Afstudeerverslag

Controle hulp voor ondernemers en adviseurs

Naam: Floris Smit
E-mailadres: 464337@student.saxion.nl
School: Hogeschool Saxion Enschede
Studie: HBO-ICT richting Software Engineering

Student

Naam:	Floris Smit
Hogeschool:	Saxion
Opleiding:	HBO-ICT Software Engineering
Studentnummer:	464337
E-mail:	floris@moneybird.com; floris_smit@outlook.com; 464337@student.saxion.nl

Begeleiding Saxion

Naam:	Eelco Jannink
E-mail:	e.h.a.jannink@saxion.nl

Bedrijf

Bedrijfsnaam:	Moneybird
Website:	https://www.moneybird.com
Straat en huisnummer:	Moutlaan 35
Postcode en plaats:	7523 MC, Enschede
Primair e-mailadres:	support@moneybird.com

Begeleider 1 Moneybird

Naam:	Martijn Sneujink
Functie:	Software & DevOps Engineer
Opleiding(en):	Bachelor of Information and Communication Technology, Computer Science
E-mail:	martijn@moneybird.com

Begeleider 2 Moneybird

Naam:	Nicky ter Maat
Functie:	Software Engineer
Opleiding(en):	Bachelor HBO-ICT Software Engineering – Saxion Enschede
E-mail:	nicky@moneybird.com

Begeleider 3 Moneybird

Naam:	Gijs Abbring
Functie:	Software Engineer
Opleiding(en):	Bachelor HBO-ICT Software Engineering – Saxion Enschede
E-mail:	gijs@moneybird.com

Voorwoord

Dit is het afstudeerverslag getiteld "Controlehulp voor ondernemers en adviseurs", dat ik heb geschreven ter afsluiting van mijn studie HBO-ICT: Software Engineering aan Hogeschool Saxion Enschede.

Tijdens mijn afstudeerperiode bij Moneybird in Enschede van 21 november tot 26 april, heb ik op een onderzoekende manier een Proof of Concept ontwikkeld voor het voorspellen van btw-tarieven in verkoopfacturen.

Ik wil graag mijn begeleiders Martijn Sneujink, Gijs Abbring en Nicky ter Maat bedanken voor hun ondersteuning en begeleiding tijdens dit onderzoek. Ook wil ik mijn afstudeerbegeleider van Saxion, Eelco Jannink, bedanken voor zijn begeleiding en feedback tijdens mijn afstudeerproject.

Floris Smit
Enschede, 5 april 2023

Samenvatting

Moneybird biedt ondernemers een platform voor het doen van hun boekhouding. Bij het ondernemerschap komt veel kijken, zo ook het controleren van hun boekhouding. Hierbij is het belangrijk dat alle fouten geïdentificeerd worden, en vervolgens eruit worden gehaald, om bijvoorbeeld boetes te voorkomen. Uit een gebruikersonderzoek, die is afgenomen bij klanten van Moneybird, bleek dat ondernemers het fijn zouden vinden als ze geholpen worden bij het controleren van hun boekhouding.

Dit afstudeerproject is erop gericht om de ondernemer het controleren van de boekhouding makkelijker te maken. Hierbij lag in de gekozen scope de focus op het automatisch controleren van de door de ondernemer ingevulde btw-tarieven binnen verkoopfacturen. Wanneer een ondernemer bijvoorbeeld een mogelijk foutief btw-tarief invult, zal deze gebruiker een suggestie omvattende melding te ontvangen.

Gedurende het literatuuronderzoek is er gekeken naar mogelijke methodes die op basis van data kunnen voorspellen of een ingevuld btw-tarief klopt of niet. Hieruit bleek dat Machine Learning de beste oplossing hiervoor bood. Echter is er ook gekeken naar statische methodes. Op basis hiervan zijn er een aantal Machine Learning algoritmes vergeleken, en is op basis van een aantal criteria het best aansluitende algoritme gekozen. Er is vervolgens een prototype ontwikkelt die gebruik maakt van dit algoritme en de gebruiker voorziet van een suggestie binnen het Moneybird platform.

Binnen het prototype vallen een aantal producten welke zijn opgeleverd:

- Een externe Flask API voor
    - Het maken van een voorspelling
    - Het verwerken van de administratieve data
    - Het trainen van een Machine Learning model
- Een CI/CD-pipeline voor een Flask applicatie
- Een integratie in Moneybird (MB2) voor het maken van suggesties binnen verkoopfacturen van de ondernemer

Het prototype omvat tests, configuratiebestanden en, indien nodig, documentatie voor geautomatiseerde integratie en implementatie van het product (CI/CD). Dit vormt het geheel van het prototype, maar het kan nog niet worden ingezet in een productieomgeving. Om het prototype te implementeren in een productieomgeving, moeten eerst de aanbevelingen worden afgerond die zijn beschreven.

Inhoudsopgave

Figuren- en tabellenlijst

Lijst met afkortingen

Afkorting	Definitie
API	Application Programming Interface
btw	Belasting over de toegevoegde waarde
CI/CD	Continious Integration-/Deployment
CSV	Comma-seperated values file
GUI	Graphical User Interface
IQR	Interquartile Range
ML	Machine Learning
MVC	Model View Controller
PoC	Proof of Concept
STD	Standard Deviation
SVM	Support vector machines

Begrippenlijst

Begrip	Definitie
Boekhouding	De financiële administratie van een bedrijf
BuildKite	Een cloudplatform voor het beheren van CI/CD-pipelines
Kanban	Een methode voor het managen van projecten binnen de softwareontwikkeling
Inconsistentie	Een waarneming welke afwijkt van het standaard patroon van waarnemingen.
Moneybird Design Systeem	Het interne Design Systeem van Moneybird
Mockdata	Data die niet overeenkomt met de realiteit en welke gebruikt kan worden voor het testen van applicaties tijdens het ontwikkelproces.
Machine Learning	Een methode die gericht is op het bouwen van systemen van die van verwerkte data kunnen leren of data kan gebruiken om beter te presteren
MoSCoW methode	Een wijze van prioriteiten stellen, vaak gebruikt bij het ontwikkelen van Software.
Pickle	Een methode waarmee een Python object omgezet kan worden naar een bytestream
Resultatenrekening	Een overzicht van de kosten en de opbrengsten van een bedrijf in een bepaald jaar.
Stimulus	Een JavaScript framework
Uitschieter	Verstaat men in de statistiek als een waarneming die niet bij de overige waarnemingen lijkt te passen, dus sterk afwijkt van de gemiddelde waarneming.

Tabel 1 Begrippenlijst

1. Inleiding

Moneybird biedt ondernemers en hun adviseurs een platform waarbij gemakkelijk en simpel de boekhouding bijgehouden kan worden. Echter, uit een gebruikersonderzoek binnen Moneybird blijkt dat er vraag is naar een manier om fouten binnen de boekhouding gemakkelijk te kunnen opsporen. Ook blijkt dat er ontevredenheid bestaat over de mogelijkheden die het platform op het gebied van het opsporen van fouten binnen hun boekhouding biedt. Bij fouten binnen een boekhouding kan gedacht worden aan uitschietende bedragen of een transactie die op een verkeerde post geboekt is.

Omdat Moneybird hun gebruikers het zo makkelijk mogelijk wil maken met het bijhouden van hun boekhouding, is er een opdracht geformuleerd om een controle hulpmiddel te laten ontwikkelen die het opsporen van fouten, inconsistenties of uitschieters makkelijker maakt voor de gebruiker. Door software te ontwikkelen die gebruik maakt van bijvoorbeeld: Statistiek, Machine Learning of Neural Networks, kunnen deze fouten opgespoord worden. Hierbij is onderzoek doen naar de verschillende scenario's en de technieken die hier op aansluiten van belang. Ook dient er onderzocht te worden wat de wens van de gebruiker is en in welke vorm men dit terug wil zien. Heeft de gebruiker bijvoorbeeld behoefte aan een uitbreiding op het platform of wil zij juist een aanpassing zien op de mogelijkheden die er al zijn?

Ten slotte is het belangrijk om deze wens te realiseren in de vorm van een werkend prototype binnen het huidige platform van Moneybird, zodat er gemakkelijk mee te werken valt en de fouten worden weergeven aan de gebruiker.

Dit onderzoek dient ertoe om de mogelijkheden van het opsporen van fouten binnen een boekhouding te onderzoeken, het gemakkelijker te maken voor de gebruiker en om dit vervolgens te realiseren binnen het platform.

2. Context

In dit hoofdstuk wordt een korte beschrijving gegeven van Moneybird in het algemeen en zullen de contactgegevens van de afstudeer- en bedrijfsbegeleiders worden opgesomd.

2.1. Moneybird

Moneybird is een middelgroot bedrijf dat al meer dan 10 jaar bestaat. Het bedrijf heeft momenteel meer dan 50 medewerkers en richt zich voornamelijk op het onderhouden en ontwikkelen van hun platform genaamd 'Moneybird'. Dit platform is bedoeld voor ondernemers, met name zzp'ers en mkb'ers, om hun boekhouding bij te houden. Het doel van Moneybird is om ondernemers een eenvoudig te begrijpen en te gebruiken platform te bieden. Daarnaast wil Moneybird het voor ondernemers gemakkelijker maken door een deel van de boekhouding te automatiseren.

2.2. Organisatie

Tijdens de gehele afstudeerperiode zal de student binnen Moneybird begeleid worden door drie begeleiders/collega's en door de andere werknemers. Elke werknemer is bereid hulp te bieden waar nodig en er zullen enkele gesprekken plaatsvinden met andere werknemers als de expertise niet valt binnen de expertise van de begeleiders, bijvoorbeeld met een UI/UX Designer.

Bedrijfsbegeleiders

Moneybird wijst elke afstudeerder een of meer begeleiders aan die in de loop van het afstudeerproject de student begeleiden in het proces. De bedrijfsbegeleiders geven de student gedurende het project terugkoppeling op geschreven code en documentatie, en geven daarnaast ook advies waar nodig. De student zal in samenspraak met de afstudeerbegeleiders keuzes maken die invloed hebben op de weg die het eindproduct inslaat.

Er is afgesproken om elke donderdag een vergadering van een uur te houden met de bedrijfsbegeleiders. Tijdens deze vergadering krijgt de student de gelegenheid om vragen te stellen, feedback te vragen en feedback te ontvangen van de bedrijfsbegeleiders. Hieronder kunnen de contactgegevens van de aangewezen bedrijfsbegeleiders gevonden worden.

Begeleider 1 Moneybird

Naam:	Martijn Sneujink
Functie:	Software & DevOps Engineer
Opleiding(en):	Bachelor of Information and Communication Technology, Computer Science — Saxion Enschede
E-mail:	martijn@moneybird.com

Begeleider 2 Moneybird

Naam:	Nicky ter Maat
Functie:	Software Engineer
Opleiding(en):	Bachelor HBO-ICT Software Engineering – Saxion Enschede
E-mail:	nicky@moneybird.com

Begeleider 3 Moneybird

Naam:	Gijs Abbring
Functie:	Software & DevOps Engineer
Opleiding(en):	Bachelor HBO-ICT Software Engineering – Saxion Enschede
E-mail:	gijs@moneybird.com

Afstudeerbegeleider

Bij Saxion wordt aan elke student een afstudeerbegeleider toegewezen, wiens taak het is om de algemene vragen van de student over het afstudeerproces te beantwoorden en het afstudeerproces te bewaken door maandelijks een vergadering te plannen waarbij de student het voortouw neemt.

Tijdens de maandelijkse vergadering met de afstudeerbegeleider is het de verantwoordelijkheid van de student om belangrijke documentatie aan te leveren. De student zorgt ervoor dat de documentatie tijdig wordt aangeleverd en dat de bedrijfsbegeleiders de documenten van feedback hebben voorzien. Hieronder vindt u de contactgegevens van de afstudeerbegeleider.

Afstudeerbegeleider Saxion

Naam:	Eelco Jannink
E-mail:	e.h.a.jannink@saxion.nl

3. Doelstelling

In dit hoofdstuk worden verschillende onderdelen uitgebreid toegelicht om een duidelijk beeld te krijgen van de probleemstelling van de opdracht en de aanpak hiervan. Allereerst zal de startsituatie en het probleem worden besproken. Vervolgens wordt er dieper ingegaan op de afstudeeropdracht zelf en de risico's die daarmee gepaard gaan. Ten slotte wordt aan het einde van het hoofdstuk de onderzoeksvraag toegelicht.

3.1. Probleemstelling

Dit hoofdstuk beschrijft de probleemstelling van het onderzoek. Allereerst wordt de huidige situatie en de opdracht in het algemeen toegelicht, evenals de afbakening ervan. Vervolgens worden mogelijke risico's, de gevolgen hiervan en de planning gedurende het onderzoek besproken. Tot slot worden de hoofd- en deelvragen opgesomd die zijn opgesteld op basis van de opdracht.

3.1.1. Startsituatie

Op dit moment hebben ondernemers en adviseurs niet de mogelijkheid automatisch inzicht te krijgen in mogelijke fouten binnen de boekhouding, alle fouten moeten dus handmatig gespot worden door de gebruiker zelf. Dit kost niet alleen veel tijd, maar het komt ook voor dat fouten hierdoor over het hoofd worden gezien. Wel is het bijvoorbeeld mogelijk om maandcijfers per kostenpost, overzichtelijk naast elkaar te laten zetten. Dit kan de gebruiker doen door de resultatenrekening te openen en te sorteren op maand. Al is dit een vrij overzichtelijke en makkelijke manier van vergelijken; de gebruiker moet alsnog zelf die vergelijking maken en moet hierbij ook bedenken of er een fout in de cijfers zit. Is er bijvoorbeeld een transactie op een verkeerde kostenpost geboekt, dan geeft de resultatenrekening een scheef beeld.

3.1.2. Opdracht

De opdracht is om het opsporen van fouten zo makkelijk mogelijk te maken voor de gebruiker. Het is zaak om te onderzoeken welke data hiervoor nodig is, welke technieken gebruikt kunnen worden om de data te verwerken en uiteindelijk tot het gewenste resultaat te komen.

Er moet binnen het onderzoek onderscheid gemaakt worden tussen verschillende mogelijke fouten binnen een boekhouding, namelijk: inconsistenties en uitschieters.

Een inconsistentie is een afwijking van de normale trend, bijvoorbeeld wanneer er elk jaar tijdens kerst twee keer zoveel wordt uitgegeven aan uitgavenpost X, maar er in jaar Y vier keer zoveel wordt uitgegeven tijdens de kerstdagen. Een uitschieter is een uitzonderlijk hoog of laag datapunt, bijvoorbeeld een uitgave die in maand X tien keer hoger is dan het gebruikelijke maandbedrag (bijvoorbeeld een telefoonabonnement).

Als onderdeel van het Moneybird platform is het nodig om de gegevens die hieruit voortkomen te visualiseren. Het is daarbij van belang om onderzoek te doen naar de wensen van de Moneybird-gebruikers en hoe zij de visualisatie voor zich zien. Het is mogelijk om feedback te verzamelen door middel van een enquête, maar alleen als deze feedback nog niet beschikbaar is.

3.1.3. Verwacht eindresultaat

In dit hoofdstuk wordt het gewenste eindresultaat toegelicht en welke documenten er uiteindelijk opgeleverd zullen worden. Vervolgens zullen de randvoorwaarden en de opgestelde requirements worden toegelicht. Tot slot komt de risicoanalyse aan bod.

Eindproduct

Er zijn enkele onderdelen die opgeleverd zullen worden aan zowel Moneybird als aan Saxion. De volgende onderdelen zullen aan Moneybird worden opgeleverd:

- Een onderzoeksrapport met de resultaten van het te verrichten onderzoeken.
- Een in Moneybird geïntegreerde interface waarbij rekening is gehouden met de begrijpbaarheid ten aanzien van de gebruiker, en waarbij de eventuele fouten binnen de boekhouding onder de aandacht van de gebruiker worden gebracht.

De volgende onderdelen zullen aan Saxion worden opgeleverd:

- Scriptie
- Reflectie
- Presentatie

Randvoorwaarden

Gedurende het afstudeerproject zijn er enkele voorwaarden die naar verwachting van de student moeten worden vervuld, zodat het project met succes afgerond kan worden. Deze zijn:

- Er moet een werkplek en een laptop beschikbaar gesteld worden door Moneybird.
- De bedrijfsbegeleiders moeten over genoeg kennis beschikken op het gebied van Software Engineering, en dienen tijd vrij te maken voor de student.

Wanneer de student begint met het ontwikkelen van een prototype (*What Is Prototype? | Definition From TechTarget*, n.d.) moet er een dataset beschikbaar zijn die gebruikt kan worden voor het analyseren van data. Dit is nodig om mogelijke fouten binnen de boekhouding van een gebruiker te detecteren.

3.1.4. Mogelijke consequenties

Mocht de opdracht niet in zijn volledigheid uitgevoerd worden, dan is de consequentie hiervan dat de situatie voor gebruikers hetzelfde blijft als eerder. Gebruikers zouden nog steeds handmatig fouten binnen de boekhouding moeten spotten. Idealiter worden deze fouten automatisch weergeven aan de gebruiker.

3.1.5. Afbakening

De opdracht wordt in een periode van 20 schoolweken uitgevoerd door de student. Om te kunnen voldoen aan de opgestelde doelen moet het project afgebakend worden.

Later in dit document worden de opgestelde requirements toegelicht, zie 4.4 Requirements. Hierbij is gebruik gemaakt van de MoSCoW methode. Binnen de 20 weken worden eerst alle requirements met een hoge prioriteit afgerond (Must-haves), voordat er aan de requirements met een lagere prioriteit wordt gewerkt. Ook is het zaak dat alle hoofd- en deelvragen worden beantwoord.

3.1.6. Risicoanalyse

Binnen een project kan iets onverhoopt misgaan. Zo kan bijvoorbeeld door uitval van een bedrijfs- of afstudeerbegeleider het project vertraging oplopen. In tabel 2 worden verschillende risico's toegelicht die van toepassing zijn op dit project.

Risico	Kans	Effect	Oorzaak	Maatregel
Het afvallen van een afstudeer- en/of bedrijfsbegeleider	Klein	Groot	Door bijvoorbeeld ziekte of andere omstandigheid kan een begeleider uitvallen.	Door Moneybird worden twee begeleiders aangesteld. De een kan de ander waar nodig opvangen.
Project complexiteit is te hoog	Klein	Middel	Gebrek aan kennis van bestaande systemen of gebruikte technieken	Vroegtijdig opmerken van het probleem en om hulp vragen
Onderzoek blijkt te 'groot' voor de afstudeerperiode	Middel	Middel	Verkeerde inschatting gemaakt, te brede MVP vastgesteld	Overleggen met de begeleiders hoe dit op te lossen
Te kleine/te weinig zeggende dataset	Klein	Groot	Dataset bevat niet de velden die belangrijk zijn voor het komen tot een goed onderzoeksresultaat	Overleggen met de begeleiders hoe dit op te lossen

Tabel 2 Risicoanalyse

3.1.7. Planning

Binnen het project zal eerst het theoretisch kader worden afgerond, voordat er aan het Proof of Concept (PoC) begonnen wordt. Bij het opstellen van het Plan van Aanpak (PvA) is er ook een globale planning opgesteld waar men zich zo goed mogelijk aan houdt. Deze planning is opgesteld in de vorm van een strokenplanning en is opgenomen in Bijlage 4: Strokenplanning.

3.1.8. Hoofdvraag

Op basis van de opdracht die beschreven is in de vorige hoofdstukken van dit document is er een hoofdvraag opgesteld. De focus zal hierbij liggen op het detecteren van fouten binnen de boekhouding van de gebruiker en op welke wijze deze fouten gepresenteerd kunnen worden aan de eindgebruiker van het Moneybird platform, namelijk de klant.

Hoe kan een systeem ontwikkeld worden om fouten/inconsistenties binnen de boekhouding te detecteren en te presenteren aan de klant?

3.1.9. Deelvragen

Om de hoofdvraag te beantwoorden zijn er verschillende deelvragen geformuleerd. Eerst zal worden onderzocht welke technieken binnen het Moneybird platform relevant zijn voor de ontwikkeling van het PoC. Vervolgens zal worden gekeken naar welke aanwezige data belangrijk is voor het opsporen van fouten in de boekhouding van de eindgebruiker. Tot slot wordt onderzocht welke methoden geschikt zijn voor het presenteren van dergelijke fouten aan de gebruiker. De resultaten van deze deelvragen zijn opgenomen in hoofdstuk 4 van het onderzoek (4. Onderzoek). Indien van toepassing worden experimentele bevindingen verwerkt in Bijlage 1: Onderzoeksrapport.

De volgende deelvragen zijn opgesteld op basis van de opdracht:

- Welke relevante technieken wordt gebruik van gemaakt?
- Welke data is relevant binnen het platform van Moneybird voor het vinden van fouten, inconsistenties en uitschieters binnen de boekhouding van de gebruiker?
- Welke methode is het meest geschikt om uitschieters binnen een dataset te herkennen?
- Welke methode is het meest geschikt om inconsistenties binnen een dataset te herkennen?
- Hoe kan een zo gebruiksvriendelijk mogelijke GUI worden ontwikkeld?
- Hoe kunnen fouten onder de aandacht van gebruikers gebracht worden?

3.2. Aanpak

In dit hoofdstuk wordt de globale aanpak van het onderzoek beschreven. Eerst wordt er ingegaan op het ontwerp van verschillende onderdelen van het product. Vervolgens zal er toegelicht worden welke onderdelen gerealiseerd worden, en hoe de kwaliteit van het product bewaakt wordt.

3.2.1. Ontwerp

De ontwerpen van het project worden gemaakt tijdens de onderzoeks- en realisatiefase. Het kan zo zijn dat de ontwerpen gedurende het onderzoeken veranderen, omdat er nieuwe kennis is opgedaan die de kijk op het ontwerp doet veranderen.

Voor de volgende onderdelen zal een ontwerp gemaakt worden:

- Het model
- De data-set
- De systeemimplementatie
- De GUI (Gebruikersinterface)

De ontwerpen die gemaakt worden, zijn gebaseerd op de opgestelde requirements. De requirements worden opgesteld op basis van het gedane onderzoek. In onderstaande opsomming worden de verschillende onderdelen in het kort toegelicht:

1) Het model: Om fouten te kunnen detecteren binnen de boekhouding van de gebruiker zal er een model ontwikkelt moeten worden die op basis van gebruikersinput een voorspelling kan maken of deze input mogelijk fout is. Hierbij kan bijvoorbeeld gedacht worden aan een Machine Learning (ML) model, of een statistisch model om fouten binnen een administratie van een klant te detecteren.
2) De dataset: Om voorspellingen te kunnen maken binnen de boekhouding van een gebruiker is er data nodig. Welke data hiervoor nodig is, zou moeten blijken uit het onderzoek wat voortkomt uit de beschrijvende deelvragen van het onderzoek. Ook zal de huidige database structuur onderzocht moeten worden om te weten te komen welke data er aanwezig is.
3) De systeemimplementatie: Het is eveneens van belang om het bestaande systeem van Moneybird nauwkeurig in kaart te brengen, bijvoorbeeld aan de hand van een Architectuurplaat. Vervolgens zal het PoC in deze architectuur worden opgenomen om te illustreren hoe het PoC in de huidige systeemarchitectuur kan worden geïmplementeerd.
4) De GUI (Gebruikersinterface): Er wordt tevens een ontwerp gemaakt voor de GUI (Graphical User Interface) van het PoC. Hierbij wordt rekening gehouden met het uiterlijk van het Moneybird-platform en de richtlijnen voor UI/UX-ontwerp (UI vs. UX Design: What's the Difference?, 2022).

3.2.2. Realisatie

Hierna worden de ontwerpen geïmplementeerd in een aantal onderdelen, die samen het PoC vormen. Tijdens het onderzoek kan echter blijken dat sommige ontwerpen of onderdelen niet gerealiseerd kunnen worden, bijvoorbeeld omdat er onvoldoende tijd beschikbaar is door een verkeerde planning.

De volgende onderdelen zullen tijdens het project worden gerealiseerd:

- Model
- Een kwalitatieve dataset
- Systeemimplementatie
- De GUI
- CI/CD-pipeline

De onderdelen worden in volgorde van belangrijkheid gerealiseerd, waarbij het eerste onderdeel als eerste wordt ontwikkeld omdat dit het belangrijkste is, en het laatste onderdeel als laatst wordt afgerond omdat dit het minst belangrijk is.

3.2.3. Kwaliteitsbewaking

De kwaliteit van het project wordt niet alleen door de bedrijfsbegeleiders bewaakt, maar ook deels door het team van collega's. Er zijn enkele afspraken gemaakt, die er voor moeten zorgen dat de student een houvast heeft en zijn of haar progressie laat zien aan het team en diens begeleiders. De gemaakte afspraken worden verder toegelicht in de volgende alinea's.

Wekelijkse afspraak bedrijfsbegeleiders

Tijdens deze afspraak wordt de progressie van het project besproken samen met de student, worden er eventueel vragen gesteld en hebben de bedrijfsbegeleiders de mogelijkheid om de student feedback te geven. Ook buiten deze gesprekken om kan natuurlijk feedback gegeven worden, maar dit gesprek zal in ieder geval altijd plaatsvinden.

Presentatie geven aan het team

Aan het einde van elk project binnen Moneybird geeft elk team een eindpresentatie. De projecten duren zes weken en er is een afkoelperiode van 2 weken tussen elke projectcyclus. De student krijgt ook de mogelijkheid om aan het einde van de afstudeerperiode een presentatie te geven, maar dit gebeurt alleen tijdens de afstudeerverdediging. De student zal dan een oefenpresentatie geven.

4. Onderzoek

De beschreven deelvragen in het vorige hoofdstuk (3. Doelstelling) zullen worden uitgewerkt in dit hoofdstuk. De bevindingen van het onderzoek zijn verwerkt in een apart onderzoeksrapport, dat te vinden is in de bijlagen aan het eind van dit document, zie Bijlage 1: Onderzoeksrapport. In de komende alinea's wordt de aanpak, de resultaten en de conclusie per deelvraag beschreven om het overzicht te behouden. Aan het eind van dit hoofdstuk zullen de deelconclusies samengevoegd worden in de vorm van een algehele conclusie, zie 4.2. Conclusie.

4.1. Hoofd- en deelvragen

Het onderzoek bestaat uit totaal zeven deelvragen. Deze deelvragen zijn zo opgesteld dat de hoofdvraag beantwoord kan worden, nadat er antwoord is verkregen op alle deelvragen. In de komende alinea's worden de deelvragen toegelicht.

4.1.1. Hoofdvraag

Hoe kan een systeem ontwikkeld worden om fouten/inconsistenties binnen de boekhouding te detecteren en te presenteren aan de klant?

4.1.2. Deelvraag 1

Welke relevante technieken gebruikt Moneybird?

Aanpak

Het is van belang om inzicht te krijgen in de gebruikte technieken van Moneybird. Deze informatie is namelijk van invloed op de keuzes die tijdens het afstudeerproject worden gemaakt. Om deze vraag te beantwoorden, wordt gekeken naar onder andere de code in GitHub (GitHub, n.d.) en worden er gesprekken gevoerd met collega's (field research) (*The DOT Framework*, 2021).

Resultaat

Het platform van Moneybird is ontwikkeld met behulp van Ruby on Rails (Ruby Corp, n.d.), een framework voor webapplicaties waarbij de Model-View-Controller-architectuur (MVC) wordt toegepast. Om data te verplaatsen tussen de eindgebruiker van Moneybird en het Moneybird-systeem wordt er gebruik gemaakt van JSON en XML, en voor het webinterface worden HTML, CSS en Stimulus gebruikt (Hotwired, n.d.). PostgreSQL is de database die gebruikt wordt binnen het platform. Om te werken met Ruby zal er voornamelijk gebruik gemaakt worden van de Ruby on Rails console. Dit geeft inzicht in de structuur van class models en de relaties tussen deze models, wat nodig is om te begrijpen hoe de applicatie werkt en welke data relevant is voor het prototype.

Phabricator (*Phabricator*, n.d.) is een tool die gebruikt wordt voor code review en het beheren van software ontwikkelingsprojecten. Het versiebeheer van Moneybird is gekoppeld aan een Git repository in GitHub. Wanneer een medewerker veranderingen in de code aanbrengt, zoals het toevoegen of verwijderen van code, wordt dit zichtbaar in Phabricator. Binnen Phabricator wordt dit een differential (ook wel bekend als een Pull Request) genoemd en kunnen collega's bij Moneybird de code bekijken en beoordelen.

BuildKite (BuildKite, n.d.) is een tool voor Continuous Integration (CI) en Continuous Delivery (CD), wat inhoudt dat het gebruikt wordt voor het automatisch testen, bouwen en deployen van software. Het is een

pipeline die in staat is om code wijzigingen te monitoren en de applicatie op een geautomatiseerde manier te bouwen en testen. Wanneer de code wijzigingen succesvol door de BuildKite pipeline gaan, wordt kan dit gedeployed worden naar de productieomgeving. Het is belangrijk om rekening te houden met deze techniek tijdens de ontwikkelfase van het onderzoek, omdat het prototype geïntegreerd moet worden in de huidige applicatie en daarbij door de BuildKite pipeline moet kunnen gaan. Binnen de pipeline worden een aantal tests opgenomen, die zelf geschreven zijn.

Conclusie
Op basis van de gegeven informatie kan geconcludeerd worden dat Moneybird gebruik maakt van Ruby on Rails als framework voor de ontwikkeling van het platform, waarbij PostgreSQL als database wordt gebruikt en JSON, XML, HTML, CSS en Stimulus voor de verwerking van data en het webinterface. Voor het versiebeheer van de code wordt er gebruik gemaakt van een Git-repository in GitHub, terwijl Phabricator wordt ingezet voor code review en software-ontwikkelingsprojectbeheer. Daarnaast wordt BuildKite gebruikt voor Continuous Integration (CI) en Continuous Delivery (CD), om de applicatie te testen, bouwen en deployen naar de productieomgeving. Het is belangrijk kennis te krijgen van deze technologieën tijdens de ontwikkelfase van het prototype om ervoor te zorgen dat het prototype geïntegreerd kan worden in de huidige applicatie.

4.1.3. Deelvraag 2
Welke data is relevant binnen het platform van Moneybird voor het vinden van fouten, inconsistenties en uitschieters binnen de boekhouding van de gebruiker?

Aanpak
Om deze vraag te beantwoorden is het van belang om eerder uitgevoerde onderzoeken te onderzoeken (door middel van literatuuronderzoek) (The DOT Framework, 2021), zodat relevante informatie kan worden onderscheiden van informatie die geen invloed heeft op de resultaten. Eerst zal er daarom literatuuronderzoek worden uitgevoerd, waarna experimenteel onderzoek zal volgen (The DOT Framework, 2021) om de accuraatheid van de geclassificeerde informatie te beoordelen.

Resultaat
Om een beter beeld te krijgen welke data binnen het Moneybird platform relevant is voor het onderzoek, zal er in dit hoofdstuk eerst gekeken worden naar het gebruikersonderzoek, waarna er gekeken wordt naar het selecteren van geschikte velden binnen deze data. Het gebruikersonderzoek, waarnaar verwezen wordt, mag niet gedeeld worden als bijlage in het afstudeerverslag. Wel zijn er een aantal speerpunten uit het onderzoek gehaald, die in de volgende alinea worden toegelicht.

Gebruikersonderzoek
Voor het gebruikersonderzoek is onderzocht wat de wensen van klanten zijn (Moneybird heeft zelf het gebruikersonderzoek uitgevoerd voordat het afstudeeronderzoek van start ging). Klanten werd gevraagd welke functionaliteit zij nog missen binnen het platform van Moneybird. Dit resulteerde in een lijst van gebruikerswensen die ontbraken of niet voldoende waren opgenomen in het platform. Enkele van deze wensen zijn relevant voor dit onderzoek, namelijk:

- Wanneer ik mijn boekhouding controleer, dan wil ik fouten opsporen, zodat ik deze eruit kan halen.
- Wanneer ik belangrijke keuzes wil maken voor mijn onderneming, dan wil ik gebruik maken van complete en betrouwbare informatie, zodat ik de juiste keuzes maak.

- Wanneer processen in Moneybird automatisch gaan, dan wil ik weten wat er gebeurd is, zodat ik het kan controleren.

Elk van deze items heeft iets te maken met het automatisch opsporen van fouten, zodat de ondernemer werkt met complete en betrouwbare informatie, dus niet met ‘foute’ informatie.

Gebruikers gaven aan het vooral belangrijk te vinden makkelijker fouten te kunnen opsporen, waar ze uiteindelijk zelf de doorslag geven of iets klopt of niet. Zo geeft gebruiker X aan:

“Ja zeker, mooi hoor die techniek, maar ik wil weten hoe dat gaat en evt. Fouten wil ik zelf de hand in hebben”

Ook gaven gebruikers aan dat ze de boekhouding wel controleren, maar dat ze dit niet wekelijks doen en enkele gebruikers doen dit op basis van een steekproef. Enkele quotes uit het gebruikersonderzoek;

“Ik ben ook altijd genoeg om het btw-rapport zelf even na te rekenen”
“Ik controleer nu op basis van een steekproef”
“Elke maand doe ik wel even een controle”

Hieruit valt te concluderen dat gebruikers niet altijd alle transacties even goed controleren, en dat is jammer. Wanneer er bijvoorbeeld teveel betaald is aan een klant of aan de belastingdienst (in de vorm van btw), kost dit de ondernemer onnodig veel geld. Hierin zijn de ondervraagden heel duidelijk, een hulpmiddel die bijvoorbeeld een kostenpost highlight in de resultatenrekening die niet lijkt te kloppen is iets wat ze wel zien zitten. Ook wanneer een transactie gekoppeld wordt aan een categorie zou een waarschuwing van pas komen, bijvoorbeeld wanneer het verkeerde btw-percentage geboekt wordt.

Dit onderzoek zal zich focussen op het waarschuwen van gebruikers als een transactie geboekt is op een mogelijk foutief btw-percentage. Wanneer bijvoorbeeld een aankoop bij de fruitwinkel consistent geboekt wordt op 9%, dan moet er bij een btw-percentage van 21% een belletje gaan rinkelen.

Dit feit kan gecontroleerd worden wanneer een gebruiker een verkoopfactuur aanmaakt en een categorie en btw-percentage invult. Hier zal dan ook op gefocust worden in de loop van het onderzoek.

Verkoopfactuur (Sales Invoice)
Een verkoopfactuur is een overzicht van geleverde diensten of goederen, met de prijs en het btw-percentage wat je aan de klant verrekend (zzpdaily, 2022). Een verkoopfactuur wordt binnen het platform "SalesInvoice" genoemd.

Inkoopfactuur (Purchase Invoice)
Een inkoopfactuur is precies het tegenovergestelde van een verkoopfactuur. Een inkoopfactuur is een overzicht van ingekochte diensten en/of goederen van een externe partij (zzpdaily, n.d.). Een inkoopfactuur wordt binnen het platform "PurchaseInvoice" genoemd.

Bonnetje (Receipt)
Een bonnetje zullen de meesten mensen wel kennen, namelijk in de vorm van een betaalbewijs die hij/zij krijgt bij het doen van aankopen in een reguliere winkel. Wanneer een ondernemer iets aankoopt bij

bijvoorbeeld een fysieke groothandel, zal deze transactie verwerkt worden inclusief een foto/scan van het bonnetje. Binnen het platform wordt het verwerken van een transactie met een bonnetje, een "Receipt" genoemd.

Memoriaal boeking (General Journal Document Entry)
Een memoriaal boeking is een transactie die vrij incidenteel nodig is. Het gaat vaak om een boeking waar geen financiële transactie voor nodig is. Een memoriaal boeking kan bijvoorbeeld gebruikt worden voor het afschrijven van de waarde van een product binnen de bedrijfsinventaris (zzpdaily, 2016). Binnen het platform wordt een memoriaal boeking een "General Journal Document Entry" genoemd. Een memoriaal boeking is in de basis het handmatig invoeren van een Journaalpost, hier wordt later in dit document dieper op ingegaan.

Product (Product)
Een product is precies wat het zegt, namelijk iets wat een ondernemer verkoopt of aankoopt. Iets wat een ondernemer vaker verkoopt, kan een product voor worden aangemaakt. Dit product heeft dan een vaste prijs en een vastgesteld btw-percentage, om het de ondernemer makkelijker te maken wanneer hij/zij weer hetgeen verkoopt.

Betaling (Payment)
Een betaling in de simpelste vorm is bijvoorbeeld een overboeking die een klant heeft gemaakt van de privérekening naar de betreffende zakelijke rekening. Een betaling kan aan een categorie/grootboekrekening gekoppeld worden, maar er wordt vaak geen document toegevoegd aan de transactie. De betalingstransactie wordt niet meegenomen in dit onderzoek en hier zal dus ook niet verder op worden ingegaan.

Journaalposten (Journal Entries)
De gemene deler van de bovenstaande transacties is, dat al deze transacties in de basis journaalposten (Kolar, 2021) zijn. Het inboeken van een transactie doe je door middel van een journaalpost, en bestaat altijd minstens uit twee regels. Dat is omdat een journaalpost altijd in evenwicht moet zijn, en bestaat uit een debet kant en een credit kant. Dit verschijnsel wordt ook wel dubbel boekhouden genoemd (Informer, 2022).

Elk van deze regels worden geboekt op een grootboekrekening. Een grootboekrekening is een verzameling van alle inkomsten en uitgaven binnen een bepaalde categorie. Binnen het platform van Moneybird worden grootboekrekening categorieën genoemd, terwijl in de ontwikkelomgeving over een "LedgerAccount" wordt gesproken; de letterlijke vertaling van een grootboekrekening (boekhouders.nl, 2022).

Om dit verhaal wat duidelijker te maken, wordt in onderstaande alinea een voorbeeld gegeven:

"Persoon X heeft een bedrijf dat Post-its verkoopt aan klanten door heel Nederland. Klant Y is geïnteresseerd en koopt tien pakjes Post-its ter waarde van 100 euro. Bovenop deze prijs geldt een btw-tarief van 21%. Persoon X stelt netjes een verkoopfactuur op en verstuurd deze naar Klant Y."

Wanneer de verkoopfactuur verstuurd is, zullen er op de achtergrond een aantal journaalposten worden opgesteld. In onderstaande tabel zijn de journaalposten opgesteld. Hierin valt te zien dat de debet en

credit kant in evenwicht zijn, en elke journaalpost op een categorie/grootboekrekening wordt geboekt. Wanneer de factuur betaald wordt door Klant Y, zal er weer een journaalpost aangemaakt worden, echter is dit niet meegenomen in het onderstaande voorbeeld.

Grootboekrekening/Categorie	**Debet**	**Credit**
Debiteuren	€ 121	
Omzet Post-its		€ 100
Te betalen btw		€ 21
Totaal	**€ 121**	**€ 121**

Tabel 3 Memoriaal boeking voorbeeld

Voor het ontwikkelen van het prototype kan er dus in zijn algemeenheid gekeken worden naar journaalposten of naar de factuurregels. Hiermee worden alle transacties gedekt, en kunnen er fouten binnen deze journaalposten/factuurregels gespot worden. Hierbij zal de focus liggen op het detecteren van fouten binnen een verkoopfactuur, omdat gebruikers het meeste invloed hebben op het opstellen van de facturen binnen hun bedrijf. Figuur 0 visualiseert de basisstappen van het proces voor het aanmaken van een verkoopfactuur. Hoewel er veel optionele opties zijn bij het maken van een verkoopfactuur, zijn deze niet opgenomen in de Flowchart. Alleen de essentiële stappen voor het maken van een verkoopfactuur zijn weergegeven.

Figuur 0: Het aanmaken van een verkoopfactuur

Om te begrijpen hoe een dergelijke transactie binnen Moneybird eruitziet, en welke velden van belang zijn, moet er gekeken worden naar de broncode. Hiervoor is er gebruik gemaakt van RubyMine (*RubyMine: The Ruby on Rails IDE by JetBrains*, n.d.) om door de code heen te spitten en is er gebruik gemaakt van het Ruby on Rails Console (Ruby Corp, n.d.) om de relaties tussen modellen in kaart te brengen. Hierbij is alleen gekeken naar de eerder beschreven relevante relaties.

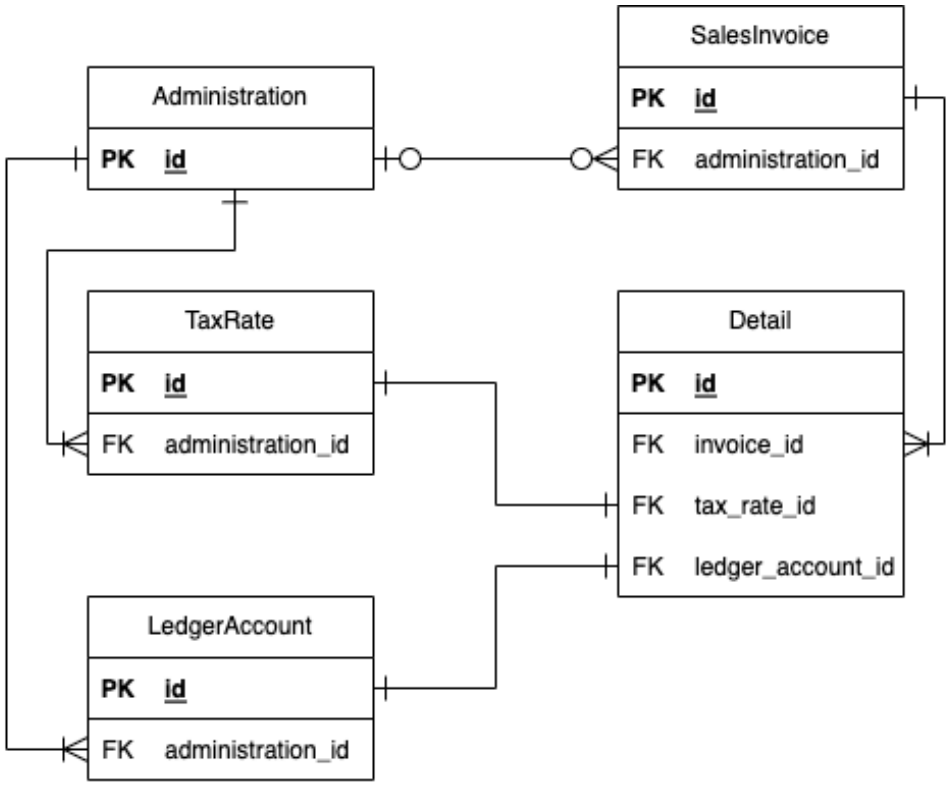
Benodigde data

De scope richt zich op het voorspellen van het juiste btw-percentage op individuele factuurregels van een verkoopfactuur. Om voorspellingen te kunnen maken, is de volgende nodig benodigd:

- De factuurregels (Details) per categorie (Ledger Account), binnen dezelfde administratie
- Het bedrag per factuurregel, en het totaalbedrag per verkoopfactuur (Sales Invoice)
- Het btw-percentage (Tax Rate) waarop de factuurregel geboekt is

Eerst is het belangrijk om te weten welke soorten transacties er binnen het platform bestaan en welk van deze transacties gebruikt zullen worden binnen het ontwikkelen van het prototype. Ook geld dat voor elke soort transactie in het systeem een Engelse benaming heeft, waarbij in de documentatie wel casussen worden toegelicht aan de hand van de Nederlandse termen. In de komende alinea's worden de verschillende transacties toegelicht. Vervolgens zal er een afbakening worden gemaakt, en tot slot zal de structuur van een transactie worden gevisualiseerd in de vorm van een Class Diagram (*UML Class Diagram Tutorial*, n.d.), zie Figuur 1.

De onderstaande afbeelding is een visuele weergave van de structuur van een transactie in de vorm van een Class Diagram. In onderstaand voorbeeld wordt er gekeken naar een verkoopfactuur (SalesInvoice), maar zou vervangen kunnen worden met andere transacties zoals een inkoopfactuur.



Figuur 1 Entity Relation Diagram

Op basis van de Class Diagram in Figuur 1 is er een tabel opgesteld (zie Tabel 4) met alle relevante velden binnen de klasse. Deze velden zijn handmatig geselecteerd. Velden die vooraf al irrelevant zijn, zijn verwijderd uit de tabel. Dit is gebaseerd op veronderstellingen en kennis van het systeem. Bijvoorbeeld, logischerwijs weet men dat de exacte tijd en datum van een verkoopfactuur geen invloed heeft op het btw-tarief dat is geselecteerd voor een factuurregel.

Class/parameter	Datatype
SalesInvoice (verkoopfactuur)	Object
id	Int64
contact_id	Int64
Detail (factuurregel)	Object
id	Int64
invoice_id	Int64
tax_rate_id	Int64
ledger_account_id	Int64
price	Float
Ledger Account (categorie/grootboekrekening)	Object
id	Int64
Tax Rate (btw categorie)	Object
id	Int64
name	String

Tabel 4 Data properties

Conclusie
Het is belangrijk om te weten welke soorten transacties er bestaan en welke van deze transacties gebruikt worden bij het ontwikkelen van het prototype. De verschillende transacties zijn verkoopfactuur, inkoopfactuur, bonnetje, memoriaal boeking, product, betaling en journaalposten. Elk van deze transacties is in de basis een journaalpost en bestaat altijd uit minstens twee regels, namelijk een debet- en een creditkant. Een journaalpost wordt geboekt op een grootboekrekening, die binnen het platform van Moneybird categorieën worden genoemd.

De classes zoals die gevisualiseerd zijn in Figuur 1 zijn relevant voor het detecteren van fouten, inconsistenties en uitschieters binnen de boekhouding van een gebruiker, in het specifiek voor het vinden van fouten in de verkoopfacturen van de gebruiker. De velden die de data bevat zijn handmatig uitgekozen, en zullen gebruikt worden bij het maken van voorspellingen gedurende de ontwikkelfase. Bij het ontwikkelen van het prototype zal de focus vooral liggen op het voorspellen van een mogelijk incorrect btw-percentage binnen een verkoopfactuur van de gebruiker, aangezien dit een transactie is waar de gebruiker veel invloed op heeft en is daarmee ook een foutgevoelige transactie.

4.1.4. Deelvraag 3

Welke methode is het meest geschikt om uitschieters binnen een dataset te herkennen?

Aanpak

Om deze deelvraag te beantwoorden, zal voornamelijk literatuuronderzoek worden uitgevoerd, met behulp van The DOT Framework. Het DOT-framework is een set met onderzoeksmethoden voor praktijkonderzoek binnen de ICT-sector. Eerdere onderzoeken zullen worden geraadpleegd om de meest geschikte methoden voor het detecteren van uitschieters te identificeren. Vervolgens zal experimenteel onderzoek worden uitgevoerd om deze methodiek toe te passen (The DOT Framework, 2021).

Resultaat

Een uitschieter is een waarde binnen een dataset die aanzienlijk afwijkt van de andere waarden. Hoewel het opsporen van een uitschieter in een tabel relatief eenvoudig kan zijn, is er altijd een risico dat er een over het hoofd wordt gezien, vooral bij grote datasets met meer dan 100 rijen. Om dit te voorkomen, kunnen verschillende technieken worden toegepast, zoals statistische methoden en moderne algoritmes, om uitschieters binnen een dataset te detecteren.

Om de beste methode te vinden, zal er literatuuronderzoek onderzoek gedaan worden naar een aantal methodes, waarvan bekend is dat deze geschikt zijn voor het vinden van uitschieters binnen een dataset (C, 2022). Er zijn een tal aan methodes om uitschieters te herkennen, waarvan er drie uitgekozen zijn op basis van populariteit, maar ook de prestatie van deze methodes bij het vinden van uitschieters (Myers, 2019). Er zijn een stuk meer methodes waarmee uitschieters herkent kunnen worden, echter kunnen niet al deze methodes uitgewerkt worden omwille van de tijd. Het gaat om de volgende methodes:

- Standard Deviation of in het Nederlands; Normaalverdeling
- Interquartile Range (IQR)
- Isolation Forest

Vervolgens zullen de bovengenoemde methoden beoordeeld worden op diverse criteria, waaronder efficiëntie, nauwkeurigheid en de prestaties op verschillende datasets. Ook worden de verschillende voor- en nadelen van elke methode samengevat. Alle methoden worden in Ruby uitgewerkt, tenzij dit niet mogelijk blijkt te zijn, in welk geval ze ook in Python worden geïmplementeerd. Python is namelijk de meest gebruikte en populaire programmeertaal voor Data Science, wat betekent dat er veel documentatie en library's beschikbaar zijn om mee te werken (Sharma, 2022). De technische uitwerking van de methoden is te vinden in Bijlage 1: Onderzoeksrapport.

Conclusie
Uit het experimenteel onderzoek (Bijlage 1: Onderzoeksrapport) kunnen een aantal conclusies getrokken worden die in dit hoofdstuk worden beschreven. Om uiteindelijk tot een keuze te komen, is een zogenaamde Beslissing Matrix opgesteld. In een Beslissing Matrix worden een aantal criteria opgesteld, en wordt elke methode getoetst op deze criteria. De volgende criteria zijn opgesteld, waarbij de snelheid en de accuraatheid het zwaarst wegen en de schaalbaarheid en de complexiteit en minst zwaar:

- Snelheid van de methode (1 = relatief traag, 2 = snel afhankelijk van het scenario, 3 = relatief snel)
- Accuraatheid (1 = Slechts accuraat in specifieke scenario's, 2 = Accuraat in de meeste scenario's, 3 = Accuraat en aanpasbaar op basis van het scenario)
- Schaalbaarheid (1 = houdt rekening met één veld binnen een data-set, 2 = kijkt naar twee of meer velden binnen een data-set)
- Complexiteit (1 =Vergt de nodige kennis en kost veel tijd om te ontwikkelen, 2 = Simpel te begrijpen en snel te ontwikkelen)

De criteria zijn gebaseerd op bevindingen uit Bijlage 1: Onderzoeksrapport. De puntenverdeling is bepaald aan de hand van het onderzoek dat in deze bijlage is beschreven. Zoals te zien is in de Beslissingsmatrix (Tabel 5) op de volgende pagina, wordt de grootste waarde gehecht aan zowel de snelheid als de accuraatheid van de methode. Deze criteria zijn essentieel, omdat het van belang is dat de eindgebruiker snel en correcte meldingen ontvangt. Het is belangrijk om de gebruiker niet onnodig te belasten met foutmeldingen die eigenlijk geen fouten zijn.

Op basis van de bovengenoemde matrix blijkt dat Isolation Forest de beste methode is om uitschieters binnen een dataset te detecteren. Isolation Forest scoort hoog op de meest belangrijke criteria, snelheid en accuraatheid, en doet het ook goed op de andere criteria. Echter, heeft Isolation Forest ook een groot nadeel: het is een Unsupervised Machine Learning-algoritme (Sklearn.ensemble.IsolationForest - Scikit-Learn 1.2.2 Documentatie, n.d.). Dit betekent dat de accuraatheid van Isolation Forest moeilijk te meten is en voorspellingen niet gemaakt kunnen worden omdat de dataset niet kan worden getraind op basis van bestaande gegevens (What Is Unsupervised ML?, n.d.). Bij het voorspellen van het btw-percentage binnen een factuurregel is het belangrijk om te kunnen vertrouwen op de nauwkeurigheid van de voorspellingen. Het feit dat de accuraatheid van Isolation Forest moeilijk te meten is, maakt het ongeschikt voor implementatie in dit geval.

De andere methoden, Standard Deviation en IQR, zijn geschikt voor het detecteren van uitschieters in de dataset, maar kunnen geen voorspellingen maken op basis van meerdere velden. Bovendien kunnen deze methoden geen trends detecteren als de dataset meerdere velden bevat. Dit maakt deze methoden ook ongeschikt voor de beoogde casus van het voorspellen van het btw-percentage binnen een factuurregel.

Geen van deze methoden zal worden gebruikt bij de implementatie, omdat het detecteren van uitschieters alleen niet voldoende is om het juiste btw-percentage per individuele factuurregel te voorspellen.

Criteria → Methode	Snelheid — 3	Accuraatheid — 3	Schaalbaarheid — 2	Complexiteit — 2	**Totalen**
Standard Deviation	3 — Big O(n)	1 — Alleen accuraat wanneer er 'weinig' uitschieters in de data-set zitten.	1 — Kijkt maar naar één veld binnen een dataset	2 — Heel veel documentatie aanwezig, goed te begrijpen en snel te ontwikkelen.	7
Interquartile Range	1 – Big O(nlogn)	2 — Accuraat in de meeste gevallen, maar niet aanpasbaar	1 — Kijkt maar naar één veld binnen een dataset	2 — Heel veel documentatie aanwezig, goed te begrijpen en snel te ontwikkelen.	6
Isolation Forest	2 –Big O(n), alleen trager dan de andere methodes bij meer velden in de data-set	3 — Zeer accuraat en de mogelijkheid tot Hyperparameter tuning o.b.v. het scenario.	2 — Kijkt naar alle velden binnen een dataset.	1 — Vergt de nodige kennis over Hyperparameter tuning en afhankelijkheid van library's.	8

Tabel 5 Beslissings-/Decision Matrix

4.1.5. Deelvraag 4

Welke methode is het meest geschikt om inconsistenties binnen een dataset te herkennen?

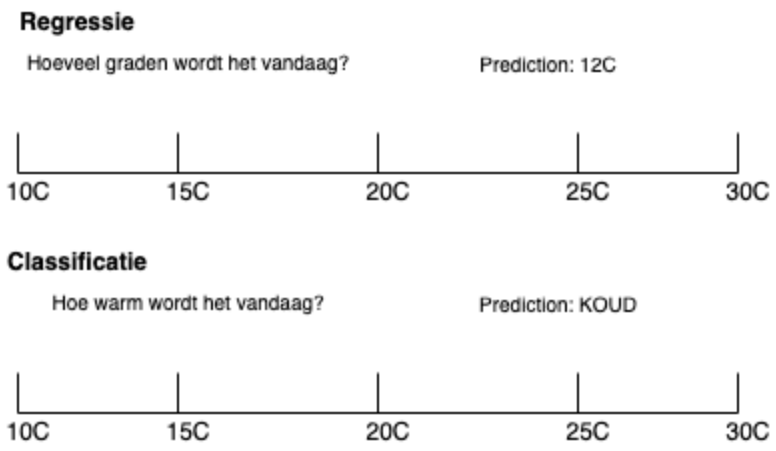
Aanpak

Voor het beantwoorden van deze vraag zal uitgebreid literatuuronderzoek worden uitgevoerd, en zal er ook experimenteel onderzoek worden verricht met behulp van een specifieke methodiek die geschikt is gebleken voor het detecteren van inconsistenties en het verwerken van de data. De redenen voor deze keuze worden uitgelegd in Bijlage 1: Onderzoeksrapport.

Resultaat

Om inconsistenties binnen een dataset te herkennen en te voorkomen, is het belangrijk om de basis terminologie te begrijpen en te weten wat als een inconsistentie wordt beschouwd. Een inconsistentie kan gezien worden als een mogelijke fout binnen data, waar de data niet overeenkomt met de andere data binnen een dataset of niet aan de verwachting van een systeem voldoet. Een voorbeeld hiervan is het btw-percentage op de in- en verkoop van fruit, dat door de overheid is vastgesteld op 6%. Als een dergelijke transactie in Moneybird op een ander btw-percentage dan 6% wordt geboekt, is dit een inconsistente transactie die invloed kan hebben op de btw-aangifte van de ondernemer.

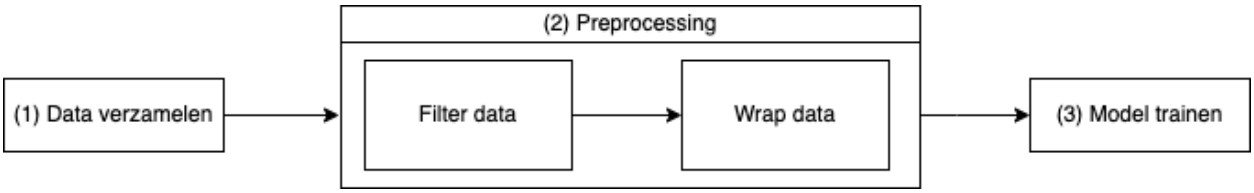
Figuur 2 Regressie vs Classificatie



Er zijn vele manieren om inconsistenties binnen een dataset te herkennen. Deze manieren vallen onder de terminologie van Data Classificatie (De Groot, 2023). Als het gegeven scenario/probleem lineair is, is het gebruik van bijvoorbeeld SVM of Logistic Regression de beste optie. Als het gegeven scenario/probleem non-lineair is, is het gebruik van Classificatie Algoritmes zoals Naive Bays en/of Random Forest de beste optie. Problemen die gebaseerd zijn op categorieën, zijn non-lineair, zoals het scenario dat in deze tekst wordt behandeld (*Regression Vs. Classification in Machine Learning: What's the Difference?*, 2021). Het verschil tussen Regressie en Classificatie is gevisualiseerd in Figuur 2 op de vorige pagina.

Voordat de data kan worden gebruikt voor bijvoorbeeld het opstellen van een verkoopfactuur en het trainen van een model, moet het worden verwerkt en gepreprocessed. Dit betekent dat de data accuraat, betrouwbaar en consistent moet zijn. Blanco velden binnen data moeten worden verwijderd of ingevuld, en onbelangrijke parameters moeten worden verwijderd. Preprocessing (*What Is Data Preprocessing? (With Importance and Examples)*, 2023) verbetert de accuraatheid, betrouwbaarheid en consistentie van de data en zorgt ervoor dat modellen de data makkelijker kunnen begrijpen en tot een beter resultaat kunnen komen

Wanneer de data dit proces heeft doorlopen, kan er een model getraind worden. Dit model kan vervolgens voorspellingen maken op basis van nieuwe input van de gebruiker. Dit gehele proces is opgenomen in onderstaand diagram.



Figuur 3 Preprocessing

Wanneer de data is geprocessed, kan er een model worden getraind. Dit model kan vervolgens voorspellingen maken op basis van nieuwe input van de gebruiker. Het trainen van het model is opgenomen in Figuur 3, waarbij de data wordt verzameld, gepreprocessed en getraind.

Tot slot, is het essentieel om een model te trainen dat een betrouwbare nauwkeurigheid heeft. Hoewel de vereiste nauwkeurigheid van een model subjectief is, wordt doorgaans een nauwkeurigheid tussen 70% en 90% als betrouwbaar beschouwd (Barkved, 2022). Dit gegeven moet in overweging worden genomen bij de ontwikkeling van het Proof of Concept (PoC). Als de nauwkeurigheid van het model lager is dan dit percentage, moet er geen melding aan de gebruiker worden weergegeven.

Conclusie
Uit het experimenteel onderzoek (Deelvraag 4) zijn een aantal conclusies te trekken die in dit hoofdstuk worden beschreven. Tijdens het experimentele onderzoek zijn er een aantal preprocessing en modeltraining methodes uitgewerkt. Hieruit bleek, na de resultaten te hebben vergeleken, dat er een groot verschil bestaat tussen methodes die gebruik maken van Regressieanalyse en/of standaard Classificatie. Bij een model wat gebruikt maakt van Regressieanalyse moet de "target" feature, dus de te voorspellen feature, een continue waarde hebben. Bij een model wat gebruik maakt van de standaard Classificatie moet de "target" feature discrete waardes bevatten om tot een resultaat te komen.

In het beoogde scenario, waarbij het btw-percentage binnen een factuurregel voorspeld wordt, zal er dus enkel gebruik gemaakt worden van Classificatie modellen en kunnen de uitgewerkte Regressiemodellen worden weggestreept. De Classificatiemodellen die uitgewerkt zijn kunnen vergeleken worden op basis van de F1-score (*Chng, 2021*). Hieruit (Deelvraag 4) bleek dat dat de hoogst scorende methode de Random Forest Classifier methode is, dit geldt zowel voor het preprocessen van de data als voor het trainen van het model. Deze methode zal dus ook gebruikt worden voor het ontwikkelen van het PoC.

4.1.6. Deelvraag 5

Hoe kan een zo gebruiksvriendelijk mogelijke GUI worden ontwikkeld?

Aanpak

Om deze deelvraag te beantwoorden zullen er interviews plaatsvinden met een van de designers binnen Moneybird, maar zal het gedane gebruikersonderzoek ook bestudeerd worden.

Resultaat

Op 3 februari 2023 is er een interview afgenomen met collega's Tim en Marijn. Het interview betrof een ongestructureerd gesprek waarbij de manier van ontwerpen binnen Moneybird besproken werd, zie Interview A: 3 februari 2023. De informatie die relevant is voor het beantwoorden van deze deelvraag zal besproken worden in de komende alinea's.

Binnen Moneybird wordt er gebruik gemaakt van een zogenaamd Design Systeem. Met een Design Systeem wordt er een set van standaarden ontwikkelt, die gebruikt kunnen worden door elke collega binnen Moneybird. Het systeem bestaat uit herbruikbare componenten, die de software binnen Moneybird zo consistent en gebruiksvriendelijk mogelijk houdt.

Wanneer er een nieuw stukje software wordt ontwikkeld, zullen er alleen componenten worden gebruikt die beschikbaar zijn binnen het Design Systeem. Echter, is het Design Systeem niet direct geïntroduceerd bij het ontstaan van Moneybird en dus zijn niet alle stukken software ontwikkelt met behulp van dit systeem. Wanneer er aanpassingen worden gedaan aan stukken van het programma welke nog niet gebruik maken van het Design Systeem, zullen deze stukken idealiter worden gemigreerd.

De bestaande componenten binnen het Design Systeem zijn ontwikkeld door collega's binnen het Product team. De componenten zijn ontwikkeld om Moneybird zo simpel, consistent en gebruiksvriendelijk mogelijk te maken voor de eindgebruiker. In het verdere verloop van dit onderzoek is het belangrijk om te bepalen hoe het ontwerp eruit moet zien, en of er behoefte is aan de ontwikkeling van een nieuw component of dat een bestaand component voldoende is.

Om dit te bepalen is er een stappenplan ontwikkelt, die doorlopen kan worden om tot een conclusie te komen. Dit stappenplan is in Bijlage 2: Design System Component Diagram opgenomen in de vorm van een Flowchart. Met behulp van deze Flowchart kan bepaald worden of er een nieuw component nodig is, of een bestaand component voldoende is. Wanneer er een nieuw component nodig is, zal er een voorstel gemaakt worden die vervolgens gedeeld wordt met één van de collega's binnen Moneybird die hierover gaat. In dit voorstel zal de noodzaak van het component toegelicht worden, evenals de specificaties. Tevens zal worden aangegeven of het component ook op andere plaatsen in het systeem bruikbaar is.

Conclusie
Binnen Moneybird wordt er gebruik gemaakt van een Design Systeem, dat bestaat uit een set van standaarden en herbruikbare componenten die de software consistent en gebruiksvriendelijk houden. Wanneer er nieuwe software wordt ontwikkeld, worden alleen componenten uit het Design Systeem gebruikt. Echter, niet alle software is ontwikkeld met behulp van dit systeem, en daarom worden aanpassingen gemaakt aan de bestaande software om deze te migreren naar het Design Systeem.

De componenten binnen het Design Systeem zijn ontwikkeld door het Product team en zijn gericht op het verbeteren van de gebruikerservaring en consistentie van de applicatie. Bij het ontwerpen van nieuwe software of het aanpassen van bestaande software, wordt er gebruik gemaakt van een stappenplan in de vorm van een Flowchart. Dit stappenplan bepaalt of er een nieuw component nodig is of dat een bestaand component voldoende is. Wanneer een nieuw component nodig is, wordt er een voorstel gedaan en besproken met de relevante collega's.

In het verdere verloop van dit onderzoek zal er meer nadruk gelegd worden op het bepalen van het ontwerp en de noodzaak voor nieuwe componenten. Het Design Systeem is een belangrijk onderdeel van de manier van werken binnen Moneybird en speelt een belangrijke rol in het ontwikkelen van een consistent en gebruiksvriendelijk product.

4.1.7. Deelvraag 6

Hoe kunnen fouten onder de aandacht van de gebruikers gebracht worden?

Aanpak

Ook voor deze deelvraag geldt dat er een of meerdere interviews zullen plaatsvinden met de designers van Moneybird om informatie te verkrijgen. Maar er zal ook gekeken worden naar wat de gebruiker belangrijk vindt, dit zal gedaan worden aan de hand van het bestaande gebruikersonderzoek.

Resultaat

In dit hoofdstuk wordt eerst besproken hoe fouten in het algemeen onder de aandacht worden gebracht, voordat later in dit hoofdstuk wordt ingegaan op het onder de aandacht brengen van fouten binnen Moneybird.

Bij het signaleren van fouten die door de gebruiker zijn gemaakt, worden doorgaans enkele basisregels gehanteerd om de gebruiksvriendelijkheid te waarborgen. Deze regels zijn bedoeld om de gebruiker in staat te stellen de gemaakte fout te begrijpen en het zo eenvoudig mogelijk te maken voor de gebruiker. De volgende regels zijn hierbij van belang (*Duffy, n.d.*):

- Wees duidelijk in de berichtgeving naar de gebruiker en vermijd technische termen.
- Houdt het bericht kort maar duidelijk.
- Geef de gebruikers een oplossing wanneer het om een oplosbare fout gaat, bijvoorbeeld in het geval van het gebruik van een verkeerd postcode formaat.
- Zorg ervoor dat het bericht dusdanig gepositioneerd is dat de gebruiker het direct opvalt, bijvoorbeeld door het bericht te plaatsen waar de fout gemaakt is.

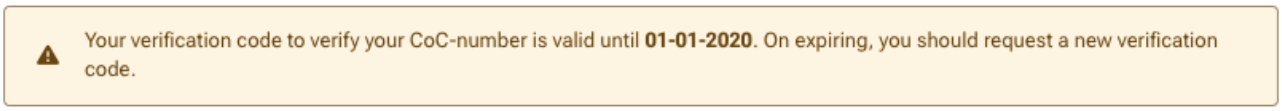
In 4.2.5 Deelvraag 5 is er in gegaan op het Design Systeem binnen Moneybird en welke voordelen dit systeem biedt. Zo is er ook voor het weergeven van fouten een component beschikbaar, die aangepast kan worden naar wens. Dit component, het "Alert" component, bestaat uit verschillende soorten, namelijk:

- Info alert; wordt gebruikt wanneer het geen Success of Warning Alert betreft.
- Success alert; wordt gebruikt wanneer een actie van de gebruiker of het opslaan van gebruikersdata goed is verlopen.
- Warning Alert; wordt gebruikt om de gebruiker te waarschuwen voor eventuele bijwerkingen als gevolg van een actie.
- Danger Alert; wordt gebruikt om de gebruiker te waarschuwen voor "gevaarlijke" bijwerkingen als gevolg van een actie of wanneer een gebruiker niet zijn/haar actie kan doorzetten als gevolg van een gemaakte fout.

Ook gelden er binnen het Design Systeem een aantal voorwaarden waaraan gehouden moet worden, zogenaamde dont's (Interview A: 3 februari 2023), namelijk:

- Een Warning- of Danger Alert mag nooit automatisch gesloten worden. De informatie moet zorgvuldig door de gebruiker worden gelezen en op verzoek van de gebruiker worden verwijderd.
- Een Warning- of Danger Alert mag niet geopend worden met "Waarschuwing!" of "Let op!" De kleur en het icoon van de melding zijn voldoende om deze boodschap over te brengen. Door deze woorden te gebruiken, kan de gebruiker onnodig bezorgd worden over het uitvoeren van een actie of het vervolgen van de volgende stap.
- Een Alert mag niet de enige visuele aanwijzing zijn voor een gevaarlijke actie. Gebruik in plaats daarvan de Danger Button component en het Dialogue component om dit te benadrukken. Een Alert mag alleen worden gebruikt om de gebruiker te informeren over de situatie.
- Gebruik niet meer dan één Alert voor een sectie in de inhoud. Vermijd een situatie die meer dan twee Alerts op dezelfde pagina vereist. Te veel Alerts verhoogt het risico op informatiestress voor de gebruiker, waardoor de gebruiker kan gaan navigeren naar een andere pagina of zijn taak niet voltooid.

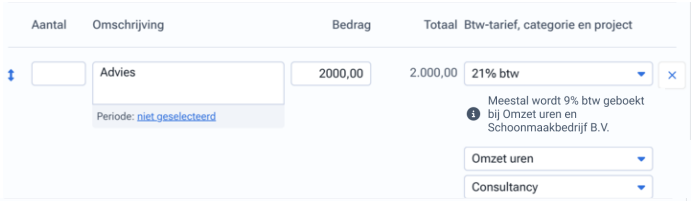
In het beoogde scenario probeert een model te voorspellen of de gebruiker een btw-percentage correct heeft geboekt. Dit doet het model aan de hand van administratie data, maar weet hierbij nooit zeker of de voorspelling klopt. Daarom is de Warning Alert in dit geval de beste optie, sinds het de gebruiker waarschuwt voor eventuele bijwerkingen, namelijk het verkeerd boeken van een btw-percentage. Dit kán gevolgen hebben op de btw-aangifte, maar dat hoeft niet het geval te zijn sinds de voorspellingen van het model niet altijd correct zijn.



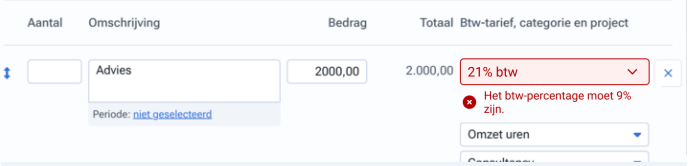
Figuur 4 Alert Component

Nadat het beoogde doel goed geformuleerd was, heeft er nog een interview plaatsgevonden (Interview B: 2 maart 2023) waarbij gesproken is over het ontwerpen van de waarschuwing binnen een factuurregel. Hierbij is de conclusie getrokken dat een Alert-component niet voldoet aan de limieten en voorwaarden zoals het Product team die voor ogen heeft, zoals deze opgesteld zijn in het resultatenhoofdstuk van deze deelvraag. Hieruit zijn een drietal designs ontstaan, welke opgenomen zijn in Figuur 5, 6 en 7 op de volgende pagina.
Uiteindelijk is er voor gekozen om gebruik te maken van het ontwerp in Figuur 5, sinds deze aan voldoet aan alle voorwaarden die eerder in dit hoofdstuk zijn toegelicht.

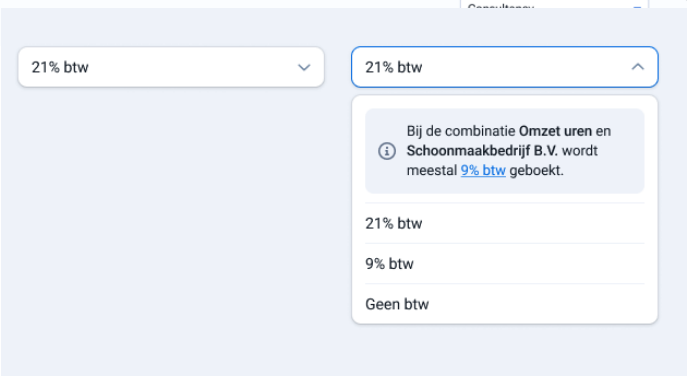
Figuur 6 voldoet niet aan de voorwaarden, omdat men kan denken dat het zéker om een fout gaat, terwijl het model dat niet altijd 100% zeker weet. Figuur 7 voldoet niet aan de voorwaarden, sinds de gebruiker het bericht niet direct opvalt door de positionering van het bericht, maar hier een extra handeling voor moet doen. Ook wordt in Figuur 7 het bericht niet geplaatst waar de fout gemaakt wordt.



Figuur 5: Applicatie ontwerp Optie 1



Figuur 6: Applicatie ontwerp Optie 2



Figuur 7: Applicatie ontwerp Optie 3

Conclusie

In dit hoofdstuk is besproken hoe fouten in het algemeen onder de aandacht worden gebracht, voordat er specifiek werd ingegaan op het onder de aandacht brengen van fouten binnen Moneybird. Bij het signaleren van fouten die door de gebruiker zijn gemaakt, worden enkele basisregels gehanteerd om de gebruiksvriendelijkheid te waarborgen. Daarnaast zijn er binnen het Design Systeem van Moneybird verschillende soorten Alert-componenten beschikbaar, die aangepast kunnen worden naar wens. Ook gelden er binnen het Design Systeem een aantal voorwaarden waaraan gehouden moet worden, de zogenaamde dont's.

In het beoogde scenario, waarbij een model probeert te voorspellen of de gebruiker een btw-percentage correct heeft geboekt, bleek de Warning Alert niet de beste optie om de gebruiker te waarschuwen voor eventuele fouten. In plaats daarvan is er voor gekozen om een Tooltip component in combinatie met een label te gebruiken.

Ten slotte is er besproken hoe het ontwerp van de waarschuwing binnen een factuurregel tot stand is gekomen. Er zijn drie designs ontstaan, waarbij er uiteindelijk is gekozen voor het ontwerp dat voldoet aan alle voorwaarden die binnen het Design Systeem van Moneybird zijn gesteld, namelijk het ontwerp wat weergegeven is in Figuur 5. Het is van belang om bij het ontwerp van waarschuwingen rekening te houden met de gestelde voorwaarden, zodat deze bijdragen aan een optimale gebruikservaring en gebruiksvriendelijkheid van de applicatie.

4.2. Conclusie

Er zijn diverse manieren om fouten en inconsistenties binnen een systeem op te sporen. Het is van groot belang om hierbij kwalitatieve data te gebruiken om statistische en/of Machine Learning modellen te kunnen trainen.

Er is er voor gekozen om voor het ontwikkelen van het PoC gebruik te maken van het algoritme met de hoogste accuraatheid om het model te trainen, namelijk het Random Forest Classifier algoritme, welke voortvloeit uit het Bijlage 1: Onderzoeksrapport. Om tot een kwalitatieve dataset te komen, zullen de stappen die toegelicht zijn in 4.2.4 Deelvraag 4 worden doorlopen. Zo zal onder andere de data gefilterd worden en vervolgens zullen onnodige velden verwijderd worden door het Random Forest Classfier algoritme in combinatie met RFECV. Het geheel, dus het trainen van het model en het verwerken van de data, zal worden ontwikkeld met behulp van een standalone Flask REST-API, welke kan communiceren met het bestaande Moneybird systeem.

Om de eindgebruiker te waarschuwen voor mogelijke fouten binnen een aangemaakte factuurregel, zal het design worden ingezet wat toegelicht is in 4.2.6 Deelvraag 6.

4.3. Requirements

Dit hoofdstuk categoriseert alle vastgestelde vereisten met behulp van de MoSCoW-methode. De vereisten zijn verdeeld in Must-haves, Should-haves, Could-haves en Would-haves, en zijn afgeleid van het theoretisch kader.

4.3.1. Functional requirements

Must-haves

- ***F.001***: Binnen een verkoopfactuur moet het systeem in staat zijn om het btw-percentage per factuurregel te voorspellen op basis van de door de gebruiker ingevoerde gegevens.
- ***F.002***: Als het voorspelde btw-percentage niet overeenkomt met het ingevoerde btw-percentage, dient de gebruiker een waarschuwing te ontvangen.
- ***F.003***: De gebruiker moet alleen een waarschuwing ontvangen als de accuraatheid van het model hoger dan 70% ligt.
- ***F.004***: De ontwikkelde toepassing moet worden getest met bijbehorende tests en succesvol de huidige CI/CD-pipeline doorlopen.

Should-haves

- ***F.005***: Een gebruiker kan op de pagina zien op basis waarvan de voorspelling gedaan is.
- ***F.006***: Er wordt een suggestie getoond aan de gebruiker met een voorspelling wat de eventueel foute waarde wel had moeten zijn volgens de voorspelling.

Could-haves

- ***F.007:*** De gebruiker kan transacties/facturen markeren als ‘opgelost’, wanneer de transactie/factuur gecontroleerd is.
- ***F.008:*** De gebruiker kan de vanaf datum instellen voor het gebruik van controle hulp.

Would-haves

- ***F.009:*** De gebruiker kan er voor kiezen om controle hulp uit te schakelen.

4.3.2. Non-functional requirements

- ***NF.010:*** Het voorspellend algoritme moet de privacy van de gebruikers in acht nemen, door alleen (geanonimiseerde) data te gebruiken van gebruikers die hiervoor toestemming hebben gegeven.
- ***NF.011:*** Bij het realiseren van het ontwerp moeten de voorwaarden van het Moneybird Design Systeem in acht worden genomen.
- ***NF.012:*** De toepassing moet werken binnen het Ruby (Ruby Corp, n.d.) ‘landschap’ van Moneybird.
- ***NF.013:*** De code moet begrijpbaar zijn voor de collega’s en bedrijfsbegeleiders binnen Moneybird, zo dat na de afstudeerfase elke Software Developer er mee verder kan.

4.4. Backlog

Dit hoofdstuk introduceert de Backlog, die alle taken bevat die idealiter voltooid zullen worden. De taken zijn geprioriteerd en worden bijgehouden in een intern systeem bij Moneybird genaamd Phabricator. Taken in de Backlog kunnen worden verplaatst naar verschillende fasen: Needs Discussion, voor taken waarover gediscussieerd moet worden, bijvoorbeeld vanwege onduidelijkheid of relevantie; Next, voor taken die als volgende op de lijst staan; Doing, voor taken waaraan momenteel wordt gewerkt; en Done, voor taken die zijn afgerond. Alle taken die opgesteld zijn in de Backlog kunnen worden gevonden in de bijlage, namelijk in Bijlage 5: Backlog.

5. Development

5.1. Ontwikkelmethodes

Dit hoofdstuk beschrijft de ontwikkelmethode die wordt gebruikt tijdens de implementatiefase van het onderzoek. Bij Moneybird wordt er gewerkt volgens het ShapeUp-principe (Singer, n.d.), dat bestaat uit iteraties van zes weken, gevolgd door een cooldown-periode van twee weken. Tijdens de eerste zes weken werken verschillende teams aan verschillende projecten die aan het begin van deze fase worden gekozen. Gedurende de cooldown-periode van twee weken wordt er niet aan een project gewerkt, maar worden er in plaats daarvan voorstellen geschreven voor de volgende projectronde.

Omdat het ShapeUp-principe met 8 weken werkt, is er voor gekozen om te werken met de Kanban-methode (Radigan, n.d.), omdat cycles van 8 weken te lang zijn om voldoende feedback te vergaren. Kanban is een populaire methode voor projectmanagement binnen de softwareontwikkeling. Bij het gebruik van Kanban worden taken die voortvloeien uit de vereisten onderverdeeld in kolommen op een Kanban-bord. Zoals in dit onderzoek beschreven, bevat het Kanban-bord verschillende kolommen, namelijk: Needs Discussion, Next, Doing en Done. In tegenstelling tot Scrum kent Kanban geen sprints of iteraties en worden taken binnen een project simpelweg toegevoegd aan het bord. Er kunnen wel doelen gesteld worden binnen het team, maar men kent binnen Kanban geen harde deadlines voor taken. Zaken zoals daily standups en retrospectives, welke worden gebruikt binnen Scrum, worden vervangen met wekelijkse gesprekken met de aangewezen afstudeerbegeleiders binnen Moneybird. Tijdens deze gesprekken wordt er gepraat over de voortgang van het onderzoek, maar ook wordt er geëvalueerd op de producten die eventueel opgeleverd zijn.

De taken in het Kanban-bord zullen in Phabricator worden bijgehouden en geprioriteerd op basis van de vereisten. Op deze manier hebben de afstudeerbegeleiders te allen tijde toegang tot de voortgang van de implementatiefase en kunnen ze op elk gewenst moment feedback geven over het te volgen proces.

5.2. Kwaliteitsbewaking

Dit hoofdstuk beschrijft de procedures die binnen Moneybird worden toegepast om de kwaliteit van de code en het systeem als geheel te waarborgen. Wanneer een collega wijzigingen aanbrengt in de codebase, wordt er een diff, ook wel een Pull Request genoemd, ingediend voor controle door een andere collega. Deze collega zal feedback geven indien nodig voordat de wijzigingen worden geïntegreerd in het huidige systeem. Bovendien moet alle code gedekt zijn door een aantal tests en alle bestaande tests doorstaan. Moneybird hanteert drie verschillende Tests (ofwel specs): Regressie tests (Feature specs), Unit tests (Unit specs) en Visuele regressie tests (Styling specs) om de kwaliteit van de code te waarborgen. In onderstaande alinea's worden de verschillende specs toegelicht.

Feature specs

Een Feature spec (regressie test) is een softwaretest die gericht is op het controleren of een bepaalde functie van de software correct werkt en voldoet aan de gestelde eisen en verwachtingen. Dit type testen wordt toegepast vanuit het perspectief van de gebruiker en is bedoeld om de functionaliteit van de software te valideren, bijvoorbeeld in het geval van het registreren/aanmelden van een nieuwe gebruiker.

Unit specs
Een Unit spec (Unit test) is een softwaretest dat zich richt op het controleren van individuele stukken code, zoals functies, of methoden. Het doel van een unit spec is om te verifiëren of de input(s) en/of output(s) van de code het verwachte resultaat teruggeven.

Styling specs
Een Styling spec (Visuele regressie test) is een softwaretest dat gericht is op het controleren van de visuele aspecten van de software. Het doel van een styling test is om te verifiëren of de visuele elementen van de software correct worden weergegeven, zoals kleuren, lettertypen, en andere aspecten.
Moneybird maakt gebruik van automatische styling tests. Deze tests maken schermafbeeldingen en vergelijken deze met het verwachte resultaat. Als er bijvoorbeeld afwijkingen zijn tussen een knop en het ontwikkelde component in het Design Systeem, zal de styling spec niet slagen. Door het uitvoeren van deze tests wordt de gebruikerservaring gewaarborgd en wordt de consistentie binnen het product gehandhaafd.

5.3. Tooling

Uit deelvraag 4 van het 4. Onderzoek is gebleken dat de Random Forest Classifier gebruikt zal worden voor het Proof of Concept. In de komende alinea's wordt beschreven op welke manier dit wordt aangesprongen en welke tooling hierbij komt kijken.

De uitwerking van deze methode kan makkelijk gebruikt worden, sinds de getrainde modellen geexporteerd zijn naar Pickle (*Pickle — Python Object Serialization — Python 3.11.1 Documentation*, n.d.) en dus alleen nog ingeladen moeten worden in een standalone Python service.

Aangezien de methodes uitgewerkt zijn in Python (voor toelichting zie 4.2.3 Deelvraag 3), is er voor gekozen om het model uit te werken in een standalone Python REST-API, welke direct communiceert met het platform van Moneybird. Hierbij is er voor gekozen om het PoC te ontwerpen met behulp van Flask (Flask, n.d.). Dit is een framework waarmee vaak applicaties worden die ontwikkelt die gebasseerd zijn op Machine Learning/Data Science. Er zijn andere frameworks waarmee het zelfde bereikt kan worden zoals FastAPI, maar deze zijn vaak nog te onvolwassen voor het ontwikkelen van dergelijke applicaties (Barguzar, 2022).

De REST-API kan gebruik maken van de geëxporteerde trainingsmodellen en deze gebruiken om voorspellingen te maken. Tot slot is het belangrijk om ervoor te zorgen dat de voorspellingen betrouwbaar zijn. Deze voorspellingen kunnen als betrouwbaar worden beschouwd als de nauwkeurigheid van het model hoger is dan 70% (Barkved, 2022). Vervolgens het platform van Moneybird het REST-API endpoint aanspreken die een voorspelling maakt op basis van gebruikersinput, die is ingevuld in de Moneybird applicatie.

5.4. Ontwerp

In dit hoofdstuk wordt het ontwerp/architectuur van zowel het systeem, als het ontwerp van de applicatie toegelicht. Ook wordt er dieper ingegaan op de CI/CD-pipeline en de REST-API.

High-level architecture

Figuur 6 geeft de high-level architectuur van het systeem weer. Het systeem bestaat uit verschillende componenten: het Moneybird-systeem dat de volledige applicatie aandrijft via een bestaande API, de Flask API die extra functionaliteit biedt voor het verwerken van data en het voorspellen van gebruikersinvoer, en de Firewall die ervoor zorgt dat alleen het Moneybird-systeem kan communiceren met de Flask API. Bijlage 6: API Flowchart illustreert het proces van een API-verzoek binnen de Flask API, specifiek het verzoek dat een voorspelling genereert op basis van de gegevens van een ingevulde verkoopfactuur door de gebruiker.

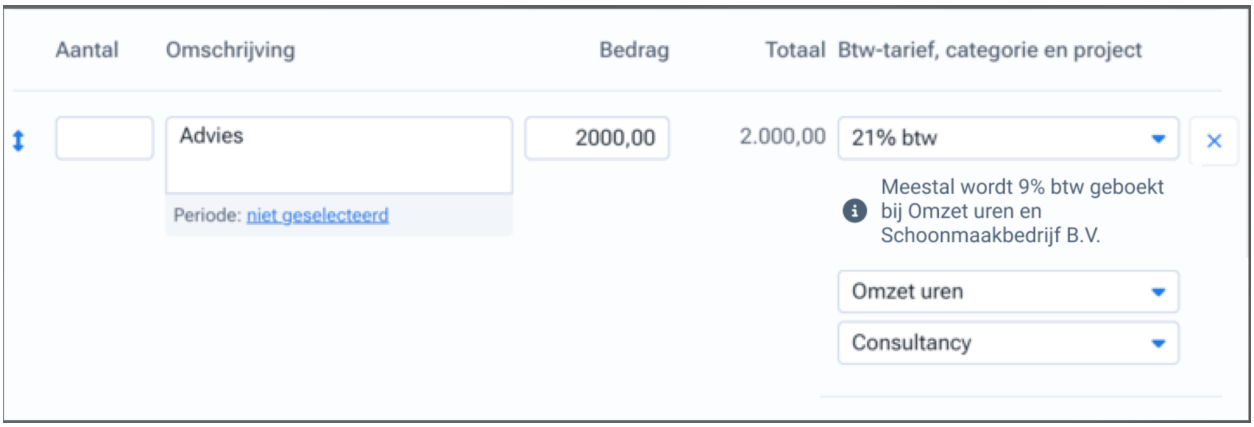
Figuur 8: API Diagram

CI/CD-flowchart

Moneybird maakt gebruik van BuildKite om het testen en implementeren van applicaties binnen hun applicatielandschap te automatiseren. BuildKite is een applicatie waarmee dit mogelijk is. Bij het testen van de Flask API wordt deze eerst gedockerized binnen BuildKite. Vervolgens worden verschillende tests uitgevoerd om de kwaliteit van de code en de functionaliteit van de applicatie te waarborgen. De laatste stap is het deployen van de app, echter zal gedurende het onderzoek deze stap niet worden uitgewerkt. Dit proces is weergegeven in Bijlage 7: CI/CD-flowchart.

Design applicatie

Bij het ontwerpen van een onderdeel binnen het Moneybird systeem is rekening gehouden met de eerder opgestelde voorwaarden in dit document (4.1.6. Deelvraag 5). Voor het ontwerp zijn ongestructureerde interviews afgenomen met collega's van het productteam om inzichten te verkrijgen, zie Interview A: 3 februari 2023 en Interview B: 2 maart 2023. Nadat het ontwerp is geïmplementeerd in de applicatie, zal het worden gevalideerd door een enkele collega's om te bepalen of het voldoet aan de beoogde doelstellingen en begrijpelijk is voor de gebruiker.



Figuur 9 Design factuurregel melding

Figuur 9 toont hoe er een melding wordt weergegeven met een suggestie wanneer het btw-percentage in een factuurregel, die is ingevuld door de gebruiker, potentieel niet klopt. Zoals eerder in dit document vermeld, bevat deze suggestie het voorspelde btw-percentage, de categorie en het contact. Het uiteindelijke ontwerp kan echter enkele kleine verschillen vertonen ten opzichte van het gepresenteerde design in figuur 9, omdat het ontwerp gedurende het onderzoek zal worden gevalideerd door collega's en andere belanghebbenden. Door het ontwerp te valideren kan het ontwerp aangepast worden om te voldoen aan de gestelde eisen en de behoeften van de gebruikers.

5.5. Testplan

In dit hoofdstuk wordt het testplan van de applicatie beschreven. Er wordt eerst in het kort beschreven hoe de kwaliteit van de Flask API gewaarborgd kan worden, vervolgens zal er beschreven worden hoe de Flask API automatisch geintregeerd en gedeployed kan worden met behulp van BuildKite, en tot slot zal er beschreven worden hoe de koppeling tussen de Flask API en de bestaande Moneybird API gerealiseerd kan worden.

Flask API

De kwaliteit van de Flask API wordt op verschillende manieren gewaarborgd. De kwaliteit van de code wordt bewaakt op de manier die eerder beschreven is in hoofdstuk 5.2 Kwaliteitsbewaking. Ook zal er een enkele test geschreven worden om de Machine Learning modellen te valideren, in de vorm van het controleren van de data op eventuele fouten.

BuildKite

Binnen BuildKite kan een CI/CD-pipeline geconfigureerd worden welke de Flask API automatisch build en deployed. Ook kunnen de ontwikkelde tests uitgevoerd worden binnen deze pipeline. Binnen het huidige systeem wordt hetzelfde gedaan voor de systemen van Moneybird. Elke verandering aan de productieomgeving wordt uitgebreid automatisch getest en gedeployed.

Flask API / Moneybird systeem

De koppeling tussen de Flask API en het Moneybird systeem kan getest worden door middel van een spec binnen het Moneybird systeem, echter is het endpoint waarmee gecommuniceerd wordt lastig te testen, sinds er gewerkt wordt met echte data en een model wat eventueel niet accuraat is. Toch kan er getest worden of de Flask API de gewenste statuscode teruggeeft o.i.d.

6. Realisatie

6.1. Het ophalen van kwalitatieve data

Om gegevens op te halen, is het van belang dat de data niet herleidbaar is naar de gebruiker en alleen de relevante data bevat die het model kan gebruiken. Bij het voorspellen van non-lineaire waarden moet de dataset alleen non-lineaire waarden bevatten. Om dit te bereiken kan bij het maken van een database query van tevoren alle non-lineaire waarden of waarden die geen invloed hebben op de data worden verwijderd. Dit proces staat bekend als handmatige selectie van features en is ook toegepast bij het maken van de onderstaande database query.

```
SELECT details.tax_rate_id, details.ledger_account_id, details.price, invoices.contact_id AS
contact
FROM details
INNER JOIN invoices ON details.invoice_id = invoices.id
WHERE details.administration_id = 0123456789
```

Er is besloten om gegevens te gebruiken die afkomstig zijn uit slechts één administratie. Dit is omdat er bij een overzicht van alle administraties enkele problemen naar voren komen die het verwerken van deze gegevens bemoeilijken. Er zijn namelijk wel gestandaardiseerde btw-groepen en categorieën, echter hebben de btw-groepen en categorieën per administratie een unieke identifier, waardoor deze velden niet met elkaar te vergelijken zijn op administratief niveau. Bovendien is het analyseren van de namen van categorieën niet haalbaar, aangezien elke administratie verschillende namen voor categorieën gebruikt en deze ook op verschillende manieren vormgeeft. Deze keuze heeft wel een invloed op de grootte van de dataset, omdat er ook kleinere administraties zijn die mogelijk te klein zijn om voorspellingen te kunnen maken.

De bovenstaande query haalt alle factuurregels op, inclusief hun categorieën, btw-percentages en gekoppelde contacten. Om ervoor te zorgen dat het model voorspellingen kan doen op basis van een enkele administratie, is er een "WHERE-clause" toegevoegd die filtert op administratie-ID. Voor elke administratie wordt de bovenstaande query uitgevoerd. Het resultaat van de query wordt vervolgens geëxporteerd als een CSV-bestand, zodat het model ermee kan werken.

De grootte van een dataset
De dataset die wordt gebruikt om het PoC te ontwikkelen, bestaat uit 252 factuurregels afkomstig uit de administratie van een willekeurig bedrijf. Met behulp van de eerder besproken query kan het model aan de slag met deze data. Bij het bepalen of de dataset gebruikt mag worden, is er gecontroleerd of de klant toestemming heeft gegeven voor het delen van zijn/haar data. Ook is alle data welke gebruikt wordt voor de data-analyse geanonimiseerd en valt dus niet direct te herleiden naar de klant.

Een belangrijke overweging bij het bepalen of een dataset groot genoeg is, is het aantal factuurregels dat beschikbaar is voor elk btw-tarief. In deze dataset komt het eerste btw-tarief bijvoorbeeld 207 keer voor en het tweede btw-tarief 45 keer. Voor elke feature in de dataset geldt dat er minimaal tien keer zoveel regels nodig zijn (How Much Data Is Needed For Machine Learning?, 2022). Op basis hiervan is de mock-dataset groot genoeg.

Het is echter belangrijk om op te merken dat de kwaliteit van de dataset een grote invloed kan hebben op de nauwkeurigheid van het uiteindelijke model. Als bijvoorbeeld de features in de dataset niet relevant zijn of de kwaliteit van de labels laag is, kan dit leiden tot een minder nauwkeurige classificatie.

6.2. Het verwerken van data

Voor de verwerking van de gegevens in het CSV-bestand wordt een Preprocessing-model gebruikt, zoals eerder beschreven in dit document. Tijdens dit proces worden rijen met lege waarden verwijderd, worden non-numerieke waarden omgezet naar numerieke waarden en worden indexen verwijderd als deze aanwezig zijn. Er wordt gebruik gemaakt van RFECV en een RandomForestClassifier, die beide geschikt zijn voor het selecteren van de meest geschikte non-lineaire velden binnen een dataset. Deze algoritmen voldoen aan de verwachtingen van het beoogde doel, zoals deze zijn opgenomen in Deelvraag 4 van het Bijlage 1: Onderzoeksrapport. De onderstaande code leest het CSV-bestand in en verwijdert vervolgens alle rijen die onbruikbare gegevens bevatten.

```
# Declare the field to predict and read the CSV file
field_to_predict = "vat"
df_ = pd.read_csv('mb/preprocessing/INC_MOCK_DATA.csv')

# Remove NaN rows, non_numeric values and reset the index
df_ = df_.apply(pd.to_numeric, errors='coerce')
df_ = df_.dropna()
df_ = df_.reset_index(drop=True)
```

Vervolgens kunnen de eerdergenoemde modellen voorspellen welke velden binnen de dataset het meest geschikt zijn voor het maken van de voorspellingen. De velden die niet geschikt blijken te zijn, worden automatisch verwijderd uit de dataset. De onderstaande code is hier de uitvoering van.

```
estimator = RandomForestClassifier(random_state=42)
selector = RFECV(estimator, step=1)
selector = selector.fit(x_values, y_values.ravel())
selector = selector.transform(x_values)
```

6.3. Het trainen en exporteren van een ML model

Om voorspellingen te kunnen maken op basis van de verwerkte data, moet er een ML model getraind worden. Er is voor gekozen om hierbij gebruik te maken van het Decision Trees model. De werking van dit model is eerder in dit document toegelicht en heeft op de test data de hoogste accuraatheid. Om te voorkomen dat het model ge-overfit (*What Is Overfitting? - Overfitting in Machine Learning Explained - AWS*, n.d.) wordt, wordt de data eerst gesplit in een dataset welke gebruikt wordt om het model te trainen en in een test dataset welke gebruikt wordt om het model te testen. Hiervoor wordt de train_test_split library van sklearn gebruikt.

```
# Test size = 30%, Train size = 70%
train, test = train_test_split(df_, test_size=0.3)
```

Vervolgens wordt de training data gebruikt om het model te trainen. Wanneer het model getraind is, zal het model geëxporteerd worden naar een Pickle, zodat het model weer hergebruikt kan worden op een later moment. Vervolgens kan er een classificatie rapport worden gemaakt welke de accuraatheid van het model meet op basis van de test set. De onderstaande code voert dit proces uit.

```
# Define the DecisionTreeClassifier model
clf = tree.DecisionTreeClassifier()
clf = clf.fit(train_columns.values, np.ravel(train_target_column))


pickle.dump(clf, open(f'models/{administration_id}.pickle', "wb"))
# Supervised accuracy measurement
test_columns = test.drop([label_to_predict], axis=1)
score = clf.score(test_columns.values, test[label_to_predict])
predictions = clf.predict(test_columns.values)

report = classification_report(test[label_to_predict], predictions)
```

Op basis van de dataset is een meting gedaan. Deze meting geeft de precision, recall en de F1-score per soort btw-percentage. Hierbij is vooral de F1-score belangrijk, sinds het een balans biedt tussen precision (hoeveel van de voorspelde positieven zijn daadwerkelijk positief) en recall (hoeveel van de daadwerkelijke positieven zijn correct voorspeld als positieven). De gemiddelde F1-score is 93%, wat betekent dat het model vrij accurate voorspellingen kan maken. Onderstaande tabel is een visuele weergave van de gedane meeting.

```
                   precision    recall  f1-score   support

226909703498630207       0.50      0.20      0.29         5
237333135720711416       0.94      0.98      0.96        66

          accuracy                           0.93        71
         macro avg       0.72      0.59      0.62        71
      weighted avg       0.91      0.93      0.92        71
```

In de linkerkolom staan twee lange getallen die corresponderen met de btw-percentages. Als er meer btw-percentages in de dataset zijn, zal het rapport meer rijen bevatten. Het eerste btw-percentage heeft een lage F1-score, wat betekent dat het moeilijk te voorspellen is. Dit komt doordat er slechts 5 van de 253 rijen dit btw-percentage bevatten, wat te wijten is aan de grootte van de dataset. Het algoritme kan hierdoor niet goed bepalen wanneer dit btw-percentage voorkomt. Dit probleem zal echter afnemen naarmate er meer factuurregels in een administratie aanwezig zijn.

6.4. Het maken van voorspellingen

Wanneer het model getraind is, kan vervolgens de geëxporteerde Pickle gebruikt worden om een voorspelling te maken op basis van nieuwe user input. Hierbij staat in de onderstaande code het “y_predicted” veld voor de voorspelling die gemaakt is door het model.

```
loaded_model = pickle.load(open("naive_bayes.pickle", "rb"))
y_predicted = loaded_model.predict([[ledger_account_id, price, contact_id]])
```

Via het API endpoint kan de Moneybird applicatie communiceren en zo een voorspelling doen. De benodigde documentatie van het endpoint is beschikbaar in de onderstaande tabel.

POST	/predict/:administration_id	
Het maken van een voorspelling op basis van een verkoopfactuur		
HEADERS (JSON)		
Name	**Type**	**Required**
ledger_account_id	integer(int64)	Yes
price	float	Yes
contact_id	integer(int64)	Yes
RESPONSES		

Status	Meaning	Description
200	OK	Succesful operation
400	Bad Request	Invalid header request
RESPONSE SCHEMA (JSON) Status code 200		
Name	**Type**	**Description**
predicted_tax_rate_id	integer(int64)	The tax_rate_id predicted by the ML model

Tabel 6 API predict endpoint

6.5 Moneybird-systeem (MB2) integratie

Daarna is er gewerkt aan de integratie van het GUI-ontwerp in het bestaande MB2-platform en het maken van API-verzoeken wanneer klanten verkoopfacturen aanmaken of bewerken. De API-verzoeken worden alleen geregistreerd als de gebruiker een factuurregel aanmaakt of verwijdert en alle velden heeft ingevuld. Het proces wordt hieronder beschreven, te beginnen bij de meest zichtbare laag, namelijk de GUI. Vervolgens wordt uitgelegd hoe een client-side (*What Is the Client Side? | Definition by Elementor*, n.d.) verzoek wordt verzonden dat communiceert met een endpoint in MB2. MB2 stuurt dan een verzoek naar de Flask API.

Om het ontwerp in Figuur 9 te realiseren, moest het factuurregelcomponent (Details) worden aangepast, zodat het zou werken bij het aanmaken en bewerken van elke factuurregel. Hiervoor is het bestaande tooltip-component van het Design Systeem gebruikt. De onderstaande code toont de volledige waarschuwing aan de gebruikers, die overeenkomt met het ontwerp in Figuur 9:

```
<div style="display: none; margin: 3px" data-tax-rate-suggestion-target="tooltip">
 <div style="align-self: center">
   <%= ui_component('tooltip') %>
 </div>
 <p data-tax-rate-suggestion-target="label" style="margin-left: 10px"></p>
</div>
```

Vervolgens zijn er Stimulus-acties toegevoegd aan de categorie- en btw-tariefvelden. Hierdoor zorgt Stimulus ervoor dat wanneer een gebruiker bijvoorbeeld wijzigingen aanbrengt, de waarde van de wijziging wordt gecommuniceerd met de bijbehorende Stimulus-controller. Deze controller wordt geladen wanneer een verkoopfactuur wordt aangemaakt en maakt vervolgens, wanneer alle velden zijn ingevuld, een API-request naar een Ruby on Rails controller en stuurt hierbij de veldwaarden mee in JSON-formaat. De Stimulus-controller is opgenomen in Bijlage 10: Stimulus controller.

Daarna is het de taak van de Ruby on Rails-controller (opgenomen in Bijlage 11: Ruby on Rails controller) om de meegestuurde veldwaarden via een API-request met de Flask API te communiceren. De Flask API stuurt een btw-tariefwaarde door, die overeenkomt met een btw-tarief in de administratie van een klant, en kan deze converteren naar een naam. Ook moeten de andere meegestuurde waarden worden omgezet naar leesbare waarden. Het btw-tarief wordt op de volgende manier geconverteerd naar een leesbaar formaat:

```
ledger_account_name = administration.ledger_accounts
                                    .find(params[:ledger_account_id]).name
predicted_tax_rate_name = administration.tax_rates
             .find(response_body["predicted_tax_rate_id"].to_i).name
```

Vervolgens worden deze leesbare waarden gebruikt om een bericht op te stellen zoals weergegeven in Figuur 9. Hierbij wordt rekening gehouden met het vertalen van de tekst naar verschillende talen, zoals Duits, Engels en Nederlands, op basis van de taal die de klant heeft geselecteerd binnen de administratie. Dit ziet er als volgt uit:

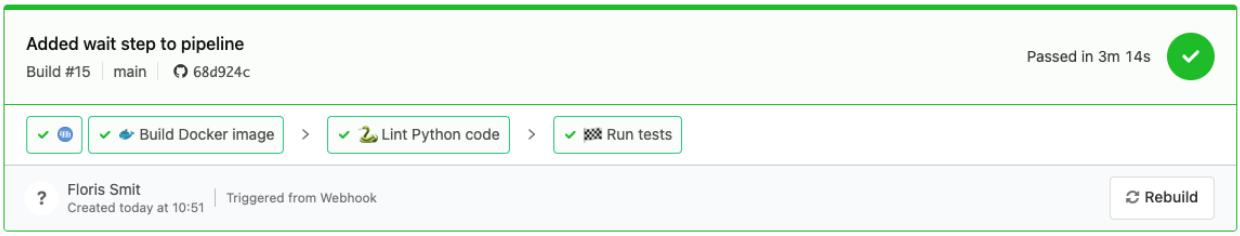
```
message = t('tax_rate_prediction_message',
           predicted_tax_rate_name: predicted_tax_rate_name,
           ledger_account_name: ledger_account_name,
           company_name: params[:contact_name])
```

Het opgestelde bericht wordt vervolgens als antwoord op het API-request van de Stimulus-controller verstuurd. De Stimulus-controller kan het bericht dan renderen en weergeven binnen de factuurregel en dus de GUI van het platform.

6.6. CI/CD-pipeline

Ten slotte wordt de Flask API gedockerized en worden er tests en een linter toegevoegd met behulp van PyTest en PyLint. Deze stappen kunnen in de BuildKite-pipeline worden geïntegreerd. Om deze stappen te specificeren, moet er een pipeline.yml-bestand worden aangemaakt en toegevoegd aan de repository. Het bijgevoegde voorbeeld van een pipeline.yml-bestand is te vinden in Bijlage 9: CI/CD-pipeline.yml. Wanneer nieuwe code aan de Git-repository wordt toegevoegd, zal een webhook (Guay, n.d.) BuildKite op de hoogte stellen van de nieuwe applicatieversie. Hierdoor zal de pipeline worden gestart. Figuur 10 toont de gedefinieerde stappen in de BuildKite-pipeline, die in dit geval allemaal zijn geslaagd.

Figuur 10: Weergave BuildKite CI/CD-pipeline



7. Validatie

In dit hoofdstuk worden allereerst de onderdelen, zoals beschreven in 5.4 Testplan, verder toegelicht inclusief de implementatie ervan. Ten slotte wordt het Machine Learning-model dat is geïmplementeerd in het PoC uitgelegd en gevalideerd aan de hand van enkele gegenereerde rapporten. In 6.3 Het trainen en exporteren van een ML model is uitgelegd hoe deze rapporten tot stand komen en waarom ze betrouwbaar zijn.

7.1. Het testen van de Flask API

Er zijn verschillende tests opgezet om de Flask API te testen, waaronder zowel unit-tests als functionele tests. Deze tests kunnen automatisch worden uitgevoerd met behulp van de PyTest-package (PyTest, n.d.). Om de kwaliteit van de code te waarborgen, wordt ook gebruik gemaakt van een Linter genaamd PyLint (PyPi, n.d.). PyLint zorgt ervoor dat de code voldoet aan een standaard formaat dat door ontwikkelaars moet worden gevolgd.

7.2. De opgezette CI/CD-pipeline

Er is een CI/CD-pipeline opgezet in BuildKite om de applicatie automatisch te integreren. Binnen deze pipeline wordt de Flask API eerst geïntegreerd in een Docker-container en vervolgens doorloopt de applicatie alle tests en linter checks met behulp van de PyTest en PyLint packages, zoals beschreven in het vorige hoofdstuk.

7.3. Het testen van de MB2 implementatie

De Ruby on Rails-controller is ook voorzien van een spec/test. Deze spec test de functionaliteit van de API door de API te mocken en vooraf ingestelde waardes te controleren en de antwoorden op de API-request te controleren. De API wordt gemockt om te voorkomen dat de tests falen als de Flask-API niet bereikbaar is en omdat de Flask-API alleen werkt op basis van echte administratieve data. De spec is zijn geheel opgenomen in Bijlage 12: Ruby on Rails controller spec. De spec zal automatisch toegevoegd worden aan de bestaande CI/CD-pipeline van MB2 wanneer de code wordt toegevoegd aan de Development en/of Productie omgeving.

7.4. Het valideren van het Machine Learning model

Om er zeker van te zijn dat het ontwikkelde ML-model in het Proof of Concept (PoC) betrouwbaar is, zal er in dit hoofdstuk een conclusie worden getrokken op basis van een aantal Classification Reports (SkLearn, n.d.) welke zijn opgesteld op basis van een aantal random gekozen administraties. Deze reports bevatten onder andere de precision, recall, de F1-score per administratie en de grootte van de dataset, zie 6.3 Het trainen en exporteren van een ML model. Op basis hiervan kan iets gezegd worden over de betrouwbaarheid en de accuraatheid van het ML-model, en in hoeverre dit model per administratie verschilt.

7.4.1. Het belang van het macro-gemiddelde en de accuraatheid

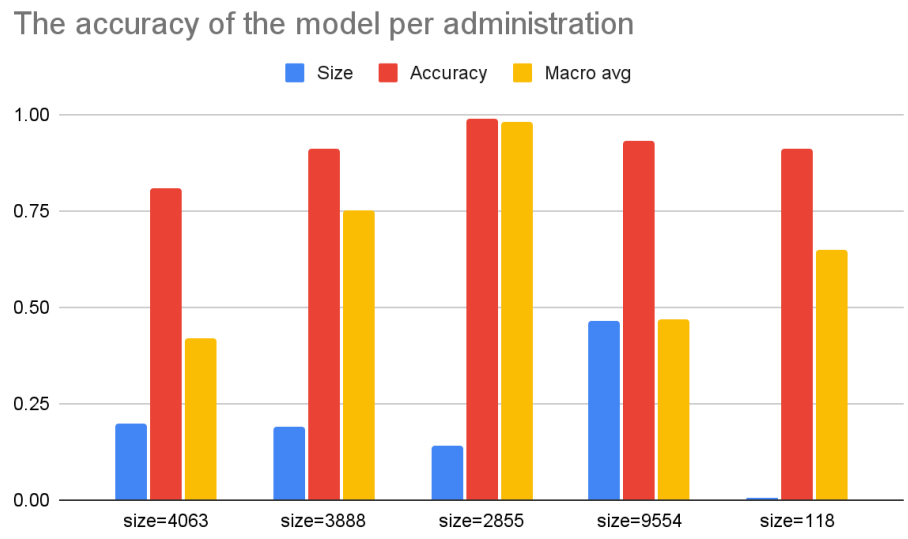
Om ervoor te zorgen dat er geen ongeoorloofde data wordt gebruikt voor Data-Analyse doeleinden, worden de administraties die worden gebruikt voor de validatie van het model willekeurig gekozen op basis van een lijst van administration_ids. Deze lijst wordt gegenereerd door een query die controleert of de betreffende administraties toestemming hebben gegeven voor het delen van hun data. Op de onderstaande grafiek worden de grootte van de dataset, de accuraatheid van het model en het macro-gemiddelde per administratie weergegeven. De accuraatheid van het model geeft het percentage correct voorspelde factuurregels weer, terwijl het macro-gemiddelde het gemiddelde neemt van de accuraatheid per btw-tarief.

Als de accuracy hoog is maar de macro average laag, kan dit betekenen dat het model bijvoorbeeld goed presteert in het voorspellen van het meest voorkomende btw-tarief, maar minder goed in het voorspellen van minder voorkomende btw-tarieven. Dit kan problematisch zijn als alle btw-tarieven even belangrijk zijn voor de voorspellingen gemaakt worden. In dat geval kan het nodig zijn om in een vervolg onderzoek aanpassingen te doen aan het model of aan de dataset, door bijvoorbeeld het toevoegen van meer trainingsgegevens voor minder voorkomende btw-tarieven, of door gebruik te maken van een andere classificatiemethode.

7.4.2. Het resultaat per administratie

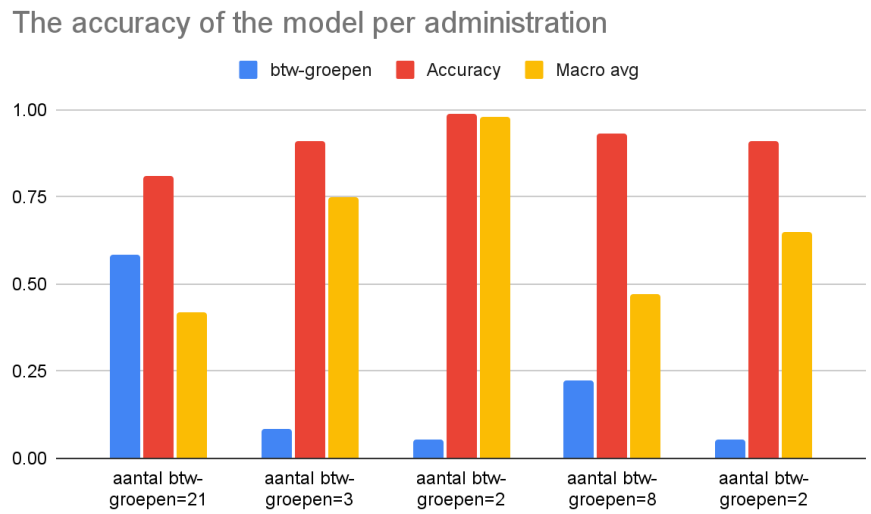
Uit de onderstaande grafiek lijkt het op het eerste gezicht dat de grootte van de dataset geen invloed heeft op zowel de accuraatheid als het macro-gemiddelde van het model (Scharwächter, 2022). Ook lijkt er geen wederzijdse invloed te zijn tussen de accuraatheid en het macro-gemiddelde. Er dient echter wel te worden onderzocht welke andere factoren mogelijk van invloed zijn op deze resultaten, sinds het macrogemiddelde in sommige administraties lager is dan gewenst; elk btw-tarief is namelijk even belangrijk wanneer er voorspellingen gemaakt worden.

Figuur 11: Accuraatheid van het ML-model tegenover grootte



Een ander beeld wordt gevormd wanneer de grootte van de dataset wordt vervangen door het aantal btw-groepen per administratie. Dan lijkt het op het eerste gezicht alsof het aantal btw-groepen invloed heeft op het macro-gemiddelde. En dat is ook verklaarbaar, sinds deze administraties over minder gegevens beschikken van minder voorkomende btw-groepen, waardoor het voorspellen hiervan moeilijker wordt. En sinds alle btw-groepen even belangrijk zijn als het gaat om het maken van voorspellingen, is het model niet geschikt wanneer een administratie veel weinig voorkomende btw-groepen bevat.

Figuur 12: Accuraatheid van het ML-model tegenover btw-groepen



De betrouwbaarheid van het model

Wanneer er in de dataset weinig factuurregels aanwezig zijn voor een bepaalde btw-groep, kunnen er minder nauwkeurige voorspellingen gemaakt worden op basis van nieuwe gebruikersinput. Als vuistregel geldt dat er minstens tien keer zoveel factuurregels met een bepaald btw-tarief aanwezig moeten zijn in een dataset als het aantal features. Dit betekent dat een btw-groep minstens zestig regels moet bevatten voordat er iets gezegd kan worden over de betrouwbaarheid van de voorspellingen. Hoewel het model de meeste factuurregels goed kan voorspellen, kan het moeite hebben met het voorspellen van btw-groepen die niet voldoende vertegenwoordigd zijn in de dataset van een administratie. In deze gevallen kan het model niet betrouwbaar zijn en kan het niet worden gebruikt voor het maken van dergelijke voorspellingen.

8. Conclusie

In dit onderzoek is gezocht naar een antwoord op de vraag: *"Hoe kan een systeem ontwikkeld worden om fouten/inconsistenties binnen de boekhouding te detecteren en te presenteren aan de klant?"*

Om deze onderzoeksvraag te beantwoorden is er gezocht naar verschillende manieren om fouten binnen de boekhouding van de klant te detecteren en een suggestie te geven wanneer er een fout wordt gemaakt. Uit het gedane gebruikersonderzoek bleek dat ondernemers vooral controlehulp zochten bij het doen van hun belastingaangifte. Daarom lag de focus van dit onderzoek op twee belangrijke aspecten die van invloed zijn op de aangifte, namelijk het opstellen van verkoopfacturen en het correct koppelen van btw-tarieven aan deze facturen.

Tijdens het onderzoek werd duidelijk dat statistische methoden, zoals de Normaalverdeling en de IQR-methode, niet geschikt waren om btw-tarieven binnen een verkoopfactuur te voorspellen. In dit geval bood een Machine Learning-methode de beste oplossing, omdat statistische methoden geen verbanden kunnen leggen tussen features in een dataset, maar ML-methodes wel. Om de meest geschikte ML-methode te vinden, is er literatuuronderzoek uitgevoerd en zijn verschillende methoden op basis van resultaten met elkaar vergeleken. Omdat de beschikbare data administratie-breed niet gestandaardiseerd is, is er voor gekozen voor elke administratie een apart model te trainen. De resultaten hiervan voldoen aan de verwachting, echter scoren de modellen waarbij de administratie relatief klein is en de btw-tarieven onevenredig verdeeld zijn ondermaats.

Hoewel de modellen niet optimaal presteren, zijn er wel meerdere producten opgeleverd die samen het Proof of Concept (PoC) vormen. Deze producten zijn:

- Een Flask API met Docker support voor het maken van voorspellingen.
    - Waarbij het gekozen ML-model vervangen kan worden met een nieuw/verbeterd model.
    - Waarbij het volledig product nog functioneert wanneer de dataset vervangen wordt.
- Een SQL-query voor het ophalen van de benodigde data (ook administratie-breed).
- Een weergave in Moneybird die suggesties toont bij het aanmaken van een factuurregel.

Het PoC is ontworpen met het oog op mogelijke toekomstige wijzigingen, waarbij alleen aanpassingen nodig zijn aan de data en het ML-model, indien nodig. Als het ML-model wordt verbeterd, hoeft er niets te worden aangepast aan de Moneybird systeem implementatiecode of aan de rest van de code binnen de API. Dit zorgt ervoor dat het PoC een toekomstbestendig product is waar iedereen gemakkelijk aan kan werken.

De "Must Have" requirements zijn als volgt opgesteld na het afronden van het theoretisch kader:

- ***F.001***: Binnen een verkoopfactuur moet het systeem in staat zijn om het btw-percentage per factuurregel te voorspellen op basis van de door de gebruiker ingevoerde gegevens.
- ***F.002***: Als het voorspelde btw-percentage niet overeenkomt met het ingevoerde btw-percentage, dient de gebruiker een waarschuwing te ontvangen.
- ***F.003***: De gebruiker moet alleen een waarschuwing ontvangen als de accuraatheid van het voorspelde btw-percentage hoger dan 70% ligt.
- ***F.004***: De ontwikkelde toepassing moet worden getest met bijbehorende tests en succesvol de huidige CI/CD-pipeline doorlopen.

Aan het einde van het afstudeerproject zijn alle vereisten die als "Must Have" zijn aangemerkt behaald en bewezen met behulp van het ontwikkelde prototype. Naast het behalen van F.004 is er een nieuwe CI/CD-pipeline gemaakt om de Flask API te testen, naast het doorlopen van de huidige CI/CD-pipeline met een extra test.

Ook zijn er aantal non-functional requirements opgesteld na het afronden van het theoretisch kader:

- ***NF.010:*** Het voorspellend algoritme moet de privacy van de gebruikers in acht nemen, door alleen (geanonimiseerde) data te gebruiken van gebruikers die hiervoor toestemming hebben gegeven.
- ***NF.011:*** Bij het realiseren van het ontwerp moeten de voorwaarden van het Moneybird Design Systeem in acht worden genomen.
- ***NF.012:*** De toepassing moet werken binnen het Ruby (Ruby Corp, n.d.) 'landschap' van Moneybird.
- ***NF.013:*** De code moet begrijpbaar zijn voor de collega's en bedrijfsbegeleiders binnen Moneybird, zo dat na de afstudeerfase elke Software Developer er mee verder kan.

Aan het eind van het afstudeerproject zijn alle non-functional requirements behaald. Requirement **NF.010** is behaald door bij het opstellen van de SQL-query alleen data op te halen van administraties die toestemming hebben gegeven voor het gebruiken van hun data voor Data-analyse. Om aan requirement **NF.011** te voldoen, zijn interviews afgenomen met medewerkers die verantwoordelijk zijn voor het ontwerpen van de applicatie. Hierbij is rekening gehouden met de gestelde eisen en beschikbare componenten tijdens het ontwerpproces. **NF.012** behaald door een PoC te ontwikkelen die ondersteuning biedt aan het framework (Ruby on Rails) waar het Moneybird platform gebruik van maakt. Ten slotte is **NF.013** behaald door onder andere een README.md-bestand toe te voegen aan de API-repository, die een handleiding bevat voor het lokaal draaien van de API. Ook is elke gebruikte methode gedocumenteerd in het Bijlage 1: Onderzoeksrapport.

Uit dit onderzoek is het helder geworden hoe het Moneybird platform aangepast kan worden om ondernemers te helpen fouten/inconsistenties binnen hun boekhouding te detecteren. Het eerste onderdeel hiervan is gerealiseerd door de gebruikers suggesties te tonen wanneer het model denkt te weten dat het ingevulde btw-tarief binnen een factuurregel incorrect is. Echter dienen de aanbevelingen en adviezen die gegeven zijn opgevolgd te worden om daadwerkelijk gebruik te maken van het gerealiseerde product. Wanneer dit gedaan is kan Moneybird de ondernemer nog blijer maken door te helpen bij het detecteren van fouten/inconsistenties van de boekhouding.

9. Aanbevelingen

Dit hoofdstuk beschrijft en licht de aanbevelingen toe die voortkomen uit het onderzoek. Uitgebreide toelichting van de aanbevelingen is te vinden in Bijlage 8: Adviesrapport. Dit hoofdstuk begint met een bespreking van het synchroniseren van de database, gevolgd door het verwerken van data en het trainen van een model, het geautoriseerd communiceren met de API en tot slot het deployen van de app naar een productieomgeving.

9.1. Het synchroniseren van de database

Op dit moment wordt er in het Proof of Concept (PoC) gebruik gemaakt van echte klantendata, maar deze moet nog handmatig worden geïmporteerd. Het is wenselijk dat deze data dagelijks automatisch wordt geïmporteerd, zodat het model regelmatig wordt gevoed met nieuwe administratieve gegevens, om zo de accuraatheid van het ML-model mogelijk te verbeteren. Om dit te realiseren, kan een cronjob binnen de Flask API worden gebruikt in combinatie met een SQL-query. Om dit te kunnen doen, moeten er eerst leesrechten worden verleend voor de database van Moneybird. Hoe de cronjob en SQL-query moeten worden opgesteld, is te vinden in Bijlage 8: Adviesrapport.

9.2. Het verwerken van de data en het trainen van het ML-model

Om de gesynchroniseerde data te kunnen gebruiken voor het trainen van het ML-model, moet deze eerst worden verwerkt. De stappen die hiervoor nodig zijn, staan beschreven in hoofdstuk 6. Realisatie. De functionaliteit die nodig is voor deze stappen is reeds op een herbruikbare manier opgesteld en dient enkel nog te worden geïntegreerd in de eerder vermelde cronjob.

9.3 Het geautoriseerd communiceren met de Flask API

Met behulp van deze gegevens kan het model voorspellingen genereren. Om dit te bereiken kan het Moneybird-systeem een API-verzoek sturen naar de Flask API. Momenteel is er geen autorisatie of authenticatie geïmplementeerd in het PoC. Dit betekent dat elk systeem onveilige verzoeken kan versturen naar de Flask API. Voordat de Flask API kan worden geïmplementeerd in een productieomgeving, moet er een oplossing worden ontwikkeld die aansluit bij de huidige autorisatie-flow van het Moneybird-systeem. Om dit te bereiken moet er gezocht worden naar een passende oplossing.

9.4 Het deployen van de Flask API naar een productieomgeving

Wanneer alle bovenstaande stappen zijn voltooid, kan de Flask API worden gedeployed naar een productieomgeving. De exacte vorm van deze implementatie moet nog worden onderzocht. Mogelijke opties zijn het implementeren van de API op een zelfbeheerde server of gebruik maken van App Services die worden aangeboden door Cloud Providers zoals Azure en Amazon Web Services (AWS). In BuildKite kan een stap worden toegevoegd om de Docker-container met de Flask API te implementeren op een dergelijke service. Deze stap is al opgenomen in Bijlage 7: CI/CD-flowchart.

9.5 Het onderzoek doen naar de invloed van features binnen een dataset

Om te bepalen hoeveel btw-groepen een administratie gemiddeld heeft, is het belangrijk om te kijken naar de grootte en diversiteit van de dataset. Het kan zijn dat sommige administraties meer unieke btw-groepen hebben dan andere administraties, afhankelijk van bijvoorbeeld de aard van hun activiteiten of het land waarin ze opereren. Het is hierdoor ook mogelijk dat er grote variatie bestaat tussen verschillende administraties.

Het onderzoeken van btw-groepen binnen administraties kan waardevolle inzichten bieden in hoe het btw-systeem wordt toegepast in de praktijk. Dit kan op zijn beurt bijdragen aan de verbetering van de modellering van btw-groepen en het nauwkeuriger maken van de voorspellingen van btw-tarieven.

Een ander aspect dat nog verder onderzocht kan worden is hoe de grootte van een btw-groep invloed heeft op de nauwkeurigheid van het model. Het kan bijvoorbeeld zijn dat grotere btw-groepen beter voorspeld kunnen worden dan kleinere groepen, of dat er een bepaalde ondergrens is aan het aantal factuurregels dat nodig is voor een betrouwbare voorspelling.
Het is dus van belang om verder onderzoek te doen naar de grootte en diversiteit van btw-groepen binnen administraties, zodat het voorspellen van btw-tarieven nog accurater kan worden gemaakt.

10. Discussie

Tijdens de uitvoering van het onderzoek en de realisatie van de PoC zijn diverse verbeterpunten naar voren gekomen. In dit hoofdstuk worden deze punten besproken, inclusief de aspecten van projectmanagement en het afstudeerproces. Per onderwerp volgt een toelichting in de onderstaande alinea's.

10.1. Planning en projectmanagement

In de eerste week van de afstudeerperiode is er een planning opgesteld die is opgenomen in het PvA. Echter, er is tijdens het onderzoek weinig aandacht besteed aan deze planning, omdat er regelmatig nieuwe planningen werden opgesteld in overleg met de afstudeerbegeleiders. Dit kwam doordat de oorspronkelijke planning te vaag en niet realistisch bleek te zijn. Uiteindelijk bleek het beter om wekelijks terug te kijken op de voorgaande weken en op basis daarvan een nieuwe planning op te stellen, zonder te ver vooruit te kijken. Het werkte bovendien beter om per dag taken in te plannen om zo meer gedaan te krijgen. Tijdens het onderzoek is een backlog opgesteld in een Kanban-bord op Phabricator. Deze backlog is goed bijgehouden om de status van taken bij te houden. Echter, het afwerken van taken bleek soms lastig vanwege de beperkte inhoudelijke feedback vanuit de opleiding en het af en toe nodig hebben van stimulans.

10.2. Python vs. Ruby

In de eerste weken van het theoretisch kader lag de nadruk op het vinden van een oplossing in Ruby, omdat het grootste deel van het Moneybird-systeem geschreven is in Ruby on Rails. Uiteindelijk bleek het ML-algoritme dat gewenst was, geen ondersteuning te bieden aan een Ruby-oplossing. Het duurde lang voordat deze beslissing werd genomen, wat leidde tot tijdverlies.

10.3. Statistische methoden vs. ML-algoritmes

Tijdens het opstellen van het theoretisch kader werd duidelijk dat statistische methoden ongeschikt waren voor het detecteren van uitschieters op basis van meerdere velden. Er werd veel tijd besteed aan het uitwerken van deze methoden in Ruby, hoewel deze uiteindelijk niet werden gebruikt voor de ontwikkeling van de PoC. Dit leidde tot kostbare tijdverspilling die beter besteed had kunnen worden aan andere belangrijke taken. Een betere definitie van het doel van de taak had dit kunnen voorkomen.

10.4. Validatie in een productieomgeving

Het PoC is niet als functionerende productieomgeving tot stand gekomen, omdat de database niet is gesynchroniseerd met de werkelijke data. Hoewel dit aspect niet binnen de scope van het onderzoek viel, betekent dit dat het PoC alleen kan worden gevalideerd door collega's en niet door uiteindelijke gebruikers. Dit maakt het PoC nog niet geschikt voor gebruik in combinatie met de productieomgeving van Moneybird. Het belang van database synchronisatie wordt in het adviesrapport opgenomen.

11. Literatuurlijst

Barguzar, A. (2022, April 21). *Python Flask versus FastAPI: Which Should You Choose?* Netguru. Retrieved March 15, 2023, from https://www.netguru.com/blog/python-flask-versus-fastapi

Barkved, K. (2022, March 9). *How To Know if Your Machine Learning Model Has Good Performance*. Obviously AI. Retrieved March 15, 2023, from https://www.obviously.ai/post/machine-learning-model-performance

Barkved, K. (2022, March 9). *How To Know if Your Machine Learning Model Has Good Performance*. Obviously AI. Retrieved March 22, 2023, from https://www.obviously.ai/post/machine-learning-model-performance

boekhouders.nl. (2022, October 10). *Wat is een grootboekrekening en welke moet ik gebruiken?* Boekhouders.nl. Retrieved December 15, 2022, from https://www.boekhouders.nl/blog/wat-is-grootboekrekening

BuildKite. (n.d.). Buildkite. Retrieved March 16, 2023, from https://buildkite.com/

C, B. P. (2022, July 5). *How to Detect Outliers in Machine Learning – 4 Methods for Outlier Detection*. freeCodeCamp. Retrieved January 18, 2023, from https://www.freecodecamp.org/news/how-to-detect-outliers-in-machine-learning/

Case Studies | Machine Learning Classifications. (n.d.). Blackbelt Digital. Retrieved January 18, 2023, from https://www.blackbelt.digital/choosing-the-best-classification-model-for-machine-learning/

Chng, J. (2021, August 31). *The F1 score*. Towards Data Science. Retrieved January 31, 2023, from https://towardsdatascience.com/the-f1-score-bec2bbc38aa6

CronJob. (2023, March 9). Kubernetes. Retrieved March 21, 2023, from https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

De Groot, J. (2023, February 8). *What is Data Classification? A Data Classification Definition*. Digital Guardian. Retrieved March 15, 2023, from https://www.digitalguardian.com/blog/what-data-classification-data-classification-definition

The DOT Framework. (2021, July 8). ICT research methods. Retrieved March 14, 2023, from https://ictresearchmethods.nl/The_DOT_Framework

Duffy, M. (n.d.). *Essential Components of Web Accessibility | Web Accessibility Initiative (WAI)*. W3C. Retrieved March 7, 2023, from https://www.w3.org/WAI/fundamentals/components/

Flask. (n.d.). Welcome to Flask — Flask Documentation (2.2.x). Retrieved March 15, 2023, from https://flask.palletsprojects.com/en/2.2.x/

GitHub. (n.d.). GitHub: Let's build from here · GitHub. Retrieved March 16, 2023, from https://github.com/

Guay, M. (n.d.). *What are webhooks?* Zapier. Retrieved March 23, 2023, from https://zapier.com/blog/what-are-webhooks/

Hotwired. (n.d.). Stimulus: A modest JavaScript framework for the HTML you already have. Retrieved March 14, 2023, from https://stimulus.hotwired.dev/

How Much Data Is Needed For Machine Learning? (2022, December 15). Graphite Note. Retrieved March 22, 2023, from https://graphite-note.com/how-much-data-is-needed-for-machine-learning

Hyperparameter Tuning in Python: a Complete Guide - neptune.ai. (2022, December 13). Neptune.ai. Retrieved January 18, 2023, from https://neptune.ai/blog/hyperparameter-tuning-in-python-complete-guide

Informer. (2022, August 24). *Wat is een Journaalpost?* Informer. Retrieved December 15, 2022, from https://www.informer.nl/boekhouden-wiki/wat-is-journaalpost/

Jupyter. (n.d.). Project Jupyter | Home. Retrieved February 2, 2023, from https://jupyter.org/

Kolar, J. A. (2021, December 29). *Double-entry accounting for software engineers*. Balanced. Retrieved January 18, 2023, from https://www.balanced.software/double-entry-bookkeeping-for-programmers/

Machine learning-algoritmen. (n.d.). Microsoft Azure. Retrieved January 18, 2023, from https://azure.microsoft.com/nl-nl/resources/cloud-computing-dictionary/what-are-machine-learning-algorithms/

Myers, W. (2019, March 5). *5 Ways to Detect Outliers That Every Data Scientist Should Know (Python Code)*. Towards Data Science. Retrieved March 15, 2023, from https://towardsdatascience.com/5-ways-to-detect-outliers-that-every-data-scientist-should-know-python-code-70a54335a623

Phabricator. (n.d.). Phacility. Retrieved March 16, 2023, from https://www.phacility.com/phabricator/

pickle — Python object serialization — Python 3.11.1 documentation. (n.d.). Python Docs. Retrieved February 1, 2023, from https://docs.python.org/3/library/pickle.html

Precisie en terugroeping — Wiskunde en Statistiek — DATA SCIENCE. (2020, May 21). DATA SCIENCE. Retrieved January 31, 2023, from https://datascience.eu/nl/wiskunde-statistiek/precisie-en-terugroeping/

PyPi. (n.d.). *pylint · PyPI*. PyPI. Retrieved March 21, 2023, from https://pypi.org/project/pylint/

PyTest. (n.d.). pytest: helps you write better programs — pytest documentation. Retrieved March 21, 2023, from https://docs.pytest.org/en/7.2.x/

Radigan, D. (n.d.). *Kanban - A brief introduction*. Atlassian. Retrieved March 16, 2023, from https://www.atlassian.com/agile/kanban

Regression vs. Classification in Machine Learning: What's the Difference? (2021, October 6). Springboard. Retrieved March 15, 2023, from https://www.springboard.com/blog/data-science/regression-vs-classification/

Rowe, W. (2018, July 5). *Mean Square Error & R2 Score Clearly Explained*. BMC Software. Retrieved January 31, 2023, from https://www.bmc.com/blogs/mean-squared-error-r2-and-variance-in-regression-analysis/

Ruby Corp. (n.d.). Ruby on Rails — A web-app framework that includes everything needed to create database-backed web applications according to the Model-View-Controller (MVC) pattern. Retrieved January 18, 2023, from https://rubyonrails.org/

RubyMine: The Ruby on Rails IDE by JetBrains. (n.d.). JetBrains. Retrieved March 6, 2023, from https://www.jetbrains.com/ruby/

Scharwächter, V. (2022, August 26). *Indruksvaliditeit (Face Validity) | Betekenis & Voorbeelden*. Scribbr. Retrieved March 21, 2023, from https://www.scribbr.nl/onderzoeksmethoden/indruksvaliditeit/

Sharma, R. (2022, September 15). *Top 6 Data Science Programming Languages 2023 [Hand-Picked]*. upGrad. Retrieved January 18, 2023, from https://www.upgrad.com/blog/data-science-programming-languages/

Singer, R. (n.d.). *Shape Up: Stop Running in Circles and Ship Work that Matters*. Basecamp. Retrieved March 16, 2023, from https://basecamp.com/shapeup

SkLearn. (n.d.). *sklearn.metrics.classification_report — scikit-learn 1.2.2 documentation*. Scikit-learn. Retrieved March 21, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

sklearn.ensemble.IsolationForest — scikit-learn 1.2.0 documentation. (n.d.). Scikit-learn. Retrieved January 18, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

sklearn.ensemble.IsolationForest — scikit-learn 1.2.2 documentation. (n.d.). Scikit-learn. Retrieved March 15, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html
sklearn.feature_selection.VarianceThreshold — scikit-learn 1.2.1 documentation. (n.d.). Scikit-learn. Retrieved January 31, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html
Uitleg van Big O notatie. (n.d.). Computer kennis. Retrieved January 18, 2023, from http://www.nldit.com/programmering/computer-programming-languages/201309/86643.html
UI vs. UX Design: What's the Difference? (2022, August 2). Coursera. Retrieved March 16, 2023, from https://www.coursera.org/articles/ui-vs-ux-design
UML Class Diagram Tutorial. (n.d.). Lucidchart. Retrieved March 6, 2023, from https://www.lucidchart.com/pages/uml-class-diagram
What Is Data Preprocessing? (With Importance and Examples). (2022, September 5). Indeed. Retrieved January 20, 2023, from https://ca.indeed.com/career-advice/career-development/data-preprocessing
What Is Data Preprocessing? (With Importance and Examples). (2023, March 12). Indeed. Retrieved March 15, 2023, from https://ca.indeed.com/career-advice/career-development/data-preprocessing
What is Overfitting? - Overfitting in Machine Learning Explained - AWS. (n.d.). Amazon AWS. Retrieved March 6, 2023, from https://aws.amazon.com/what-is/overfitting/
What is prototype? | Definition from TechTarget. (n.d.). TechTarget. Retrieved March 14, 2023, from https://www.techtarget.com/searcherp/definition/prototype
What Is the Client Side? | Definition by Elementor. (n.d.). Elementor. Retrieved April 7, 2023, from https://elementor.com/resources/glossary/what-is-the-client-side/
What is Unsupervised ML? (n.d.). Iguazio. Retrieved March 15, 2023, from https://www.iguazio.com/glossary/unsupervised-ml/
zzpdaily. (n.d.). *Home*. YouTube. Retrieved December 15, 2022, from https://zzpdaily.nl/inkoopfacturen
zzpdaily. (2016, July 16). *Memoriaal en memoriaalboeking uitgelegd - zzp daily*. ZZP Daily. Retrieved December 15, 2022, from https://zzpdaily.nl/memoriaal-memoriaalboeking/
zzpdaily. (2022, June 10). *Verkoopfacturen | Uitleg en voorbeelden*. ZZP Daily. Retrieved December 15, 2022, from https://zzpdaily.nl/verkoopfacturen/

12. Bijlagen

Bijlage 1: Onderzoeksrapport

In dit document wordt het experimentele onderzoek per deelvraag toegelicht. Zo zijn er bij een enkele deelvragen meer uitleg nodig, maar ook worden de resultaten van het experimenteel onderzoek toegelicht.

Deelvraag 3

Welke methode is het meest geschikt om uitschieters binnen een dataset te herkennen?

Voor het experimenteel onderzoek zijn de volgende methodes van relevantie:

- Standard Deviation of in het Nederlands; Normaalverdeling
- Interquartile Range (IQR)
- Isolation Forest

De definitie van de methodes zal in de komende hoofdstukken worden toegelicht, samen met de uitwerkingen van de methodes zelf en de resultaten van de uitwerkingen. Ten slot zal er op basis van deze resultaten een conclusie getrokken worden, die overgenomen wordt in het Resultatenhoofdstuk van het afstudeerverslag.

Benodigde data

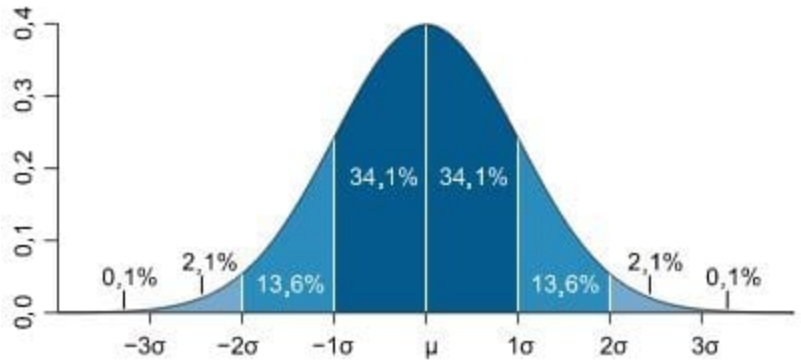
Bij gebruik van statistische technieken, zoals het gebruik van de Normaalverdeling en de IQR-methode, wordt er maar één specifieke parameter binnen een dataset getest. Er kan dus geen verband worden getrokken door twee parameters van een model te combineren. Daarom is er voor gekozen om alleen naar het btw-percentage en de prijs van de verkoopfacturen te kijken en hier de uitschieters uit te halen. Ook wordt de categorie van de verkoopfactuur meegenomen om uiteindelijk handmatig, of met behulp van een ander algoritme een verband te kunnen trekken.

Om alle data binnen te halen die nodig is moet er een commando uitgevoerd worden in het Ruby on Rails console. Om alle data binnen te halen uit de demo-omgeving, moet het volgende commando uitgevoerd worden in de root van de applicatie;

```
$ bin/rails c
$ Detail.pluck(:id, :invoice_type, :ledger_account_id, :tax_rate_id, :price).to_json
```

Hiermee worden alle factuurregels opgehaald, waarbij elke factuurregel op een categorie en een btw-percentage is geboekt. Hiermee haal je ook de totaalprijs per factuur op. De data wordt als JSON-bestand opgehaald, en kan eventueel omgezet worden naar een csv-bestand. Dit bestand kan vervolgens gebruikt worden om de analyse op uit te voeren. De uitwerking hiervan volgt in de komende hoofdstukken.

Standard Deviation / Normaalverdeling
De Normaalverdeling geeft een idee hoe de data binnen een rij van waardes verdeeld is. De meeste waardes binnen de dataset zullen rond het gemiddelde liggen. Hoe verder men van het gemiddelde afwijkt, hoe minder waardes men zal tegenkomen. Met de standaardafwijking wordt bepaald in welke mate de data verspreid is.



Waarbij het gemiddelde is aangegeven met: **μ**, en de standaardafwijking is aangegeven met: **σ**. Stel dat de mediaan een waarde van 10 heeft, en de standaardafwijking een waarde van 1 heeft, dan zullen 68,2% van de waardes tussen de 9 en de 11 liggen. Deze verdeling valt ook te vinden in de bovenstaande grafiek.

Aan de hand van de Z-score wordt naderhand bepaald of een waarde een uitschieter is of niet. De Z-score zegt hoeveel standaardafwijkingen de waarde van de mediaan af ligt. Wanneer de waarde een Z-score heeft van boven de 2, dan wordt dit vaak gezien als een uitschieter. Deze manier van meten zal gebruikt worden bij het uitwerken van de Normaalafwijking op de beoogde dataset.

Uitwerking Normaalverdeling
Voor het uitwerken van de Normaalverdeling is gebruik gemaakt van Jupyter Notebook. Met Jupyter Notebook kunnen eenvoudig wiskundige formules genoteerd worden, maar ook kunnen er code blokken toegevoegd worden die men kan uitvoeren zonder hierbij afhankelijk te zijn van Versiebeheer software, zoals GitHub.

Voor het testen van de implementatie van de Normaalverdeling is er gebruik gemaakt van een gegenereerde dataset, met 1000 rijen aan data bestaande uit 1 relevante kolom, namelijk de "prijs" kolom. De prijs is een getal die altijd tussen de 25 en 35 ligt, maar zijn er een aantal rijen verplaatst met prijzen die hier volledig van afwijken. Deze afwijkende rijen zouden, wanneer goed geïmplementeerd, gedetecteerd moeten worden als zijnde uitschieter.

Voor het ontwikkelen van de Standaardafwijking methode worden twee Ruby library's gebruikt, namelijk Daru en Numo. Daru is een data-analyse library welke tabellen kan plotten en velden kan toevoegen op basis van berekeningen. Voor het aanmaken/muteren/verwijderen van data in bijvoorbeeld Arrays wordt Numo gebruikt. Dit is vergelijkbaar met de NumPy library wanneer er Python gebruikt wordt

Normaalverdeling Notebook Ruby uitwerking
Daru kan het Csv-bestand inlezen welke de Mock Data bevat. De Mock Data zit in dezelfde Google Drive map als het Notebook zelf.

```
# Leest het CSV bestand in
df = Daru::DataFrame.from_csv('MOCK_DATA.csv')

# Print de 'head' van de tabel uit (de eerste 10 waarden)
df.head()
```

Vervolgens zal het gemiddelde en de standaardafwijking over de gehele dataset berekend worden. Deze functionaliteit is inbegrepen in Daru. Daaruit volgt vervolgens een visuele weergave van de data in de vorm van een tabel, die de standaardafwijking en het gemiddelde weergeeft.

```
# Calculate the Mean of the price
mean = df['price'].mean

# Calculate the Standard Deviation of the price
std = df['price'].std

df = Daru::DataFrame.new({
  :mean => [mean],
  :std => [std],
})
```

Daru::DataFrame(1x2)

	mean	std
0	30.106	4.273994739237645

Bovenstaande tabel laat zien dat het gemiddelde van de dataset +- 30,1 is en dat de Standaardafwijking ligt op +- 4,3. Dit betekent dat 68,2% van de dataset tussen de 25,8 en 34,4 ligt en dat 95,4% van de dataset tussen de 21,5 en 38,7 ligt. Alle waardes die hier buiten vallen worden als uitschieter beschouwd. Heeft een element bijvoorbeeld een waarde van 50, dan is dit een uitschieter. Hoeveel standaardafwijkingen de waarde van het gemiddelde afwijkt wordt de Z-score genoemd.

$$Z - score = \frac{\chi - \mu}{\sigma} = (\chi - \mu)/\sigma$$

En wanneer bovenstaande tabel waardes gebruikt worden en een element welke een waarde van 50 heeft:

$$\chi = 50$$

$$\mu = 30,106$$

$$\sigma = 4,274$$

$$Z - score = \frac{\chi - \mu}{\sigma} = (\chi - \mu)/\sigma$$
$$= (50 - 30,106)/4,274 = 19,894/4,274 \sim 4,655$$

Met een Z-score van hoger dan 2 of lager dan -2 wordt het element als een uitschieter beschouwd, dus het element met een waarde van 50 is een uitschieter. De Z-score is namelijk 4,655 en dus hoger dan 2.

Vervolgens zal deze Z-score berekend worden voor elke rij in de dataset en wordt deze toegevoegd aan de tabel om inzicht te krijgen in welke elementen binnen de dataset uitschieters zijn. Ook zal de tijd worden berekend om de prestatie van de methode te kunnen peilen. Op de volgende pagina is de volledige code opgenomen, gevolgd door een tabel met de mogelijke uitschieters/het resultaat.

```
starting = Process.clock_gettime(Process::CLOCK_MONOTONIC)

# Calculate the Mean of the price
mean = df['price'].mean

# Calculate Standard Deviation of the price
std = df['price'].std

# Calculate the Z-score of every element in the dataset

ids = []
prices = []
zScores = []

nArray = Numo::Int32.new(df.nrows).seq(1)
nArray.to_a.each { |index|
  ids << df.row[index-1].data[0]
  prices << df.row[index-1].data[1]
  zScores << (df.row[index-1].data[1]-mean)/std
}

ending = Process.clock_gettime(Process::CLOCK_MONOTONIC)

elapsed = ending - starting

p 'Time elapsed: ' + elapsed.to_s

df = Daru::DataFrame.new({
  :id => ids,
  :price => prices,
  :zScore => zScores,
})


df.where(df[:zScore].lt(-2) | df[:zScore].mt(2))

$ Time elapsed: 0.027027999982237816
```

Resultaat Normaalverdeling

Op basis van bovenstaande resultaten zijn er een aantal voor- en nadelen opgesteld voor het gebruik van de Normaalverdeling bij het detecteren van uitschieters binnen data.

De voordelen van het gebruik van de Normaalverdeling;

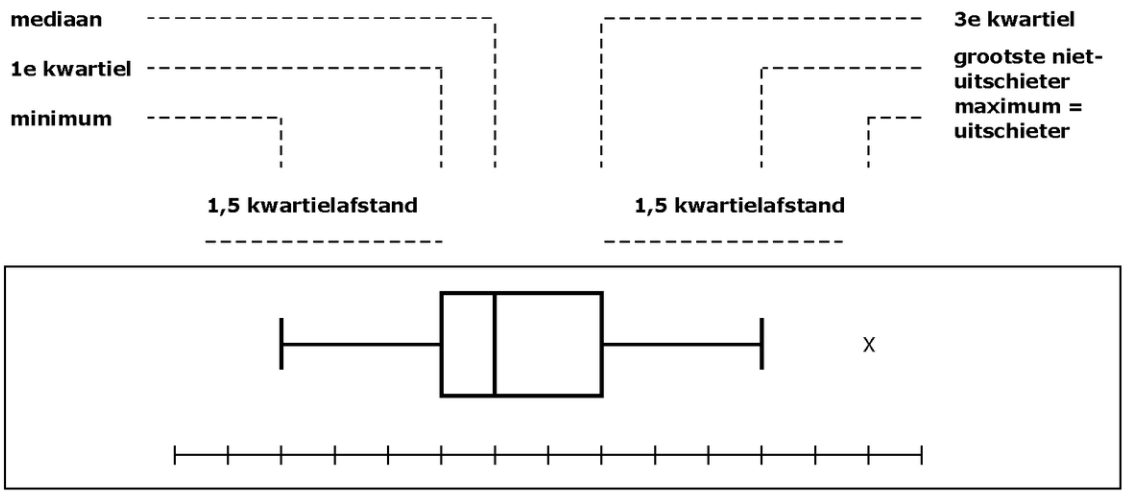
- De Big O notatie (*Uitleg Van Big O Notatie*, n.d.) staat gelijk aan $\mathcal{O}(n)$.
- De Normaalverdeling heeft een hoge accuraatheid wanneer er niet veel uitschieters aanwezig zijn binnen een dataset.
- Het implementeren van het model is mogelijk in Ruby en de nodige documentatie is aanwezig.
- Bij het gebruik van de Z-score kan de ontwikkelaar zelf bepalen bij welk percentage variatie het element binnen de dataset als uitschieter wordt beschouwd.

De nadelen van het gebruik van de Normaalverdeling;

- Wanneer er al veel, en sterk afwijkende, uitschieters binnen een dataset aanwezig zijn zal de accuraatheid van de Normaalverdeling laag zijn. In de praktijk is dit bijvoorbeeld relevant in het geval van Hyperinflatie.
- De Normaalverdeling is een statistische techniek en leert dus niet van haar fouten.
- Bij gebruik wordt er maar rekening gehouden met één veld binnen een dataset. Andere data die eventueel invloed heeft, wordt niet meegenomen in de overweging of een element een uitschieter is of niet.

Concluderend, is het gebruik van de Normaalverdeling binnen een boekhouding een risicovolle keuze. Wanneer de boekhouder/ondernemer bijvoorbeeld zijn/haar boekhouding heeft goedgekeurd voor het jaar 2022, kan deze alsnog extreme uitschieters bevatten die invloed hebben op de berekening. Dit kan als gevolg hebben dat eventuele uitschieters niet gedetecteerd worden.

Interquartile Range (IQR)
De Interquartile Range (IQR) is het verschil in afstand tussen het eerste 'Kwartiel' en het derde 'Kwartiel' van een dataset. De IQR vanaf het eerste en derde kwartiel worden nog als normaal beschouwt. Elke waarde die hier niet binnen ligt, wordt als een uitschieters beschouwd. Dit model wordt vaak gebruikt in datasets die al uitschieters bevatten, en dus niet genormaliseerd zijn. De IQR-methode kan gebruikt worden om de dataset te normaliseren voor data-analyse. In onderstaande afbeelding staat een visuele weergave van de IQR-methode, zonder het gebruik van waardes.



Uitwerking Interquartile Range (IQR)
Voor de uitwerking van de IQR is, net zoals bij de Normaalverdeling, gebruik gemaakt van Jupyter Notebook (Jupyter, n.d.) en de genoemde Mock Data. Ook worden de eerdergenoemde library's, Daru en Numo, gebruikt voor de implementatie van IQR. Eerder benoemde onderdelen, zoals het inlezen van het Csv-bestand, wordt niet meer op ingegaan in dit hoofdstuk en komende hoofdstukken binnen deze onderzoeksvraag.

Op de volgende pagina wordt eerst op basis van een voorbeeld uitgelegd hoe IQR precies werkt. Vervolgens zal de implementatie en tijdproef volgen, net zoals bij de Normaalverdeling. Aan de hand van deze resultaten wordt een conclusie getrokken.

IQR Notebook Ruby uitwerking

De IQR wordt berekend op basis van de Mediaan, de Mediaan van het eerste 'helft' van de dataset en de Mediaan van de tweede 'helft' van de dataset. Om de mediaan van de eerste en tweede helft te kunnen berekenen wordt de array eerst gesorteerd (van klein naar groot) en daarna gesplitst in twee helften. De formule voor het berekenen van de IQR ziet er als volgt uit:

$$IQR = median(Q3) - median(Q1)$$

Vervolgens kan de kleinst mogelijke niet-uitschieter berekend worden met de volgende formule:

$$sNormality = median(\sum) - (1.5 * IQR)$$

En de grootst mogelijke niet-uitschieter met de volgende formule:

$$gNormality = median(\sum) + (1.5 * IQR)$$

3	5	9	9	10	12	12	16	17	25

In bovenstaande tabel is het middelste getal de mediaan, maar omdat je te maken hebt met een even aantal getallen in de tabel is de mediaan het gemiddelde tussen de twee getallen. De mediaan is in dit geval dus $ (10+12)/2 = 11 $.

Wanneer de tabel gesplitst wordt blijven er twee kleinere tabellen over, namelijk:

3	5	9	9	10

12	12	16	17	25

De mediaan, en dus de Q1 van de eerste tabel is in dit voorbeeld 9 en de mediaan van de tweede tabel (en dus de Q3) is 16. De IQR is dus $IQR = median(Q3) - median(Q1) = 16 - 9 = 7.$

En, de laagst/grootst mogelijke niet-uitschieters zijn dus;

$$sNormality = median(\sum) - (1.5 * IQR) = 11 - (1.5 * 7) = 11 - 10.5 = 0.5$$

$$gNormality = median(\sum) + (1.5 * IQR)$$
$$= 11 + (1.5 * 7) = 11 + 10.5 = 21.5$$

In dit voorbeeld is de waarde 25 dus al een uitschieter, sinds 25 hoger is dan de hoogst mogelijke niet-uitschieter (21.5).

```
starting = Process.clock_gettime(Process::CLOCK_MONOTONIC)

# Calculate the Median of the price column
median = df[:price].median

# Sort array in ascending order and split array in half
x = df[:price].to_a()
sortedArray = x.sort()
sections = sortedArray.each_slice(x.size/2).to_a()

q1Array = sections[0]
q3Array = sections[1]
q1 = 0
q3 = 0

## Calculate Median of Q1
if q1Array.size.even?
  q1 = (q1Array[q1Array.size/2]+q1Array[(q1Array.size-1)/2])/2
else
  q1 = q1Array[q1Array.size/2]
end

## Calculate Median of Q3
if q3Array.size.even?
  q3 = (q3Array[q3Array.size/2]+q3Array[(q3Array.size-2)/2])/2
else
  q3 = q3Array[q3Array.size/2]
end

## Calculate the IQR
iqr = q3-q1

## Smallest 'normal' value
sNormality = q1-(1.5*iqr)
## Greatest 'normal' value
gNormality = q3+(1.5*iqr)

ending = Process.clock_gettime(Process::CLOCK_MONOTONIC)
elapsed = ending - starting
p 'Time elapsed: ' + elapsed.to_s

## Filter values based on the lowest 'normal' value and the highest 'normal' value
df.where(df[:price].lt(sNormality) | df[:price].mt(gNormality))

"Time elapsed: 0.0019319998100399997"
```

Resultaat IQR (Interquartile Range)
Op basis van bovenstaande resultaten zijn er een aantal voor- en nadelen opgesteld voor het gebruik van IQR bij het detecteren van uitschieters binnen data.

De voordelen van het gebruik van IQR;

- IQR kan via een inclusieve, maar ook via een exclusieve techniek behandeld worden, waardoor het zowel bij een kleine dataset, als bij een grote dataset goed functioneert.
- IQR heeft een hoge accuraatheid wanneer er niet veel uitschieters aanwezig, maar ook wanneer er wel veel uitschieters aanwezig zijn binnen een dataset.
- Het implementeren van het model is mogelijk in Ruby en de nodige documentatie is aanwezig.

De nadelen van het gebruik van IQR;

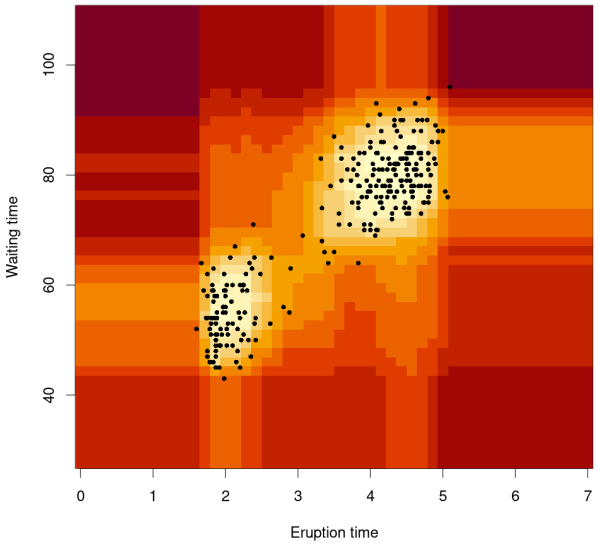
- De Big O notatie (*Uitleg Van Big O Notatie*, n.d.) staat gelijk aan $\mathcal{O}(n \log n)$ * en is dus significant trager dan het gebruiken van de Normaalverdeling.
- Bij gebruik wordt er maar rekening gehouden met één veld binnen een dataset. Andere data die eventueel invloed heeft, wordt niet meegenomen in de overweging of een element een uitschieter is of niet.

* IQR is trager, maar niet als de data standaard al gesorteerd geleverd wordt.

Concluderend, is IQR een veilige methode om uitschieters binnen een dataset te herkennen. Ook wanneer er al uitschieters aanwezig zijn in de boekhouding van de gebruiker. Het is niet complex om te implementeren en is relatief snel. IQR wordt pas bij héél grote datasets problematisch, sinds het sorteren van de lijst dan lang in beslag kan nemen. IQR houdt geen rekening met andere velden binnen een dataset, net zoals de Normaalverdeling.

Isolation Forest
Isolation Forest is een algoritme die abnormaliteit binnen een dataset kan herkennen (*Sklearn.ensemble.IsolationForest — Scikit-Learn 1.2.0 Documentation*, n.d.), dus niet alleen uitschieters, maar ook inconsistentie binnen data. Dit doet het algoritme door het verschil, of afstand, tussen de waardes te berekenen met behulp van Binary Trees. In tegenstelling tot de eerder genoemde methodes is Isolation Forest een algoritme en niet simpele statistiek. Dit betekent ook dat Isolation Forest bij kleine datasets meer tijd in beslag neemt, maar minder tijd in beslag neemt bij grotere datasets.

Het algoritme geeft, wanneer het klaar is met verwerken van de data, een indicatie of een waarde een abnormaliteit is. Dit resultaat zal verwerkt worden in een grafiek, bijvoorbeeld in de vorm van een zogenaamde Heatmap. Een voorbeeld van een dergelijke Heatmap is weergegeven in onderstaande afbeelding.



Uitwerking Isolation Forest
Voor de uitwerking van Isolation Forest is niet van dezelfde library's gebruik gemaakt als bij de Normaalverdeling en IQR. Sinds Isolation Forest een complex algoritme is, is er gebruik gemaakt van een bestaande library die het gebruik van Isolation Forest mogelijk maakt. Isolation Forest is afhankelijk van de Rover-library, die het mogelijk maakt om gebruik te maken van data in de vorm van een Csv-bestand. Rover is vergelijkbaar met het eerder gebruikte Daru, alleen bood de Isolation Forest library geen ondersteuning voor het gebruik van Daru, waardoor voor Rover gekozen is.

Op de volgende pagina wordt een deel van het algoritme uitgelegd en worden de resultaten toegelicht, ondersteund door de code-implementaties.

Isolation Forest Notebook Ruby uitwerking

Isolation Forest is aan algoritme waarmee uitschieters binnen een dataset gedetecteerd kunnen worden. Isolation Forest maakt gebruik van 'isolation', het kijkt naar de afstand / het verschil tussen alle waarden binnen de data en kijkt zo naar elementen die geïsoleerd zijn van elkaar. Isolation Forest deelt de dataset op in delen, een trainingsset en een test set. Vervolgens krijgt elke waarde een score tussen de 0 en de 1. Hoe dichter de waarde bij de 1 zit, hoe waarschijnlijker er het is het een uitschieter betreft. Een voordeel hiervan is, dat je als ontwikkelaar zelf kan bepalen waar de grens ligt. Ook kunnen Hyperparameters meegegeven worden om het algoritme naar wens te tunen (*Hyperparameter Tuning in Python: A Complete Guide - Neptune.ai*, 2022).

Er zijn een aantal 'hoofdregels' opgesteld die een element binnen een dataset een uitschieter maken, namelijk;

- Als het resultaat dicht bij 1 zit is het hoogst waarschijnlijk een uitschieter
- Als het resultaat kleiner is dan 0.5 is het resultaat waarschijnlijk 'normaal'
- Als het resultaat van vrijwel alle elementen rond de 0.5 zit dan kan er met grote zekerheid gezegd worden dat de dataset geen uitschieters bevat

```
# Leest het CSV bestand in
data = Rover::read_csv('MOCK_DATA.csv')

starting = Process.clock_gettime(Process::CLOCK_MONOTONIC)

model = IsoTree::IsolationForest.new
model.fit(data)

pred = model.predict(data)

indexes = []

pred.each_with_index do |item,index|
    if item > 0.6
      indexes << index+1
    end
end

ending = Process.clock_gettime(Process::CLOCK_MONOTONIC)

elapsed = ending - starting

p 'Time elapsed: ' + elapsed.to_s

data[data['id'].in?(indexes)]

$ "Time elapsed: 0.42666200036183"
```

Resultaat Isolation Forest

Op basis van bovenstaande resultaten zijn er een aantal voor- en nadelen opgesteld voor het gebruik van Isolation Forest bij het detecteren van uitschieters binnen data.

De voordelen van het gebruik van Isolation Forest;

- De Big O notatie (*Uitleg Van Big O Notatie*, n.d.) staat gelijk aan $\mathcal{O}(n)$.
- Er wordt rekening gehouden met alle velden binnen een dataset.
- Isolation Forest heeft in vrijwel elk scenario een hoge accuraatheid, en heeft de mogelijkheid tot het gebruik van Hyperparameter tuning.

De nadelen van het gebruik van Isolation Forest;

- Het gebruik vergt de nodige kennis over Hyperparameter tuning.
- Omdat Isolation Forest naar elk veld in de data-set kijkt is het wat trager dan de andere methodes en dus moet de irrelevante velden uit de data-set verwijderd worden om beter te presteren.
- Kans op over- en underfitting.

Concluderend, is het gebruik van Isolation Forest een veilige keuze in bijna elk scenario. Isolation Forest is snel, is accuraat in bijna alle gevallen en het algoritme houdt rekening met alle velden binnen een data-set. Zo kunnen verbanden getrokken worden om tot het beste resultaat te komen. Ook kunnen de Hyperparameters 'getuned' worden, zodat de accuraatheid van het algoritme alleen maar hoger wordt. Wel is het hierbij van belang rekening te houden met overfitting. Overfitting is een verschijnsel waarin het algoritme te goed getraind wordt op de data-set, waardoor het algoritme incorrecte voorspellingen maakt wanneer het geconfronteerd wordt met een andere data-set. Over- of underfitting is waar te nemen door de implementatie op verschillende data-sets te testen en resultaten hiervan te vergelijken.

Deelvraag 4
Welke methode is het meest geschikt om inconsistenties binnen een dataset te herkennen?

Om inconsistenties binnen een dataset te herkennen is het belangrijk om te begrijpen wat er als een inconsistentie wordt beschouwd. Een inconsistentie kan gezien worden als een mogelijke fout binnen data, waar de data niet overeenkomt met de andere data binnen een dataset, of niet aan de verwachting van een systeem voldoet.

Het meest simpele voorbeeld hiervan is het btw-percentage op de in- en verkoop van fruit. Deze is door de overheid vastgesteld op 6%. Wanneer een dergelijke transactie vervolgens geboekt wordt in Moneybird op een ander btw-percentage als 6%, dan is dit een inconsistente transactie in vergelijking met de andere transacties. Wanneer een ondernemer bijvoorbeeld elke week € 40 aan fruit koopt en dit op een verkeerd btw-percentage boekt, zal dit invloed hebben de btw-aangifte die de ondernemer doet. Dit kan de ondernemer geld kosten of meer geld aan overhouden dan de bedoeling is. Dit soort situaties dienen voorkomen te worden.

Er zijn vele manieren om inconsistenties binnen een dataset te herkennen. Deze manieren worden vaak onder dezelfde terminologie geschaard, namelijk Data Classificatie. Wanneer het gegeven scenario/probleem lineair is, is het gebruik van SVM of Logistic Regression de beste optie (*Case Studies | Machine Learning Classifications*, n.d.). Is het gegeven scenario/probleem non-lineair, dan is het gebruik van Classificatie Algoritmes zoals Naive Bays en/of Random Forest de beste optie.

Problemen die gebasseerd zijn op categorieen, zijn non-lineair. Het beste voorbeeld hiervan is eigenlijk het scenario welke in dit onderzoek behandeld wordt. Een klassiek voorbeeld van een lineair probleem is gezichtsherkenning. Wanneer men bijvoorbeeld de kleur van ogen wil voorspellen, wordt er gekeken naar een aantal eigenschappen die een lineair verband hebben. Is de kleurintensiteit hoger, dan hebben de ogen een blauw/groene kleur, is de kleurintensiteit lager dan zijn de ogen wat bruiner. Dit gegeven veranderd nooit en is dus een lineair probleem. Dit is in het scenario, wat behandeld wordt in dit verslag, dus niet het geval. Er zal dus gekeken worden naar een aantal Classificatie Algoritmes, namelijk:

- Decision Trees
- Random Forest
- Naive Bays

De algoritmes zijn gekozen op basis van de criteria in de beslissingsmatrix en op de uitleg die boven de opsomming gegeven is.

Fase 1: Data verzameling
Eerder in het afstudeerverslag is er een tabel toegelicht met alle relevante parameters binnen de toegelichte classes. Deze parameters zullen gebruikt worden om Mockdata te genereren. Dit zal op dezelfde manier gebeuren als het geval was bij de vorige deelvraag in dit onderzoeksrapport. Ook hiervoor zal Jupyter Notebook gebruikt worden om de verschillende methodes, waarin de Mockdata gebruikt wordt, uit te werken.

Fase 2: Data preprocessing
Voor het preprocessen van data zijn er heel veel verschillende methodes, die elk hun kracht hebben. Zo zijn er methodes die beter werken met het voorspellen van discrete waardes, maar zijn er ook methodes die beter werken met het voorspellen van continue waardes. Een discrete waarde kan bijvoorbeeld een categorie of een btw-percentage zijn, deze waardes veranderen vrijwel nooit. Een continue waarde is bijvoorbeeld een prijs, welke bij elke transactie schommelt.

Daarnaast zijn er verschillende soorten methodes binnen data preprocessing, namelijk Machine Learning algoritmes en statistische methodes. Om verschillende scenario's te kunnen dekken, zijn er methodes gekozen die zowel discrete als continue waardes kunnen voorspellen. Ook worden zowel Machine Learning algoritmes als statistische methodes gebruikt, of een combinatie hiervan. De volgende methodes zijn gekozen op basis van populariteit en ondersteuning:

- SelectKBest
- Chi squared
- RFE
- RFECV
- RandomForestClassifier
- VarianceThreshold
- SelectFromModel
- LogisticRegression

Zaak is om tijdens het uitwerken van bovenstaande methodes, onderzoek te doen naar de meest geschikte methode voor het beoogde scenario. In de komende hoofdstukken zullen de toelichtingen en code uitwerkingen per methode volgen.

Methode 1 (SelectKBest + Chi squared + RFE)
SelectKBest, Chi squared en RFE zijn methodes welke vaak in combinatie gebruikt worden om elkaar te "versterken". Daarom is er voor gekozen hetzelfde te doen bij het uitwerken van de methode. Deze methodes maken allemaal gebruik van Regressieanalyse en zijn dus vooral sterk in het voorspellen van continue waardes. In het scenario van het voorspellen van btw-percentages in een verkoopfactuur zal deze methode dus logischerwijs minder moeten presteren.

Uitwerking SelectKBest + Chi squared + RFE

```
array = dataframe.values
X = array[:,0:4]
Y = array[:,4]


# Feature extraction
test = SelectKBest(score_func=chi2, k=3)
fit = test.fit(X, Y)

# Summarize scores
np.set_printoptions(precision=3)
print(fit.scores_)

features = fit.transform(X)
# Summarize selected features
print(features[0:5,:])

# Feature extraction
model = LogisticRegression(max_iter=1000)
rfe = RFE(model, n_features_to_select=2, step=1)
fit = rfe.fit(X, Y)
print("Num Features: %s" % (fit.n_features_))
print("Selected Features: %s" % (fit.support_))
print("Feature Ranking: %s" % (fit.ranking_))
```

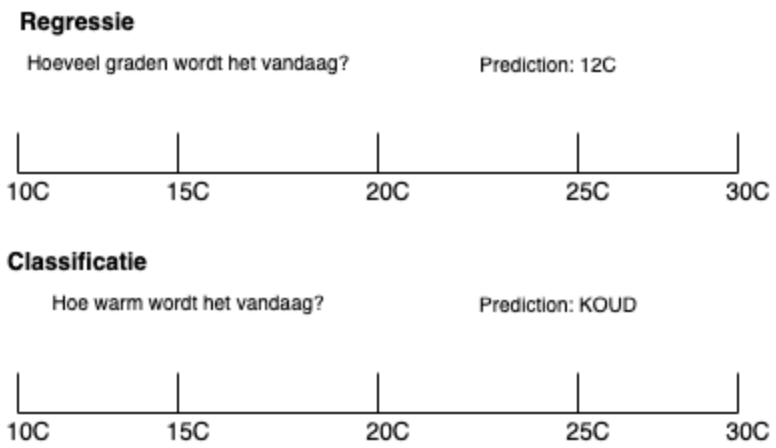
Resultaat SelectKBest + Chi squared + RFE

Bij de uitwerking van bovenstaande methode is het belangrijk om er rekening mee te houden dat er van tevoren aangegeven moet worden hoeveel features er in de dataset moeten overblijven. Dit is nadelig wanneer het geen gegeven is hoeveel features belangrijk zijn voor het beoogde scenario.

SelectKBest en Chi squared zijn methodes die gebruik maken van regressie, dus die voorspellingen maken op basis van lineaire problemen. Wanneer het btw-percentage voorspeld wordt binnen de Mock data, is dit dus geen ideale methode, sinds een btw-percentage een discrete waarde is. Aan onderstaand resultaat is dit ook te zien, sinds de methode twee features uitkiest met een continue waarde, namelijk de prijs en de datum.

```
Num Features: 2
Selected Features: [False  True False  True]
Feature Ranking: [2 1 3 1]
```

In eerste instantie leken de resultaten dus tegen te vallen, maar vallen te verklaren door hetgeen waar regressieanalyse methodes voor bedoeld zijn. Om het verschil tussen Classificatie en Regressie wat te verduidelijken is onderstaand een afbeelding opgenomen.



Concluderend, zijn Regressieanalyse methodes niet geschikt voor het beoogde scenario. In plaats daarvan zal er gekeken moeten worden naar methodes die gebruik maken van Classificatie methodes.

Methode 2 (RFECV + RandomForestClassifier)
RFECV is een methode die recursief features verwijderd uit een dataset op basis van een gekozen algoritme en dit valideert aan de hand van cross-validatie. Het gebruikte algoritme wordt ook wel de estimator genoemd. Als estimator is er gekozen voor de RandomForestClassifier methode. Dit is een methode die features selecteert en het best presteert wanneer er discrete waardes gebruikt worden, zoals in het beoogde scenario: Categorieën (Ledger Accounts) en btw-percentages.

Uitwerking RFECV + RandomForestClassifier

```
estimator = RandomForestClassifier(random_state =42)
selector = RFECV(estimator, step=1)
fit = selector.fit(X, Y)

print("Num Features: %s" % (fit.n_features_))
print("Selected Features: %s" % (fit.support_))
print("Feature Ranking: %s" % (fit.ranking_))
```

Resultaat RFECV + RandomForestClassifier
Bij het gebruik van de RFECV en RandomForestClassifier methodes worden er features uitgekozen welke geschikt zijn voor het voorspellen van discrete waardes. Logischerwijs zullen er dan vaker features uitgekozen worden welke ook discreet zijn. Onderstaand resultaat is een weergave van de gekozen features, waarbij de categorie en het contact_id als relevant worden gezien.

```
Num Features: 2
Selected Features: [ True False  True False]
Feature Ranking: [1 3 1 2]
```

Concluderend is de RandomForestClassifier methode geschikt voor het beoogde scenario, sinds het geschikt is voor het classificeren van discrete waardes en voor supervised learning. Over de performance valt nog niet veel te zeggen, daarvoor zou eerst een realistische dataset gebruikt moeten worden.

Methode 3 (VarianceThreshold)
VarianceThreshold is een methode waarmee features worden verwijderd op basis van de variatie binnen één en dezelfde feature. Wanneer een feature bijvoorbeeld altijd een waarde van 0 heeft, zal deze feature geen invloed hebben op de rest van de features, en dus zal deze worden verwijderd. VarianceThreshold werkt niet wanneer er gewerkt wordt met discrete waardes, zoals categorieën en btw-percentages.

Uitwerking VarianceThreshold

```
selector = VarianceThreshold()
selector.fit(X, Y)
print(selector.get_support())
```

Resultaat VarianceThreshold
VarianceThreshold is getest op de eerder genoemde Mock-dataset en gaf als resultaat geen één feature te elimineren. Dit betekent dat de dataset geen features bevat waarbij de variatie binnen waardes nihil is. VarianceThreshold is een snelle manier om features te elimineren die op het oog vaak al irrelevant leken, en wordt daarom vaak gebruikt voor unsupervised learning (*Sklearn.feature_selection.VarianceThreshold — Scikit-Learn 1.2.1 Documentation*, n.d.).

Concluderend, is VarianceThreshold eveneens niet geschikt om te gebruiken in het beoogde scenario. Wanneer een model getraind wordt om een btw-percentage te trainen, zal het model van eerdere transacties moeten weten wat het geboekte btw-percentage is. Dit scenario zal dus niet gebruik maken van unsupervised learning, maar van supervised learning.

Methode 4 (SelectFromModel + LogisticRegression)
SelectFromModel en LogisticRegression werken eveneens het beste met continue waardes. Het SelectFromModel model kijkt naar het gemiddelde van feature X en wijst op basis daarvan elke feature een score aan. In de realiteit zou dit betekenen dat alle feature met discrete waardes, verwijderd worden.

Uitwerking SelectFromModel + LogisticRegression

```
selector = SelectFromModel(estimator=LogisticRegression(max_iter=1000)).fit(X, Y)
selector.estimator_.coef_

selector.threshold_

selector.get_support()
```

Resultaat SelectFromModel + LogisticRegression
Net zoals SelectKModel en Chi squared, is LogisticRegression een methode die gebruik maakt van Regressieanalyse. Regressieanalyse methodes niet geschikt voor het beoogde scenario. In plaats daarvan zal er gekeken moeten worden naar methodes die gebruik maken van Classificatie methodes.

Decision Trees
Decision Trees is een model waarmee een model getraind wordt welke leert van data die vooraf als correct is bestempeld (supervised learning). Vervolgens kan het model voorspellingen maken op basis van deze training. Het model kan voorspellingen maken op basis van verschillende input en kan dus goed gebruikt worden om inconsistenties binnen data te vinden. Decision Trees is een methode welke gebruikt gemaakt van Regressieanalyse en is dus het meest geschikt voor het voorspellen van continue waardes, zoals eerder in dit document toegelicht.

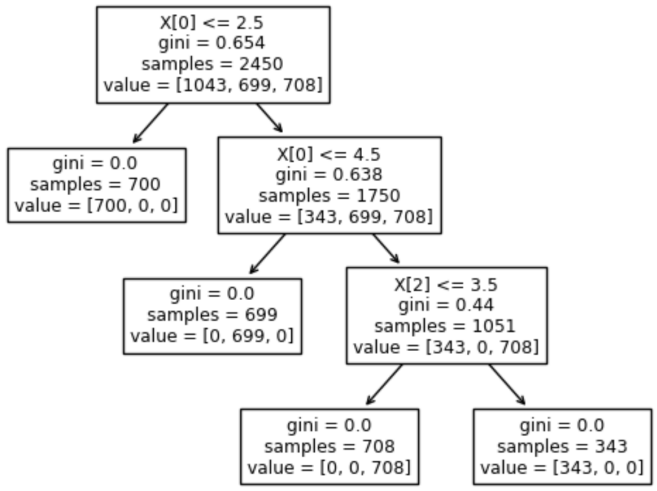
Voordat het model wordt uitgewerkt in code, zal een Feature Extraction algoritme de irrelevante input verwijderen en zullen eventuele blanco velden of extreme uitschieters worden verwijderd uit de dataset. Het is namelijk het beste voor het model om met kwalitatieve data te werken en dus niet met eventuele uitschieters. Vervolgens zal aan de hand van de F1-score bepaald worden hoe accuraat het model is, en of er sprake is van under- of overfitting. Er is bij het uitwerken van Decision Trees ervoor gekozen om de methode uit te werken in Python. Ruby biedt geen goed onderhouden library, en het zelf ontwikkelen van Decision Trees in Ruby zou veel tijd in beslag nemen.

Uitwerking Decision Trees Python Notebook

```
# Read the Mock Data
df = pd.read_csv('INC_MOCK_DATA.csv')

print(df.head())

# Test size = 30%, Train size = 70%
train, test = train_test_split(df, test_size=0.3)

# Define train columns and the target column 'vat'
# The target column is the column to be predicted
train_columns = train.drop([label_to_predict], axis=1)
train_target_column = train[label_to_predict]

# Define DecisionTreeRegressor and fit the data into the model
clf = tree.DecisionTreeClassifier()
clf = clf.fit(train_columns.values, np.ravel(train_target_column))

tree.plot_tree(clf)

# Supervised accuracy measurement
test_columns = test.drop([label_to_predict], axis=1)
predictions = clf.predict(test_columns.values)

score = r2_score(test[label_to_predict], predictions)

print('\n')
print(score)
```

Resultaat Decision Trees

In bovenstaande code is de Decision Trees methode uitgewerkt. Decision Trees is een methode die gebruik maakt van Classificatie en als het ware een boom met voorwaarden opbouwt om tot een voorspelling te komen. Hiervoor is de gegenereerde Mock-dataset gebruikt. Classificatie is, vooral wanneer er gewerkt wordt met het voorspellen van discrete waardes, geschikt. De boom werkt namelijk met voorwaarden die gebaseerd zijn op discrete waardes. Op basis hiervan zou de boom logischerwijs worden opgebouwd uit discrete waardes, in dit geval het contact_id en het ledger_account_id.



X[0] = contact_id
X[2] = ledger_account_id

Bovenstaande boom is gegenereerd op basis van het resultaat en voldoet aan de verwachting dat alleen de discrete waardes als voorwaarden binnen de nodes worden gebruikt. Bij het gebruik van meer features, of als er veel te voorspellen scenario's aanwezig zijn, kan het voorkomen dat de boom overcomplex wordt. Dit kan voorkomen worden door Hyperparameter tuning toe te passen, maar dit zou pas toegepast kunnen worden op een dataset die daadwerkelijk in gebruik is.

De methode produceert op de Mock-dataset een R2 score (Rowe, 2018) van 1.0, wat betekent dat de methode bij alle bestaande, maar ook nieuwe transacties het btw-percentage goed weet te voorspellen.

Random Forest

Net zoals de Decision Trees methode is Random Forest een methode waarbij zowel Regressieanalyse als Classificatie toegepast kan worden om waardes te voorspellen. Eerder in dit document is toegelicht dat Classificatie de beste methode is om discrete waardes te voorspellen en dus het meest geschikt is voor het beoogde scenario, namelijk het voorspellen van het btw-percentage binnen een verkoopfactuur.

Uitwerking Random Forest Python Notebook

```
# Read the Mock Data
df = pd.read_csv('INC_MOCK_DATA.csv')

print(df.head())

# Test size = 30%, Train size = 70%
train, test = train_test_split(df, test_size=0.3)

# Define train columns and the target column 'vat'
# The target column is the column to be predicted
train_columns = train.drop([label_to_predict], axis=1)
train_target_column = train[label_to_predict]

clf = RandomForestClassifier(random_state=42)
clf.fit(train_columns.values, np.ravel(train_target_column))

# Supervised accuracy measurement
test_columns = test.drop([label_to_predict], axis=1)
# Naive Bayes confuses indexes with true y_predictions, so the index label should
be dropped
test = test.reset_index(drop=True)
predictions = clf.predict(test_columns.values)

report = classification_report(test[label_to_predict], predictions)

print(report)
```

Resultaat Random Forest

Voor de Random Forest methode geldt dat de mate van accuraatheid wordt uitgedrukt in een F1-score (Chng, 2021). De F1 score is eigenlijk het harmonisch gemiddelde van de precisie en de terugroeping (*Precisie En Terugroeping — Wiskunde En Statistiek — DATA SCIENCE*, 2020). De F1-score ligt, net zoals de R2 score, altijd tussen de 0 en de 1, waarbij een score van 1.0 aangeeft dat er een model is geproduceerd welke altijd accuraat is bij het voorspellen van waardes.

	precision	recall	f1-score	support
1	1.00	1.00	1.00	447
2	1.00	1.00	1.00	299
3	1.00	1.00	1.00	304
accuracy			1.00	1050
macro avg	1.00	1.00	1.00	1050
weighted avg	1.00	1.00	1.00	1050

1 = btw vrijgesteld
2 = 21% btw
3 = 6% btw

In bovenstaande tabel kunnen de scores teruggevonden worden per btw-percentage. Het algoritme is getest op een test-dataset, die los staat van de dataset waarop het model getraind is. Dit betekent dat het algoritme een accuraatheid van 100% heeft op de Mock-dataset. Dit biedt geen garantie dat dit in de werkelijkheid dezelfde accuraatheid bereikt wordt.

Naive Bayes
Naive Bayes is een methode waarbij zowel Regressieanalyse als Classificatie toegepast kan worden. Voor het beoogde scenario is, zoals al eerder toegelicht in dit document, Classificatie nodig en dus is in onderstaande uitwerking gebruik gemaakt van de Classificatie variant van Naive Bayes.

Uitwerking Naive Bayes

```
# Read the Mock Data
df = pd.read_csv('INC_MOCK_DATA.csv')

print(df.head())

# Test size = 30%, Train size = 70%
train, test = train_test_split(df, test_size=0.3)

# Define train columns and the target column 'vat'
# The target column is the column to be predicted
train_columns = train.drop([label_to_predict, 'price', 'date'], axis=1)
train_target_column = train[label_to_predict]

# Define the model
gnb = CategoricalNB()
gnb = gnb.fit(train_columns.values, np.ravel(train_target_column))

# Supervised accuracy measurement
test_columns = test.drop([label_to_predict, 'price', 'date'], axis=1)
score = gnb.score(test_columns.values, test[label_to_predict])
predictions = gnb.predict(test_columns.values)

report = classification_report(test[label_to_predict], predictions)

print(score)

print(report)
```

Resultaat Naive Bayes
Voor de Naive Bayes methode geldt dat de mate van accuraatheid wordt uitgedrukt in een F1-score (Chng, 2021). De F1 score is eigenlijk het harmonisch gemiddelde van de precisie en de terugroeping (*Precisie En Terugroeping — Wiskunde En Statistiek — DATA SCIENCE*, 2020). De F1-score ligt, net zoals de R2 score, altijd tussen de 0 en de 1, waarbij een score van 1.0 aangeeft dat er een model is geproduceerd welke altijd accuraat is bij het voorspellen van waardes.

Een belangrijk onderdeel van het resultaat is, dat Naive Bayes alleen werkt wanneer alle continue waardes verwijderd moeten worden uit de dataset voordat het algoritme kan functioneren. In eerste instantie was de accuraatheid heel laag, echter kwam dat door de continue features die nog aanwezig waren. Dit probleem kan voorkomen worden door zowel manueel als met een Classificatie algoritme de continue features te verwijderen (zie Deelvraag 4, *preprocessing*).

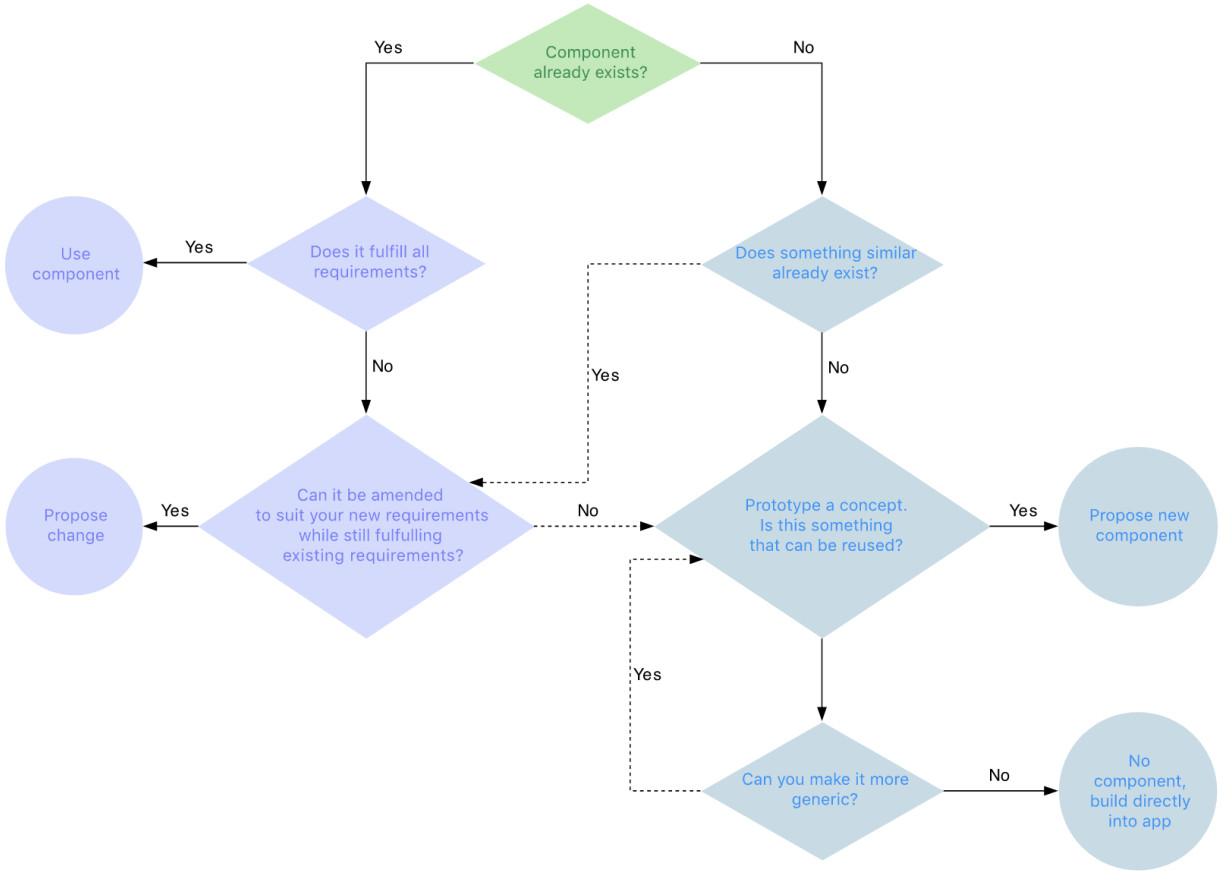
	precision	recall	f1-score	support
1	0.90	1.00	0.95	449
2	1.00	1.00	1.00	295
3	1.00	0.84	0.91	306
accuracy			0.95	1050
macro avg	0.97	0.95	0.95	1050
weighted avg	0.96	0.95	0.95	1050

1 = btw vrijgesteld
2 = 21% btw
3 = 6% btw

In bovenstaande tabel kunnen de scores teruggevonden worden per btw-percentage. Het algoritme is getest op een test-dataset, die los staat van de dataset waarop het model getraind is. Dit betekent dat het algoritme een accuraatheid van 95% heeft op de Mock-dataset. Dit biedt geen garantie dat dit in de werkelijkheid dezelfde accuraatheid bereikt wordt.

Bijlage 2: Design System Component Diagram

Om te bepalen of er een nieuw component nodig is bij het ontwikkelen van een stuk software, kan de onderstaande flowchart worden gevolgd. Het groene component is het startpunt, waarbij de vraag wordt gesteld of het benodigde component al bestaat of niet.



Bijlage 3: Interviews Product Team

In totaal hebben er twee interviews plaatsgevonden met Marijn en Tim van het Product Team. Dit team is onder andere verantwoordelijk voor het ontwikkelen van designs voor de Moneybird applicatie.

Interview A: 3 februari 2023

Tijdens het eerste ongestructureerde interview met Marijn en Tim is besproken hoe het ontwerpen van een applicatie binnen Moneybird verloopt. Uit het interview bleek dat er bij Moneybird een Design System bestaat met herbruikbare componenten die gedocumenteerd zijn in een intern systeem en toegankelijk zijn in de dev-omgeving. Tijdens de interactieve demo werd het Design System gedetailleerd getoond.

Vervolgens werd er gekeken naar het beoogde scenario en welke componenten daarbij wel of niet passen. Het Alert component werd als geschikt bevonden omdat het de gebruiker kan waarschuwen voor fouten bij het aanmaken van factuurregels in een verkoopfactuur. Er werd ook besproken waar de limieten liggen en waar men de do's en don'ts kan vinden bij het ontwerpen van een applicatie.

Interview B: 2 maart 2023

Tijdens het tweede ongestructureerde interview met Marijn en Tim is er iets dieper ingegaan op het daadwerkelijke ontwerp wat ik voor ogen had. Hierbij is er de conclusie getrokken dat een Alert-component niet voldoet aan de gestelde limieten en is er in samenspraak een niet ontwerp gemaakt. Hierbij is gebruik gemaakt van een ander component, namelijk het Tooltip component met een bijbehorend label.

Bijlage 4: Strokenplanning

Maand:	Nov		December				Januari				Februari				Maart					April			
Jaarweek:	47	48	49	50	51	52	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Schoolweek:	1.1	1.2	1.3	1.4	1.5	V	V	1.6	1.7	1.8	1.9	1.10	2.1	2.2	V	2.3	2.4	2.5	2.6	2.7	2.8	2.9	2.10
Onderdeel 1: Kennismaking																							
Plan van Aanpak																							
Onderdeel 2: Onderzoek																							
Deelvraag 1																							
Deelvraag 2																							
Deelvraag 3																							
Deelvraag 4																							
Deelvraag 5																							
Deelvraag 6																							
Deelvraag 7																							
Onderdeel 3: Features																							
n.t.b.																							
n.t.b.																							
n.t.b.																							
n.t.b.																							
Onderdeel 4: Eindproduct																							
Technisch ontwerp																							
Functioneel ontwerp																							
Scriptie																							
Evaluatie formulier																							
Presentatie																							

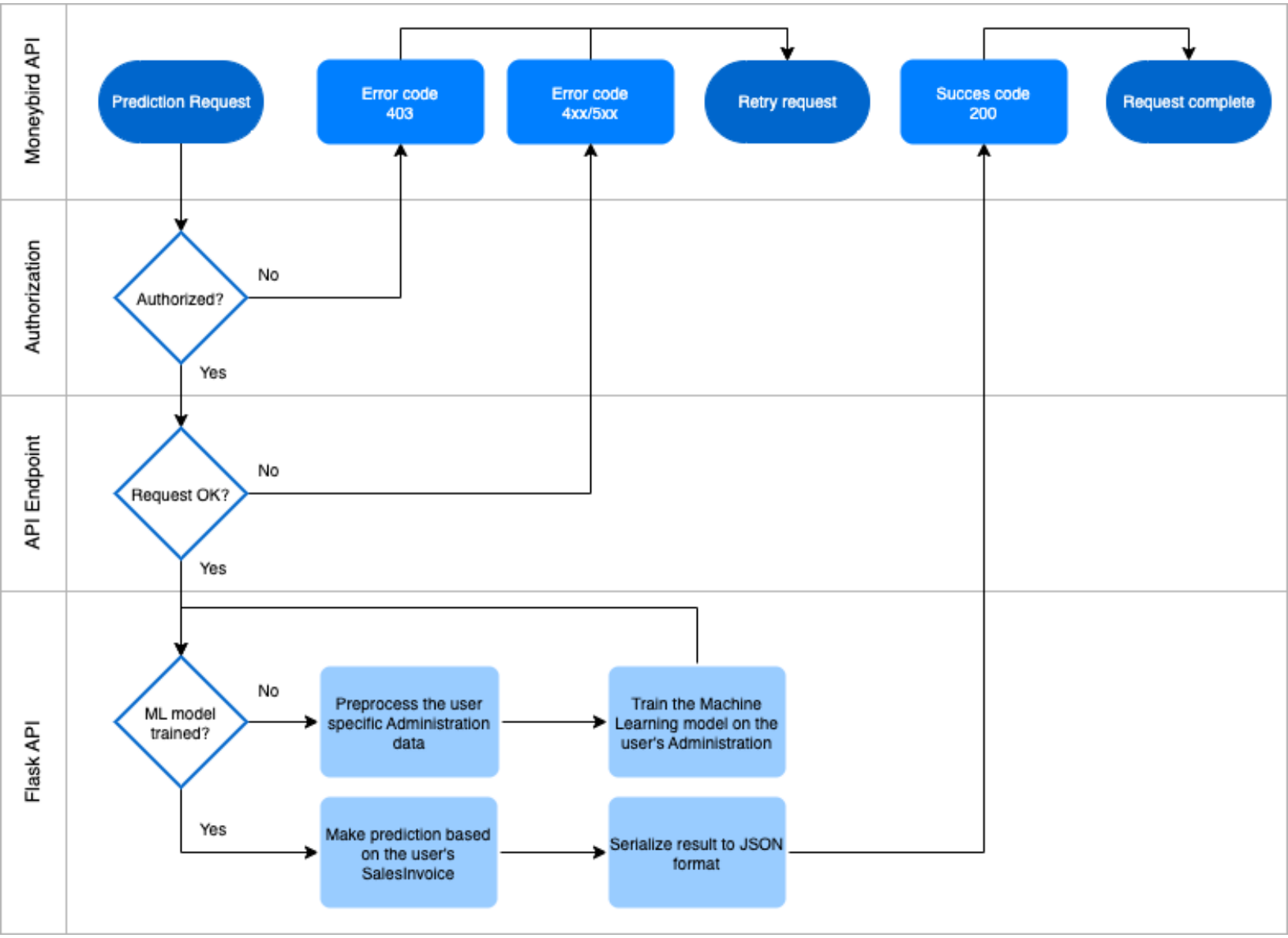
Bijlage 5: Backlog

De backlog zoals die initieel is opgesteld. De taken zijn geprioriteerd en worden bijgehouden in een intern systeem bij Moneybird genaamd Phabricator.

T26093	Eindmodel exporteren naar Pickle	**T26092**	Schrijven hoofd conclusie
T26086	DB snapshot verkrijgen en gereed maken voor gebruik in de API	**T26091**	Schrijven aanbeveling
T26084	Linter verwerken in API	**T26090**	Schrijven adviesrapport
T26083	Voorspellingen endpoint verwerken in API	**T26089**	Voorspelling API request realiseren in MB2
T26082	Preprocessing model verwerken in API	**T26080**	Valideren Design bij collega's/gebruikers
T26081	Trainingsmodel verwerken in API	**T24983**	Deelvraag 5: Hoe kan een zo gebruiksvriendelijk mogelijke GUI worden ontwikkeld?
T26079	Design/GUI ontwerp maken en verwerken in het verslag	**T24984**	Deelvraag 6: Hoe kunnen fouten onder de aandacht van gebruikers gebracht worden?
T26078	Systeemarchitectuur maken en verwerken in verslag	**T26088**	Design realiseren in MB2
T26077	Opzetten GIT repo	**T26085**	Unit tests schrijven voor functionaliteit in de API
T26076	Flask API initiele opzet	**T26087**	CI/CD pipeline opzetten en koppelen aan API
T25775	Hoofdstuk testplan		
T25774	Hoofdstuk tools		
T25773	Hoofdstuk kwaliteitsbewaking		
T25772	Hoofdstuk ontwikkelmethodes		
T25771	Requirements opstellen		
T25770	Backlog opstellen		
T25769	Conclusie hoofdvraag		
T24982	Deelvraag 4: Welke methode is het meest geschikt om inconsistenties binnen een dataset te herkennen?		
T24981	Deelvraag 3: Welke methode is het meest geschikt om uitschieters binnen een dataset te herkennen?		
T24980	Deelvraag 2: Welke data is relevant binnen het platform van Moneybird voor het vinden van fouten, inconsistenties en uitschieters binnen de boekhouding van de gebruiker?		
T24979	Opzet afstudeerverslag		
T24978	Deelvraag 1: Welke relevanten technieken wordt gebruik van gemaakt?		
T24766	Plan van Aanpak		

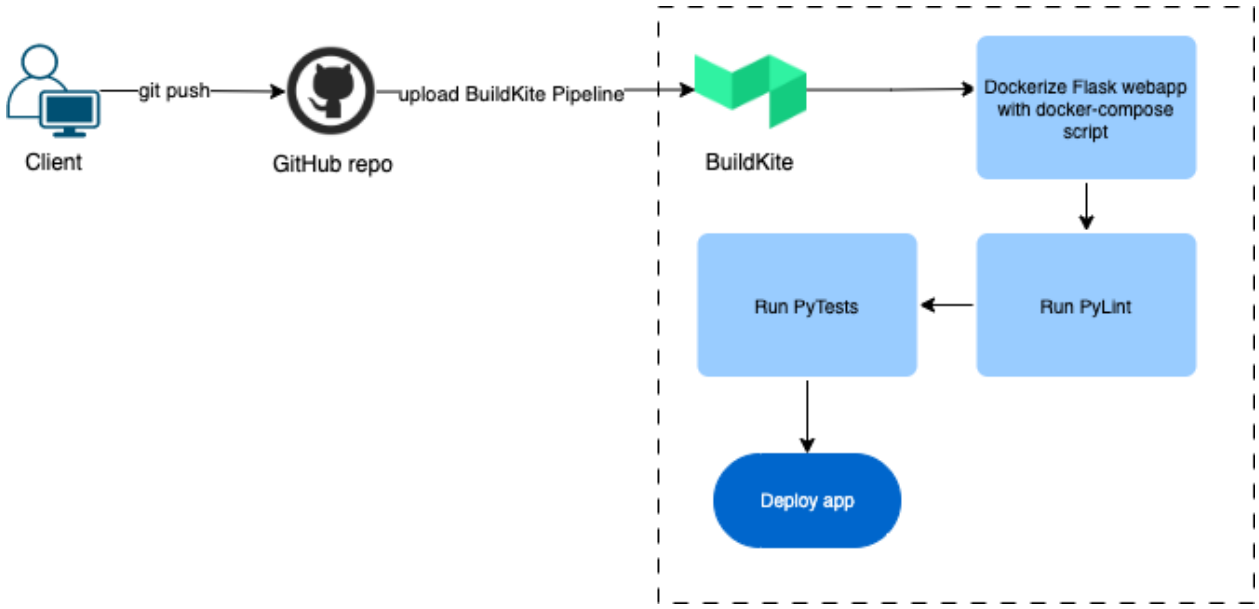
Bijlage 6: API Flowchart

In onderstaande afbeelding is het proces, wat doorlopen wordt door de API wanneer het voorspellingen endpoint wordt aangesproken door MB2, gevisualiseerd.



Bijlage 7: CI/CD-flowchart

Wanneer een nieuwe versie van de applicatie op de main branch wordt gedeployed, zal de app automatisch getest en idealiter automatisch gedeployed worden met BuildKite. BuildKite is gekoppeld met de aangemaakte repository, die de Flask API bevat.



Bijlage 8: Adviesrapport

Dit document geeft verdere toelichting op de aanbevelingen die voortkomen uit het onderzoek, zoals samengevat in hoofdstuk 6 'Aanbevelingen'. Allereerst wordt de opgestelde query besproken waarmee de database kan worden gesynchroniseerd. Vervolgens wordt er ingegaan op de manier waarop de database kan worden gesynchroniseerd met behulp van cronjobs. Ten slotte wordt kort besproken hoe het ML-model in de toekomst beter kan presteren.

Het synchroniseren van de database

Er is een SQL-query opgesteld om de database te synchroniseren via een cronjob in de Flask API. Om dit te kunnen uitvoeren, is echter read-toegang tot de database vereist of een alternatieve oplossing. De eerste codeblok bevat de geschikte SQL-query om alle factuurregels op te halen voor administraties die de Data User Agreement hebben geaccepteerd. Deze SQL-query exporteert de gegevens ook rechtstreeks naar een CSV-bestand dat kan worden gebruikt door de Flask API.

```
\copy (SELECT details.administration_id, details.tax_rate_id, details.ledger_account_id,
details.price, ledger_accounts.name, tax_rates.name, sales_invoices.contact_id FROM details
INNER JOIN ledger_accounts ON details.ledger_account_id = ledger_accounts.id INNER JOIN
tax_rates ON details.tax_rate_id = tax_rates.id INNER JOIN sales_invoices ON
details.invoice_id = sales_invoices.id INNER JOIN administrations ON
details.administration_id = administrations.id AND details.administration_id GROUP BY
details.administration_id, details.tax_rate_id, details.ledger_account_id, details.price,
ledger_accounts.name, tax_rates.name, sales_invoices.contact_id ) to '/home/app/temp.csv'
csv header;
```

Vervolgens kan de query uitgevoerd worden door middel van een cronjob. Deze cronjob kan in de root van de Flask API worden toegevoegd met behulp van onderstaande code.

```
@scheduler.task('interval', seconds=3, misfire_grace_time=None)
def job1():
   print('Here you can add the database query')
```

Hoewel de vereiste componenten voor het synchroniseren van de database aanwezig zijn, is er momenteel geen directe read-toegang beschikbaar tot de database. Om het product als productiewaardig te beschouwen, moet dit eerst worden geregeld.

Het verwerken van de data en het trainen van het ML-model

Het verbeteren van het ML-model kan op vele manieren gedaan worden, zoals het toevoegen van meer data, optimaliseren van hyperparameters, het verwijderen van outliers, en het gebruik van ensemble methoden, kan het ML-model worden verbeterd om nauwkeuriger en betere voorspellingen te doen. Het verwijderen van outliers gebeurt al, maar het optimaliseren van de hyperparameters bijvoorbeeld niet. Het laatstgenoemde voorbeeld is hierin een van de eerste stappen om tot een beter model te komen. Aangezien Moneybird zelf bijvoorbeeld geen invloed heeft op het verzamelen van meer data, sinds de ondernemer zelf verkoopfacturen toevoegt.

Bijlage 9: CI/CD-pipeline.yml

In deze bijlage volgt een codeblok die de inhoud van de pipeline.yml weergeeft. Het bestand specificeert de stappen die BuildKite moet volgen voor het CI/CD-proces, zoals deze gedefinieerd is in het onderzoek. Hetgeen wat ontbreekt in de pipeline.yml is de deploy stap. Het ontbreken hiervan is toegelicht in het Aanbevelingen hoofdstuk.

```
steps:
 - command: |
     # Build the Docker image
     docker-compose build
   label: ":docker: Build Docker image"
 - wait
 - command: |
     # Lint the Python code with PyLint
     pylint ./mb
   label: ":snake: Lint Python code"
   plugins:
     - docker-compose#v4.11.0:
         run: web
         docker-compose-file: docker-compose.yml
 - wait
 - command: |
     # Run tests with PyTest
     pytest
   label: ":checkered_flag: Run tests"
   plugins:
     - docker-compose#v4.11.0:
         run: web
         docker-compose-file: docker-compose.yml
```

Bijlage 10: Stimulus controller

De bijlage bevat een codeblok met daarin de Stimulus-controller. Binnen de controller wordt een API-request uitgevoerd naar de Ruby on Rails-controller. De Stimulus-controller verwerkt de inhoud van de ingevulde invoervelden en stuurt deze mee als parameters in het API-request. De inhoud wordt meegestuurd in JSON-formaat en er wordt gebruikgemaakt van Ajax om het API-request uit te voeren.

```
import { Controller } from "@hotwired/stimulus";
import Rails from "@rails/ujs";

export default class extends Controller {
 static targets = ["tooltip", "label", "priceField", "ledgerAccountField"];

 async makeTaxRatePrediction() {
   const path = window.location.pathname; // Get the current URL path
   const parts = path.split("/"); // Split the path into an array of parts
   const administrationId = parts[1]; // Get the first part of the path, which
should be the administration ID

   const contact = JSON.parse(this.data.element.dataset.contact);

   const price = parseInt(this.priceFieldTarget.value.replace(",", "."), 10);
   const ledger_account_id = this.ledgerAccountFieldTarget.value;

   const params = new URLSearchParams({
     ledger_account_id,
     price,
     contact_id: contact.id,
     contact_name: contact.company_name,
   }).toString();

   Rails.ajax({
     url: `${window.location.origin}/${administrationId}/tax_rate_prediction`,
     type: "GET",
     data: params,
     dataType: "json",
     success: function (content) {
       this.labelTarget.textContent = content;
       this.tooltipTarget.style.display = "flex";
     }.bind(this),
   });
 }
}
```

Bijlage 11: Ruby on Rails controller

In de bijlage is een codeblok opgenomen met daarin de Ruby on Rails-controller. De controller bevat een endpoint dat luistert naar API-requests die binnenkomen op het /administration_id/tax_rate_prediction endpoint. Dit endpoint wordt altijd aangeroepen via de Stimulus-controller. De Ruby on Rails-controller communiceert met de Flask-API en stuurt de ontvangen waardes mee naar de Flask-API. Het antwoord van de Flask-API wordt vervolgens verwerkt tot een bericht en teruggestuurd naar de Stimulus-controller.

```
class TaxRatePredictionsController < ApplicationController
 def tax_rate_prediction
   # Replace hard coded administration ID with #{params[:administration_id]}
   uri = "http://127.0.0.1:5000/predict/demo_administration"

   # Set up the JSON body with the parameters
   parameters = {
     ledger_account_id: params[:ledger_account_id],
     contact_id: params[:contact_id],
     price: params[:price],
   }

   request = {
     method: :post,
     body: parameters.to_json,
     headers: {
       'Content-Type' => 'application/json'
     }
   }

   response = Excon.post(uri, request)

   # Parse the response from JSON to a Ruby hash
   response_body = JSON.parse(response.body)

   ledger_account_name = administration.ledger_accounts
                                       .find(params[:ledger_account_id]).name
   predicted_tax_rate_name = administration.tax_rates
                .find(response_body["predicted_tax_rate_id"].to_i).name

   message = t('tax_rate_prediction_message',
               predicted_tax_rate_name: predicted_tax_rate_name,
               ledger_account_name: ledger_account_name,
               company_name: params[:contact_name])

   render json: message
 end
end
```

Bijlage 12: Ruby on Rails controller spec

Er is een spec/test geschreven om de Ruby on Rails-controller te testen, waarbij de volledige functionaliteit van de controller wordt getest. Hiervoor wordt de API gedupliceerd en wordt getest of de juiste waardes worden teruggestuurd op basis van vaste waardes.

```ruby
# frozen_string_literal: true

require 'rails_helper'

RSpec.describe TaxRatePredictionsController, type: :controller do
 include_context :administration_context

 let(:sales_invoice) { create_open_sales_invoice(invoice_data: Date.new(2023, 3,
30)) }
 let(:detail) { create_detail(sales_invoice, price: 1000) }

 before { allow(controller).to receive(:current_user).and_return(user) }

 describe "GET tax_rate_prediction" do
   it "returns a prediction in JSON-format based on an API-request" do

     api_double = double("ExternalAPI")
     allow(api_double).to receive(:get_tax_rate_prediction).and_return(12345678910)

     # Inject the test double into the controller using dependency injection
     allow(controller).to receive(:external_api).and_return(api_double)

     get :tax_rate_prediction, params:
       {
         administration_id: administration.id,
         contact_id: sales_invoice.contact_id,
         ledger_account_id: detail.ledger_account_id,
         price: detail.price
       }

     expect(response).to have_http_status(:ok)

     expect(response.parsed_body).to eq({ "predicted_tax_rate_id" => 12345678910 })
   end
 end
end
```