FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Vývoj aplikace pro RFM analýzu a její nasazení v analytickém prostředí Keboola Connection
Student:	Bc. Martin Hak
Vedoucí:	Ing. David Chobotský
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Implementujte aplikaci pro RFM segmentaci zákazníků/produktů v prostředí ETL nástroje Keboola Connection a to pro konkrétní obchodní zadání a problém. Aplikace bude znovupoužitelná a poskytne "on-demand" RFM analýzu pro data firem využívající prostředí Keboola Connection. Krom vývoje samotné aplikace také diskutujte její aplikaci na reálném příkladu firmy. Proveďte analýzu a simulaci dopadů, při vhodném využití RFM segmentace.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Karel Klouda, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 13. února 2018

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra aplikované matematiky

Diplomová práce

Vývoj aplikace pro RFM analýzu a její nasazení v analytickém prostředí Keboola Connection

Bc. Martin Hak

Vedoucí práce: Ing. David Chobotský

6. května 2018

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce Ing. Davidu Chobotskému za vstřícný přístup, poskytnuté konzultace a umožnění zapojení do tohoto projektu. Velmi děkuji také Ing. Jiřímu Tobolkovi z Bizztreat s.r.o. za poskytnuté praktické rady a informace v průběhu práce na projektu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. května 2018 .....

Odkaz na tuto práci

Hak, Martin. *Vývoj aplikace pro RFM analýzu a její nasazení v analytickém prostředí Keboola Connection.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Práce popisuje návrh a implementaci aplikace pro RFM analýzu v analytickém prostředí Keboola Connection. Dále je práce zaměřena na vizualizaci výsledků této aplikace a popis možností jejiho využití pro marketingová řešení. Po prostudování práce by čtenář měl být schopen porozumět problematice vývoje takové aplikace a být schopen využívat přínosů RFM analýzy.

Klíčová slova Keboola Connection, RFM Analýza, Segmentace, Tableau

Abstract

Thesis describes the design and implementation of the application for RFM analysis in the Keboola Connection analytical environment. Furthermore, the work is focused on visualizing the results of this application and describing the possibilities of it's use for marketing solutions. After reading the work, the reader should be able to understand the development issues of such an application and be able to use the benefits of RFM analysis.

Keywords Keboola Connection, RFM Analysis, Segmentation, Tableau

Obsah

Seznam obrázků

Úvod

Stručně o projektu

Projekt vývoje *aplikace pro RFM analýzu v analytickém prostředí Keboola Connection* vznikl se základní myšlenkou zjednodušit a zefektivnit proces RFM segmentace. RFM analýza je dnes často používána jako proof of concept a je tedy nezbytné, aby její použití zabralo minimum času a úsilí. Toho přesně se snaží docílit aplikace využívající stále populárnější ETL nástroj Keboola Connection.

Motivace

Jak již bylo řečeno, je potřeba mít k dispozici nástroj, pomocí něhož lze snadno a rychle provést RFM analýzu s minimálním úsilím. Se stále se zvětšujícím využíváním nástroje Keboola Connection by tak bylo velmi výhodné, kdyby RFM analýzu uměl provést již tento nástroj jako jeden z dalších kroků. Tím by odpadlo velké množství práce s přesunem dat a jejich formátováním, aby vyhovoval vstupům, které jednotlivé možné existující aplikace vyžadují, nemluvě o seznamování se s jejich prostředím.

Kromě samotného vývoje aplikace je motivací také rozšiřovat povědomí o RFM analýze a informovat čtenáře o možnostech jejího praktického využití. Dalším bodem je seznámit ho s užitečnou a rychlou metodou, pomoci mu seznámit se s jejími základními principy fungování a vysvětlit mu, proč je pro něj tato metoda užitečná.

Specifikace cílů

Hlavním cílem práce je vytvoření funkční aplikace pro RFM segmentaci v prostředí Keboola Connection.

Kromě vytvoření samotné aplikace pro segmentaci si práce klade za cíl také vytvoření její vhodné vizualizace tak, aby přinesla uživateli vhodný nástroj pro prezentaci výsledků, jež tato aplikace přinese.

Dalším cílem této práce je rozšiřovat povědomí o existenci tohoto užitečného nástroje a poskytnout čtenáři příležitost seznámit se s možnostmi jeho využití při řešení obchodního problému zabývajícího se cílením vhodných obchodních strategií při dalším marketingu.

Struktura práce

Struktura práce je rozdělena do následujících kapitol.

První kapitola (1) seznámí čtenáře s konceptem RFM analýzy a základními nástroji a prostředky, které jsou využity pro tvorbu aplikace. Po seznámení čtenáře s těmito prvky, jež jsou základními předpoklady pro pochopení další práce, je provedena analýza, která zahrnuje specifikaci funkčních a nefunkčních požadavků a představení modelu případů užití.

Druhá kapitola (2) se zabývá samotným návrhem aplikace pro RFM analýzu v prostředí Keboola Connection. Je zde představen model architektury, ve kterém je představena architektura zvolená pro aplikaci. Následně je prezentován návrhový model tříd, ve kterém jsou popsány všechny důležité třídy a metody aplikace, stejně tak jako vazby mezi nimi. Na závěr kapitoly je uveden model komunikace, který se zabývá fungováním nejdůležitějších metod aplikace a jejich vzájemné komunikace.

Třetí kapitola (3) představuje praktickou realizaci a nasazení aplikace. Kapitola rovněž rozebírá a diskutuje jednotlivé výstupy z aplikace stejně tak jako aplikaci možných přístupů k těmto výstupům.

Čtvrtá kapitola (4) popisuje způsob ověření a testování zvolených částí aplikace.

Kapitola 1

Analýza

Následující kapitola se primárně zaměřuje na sumarizaci a analýzu všech podstatných faktorů k úspěšnému splnění projektu. Lze zde najít popis současného stavu a návrh na jeho řešení, stejně jako popis všech důležitých nástrojů a prostředků, které k tomu jsou, nebo by mohly být, vhodným způsobem využity. Kapitola se také zabývá požadavky, které jsou na zvolené řešení kladeny a případy, které bude muset být schopné řešit.

1.1 Současný stav

V současné době není problém najít aplikaci na provedení RFM analýzy. Pokud jsou data dostatečně malá lze provést RFM analýzu snadno i v programu Microsoft Excel. Rovněž pro větší data jsou dostupné již hotové aplikace. Celý proces analýzy dat však nezahrnuje pouze samotnou segmentaci, nicméně vážou se k němu i další oblasti z oboru znalostního inženýrství jako je například předzpracování dat a vizualizace. Na trhu je celá řada aplikací, které provádí jednotlivé kroky tohoto procesu, nicméně pak je potřeba jednotlivé kroky ještě propojit dohromady, což zabírá nezanedbatelné množství času. V případě využití RFM analýzy pro proof of concept, na jehož základě se teprve zákazník rozhodne pro další spolupráci, je pak toto nezanedbatelné množství času promarněno nehledě na to, že se jedná o případ ke kterému dochází opakovaně a je proto vhodné ho zautomatizovat.

1.2 Rozbor možných řešení

Jednou z možností je navrhnout systém, který provede celý proces RFM analýzy od začátku do konce a to včetně všech dalších kroků jako je předzpracování dat a vizualizace. Takový systém však musí být velmi rozsáhlý a nikdy pravděpodobně nepokryje jednotlivé oblasti tak dobře jako specializované systémy.

Druhou možností je rozšířit již existující systém tak, aby podporoval přímé napojení na RFM analýzu. To přináší nespornou výhodu zachování odbornosti na jednotlivých dílčích specializovaných podsystémech. Je však nutné vybrat takový systém, který podporu implementace takového rozšíření umožní. Také se musí vybrat vhodný systém z pohledu propojení všech částí procesu, aby se pouze nepřenášel problém propojení systémů z jednoho systému na druhý.

Z obou diskutovaných řešení byla vybrána druhá varianta pro její nesporné výhody zachování odbornosti na jednotlivých dílčích podsystémech. Varianta číslo jedna také není vybrána z důvodu praktické nemožnosti obsáhnout dnešní rozsáhlou oblast znalostního inženýrství, což by nakonec pravděpodobně vedlo k vyřazení používání takového systému. Pro realizaci je vybráno snadno rozšiřitelné prostředí Keboola Connection s přímým napojením na vybraný vizualizační nástroj. Více o těchto nástrojích v podkapitole 1.4.

V prostředí Keboola Connection je možnost implementace také několika způsoby. První z nich je rozčlenění celého procesu do několika samostatných scriptů, které budou provádět dílčí kroky pro RFM analýzu. Toto řešení umožní maximálně se soustředit na problémy jednotlivých kroků a taktéž využít předpřipravených nástrojů prostředí Keboola Connection. Zároveň však přináší na uživatele požadavek hlubší znalosti celého procesu a také zvýšenou režii obsluhy pro každou prováděnou RFM analýzu. Druhou možností je implementace aplikace s přímou podporou pro prostředí Keboola Connection. To sice uživatele omezí pouze na možnosti použití podporované přímo touto aplikací, na druhou stranu se však jedná o řešení, které je maximálně jednoduché na použití a které lze použít bez nutnosti nastavování jednotlivých dílčích kroků. Z obou řešení je opět zvoleno druhé řešení a to zejména z důvodu své jednoduché znovupoužitelnosti pro uživatele. Další výhodou, která taktéž vedla k rozhodnutí pro tuto možnost je i skutečnost, že takto lze řešení prezentovat jako ucelený celek.

1.3 RFM Analýza

RFM analýza, někdy též nazývaná RFM segmentace, patří mezi základní nástroje pro segmentaci zákazníků (pro účely této práce bude uváděn termín zákazník, RFM analýza však může být prováděna pro různé systémy, a tak se může jednat pouze o uživatele a nebo ještě lépe o obecné entity daného systému). Jedná se o poměrně starý koncept, kdy zmínky o první primitivní formě RFM modelu pochází již z doby před více než 50 lety, kdy byla používána katalogizátory se smíšeným zbožím[1]. I přes poměrně jednoduchý koncept se tato metoda prokázala být úspěšnou v řadě různých odvětvích[2]. Nicméně i nadále bude pravděpodobně její nejběžnější použití při email marketingu, kde je oblíbená pro svoji jednoduchost a intuitivnost [3]. Že jde o užitečný nástroj dosvědčuje například i použití velkými společnostmi jako je IKEA nebo Baťa[4].

Písmena RFM v názvu zastupují jednotlivé dimenze, podle nichž lze zákazníky dělit a které jsou zároveň důležité z obchodního hlediska. R, které zastupuje Recency, je doba od posledního nákupu zákazníka. F, které zastupuje Frequency, je počet jeho nákupů. A nakonec M, které zastupuje Monetary, je objem utracených peněz - ať už se jedná o celkovou hodnotu nákupů zákazníka nebo například průměrně utracenou částku.

Průběh RFM analýzy si lze přiblížit na jednoduchém příkladě. Z transakčních údajů zákazníků lze snadno dopočítat jednotlivé dimenze (RFM), na jejichž základě budou zákazníci postupně rozděleni do několika skupin. Segmentace se pak provádí postupně napříč jednotlivými dimenzemi. Tedy nejdříve se zákazníci rozdělí na základě jejich Recency, poté jejich Frequency a nakonec podle jejich Monetary hodnoty. Nejčastěji se zákazníci v dané dimenzi dělí na předem definovaný počet stejně velkých částí. Nicméně pokud jsou k dispozici další informace nebo požadavky, tak to není nutností. Poté, co jsou zákazníci seřazeni podle jedné z vlastností, již nic nebrání jejich rozdělení (například je-li počet částí 5, tak prvních 20% zákazníku tvoří první segment, dalších 20% zákazníků druhý segment atd.). Následně jsou zákazníci seřazeni a rozděleni i pro další z vlastností. Nakonec jsou tedy všichni zákazníci rozděleni do segmentů podle recency, frequency i monetary. Jednou z výhod je, že takto jde rozdělit i malé datasety (jen s pár jednotkami či desítkami zákazníků) i když samozřejmě hodnota segmentace bude hodnotnější pro datasety větší.

Jestliže jsou pak jednotlivé segmenty kategorií očíslovány čísly (například v případě 5 segmentů čísly 1-5) tak lze mimo jiné určit jakési scóre všech zákazníků. Poté, co jsou jednotlivé vlastnosti seřazeny podle toho, jak jsou pro konkrétní studii důležité, vznikne žebříček od nejdůležitějších zákazníků po ty nejméně cenné. Ukázáno na příkladu - je-li 5 segmentů v každé kategorii a kategorie jsou důležité v pořadí frequency, recency, monetary, tak zákazník, který je přiřazen v každé kategorii do pátého segmentu (tedy má scóre 555) je nejcennějším zákazníkem systému (nakupuje nejčastěji (a to i v současné době) za největší množství peněz). Zákazník se scóre 111 bude nejhorší (nakoupil pouze jednou, před dlouhou dobou, za minimální částku). A zákazník, který má frequency segment číslo 3, recency segment číslo 2 a monetary segment číslo 4 bude mít scóre 324. To je také názorná ukázka zákazníka, na něhož se pak vyplatí cílit vybranou marketingovou kampaň, jelikož se jedná o pravidelného zákazníka, který objednává za velké částky, ale již dlouho nenakoupil. Je tak načase se mu připomenout, nejlépe s konkrétní výhodnou nabídkou. Existuje plno dalších způsobů interpretace RFM analýzy. Některým z nich se práce podrobněji věnuje v kapitole 3.3.

Od RFM je odvozeno plno dalších metod jako například RFD kde D zastupuje Duration (tedy například dobu, kterou zákazník strávil na naší stránce) či RFR kde druhé R zastupuje Reach (tedy například dosah v různých sociálních sítích) atp.[2]. Vždy se ale jedná o aplikaci na stejném principu, jen nad jinou doménou dat.

Jak již bylo řečeno, segmentace jednotlivých dimenzí lze provést různými

způsoby a ne vždy je možné upřednostnit nějaký způsob na úkor jiného. Segmenter z tohoto projektu podporuje následující způsoby segmentace:

1.3.1 Specifikování počtu segmentů

Jedním z nejednodušších přístupů k segmentaci je pouhé definování počtu segmentů. Toto je také nejčastější přístup především proto, že vychází z marketingové poučky, která říká, že horních 20% zákazníků tvoří 80% zisku[5]. Základem je tedy postupné seřazení zákazníků dle jejich vlastností recency, frequency a monetary. Následně jsou dle počtu segmentů vybráni jednotliví zákazníci na hranicích segmentů a jejich hodnota vlastností recency, frequency a monetary je brána jako hranice mezi jednotlivými segmenty. Zákazníci se pak pomocí těchto vlastností přiřadí do jednotlivých segmentů. Přístup definování hranic namísto prostého rozčlenění zákazníků je zde především pro zachování determinismu celé operace. V praxi totiž často nastanou situace, kdy velké množství zákazníků bude mít stejnou hraniční hodnotu. Pokud tedy není zaručeno, že seřazení zákazníků dopadne vždy stejně, nelze zaručit do kterého segmentu bude zákazník přiřazen. Navíc by pak bylo k zákazníkům se stejnou vlastností přistupováno odlišně, což je v přímém rozporu s důvody segmentace.

1.3.2 Vlastní hranice

V praxi se lze snadno setkat se situací, kdy je navíc přítomná specifická znalost, která má přímý vliv na to, jak by měla být segmentace provedena. Nemusí se však jednat přímo o konkrétní znalost, ale může to být také výskyt specifických požadavků, na jejichž základně je opět nutné segmentaci přizpůsobit. V takovém případě lze využít možnosti, kdy lze přímo specifikovat jednotlivé hranice. Další nespornou výhodou tohoto přístupu je možnost upravování hranic, které byly definovány způsobem specifikování počtu segmentů. Na základě zhodnocení výsledků specifikování počtu segmentů lze dojít k potřebě menších či větších úprav a tímto způsobem toho je možné snadno dosáhnout.

1.3.3 Specifikování počtu segmentů s postupným vyhodnocováním

Tvoří-li část dimenze početná skupina, která má stejnou hodnotu dané vlastnosti, podle níž se zákazníci člení a rozprostírá-li se tato skupina přes více než jeden segment, stane se v případě segmentace způsobem specifikování počtu segmentů to, že dva sousedící segmenty budou mít stejné hraniční hodnoty. To při následném třídění zákazníků vyústí v to, že jeden segment pohltí zákazníky z obou segmentů pod sebe a druhý zůstane prázdný. V mnoha případech to sice může dávat smysl, nicméně v některých dalších případech je žádoucí se této situaci vyhnout. Nástrojem, jak se tomuto stavu vyhnout, je metoda specifikování počtu segmentů s postupným vyhodnocováním. Místo toho, aby

proběhlo specifikování všech segmentů najednou, jsou tyto segmenty definovány postupně. Pokaždé, když je definován segment, jsou vyřazeni všichni zákazníci patřící do daného segmentu a další rozdělení na segmenty probíhá již jen nad zbylými zákazníky.

1.4 Postupy a technologie

Podkapitola se zaměřuje na popis předpokládaných technologií a nástrojů, které budou použity v rámci práce na tomto projektu. Jednotlivé sekce vždy krátce popisují vybrané technologie a slouží k vytvoření základního obrazu o těchto technologiích. Pro získání více detailnějších informací pak slouží uvedené odkazy na příslušné zdroje.

1.4.1 Keboola Connection

Keboola Connecion[6] je cloudový nástroj obsahující vše potřebné k provozu datového skladu. Skrze své webové rozhraní umožňuje uživateli provádět a konfigurovat jednotlivé kroky ETL procesu, tedy nahrávat data, transformovat je a dále je poskytovat dalším stranám pro jejich analytické zpracování tak, aby byla vizualizována uživateli pochopitelným způsobem. Data nahraná do Keboola Connection jsou fyzicky uložena na serverech Amazonu AWS. Kromě webového rozhraní poskytuje Keboola Connection také API, pomocí něhož lze provádět vše, co lze provést i přes toto webové rozhraní.

Pro obsluhu datového úložiště poskytuje Keboola Connection své Storage API. Data přitom v tomto úložišti nemusí pocházet všechna ze stejného zdroje ani formátu. Může jít o data uložená v různých druzích databází (například MySQL, PostgreSQL), ale také lze využít prakticky skoro libovolný soubor s daty ať už se jedná o excelovskou tabulku nebo jinak strukturovaná data. Data v úložišti jsou pak rozdělena do takzvaných bucketů, jež lze chápat jako oddíly obsahující jednotlivá data. Buckety jsou přitom předponami *in*, *out* a *sys* ještě dále děleny dle jejich účelu, kde bucket označený předponou *in* slouží jako bucket pro vstupní data, bucket s předponou *out* slouží pro odchozí data a bucket s předponou *sys* pro data obsahující systémové informace o konkrétním Keboola Projektu. Mimo dat samotných jsou zde ještě uloženy keboola tokeny, které slouží jako klíče pro přístup k úložišti externím aplikacím.

Pro zajištění všech kroků ETL procesu má Keboola Connection podporu pro takzvané extraktory, transformace a writery. Extraktory, jak již název napovídá, slouží jako nástroje pro extrakci dat z různých druhů široce dostupných datových zdrojů. Kromě extraktorů pro standardní databáze, jako jsou například MySQL, PostrgeSQL a MSSQL, existuje mnoho dalších extraktorů pro napojení přes REST API na mnoho dalších služeb, které mohou poskytnout data. V nabídce jsou extraktory pro služby jako například Google Drive, Dropbox či Twitter. Keboola Connection také umožňuje implementaci vlast-

ního extraktoru pro případy, kdy je potřeba se napojit na zatím nezahrnutý zdroj dat.

Poté, co jsou data úspěšně nahrána do úložiště, může dojít k jejich transformacím tak, aby se data dala úspěčně analyzovat. Pro všechny transformace přitom platí, že je nutno definovat, z jakého bucketu budou data do transformace vstupovat a naopak do jakého bucketu budou výsledná data vkládána. Data lze transformovat pomocí SQL dotazů, či k jejich transformaci může být využit jakýkoliv program, který bude součástí nahraného Docker image. Tento systém pak umožňuje provádět nad daty i velmi složité transformace, které by pomocí SQL dotazů byly jen těžko proveditelné.

Pro poslední část ETL procesu, tedy nahrání dat do systémů pro další zpracování (typicky vizualizaci), jsou použity již zmíněné writery. Keboola Connection nabízí několik předpřipravených writerů, ale i zde platí, že existuje možnost naimplementovat si vlastní writer, který splňuje specifické požadavky.

Kromě již zmíněných extraktorů, transformací a writerů existuje ještě možnost využití aplikací. Ty je možné, podobně jako v předchozích případech, kdykoliv nahrát do Kebooly v podobě Docker image. Ty následně mohou zpracovávat data libovolným potřebným způsobem a Keboola Connection tak poskytuje řešení pro práci s daty, které lze kdykoli snadno modifikovat a rozšířit.

1.4.2 Tableau

Platforma Tableau[7] poskytuje řadu aplikací zaměřených na datové průzkumy, analýzy a zejména pak jejich vizualizace. Tableau dělí své produkty na několik částí mezi něž patří patří Tableau Desktop, Tableau Server, Tableau Online a Tableau Mobile. V praxi se však jednotlivé aplikace dost překrývají a jde tedy hlavně o platformu, pro kterou jsou určeny. Pro pouhé zobrazení jednotlivých výstupů z aplikací je pak možné využít zdarma dostupnou aplikaci Tableau Reader. Platforma kromě přehledného rozhraní pro práci s daty, velkou podporou různých datových formátů (samozřejmostí je zpracování souborů CSV, Json, Microsoft Office a dalších nebo připojení na rozmanitou nabídku databází a serverů jako například MySQL, Amazon Redshift, Snowflake, Google Drive, Teradata a další) a širokou možností nástrojů pro vizualizaci, vyniká také rychlostí, kterou dokáže zpracovávat velké množství dat. To potvrzuje navíc i fakt, že Tableau od verze 10.5 přichází s vlastní novou technologií Hyper, o které tvrdí, že je až třikrát rychlejší pro zápisové příkazy a až pětkrát rychlejší pro klasické query dotazy, než dosavadní existující přístupy k datům[8]. To, že vývoj firmě Tableau není cizí, dosvědčuje i vývoj další vlastní úspěšné technologie, jíž lze využít v jejich produktech, VizQL. VizQL je zkratka pro Visual Query Language a její hlavní funkcí je převádět sérii drag-and-drop akcí na klasické datové dotazy a jejich následné grafické zobrazení[9]. VizQL tak přináší velmi intuitivní přívětivé prostředí i pro práci nad složitými a komplexními datasety. To, že analytické a vizualizační nástroje od firmy

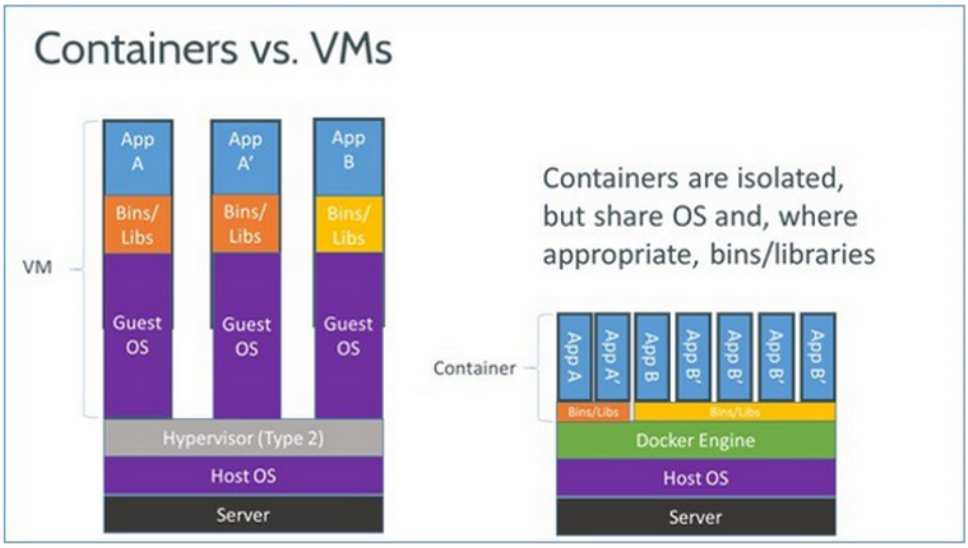
Tableau patří mezi současné nejlepší nástroje, dosvědčuje i umístění už šestým rokem v Leader quadrantu žebříčku Gartner v kategorii Analytics and Business Inteligence[10].

1.4.3 GoodData

Společnost GoodData byla založena Romanem Staňkem v roce 2007 a jejím cílem je poskytovat platformu pro business intelligence jako službu (Platform-as-a-Service)[11]. Výhodami tohoto přístupu jsou zejména škálovatelost, kdy uživatel využívá pouze tolik zdrojů, kolik skutečně využije, vždy aktuální použitá verze a snadná instalace, kdy průměrná doba zavedení produktu do praktického použití u zákazníka je 8 týdnů[12]. Uživateli je poskytnuta možnost si samotný produkt na webových stránkách zdarma vyzkoušet. Podobně jako Tableau i GoodData v současné době disponuje hned celou řadou konektorů na různé datové zdroje ovšem pro práci s nimi vyžaduje přesunutí těchto dat do vlastního cloudového řešení. Ze subjektivního pocitu disponuje oproti Tableau mírně lepšími nástroji pro datovou analytiku a samotnou práci s daty. Co se však týče samotné vizualizace výsledků, zde naopak vede Tableau, které přináší rychlejší a intuitivnější prostředí. Vizualizace je v nástroji GoodData stále na velice dobré úrovni, avšak její vytvoření nebo změny jsou přeci jen o něco komplikovanější.

1.4.4 Docker

Docker[13] lze zjednodušeně chápat jako určitý virtualizační nástroj. S rostoucí oblibou cloudových řešení a přesunem aplikací právě do cloudu lze narazit na problém, kdy aplikace potřebují určité nastavení různých proměnných. Docker pak představuje balíček (přesněji řečeno kontejner) do něhož lze tuto aplikaci vložit a specifické proměnné nastavit. To celé ve velmi minimalistickém provedení a tedy bez další větší režie, které by bylo potřeba u virtualizace[14]. Pro lepší představu se lze podívat na obrázek 1.1[15], kde je názorně vyobrazen rozdíl mezi virtualizací a Docker kontejnerem, a jak velice užitečný dokáže Docker být. Jak je možné vidět, celé to je opravdu velmi minimalistické, jednoduché a přesto to splňuje vše, co je potřeba. Každá aplikace má své knihovny a vše, co je potřeba pro její běh přímo u sebe. Vše ostatní již běží na sdíleném OS. Odpadává tak potřeba spousty prostředků, které byly potřeba pro virtualizaci OS. To vše je ještě umocněno v případě, kdy běží na jednom stroji několik aplikací zároveň (což je v dnešní době cloudu běžná záležitost). Že se nejedná o zanedbatelnou záležitost dokazuje i fakt, že k dnešnímu dni používá Docker více jak 3,5 miliónů aplikací[15] a přitáhl pozornost i takových společností jako je Microsoft nebo Oracle (viz články [16] a [17]).



Obrázek 1.1: Docker kontejnery vs. virtualizace

1.4.5 Travis CI

Pro usnadnění nasazování projektu je využita webová služba Travis CI[18], což je nástroj podporující průběžnou integraci projektů z veřejných repositářů GitHubu. Služba zajišťuje správné sestavení (build) výsledného projektu. Travis CI sám automaticky detekuje všechny změny provedené v repositáři a při každé takové změně se pokusí sestavit projekt. Následně služba notifikuje uživatele, zda bylo sestavení a nasazení úspěšné. Aby webová služba Travis CI fungovala správně, je nutné mít v hlavním adresáři repositáře konfigurační soubor *.travis.yml* ve formátu YAML[19]. Tento soubor specifikuje použitý programovací jazyk, nastavení prostředí pro sestavení projektu stejně tak jako další parametry, které jsou pro správné sestavení projektu potřeba.

1.4.6 Apache Maven

Apache Maven[20] je jedním z nejzákladnějších a nejznámějších nástrojů pro řízení projektů. Jedná se o nástroj pro řízení projektu využívající POM (Project Object Model), sadu standardů, životní cyklus projektu, systém pro správu závislostí a logiku pro vykonání cílů vložených pluginů v definovaných fázích jejich životního cyklu[21]. Mezi nesporné výhody Mavenu patří to, že vyžaduje, aby projekty dodržovaly předepsanou konzistentní strukturu, která zajišťuje snadnou modifikaci projektu a usnadňuje tak také vhledu do projektu dalším budoucím vývojářům. Ještě užitečnější je však pro svojí schopnost maximálně zjednodušené práce s projektovými závislostmi. Všechny závislosti lze jednoduše specifikovat v souboru POM.xml (a to včetně vyžadovaných verzí)

a již nechat na Mavenu jejich automatické stažení a integrování do projektu. POM.xml je XML dokument, který kromě specifikací jednotlivých závislostí, také například pojmenovává projekt a může sloužit i jako dokumentační nástroj či přizpůsobit chování pluginů nebo buildu. Apache Maven se stal natolik rozšířeným nástrojem, že je již dnes automaticky podporován celou řadou IDE.

1.4.7 IntelliJ IDEA IDE

Pro části procesu, které jsou implementovány v jazyce java je zvoleno IntelliJ IDEA IDE od společnosti JetBrains a to zejména pro intuitivní práci, jež toto prostředí poskytuje. Více informací o IntelliJ IDEA IDE lze dohledat na oficiálních webových stránkách [22].

1.5 Model požadavků

Následující podkapitola představuje model požadavků. Model požadavků v zásadě popisuje co všechno musí aplikace dělat a jaké služby musí umět poskytovat. Požadavky zároveň nepopisují to, jak budou jednotlivé funkce realizovány. Požadavky lze rozdělit na funkční a nefunkční.

1.5.1 Funkční požadavky

Funkční požadavky specifikují všechny hlavní funkce, které výsledná aplikace bude umět a poskytovat. Aplikace bude splňovat následující funkční požadavky:

1.5.1.1 Data rozdělená do clusterů

Aplikace musí podporovat data rozdělená do více clusterů. K jednotlivým clusterům se pak bude chovat jako k odděleným datům a segmentaci provede pro každý cluster zvlášť.

1.5.1.2 Více možných řešení RFM segmentace

Aplikace musí být navržena a implementována takovým způsobem, aby bylo možné volit mezi více metodami pro výpočet RFM segmentace. Také musí být navržena tak, aby bylo možné bez větších obtíží doimplementovat další metodu.

Vlastní hranice pro jednotlivé segmenty

Mezi základní implementovanou metodou segmentace bude již od začátku patřit metoda, při níž si uživatel sám zadá hranice pro jednotlivé segmenty.

Definování počtu segmentů

Kromě metody při níž si uživatel sám zadá hranice pro jednotlivé segmenty bude aplikace podporovat metodu, při níž si uživatel zadá výsledný počet segmentů. Aplikace si pak sama dopočítá hranice pro jednotlivé segmenty. Tento počet segmentů musí jít definovat pro každou dimenzi (z dimenzí RFM) zvlášť.

1.5.1.3 Vizualizace výsledků

Aplikace musí vhodným způsobem podporovat vizualizaci výsledků. Za vizualizaci výsledků přitom nemusí být zodpovědná přímo sama aplikace, ale může využít služeb třetích stran.

1.5.1.4 Ošetření vstupů

Aplikace musí vhodným způsobem zajistit ošetření vstupů při jejich načítání a případně zpravit uživatele aplikace o chybném vstupním nastavení.

1.5.1.5 Konfigurační soubor

Aplikace musí být schopná konfigurace parametrů RFM analýzy pomocí konfiguračního souboru.

1.5.1.6 Dostupnost v Keboola Connection

Aplikace musí být dostupná jako aplikace v prostředí ETL nástroje Keboola Connection.

1.5.2 Nefunkční požadavky

Nefunkční požadavky jsou pojmenovány podle toho, že nespecifikují funkce aplikace, nýbrž specifikují omezující podmínky a vlastnosti aplikace. Aplikace bude splňovat následující nefunkční požadavky:

1.5.2.1 Dostatečné množství paměti

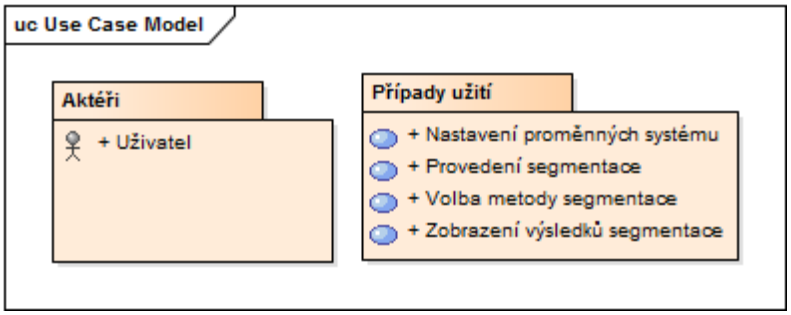
Aplikaci bude pro svůj běh poskytnuto dostatek operační paměti na to, aby mohla být všechna data nahrána do aplikace a zároveň s nimi bylo možno vhodným způsobem manipulovat. Vhodný způsob manipulování může představovat především řazení dat.

1.5.2.2 Podpora CSV souborů

Aplikace musí být připravena na formu CSV souborů jako primárního zdroje dat.

1.6 Model případů užití

Model případů užití (známější též z angličtiny jako Use Case Model), který je zobrazen na obrázku 1.2, je model, ve kterém je zachycena funkcionalita systému, stejně tak jako popis všech aktérů aplikace.



Obrázek 1.2: Model případů užití

1.6.1 Aktéři

Aplikace má následující aktéry.

1.6.1.1 Uživatel

Aplikace nemá žádné speciální dělení uživatelů dle jejich práv nebo jiných parametrů. Uživatelem se stává každá osoba, která chce provést RFM analýzu v prostředí Keboola Coonection. V nejčastějším případě to však bude programátor či spíše datový analytik.

1.6.2 Případy užití

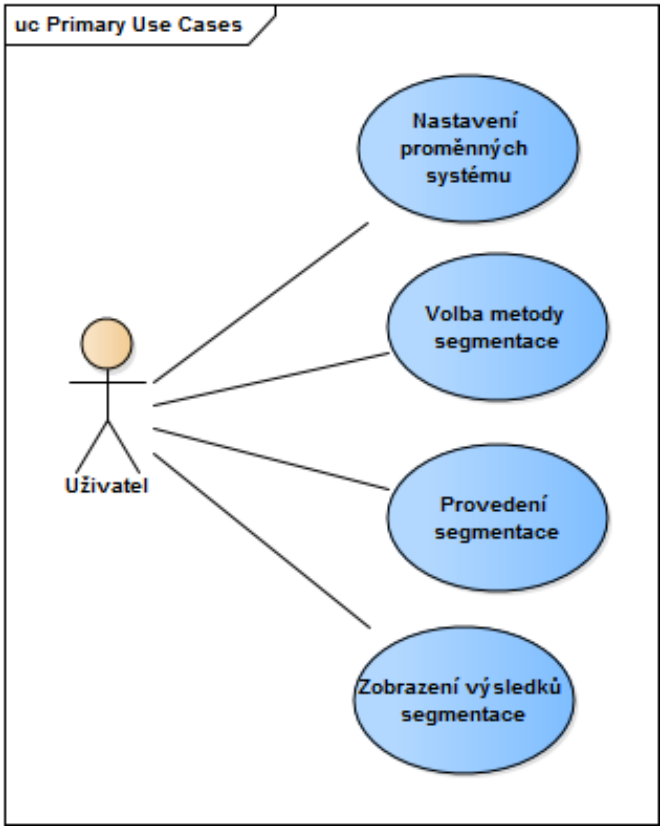
Případy užití, které je nutno brát v potaz, jsou zobrazeny na obrázku 1.3.

1.6.2.1 Nastavení proměnných aplikace

Aplikace musí umožnit uživateli nastavení vstupních proměnných potřebných pro správné provedení segmentace. Jedná se hlavně o správné nastavení cesty ke zdroji dat.

1.6.2.2 Volba metody segmentace

Aplikace uživateli umožní vybrat si metodu, jakým způsobem má být segmentace provedena. V základní verzi to bude volba mezi přímým specifikováním hranic jednotlivých segmentů v daných kategoriích (přičemž počet hranic pro



Obrázek 1.3: Případy užití

jednotlivé kategorie nemusí být stejný) a specifikováním počtu segmentů (i zde platí, že počet segmentů pro jednotlivé kategorie nemusí být stejný).

1.6.2.3 Provedení segmentace

Při správně vyplněných vstupních údajích provede aplikace segmentaci bez dalších dotazů na uživatele systému.

1.6.2.4 Zobrazení výsledků segmentace

Aplikace pomocí vybraného vizualizačního prostředí umožní uživateli seznámit se s výsledky RFM segmentace.

Kapitola 2

Návrh

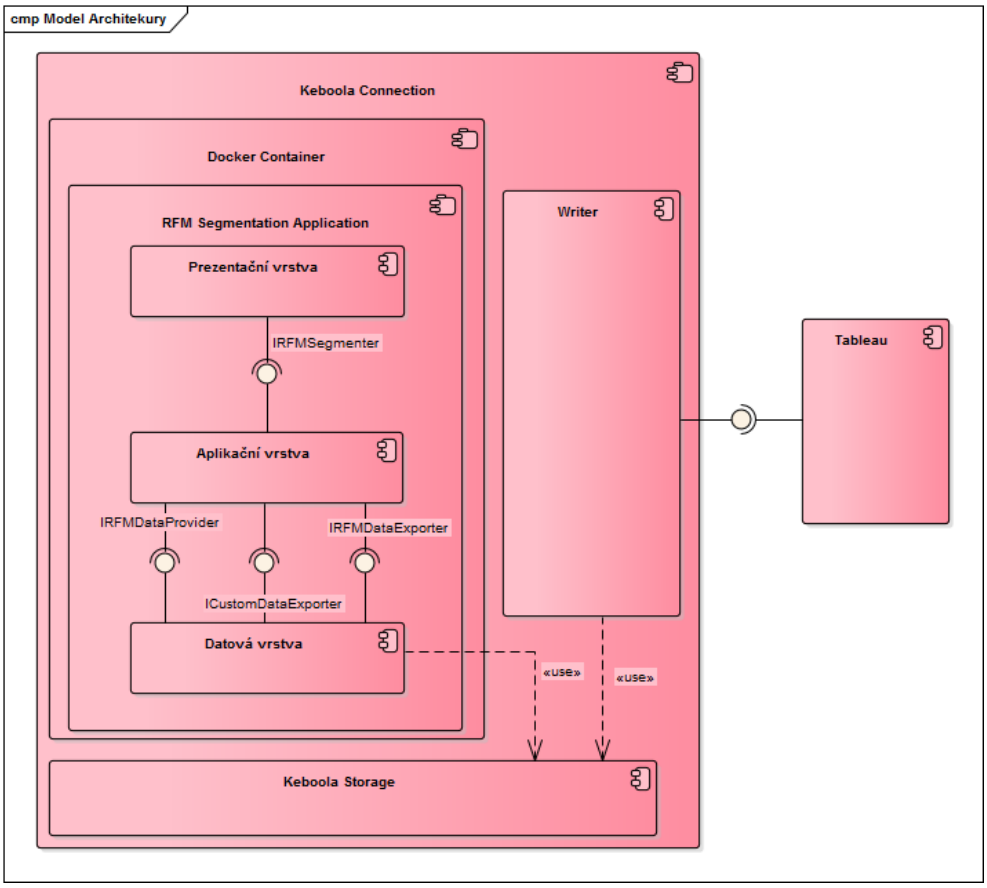
Následující kapitola přichází s návrhem konkrétního řešení, které splňuje všechny požadavky specifikované v předchozí části práce. Pomocí několika modelů jsou zde rozebrány klíčové problémy a navrhnuto funkční řešení využívající specifikované technologie. Lze zde najít model architektury zabývající se architekturou celé aplikace a komunikace mezi jejími důležitými dílčími částmi a také návrhový model tříd popisující všechny důležité třídy a metody nezbytné pro správné fungování aplikace. Kromě zmíněného modelu architektury a návrhového modelu tříd je na konci kapitoly také prezentován model komunikace zabývající se fungováním nejdůležitějších metod aplikace a jejich vzájemné komunikace napříč třídami.

2.1 Architektura systému

Architekturu celého systému lze vidět na obrázku 2.1. Celý proces RFM analýzy je rozdělen do dvou větších částí. V prostředí Keboola Connection je nejdříve provedena celá RFM segmentace o kterou se stará komponenta (aplikace) *RFM Segmentation Application*, která je pro správný běh (jak to vyžaduje prostředí Keboola Connection) ještě zabalena uvnitř Docker kontejneru, který zajistí správné nastavení proměnných systému a potřebných knihoven. Následně jsou zpracovaná data pomocí writeru nahrána do vizualizačního prostředí. Ze zvažovaných vizualizačních prostředí bylo v tomto případě vybráno vizualizační prostředí Tableau.

2.1.1 RFM Segmentation Application

Pro aplikaci RFM Segmentation Application je zvolena třívrstvá architektura, přičemž komunikaci mezi jednotlivými vrstvami zajišťují rozhraní (interface). Díky přesné definici jednotlivých rozhraní tak lze implementaci libovolné vrstvy kdykoli zaměnit za alternativní implementaci splňující pouze dané podmínky rozhraní. To se může stát například ve chvíli, změní-li se po-



Obrázek 2.1: Model Architektury

žadavky na uživatelské rozhraní nebo podobu datového úložiště. Podrobnější popis jednotlivých rozhraní aplikace je vyobrazen na obrázku 2.2.

2.1.1.1 Prezentační vrstva

Prezentační vrstva obsahuje třídy, které zajišťují komunikaci s uživatelem. V případě prezenční vrstvy *RFM Segmentation Application* se vrstva stará pouze o správné zpracování uživatelova vstupu, jelikož uživatelské rozhraní obstarává již prostředí Keboola Connection. Kdyby však existoval požadavek na běh aplikace mimo prostředí Keboola Connection, tak lze snadno rozšířit či nahradit právě tuto vrstvu o třídy potřebné k zobrazení a obsloužení uživatelského rozhraní. Uživatelův vstup je dále (pomocí rozhraní *IRFMSegmenter*) předán aplikační vrstvě aplikace.

2.1.1.2 Aplikční vrstva

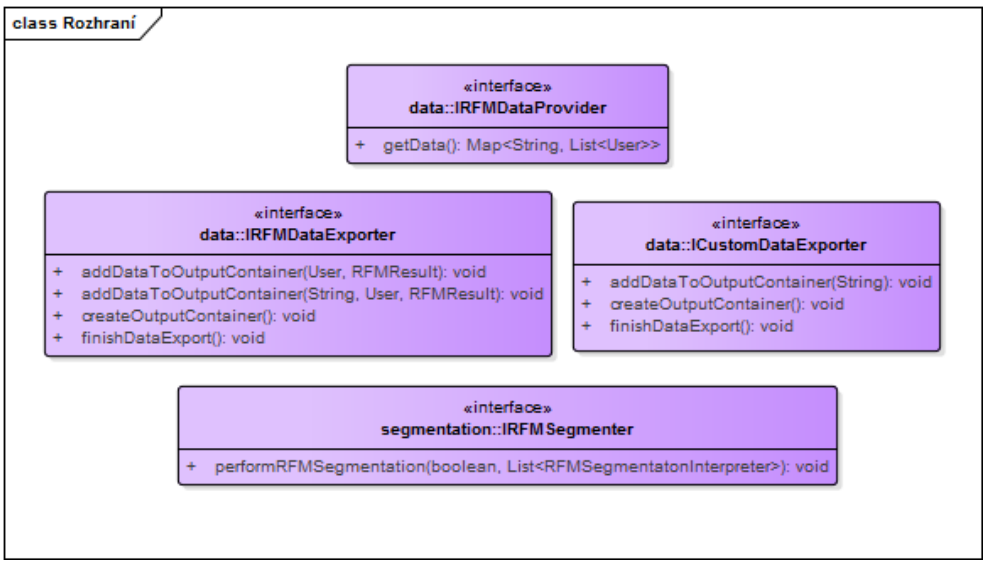
Aplikační vrstva obsahuje třídy, které obstarávají veškerou logiku systému. To znamená, že se starají o to, jak systém funguje a co je jeho činností. V případě aplikační vrstvy *RFM Segmentation Application* tedy zajišťují správný průběh RFM segmentace.

2.1.1.3 Datová vrstva

Datová vrstva obsahuje třídy, které mají na starost ukládání a načítání dat. V případě datové vrstvy *RFM Segmentation Application* jsou zde zastoupeny třídy jak pro načtení, tak i následné uložení dat zpět do úložiště Keboola Storage.

2.1.1.4 Rozhraní

Sekce podrobněji popisuje jednotlivé rozhraní aplikace, které lze vidět na obrázku 2.2.



Obrázek 2.2: Rozhraní RFM Segmentation Application

IRFMSegmenter

Rozhraní *IRFMSegmenter* zajišťuje, aby data předávaná aplikační vrstvě aplikace byla správně zpracována pomocí metody performRFMSegmentation. Zároveň informuje daný segmenter, zda jsou data rozdělena do clusterů či nikoliv.

IRFMDataProvider

Rozhraní *IRFMDataProvider* zajišťuje, aby byla data načítána pomocí stejné metody *getData*, ať už bude použit jakýkoli zdroj dat. Zároveň zajišťuje, aby byla data rozdělena dle clusterů a aby jednotlivé entity (zákazníci) již měli napočítané vlastnosti recency, frequency a monetary.

IRFMDataExporter

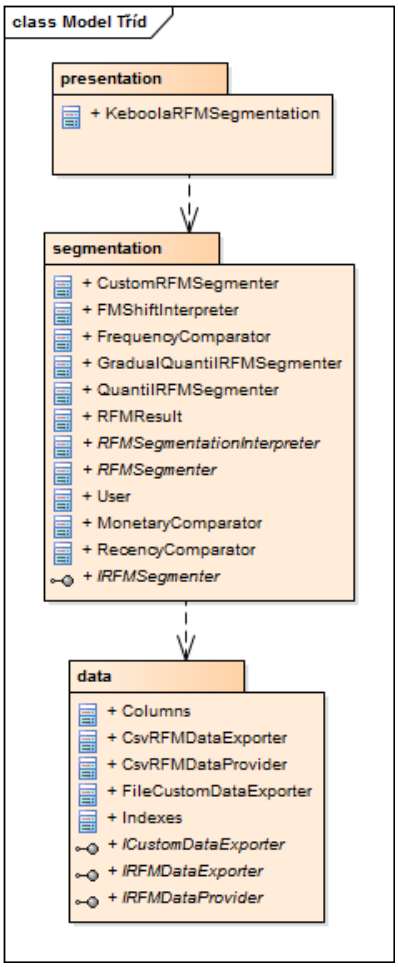
Rozhraní *IRFMDataExporter* zajišťuje jednotný přístup k exportu výsledných dat týkajících se segmentace z aplikace. Z důvodu optimalizace pro větší množství dat a s tím související podpory postupného exportu výsledků je vyžadována implementace tří metod. První z nich je metoda *createOutputContainer*, která zajistí vytvoření kontejneru, který bude obsahovat výsledná data. Může jít například o soubor v souborovém systému. Dalším příkladem by bylo například vytvoření SQL tabulky. Druhou metodou je metoda *addDataToOutputContainer* jež slouží pro zapsání jednotlivých dílčích výsledků segmentace. Tato metoda je přetížena tak, aby podporovala jednak ukládání s uvedeným clusterem, ale aby dokázala uložit data i bez clusteru. Poslední vyžadovanou metodou je pak metoda *fininishDataExport*. Ta ukončí práci s úložištěm. Například tedy zajistí ukončení práce se souborovým systémem nebo konec komunikace s SQL strojem.

ICustomDataExporter

Rozhraní *ICustomDataExporter* je zde zastoupeno z důvodu umožnění různých výstupů, které mohou poskytnout různé interpretace RFM segmentace. Formát těchto výstupů nelze dopředu odhadnout, a je proto k dispozici toto obecné rozhraní. Stejně jako v případě rozhraní *IRFMDataExporter* je zachován přístup s využitím tří metod pro vytvoření datového kontejneru v úložišti, zapisování dat a ukončení práce s úložištěm.

2.2 Návrhový model tříd

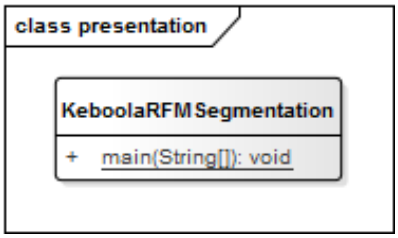
Tato kapitola se zabývá statickým popisem systému a také podrobněji popisuje jednotlivé balíčky, třídy a rozhraní realizující aplikaci. Celý model je přizpůsoben programovacímu jazyku java následkem čehož balíčky, třídy a rozhraní odpovídají balíčkům, třídám a rozhraním v tomto programovacím jazyce. Návrhový model tříd lze vidět na obrázku 2.3. Celá aplikace je rozdělena do tří základních balíčků, které lze promítnout na jednotlivé vrstvy třívrstvé architektury. Z výše uvedeného vyplývá, že balíček *presentation* odpovídá prezentační vrstvě, balíček *segmentation* odpovídá aplikační vrstvě a balíček *data* odpovídá datové vrstvě aplikace.



Obrázek 2.3: Návrhový model tříd

2.2.1 Balíček presentation

Balíček *presentation*, který je vyobrazen na obrázku 2.4, představuje prezenční



Obrázek 2.4: Balíček presentation

vrstvu aplikace a jsou v něm proto zastoupeny třídy starající se o tuto činnost. Jelikož celá aplikace starající se o segmentaci nemá vlastní uživatelské rozhraní (to zajistí prostředí Keboola Connection), vystačí si pouze s jednou třídou starající se o zpracování a validaci uživatelova vstupu.

2.2.1.1 KeboolaRFMSegmentation

Jedná se o třídu starající se o zpracování celého uživatelova vstupu. Třída má jen jedinou statickou metodu *main*, která slouží jako vstupní bod aplikace a zároveň přebere od uživatele cestu ke konfiguračnímu souboru. Třída má rovněž za úkol celý tento konfigurační soubor zpracovat a předat informace v něm obsažené dalším třídám.

2.2.2 Balíček segmentation

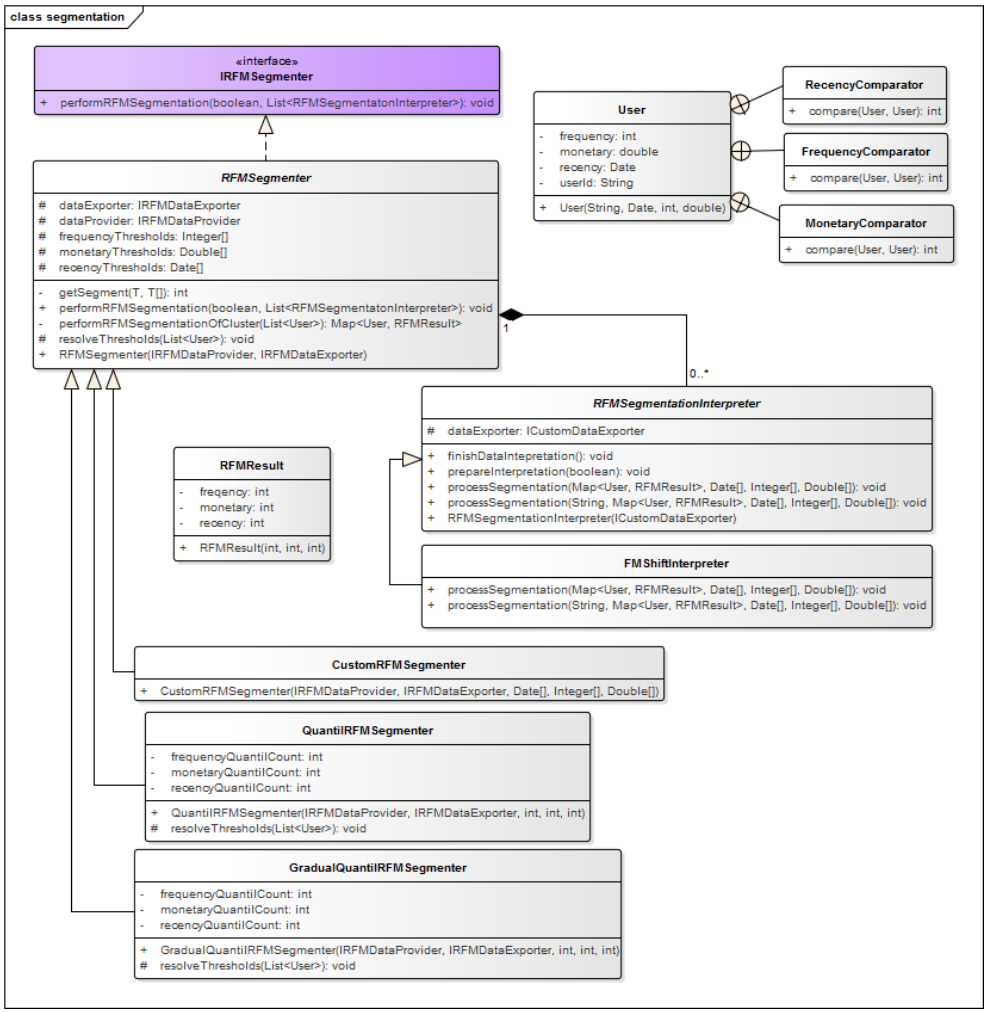
Třídy balíčku *segmentation* lze vidět na obrázku 2.5. Balíček představuje aplikační vrstvu aplikace a jsou v něm obsaženy všechny třídy starající se o tuto činnost.

2.2.2.1 RFMSegmenter

RFMSegmenter je abstraktní třída, tvořící současné jádro celé aplikace, od které dědí všechny současné třídy implementující konkrétní formu RFM segmentace. Třída implementuje rozhraní *IRFMSegmenter*, takže celou segmentaci (včetně načtení dat) provede po zavolání metody *performRFMSefmentation*. Aby toho byla schopna, požaduje v konstruktoru zdroj dat (*IRFMDataProvider*) a taktéž datový výstup (*IRFMDataExporter*). Datový výstup je požadován již zde z důvodů možné nadměrné velikosti dat. Metoda, je-li to možné, totiž ukládá již zpracované výsledky už během segmentace, aby zmenšila požadavky na velikost RAM paměti. Jelikož jsou data zpracovávána po jednotlivých clusterech, tak v rámci metody *performRFMSefmentation* je využita ještě metoda *performRFMSefmentationOfCluster*, která provede samotnou segmentaci jednotlivých clusterů. Aby toho byla schopná, je k dispozici ještě metoda *resolveThresholds*, která je přepisována potomky této třídy a která má za úkol stanovení hranic mezi jednotlivými segmenty. Pro přiřazení uživatele do konkrétního segmentu pak slouží generická metoda *getSegment*.

2.2.2.2 QuantilRFMSegmenter

QuantilRFMSegmenter představuje třídu, která dědí od abstraktní třídy *RFMSegmenter*. Třída přepisuje zděděnou metodu *resolveThresholds* pomocí které specifikuje hranice sloužící pro rozdělení uživatelů do segmentů. Tyto hranice má za úkol stanovit pokud možno rovnoměrně, aby bylo možné rozdělit uživatele na předem specifikovaný počet quantilů. Rozdělení musí proběhnout jednotlivě pro všechny dimenze (recency, frequency, monetary).



Obrázek 2.5: Balíček segmentation

2.2.2.3 CustomRFMSegmenter

Třída *CustomRFMSegmenter*, která rovněž dědí od abstraktní třídy *RFMSegmenter*, se uplatní pro segmentaci v případě požadavku na vlastní stanovení hranic pro jednotlivé segmenty. Podobně jako v případě třídy *QuantilRFMSegmenter* i zde platí, že lze pro jednotlivé dimenze zvolit různý počet segmentů.

2.2.2.4 GradualQuantilRFMSegmenter

Třída *GradualQuantilRFMSegmenter* také dědí od abstraktní třídy *RFMSegmenter*. Stejně jako v případě třídy *QuantilRFMSegmenter* i zde je přepsána zděděná metoda *resolveThresholds* pomocí které jsou specifikovány hranice

sloužící pro rozdělení uživatelů do segmentů. Tyto hranice by však v tomto případě měly brát větší ohled na jednotlivé uživatele a stanovovat je postupným rozdělováním uživatelů, aby vznikl specifikovaný počet quantilů. I v tomto případě musí rozdělení proběhnout jednotlivě pro všechny dimenze (recency, frequency, monetary).

2.2.2.5 RFMSegmentationInterpreter

Abstraktní třída *RFMSegmentationInterpreter* slouží ke složitějšímu vhledu do výsledků RFM segmentace. Jelikož vizualizace si poradí pouze s jednoduchým zobrazením výsledků, ale pro složitější interpretaci již potřebuje dopočtení dalších dat, existuje možnost specifikovat interpretaci, která může provést tyto další složitější výpočty. Tato třída poskytuje základní podporu právě pro tuto funkci. Jelikož je zapotřebí výsledné interpretace uložit, aby mohly posloužit jako vstup do vizualizačního nástroje, je k dispozici dvojice metod *prepareDataInterpretation* a *finishDataInterpretation* pomocí nichž je toto možné provést. Krom toho lze také využít tyto metody k dalším činnostem, které je potřeba provést před a po samotné interpretaci. K samotné interpretaci pak slouží metoda *processSegmentation*, která má za úkol právě zpracování jednotlivých výsledků RFM segmentace. Podobně jako v případě třídy *RFMSegmenter* je tato metoda přetížena, aby podporovala jak data rozdělená do clusterů, tak i data bez clusterů.

2.2.2.6 FMShiftInterpreter

V současné době jediným navrženým a implementovaným interpreterem je *FMShiftInterpreter*, který má za úkol poskytovat informaci o obchodním potenciálu vyplývajícím ze zlepšování vlastností frequency a monetary.

2.2.2.7 RFMResult

Tato třída slouží pro udržení výsledků segmentace a to od bodu jejich zjištění až do doby, než dojde k jejich uložení pomocí *IRFMDataExporteru*.

2.2.2.8 User

Třída *User* slouží pro udržení vstupních dat o zákazníkovi po dobu výpočtu segmentace. Při načítání dat jsou zákazníkovi dopočítány jednotlivé vlastnosti sloužící pro segmentaci (recency, frequency, monetary), aby se již během výpočtu nemusely dopočítávat. Během výpočtu hranic segmentů se totiž zákazníci opakovaně porovnávají mezi sebou a opakovaný výpočet jednotlivých vlastností by pak zdržoval výpočet samotné segmentace. *IDataExporteru*.

2.2.2.9 RecencyComparator

Tato pomocná třída pro potřeby řazení. Třída umí správně porovnat zákazníky na základě jejich vypočítané vlastnosti recency.

2.2.2.10 FrequencyComparator

Tato pomocná třída pro potřeby řazení. Třída umí správně porovnat zákazníky na základě jejich vypočítané vlastnosti frequency.

2.2.2.11 MonetaryComparator

Tato pomocná třída pro potřeby řazení. Třída umí správně porovnat zákazníky na základě jejich vypočítané vlastnosti monetary.

2.2.3 Balíček data

Třídy balíčku *data* lze vidět na obrázku 2.6. Balíček představuje datovou vrstvu aplikace a jsou v něm obsaženy všechny třídy starající se o tuto činnost.

2.2.3.1 CsvRFMDataProvider

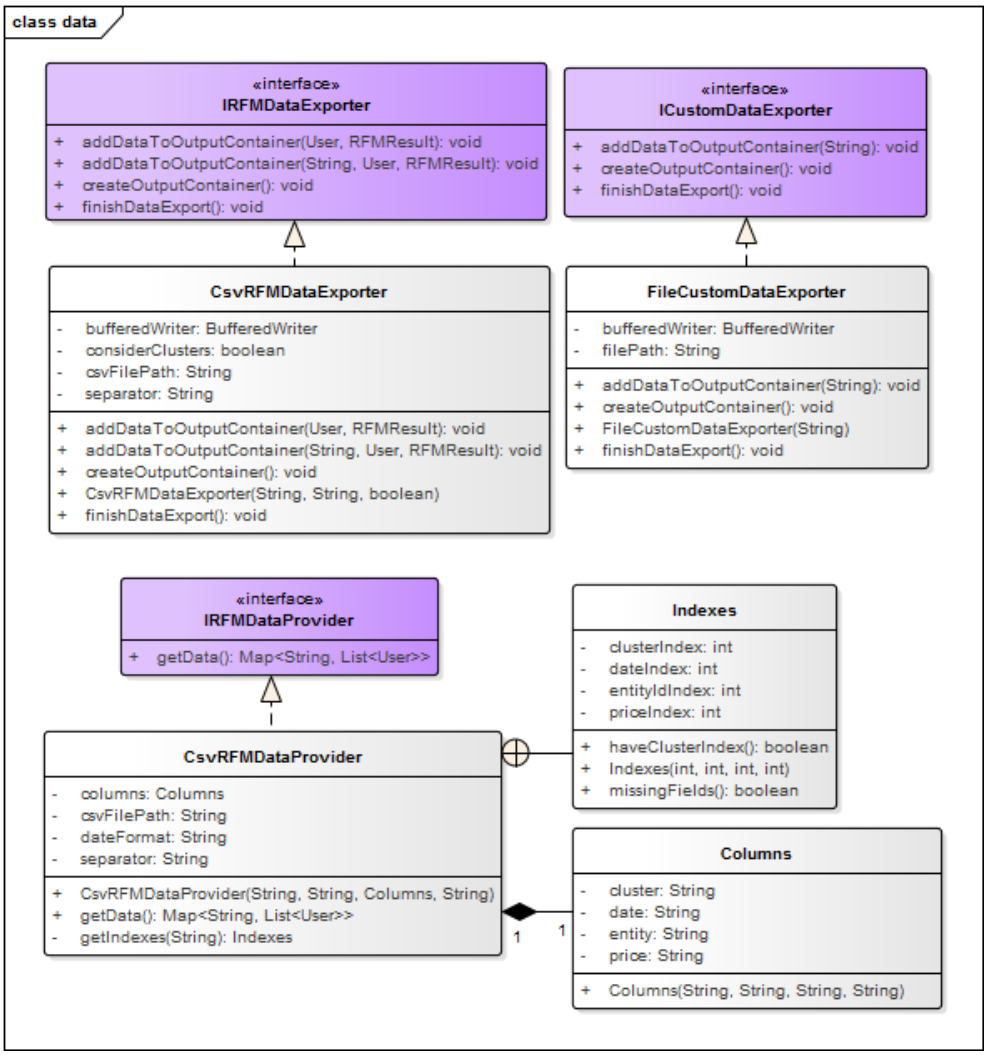
Jelikož v nástroji Keboola Connection jsou námi data ukládána ve formě CSV souborů, je k dispozici třída, která implementuje metodu *getData* z rozhraní *IRFMDataProvider* a jejímž zdrojem dat je právě CSV soubor. Na výstupu metody *getData* jsou pak již standardně seznamy uživatelů s napočítanými vlastnostmi recency, frequency a monetary rozdělené dle clusterů. Aby toho byla třída *CsvEFMDataProvider* schopná, požaduje v konstruktoru specifikaci separátoru (oddělovače jednotlivých dat, nejčastěji čárka nebo středník), cestu k souboru s daty, formát v jakém je uloženo datum (případně i čas) a také specifikaci názvů jednotlivých sloupečků s daty.

2.2.3.2 Columns

V rámci snížení počtu parametrů konstruktoru *CsvRFMDataProvider* (praktické důvody se lze dočíst například zde [23]) slouží tato třída k předání názvů jednotlivých důležitých sloupečků, které jsou poskytnuty uživatelem na vstupu aplikace. Je tak dosaženo přesnějšího určení, který sloupeček má jaký název, než kdyby názvy těchto sloupečků byly předány pomocí jednorozměrného pole nebo samostatných stringů.

2.2.3.3 Indexes

Jedná se o pomocnou třídu, která slouží k uchování informace o nalezených pozicích jednotlivých důležitých sloupečků. Třída má metodu *haveClusterIndex* sloužící k ověření (nebo zamítnutí) předpokladu, že byl nalezen sloupeček,



Obrázek 2.6: Balíček data

podle něhož mají být uživatelé rozděleni do clusterů. Druhou metodou, kterou třída *Indexes* disponuje, je pak metoda *missingClusters*, která zkontroluje, zda byly nalezeny všechny povinné sloupečky.

2.2.3.4 CsvRFMDataExporter

Stejně jako v případě *CsvRFMDataProvideru* je potřeba výsledek segmentace zase uložit do souboru typu CSV, aby s ním bylo prostředí Keboola Connection schopné dále pracovat. Třída implementuje rozhraní *IRFMDataExporter*, a tak má všechny potřebné metody, aby toho byla schopná. Při volání metody *createOutputContainer* je vytvořen soubor CSV. Následně jsou

do něj zapsány jednotlivé řádky s výsledky pomocí metody *addDataToOutputContainer* a nakonec je práce se souborovým systémem ukončena pomocí metody *finishDataExport*. Podobně jako v případě *CsvRFMDataProvideru* i zde je potřeba v konstruktoru specifikovat kromě názvu výsledného souboru i separátor, jímž budou jednotlivé sloupečky dat odděleny.

2.2.3.5 FileCustomDataExporter

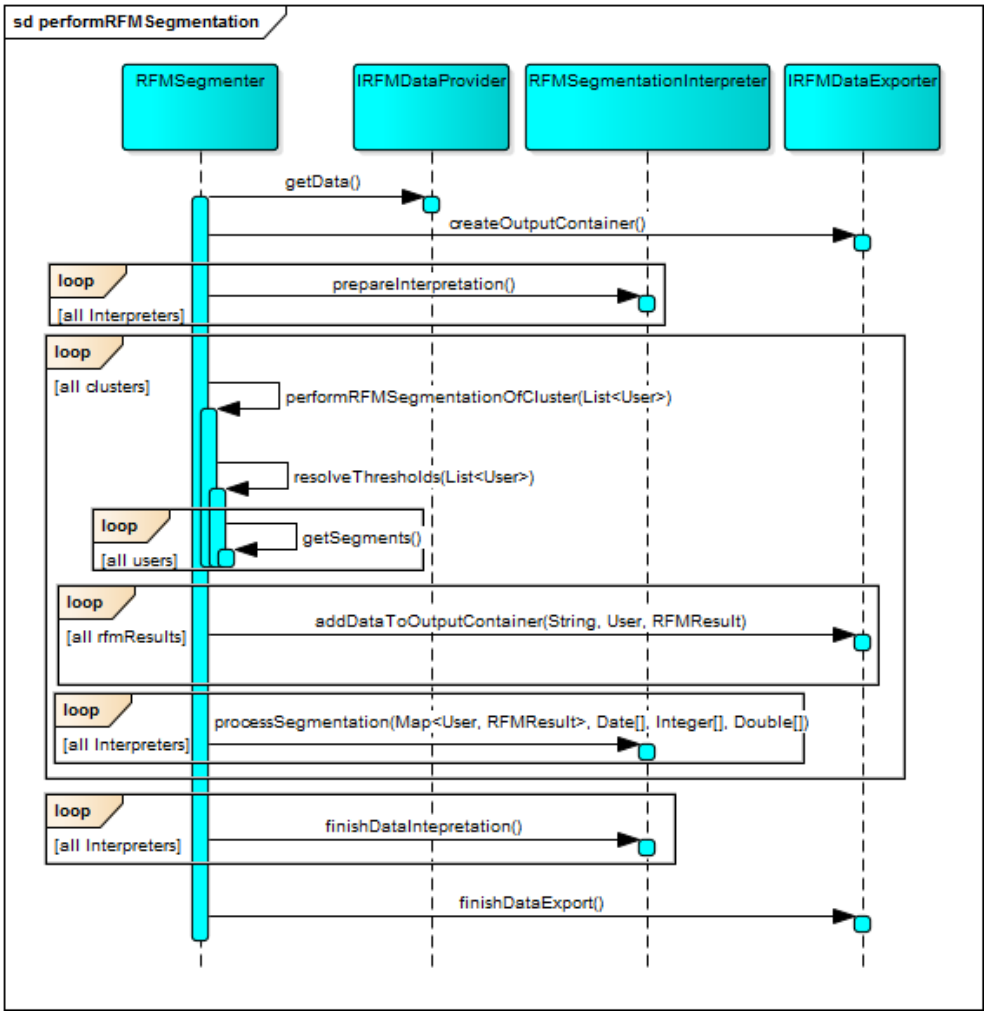
Podobně jako v případě exportu dat s využitím *CsvRFMDataExporter* je v případě požadavku na interpretaci potřeba výslednou interpretaci také zapsat do CSV souboru a nahrát do úložiště. *FileCustomDataExporter* v podstatě zachovává funkcionalitu známou z *CsvRFMDataExporteru* a *CsvRFMDataProvideru*. V tomto případě však třída programátorovi neulehčuje práci se samotným formátováním textu, jelikož nemá jak požadovaný formát obsáhnout. Nechává tak samotné formátování na přímo uživateli a pouze se stará o to, aby data byla zapsána ve formě souboru na zvoleném místě.

2.3 Model komunikace

Tato podkapitola se podrobněji zabývá fungováním nejdůležitějších metod aplikace, a to pak zejména z hlediska komunikace mezi jednotlivými třídami. Model rozebírá, jaké konkrétní kroky nastanou a jak spolu třídy komunikují v případě volání zvolených metod, čímž čtenáři umožní podrobnější vhled do problematiky fungování jednotlivých metod a poskytne tak lepší pochopení principu celé aplikace.

2.3.1 performRFMSegmentation

Popis činnosti a komunikaci tříd napříč aplikací, která nastává po zavolání metody *performRFMSegmentation* třídy *RFMSegmenter*, lze spatřit na obrázku 2.7. Jak je možné vypozorovat, celý proces segmentace proběhne poměrně jednoduše. Nejprve je zažádáno o data. Po jejich obdržení jsou vytvořeny kontejnery pro výsledná data, která budou za chvíli vypočítána a provedeny přípravy na interpretace těchto dat. Pak je již po jednotlivých clusterech provedena samotná RFM segmentace. V této fázi jsou vypočítány hranice mezi jednotlivými segmenty a následně jsou všichni zákazníci přiřazeni do patřičných segmentů. Vždy po segmentaci clusteru dojde za pomocí metody *addDataToOutputContainer* k uložení všech výsledků do úložiště, aby se data nemusela držet v RAM paměti. Po uložení výsledků do úložiště jsou tyto výsledky dále předány jednotlivým interpreterům ke zpracování jejich metodou *processSegmentation*. V rámci metod jednotlivých interpreterů také dochází k ukládání výsledků, které vypočítaly, do úložiště. Toto není v rámci zachování přehlednosti a kompaktnosti do diagramu zakresleno. Po spočítání segmentace pro všechny clustery metoda uzavře všechny spojení s úložištěm.



Obrázek 2.7: performRFMSegmentation

2.4 Velikost RAM paměti

Ačkoli v nefunkčních požadavcích na aplikaci (viz sekce 1.5.2) je uvedeno, že aplikace bude mít dostatek paměti pro svůj běh, přináší následující kapitola několik pohledů na to, jak se případně s nedostatkem RAM paměti vypořádat.

První řešení, které lze i v současné době provést, je následující. Jsou-li data rozdělena do clusterů, je sice možné zvolit provedení RFM analýzy nad celým takovým datasetem a očekávat provedení segmentace zvlášť pro jednotlivé clustery. Jelikož však aplikace nevyžaduje a ve svém důsledku tedy ani nepředpokládá, že by tato data byla seřazena tak, aby data pro jednotlivé clustery následovala po sobě (typicky data představují transakce rozdělené

do různých kategorií (clusterů) seřazená dle času provedení), musí je načíst všechna najednou. Mohla by sice pokaždé načíst pouze data z jednoho clusteru, to by však vedlo k opakovanému dotazování se na zdroj a v případě čtení dat ze souboru typu CSV by zde docházelo k náročnému opakovanému čtení dat. Řešením je tak provést rozdělení těchto dat a místo jednoho běhu aplikace s daty rozdělenými do clusterů provést n samostatných běhů aplikace.

Dalším řešením pro případy, kdy data nejsou rozdělená do clusterů, které by však vyžadovalo podporu ze strany samotné aplikace a v současné době není k dispozici, by bylo provést segmentaci jednotlivých dimenzí postupně a vždy načítat pouze data potřebná pro danou dimenzi. Jelikož je potřeba provést segmentaci tří dimenzí, tak by se v tomto případě velikost potřebné paměti mohla omezit až zhruba na třetinu (v reálném případě kvůli režii potřebné k řazení a dalším činnostem spíše na polovinu). Zároveň s tím by zde však musel být zastoupen požadavek na čtení dat třikrát místo pouhého jednoho běhu čtení dat.

V případě, že by první dvě řešení stále ještě neposkytovala dostatek prostoru pro výpočet, bylo by možné využít faktu, že data k segmentaci je potřeba seřadit. Nejdříve by se tak musel spočítat počet uživatelů, kteří transakce provedli a následně by se dala načítat pouze určitá podskupina potřebná k definování hranice. Celá situace by v tomto případě přinášela ještě několik dalších vlastních problémů (jako například to, že některé vlastnosti jsou dopočítávány), které by bylo potřeba vyřešit, to však není součástí cílů této práce.

Kapitola 3

Realizace

Kapitola realizace se zabývá samotným uvedením předchozích kapitol do praxe. Nejprve je zde uvedena implementace všech důležitých částí. Ta je rozdělena na popis vytvoření vhodného Dockrového image pro nasazení do prostředí Keboola Connection, samotný popis konfigurace pro automatické nasazení a popis nejdůležitějších dílčích částí implementace aplikace RFM Segmentation Application. Po seznámení čtenáře s touto implementační částí je představena část týkající se vizualizace výsledků a možné způsoby aplikování RFM analýzy v praxi.

3.1 Implementace

3.1.1 Docker

Pro správný běh vytvořené aplikace *RFM Segmentation Application* v prostředí Keboola Connection je potřeba jejího zabalení do Docker image. Existuje množství postupů na vytvoření takového image. Naneštěstí však ne všechny jsou kvalitní a následování takových návodů velmi často vede k vytvoření nekvalitního image, který zabírá zbytečně moc místa a jeho sestavení zabere zbytečně moc času [24]. Pro běh javové aplikace není potřeba vytvářet celý Docker image úplně od začátku, ale lze využít základního image z oficiálního repozitáře Dockeru. Jelikož oficiální java image je nyní ve stavu deprecated [25], vychází Docker image pro tento projekt ze základního Docker image openjdk, který je rovněž zdarma dostupný z oficiálního repozitáře Dockeru [26]. Projekt využívá tohoto základního image ve verzi 8.

```
1 FROM openjdk:8
2 COPY . /src/
3 WORKDIR /data/
4 CMD ["java", "-XX:MaxRAM=256M", "-jar", "/src/
    KeboolaRFMSegmentation.jar","config.json"]
```

Listing 3.1: Dockerfile pro RFM Segmentation Application

3.1.2 Automatické sestavení a nasazení

Nasazení dockrového image do Keboola Connection je provedeno pomocí webové služby Travis CI. Aby byla služba Travis CI schopna automatického sestavení a následného nasazení projektu, je v kořenovém adresáři projektu na githubu umístěn konfigurační soubor *travis.yml*. Pomocí tohoto souboru je nejprve definováno, že z široké nabídky jazyků, jež lze použít, je použitým jazykem scriptovací jazyk *bash* a je zapotřebí služby Docker, která zajistí sestavení požadovaného Docker image. Dále definuje, že nasazení bude prováděno pokaždé při vydání nové verze (tagu), přičemž samotné nasazení provede pomocí scriptu, který je v samostatném souboru *deploy.sh*.

Script *deploy.sh* sloužící k nasazení projektu používá pro komunikaci s *Keboola Developer Portal API*[27] keboolí *Developer Portal CLI tool*[28]. Nástroj samotný je přitom poskytován jako Docker image. Skript pro svůj správný běh používá několik enviromentálních proměnných. Jsou to KBC_DEVELOPERP ORTAL_APP pro id komponenty, KBC_DEVELOPERPORTAL_USERNA ME a KBC_DEVELOPERPORTAL_PASSWORD pro přístupové údaje a KBC_DEVELOPERPORTAL_VENDOR pro specifikování vendora komponenty. Všechny tyto proměnné jsou nastaveny skrze webové rozhraní služby Travis CI.

Více o celém procesu sestavení a nasazení aplikace, lze nalézt v dokumentaci na webových stránkách Keboola Connection odkazující na toto téma dostupné z [29]. Zde jsou pak také i podrobněji popsány příklady jednotlivých souborů *travis.yml* a *deploy.sh*.

Ukázky nasazené aplikace si lze prohlédnout na obrázcích v příloze B této práce.

3.1.3 RFM Segmentation Application

Následující podkapitoly se zabývají popisem nejdůležitějších částí implementace aplikace, která je zodpovědná za samotnou RFM segmentaci. Jednotlivé podkapitoly rozebírají algoritmickou logiku a principy fungování nejdůležitějších dílčích částí implementace.

3.1.3.1 Zpracování vstupních parametrů

Aplikace *RFM Segmentation Application* napsaná v javě, která běží v Docker kontejneru v Keboola Connection, se nejprve pomocí JSON parseru (je použita knihovna GSON[30] od společnosti Google, která byla vybrána napříč dalšími knihovnami[31], hlavně pro svoje jednoduché a praktické použití) pokusí z konfiguračního souboru *config.json* načít všechny vstupní parametry vyžadované pro správný chod programu. Tento konfigurační soubor je automaticky generován pomocí uživatelského rozhraní v Keboola Connection, které je generováno za pomoci JSON konfiguračního schématu. Více o možnostech projektu JSON Schema se lze dočíst na jeho webových stránkách [32]. Bohužel k tomu, aby

došlo ke správnému zobrazení všech parametrů v uživatelském rozhraní Keboola Connection, musí být všechny parametry označeny jako *required*[33]. Aby bylo možné mít volitelné parametry, lze nastavit volitelným parametrům defaultní hodnotu. Nelze však pak využít úplně všech možností validace, které využití JSON schématu nabízí a načtení a ověření správnosti vstupních parametrů tak musí zajistit v některých případech až sama aplikace. Samotné ověření je naneštěstí část kódu, kterou nelze moc zoptimalizovat a je provedeno pomocí série *try, catch* a *if, else* klauzulí. To je nutné také z důvodu, aby mohl být uživatel upozorněn smysluplnými chybovými hláškami, kterou část vstupu zadal chybně.

```
1  if (args.length < 1) {
2      System.err.println("Path to config must be specified");
3      System.exit(1);
4  }
5  JsonObject config = null;
6
7  try {
8      JsonReader jsonReader = new JsonReader(new FileReader(args[0]))
       ;
9      JsonParser jsonParser = new JsonParser();
10     config = jsonParser.parse(jsonReader).getAsJsonObject();
11     jsonReader.close();
12 } catch (Exception ex) {
13     System.err.println("System could not parse the config file
       properly.\n" + ex.getMessage());
14     System.exit(1);
15 }
16
17 if (config.getAsJsonPrimitive("method") == null) {
18     System.err.println("Segmentation method must be specified");
19     System.exit(1);
20 }
21 String method = config.getAsJsonPrimitive("method").getAsString();
22 ...
```

Listing 3.2: Ukázka parsování vstupu

3.1.3.2 Načtení dat

Načítání dat před samotnou segmentací probíhá jednoduchým způsobem z poskytnutých CSV souborů. Na vstupu je očekáván vždy jeden CSV soubor, jež v sobě obsahuje všechny transakce, které budou brány v potaz pro zákaznické historie. Specifické pořadí sloupečků s daty není požadováno. Nicméně právě z tohoto důvodu musí první řádek tohoto souboru specifikovat názvy jednotlivých sloupečků, aby je bylo možné správně přiřadit. Každý soubor musí obsahovat sloupečky obsahující zákazníkovo id, datum (a případně čas) transakce a hodnotu transakce. Dále je možno zvolit sloupeček, podle něhož se data rozdělí do clusterů. Soubor může obsahovat libovolné množství dalších

sloupečků (není tak nutné dalšího formátování dat v případě, že v datech jsou zároveň uloženy další informace), nicméně tyto sloupečky pro samotnou analýzu nebudou brány v potaz. Vstupní parametr odkazující na sloupeček s id zákazníka je označen v parametrech jako *entity* z důvodu, že ačkoli je nejčastěji analýza prováděna nad vybranou obchodní doménou, tak to nemusí býti pravda a lze analyzovat i jiné domény (místo zákazníka může jít například o procesy).

```
1  private Indexes getIndexes(String headerLine){
2      Indexes result = new Indexes();
3      String[] header = headerLine.split(separator);
4      for (int i = 0; i < header.length; i++){
5          String column = header[i].trim();
6          if (column.equals(columns.getCluster())) {
7              result.setClusterIndex(i);
8          } else if (column.equals(columns.getEntity())) {
9              result.setEntityIdIndex(i);
10         } else if (column.equals(columns.getPrice())) {
11             result.setPriceIndex(i);
12         } else if (column.equals(columns.getDate())) {
13             result.setDateIndex(i);
14         }
15     }
16     return result;
17 }
```

Listing 3.3: Přiřazení sloupečků s daty

CsvRFMDataProvider nejprve pomocí metody *getIndexes* vyřeší, jaké sloupečky odpovídají jednotlivým vlastnostem transakcí. Následně začne načítat jednotlivé řádky z poskytnutého CSV souboru. Zároveň s tímto načítáním jednotlivých řádků zákazníky třídí do clusterů (je-li to požadováno) a dopočítává pro jednotlivé zákazníky jejich hodnoty recency, frequency a monetary. Jelikož není zaručeno (ve skutečnosti je spíše předpokládán opak), že transakce jsou ve vstupním souboru seřazeny tak, aby pro stejného zákaníka následovaly po sobě, tak si *CsvRFMDataProvider* po dobu načítání transakcí drží zákazníky pro jejich co nejrychlejší vyhledávání (adresaci) v HashMapě. Před vrácením výsledků z metody *getData* tedy ještě musí dojít ke konvezi z této mapy na požadovaný list.

```
1 result.putIfAbsent(cluster, new HashMap<String, User>());
2
3 User user = result.get(cluster).putIfAbsent(userId, new User(
      userId, date, 1, price));
4 if (user != null) {
5     user.setFrequency(user.getFrequency() + 1);
6     user.setMonetary(user.getMonetary() + price);
7     user.setRecency(user.getRecency().compareTo(date) > 0 ? user.
      getRecency() : date);
8 }
```

Listing 3.4: Aktualizace uživatelů při načítání

3.1.3.3 Segmentace

Všechny způsoby segmentace zastřešuje v současné době abstraktní třída *RFMSegmenter*. Při zavolání její metody *performRFMSegmentation* dojde k načtení dat, vytvoření výstupních kontejnerů a následné iteraci přes jednotlivé clustery, pro které je spuštěna segmentace a případné interpretace. Třída nechává své zděděné třídy provést samotné nastavení jednotlivých hranic, čímž se určí, jak vlastně daná segmentace má vypadat a následně zajistí přiřazení jednotlivých zákazníků do daných segmentů. Jelikož je možné, aby nastala situace, kdy mají dva segmenty stejné hranice, je zvolen přístup, kdy je zákazník přiřazen nejprve do toho horšího z nich.

```
1  private Map<User, RFMResult> performRFMSegmentationOfCluster(List<
       User> users) {
2      resolveThresholds(users);
3      Map<User, RFMResult> result = new HashMap<User, RFMResult>(
       users.size());
4      for (User user: users) {
5          RFMResult rfm = new RFMResult(
6              getSegment(user.getRecency(), recencyThresholds),
7              getSegment(user.getFrequency(), frequencyThresholds),
8              getSegment(user.getMonetary(), monetaryThresholds));
9          result.put(user, rfm);
10     }
11     return result;
12 }
```

Listing 3.5: Segmentace clusteru

```
1  private <T extends Comparable> int getSegment(T value, T[]
       thresholds){
2      for (int i = 0; i < thresholds.length; i++) {
3          if (value.compareTo(thresholds[i]) <= 0) {
4              return i + 1;
5          }
6      }
7      return thresholds.length + 1;
8  }
```

Listing 3.6: Přiřazení uživatele do segmentu

Implementace funkcionality třídy *CustomRFMSegmenter*, která zajišťuje segmentaci v případě, kdy je zvolen přístup vlastního definování jednotlivých hranic, je velmi jednoduchá. Jelikož sám uživatel poskytne aplikaci přímo potřebné hranice, již není zapotřebí tyto hranice dopočítávat. Třída tak v konstruktoru pouze přebere tyto hranice a provede jejich seřazení od nejmenšího

po největší pro případy, kdy uživatel zadá hranice v jiném pořadí. Zbytek práce již nechá na implementaci svého předka.

Implementace funkcionality třídy *QuantilRFMSegmenter*, která zajišťuje segmentaci v případě, kdy je zvolen přístup definování počtu segmentů, je již oproti funkcionalitě třídy *CustomRFMSegmenter* trochu složitější, avšak i zde poměrně přímočará. Jelikož třída *User* zároveň poskytuje k dispozici i komparátory pro řazení zákazníků dle jejich vlastností, stačí pouze vyvolat toto seřazení zákazníků postupně pro zvolené vlastnosti a následně vybrat hraniční zákazníky, jejichž daná vlastnost se stane hraniční hodnotou pro jednotlivé segmenty. Při tomto procesu je ještě potřeba ošetřit krajní případy, ke kterým může dojít, je-li počet segmentů roven 1 (v tom případě zde není žádná hranice) nebo je počet uživatelů menší než počet segmentů, do kterých se mají rozdělit (v tomto případě bude jeden zákazník definovat více hranic, které budou shodné).

```
1 recencyThresholds = new Date[this.recencyQuantilCount-1];
2
3 Collections.sort(users, new UserRecencyComparator());
4 double step = users.size() / (double)recencyQuantilCount;
5 for (int i = 1; i <= recencyThresholds.length; i++) {
6     recencyThresholds[i-1] = users.get((int)Math.max(0,(step*i)-1))
      .getRecency();
7 }
```

Listing 3.7: Ukázka získání hranic ve třídě QuantilRFMSegmenter

Třída *GradualQuantilRFMSegmenter* obstarává funkcionalitu, která zajišťuje segmentaci v případě, kdy je zvolen přístup definování počtu segmentů s postupným vyhodnocováním. Tento přístup si musí poradit s početnou skupinou, jež se rozprostírá přes několik segmentů, nicméně funguje na velmi podobném principu jako *QuantilRFMSegmenter*. Také nejprve seřadí zákazníky podle dané vlastnosti a následně vybere hraniční zákazníky. Ale jak již název napovídá, tak namísto toho, aby hned vybrala všechny hraniční zákazníky najednou, vybírá tyto zákazníky postupně. Po každém vybrání zákazníka dojde totiž k vyřazení všech zákazníků se stejnou (a nižší) hodnotou dané vlastnosti a další hraniční zákazníci jsou již vybráni jen ze zbylých zákazníků.

```
1  recencyThresholds = new Date[this.recencyQuantilCount-1];
2
3  Collections.sort(users, new UserRecencyComparator());
4  int taken = 0;
5  for (int i = 1; i <= this.recencyThresholds.length; i++) {
6      int index = (int)Math.max(0,((users.size() - taken) / ((double)
       recencyQuantilCount-i+1))-1+taken);
7      recencyThresholds[i-1] = users.get(index).getRecency();
8      while (index < users.size() && users.get(index).getRecency().
       compareTo(recencyThresholds[i-1]) == 0) {
9          index++;
10     }
11     taken = index;
```

```
12 }
```

Listing 3.8: Ukázka získání hranic ve třídě GradualQuantilRFMSegmenter

3.2 Data

Pro praktické příklady a diskuzi nad aplikací RFM analýzy (jíž se tato práce věnuje v podkapitole 3.3) jsou zvolena anonymizovaná reálná data představující transakční údaje zákazníků online obchodu, která byla poskytnuta firmou Bizztreat s.r.o.[34]. Data představují posledních zhruba 800 tisíc objednávek, které provedlo zhruba 200 tisíc zákazníků blíže nespecifikovaného e-shopu. To, že se jedná o pouze posledních několik objednávek a ne celou transakční historii, není pro RFM analýzu zrovna optimální, jelikož pravděpodobně dojde k zhoršení některých vlastností určitých zákazníků (zejména těch, co dlouho nenakoupili). Pro účely této práce a příkladů, jimž se věnuje, se bude však tato množina dat považovat (ve většině případů) za úplnou a tento fakt bude zmíněn pouze ve chvíli nestandardní situace.

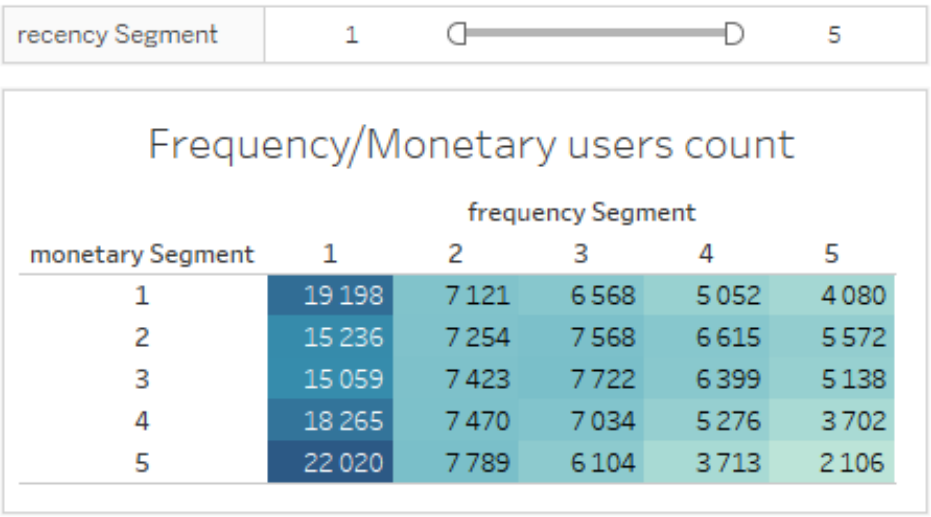
3.3 Vizualizace a aplikace v praxi

Možností, jak zobrazit samotné výsledky segmentace je samozřejmě celá řada. Konkrétní pohled na data se však vždy musí přizpůsobit účelu informace, kterou má sdělit. Jelikož v tomto projektu jde hlavně o praktický analytický pohled na data, není vytvořena žádná 3D vizualizace, která by zobrazovala výsledky RFM segmentace jako třírozměrnou matici (krychli). Taková vizualizace sice na první pohled zaujme, jde-li však o její praktické použití, její funkce zaostávají. Její nedostatky tkví především v tom, že vzhledem k množství dat, které jsou obvykle použity (ve většině případů jde o miliony řádků transakcí nad několika desítkami až stovkami tisíc zákazníků), je stejně prakticky nemožné projektovat výsledky segmentace přímo do této třírozměrné matice. Tato matice se pak stává jen jakýmsi ovládacím prvkem, jež určuje, která data se budou zobrazovat. Tento ovládací prvek má ve výsledku stejně horší použitelnost, než v případě, je-li zvoleno vhodné nastavení filtrů a průřezů touto maticí. Místo vytvoření takové 3D matice je tedy zvolena vizualizace právě ve formě průřezů touto maticí. V následujících sekcích jsou popsány vizualizované průřezy a jejich aplikace na praktických příkladech z praxe.

3.3.1 Frequency/Monetary

Jedním z nejčastěji používaných průřezů RFM matice je ten, který zobrazuje dimenze frequency a monetary. Na obrázku 3.1 je možné vidět jeho vizualizaci v prostředí Tableau. K průřezu je přidán praktický filtr, pomocí něhož lze nastavit agregaci přes zbývající dimenzi recency. Je tak možné sledovat nejen

projekci napočítanou přes vyřazenou recency, ale přímo i jednotlivé segmenty s konkrétní recency, frequency a monetary. V tomto průřezu je pak vidět počet jednotlivých zákazníků v daných segmentech, ze kterého lze vyvozovat závěry důležité pro obchodní strategie. Po rozkliknutí příslušného okénka lze v aplikaci Tableau Desktop zobrazit všechny zákazníky patřící do této zvolené kategorie a je-li to požadováno, lze vyexportovat data o příslušných zákaznících například ve formě souboru CSV. Formát tohoto zobrazení je možné vidět na obrázku 3.2, kde jsou v tabulce vypsáni všichni zákazníci s monetary a frequency segmenty rovnající se pěti. Vzhledem k tomu, že pro tento příklad byla použita anonymizovaná data, jsou z dat o zákaznících uvedeny pouze jejich anonymizované id. V případě, že by však šlo o reálné použití v praxi, tak zde budou samozřejmě uvedeny i údaje o zákazníkovi potřebné pro další práci, jako je například jeho jméno a kontaktní údaje, které lze předat marketingovému oddělení.



Obrázek 3.1: Frequence/Monetary(Avg) průřez s počety uživatelů

Segmentace z obrázku 3.1 byla provedena způsobem specifikování počtu segmentů s postupným vyhodnocováním, kde monetary jednotlivých zákazníků je specifikována jako hodnota jejich průměrného nákupu. Mimo jiné je zde možné pozorovat typický jev vyskytující se u internetových obchodů. Tímto typickým jevem vyskytujícím se běžně u internetových obchodů je velké množství zákazníků, kteří nakoupili právě jednou a již neprovedli žádný další nákup (frequency=1). Propast mezi prvním a druhým segmentem je opravu velká a jen potvrzuje fakt, že jednou z nejtěžších věcí je zákazníka přesvědčit ke druhému nákupu. Ve všech popsaných případech platí, že čím vyšší segment, tím

lépe a kromě výrazných segmentů se zákazníky, kteří nakoupili pouze jednou, je možné pozorovat skoro až vzácně vyrovnané spektrum zákazníků. Jediní častější zákazníci, kterých je výrazněji méně, jsou ti v segmentu frequency=5, monetary=5. To je však očekávatelné vzhledem k tomu, že se jedná o často nakupující zákazníky, kteří pravidelně nakupují za velké částky a takové je samozřejmě nejtěžší sehnat.

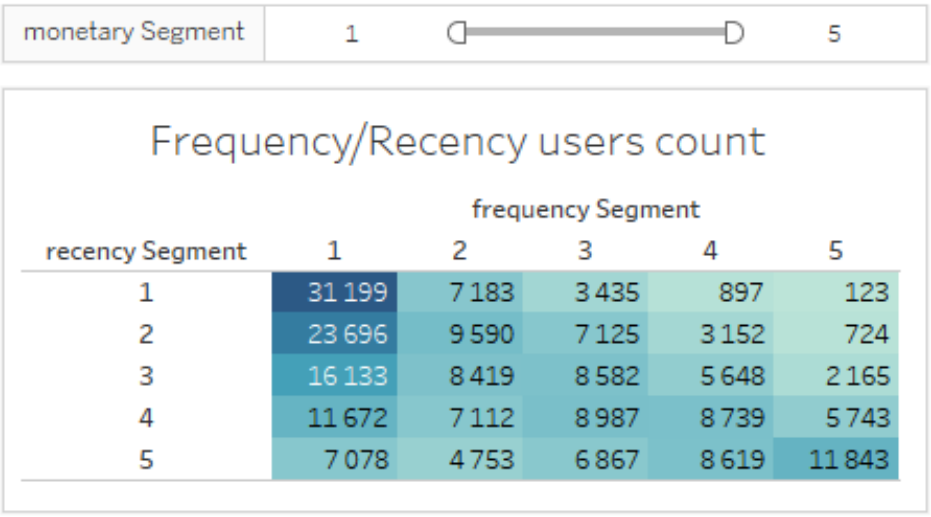
2 106 rows | Show aliases | Show all fields | Copy | Export All

Entity Id	RFM Score	# of orders	Avg order value	Days since last purchase
2 276 276	455	16	474,81	11
1 154 689	255	21	455,00	27
802 693	555	10	427,30	6
881 667	355	12	433,08	19
1 092 243	555	9	607,56	7
159 183	455	9	441,11	11
2 548 470	455	9	713,11	12
648 117	555	9	1 076,11	6
1 675 167	555	10	474,10	8
56 269	555	9	475,56	8
392 271	555	23	546,35	8
2 413 227	455	14	492,86	11
1 844 883	455	9	442,44	13
2 073 371	555	14	405,86	7
1 322 555	455	16	449,13	14
377 407	555	9	588,44	8
2 525 813	455	12	557,71	10
2 521 458	555	9	428,67	8

Obrázek 3.2: Výčet uživatelů v segmentu F=5, M=5

3.3.2 Frequency/Recency

Dalším možným průřezem RFM matice je průřez, který zobrazuje dimenze frequency a recency. Takovouto vizualizaci, která byla provedena nad poskytnutými daty (viz kapitola 3.2), je možné vidět na obrázku 3.3. Jde o stejnou segmentaci jako v případě průřezu frequency/monetary a platí tedy, že byla rovněž provedena způsobem specifikování počtu segmentů s postupným vyhodnocováním. I u tohoto průřezu platí, že je zachována možnost rozkliknutí jednotlivých okének tabulky, stejně jako filtrace pomocí zbývající osy. Na tomto zobrazení frequency/recency vynikají zákazníci s jediným nákupem

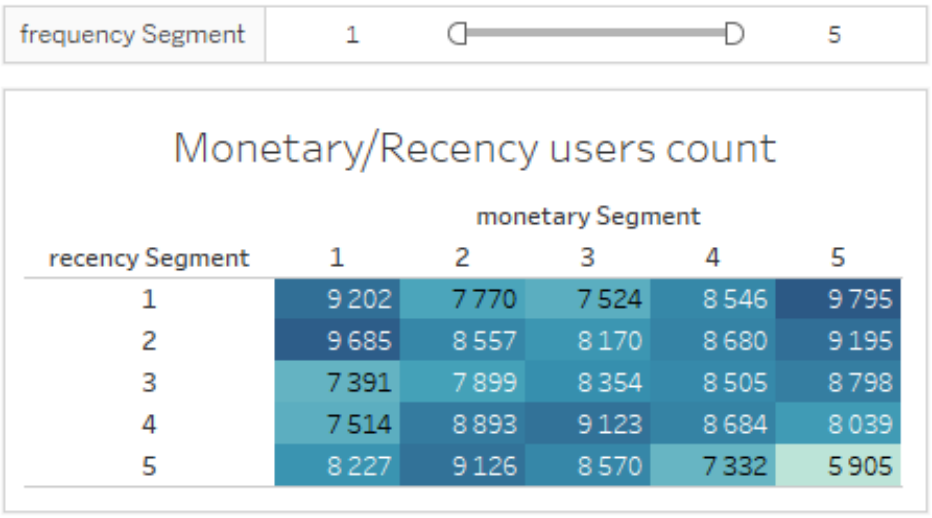


Obrázek 3.3: Frequence/Recency průřez s počety uživatelů

o něco méně, než tomu je v případě průřezu frequency/monetary. Při bližším pohledu na data ale lze právě takovéto rozložení očekávat. Největší skupiny zastupují segmenty s frequency=1 a recency=1 a 2. To jsou přesně ti zákazníci, kteří nakoupili před dávnou dobou a kteří se již nevrátili. Tito zákazníci s jediným nákupem se totiž, jak jde čas, postupně propadávají do segmentu recency=1, kde již zůstanou (pokud se náhodou nevrátí), a tak se zde kumulují. Na druhou stranu je poměrně rozumné předpokládat větší zastoupení v nejlepším segmentu frequency=5, recency=5. Lze totiž očekávat, že zákazníci, kteří nakupují nejčastěji (frequency=5), totiž pravděpodobně nakoupili i v nedávné době, a tak se ocitnou v segmentu recency=5. Velmi dobrým znamením je malé množství zákazníků v pravém horním rohu tabulky. Zde jsou totiž kdysi loajální zákazníci, kteří nakupovali velmi často, ale z neznámého důvodu přestali. V případech, kdy se zákazníci dostanou až do segmentu recency=1, je již většinou pozdě a zákazníci jsou navždy ztraceni. I tak je ale vhodné (pokud to zdroje dovolují), pokusit se je kontaktovat s vybranou výhodnou nabídkou. Pokud se totiž rozhodnou pro opětovný nákup, je větší šance, že stanou zase hodnotnými stálými zákazníky, než u nových zákazníků. V ideálním případě je samozřejmě vhodné se této situaci pokusit předejít a zahájit kroky již pro segmenty s recency=2 až 4, přičemž většinou platí, že čím dříve, tím lépe. Další výraznou skupinou, která si zaslouží pozornost je skupina v segmentu receny=5, frequency=1. Toto jsou úplně noví nedávní zákazníci a je pravděpodobně nejvhodnější čas pokusit se je přesvědčit, aby i příště nakoupili znovu.

3.3.3 Monetary/Recency

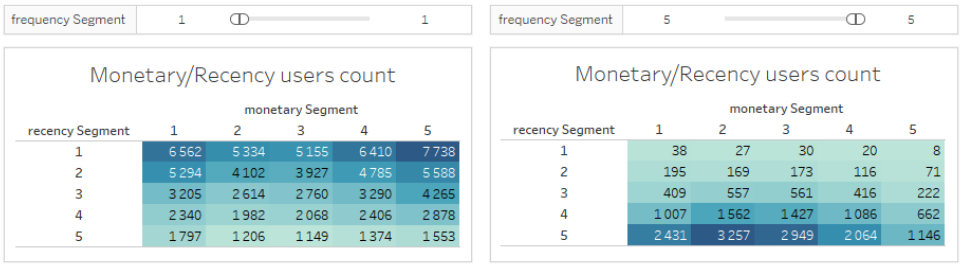
Posledním možným průřezem RFM matice je průřez, který zobrazuje dimenze recency a monetary. Konkrétní vizualizace, která byla rovněž provedena nad poskytnutými daty a způsobem specifikování počtu segmentů s postupným vyhodnocováním, je vyobrazena na obrázku 3.4. Nejzajímavější informaci prav-



Obrázek 3.4: Monetary/Recency průřez s počety uživatelů

děpodobně podává poslední sloupeček představující segment monetary=5, na kterém lze zhodnotit vývoj zákazníků utrácejících za své nákupy největší částky. Jak je možné vidět, tak soustavně docházelo (a dochází) k úbytku zákazníků, kteří jsou ochotní utratit větší množství peněz. Alarmující je pak ale zejména propad v nedávné době (rozdíl mezi 4. a 5. segmentem). Mezi možné příčiny může patřit například výraznější změna produktové nabídky, nová konkurence na trhu nebo pokles kvality produktů a služeb. Následující kroky v této situaci jsou předmětem marketingové analýzy. Při zapojení frequency je možné pozorovat (viz porovnání průřezů s porovnáním frequency segmentů na obrázku 3.5), že se výrazně změnil přístup zákazníků a zatím co v minulosti existovala početná skupina zákazníků, kteří utratili velký obnos peněz, ale byl to jejich jediný nákup, tak v současnosti jsou zde spíše pravidelní zákazníci, kteří však utrácí menší množství peněz. Tuto změnu lze chápat jako pozitivní událost, jelikož pravidelní zákazníci budou, na rozdíl od příležitostných zákazníků, pravděpodobně pokračovat v nákupech. Na tomto místě je také nutno zmínit, že na tyto výsledky má s největší pravděpodobností dopad také způsob, jakým byla data pořízena. Jelikož se jedná o dataset pořízený jako určitá část posledních objednávek (a nezahrnuje tedy všechny

objednávky, které kdy byly provedeny), tak je velmi pravděpodobné, že starším zákazníkům nebyly započítány některé transakce. I přes tento fakt však lze pozorovat zmíněný přesun v druhu zákazníků v prvním řádku první tabulky (frequency=1, recency=1) a posledním řádku druhé tabulky (frequency=5, recency=5) na obrázku 3.5.

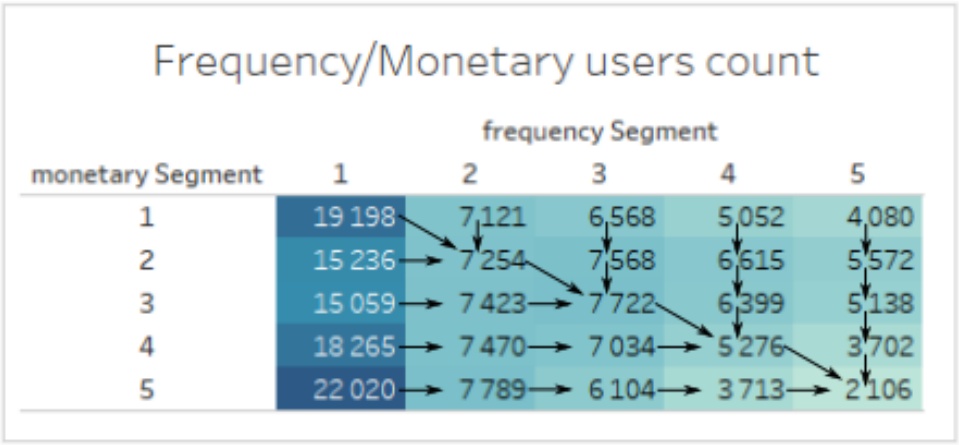


Obrázek 3.5: Monetary/Recency průřez s počety uživatelů - porovnání frequency segmentů

3.3.4 Obchodní potenciál

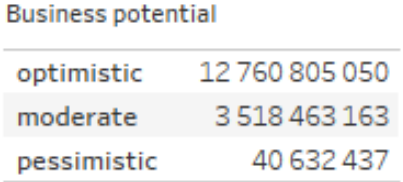
Po analýze průřezu frequency/monetary je možné rozčlenit zákazníky podle jejich finančního přínosu eshopu. Obecně platí, že v čím větším segmentu se zákazník nachází, tím více utrácí v eshopu peněz. Platí to přitom pro obě osy. Lze prohlásit, že zákazník v segmentu frequency=1, monetary=5 utrácí zhruba stejně peněz jako zákazník v segmentu frequency=5, monetary=1 (ten ale bude trochu cennější z důvodu své loajality, viz podkapitola 3.3.5). První zákazník totiž sice utrácí hodně peněz, ale přijde nakoupit jednou za čas, druhý zase nakupuje často, ale za minimální částky. Ve výsledku tedy opravdu utratí zhruba stejně. Obchodním cílem je snažit se maximum uživatelů posunout v tabulce co nejblíže segmentu frequency=5, recency=5. Tedy vyjádřeno slovy obchodníka, snažit se je přimět nakupovat častěji za větší částky.

Z dostupných dat je možné spočítat rozdíl mezi současným stavem a stavem, kdy bude každý zákazník posunut blíže k segmentu frequency=5, monetary=5. Po posunutí uživatele po diagonále (vyskytuje-li se na diagonále) nebo posunutí uživatele blíže k diagonále (není-li na diagonále), jak to lze vidět na obrázku 3.6, lze vypočítat obchodní potenciál se stávající zákaznickou bází. Je tak možné vypočítat rozdíl v příjmech od zákazníků v současném stavu a za podmínek, kdy jsou všichni zákazníci motivování utrácet častěji nebo více. Samotný obchodní potenciál lze přitom spočítat v několika rovinách. První z nich je rozdíl oproti stavu, kdy budou zákazníci přesunuti na samotný začátek dalšího segmentu, tedy na hranici mezi současným a dalším segmentem. Toto vyčíslení obchodního potenciálu je nazváno pesimistickým (pessimistic).



Obrázek 3.6: Posun pro výpočet obchodního potenciálu

Další vyčíslení obchodního potenciálu, které je nazváno optimistickým (optimistic) je naopak na druhé straně spektra a jde o rozdíl mezi současnou situací a situací, kdy budou zákazníci přesunutí až na konec dalšího segmentu (dostanou se mezi nejlepší uživatele dalšího segmentu). Poslední spočítané vyčíslení obchodního potenciálu, které je pravděpodobně nejvíce vypovídající, je nazvané střídmým (moderate). Jde o kompromis mezi pesimistickým a optimistickým vyčíslením, kdy je toto vyčíslení vypočítáno jako rozdíl současného stavu oproti stavu, kdy budou zákazníci přesunuti do středu následujícího segmentu. Jelikož samotný výpočet tohoto obchodního potenciálu již vyžaduje složitější iteraci přes segmenty a zákazníky, je prováděn v rámci *FMShiftInterpreteru* v aplikaci *RFM Segmentation Application* a vizualizačnímu nástroji Tableau jsou již předány pouze vypočítané hodnoty tohoto potenciálu. Jak vypadají konkrétní vypočítané hodnoty obchodního potenciálu pro poskytnutá data je možné vidět na obrázku 3.7. Je nutné uvést, že obrovská čísla, která



Obrázek 3.7: Obchodní potenciál

vychází, rozhodně nejsou v praxi dosažitelná. Dokáží však poměrně dobře po-

sloužit k představě o možných potenciálních dopadech v případně podniknutí obchodních kroků ke zvýšení frekvence nákupů nebo zvýšení průměrné velikosti objednávky. Je možné totiž snadno vypočítat, jaké dopady na tržby bude mít zvýšení těchto hodnot například u pouhého půl procenta zákazníků.

3.3.5 RFM Score a kategorizace

Mimo zobrazení jednotlivých průřezů maticí je možné zákazníkům na základě jejich příslušnosti v jednotlivých segmentech také přiřadit jejich RFM Score. To je možné buď přímo využít pro seřazení jednotlivých zákazníků a nebo ještě lépe využít k rozdělení těchto zákazníků do kategorií. Jak přitom přitom budou dané kategorie vypadat je, v tomto případě, velmi závislé na požadavcích, ke kterým RFM analýza slouží. Pro příklad e-shopu z tohoto projektu může být vhodné rozdělit zákazníky například na následující kategorie:

Best Customers (R=5, F=5, M=5): Toto jsou nejlepší zákazníci e-shopu. Jedná se o pravidelné loajální zákazníky utrácející velké částky a to i v současné době. Těmto VIP zákazníkům by mělo být vyhověno pokud možno ve všech jejich smysluplných požadavcích za účelem udržení si jejich přízně a z toho plynoucích zisků (větší režijní náklady vyváží jejich obchodní aktivita). Z tohoto důvodu pro ně můžou být zavedeny například speciální support linky nebo jiné výhody.

New Promising Customers (R=5, F=1, M=3-5): Toto jsou úplně noví zákazníci, kteří nakoupili v nejbližší době za nezanedbatelnou částku. Vzhledem k tomu, že utratili významnou částku a zkušenost z nákupu je ještě čerstvá, tak je určitě na místě pokusit se je motivovat k dalšímu nákupu, aby se z nich stali stálí zákazníci. Toto je právě ta skupina lidí, na kterou se vyplatí cílit typickou strategii poskytnutí slevy na další nákup.

Highly Non-Interesting Customers (R=1-2, F=1, M=1-2): Tato skupina obsahuje přesně ty nezajímavé zákazníky, kteří nakoupili před dávnou dobou za málo peněz a již se nevrátili. Podnikat marketingové kroky s touto kategorií je ztráta prostředků.

Lost Small Customers (R=1, F=X, M=1-2): Toto je skupina ztracených malých zákazníků. Podobně jako v případě skupiny Highly Non-Interesting Customers, ani zde se příliš nevyplatí podnikat marketingové kroky.

Lost Big Customers (R=1, F=X, M=5): Skupina obsahující ztracené velmi dobře platící zákazníky. Zde se jedná o skupinu, na kterou se určitě vyplatí cílit s nabídkami pro obnovení obchodní činnosti. Zároveň není potřeba extra velké opatrnosti, jelikož se jedná o zákazníky, kteří jsou již v současné době považováni za ztracené, a tak nelze kampaní moc co ztratit.

Leaving Customer (R=3, F=3-5, M=X): Zákazníci v této skupině představují jednu z nejdůležitějších skupin zákazníků získanou pomocí RFM analýzy. Jedná se o pravidelné zákazníky, kteří v současné době ztrácejí o eshop zájem. U této skupiny zákazníků je vhodné uplatnit největší množství zdrojů na jejich přivedení zpět, jelikož se jedná skupinu tvořící jádro stabilních objednávek. V tomto případě je vhodné provést další průzkumy, proč tito zákazníci ztrácejí o eshop zájem - souvisí to se změnou v eshopu či zákazník opravdu pouze ztratil zájem a je potřeba ho motivovat, připomenout se mu.

Big Customers (R=X, F=X, M=5): Toto jsou zákazníci, kteří v eshopu utrácejí velké objemy peněz. Tito zákazníci mohou tvořit nemalou část tržeb a je dobré jim poskytovat výhody například ve formě dopravy zdarma. Zároveň se u nich vyplatí cílit na zvyšování jejich frekvence nákupů. Toho lze dosáhnout například pomocí různých časově omezených slevových nabídek a to ideálně na produkty, které tito zákazníci nakupují.

Lost Loyal Customers (R=1-2, F=5, M=X): Jedná se o skupinu, ke které se bude přistupovat velmi podobně jako ke skupině Lost Big Customers. Stejně jako v jejím případě, jde o ztracené zákazníky, kteří představovali velmi dobrý zdroj zisků, a tak je dobré pokusit se s nimi obnovit vztah.

Loyal Customers (R=X, F=5, M=X): Zákazníci v této skupině taktéž představují jednu z velmi důležitých skupin zákazníků získanou pomocí RFM analýzy. Jedná se o pravidelné loajální zákazníky. Prvním poznatkem je, že je prakticky zbytečné snažit se je motivovat k častějším nákupům či jim výrazněji nabízet různé slevové akce. Jedná se totiž o zákazníky, kteří nakupují velmi často a pravidelně. Druhým poznatkem je, že je zbytečné snažit se je motivovat různými loajálními programy, jelikož již loajální jsou a je to tedy ztráta prostředků. V případě, že by přestali pravidelně nakupovat, propadnou se do jiného segmentu, pro který už by měla být nastavená vhodná pravidla pro obnovení jejich zájmu. Dalším poznatkem však je, že to ale jsou právě tito zákazníci, které je vhodné udržovat maximálně spokojené a nabízet jim různé věrnostní výhody typu předčasného přístupu k určitému druhu zboží, vyhrazené support linky a podobně. Kromě toho právě tuto skupinu lze velmi dobře využít k šíření dobrého jména o eshopu a jsou to tedy právě tito zákazníci, které je vhodné žádat o recenze na produkty, sdílení statusů na sociálních médiích a podobně.

Ordinary Customers Tuto skupinu tvoří zákazníci, kteří nespadají do žádné výše zmíněné kategorie. Jedná se o obyčejné zákazníky, kteří nevyžadují žádný zvláštní přístup.

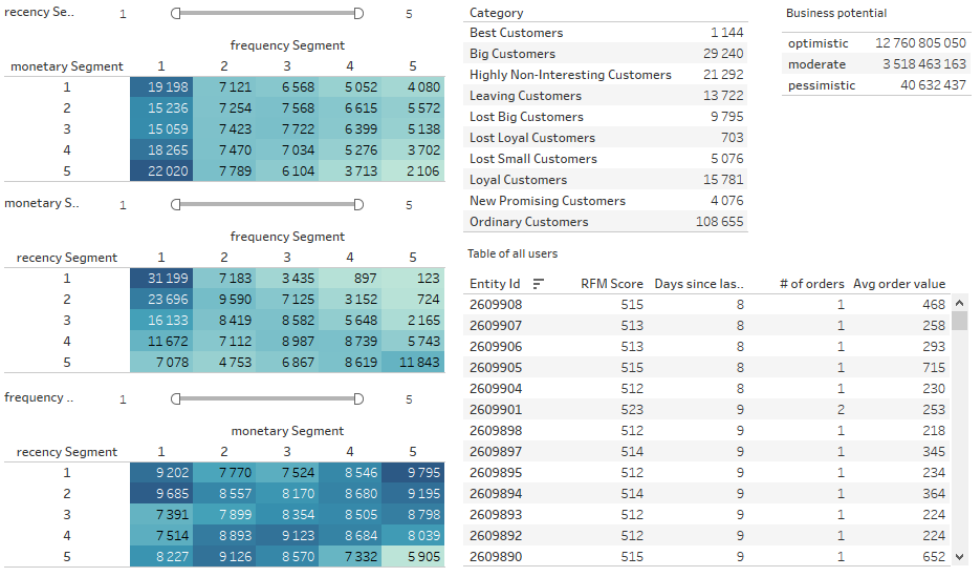
Kategorie s jednotlivými zákazníky je možné zobrazit v praktické tabulce, kterou lze vidět na obrázku 3.8, přičemž platí, že v tomto případě každý zákazník patří právě do jedné kategorie, která ho vystihuje nejpřesněji. To je provedeno zejména z toho důvodu, aby, budou-li na základě těchto kategorií rozeslány reklamní nabídky, nedošlo k zaslání více informačních emailů s nabídkami jednomu zákazníkovi. Pokud ale existují kontrolní mechanismy nebo je zvoleno jiné vhodné využití těchto kategorií (například zobrazování různých reklamních banerů atp.), není nemožné provést i vizualizaci, kdy bude zákazník součástí více skupin. Rozkliknutím příslušné kategorie je možné v nástroji Tableau Desktop opět zobrazit a vyexportovat seznam jednotlivých zákazníků spadajících do dané kategorie.

Category	
Best Customers	1 146
Big Customers	29 245
Highly Non-Interesting Customers	21 292
Leaving Customers	13 722
Lost Big Customers	9 796
Lost Loyal Customers	703
Lost Small Customers	5 076
Loyal Customers	15 783
New Promising Customers	4 076
Ordinary Customers	108 657

Obrázek 3.8: Kategorie s roztříděnými uživateli

3.3.6 Dashboardy

Samozřejmostí pro prostředí Tableau je i spojení jednotlivých dílčích vizualizací do praktických interaktivních dashboardů. Na obrázku 3.9 je zobrazena jednoduchá verze takového dashboardu spojující vizualizace popsané v předchozích částech této práce. Tyto dashboardy je v praxi dobré obohatit o další zdroje týkající se dané domény pro vytvoření ještě ucelenějšího pohledu na výsledky a podpory při dalším rozhodování.



Obrázek 3.9: Dashboard pro vizualizaci výsledků RFM analýzy

Kapitola 4

Ověření a testování

Kapitola se zabývá způsobem, jímž byly provedeny testy, které ověřují funkčnost vyvíjené aplikace. Také ověřuje skutečnost, zda má aplikace vhodné asymptotycké složitosti z pohledu doby běhu aplikace.

4.1 Unit testy

Pro ověření správné funkcionality napsaného kódu jsou pro části kódu, které to svojí podstatou vyžadují (a zároveň dovolují), napsány unit testy. Cílem těchto unit testů je ověřování správné funkčnosti jednotlivých dílčích částí (neboli jednotek - unit) zdrojového kódu. Za jednotku zdrojového kódu se nejčastěji považuje konkrétní třída nebo metoda (v následujících částech práce lze proto považovat pojmy metoda a jednotka za zaměnitelné). V ideálním případě by zároveň měl každý unit test testovat pouze konkrétní danou jednotku a být nezávislý na ostatních jednotkách. Toho lze dosáhnout izolováním ostatních částí programu. Pokud jsou takové části programu závislé na ostatních, lze je nahradit vytvořením pomocných objektů, které simulují předpokládaný kontext, který by měly poskytovat. Tyto pomocné objekty se nazývají mocky. Pro podporu automatizovaného testování pomocí těchto unit testů je zvolen framewrok JUnit[35] ve verzi 4. Pro vytvoření a správu mocků je použit framework Mockito[36] ve verzi 2.18.3, což je v současnosti jeho nejvyšší podporovaná verze dostupná přes Maven.

Všechny současné třídy dědící od třídy *RFMSegmenter* a implementující metodu *resolveThresholds* tak, že definují vlastní způsob pro vytvoření těchto hranic, mají implementované následující unit testy:

largeFirstSegment Účelem tohoto unit testu je ověřit předpokládané chování pro situaci, kdy bude skupina uživatelů se stejnou hodnotou přesahující několik segmentů.

sameValueForEveryUser Tento unit test ověřuje, že je segmenter schopen

správně fungovat i v případě, kdy všichni uživatelé budou mít stejnou hodnotu vlastnosti.

oneSegmentOnly Jedná se o unit test pro ověření správné funkcionality pro případ, kdy existuje pouze jeden segment. V tomto případě tudíž segmentery nesmí vrátit žádnou hranici jelikož všichni uživatelé patří do stejného segmentu.

fewerUsersThanBoundaries Tento unit test slouží k ověření správné funkčnosti segmenterů i v případě, že je dostupných méně uživatelů, než požadovaných segmentů.

Tyto unit testy slouží především pro ověření jejich funkcionality v krajních případech, které mohou nastat. Níže je zobrazena možná forma implementace takového unit testu.

```
1  @Mock
2  public CsvRFMDataProvider csvRFMDataProvider;
3
4  @Mock
5  public CsvRFMDataExporter csvRFMDataExporter;
6
7  @Test
8  public void fewerUsersThanBoundaries() throws Exception {
9      RFMSegmenter rfmSegmenter = new QuantilRFMSegmenter(
       csvRFMDataProvider, csvRFMDataExporter, 3, 3, 3);
10
11     rfmSegmenter.resolveThresholds(Arrays.asList(new User("
       TestUserId1", new SimpleDateFormat("yyyy-MM-dd").parse("
       2018-10-10"), 2, 1.0)));
12     Assert.assertEquals(2, rfmSegmenter.monetaryThresholds.length);
13     Assert.assertArrayEquals(new Double[]{1.0, 1.0}, rfmSegmenter.
       monetaryThresholds);
14 }
```

Listing 4.1: Unit test třídy QuantilRFMSegmenter

4.2 Ověření doby běhu

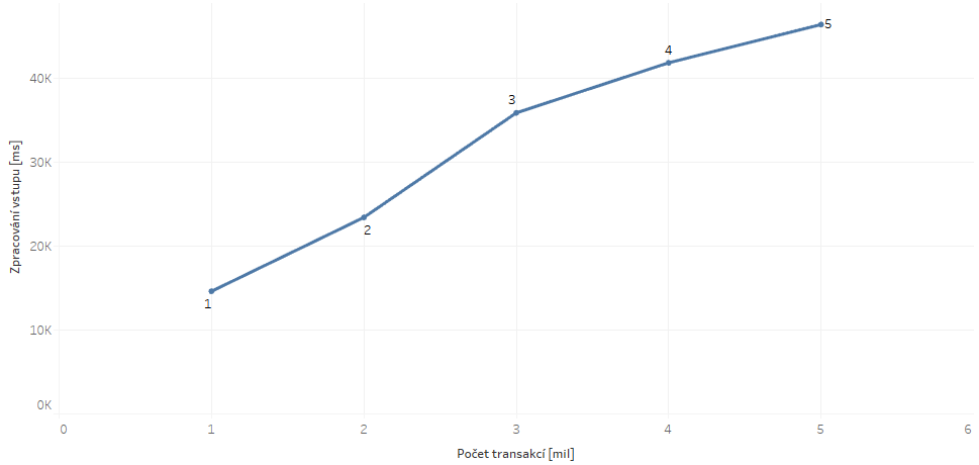
Doba běhu aplikace je závislá na dvou hlavních činnostech. První z nich je načtení dat do aplikace. Druhou z nich je samotná segmentace těchto dat. Následující sekce se zabývají ověřením, zda se obě tyto činnosti chovají dle předpokladů.

4.2.1 Zpracování vstupních dat

Načítání vstupu je implementováno tak, že algoritmus načítá jednotlivé řádky transakcí a ty zpracovává. Tento algoritmus má tedy lineární složitost *O(n)*.

Samotné zpracování je sestaveno z vyhledání uživatele v hash tabulce a modifikaci jeho hodnot nebo jeho přidání do této tabulky, pokud v ní ještě není. V obou případech se jedná o konstantní asymptotickou složitost *O(1)*. Z těchto uvedených informací tedy vyplývá fakt, že načítání vstupu by mělo být časově závislé pouze na počtu transakcí a nikoliv na počtu uživatelů, které tyto transakce provedli. To lze experimentálně ověřit v praxi. V rámci této práce byla provedena série měření na lokálním počítači při spuštění samotné aplikace RFM Segmentation Application (aplikace tedy běžela přímo v prostředí OS). To bylo provedeno z důvodu lepších možností automatizace tohoto měření. Na výsledky by tento přístup neměl mít žádný vliv, jelikož aplikace byla navržena tak, aby byla nezávislá na prostředí ve kterém je spuštěna. Měření byla provedena na náhodně generovaných datech.

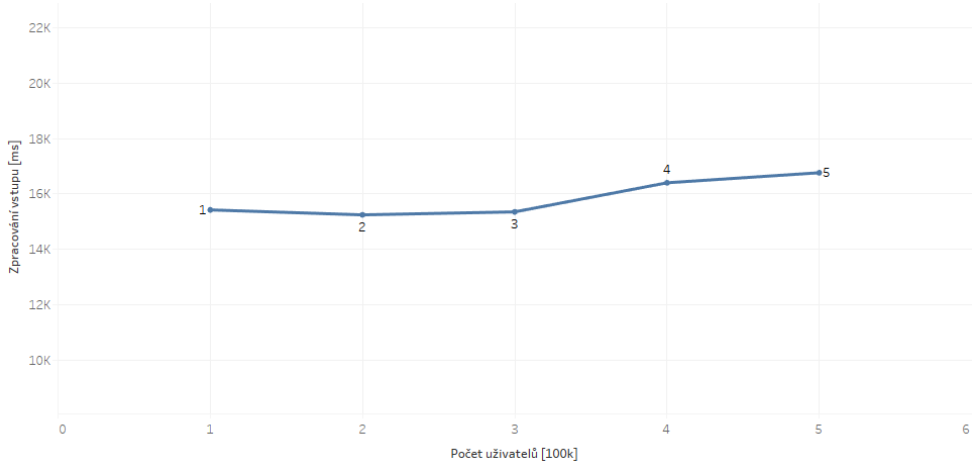
Na grafu na obrázku 4.1 lze vidět výsledky měření z hlediska načítání transakcí v závislosti na počtu transakcí. Náhodně generovaná data tvoří jeden až pět miliónů transakcí rozdělených mezi sto tisíc uživatelů. Je patrné, že výsledná křivka grafu opravdu odpovídá lineární složitosti.



Obrázek 4.1: Doba načítání transakcí v závislosti na počtu transakcí

Na grafu na obrázku 4.2 jsou zobrazeny výsledky měření z hlediska načítání transakcí v závislosti na počtu uživatelů. Náhodně generovaná data tvoří vždy jeden milión transakcí rozdělený mezi sto až pět set tisíc uživatelů. Jak lze vidět, tak výsledná křivka grafu opravdu skoro odpovídá konstantní složitosti. Drobné rozdíly, které lze pozorovat, je možné zdůvodnit dvěma fakty. První z nich je fakt, že měření nejsou úplně přesná a každý běh programu dopadne s určitou odchylkou z důvodu prostředí, na němž byla měření prováděna (OS na němž běžela aplikace nelze mít zcela pod kontrolou a nelze tak vyloučit zásahy dalších procesů tohoto systému na čas potřebný k dokončení stejně tak jako výchozí hardwarové podmínky). Druhým z nich je fakt,

že operace nad hash tabulkou mají sice konstantní asymptotickou složitost, nicméně to neznamená, že všechny operace nad touto tabulkou tuto složitost mít opravdu budou. S větším množstvím uživatelů přibývá konfliktů v této tabulce a tabulka tak musí reagovat na vzniklou situaci (ať už se jedná třeba o zvýšení alokovaného prostoru nebo jiné další specifické operace). To sice není vidět ve větším měřítku, ale v rámci měřených milisekund lze očekávat, že bude možné určité menší změny v době běhu pozorovat.



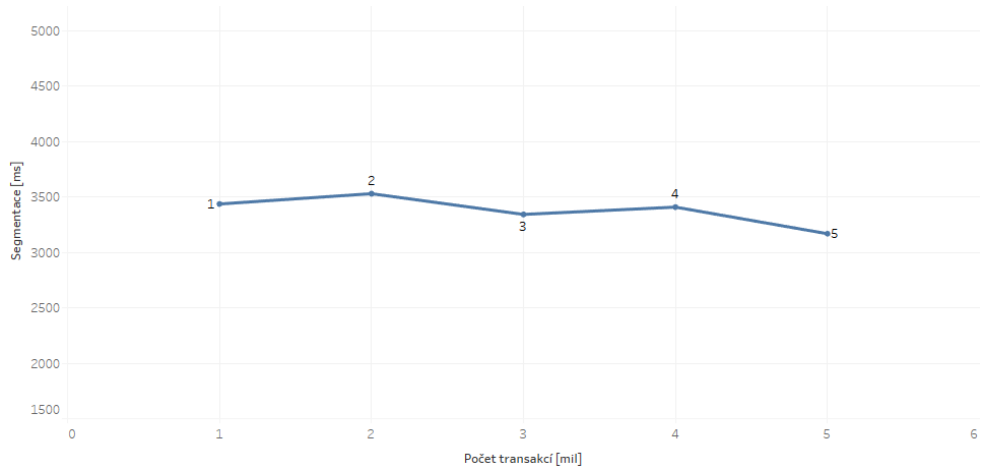
Obrázek 4.2: Doba načítání transakcí v závislosti na počtu uživatelů

4.2.2 Segmentace

Na rozdíl od načítání vstupu je možné očekávat přesně opačnou situaci, co se týče časových složitostí v závislosti na počtu transakcí a počtu uživatelů. Počet transakcí nemůže mít žádný vliv na dobu provádění segmentace, jelikož ta pracuje s uživateli a ne jejich jednotlivými transakcemi. Hlavní činností segmentace je řazení, jehož implementace se napříč verzemi javy značně liší, nicméně lze usuzovat na známou optimální asymptotickou složitost *O(n*Log(n))*.

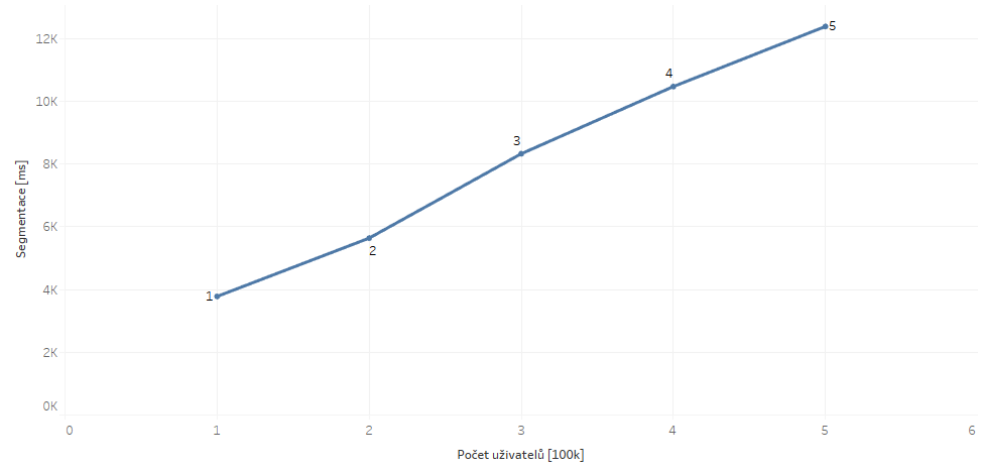
Na grafu na obrázku 4.3, který obsahuje výsledky měření z hlediska doby trvání segmentace v závislosti na počtu transakcí, je možné potvrdit, že počet transakcí nemá na dobu trvání segmentace žádný vliv. Stejně jako v případě měření doby trvání z hlediska načítání transakcí v závislosti na počtu uživatelů (graf na obrázku 4.2) lze drobné rozdíly připsat nepřesnostem v měření způsobené prostředím na němž jsou měření prováděna.

Z grafu na obrázku 4.4, který obsahuje výsledky měření z hlediska doby trvání segmentace v závislosti na počtu uživatelů, je možné skutečně pozorovat, že složitost není horší, než *O(n*Log(n))*. V tomto konkrétním případě je



Obrázek 4.3: Doba segmentace v závislosti na počtu transakcí

dokonce ještě nižší, což je možné připsat obsahu dat, jelikož konkrétní implementace řazení jsou datově závislá a značně optimalizovaná.



Obrázek 4.4: Doba segmentace v závislosti na počtu uživatelů

Závěr

Hlavním cílem této práce byla realizace snadno použitelné aplikace poskytující podporu pro provedení RFM analýzy v prostředí Keboola Connection. V rámci práce tak byla nejprve provedena analýza, která čtenáře seznámí s možnostmi RFM analýzy, všemi nástroji a technologiemi, kterých bude pro vývoj takové aplikace potřeba a taktéž rozbor všech požadavků na výslednou aplikaci. Po provedení analýzy byl popsán návrh celé aplikace zabývající se zejména pohledem na její architekturu a její vnitřní funkční logiku. Poté byla implementována samotná aplikace, která je v současnosti nasazena do prostředí Keboola Connection, kde byla také experimentálně ověřena její funkčnost. Z důvodu časové tísně se zatím bohužel nepodařilo dokončit proces jejího zveřejnění všem uživatelům prostředí Keboola Connection. Aplikace je však již nyní v tomto prostředí neveřejně dostupná a využívána například firmou Bizztreat s.r.o.

Po realizaci aplikace byly provedeny vizualizace výsledků, které aplikace poskytuje na anonymizovaných reálných datech. Práce v této části čtenáře seznamuje s možnostmi praktické aplikace RFM analýzy, a to zejména z pohledu jejího využití pro řešení obchodního problému zabývajícího se cílením vhodných obchodních strategií při dalším marketingu a jeho možných dopadech.

Náměty pro další rozvoj

Samotná aplikace byla navržena s důrazem na její další možný rozvoj. Je tak více než pravděpodobné, že do aplikace přibude podpora vstupních dat v dalších formátech, stejně tak jako nástroje pro podporu složitější interpretace RFM analýzy (v současné době byl implementován jediný nástroj zabývající se výpočtem obchodního potenciálu). Samozřejmostí je také dokončení procesu jejího zveřejnění všem uživatelům prostředí Keboola Connection.

Literatura

[1] Blattberg, R. C.; Kim, B.-D.; Neslin, S. A.: *RFM Analysis*. New York, NY: Springer New York, 2008, ISBN 978-0-387-72579-6, s. 323–337, doi: 10.1007/978-0-387-72579-6_12, [cit. 2018-04-03]. Dostupné z: https://doi.org/10.1007/978-0-387-72579-6_12

[2] Birant, D.: Data Mining Using RFM Analysis. 2011: s. 91–108, [cit. 2018-04-03]. Dostupné z: http://www.intechopen.com/books/knowledge-oriented-applications-in-data-mining/data-mining-using-rfm-analysis

[3] RFM Segmentation [online]. [cit. 2018-04-15]. Dostupné z: https://www.optimove.com/learning-center/rfm-segmentation

[4] Etnetera Activate [online]. [cit. 2018-04-03]. Dostupné z: https://www.activate.cz/category/segmentace/

[5] Rouse, M.: RFM analysis (recency, frequency, monetary) [online]. [cit. 2018-04-22]. Dostupné z: https://searchdatamanagement.techtarget.com/definition/RFM-analysis

[6] Keboola [online]. [cit. 2018-04-03]. Dostupné z: https://www.keboola.com

[7] Tableau [online]. [cit. 2018-04-21]. Dostupné z: https://www.tableau.com

[8] Tableau Hyper Technology [online]. [cit. 2018-04-28]. Dostupné z: https://www.tableau.com/products/new-features/hyper

[9] Tableau VizQL [online]. [cit. 2018-04-28]. Dostupné z: https://www.tableau.com/about/mission#vizql

[10] Six Years A Leader in the Gartner Magic Quadrant for Analytics and Business Intelligence [online]. [cit. 2018-04-28]. Dostupné z: https://www.tableau.com/reports/gartner

[11] GoodData Platform [online]. [cit. 2018-04-21]. Dostupné z: https://www.gooddata.com/platform

[12] GoodData [online]. [cit. 2018-04-21]. Dostupné z: https://www.gooddata.com

[13] Docker [online]. [cit. 2018-04-03]. Dostupné z: https://www.docker.com

[14] Augustýn, M.: Proč používat Docker [online]. [cit. 2018-04-03]. Dostupné z: https://www.zdrojak.cz/clanky/proc-pouzivat-docker/

[15] Vaughan-Nichols, S. J.: What is Docker and why is it so darn popular? [online]. [cit. 2018-04-15]. Dostupné z: https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular

[16] Foley, M. J.: Here's how Microsoft is supporting the open-source Docker container model [online]. [cit. 2018-04-15]. Dostupné z: https://www.zdnet.com/article/heres-how-microsoft-is-supporting-the-open-source-docker-container-model

[17] Wolpe, T.: Oracle steps up Solaris cloud push with Docker integration [online]. [cit. 2018-04-15]. Dostupné z: https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular

[18] Travis CI [online]. [cit. 2018-04-16]. Dostupné z: https://travis-ci.com

[19] Yaml.org [online]. [cit. 2018-04-16]. Dostupné z: http://yaml.org

[20] Apache Maven [online]. [cit. 2018-04-28]. Dostupné z: https://maven.apache.org

[21] Tim, O.; Manfred, M.; John, C.; aj.: *Maven: The Complete Reference*. Sonatype, Inc., [cit. 2018-04-28]. Dostupné z: http://books.sonatype.com/mvnref-book/pdf/mvnref-pdf.pdf

[22] IntelliJ IDEA [online]. [cit. 2018-04-16]. Dostupné z: https://www.jetbrains.com/idea/

[23] Martin, R. C.: *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ, USA: Prentice Hall PTR, první vydání, 2008, ISBN 0132350882, 9780132350884.

[24] Ledenev, A.: Crafting perfect Java Docker build flow [online]. [cit. 2018-04-21]. Dostupné z: https://codefresh.io/docker-tutorial/java_docker_pipeline/

[25] Docker Official Repository - Java Image [online]. [cit. 2018-04-21]. Dostupné z: https://hub.docker.com/_/java/

[26] Docker Official Repository - OpenJDK Image [online]. [cit. 2018-04-21]. Dostupné z: https://hub.docker.com/_/openjdk/

[27] Keboola Developer Portal [online]. [cit. 2018-04-21]. Dostupné z: https://kebooladeveloperportal.docs.apiary.io/

[28] Keboola Developer Portal CLI Tool [online]. [cit. 2018-04-21]. Dostupné z: https://github.com/keboola/developer-portal-cli-v2

[29] Keboola Component Deployment [online]. [cit. 2018-04-21]. Dostupné z: https://developers.keboola.com/extend/component/deployment/

[30] GSON [online]. [cit. 2018-04-12]. Dostupné z: https://github.com/google/gson

[31] Jenkov, J.: Java JSON Tutorial [online]. [cit. 2018-04-12]. Dostupné z: http://tutorials.jenkov.com/java-json/index.html

[32] JSON Schema [online]. [cit. 2018-04-22]. Dostupné z: http://json-schema.org

[33] Keboola Component Configuration [online]. [cit. 2018-04-22]. Dostupné z: https://developers.keboola.com/extend/publish/

[34] Bizztreat s.r.o. [online]. [cit. 2018-04-22]. Dostupné z: https://www.bizztreat.com/

[35] JUnit v4 [online]. [cit. 2018-04-29]. Dostupné z: https://junit.org/junit4/

[36] Mockito [online]. [cit. 2018-04-29]. Dostupné z: http://site.mockito.org

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

CLI Commanad-line Interface

CSV Comma Separated Value

ETL Extract-Transform-Load

IDE Integrated Development Environment

JSON JavaScript Object Notation

OS Operační systém

POM Project Object Model

REST Representational State Transfer

RFM Recency-Frequency-Monetary

SQL Structured Query Language

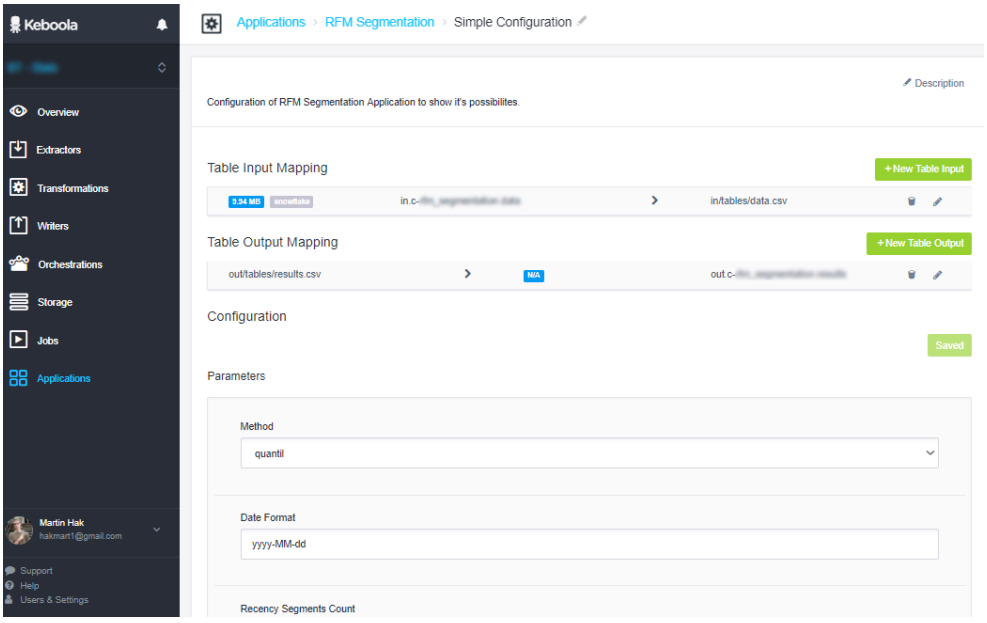
VIP Very Important Person

VizQL Vizual Query Language

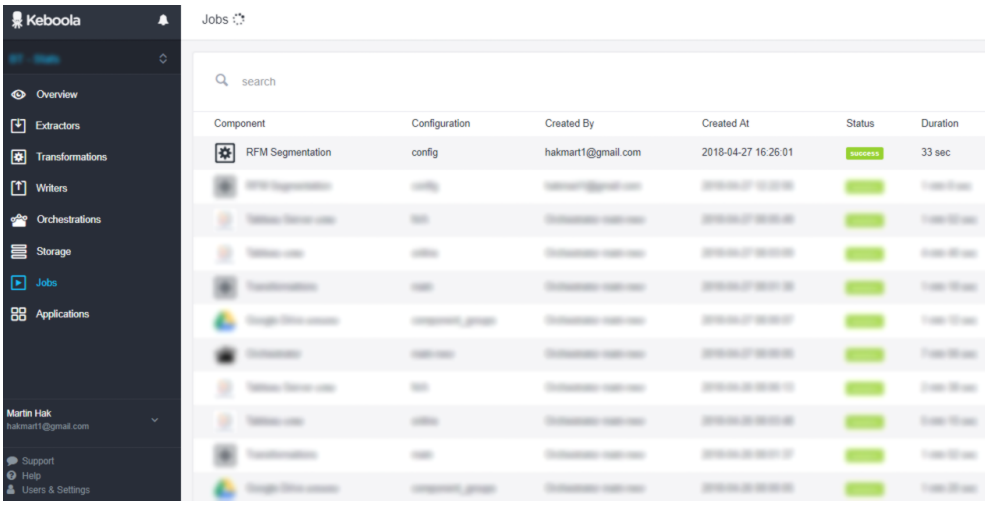
XML Extensible Markup Language

Příloha B

Obrázky



Obrázek B.1: Ukázka UI pro konfiguraci parametrů v aplikace RFM Segmentation v prostředí Keboola Connection



Obrázek B.2: Ukázka úspěšného Jobu aplikace RFM Segmentation v prostředí Keboola Connection

Příloha C

Zdrojové kódy

C.1 JSON Configuration Schema

```
{
  "type":"object",
  "title":"Parameters",
  "required":["method","dateFormat","columns","potential","
    recencySegmentsCount","frequencySegmentsCount","
    monetarySegmentsCount","thresholds"],
  "properties":{
    "method":{
      "type":"string",
      "title":"Method",
      "default":"quantil",
      "enum": ["quantil","gradualQuantil","custom"],
      "propertyOrder":1
    },
    "dateFormat":{
      "type":"string",
      "title":"Date Format",
      "default":"yyyy-MM-dd",
      "minLength":1,
      "propertyOrder":2
    },
    "recencySegmentsCount":{
      "type":"integer",
      "title":"Recency Segments Count",
      "default":3,
      "propertyOrder":3
    },
    "frequencySegmentsCount":{
      "type":"integer",
      "title":"Frequency Segments Count",
      "default":3,
      "minLength":1,
      "propertyOrder":4
    },
```

```
33      "monetarySegmentsCount":{
34        "type":"integer",
35        "title":"Monetary Segments Count",
36        "default":3,
37        "propertyOrder":5
38      },
39      "columns":{
40        "type":"object",
41        "title":"Columns",
42        "required":["entity","price","date","cluster"],
43        "properties":{
44          "date":{
45            "type":"string",
46            "title":"Date",
47            "default":"date",
48            "minLength":1
49          },
50          "price":{
51            "type":"string",
52            "title":"Price",
53            "default":"price",
54            "minLength":1
55          },
56          "entity":{
57            "type":"string",
58            "title":"Entity",
59            "default":"user_id",
60            "minLength":1
61          },
62          "cluster":{
63            "type":"string",
64            "title":"Cluster",
65            "default":"",
66            "minLength":1
67          }
68        },
69        "propertyOrder":6
70      },
71      "potential":{
72        "type":"string",
73        "title":"Calculate Bussiness Potential",
74        "enum": ["Yes","No"],
75        "default":"No",
76        "propertyOrder":7
77      },
78      "thresholds":{
79        "type":"object",
80        "title":"Thresholds",
81        "required":["recency","frequency","monetary"],
82        "properties":{
83          "recency":{
84            "type":"array",
85            "items":{"type":"string"},
86            "title":"Recency"
```

```
87          },
88          "monetary":{
89            "type":"array",
90            "items":{"type":"number"},
91            "title":"Monetary"
92          },
93          "frequency":{
94            "type":"array",
95            "items":{"type":"integer"},
96            "title":"Frequency"
97          }
98        },
99        "propertyOrder":8
100     }
101   }
102 }
```

Listing C.1: JSON Configuration Schema

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se zkompilovanou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Hak_Martin_2018.pdf` ..... text práce ve formátu PDF