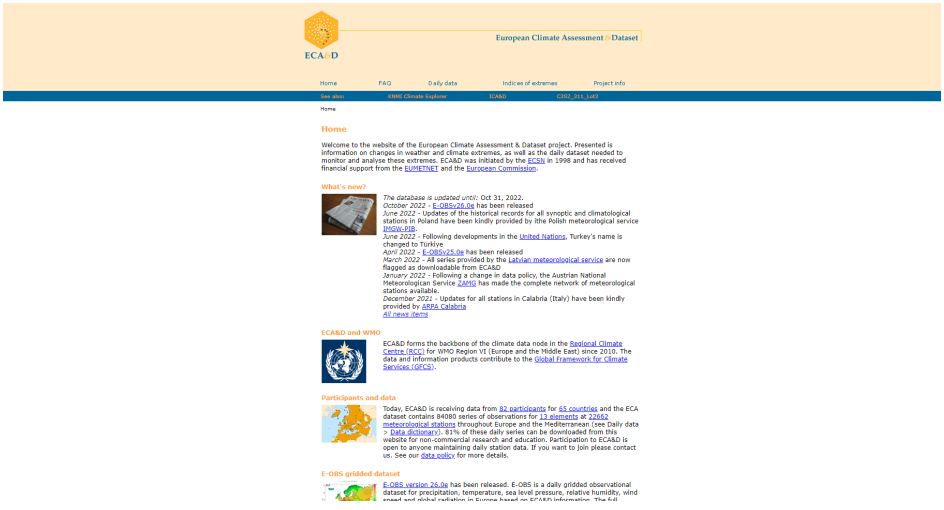
[1]stracensk@gmail.com
[2]osima1@student.euba.sk

1 Informácie o dátach

Dáta sme vzali v surovom stave zo stránky ecad.eu. Sú v nich uvedené minimálne, maximálne, priemerné teploty a rôzne ďalšie meteorologické údaje pre európske meteorologické stanice. Keďže sme chceli pracovať len s minimálnou, maximálnou a priemernou teplotou za posledných 70 rokov na Slovensku, tieto údaje nám spĺňali nasledovné meteorologické stanice:

- Hurbanovo (skratka: HUR)
- Košice (skratka: KE)
- Poprad (skratka: PP)
- Oravská Lesná (skratka: ORL)

Pre každú stanicu sme mali samostatný `.txt` súbor. Keďže úprava priamo v Pythone by bola mierne kompliko-vaná, rozhodli sme sa dáta do základného datasetu spracovať v Exceli.



Obrázok č. 1: Zdrojová stránka dát.

Výsledný dataset tvorí 13 stĺpcov, z ktorých jeden tvorí dátum a ďalsích dvanásť obsahuje teploty. V súbore sa takisto nachádzajú dni bez nameraných teplôt. Celý dataset si najprv očistíme a následne sa v grafickej forme pozrieme, ako sa vyvíjali teploty za posledných 70 rokov na Slovensku.

2 Exploratívna analýza

[1]:
```
#import pouzitych kniznic
import os as os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
import math
```

2.1 Ignorovanie warningov v celom kóde

[2]:
```
#ignorovanie warningov
import warnings
warnings.filterwarnings('ignore')
```

2.2 Importovanie .csv filu a nastavenie max. počtu riadkov

Funkcia **pd.set_option**, nam umožní nastaviť .max pocet riadkov, ktoré chceme zobraziť, a tie si nasledne korigujeme vo funkcii **head**.

[3]:
```
#import csv filu
df = pd.read_csv(r"C:/Users/strac/Desktop/5_rocnik_AS/ZS_22_23/ML/projekt_final/
 ↪vsetky_udaje_sprocesovane_nofix.csv", sep = ";")
#nastavenie maximalneho zobrazenia riadkov pre pandas, ktore si nasledne vieme␣
 ↪korigovat ich mnozstvom v heade
pd.set_option('display.max_rows', 30000)
df.head(5)
```

[3]:

	DATE	HUR_MEAN_TMP	KE_MEAN_TMP	ORL_MEAN_TEMP	PP_MEAN_TEMP	\
0	19510101	0	-9999	-9999	-9999	
1	19510102	33	-9999	-9999	-9999	
2	19510103	47	13	28	-8	
3	19510104	23	23	-6	0	
4	19510105	15	10	1	-20	

	HUR_MIN_TEMP	KE_MIN_TEMP	ORL_MIN_TEMP	PP_MIN_TEMP	HUR_MAX_TEMP	\
0	-22	-9999	-9999	-9999	-6	
1	4	-9999	-9999	-9999	-4	
2	19	-34	-7	-40	15	
3	11	-25	-17	-38	-19	
4	-7	-7	-40	-40	-22	

	KE_MAX_TEMP	ORL_MAX_TEMP	PP_MAX_TEMP
0	-9999	-9999	-9999
1	-9999	-9999	-9999
2	61	50	26
3	56	48	24
4	29	8	13

Ako môžeme vidieť vyššie hodnoty -9999 pri teplotách boli chýbajúce, tie sme nahradili hodnotou NaN. Následne sme zmenili stĺpec **DATE**, ktorý bol typu string na objekt **datetime**, pre lepšiu a jednoduchšiu

manipuláciu pri tvorbe grafov. Následne sme vynásobili teploty * 0.1, keďže boli hodnoty reprezentované ako dekadická (*decimal*) hodnota s pevným (v datasete skrytým) exponentom. Prvú časť čistenia dát teda máme za sebou.

[5]:
```
#nahradime zle hodnoty (-9999) prazdnou hodnotou (NaN)
df.replace(-9999,np.nan,inplace=True)
#uprava datumu (string) na objekt (datetime)
df["DATE"]=pd.to_datetime(df["DATE"],format="%Y%m%d")
#vynasobenie priemernych, max, min teplot, lebo zdrojova stranka udajov nacitavala␣
↪udaje len v celych cislach
df.loc[:, df.columns != "DATE"]=df.loc[:, df.columns != "DATE"]*0.1
```

V tomto kroku sme pomocou interpolácie nahradili chýbajúce hodnoty. Napríklad si môžeme všimnúť, že údaje k teplotám za prvé 2 dni roku 1951 chýbali, takže cez interpoláciu sme doplnili hodnoty z dňa 3.1.1951 do predošlých dvoch prvých januárových dní. Prečo sme to tak spravili ? Ak by sme chýbajúce hodnoty nahradili len priemerom, tak teplota napr. 11.2 stupňa 1.1.1951 by zrejme nedávala zmysel, preto sme sa rozhodli použiť interpoláciu, ktorá pre nás predstavovala lepšie riešenie, než úplne skreslené číslo z priemeru.

[7]:
```
#linearna interpolacia neplatnych hodnot, okrem datumov, max 5 za sebou iducich,
#obojsmerne (doplni hodnotu aj ked pred nou hodnota este neexistovala)
for stlpec in df.drop('DATE',axis=1).columns:
    df[stlpec].interpolate(method='linear',limit=5,limit_direction='both',inplace=True)
```

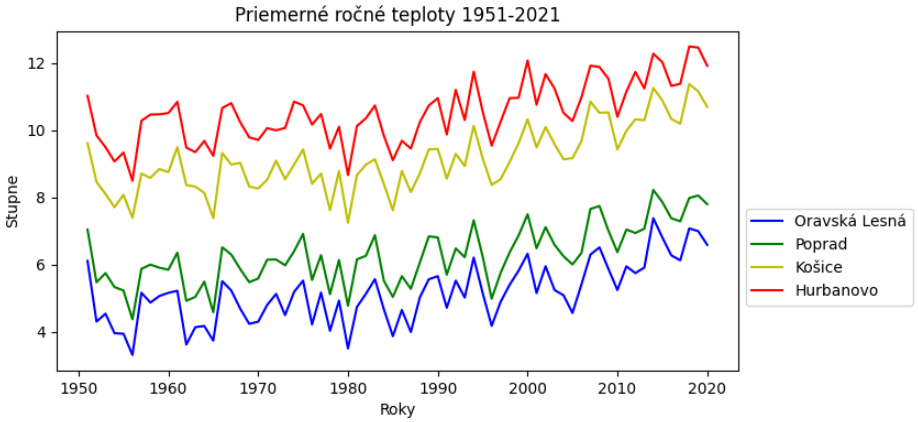
Následne sme pri prechádzaní dataframe objavili, že od dátumu 1.1.2021 zdroj datasetu nedisponuje údajmi, tak sme sa ich rozhodli odstrániť a zároveň sme nastavili teploty na maximálne dve desatinné miesta.

[8]:
```
#odstranenie hodnot NaN od datumu 1.1.2021, dovod: stranka nedisponuje od tohto datumu␣
↪hodnotami od SHMU
#fyi pre riadky plati n-1, preto v kode od 31.12.2020
df.drop(df[(df['DATE'] > '2020-12-31')].index, inplace=True)
#nastavenie kazdej hodnoty dataframe na max 2 des. miesta
pd.set_option('float_format', '{:.2f}'.format)
#odstranenie zvysnych riadkov s nulovymi hodnotami
df.dropna(inplace = True)
#overenie, ci sa riadky dropli
df.isnull().sum()
```

[8]:
```
DATE            0
HUR_MEAN_TMP    0
KE_MEAN_TMP     0
ORL_MEAN_TEMP   0
PP_MEAN_TEMP    0
HUR_MIN_TEMP    0
KE_MIN_TEMP     0
ORL_MIN_TEMP    0
PP_MIN_TEMP     0
HUR_MAX_TEMP    0
KE_MAX_TEMP     0
ORL_MAX_TEMP    0
PP_MAX_TEMP     0
dtype: int64
```

V tomto kroku sme si spravili ročné priemery priemernej, maximálnej a minimálnej teploty pre všetky mestá za posledných 70 rokov. Priemerné ročné teploty sme následne zobrazili na grafe.

[9]:
```
#skusime rocne priemery
rocne=df.groupby(df.DATE.dt.year).mean()
hurbanovo = rocne[['HUR_MEAN_TMP', 'HUR_MAX_TEMP', 'HUR_MIN_TEMP']]
kosice = rocne[['KE_MEAN_TMP', 'KE_MAX_TEMP', 'KE_MIN_TEMP']]
poprad = rocne[['PP_MEAN_TEMP', 'PP_MAX_TEMP', 'PP_MIN_TEMP']]
kosice = rocne[['ORL_MEAN_TEMP', 'ORL_MAX_TEMP', 'ORL_MIN_TEMP']]
#tvorba grafu
plt.figure(figsize=(8,4))
plt.plot(rocne.index, rocne["ORL_MEAN_TEMP"], 'b', label = "Oravská Lesná")
plt.plot(rocne.index, rocne["PP_MEAN_TEMP"], 'g', label = "Poprad")
plt.plot(rocne.index, rocne["KE_MEAN_TMP"], 'y', label = "Košice")
plt.plot(rocne.index, rocne["HUR_MEAN_TMP"],'r', label = "Hurbanovo")
plt.legend(loc = 'best', bbox_to_anchor = (1, 0.5))
plt.title("Priemerné ročné teploty 1951-2021")
plt.xlabel("Roky")
plt.ylabel("Stupne")
plt.show()
```



4.4 Praktická ukážka č. 3

V tretej praktickej ukážke sme sa zamerali na tvorbu HTML dokumentu, ktorý v podstate môžeme nazvať aj HTML stránkou. Na jeho tvorbu sme využili vstavaný R dataset `mtcars` a balíček **`plotly`**. V YAML hlavičke sme špecifikovali, že kód má byť skrytý pri otvorení stránky pomocou parametra `code_folding: hide` a taktiež sme si špecifikovali tému cez parameter `theme` a pridali parameter `runtime: shiny`, kvôli funkčnosti `.tabset` parametrov. Ostatné parametre sú nám už známe z praktickej ukážky číslo 2 v R.

```
---
title: "R Markdown dataset mtcars"
author: "Alena Stracenska"
date: "`r format(Sys.time(), '%d %B %Y')`"
output:
  html_document:
    runtime: shiny
    code_folding: hide
    theme: flatly
    df_print: paged
    toc: true
    toc_float:
      collapsed: true
      smooth_scroll: false
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(plotly)
library(DT)
```

mtcars

`mtcars` je vstavaný dataset v jazyku R. Tento dataset obsahuje informácie
o 32 rôznych automobiloch, ktoré boli vyrobené v 70. rokoch 20. storočia.
Každý riadok je dedikovaný pre jeden automobil a obsahuje 11 premenných.
Viac o datasete sa môžeme dočítať na [mtcars](https://www.rdocumentation.org/
packages/datasets/versions/3.6.2/topics/mtcars). Celkovo máme v jazyku R
104 vstavaných datasetov a ich zoznam dostaneme po napísaní
príkazu `data()` do konzoly.

```{r}
summary(mtcars[c("mpg", "cyl")])
```
```

````
## Prehľad datasetu

Parameter `df_paged` v YAML hlavičke nám umožňuje zobraziť
interaktívne dataset a pozrieť z akých premenných sa skladá.
To funguje pri štandardnom HTML výstupe na localhoste.
Pri nasadzovaní na Shiny server je potrebné využiť balíček *`DT`*
a z neho funkciu `datatable()`, do ktorej treba dataset zabaliť.


```{r, rows.print = 6}
datatable(mtcars)
```

## Grafy {.tabset .tabset-fade .tabset-pills}

Z datasetu `mtcars` je možné vytvoriť prehľadné grafy pomocou balíčka `plotly`.


### 3D Scatter plot


```{r, warning = FALSE}

plot_ly(mtcars, x=mtcars$wt, y=mtcars$mpg, z=mtcars$hp,
        type="scatter3d", mode="markers",
        color=mtcars$drat, size=mtcars$qsec) %>%
  layout(scene=list(
    xaxis = list(title = "Weight (1000 lbs)"),
    yaxis = list(title = "Miles per gallon"),
    zaxis = list(title = "Gross horsepower)")))

```

### Box plot

```{r, warning = FALSE}

 plot_ly(y=mtcars$mpg,color = as.factor(mtcars$am),type = "box") %>%
  layout(xaxis = list(title = 'Transmission (0 = automatic, 1 = manual)'),
         yaxis = list(title = 'Miles per gallon'))

```

### Histogram

```{r, warning = FALSE, showlegend = F}

plot_ly(x=mtcars$mpg,type = "histogram",nbinsx=20,color = mtcars$hp) %>%
  layout(xaxis = list(title = 'Miles per gallon'),
         yaxis = list(title = 'Count'))

```
````

Obrázok 39: Vizualizácia HTML stránky, zdroj: [vlastné spracovanie]

4.5 Praktická ukážka č. 4

Vo štvrtej praktickej ukážke sme sa zamerali na tvorbu **R Markdown** dashboardu pomocou balíčka **`flexdashboard`**. Na vizualizáciu a spracovanie dát sme použili aj balíčky **`DT`** a **`plotly`**. Dáta, ktoré sme použili sa týkajú vizuálne potvrdených strát vojenskej techniky vo vojne na Ukrajine. Zdrojom dát je open-source projekt Oryxspioenkop. Samotné datasety sú dostupné na Kaggle.

V YAML hlavičke sme použili štandardné nastavenie pre tento typ dashboardu a pridali sme tiež parametre `theme` a `logo`. Rozloženie je spravené pre viacero stránok, na ktoré sa vieme preklikať. Prvá stránka `Tanks` zahŕňa stĺpcové grafy pre straty všetkých typov tankov, druhá stránka `Equipment incl. Tanks` zahŕňa koláčové grafy pre straty celkového vybavenia a na poslednej tretej stránke `Data` môžeme vidieť prehľad datasetov, s ktorými sme pracovali.

````
---
title: "Losses in Russia Ukraine War"
output:
  flexdashboard::flex_dashboard:
    theme: cerulean
    logo: ua.png
    orientation: columns
    vertical_layout: fill
---

```{r setup, include=FALSE}

library(flexdashboard)
library(DT)
library(plotly)
df_ukraine <- read.csv("losses_ukraine.csv", sep = ",", stringsAsFactors = FALSE)
df_russia <- read.csv("losses_russia.csv", sep = ",", stringsAsFactors = FALSE)

```

Tanks
==================================

### Ukraine

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

```
tanks <- df_ukraine[df_ukraine$equipment == "Tanks",]
total_tanks <- group_by(tanks, model)
sum_total_tanks <- data.frame(summarise(total_tanks, sum(losses_total)))
plot_ly(sum_total_tanks, x = ~model, y = ~sum.losses_total., type = 'bar',
name = ~model, color = ~model, textposition = "outside") %>%
layout(yaxis = list(title = 'Count'), xaxis = list(title = "Model",
categoryorder = "total descending"), barmode = 'group')
```

Russia

```{r}

tanks_russia <- df_russia[df_russia$equipment == "Tanks",]
total_tanks_russia <- group_by(tanks_russia, model)
sum_total_tanks_russia <- (data.frame(summarise(total_tanks_russia,
sum(losses_total))))
plot_ly(sum_total_tanks_russia, x = ~model, y = ~sum.losses_total.,
type = 'bar', name = ~model, color = ~model, textposition = "outside") %>%
layout(yaxis = list(title = 'Count'), xaxis = list(title = "Model",
categoryorder = "total descending"), barmode = 'group')

```

Equipment incl. Tanks
==========

Row {.tabset}

Ukraine

```{r}

total <- group_by(df_ukraine, equipment)
sum_total_ukraine <- data.frame(summarise(total, sum(losses_total)))
plot_ly(sum_total_ukraine, labels = ~equipment, values = ~sum.losses_total.,
type = 'pie') %>%
layout(title = 'Total War Equipment Losses - Ukraine',
xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE),
yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE))

```

Russia

```{r}
```

```
total <- group_by(df_russia, equipment)
sum_total_russia <- data.frame(summarise(total, sum(losses_total)))
plot_ly(sum_total_russia, labels = ~equipment, values = ~sum.losses_total.,
type = 'pie') %>%
layout(title = 'Total War Equipment Losses - Russia',
xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE),
yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE))

```

Data
==========

Row {.tabset}

Ukraine

```{r}

df_ukraine %>%
  datatable(options = list(pageLength = 30,
lengthMenu = c(10, 20, 50, 100)), filter = "bottom")

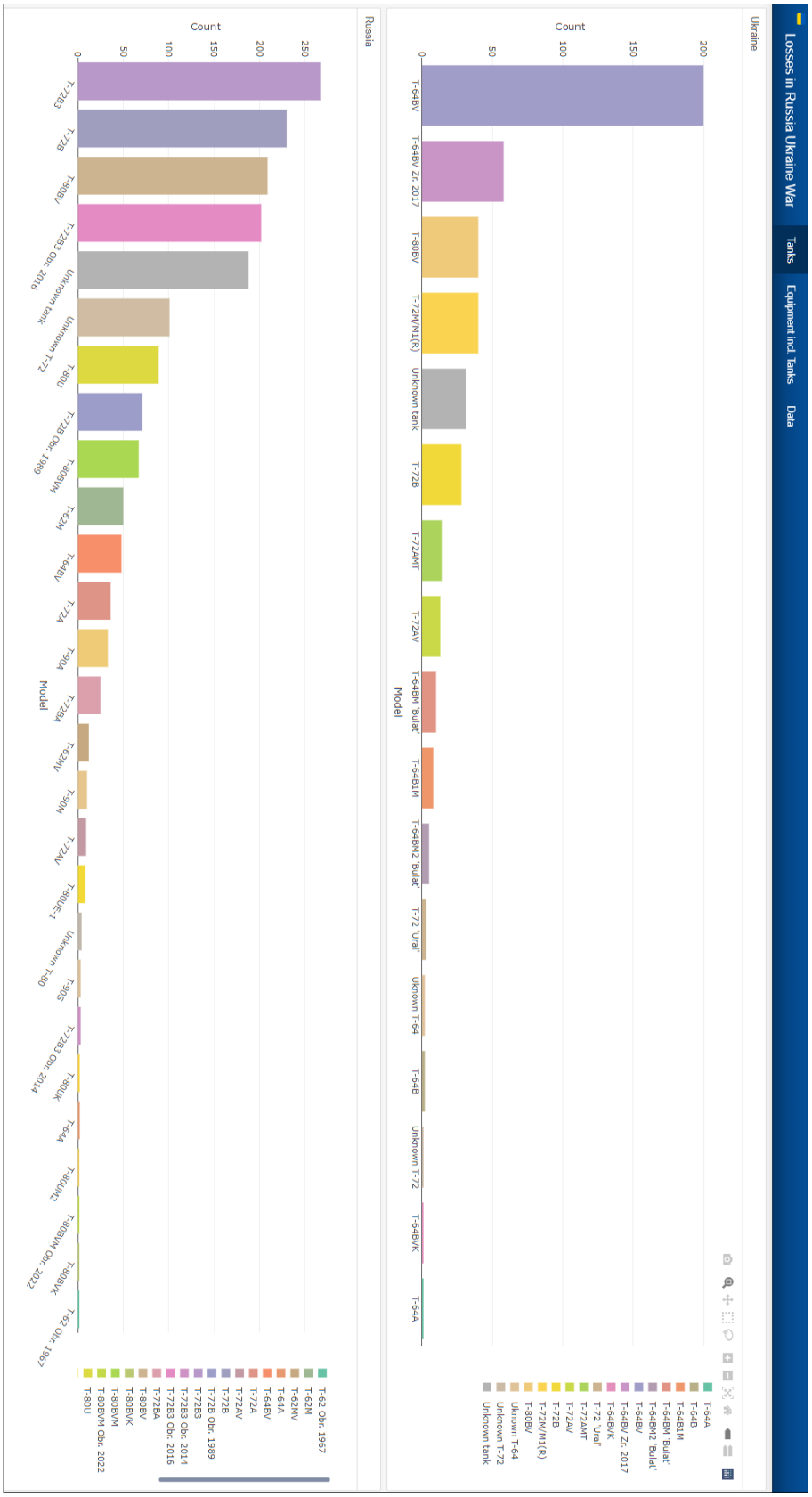
```

Russia

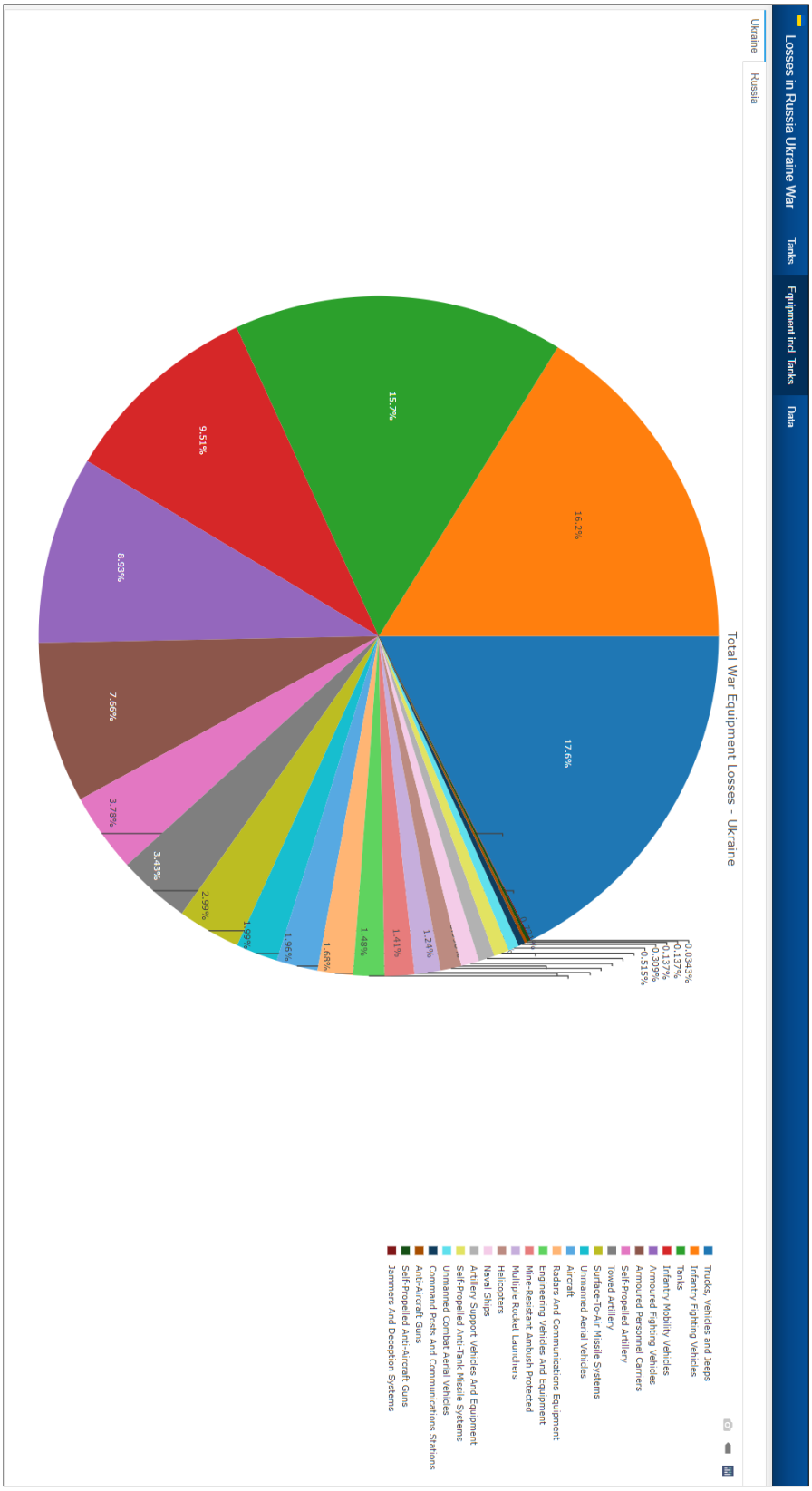
```{r}

df_russia %>%
  datatable(options = list(pageLength = 30,
lengthMenu = c(10, 20, 50, 100)), filter = "bottom")

```



Obrázok 40: Vizualizácia obsahu stránky `Tanks` v **R Markdown** dashboarde, zdroj: [vlastné spracovanie]



Obrázok 41: Vizualizácia obsahu stránky `Equipment incl. Tanks` v **R Markdown** dashboarde, zdroj: [vlastné spracovanie]

Obrázok 42: Vizualizácia obsahu stránky `Data` v **R Markdown** dashboarde, zdroj: [vlastné spracovanie]

4.6 Možnosti nasadenia na web

Pre nasadanie **R Shiny** aplikácií máme viacero možností. Patria k nim **Shiny server**, **shinyapps.io** a v komerčnom prostredí **RStudio Connect**.

RStudio Connect (resp. Posit Connect) je on-premise platená publikačná platforma pre RStudio. Umožňuje jednoduché a automatizované nasadenie a publikáciu R a Python aplikácií priamo z prostredia RStudio na **Shiny Server Pro**. Okrem **R Shiny** a **R Mardown** aplikácií umožňuje publikovať `Flask`, `Dash`, `Streamlit` a `Bokeh` aplikácie.

shinyapps.io je cloudová platforma, ktorá umožňuje zdieľanie a hostovanie Shiny aplikácií bez nutnosti správy a konfigurácie vlastného servera. Služba **shinyapps.io** je dostupná v obmedzenej bezplatnej verzii alebo niekoľkých platených úrovniach podľa potreby zákazníka. Bezplatná verzia umožňuje používateľovi hosťovať maximálne 5 **R Shiny** alebo Shiny Python aplikácií súčasne, s obmedzením do 25 aktívnych hodín mesačne (čas keď je aplikácia aktívne vyťažovaná).

Spôsob publikácie **R Shiny** aplikácie, ktorý sme zvolili v tejto práci je on-premise **Shiny server**. Jedná sa o open-source program ktorý umožňuje publikáciu **R Shiny** aplikácií a **R Mardown** dokumentov. **Shiny server** je v súčasnosti dostupný v komunitnej, open-source verzii a v platenej verzii **Shiny Server Pro**. Dostupné sú balíky pre Linuxové distribúcie `RHEL`, `Debian/Ubuntu` a `SUSE` a distribúcie vychádzajúcich z nich.

4.6.1 Nasadenie aplikácie na Shiny server

Pred samotnou inštaláciou je potrebné pripraviť si prostredie na ktoré budeme **Shiny server** inštalovať. V prípade publikácie aplikácie do internetu je vhodné zabezpečiť si registráciu domény a nanajvýš vhodné pred **Shiny server** postaviť reverzné proxy. V našom príklade budeme používať virtuálny server s operačným systémom `Ubuntu server 22.04`. Tento návod bude teda vhodný aj pre iné Linux distribúcie založené na `Debiane`. V prípade použitia inej podporovanej distribúcie (`RHEL` a `SUSE`) sa môžu niektoré príkazy a koncepty líšiť, návod zverejnený v dokumentácii **Shiny server** sa venuje aj týmto distribúciám.

V našom prípade tiež použijeme `NGINX` ako reverzné proxy a vygenerujeme a podpíšeme SSL certifikát pomocou `Let´s Encrypt` nástroja.

Inštalácia

Pred samotnou inštaláciou **Shiny servera** je potrebné ešte nainštalovať samotné R a **`shiny`** balíček, no spravidla ešte než začneme s inštaláciami je vhodné vykonať aktualizáciu repozitárov a aktualizáciu balíčkov a systému. Samotné R potom nainštalujeme príkazom:

```
sudo apt install r-base
```

Balíček **`shiny`** nainštalujeme buď z konzoly samotného R (spustenej s root oprávneniami), alebo pomocou príkazu ktorý spustí R na pozadí:

```
$ sudo su - -c \
"R -e \"install.packages('shiny', repos='https://cran.rstudio.com/')\""
```

Podobným systémom môžeme postupne nainštalovať aj ostatné R balíčky potrebné pre našu aplikáciu. `.deb` balík `Shiny server` potom stiahneme a nainštalujeme:

```
$ wget \
https://download3.rstudio.org/ubuntu-14.04/x86_64/shiny-server-1.5.15.953-amd64.deb
$ sudo apt install ./shiny-server-1.5.15.953-amd64.deb
```

Alternatívne môžeme použiť aj aplikáciu `gdebi`, ktorá je určená na rozbaľovanie `.deb` balíkov, pričom automaticky doinštaluje potrebné závislosti:

```
$ sudo apt-get install gdebi-core
$ wget \
https://download3.rstudio.org/ubuntu-14.04/x86_64/shiny-server-1.5.15.953-amd64.deb
$ sudo gdebi shiny-server-1.5.15.953-amd64.deb
```

Po úspešnom nainštalovaní by sa nám mala spustiť služba `shiny-server.service`, čo môžeme overiť pomocou `systemctl`:

```
$ sudo systemctl status shiny-server.service
shiny-server.service - ShinyServer
     Loaded: loaded (/etc/systemd/system/shiny-server.service;
     enabled; vendor preset: enabled)
     Active: active (running) since Fri 2023-02-24 15:06:34 CET;5h 44min ago
   Main PID: 8557 (shiny-server)
      Tasks: 11 (limit: 4575)
     Memory: 44.1M
        CPU: 4.820s
     CGroup: /system.slice/shiny-server.service
             8557 /opt/shiny-server/ext/node/bin/
             shiny-server /opt/shiny-server/lib/main.js
```

V prípade ak `Shiny server` nie je aktívny, môžeme sa pokúsiť o jeho spustenie štandardne, príkazom:

```
$ sudo systemctl start shiny-server.service
```

`Shiny server` je v základnej konfigurácii publikovaný na porte `3838` a štandardne je s ním pribalená aj ukážková stránka s informáciami a dvomi ukážkovými Shiny aplikáciami. Táto ukážková web aplikácia by mala byť nasadená spolu s inštaláciou, takže si funkčnosť **Shiny servera** môžeme overiť aj vďaka nej, je dostupná v prehliadači na `http://localhost:3838`. Ak pracujeme so vzdialeným serverom, ku ktorému nemáme pripojenie na lokálnej sieti ale je pripojený len do internetu (VPS, VDS, Cloud), nie je vhodné publikovať samotný **Shiny server** (povoliť pripojenie na port 3838) do internetu bez proxy servera. Dostupnosť web aplikácie si v takom prípade môžeme overiť napríklad aj pomocou techniky `"SSH port tuneling"`:

```
ssh -L "lokalny_port":127.0.0.1:"vzdialeny_port" "username"@"server"
```

Napríklad:

```
ssh -L "80":127.0.0.1:"3838" "username"@"server"
```

Horeuvedený príkaz nám tuneluje port 3838 na strane servera na port 80 na strane klienta. Takýmto spôsobom sa ku našej Shiny aplikácii dostaneme z nášho PC. Táto technika je vhodná napríklad ku prístupu na `Web admin` rozhranie **Shiny servera** (pokiaľ

sa rozhodneme ho aktivovať) na vzdialenom serveri, bez toho, aby bolo toto `Web Admin` rozhranie publikované do siete.

Konfigurácia

Konfiguračný súbor **Shiny servera** je dostupný na nasledujúcej ceste: `/etc/shiny-server/shiny-server.conf`. Jeho obsah je uvedený nižšie.

```
$ sudo cat /etc/shiny-server/shiny-server.conf
# Instruct Shiny Server to run applications as the user "shiny"
run_as shiny;

# Define a server that listens on port 3838
server {
  listen 3838;

  # Define a location at the base URL
  location / {

    # Host the directory of Shiny Apps stored in this directory
    site_dir /srv/shiny-server;

    # Log all Shiny output to files in this directory
    log_dir /var/log/shiny-server;

    # When a user visits the base URL rather
    # than a particular application,
    # an index of the applications available
    # in this directory will be shown.
    directory_index on;
  }
}
```

Toto nastavenie je vyhovujúce pre bežné použitie Shiny aplikácie, v našom prípade môžeme doplniť jeden príkaz. Keďže prevádzkujeme len jednoduchú aplikáciu, príkazom `app_idle_timeout 0;` nastavíme, aby sa po odpojení relácie posledného aktívneho používateľa nikdy neterminoval proces v ktorom je spustené R. Týmto docielime o niečo rýchlejšie načítanie aplikácie pre prvého používateľa, ak je nie je na **Shiny server** pripojený žiadny ďalší používateľ.

```
$ sudo cat /etc/shiny-server/shiny-server.conf
# Instruct Shiny Server to run applications as the user "shiny"
```

```
run_as shiny;

# Define a server that listens on port 3838
server {
  listen 3838;

  # Define a location at the base URL
  location / {

    app_idle_timeout 0;
    # Host the directory of Shiny Apps stored in this directory
    site_dir /srv/shiny-server;

    # Log all Shiny output to files in this directory
    log_dir /var/log/shiny-server;

    # When a user visits the base URL rather
    #than a particular application,
    # an index of the applications available
    # in this directory will be shown.
    directory_index on;
  }
}
```

Možnosti pokročilej konfigurácie **Shiny servera** sú dostupné v dokumentácii.

Nasadenie a prevádzka aplikácie

Pred samotným nasadením aplikácie je potrebné, aby boli na serveri nainštalované všetky potrebné balíčky, inak **Shiny server** nespustí Shiny aplikáciu. Môžeme tak spraviť zo samotnej konzoly jazyka R spustenej so zvýšenými oprávneniami alebo pomocou príkazu, ktorý spustí R na pozadí:

```
$ sudo su - -c "R -e \
\"install.packages('"balicek"', repos='https://cran.rstudio.com/')\""
```

Je nutné podotknúť, že niektoré balíčky, ako napríklad `tidyquant` čerpajú závislosti zo systémových balíkov. Ak takáto závislosť nie je splnená, R nenainštaluje balíček a na základe chybového hlásenia je potrebné nainštalovať daný balíček, napríklad pomocou `apt`.

Priečinok, z ktorého **Shiny server** číta zdrojové kódy, sa nachádza nastavený v súbore `/etc/shiny-server/shiny-server.conf`, štandardne sa jedná o `/srv/shiny-server`.

Shiny server používa stromovú štruktúru, takže názov priečinka zodpovedá `URI`. Zdrojové kódy našej aplikácie vložíme do priečinka `markets` ako `.R` súbor (absolútna cesta: `/srv/shiny-server/markets/app.R`), aplikácia bude dostupná z `URI /markets`. V prípade že máme viacero aplikácií, môžeme ich takýmto spôsobom vkladať do jednotlivých priečinkov a budú oddelené do samostatných `URI`.

Pri prevádzke akejkoľvek aplikácie sú dôležité `logy`. Nastavenie priečinku, do ktorého bude **Shiny server** generovať `logy` je dostupné v konfigurácii. Štandardné umiestnenie je `/var/log/shiny-server`. K dispozícii máme `/var/log/shiny-server.log`, ktorý zaznamenáva beh samotného **Shiny servera**. `Logy` ku jednotlivým `shiny` aplikáciám sa nachádzajú v priečinku `/var/log/shiny-server/`. Pre každú aplikáciu a každé spustenie **Shiny servera** sa vytvorí súbor v tvare `<nazov>-shiny-<yyyymmdd>-<HHMMSS>-<cislo socketu alebo portu>.log`. Napríklad:

```
$ sudo less markets-shiny-20230222-235135-38905.log

su: ignoring --preserve-environment, it's mutually exclusive with --login

Attaching package: ‘shinydashboard’

The following object is masked from ‘package:graphics’:

    box

Loading required package: lubridate

Attaching package: ‘lubridate’

The following objects are masked from ‘package:base’:

    date, intersect, setdiff, union

Loading required package: PerformanceAnalytics
Loading required package: xts
Loading required package: zoo

Attaching package: ‘zoo’

The following objects are masked from ‘package:base’:

    as.Date, as.Date.numeric
```

```
################################### WARNING ###################################
# We noticed you have dplyr installed. The dplyr lag() function breaks how    #
# base R's lag() function is supposed to work, which breaks lag(my_xts).      #
#                                                                             #
# If you call library(dplyr) later in this session, then calls to lag(my_xts) #
# that you enter or source() into this session won't work correctly.          #
#                                                                             #
# All package code is unaffected because it is protected by the R namespace   #
# mechanism.                                                                  #
#                                                                             #
# Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
#                                                                             #
# You can use stats::lag() to make sure you're not using dplyr::lag(), or you #
# can add conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop   #
# dplyr from breaking base R's lag() function.                                #
################################### WARNING ###################################

Attaching package: ‘PerformanceAnalytics’

The following object is masked from ‘package:graphics’:

    legend

Loading required package: quantmod
Loading required package: TTR
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo

Attaching package: ‘plotly’

The following object is masked from ‘package:ggplot2’:

    last_plot

The following object is masked from ‘package:stats’:

    filter

The following object is masked from ‘package:graphics’:

    layout


Listening on http://127.0.0.1:38905
Getting holdings for DOW
Getting holdings for DOW
```

V tomto `log` súbore je zaznamenaný štandardný výstup `stdout` a chybový výstup `stderr` z R. Tento `log` môže byť užitočný napríklad pri nefunkčnosti Shiny aplikácie z dôvodu chyby v zdrojovom kóde. **Shiny server** nezaznamenáva HTTP požiadavky, čo je ďalší dôvod prečo spolu so serverom použiť reverzné proxy.

Ďalším dôvodom prečo je pri prevádzke open source verzie **Shiny servera** vhodné použiť reverzné proxy je nemožnosť použitia `SSL/TLS`.

4.6.2 Zabezpečenie Shiny servera

Niekoľko dôvodov, prečo nie je vhodné publikovať samotný **Shiny server**, najmä open-source verziu, do internetu bolo opísaných už v predchádzajúcej kapitole. Okrem už spomínaných dôvodov ako implementácia SSL/TLS alebo zaznamenávanie HTTP požiadaviek nám poskytne použitie reverzného proxy aj ďalšie výhody. K daným výhodam patria všeobecne väčšia odolnosť voči `fuzzingu` (*technika, keď útočník hľadá chyby na stránke zväčša automatizovaným vypĺňaním hodnôt do polí v HTTP hlavičke a tele*) a `scanom`. Ďalej sú to praktické a prevádzkové dôvody ako možnosť publikovania aplikácie na poddomény, `URL rewrite` pravidlá, použitie `load balancera` alebo aj možná implementácia bezpečnostných prvkov ako `Webový aplikačný firewall` (`WAF`) a iné. Jednou z veľkých výhod použitia reverzného proxy servera je možnosť nastavenia bezpečnostných hlavičiek kvôli ochrane proti technikám ako `clickjacking`, `Cross-Site Scripting` a iné. V našom prípade použijeme proxy `NGINX`. V prípade potreby je možné použiť napríklad aj `Apache` server nakonfigurovaný ako proxy, `Caddy`, `HAProxy` alebo iný proxy server ktorý umožňuje funkcionalitu reverzného HTTP proxy a komunikáciu cez websockety.

Inštalácia a konfigurácia NGINX

`NGINX` môžeme nainštalovať priamo z repozitára, pomocou:

```
$ sudo apt install nginx
```

Správne nainštalovaný a spustený `NGINX` môžeme skontrolovať pomocou `systemctl`.

Konfigurácie pre jednotlivé inštancie vytvárame v `/etc/nginx/sites-available/`,

no NGINX číta konfigurácie z /etc/nginx/sites-enabled/. To nám umožní si konfiguráciu celú pripraviť a potom pomocou vytvorenia symbolického odkazu konfiguráciu vložiť do priečinka, ktorý NGINX číta. V našom prípade sme si pre náš **Shiny server** vytvorili súbor shiny:

```
$ sudo vi /etc/nginx/sites-available/shiny
```

Pomocou textového editora sme do neho vložili konfiguráciu:

```
http {
  map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
  }

  server {
    listen 80;
    #vymyslena nefunkcna domena
    server_name nazov-domeny.sk;
    #alternativne mozeme uviest IP adresu
    #alebo neuvadzat nic a mal by pocuvat na 0.0.0.0

    location / {
      proxy_pass http://localhost:3838/market;
      proxy_redirect / $scheme://$http_host/;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection $connection_upgrade;
      proxy_read_timeout 20d;
      proxy_buffering off;
    }
  }
}
```

Potom sme vytvorili dynamický odkaz.

```
sudo ln -s /etc/nginx/sites-available/shiny /etc/nginx/sites-enabled/shiny
```

a znovu načítali službu NGINX.

```
sudo systemctl reload nginx
```

Tým sme dosiahli niekoľko vecí. NGINX teraz počúva na porte 80 všetky požiadavky ktoré smerujú na adresu http://nazov-domeny.sk. Na túto adresu NGINX posúva obsah, ktorý má dostupný na http://localhost:3838/market, čiže našu konkrétnu Shiny aplikáciu. V prípade že máme nasadených viac Shiny aplikácií alebo **Markdown** dokumentov, môžeme napríklad nastaviť proxy_pass na kmeňovú URI, v ktorej môže mať **Shiny server** HTML súbor s odkazmi na jednotlivé Shiny aplikácie.

Pokročilé nastavenie NGINX servera je mimo rozsah tejto práce, keďže sa jedná o veľmi rozsiahly nástroj s veľkým množstvom funkcionalít. V spojení so **Shiny serverom** je zaujímavá napríklad možná konfigurácia, ktorá použitím URL rewrite pravidla môže vkladať Shiny aplikácie do stromovej štruktúry URI inej web aplikácie spustenej na inom ako **Shiny serveri**. Pre takéto a podobné použitia NGINX spolu so **Shiny serverom** je vhodné, aby sa používateľ oboznámil s NGINX dokumentáciou.

Zabezpečenie Shiny aplikácie - TLS

TLS (Transport Layer Security) je kryptografický protokol, ktorého najväčšie využitie je práve na kryptografické zabezpečenie HTTP spojenia. Keďže v našom prípade pomocou HTTP neprenášame citlivé dáta, môže sa zdať, že je šifrovanie prenášaného obsahu pomocou TLS zbytočné. Napriek tomu aj naša Shiny aplikácia benefituje z použitia TLS. Okrem ochrany pred neautorizovaným čítaním citlivých dát prenášaných po sieti, TLS zabezpečuje integritu a dôveryhodnosť prenášaných dát.

Pre našu aplikáciu využijeme na overenie a podpis nášho TLS certifikátu bezplatnú službu Let´s Encrypt. Ďalšou výhodou je dostupnosť jednoduchej aplikácie certbot a jej plugin modul pre NGINX, vďaka čomu nám certbot nielen certifikát vygeneruje, zabezpečí jeho podpísanie certifikačnou autoritou, ale aj upraví konfiguráciu NGINX pre použitie s konkrétnym TLS certifikátom. Balík s aplikáciou certbot sa nenachádza v základných Ubuntu repozitároch, preto si pred inštaláciou pridáme do APT príslušný repozitár. Následne nainštalujeme certbot a jeho plugin certbot-nginx:

```
sudo add-apt-repository ppa:certbot/certbot && sudo apt update
sudo apt install python3-certbot-nginx
sudo apt-get install certbot
```

Následne môžeme pristúpiť ku vygenerovaniu certifikátu s `NGINX pluginom` pre zvolenú doménu a prípadné poddomény.

```
sudo certbot --nginx -d nazov-domeny.sk -d www.nazov-domeny.sk
```

`certbot` nám doplnil informácie o certifikáte do konfiguračného súboru pre `NGINX`.

```
map $http_upgrade $connection_upgrade {
  default upgrade;
  '' close;
}

server {
         #vymyslena nefunkcna domena
         server_name nazov-domeny.sk;

        location / {
                proxy_pass http://localhost:3838/markets/;
                proxy_redirect / $scheme://$http_host/;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection $connection_upgrade;
                proxy_read_timeout 20d;
                proxy_buffering off;
        }
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/nazov-domeny.sk/fullchain.pem;
    # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/nazov-domeny.sk/privkey.pem;
    # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

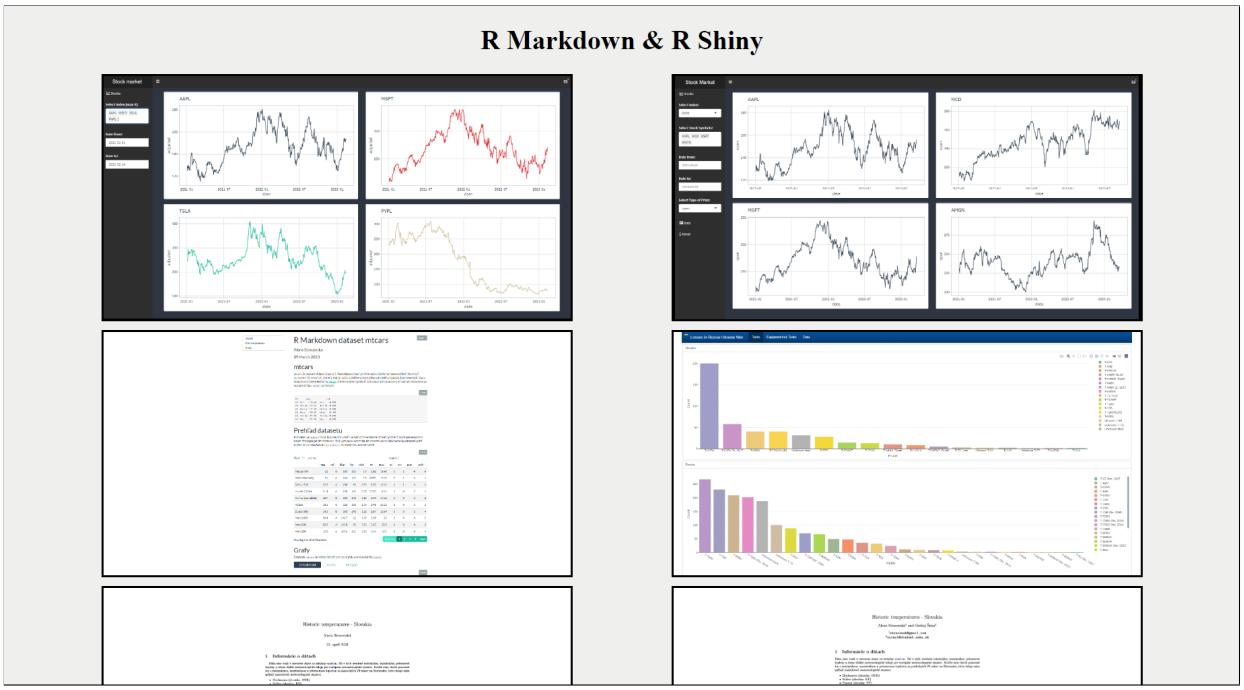
server {
    if ($host = nazov-domeny.sk) {
        return 301 https://$host$request_uri;
    } # managed by Certbot
        listen 80;
        server_name nazov-domeny.sk;
    return 404; # managed by Certbot
}
```

Riadky doplnené `certbotom` sú označené komentárom `# managed by Certbot`. Ako možno vidieť, `certbot` do konfiguračného súboru pridal cesty ku `SSL/TLS` certifikátom, nastavil port na ktorom `NGINX` počúva na `443` a na port `80` nastavil vrátenie `HTTP` kódu `404`.

Takisto je nastavené presmerovanie na `HTTPS URI` pre všetky spojenia ktoré podľa `HTTP` hlavičky `host:nazov-domeny.sk` dopytujú doménu. [24],[43]

4.7 Komplexný prehľad

Podobným spôsobom ako sme nasadili pokročilú aplikáciu sme nasadili aj ďalšie praktické ukážky. Predpokladom bolo vytvorenie jednoduchej stránky, ktorá nás po kliknutí na obrázok presmeruje na požadovaný **R Markdown** dokument alebo **R Shiny** aplikáciu. Všetky praktické ukážky nájdeme nasadené na stránke shiny.stracenska.sk.



Obrázok 43: Vizualizácia výslednej stránky s nasadenými praktickými ukážkami, zdroj: [vlastné spracovanie]

Záver

Hlavným cieľom práce bolo vytvoriť návod pre rozhrania **R Markdown** a **R Shiny**, opísať technickú špecifikáciu každého rozhrania, jednotlivé funkcie a následne deklarovať použitie niektorých funkcií na praktických príkladoch a opísať nasadenie webovej aplikácie, vrátane nasadenia ostatných praktických príkladov na web. Ciele, ktoré sme si zadali, boli splnené.

Napriek tomu, že rozhranie **R Markdown** ponúkalo viacero možností výstupných formátov rozhodli sme sa vybrať, opísať funkcie a následne v praktickej forme demonštrovať využitie troch formátov, pri ktorých vieme, že sa najčastejšie používajú v praxi.

Pre porovnanie sme **Markdown** použili aj v jazyku Python. Zatiaľ čo tvorba takéhoto dokumentu v jazyku R nám poskytla viacero pokročilých fukcionalít, platforma Jupyter Notebook nám umožnila použiť **Markdown** v interaktívnej forme.

Pri rozhraní **R Shiny** sme opísali nie len funkcionalitu a použitie samotného **R Shiny** ale aj nadstavby **R Shiny dashboard**. **R Shiny dashboard** dopĺňa funkcionalitu samotnej Shiny webovej aplikácie a zjednodušuje jej použitie v praxi ako nástroj na prezentáciu dát.

V praxi je možné, vďaka **R Shiny** a **R Shiny dashboard**, vytvoriť v krátkom čase a bez nutnosti rozsiahlej znalosti webových technológií reaktívnu webovú aplikáciu, natívne v jazyku R a využiť tak aspekty tohto programovacieho jazyka. Vďaka open-source koncepcii je možné toto rozhranie použiť na súkromné projekty, v akademickom prostredí alebo aj v biznis prostredí. Použitie toho rozhrania ako kritického informačného systému alebo pre podporu kritických systémov a procesov v biznis prostredí je umožnené vďaka komerčne licencovaným produktom s technickou podporou ako Shiny Server Pro a R Studio Connect.

Použitie **R Shiny** aplikácií a ich nasadenie v intranete alebo internete so sebou ale nesie bezpečnostné riziká. Jazyk R ako taký nemá implementované bezpečnostné opatrenia, ktoré majú implementované iné programovacie jazyky používané primárne na tvorbu

webových aplikácií. Z toho dôvodu nemožno zdrojový kód napísaný v jazyku R považovať za bezpečný a je obzvlášť vhodné pri vývoji webovej aplikácie pomocou **R Shiny** dodržiavať zásady bezpečného programovania a vykonať aspoň základný hardening systému, na ktorom je aplikácia nasadená. Táto práca sa ale z dôvodu obmedzeného rozsahu venovala bezpečnosti Shiny webovej aplikácie len okrajovo, a to najmä pri nasadzovaní aplikácie na Shiny server.

Výsledky predkladanej záverečnej práce možno tiež chápať ako určitý návod pre používateľa o možnostiach využitia oboch rozhraní, konkrétne balíčkov `shiny` a `rmarkdown` a ich implementácia na štyroch praktických príkladoch a následné nasadenie aplikácie na Shiny server.

Zoznam použitej literatúry

[1] Britannica. 2022. *markup language* In britannica.com [online]. [citované dňa 28.10.2022]. Dostupné na internete: https://www.britannica.com/technology/markup-language.

[2] Carson Sievert, Richard Iannone, JJ Allaire, Barbara Borges. 2023. *flexdashboard: R Markdown Format for Flexible Dashboards* In pkgs.rstudio.com [online]. [citované dňa 27.1.2023]. Dostupné na internete: https://pkgs.rstudio.com/flexdashboard/.

[3] Chanin Nantasenamat. 2021. *R Shiny for Data Science Tutorial - Build Interactive Data-Driven Web Apps* In youtube.com [online]. [citované dňa 28.1.2023]. Dostupné na internete: https://youtu.be/9uFQECk30kA.

[4] CODEACADEMY. *What is R used for?* In codeacademy.com [online]. [citované dňa 10.10.2022]. Dostupné na internete: https://www.codecademy.com/resources/blog/what-is-r-used-for/.

[5] CRAN. *Contributed Packages* In cran.r-project.org [online]. [citované dňa 11.10.2022]. Dostupné na internete: https://cran.r-project.org/web/packages/.

[6] Daróczi Gergely. 2022. *Number of R packages submitted to CRAN* In gist.github.com [online]. 2022. [citované dňa 11.10.2022]. Dostupné na internete: https://gist.github.com/daroczig/3cf06d6db4be2bbe3368#file-number-of-submitted-packages-to-cran-png/.

[7] debruine. 2022. *Building Web Apps with R Shiny* In debruine.github.io [online]. [citované dňa 30.1.2023]. Dostupné na internete: https://debruine.github.io/shinyintro/index.html.

[8] Garret Grolemund. 2014. *The R Markdown cheat sheet* In shiny.rstudio.com [online]. [citované dňa 26.1.2023]. Dostupné na internete: https://shiny.rstudio.com/articles/rm-cheatsheet.html.

[9] Garrett Grolemund. 2014. *Introduction to interactive documents* In shiny.rstudio.com [online]. [citované dňa 03.02.2023]. Dostupné na internete: https://shiny.rstudio.com/articles/interactive-docs.html.

[10] Gary Ernest Davis. 2016. *Over 16 years of Project History* In blog.revolutionanalytics.com [online]. 2016. [citované dňa 10.10.2022]. Dostupné na internete: https://blog.revolutionanalytics.com/2016/03/16-years-of-r-history.html.

[11] GeeksForGeeks. *Difference between HTML and CSS* In geeksforgeeks.org [online]. [citované dňa 22.10.2022]. Dostupné na internete: https://www.geeksforgeeks.org/difference-between-html-and-css/.

[12] Gruber John. 2022. *Markdown: Syntax* In daringfireball.net [online]. 2022. [citované dňa 14.10.2022]. Dostupné na internete: https://daringfireball.net/projects/markdown/syntax#philosophy.

[13] Hadley Wickham, Garret Grolemund. 2016. *R for Data Science* In batrachos.com [online]. ISBN 978-1-491-91039-9. [citované dňa 26.1.2023]. Dostupné na internete: https://batrachos.com/sites/default/files/pictures/Books/Wickham_Grolemund_2017_R%20for%20Data%20Science.pdf.

[14] Jeroen Ooms. *Previous Releases of R for Windows* In cran.r-project.org [online]. [citované dňa 10.10.2022]. Dostupné na internete: https://cran.r-project.org/bin/windows/base/old/.

[15] Kasprzak Peter, Mitchell Lachlan, Kravchuk Olena, Timmins Andy. 2020. *Six Years of Shiny in Research - Collaborative Development of Web Tools in R* In The R Journal [online]. ISSN 2073-4859. [citované dňa 15.10.2022]. Dostupné na internete: https://journal.r-project.org/archive/2021/RJ-2021-009/RJ-2021-009.pdf.

[16] LaTeX-Project. *An introduction to LaTeX* In latex-project.org [online]. [citované dňa 03.02.2023]. Dostupné na internete: https://www.latex-project.org/about/.

[17] MDN Web Docs. 2022. *CSS: Cascading Style Sheets* In developer.mozilla.org [online]. [citované dňa 22.10.2022]. Dostupné na internete: https://developer.mozilla.org/en-US/docs/Web/CSS.

[18] MDN Web Docs. 2022. *What is JavaScript?* In developer.mozilla.org [online]. [citované dňa 22.10.2022]. Dostupné na internete: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript#so_what_can_it_really_do.

[19] Netcraft. 2023. *January 2023 Web Server Survey* In news.netcraft.com [online]. [citované dňa 26.02.2023]. Dostupné na internete: https://news.netcraft.com/archives/2023/01/27/january-2023-web-server-survey.html.

[20] Nicholas Tierney. 2020. *R Markdown for Scientists* In rmd4sci.njtierney.com [online]. [citované dňa 28.1.2023]. Dostupné na internete: https://rmd4sci.njtierney.com/.

[21] Nik Lilovski. 2022. *Package 'dashboardthemes'* In cran.r-project.org [online]. [citované dňa 04.03.2023]. Dostupné na internete: https://cran.r-project.org/web/packages/dashboardthemes/dashboardthemes.pdf.

[22] Peng D. Roger. 2012. *R Programming for Data Science* In cs.upc.edu [online]. ISBN 9781365056826. [citované dňa 09.10.2022]. Dostupné na internete: https://www.cs.upc.edu/~robert/teaching/estadistica/rprogramming.pdf.

[23] Ramnath Vaidyanathan, Kenton Russel, RStudio. 2014-2015. *htmlwidgets for R* In htmlwidgets.org [online]. [citované dňa 27.1.2023]. Dostupné na internete: http://www.htmlwidgets.org/index.html.

[24] Reese, Will. 2008. *Nginx: The high-performance web server and reverse proxy* In linuxjournal.com [online]. [citované dňa 25.02.2023]. Dostupné na internete: https://www.linuxjournal.com/article/10108.

[25] RFC Production Center. 1999. *Hypertext Transfer Protocol – HTTP/1.1* In rfc-editor.org [online]. [citované dňa 04.03.2023]. Dostupné na internete: https://www.rfc-editor.org/rfc/rfc2616?f.

[26] RFC Production Center. 2000. *Upgrading to TLS Within HTTP/1.1* In rfc-editor.org [online]. [citované dňa 04.03.2023]. Dostupné na internete: https://www.rfc-editor.org/rfc/rfc2817.

[27] RFC Production Center. 2011. *The Secure Sockets Layer (SSL) Protocol Version 3.0l* In rfc-editor.org [online]. [citované dňa 04.03.2023]. Dostupné na internete: https://www.rfc-editor.org/rfc/rfc6101.

[28] RFC Production Center. 2011. *The WebSocket Protocol* In rfc-editor.org [online]. [citované dňa 04.03.2023]. Dostupné na internete: https://www.rfc-editor.org/rfc/rfc6455.

[29] RFC Production Center. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3l* In rfc-editor.org [online]. [citované dňa 04.03.2023]. Dostupné na internete: https://www.rfc-editor.org/rfc/rfc8446.

[30] Robin Linacre. 2016. *Relationship between R Markdown, Knitr, Pandoc, and Bookdown* In stackoverflow.com [online]. [citované dňa 28.10.2022]. Dostupné na internete: https://stackoverflow.com/questions/40563479/relationship-between-r-markdown-knitr-pandoc-and-bookdown.

[31] Ross Ihaka. *The R Project: A Brief History and Thoughts About the Future* In The University of Auckland [online]. [citované dňa 10.10.2022]. Dostupné na internete: https://www.stat.auckland.ac.nz/~ihaka/downloads/Massey.pdf.

[32] RStudio. 2014. *Background Shiny and HTML* In rstudio.github.io [online]. [citované dňa 02.02.2023]. Dostupné na internete: http://rstudio.github.io/shinydashboard/structure.html.

[33] RStudio. 2014. *Shiny Themes* In rstudio.github.io [online]. [citované dňa 03.02.2023]. Dostupné na internete: https://rstudio.github.io/shinythemes/.

[34] RStudio. 2023. *rmarkdown: Dynamic Documents for R* In rmarkdown.rstudio.com [online]. [citované dňa 16.03.2023]. Dostupné na internete: https://rmarkdown.rstudio.com/docs/news/.

[35] RStudio Team. 2022. *RStudio: Integrated Development Environment for R* In RStudio, PBC [online]. [citované dňa 27.10.2022]. Dostupné na internete: http://www.rstudio.com/.

[36] Santiago Rodrigues Manica. 2022. *Bye RStudio, Hello Posit!* In medium.com [online]. [citované dňa 27.10.2022]. Dostupné na internete: https://medium.com/evidentebm/bye-rstudio-hello-posit-f1d0a4213188.

[37] Thomas Park. *Free themes for Bootstrap* In bootswatch.com [online]. [citované dňa 01.02.2023]. Dostupné na internete: https://bootswatch.com/.

[38] University of Oslo. 2022. *Quarto - Next generation R Markdown* In ub.uio.no [online]. [citované dňa 16.03.2023]. Dostupné na internete: https://www.ub.uio.no/english/courses-events/events/all-libraries/2023/digital-scholarship-days/quarto.html#:~:text=Register-,Quarto%20is%20the%20next%20generation%20of%20R%20Markdown%2C%20and%20has,come%20before%20in%20R%20Markdown.

[39] Victor Perrier, Fanny Meyer. *fresh* In dreamrs.github.io [online]. [citované dňa 03.02.2023]. Dostupné na internete: https://dreamrs.github.io/fresh/.

[40] w3schools. *HTML Introduction* In w3schools.com [online]. [citované dňa 16.10.2022]. Dostupné na internete: https://www.w3schools.com/html/html_intro.asp.

[41] w3schools. *JavaScript Where To* In w3schools.com [online]. [citované dňa 22.10.2022]. Dostupné na internete: https://www.w3schools.com/js/js_whereto.asp.

[42] Wickham Hadley. 2022. *Mastering Shiny* In mastering-shiny.org [online]. ISBN 978-1-492-04738-4. [citované dňa 15.10.2022]. Dostupné na internete: https://mastering-shiny.org/.

[43] Winston Chang, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerk, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, Barbara Borges. 2023. *shiny: Web Application Framework for R* In shiny.rstudio.com

[online]. [citované dňa 25.02.2023]. Dostupné na internete: https://shiny.rstudio.com/.

[44] Xie Yihui, Allaire J. J., Grolemund Garret. 2022. *R Markdown: The Definitive Guide* In bookdown.org [online]. ISBN 978-1-138-35942-0. [citované dňa 14.10.2022]. Dostupné na internete: https://bookdown.org/yihui/rmarkdown/.

[45] Yihui Xie, Christophe Dervieux, Emily Riederer. 2022. *R Markdown Cookbook* In bookdown.org [online]. [citované dňa 29.10.2022]. Dostupné na internete: https://bookdown.org/yihui/rmarkdown-cookbook/.