HO
GENT

Verwerken en visualiseren van bankgegevens met Python

Quint Gyselinck

202076524

GO5 By HOGENT

14/06/23

Graduaatsproef Programmeren

Verwerken en visualiseren van bankgegevens
met Python

14/06/23

Student
Quint Gyselinck
quint.gyselinck@student.hogent.be
202076524

Lector
Luc Vervoort
luc.vervoort@hogent.be

Woord vooraf

Voor u ligt de graduaatsproef “Verwerken en visualiseren van data in Python”. Het idee om deze applicatie te programmeren is ontstaan tijdens het ontwerpen van ons projectwerk, hierbij moesten we een bank-applicatie ontwerpen en programmeren. Het visualiseren van de transacties leek mij dus een ideaal onderwerp voor mijn graduaatsproef.

Tijdens het maken van mijn graduaatsproef heb ik leren programmeren in Python en het leren gebruiken van enkele Python libraries, daarnaast heb ik hierbij de API van OpenAI gebruikt voor een personal AI bot die je wat meer inzicht geeft over de gegevens. Tot slot heb ik Docker containers leren gebruiken voor het containerizen van de applicatie en het deployen er van.

Ik wil mijn lector Luc Vervoort bedanken voor de begleiding en ondersteuning.

Q. Gyselinck

Gent, 14 juni 2023

Inhoudstafel

1. Samenvatting

Dit afstudeerwerk focust op het belang van data verwerking en visualisatie in de financiële sector. De applicatie zorgt voor een oplossing op het probleem dat gebruikers van banken soms weinig visueel inzicht hebben op hun inkomsten en uitgaven.

Het doel van dit afstudeerproject is het ontwikkelen van een applicatie die banktransactiedata kan verwerken en visualiseren, waardoor gebruikers een beter inzicht krijgen in hun financiële gegevens. Daarnaast wordt de OpenAI API geïntegreerd in de applicatie, wat een interactieve ervaring biedt waarin gebruikers vragen kunnen stellen over hun financiën.

De ontwikkelde applicatie maakt gebruik van verschillende technologieën en methodologieën. Ten eerste wordt data verzameld en voorbereid via een JSON-bestand dat vervolgens wordt geparsed naar een Pandas DataFrame. Voor de visualisatie van de gegevens wordt de python library Plotly gebruikt, die de transactiedata omzet naar interactieve grafieken. De OpenAI API wordt geïntegreerd voor interactie met gebruikers, en de applicatie wordt ingekapseld in een Docker-container voor eenvoudigere implementatie. Tenslotte wordt de applicatie geïmplementeerd met het Flask-webframework om de resultaten en visuals naar de frontend te versturen.

Het werk eindigt met een technische implementatie van de backend, data verwerking, en visualisatie, en de integratie van de OpenAI API. Bovendien wordt de implementatie van de applicatie in een Docker-container besproken.

De resultaten van het project tonen een succesvolle visualisatie van banktransactiegegevens en een functionele integratie van de OpenAI API, wat een interactieve gebruikerservaring mogelijk maakt. De studie eindigt met een discussie over de resultaten en een vergelijking tussen Python en Javascript.

2. Summary

This graduation project focuses on the importance of data processing and visualization in the financial sector. The application provides a solution to the problem that bank users sometimes have little visual insight into their income and expenditure.

The aim of this graduation project is to develop an application that can process and visualize bank transaction data, giving users a better understanding of their financial data. In addition, the OpenAI API is integrated into the application, providing an interactive experience in which users can ask questions about their finances.

The developed application uses various technologies and methodologies. First, data is collected and prepared via a JSON file, which is then parsed into a Pandas DataFrame. For the visualization of the data, the Python library Plotly is used, which converts the transaction data into interactive graphs. The OpenAI API is integrated for interaction with users, and the application is encapsulated in a Docker container for easier implementation. Finally, the application is implemented with the Flask web framework to send the results and visuals to the frontend.

The work concludes with a technical implementation of the backend, data processing, and visualization, and the integration of the OpenAI API. Moreover, the implementation of the application in a Docker container is discussed.

The results of the project show a successful visualization of bank transaction data and a functional integration of the OpenAI API, enabling an interactive user experience. The study concludes with a discussion of the results and a comparison between Python and JavaScript.

3. Inleiding

3.1. Het belang van gegevensverwerking en visualisatie in de financiële sector

In de moderne wereld van de financiële sector genereren banken enorme hoeveelheden data, variërend van transactiegegevens tot klantinformatie. Het verwerken en visualiseren van deze transactiegegevens is van belang voor klanten om zo een beeld te schetsen over hun uitgaven en inkomsten zonder hier zelf berekeningen voor te moeten doen.

3.2. Doelstellingen van het afstudeerwerkstuk

Dit afstudeerwerkstuk richt zich op het ontwikkelen van een applicatie die data verwerkt en visualiseert. Het doel is om een applicatie te creëren dat transactiegegevens van een bank verzamelt en omzet naar grafieken, waardoor de gebruiker beter inzicht krijgt op zijn financiële gegevens. Daarnaast moet de applicatie de gebruikers in staat te stellen vragen te stellen over transacties gegevens en deze vragen te beantwoorden met behulp van de OpenAI API, waardoor een interactieve ervaring wordt geboden.

4. Probleemstelling

4.1. Uitdagingen bij het verwerken van bankgegevens en inzichten verkrijgen

Het verwerken van gegevens kan altijd uitdagingen met zich meebrengen. Vaak zijn het grote datasets met complexe structuren en variabelen. Dit maakt het moeilijker om de bruikbare data te identificeren en extraheren. Daarnaast kunnen ontbrekende of onjuiste gegevens de betrouwbaarheid beïnvloeden.

4.2. Beperkingen van traditionele methoden en handmatige analyse

Als klant is het vaak moeilijk om eigen uitgaven en inkomsten te verwerken en analyseren. Vaak worden er rekenfouten gemaakt en is het tijdrovend. Bovendien geven spreadsheets of dergelijke niet altijd een dynamisch overzicht van de uitgaven en de categorieën waarin ze zijn onderverdeeld. Dit beperkt de mogelijkheden voor gebruikers om snel een inzicht te krijgen in hun uitgavenpatronen.

4.3. Noodzaak van geautomatiseerde gegevensverwerking en visualisatie

Door bovenstaande reden is het dus essentieel als bank om aan automatische gegevensverwerking en visualisatie te doen bij het werken met transactiegegevens. Door gebruik te maken van deze methodes stelt het klanten in staat om real-time inzicht te krijgen in hun uitgaven en om dus te zien wat ze bijvoorbeeld elke maand uitgeven en aan welke categorie, zoals voeding, transport, entertainment, ...

5. Methodologie en aanpak

5.1. Gegevensverzameling en -voorbereiding

Voor het verzamelen van de gegevens hebben we een JSON bestand gemaakt met daar in de transactiegegevens, elke transactie bevat volgende gegevens: beschrijving, id, hoeveelheid, type (krediet of debiet), datum, en de categorie waar de transactie toe behoort. We hebben voor dit voorbeeld een lijst gegenereerd met daarin 50 random transacties. In een real-life situatie zouden we deze gegevens kunnen ophalen uit de database van de bank.

De gegevens worden aan het begin van de applicatie geparsed naar een DataFrame met behulp van de Pandas package. Dit zorgt er voor dat we van een JSON format naar een string format gaan. Hierbij een klein voorbeeldje van wat de input en output is van zo een DataFrame.

```
json = {
      "title": "Test",
      "transaction_id": "5212",
      "amount": 25.55,
      "type": "credit",
      "date": "10/05/23",
      "category": "Food and Dining"
}

df = Panda.DataFrame(json)

df =  title   transaction_id    amount    type    date      category
      Test    5212              25.55     credit  10/05/23  Food
```

5.2. Visualisatie van gegevens met Plotly

Voor de visualisatie van de gegevens hebben we gebruik gemaakt van een python library genaamd Plotly. Plotly is zoals ze het zelf noemen een “Open Source Graphing Library”, deze library maakt het dus gemakkelijk om data van bijvoorbeeld een dataframe om te zetten in mooie en interactieve grafieken. Wij maken momenteel gebruik van de Bar en Pie chart die wordt aangeboden door Plotly.

5.3. Integratie van de OpenAI API

Om de applicatie wat interactiever te maken voor de gebruiker maken we gebruik van de momenteel zeer populaire API van OpenAI, hiermee maken we het mogelijk voor de gebruiker om een vraag te stellen in de app over hun financieën aan een AI bot of “Personal AI Assitant” die op zijn beurt een antwoord terug geeft met betrekking tot alle verzamelde gegevens van de klant. Voor dit te bekomen is er een account gecreëerd op het OpenAI developer platform en is er een API key aangevraagd om toegang te krijgen tot de API via onze applicatie. We hebben gekozen om gebruik te maken van het “text-davinci-003” model, dit model voldoet zeker aan de eisen van onze applicatie.

5.4. Implementatie in een Docker-container

We hebben er voor gekozen om de applicatie te encapsuleren in een Docker container. Door de applicatie in een container te steken zorgen we ervoor dat de applicatie verpakt zit met al zijn dependencies, libraries, etc. Hierdoor kan de applicatie op bijna elk besturingssysteem runnen. Voor ons maakt het het deployment process een stuk gemakkelijk, we kunnen de Docker image op een cloud platform zetten, hierbij kozen wij voor Heroku, om zo online onze webservice te hosten.

5.5. Implementatie met Flask-webframework

Om de bekomen resultaten en visuals naar de front-end te versturen hebben we een kleine REST API opgezet. Dit deden we met het Flask-webframework. De reden waarom we hiervoor gekozen hebben is omdat Flask enorm lightweight en flexibel is. Het doet alles wat het moet doen voor mijn applicatie met maar enkele lijnen code die zeer gemakelijk te begrijpen zijn.

6. Technische implementatie

6.1. Backend-implementatie

Een belangrijk onderdeel van de backend is de Flask-applicatie. Om communicatie met de frontend mogelijk te maken gebruiken we verschillende routes en endpoints. We beginnen met flask te importeren bovenaan in onze code.

```
from flask import Flask, Response, request
```

Daarna maken we een instantie van onze Flask-app aan

```
app = Flask(__name__)
```

Met deze “app” kunnen we nu routes gaan beginnen definieren. We geven de endpoint mee en welke HTTP method we willen gebruiken. Daarin koppelen we een functie die wordt uitgevoerd bij het aanroepen van de endpoint. In dit voorbeeld willen we dus een taartdiagram terug krijgen. Het is dus een “GET” request met als route /get_pie met daarin een function die een taartdiagram maakt en retourneerd.

```
@app.route('/get_pie', methods=['GET'])
def get_pie_chart():
    #Inhoud functie
    return #Retourneren van gewenste resultaat
```

De OpenAI endpoint ziet er anders uit omdat we hier gebruik maken van een “POST” request. Dit doen we omdat deze request data verstuurd bij het aanroepen. Hierbij wordt de vraag van de klant meegegeven in de body van de request en wordt een antwoord als response teruggegeven, beide in een JSON format.

```
@app.route('/answer_question', methods=['POST'])
def answer_question():
    #Inhoud function
    return {'answer': answer}
```

Op het einde van de applicatie vergelijken we of dit programma wel het hoofdprogramma is en runnen we de applicatie als dit zo is.

```
if __name__ == '__main__':
    app.run(port=int("3000"), debug=True)
```

6.2. Data-verwerking en -visualisatie

Voor het verweken en visualiseren van de gegevens met Plotly zijn er enkele stappen. We schrijven een functie voor het parsen van de JSON data naar een Pandas DataFrame. Dit maakt het gemakkelijk om de gegevens te verwerken en sorteren.

```
def load_data():
    with open('transactions.json') as f:
        data = json.load(f)
    df = pd.DataFrame(data)
    return df
```

Bij het genereren van een pie chart grouperen we eerst de data die nodig is om onze grafiek te maken en tellen alle uitgaven op per categorie. Dit geven we dan mee in de Plotly functie “px.pie” samen met de ‘values’, ‘name’ en ‘title’.

```
df = load_data()
category_totals = df.groupby('category')['amount'].sum().reset_index()
fig = px.pie(category_totals, values='amount', names='category',
title='Spending by Category')
```

De gegenereerde grafiek zetten we hierna om in een svg format en retourneren we als response. Dit doen we met de ingebouwde functie van plotly ‘pio.to_image’.

```
svg_bytes = pio.to_image(fig, format="svg")
svg_string = svg_bytes.decode()
return Response(svg_string, mimetype='image/svg+xml')
```

6.3. Integratie van de OpenAI API

Om gebruik te kunnen maken van de OpenAI API, moet je een account aanmaken op het OpenAI developer platform en een API-toegangssleutel verkrijgen. Deze sleutel geeft je de benodigde toegang tot de AI-mogelijkheden van OpenAI. Deze sleutel geven we later mee in de code voor een response te genereren.

Hierna installeren we de OpenAI library. Hierdoor kunnen we alle functionaliteit van de OpenAI API gebruiken in onze code. Dit kan gemakkelijk gedaan worden via de Command Line Interface van ons systeem door de het volgende commando uit te voeren: “pip install openai”.

Om het antwoord te genereren en terug te sturen naar de frontend schrijven we deze keer een POST methode naar de route “/answer_question van onze API. We krijgen via de body van ons request een vraag mee van de gebruiker en laden onze transactiedata in. We zetten ons dataframe ook gelijk om in een dictionary.

```
@app.route('/answer_question', methods=['POST'])
def answer_question():
    question = request.get_json().get('question')
    df = load_data()
    data = df.to_dict('records')
```

Hierna generen we een response met de openai.completion methode uit de OpenAI library. Deze bevat enkele parameters die bepalen wat voor antwoord we verwachten.

Engine: Geeft aan welk ai model we gaan gebruiken
Promt: Hiermee geven we de vraag mee die we aan het model stellen met daarin de data geïnjecteerd.
Dit zijn de 2 belangrijkste, de rest van de parameters gaan bepalen hoe creatief het antwoord van de ai gaan zijn.

Het antwoord van de API gaat extra data bevatten die we niet nodig hebben dus extraheren we enkel de data we wel nodig hebben en geven we die data terug mee in de response van de HTTP request. Zie volgende bladzijde voor de code.

```
    response = openai.Completion.create(
        engine='text-davinci-003',
        prompt=f"De data toont dit aan {data}. De vraag is:
{question}\n Antwoord in het nederlands
        max_tokens=300,
       temperature=0.7,
    )
    answer = response.choices[0].text.strip()
    return {'answer': answer}
```

6.4. Docker-implementatie

Zoals eerder uitgelegd gaan we gebruiken maken van Docker om onze applicatie te gaan containerizen zodat we deze gemakkelijk kunnen deployen op een hosting platform zoals Heroku. We starten door Docker op ons systeem te gaan installeren, dit kan via de website van Docker. Hierna maken we een Dockerfile aan, hierin beschrijven we enkele zaken over de applicatie. We kiezen de versie van python, wij kozen voor Python 3.11. Daarnaast kiezen we de directory waarin de app gaat draaien en geven we een “requirements.txt” file mee. In deze file staan alle libraries en pakketten waarvan onze app gebruik maakt zodat deze kunnen geinstalleerd worden bij het maken van de docker image. De poort die opengezet wordt voor de connectie naar onze app staat hier in en welk commando er wordt uitgevoerd bij het opstarten van de container. Dit is bij python het commando “python ./’pythonfile’”. Onderstaand is de dockerfile voor dit project.

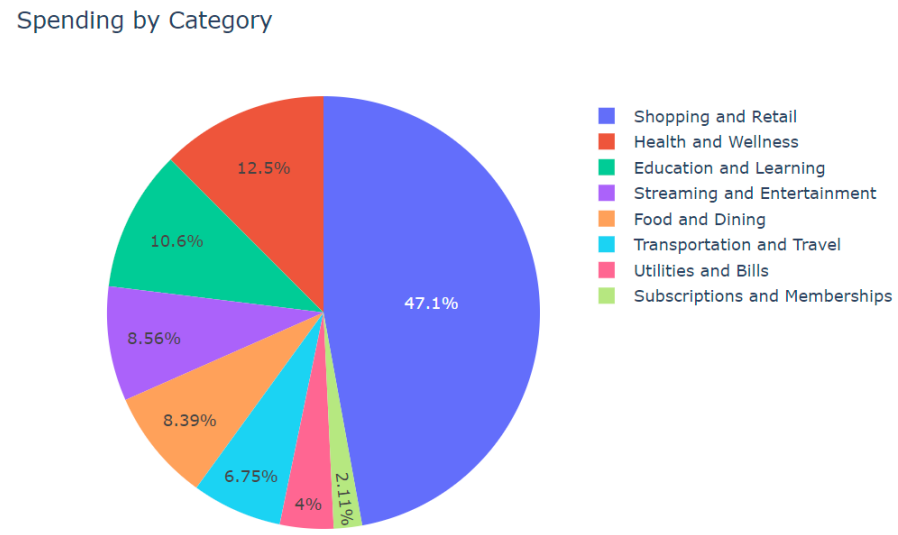
```
FROM python:3-alpine3.11
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 3000
CMD python ./api.py
```

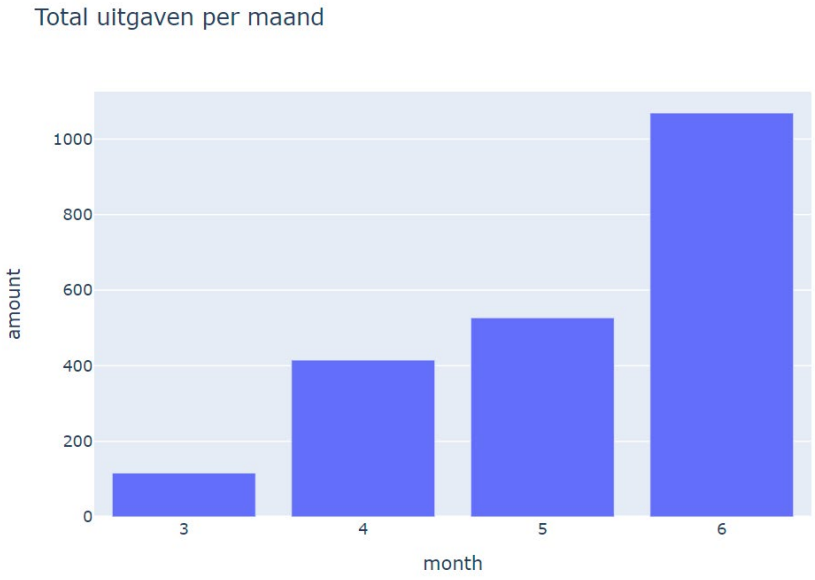
Als de requirements.txt file en de Dockerfile in orde zijn kunnen we beginnen aan het builden van onze image. Dit doen we door het commando “docker build -t ‘naam van container’“ uit te voeren.

Met deze image kan je dan je applicatie op de meeste cloud platformen deployen.

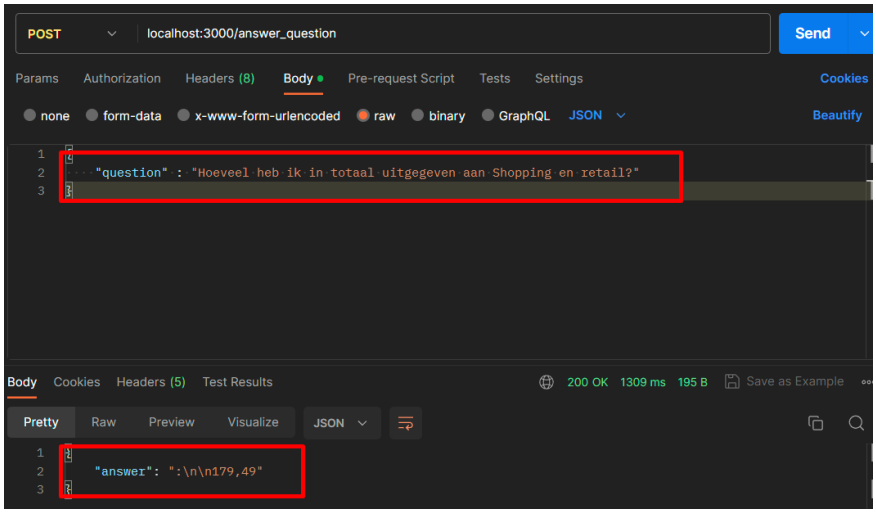
7. Resultaten en Discussie

7.1. Resultaat visualisatie





7.2. Resultaat OpenAI API



7.3. Vergelijking Python vs Javascript

Bij het programmeren van hetzelfde programma in javascript kwamen we tot de constateringen. De code lijkt heel erg op elkaar en er wordt gebruik gemaakt van enkele zelfde libraries zoals Plotly en OpenAI. Bij python wordt Flask gebruikt voor de REST API en bij javascript wordt dan weer het welgekende Express.js gebruikt. Iets heel opvallend was dat we voor het javascript programma bijna het dubbele in lijnen code nodig hadden voor dezelfde functionaliteit. We kunnen concluderen dat voor dit soort services python eigenlijk ideaal is, het verwerken van veel data en simpele, flexible api servers opzetten is hierbij zeer gemakkelijk.

7.4. Vergelijking met doelstellingen

Bij aanvang van het project werden volgende doelstellingen geformuleerd:

De applicatie moest in staat zijn om gegevens te verwerken en te visualiseren in grafieken.

> Resultaat: De applicatie is succesvol in het verwerken van de transactiegegevens, ze werden omgezet in de nodige formats en worden gegroepeerd en berekend. De Plotly-library heeft het mogelijk gemaakt om de gewenste grafieken en inzichten te genereren. Hiermee wordt voldaan aan de doelstelling van het verwerken en visualiseren van gegevens.

De applicatie moest de mogelijkheid bieden voor de gebruiker om vragen te stellen over zijn/haar financiële gegevens en antwoorden te ontvangen van een AI-assistent.

> Resultaat: Door de integratie van de OpenAI API is het mogelijk geworden voor de gebruiker om vragen te stellen aan de AI-assistent. De AI-assistent geeft antwoorden op basis van de beschikbare gegevens en biedt aanvullende inzichten en informatie. Hiermee wordt voldaan aan de doelstelling van het integreren van de OpenAI API.

Door het succesvol bereiken van de gestelde doelen en de waardevolle bijdrage aan het verwerken en visualiseren van bankgegevens, kan geconcludeerd worden dat de applicatie een waardevol instrument is voor individuele gebruikers om hun financiële gezondheid te monitoren en te verbeteren.

8. Bronnenlijst

3.11.4 Documentation. (z.d.). https://docs.python.org/3/

DataFrame — pandas 2.0.2 documentation. (z.d.). https://pandas.pydata.org/docs/reference/frame.html

Docker Docs: How to build, share, and run applications. (2023, 9 juni). Docker Documentation. https://docs.docker.com/

Heroku Dev Center. (z.d.). https://devcenter.heroku.com/

OpenAI API. (z.d.). https://platform.openai.com/docs/introduction

Python Packaging User Guide — Python Packaging User Guide. (z.d.). https://packaging.python.org/en/latest/

Python Tutorial. (z.d.). https://www.w3schools.com/python/

Welcome to Flask — Flask Documentation (2.3.x). (z.d.). https://flask.palletsprojects.com/en/2.3.x/

9. Bijlagen

Github project: https://github.com/QuintGyselinck/Graduaatsproef