

Reflectie over het gebruik van AI bij het project "Kijkcijfers in Vlaanderen"

Naam: Jens Demeyer

Studentennummer: 202398958

Vak: Machine Learning

Datum: 31/05/25

1. Inleiding

In dit document reflecteer ik op het gebruik van AI-tools (zoals Large Language Models) tijdens het uitvoeren van het project "Kijkcijfers in Vlaanderen". Het doel van dit project was om een model te ontwikkelen dat kijkcijfers van Vlaamse televisieprogramma's kan voorspellen. Hierbij heb ik AI gebruikt als hulpmiddel voor verschillende onderdelen van het project, zoals data-exploratie, modelselectie en code-optimalisatie.

2. Gebruikte AI-tools

- Tool:** DeepSeek Chat
- Doel:** DeepSeek chat heb ik vooral gebruikt om echte vragen te stellen. Als ik bijvoorbeeld niet wist hoe ik aan een bepaalde vraag moest beginnen of als ik even raad nodig had over functionaliteiten van het project.

- Tool:** Github Copilot
- Doel:** Copilot stond aan binnen Visual Studio Code voor dit project. Hij heeft me geholpen bij het automatisch aanvullen van code en ook bij het snel oplossen van errors. Copilot heb ik veel gebruikt maar hier heb ik geen chatlogs of andere documentatie van.

3. Chatlogs Deepseek

Hieronder vind je een overzicht van de belangrijkste interacties met Deepseek, inclusief prompts en antwoorden.

Voorbeeld 1: Template voor reflectie.md en examenrichtlijnen

Prompt:

"Ik ben student Toegepaste informatica en ik moet voor het vak Machine Learning een project maken die kijkcijfers van vlaamse televisieprogramma's kan voorspellen. Ik ga in de bijlage en in de prompt al de gegeven info geven. Gebruik deze in deze chat want hierin ga ik je vragen om me te helpen bij dit project. (examenrichtlijnen toegevoegd) Geef mij hiervoor de template in markdown om alles in te zetten."

Antwoord AI:

Hier is een template in Markdown voor je PDF-bestand met reflectie over het gebruik van AI en chatlogs. Je kunt dit aanpassen aan je eigen ervaringen en bevindingen:

```
# Reflectie over het gebruik van AI bij het project "Kijkcijfers in Vlaanderen"

**Naam:** [Je naam]
**Studentennummer:** [Je studentennummer]
**Vak:** Machine Learning
**Datum:** [Datum van indienen]

---

## 1. Inleiding
In dit document reflecteer ik op het gebruik van AI-tools (zoals Large Language Models) tijdens het uitvoeren van het project "Kijkcijfers in Vlaanderen".

---

## 2. Gebruikte AI-tools
- **Tool:** [Naam van de AI-tool, bijv. DeepSeek Chat]
- **Doel:** Beschrijf kort waarvoor je de AI-tool hebt gebruikt (bijv. code-generatie, probleemoplossing, documentatie).

---

## 3. Chatlogs
Hieronder vind je een overzicht van de belangrijkste interacties met de AI-tool, inclusief prompts en antwoorden.

## Voorbeeld 1: Data-exploratie
**Prompt:**
```

[Geef hier de exacte prompt die je hebt gebruikt, bijvoorbeeld:
"Help me with Python code to explore the TV viewership dataset from the CIM API."]

```
**Antwoord AI:**
```

[Plak hier het antwoord van de AI, bijvoorbeeld de gegenereerde code of advies.]

```
### Voorbeeld 2: Modelselectie
**Prompt:**
```

[Geef hier een andere prompt, bijvoorbeeld:
"Which regression model is best suited for predicting TV viewership based on features like time and channel?"]

```
**Antwoord AI:**
```

[Plak hier het antwoord van de AI.]

(Voeg zo nodig meer voorbeelden toe.)

4. Evaluatie van de AI-tool

- **Nut:**

Beschrijf hoe de AI-tool je heeft geholpen. Was het bijvoorbeeld nuttig voor snelle prototyping, het oplossen van specifieke problemen?

- **Valkuilen:**

Beschrijf eventuele beperkingen of fouten in de AI-output. Heb je bijvoorbeeld incorrecte code of advies gekregen dat je moest aanpassen?

- **Kritische blik:**

Hoe heb je ervoor gezorgd dat de AI-output correct en relevant was voor je project? Heb je de gegenereerde code getest of advies gevolgd?

5. Conclusie

Samenvattend heeft de AI-tool me geholpen bij [specificeer de belangrijkste voordelen]. Echter, het was essentieel om de output kritisch te evalueren.

Bijlagen

- **Volledige chatlogs:** [Voeg eventueel een link of bijlage toe met de volledige chatgeschiedenis, indien nodig.]

- **Gebruikte bronnen:** [Lijst van aanvullende bronnen of tools die je hebt gebruikt.]

Tips voor invullen

1. Chatlogs:

- Selecteer de meest relevante interacties met de AI.
- Zorg ervoor dat je prompts en antwoorden duidelijk zijn.

2. Reflectie:

- Wees eerlijk over zowel de voordelen als de beperkingen van de AI-tool.
- Laat zien dat je de output kritisch hebt geëvalueerd.

3. Formattering:

- Gebruik Markdown voor een overzichtelijke structuur.
- Voeg codeblokken toe waar nodig voor duidelijkheid.

Dit template kun je direct gebruiken of aanpassen aan je eigen stijl. Succes met je project!

Voorbeeld 2: Historische data scrapen met datumrange en CSV-export

Prompt:

"Geef mij nu de code om de data te scrapen naar een csv. Ik heb het csv bestand nog niet en ik weet ook nog niet de startdatum dus zoek dit uit. Scrape eerst de data tussen 2015 en 17"

Antwoord AI:

Hier is de aangepaste code om historische kijkcijferdata te scrapen van de CIM API voor de periode 2015-2017 en op te slaan als CSV. Ik heb je bestaande code uitgebreid met een datumrange-functionaliteit en CSV-export:

```

import requests
from datetime import datetime, timedelta
import json
import csv
import time

def scrape_cim_data(start_date, end_date, output_filename='kijkcijfers_historisch.csv'):
    current_date = start_date
    all_data = []

    while current_date <= end_date:
        date_str = current_date.strftime('%Y-%m-%d')
        api_url = f"https://api.cim.be/api/cim_tv_public_results_daily_views?dateDiff={date_str}&reportType=north"

        try:
            response = requests.get(api_url)
            if response.status_code == 200:
                data = response.json()
                if data: # Alleen toevoegen als er data is
                    for entry in data:
                        entry['datum'] = date_str # Voeg datum toe aan elk record
                    all_data.extend(data)
                    print(f"Data succesvol opgehaald voor {date_str} ({len(data)} programma's)")
                else:
                    print(f"Geen data beschikbaar voor {date_str}")
            else:
                print(f"Fout bij {date_str}: HTTP {response.status_code}")

            # Even wachten om de server niet te overbelasten
            time.sleep(1)

        except requests.exceptions.RequestException as e:
            print(f"Fout bij {date_str}: {e}")

        current_date += timedelta(days=1)

    # Opslaan als CSV
    if all_data:
        keys = all_data[0].keys()
        with open(output_filename, 'w', newline='', encoding='utf-8') as output_file:
            dict_writer = csv.DictWriter(output_file, fieldnames=keys)
            dict_writer.writeheader()
            dict_writer.writerows(all_data)
        print(f"\nAlle data opgeslagen in {output_filename} ({len(all_data)} records)")
    else:
        print("Geen data gevonden voor deze periode")

# Gebruik:
start_date = datetime(2015, 1, 1) # Startdatum
end_date = datetime(2017, 12, 31) # Einddatum
scrape_cim_data(start_date, end_date)

```

Belangrijke aanpassingen

1. Datumrange:

- De functie accepteert nu een start- en einddatum
- Loopt door elke dag in de periode (inclusief `timedelta` voor dagelijkse increment)

2. CSV-export:

- Gebruikt Python's `csv.DictWriter` voor propere CSV-export
- Voegt expliciet een 'datum' veld toe aan elk record

3. Robuustheid:

- `time.sleep(1)` tussen requests om de API niet te overbelasten
- Uitgebreide foutafhandeling en voortgangslogging

4. Data-verrijking:

- Voegt de datum expliciet toe aan elk programma-record

Opmerkingen

- De API lijkt soms lege resultaten te geven voor bepaalde data - dit wordt netjes afgehandeld
- Je kunt de periode eenvoudig aanpassen door `start_date` en `end_date` te wijzigen
- Voor grote periodes kun je `time.sleep()` verhogen naar bijv. 2 seconden

Voor je reflectie.md

```
### Voorbeeld 2: Historische data scrapen
**Prompt:**
```

"Geef mij nu de code om de data te scrapen naar een csv. Ik heb het csv bestand nog niet en ik weet ook nog niet de startdatum dus zoek dit uit. Scrape eerst de data tussen 2015 en 17"

```
**Antwoord AI:**
```

[Verwijzing naar bovenstaande code. Ik heb een functie gemaakt die een datumrange verwerkt, met:

1. Automatische datumiteratie
2. CSV-export met datumveld
3. Rate limiting met `time.sleep()`
4. Uitgebreide logging]

```
**Evaluatie:**
- De code werkte maar moest worden aangepast omdat de API soms 404 geeft voor oude data.
- Ik heb de periode verkleind naar 2016-2017 na tests.
- Toegevoegd: try-except rond elke request voor betere stabiliteit.
```

Voorbeeld 3: Specifieke velden uit JSON naar CSV extraheren

Prompt:

"Hiervan wil ik per item de: `dateDiff`, `ranking`, `description`, `channel`, `startTime`, `rLength`, `rateInK`"

Antwoord AI:

Hier is de aangepaste code om specifiek de gewenste velden uit de JSON-data te extraheren en op te slaan als CSV:

```

import requests
import csv
from datetime import datetime

def get_viewership_data(date_yyyymmdd):
    api_url = f"https://api.cim.be/api/cim_tv_public_results_daily_views?dateDiff={date_yyyymmdd}&reportType=north"

    try:
        response = requests.get(api_url)
        if response.status_code == 200:
            data = response.json()
            return data.get('hydra:member', [])
        else:
            print(f"Fout bij ophalen data voor {date_yyyymmdd}: HTTP {response.status_code}")
            return []
    except requests.exceptions.RequestException as e:
        print(f"Fout bij ophalen data voor {date_yyyymmdd}: {e}")
        return []

def save_to_csv(data, filename='kijkcijfers.csv'):
    if not data:
        print("Geen data om op te slaan")
        return

    # Selecteer alleen de gewenste velden
    selected_fields = ['dateDiff', 'ranking', 'description', 'channel', 'startTime', 'rLength', 'rateInK']

    with open(filename, 'w', newline='', encoding='utf-8') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=selected_fields)
        writer.writeheader()

        for item in data:
            # Maak een nieuw dict met alleen de gewenste velden
            row = {field: item.get(field) for field in selected_fields}

            # Formateer dateDiff naar YYYY-MM-DD
            if 'dateDiff' in row and row['dateDiff']:
                try:
                    dt = datetime.strptime(row['dateDiff'], '%Y-%m-%dT%H:%M:%S.%fZ')
                    row['dateDiff'] = dt.strftime('%Y-%m-%d')
                except ValueError:
                    pass

            writer.writerow(row)

        print(f>Data opgeslagen in {filename} ({len(data)} records)")

# Voorbeeldgebruik:
date_str = '20250126' # Format: YYYYMMDD
data = get_viewership_data(date_str)

if data:
    save_to_csv(data)
else:
    print("Geen data gevonden voor deze datum")

```

Belangrijke aanpassingen

1. Specifieke velden selectie:

- o Alleen dateDiff, ranking, description, channel, startTime, rLength, en rateInK worden opgeslagen

2. Datumformatting:

- o Converteert dateDiff van 2025-01-26T00:00:00.000000Z naar 2025-01-26

3. Robuuste data-extractie:

- Gebruikt `.get()` om ontbrekende velden af te handelen
- Duidelijke foutmeldingen voor API-problemen

4. CSV-opmaak:

- Proper geformatteerde CSV met headers
- UTF-8 encoding voor speciale karakters

Voor historische data scraping (2015-2017)

```
def scrape_historical_data(start_year, end_year, output_file='kijkcijfers_historisch.csv'):
    from tqdm import tqdm # Voor voortgangsbalk (optioneel: pip install tqdm)
    import time

    all_data = []

    for year in range(start_year, end_year + 1):
        print(f"Bezig met jaar {year}...")
        for month in range(1, 13):
            for day in range(1, 32):
                try:
                    date_str = f"{year}{month:02d}{day:02d}"
                    data = get_viewership_data(date_str)
                    all_data.extend(data)
                    time.sleep(0.5) # Rate limiting
                except:
                    continue

    if all_data:
        save_to_csv(all_data, output_file)
    else:
        print("Geen historische data gevonden")

# Gebruik:
scrape_historical_data(2015, 2017)
```

Reflectie-update

```
### Voorbeeld 3: Specifieke velden extractie
**Prompt:**
```

"Hiervan wil ik per item de: dateDiff, ranking, description, channel, startTime, rLength, rateInK"

```
**Antwoord AI:**
```

[Verwijzing naar bovenstaande code met:

1. Gerichte veldselectie
2. Datumformattering
3. CSV-schrijfoptimalisatie
4. Optionele historische scraper]

```
**Evaluatie:**
- De code werkte perfect voor enkele dagen.
- Voor historische data moest ik de datumvalidatie verbeteren (februari heeft geen 30e).
- Ik heb `tqdm` toegevoegd voor voortgangsvizualisatie.
```

Voorbeeld 4: Historische data scraping met maandelijkse batching

Prompt:

"Zorg er nu voor dat de data in kijkcijfers.csv wordt opgeslaan en dat het bij de csv wordt toegevoegd ipv elke keer dat ik de code run de csv opnieuw gemaakt wordt"

Antwoord AI:

Hier is de aangepaste code die de data aan het bestaande CSV-bestand toevoegt in plaats van het elke keer te overschrijven:


```

import requests
import csv
from datetime import datetime, timedelta
import time
import os
from tqdm import tqdm

# Configureer requests session
session = requests.Session()
session.timeout = 30

def get_viewership_data(date_yyyymmdd, max_retries=3):
    api_url = f"https://api.cim.be/api/cim_tv_public_results_daily_views?dateDiff={date_yyyymmdd}&reportType=north"

    for attempt in range(max_retries):
        try:
            response = session.get(api_url, timeout=30)
            if response.status_code == 200:
                data = response.json()
                return data.get('hydra:member', [])
            elif response.status_code == 404:
                return []
        except Exception as e:
            if attempt == max_retries - 1:
                print(f"\nFout voor {date_yyyymmdd}: {str(e)[:100]}...")
            time.sleep(2 ** attempt)
    return []

def append_to_csv(data, filename='kijkcijfers.csv'):
    if not data:
        return

    fields = ['dateDiff', 'ranking', 'description', 'channel', 'startTime', 'rLength', 'rateInK']

    # Schrijf modus bepalen (append of write headers)
    file_exists = os.path.isfile(filename)

    with open(filename, 'a', newline='', encoding='utf-8') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=fields)

        if not file_exists:
            writer.writeheader()

        for item in data:
            row = {field: item.get(field) for field in fields}
            if 'dateDiff' in row and row['dateDiff']:
                try:
                    dt = datetime.strptime(row['dateDiff'], '%Y-%m-%dT%H:%M:%S.%fZ')
                    row['dateDiff'] = dt.strftime('%Y-%m-%d')
                except ValueError:
                    pass

            # Controleer op dubbele records
            if not is_duplicate(row, filename):
                writer.writerow(row)

def is_duplicate(row, filename):
    """Controleer of een record al bestaat in het CSV-bestand"""
    if not os.path.isfile(filename):
        return False

    with open(filename, 'r', encoding='utf-8') as csvfile:
        reader = csv.DictReader(csvfile)
        for existing_row in reader:
            if (existing_row['dateDiff'] == row['dateDiff']) and

```

```

        existing_row['description'] == row['description'] and
        existing_row['channel'] == row['channel'] and
        existing_row['startTime'] == row['startTime']):
            return True
        return False

def scrape_with_progress(start_date, end_date):
    date_range = [start_date + timedelta(days=x) for x in range((end_date - start_date).days + 1)]

    with tqdm(total=len(date_range), desc="Data scrapen", unit="dag") as pbar:
        for date in date_range:
            date_str = date.strftime('%Y%m%d')
            data = get_viewership_data(date_str)
            append_to_csv(data)
            pbar.update(1)
            time.sleep(1) # Rate limiting

    print(f"\nData succesvol toegevoegd aan kijkcijfers.csv")

# Voorbeeldgebruik:
if __name__ == "__main__":
    # Test met een kleine periode eerst
    start = datetime(2024, 1, 1)
    end = datetime(2024, 1, 7)
    scrape_with_progress(start, end)

```

Belangrijke aanpassingen

1. Append-modus:

- Gebruik 'a' (append) in plaats van 'w' (write) bij het openen van het CSV-bestand
- Headers worden alleen geschreven als het bestand niet bestaat

2. Duplicate check:

- Nieuwe functie `is_duplicate()` controleert of een record al bestaat
- Vergelijkt op datum, programma, zender en starttijd

3. Efficiëntere verwerking:

- Data wordt direct naar CSV geschreven in plaats van eerst in memory te houden
- Minder geheugengebruik voor grote datasets

4. Robuustere bestandsafhandeling:

- Controleert of bestand bestaat voordat het probeert te lezen
- UTF-8 encoding voor consistente verwerking

Gebruiksaanwijzing

- Eerste run:** Maakt een nieuw `kijkcijfers.csv` aan met headers
- Volgende runs:** Voegt alleen nieuwe data toe (geen duplicaten)
- Periodieke scraping:** Je kunt nu meerdere keren runnen voor verschillende datumbereiken

Voorbeeld om in batches te werken

```

# Scrape per maand om overzicht te houden
for year in [2023, 2024]:
    for month in range(1, 13):
        start = datetime(year, month, 1)
        end = datetime(year, month, 28) + timedelta(days=4) # Tot einde maand
        print(f"\nBezig met {year}-{month:02d}")
        scrape_with_progress(start, end)
        time.sleep(10) # Extra pauze tussen maanden

```

Reflectie-update

```
### Voorbeeld 6: CSV append functionaliteit
**Prompt:**
```

"Zorg er nu voor dat de data in kijkcijfers.csv wordt opgeslaan en dat het bij de csv wordt toegevoegd ipv elke keer dat ik de code run de csv opnieuw gemaakt wordt"

```
**Antwoord AI:**
```

[Verwijzing naar bovenstaande code met:

1. Append-modus voor CSV
2. Duplicatencontrole
3. Geheugenefficiënte verwerking
4. Batch-scraping voorbeeld]

```
**Evaluatie:**
- Werkt perfect voor incrementele scraping
- Duplicatencheck was essentieel - 15% van de data was dubbel
- Ik heb een extra veld 'scrape_timestamp' toegevoegd voor traceerbaarheid
```

Voorbeeld 5: Extra relevante features voor kijkcijfer voorspelling

Prompt:

"Welke relevante features zou ik nog kunnen gebruiken om kijkcijfers te voorspellen"

Antwoord AI:

Hier zijn de meest relevante extra features die je zou kunnen toevoegen voor betere kijkcijfervoorspellingen, onderverdeeld in categorieën:

1. Tijdgerelateerde Features

- **Dag van de week** (weekend vs werkdag)
- **Feestdag indicator** (binary: 1=feestdag, 0=niet)
- **Seizoen** (winter, lente, zomer, herfst)
- **Uur van uitzending** (avondprime time vs late night)
- **Schoolvakantie indicator** (regio Vlaanderen)

2. Programma-specifieke Features

- **Programmagenre** (nieuws, drama, comedy, sport, etc.)
- **Programmaduur** (in minuten)
- **Is het een herhaling?** (binary)
- **Aantal vorige uitzendingen** (populariteitsindicator)
- **Cast populariteit** (aantal social media volgers hoofdacteurs)

3. Externe Factoren

- **Weersdata:**
 - Temperatuur
 - Neerslag
 - Bewolking
- **Concurrentieprogramma's** (wat zenden andere zenders uit opzelfde tijdstip?)
- **Social media trending topics** (aantal mentions gerelateerd aan programma)
- **Sportevenementen** (voetbalwedstrijden, Olympische Spelen etc.)

4. Historische Kijkcijfers

- **Gemiddelde kijkcijfers vorige afleveringen** (rolling average)
- **Kijkcijfer trend** (stijgend/dalend over laatste 5 afleveringen)
- **Seizoensgebonden populariteit** (bijv. sinterklaasprogramma's in december)

5. Technische Features

- **Beschikbaarheid op streamingplatforms** (Is het programma ook online te zien?)
- **Uitzendformaat** (HD, 4K, live vs opgenomen)

Implementatievoorbeeld

```
from datetime import datetime

def add_features(row):
    # Tijdgerelateerde features
    date = datetime.strptime(row['dateDiff'], '%Y-%m-%d')
    row['weekday'] = date.weekday() # 0=maandag, 6=zondag
    row['is_weekend'] = 1 if row['weekday'] >= 5 else 0

    # Programma-duur in minuten
    h, m, s = map(int, row['rLength'].split(':'))
    row['duration_min'] = h * 60 + m + s / 60

    # Avondprime time (18:00-23:00)
    row['is_prime_time'] = 1 if 18 <= int(row['startTime'][:2]) < 23 else 0

    return row

# Toepassen op elke rij voor het trainen
enhanced_data = [add_features(row) for row in your_data]
```

Belangrijkste features voor starters

1. **Prime time indicator** (avonduren)
2. **Weekend vs werkdag**
3. **Programmagenre**
4. **Zendernaam** (VRT vs VTM vs andere)
5. **Historisch gemiddelde** voor vergelijkbare programma's

Data-bronnen

- Feestdagen: <https://www.feestdagen-belgie.be/> (<https://www.feestdagen-belgie.be/>)
- Weerdata: <https://www.meteo.be/nl/weer/waarnemingen> (<https://www.meteo.be/nl/weer/waarnemingen>)
- Programmagids: <https://www.tvgids.nl/> (<https://www.tvgids.nl/>) of officiële zendergidsen

Hoe meer van deze features je kunt toevoegen (met gezond verstand), hoe accurater je voorspellingsmodel zal worden!

Voorbeeld 6: Programma genre, feestdagen en schoolvakantie bepalen via API's

Prompt:

"Hoe kan ik met code bepalen of welk genre een programma is? kan ik deze data online curlen? Hetwelfde voor feestdagen en Schoolvakantie indicators. Indien er hier api's voor bestaan geef ze me dan"

Antwoord AI:

Hier is een overzicht van hoe je programmagenres, feestdagen en schoolvakanties kunt bepalen via API's of webscraping:

1. Programmagenres bepalen

API Opties:

- **TMDB API** (The Movie Database):
Biedt genres voor films en TV-programma's via `genre/movie/list` en `genre/tv/list` endpoints.
Voorbeeld: https://api.themoviedb.org/3/genre/tv/list?api_key=YOUR_KEY *Voordelen:* Gestructureerde data, gratis tier beschikbaar
Nadelen: Beperkt tot films/series, geen Vlaamse specifieke data

Alternatief:

- Scrape genres van programmagidsen zoals:
 - VRT MAX of VTM GO websites
 - IMDb via hun API of scraping tools

2. Feestdagen API's

Voor België/Vlaanderen:

- **OpenHolidaysAPI**

Biedt feestdagen voor België (en 30+ andere landen):

```
curl -X GET 'https://openholidaysapi.org/PublicHolidays?countryIsoCode=BE&languageIsoCode=NL&validFrom=2025-01-01&validTo=2025-
```

Velden: name, startDate, endDate, nationwide

Alternatieven:

- Scrape officiële bronnen zoals:
 - https://www.belgium.be/nl/over_belgie/land/feestdagen (https://www.belgium.be/nl/over_belgie/land/feestdagen)
 - Wikipedia-pagina's over Belgische feestdagen

3. Schoolvakantie Indicators

API Opties:

- **OpenHolidaysAPI** (zelfde als boven):

```
curl -X GET 'https://openholidaysapi.org/SchoolHolidays?countryIsoCode=BE&subdivisionCode=BE-VLG&languageIsoCode=NL&validFrom=2
```

Filterbaar per regio (bv. Vlaanderen met BE-VLG)

Vlaamse bronnen:

- Officiële kalender:
<https://onderwijs.vlaanderen.be/nl/schoolvakanties> (<https://onderwijs.vlaanderen.be/nl/schoolvakanties>)
- Scrape data van onderwijsportalen

Implementatievoorbeeld (Python)

```
import requests
from datetime import datetime

# Config
TMDB_API_KEY = 'your_tmdb_key'
today = datetime.now().strftime('%Y-%m-%d')

# 1. Haal genres op
def get_genres(programma_naam):
    url = f"https://api.themoviedb.org/3/search/tv?query={programma_naam}&api_key={TMDB_API_KEY}"
    response = requests.get(url).json()
    if response['results']:
        return response['results'][0]['genre_ids'] # Gebruik genre IDs voor verdere lookup
    return []

# 2. Check feestdag
def is_feestdag():
    url = f"https://openholidaysapi.org/PublicHolidays?countryIsoCode=BE&validFrom={today}&validTo={today}"
    return bool(requests.get(url).json())

# 3. Check schoolvakantie
def is_schoolvakantie():
    url = f"https://openholidaysapi.org/SchoolHolidays?countryIsoCode=BE&subdivisionCode=BE-VLG&validFrom={today}&validTo={today}"
    return bool(requests.get(url).json())
```

Aanvullende tips

1. **Caching:** Sla API-responses lokaal op om rate limits te vermijden
2. **Fallback:** Voor programma's niet in TMDB, scrape EPG-data van zenders
3. **Dataverrijking:** Combineer met weersdata voor betere voorspellingen

Let op: Voor scraping altijd:

- Check robots.txt

- Gebruik delays tussen requests (bv. `time.sleep(1)`)
- Respecteer copyright

Voorbeeld 7: Feestdagen CSV opschonen naar enkel NL naam en datum

Prompt:

"Ik heb de csv's zelf gedownload. Zorg ervoor dat de csv aangepast wordt door naar enkel Startdate, en naam maar pak hier enkel de NL vertaling uit, al de rest mag weg."

Antwoord AI:

Hier is de Python code om de CSV aan te passen volgens je vereisten:

```
import csv
from datetime import datetime

def clean_holidays_file(input_file, output_file):
    with open(input_file, 'r', encoding='utf-8') as infile, \
        open(output_file, 'w', newline='', encoding='utf-8') as outfile:

        reader = csv.DictReader(infile, delimiter=';')
        writer = csv.DictWriter(outfile, fieldnames=['date', 'holiday_name'], delimiter=';')
        writer.writeheader()

        for row in reader:
            # Pak alleen de NL naam (alles tussen "NL " en de volgende komma)
            name_nl = row['Name'].split('NL ')[1].split(',')[0].strip()

            # Schrijf alleen de gewenste velden
            writer.writerow({
                'date': row['StartDate'],
                'holiday_name': name_nl
            })

# Pas aan voor jouw bestandsnamen
clean_holidays_file('holidays.public.csv', 'holidays_clean.csv')
```

Wat deze code doet

1. Leest het originele CSV-bestand in
2. Filtert alleen de Nederlandse namen uit het 'Name' veld
3. Bewaart alleen:
 - date: Startdatum van de feestdag
 - holiday_name: Nederlandse naam van de feestdag
4. Schrijft naar een nieuw CSV-bestand met puntkomma als delimiter

Voorbeeld output

```
date;holiday_name
2020-01-01;Nieuwjaarsdag
2020-04-12;Pasen
2020-04-13;Paasmaandag
...
```

Extra opties

1. **Datumformatting** aanpassen:

```
# Als je de datum in ander formaat wilt
date_obj = datetime.strptime(row['StartDate'], '%Y-%m-%d')
formatted_date = date_obj.strftime('%d/%m/%Y') # DD/MM/YYYY
```

2. **Alternatief voor naam extractie** (robuuster):

```
import re
name_nl = re.search(r'NL ([^,]+)', row['Name']).group(1).strip()
```

3. Command line versie:

```
python -c "import csv,sys;[print(f\"{r['StartDate']};{r['Name'].split('NL ')[1].split(',')[0].strip()}\") for r in csv.DictReader(open('holidays.school.vlaanderen.csv'))]"
```

Voorbeeld 8: Schoolvakantie CSV opschonen naar enkel NL naam en datums

Prompt:

"Doe nu hetzelfde voor deze csv. enkel startdate, enddate zijn nodig en de naam vd vakantie in het NL"

Antwoord AI:

Hier is de Python code om de schoolvakantie CSV aan te passen volgens je vereisten:

```
import csv

def clean_school_holidays(input_file, output_file):
    with open(input_file, 'r', encoding='utf-8') as infile, \
        open(output_file, 'w', newline='', encoding='utf-8') as outfile:

        reader = csv.DictReader(infile, delimiter=';')
        writer = csv.DictWriter(outfile, fieldnames=['start_date', 'end_date', 'holiday_name'], delimiter=';')
        writer.writeheader()

        for row in reader:
            # Pak alleen de NL naam (alles tussen "NL " en de volgende komma)
            name_nl = row['Name'].split('NL ')[1].split(',')[0].strip()

            # Schrijf alleen de gewenste velden
            writer.writerow({
                'start_date': row['StartDate'],
                'end_date': row['EndDate'],
                'holiday_name': name_nl
            })

# Pas aan voor jouw bestandsnamen
clean_school_holidays('holidays.school.vlaanderen.csv', 'school_holidays_clean.csv')
```

Wat deze code doet

1. Leest het originele CSV-bestand in met schoolvakanties
2. Filtert alleen de Nederlandse namen uit het 'Name' veld
3. Bewaart alleen:
 - o start_date: Begin datum van de vakantie
 - o end_date: Eind datum van de vakantie
 - o holiday_name: Nederlandse naam van de vakantie
4. Schrijft naar een nieuw CSV-bestand met puntkomma als delimiter

Voorbeeld output

```
start_date;end_date;holiday_name
2019-12-23;2020-01-05;Kerstvakantie
2020-02-24;2020-03-01;Krokusvakantie
2020-04-06;2020-04-19;Paasvakantie
...
```

Extra opties

1. Datumformatting aanpassen:

```
from datetime import datetime

# Als je de datum in ander formaat wilt
start_date = datetime.strptime(row['StartDate'], '%Y-%m-%d').strftime('%d/%m/%Y')
end_date = datetime.strptime(row['EndDate'], '%Y-%m-%d').strftime('%d/%m/%Y')
```

2. Alternatieve naam extractie (robuuster voor verschillende formaten):

```
import re
name_nl = re.search(r'NL ([^,;]+)', row['Name']).group(1).strip()
```

3. Command line versie (voor snelle verwerking):

```
python -c "import csv,sys;[print(f\"{r['StartDate']};{r['EndDate']};{r['Name'].split('NL ')[1].split(',')[0].strip()}\") for r in csv.reader(sys.stdin)]"
```

Voorbeeld 9: Dataframes combineren en is_holiday feature toevoegen

Prompt:

"Ok ik heb nu deze features: dateDiff,ranking,description,channel,startTime,rLength,rateInK en dan uit de andere csv's: date;holiday_name en start_date;end_date;holiday_name. Ik ga later nog aan feature engineering doen maar eerst wil ik deze csv's in pandas krijgen. Geef mij de code om een df te maken met dateDiff,ranking,description,channel,startTime,rLength,rateInK,is_holiday. is_holiday gaat dus de holiday csv's af en als de datediff op een holiday zit dan geeft hij dat aan"

Antwoord AI:

Hier is de Python code om een Pandas DataFrame te maken met alle features, inclusief de is_holiday indicator:


```

import pandas as pd
from datetime import datetime

# Laad de hoofddata
df_main = pd.read_csv('kijkcijfers.csv', delimiter=';')

# Laad de feestdagen data
df_holidays = pd.read_csv('holidays_clean.csv', delimiter=';')
df_holidays['date'] = pd.to_datetime(df_holidays['date'])

# Laad de schoolvakanties data
df_school_holidays = pd.read_csv('school_holidays_clean.csv', delimiter=';')
df_school_holidays['start_date'] = pd.to_datetime(df_school_holidays['start_date'])
df_school_holidays['end_date'] = pd.to_datetime(df_school_holidays['end_date'])

# Converteer dateDiff naar datetime
df_main['dateDiff'] = pd.to_datetime(df_main['dateDiff'])

# Maak is_holiday feature
def check_holiday(date):
    # Check publieke feestdagen
    if date in df_holidays['date'].values:
        return 1

    # Check schoolvakanties
    for _, row in df_school_holidays.iterrows():
        if row['start_date'] <= date <= row['end_date']:
            return 1

    return 0

df_main['is_holiday'] = df_main['dateDiff'].apply(check_holiday)

# Optioneel: Voeg naam van feestdag/vakantie toe voor verificatie
def get_holiday_name(date):
    # Publieke feestdagen
    holiday = df_holidays[df_holidays['date'] == date]
    if not holiday.empty:
        return holiday.iloc[0]['holiday_name']

    # Schoolvakanties
    for _, row in df_school_holidays.iterrows():
        if row['start_date'] <= date <= row['end_date']:
            return row['holiday_name']

    return 'Geen feestdag'

df_main['holiday_name'] = df_main['dateDiff'].apply(get_holiday_name)

# Toon resultaat
print(df_main.head())
print(f"\nAantal dagen met feestdagen: {df_main['is_holiday'].sum()}")

```

Wat deze code doet

1. **Laadt alle CSV-bestanden** in aparte DataFrames
2. **Converteert datums** naar datetime objecten voor correcte vergelijking
3. **Maakt is_holiday feature:**
 - o 1 als de datum in een feestdag of schoolvakantie valt
 - o 0 anders
4. **Optioneel:** Voegt ook de naam van de feestdag/vakantie toe voor verificatie
5. **Toont resultaat** met voorbeeldrijen en aantal feestdagen

Aanvullende optimalisaties

1. Voor grote datasets kun je de `iterrows()` vervangen door vectorized operaties:

```
# Snellere implementatie voor grote datasets
holiday_dates = set(df_holidays['date'])
school_ranges = [(row['start_date'], row['end_date']) for _, row in df_school_holidays.iterrows()]

def check_holiday_fast(date):
    if date in holiday_dates:
        return 1
    return any(start <= date <= end for start, end in school_ranges)

df_main['is_holiday'] = df_main['dateDiff'].apply(check_holiday_fast)
```

2. Feature engineering suggesties voor later:

```
# Voorbeeld: dag van de week toevoegen
df_main['day_of_week'] = df_main['dateDiff'].dt.dayofweek # 0=maandag, 6=zondag

# Voorbeeld: weekend indicator
df_main['is_weekend'] = df_main['day_of_week'].apply(lambda x: 1 if x >= 5 else 0)
```

3. Opslaan van het resultaat:

```
df_main.to_csv('kijkcijfers_met_features.csv', index=False, sep=';')
```

Voorbeeld 10: Feature engineering voor tijd en dag

Prompt:

"Het volgende dat ik wil doen is aan feature engineering doen. Ik wil dat volgende features er allemaal bijkomen: weekday: dag van de week, month: maandnummer, is_primetype: binaire feature (1 = tussen 18:30u-22u), is_weekend: binaire feature (1 = zaterdag/zondag, 0 = anders)"

Antwoord AI:

Hier is de Python code om de gevraagde features toe te voegen aan je DataFrame:

```

import pandas as pd
from datetime import datetime

# Aannname: df_main bestaat al en heeft een 'dateDiff' kolom in datetime formaat
# Zo niet, converteer eerst:
# df_main['dateDiff'] = pd.to_datetime(df_main['dateDiff'])

# 1. Voeg weekday toe (0=maandag, 6=zondag)
df_main['weekday'] = df_main['dateDiff'].dt.dayofweek

# 2. Voeg maandnummer toe
df_main['month'] = df_main['dateDiff'].dt.month

# 3. Voeg is_primetime toe (18:30-22:00)
# Eerst startTime omzetten naar datetime voor vergelijking
df_main['startTime'] = pd.to_datetime(df_main['startTime'], format='%H:%M:%S').dt.time

# Maak primetime begin en eind
primetime_start = pd.to_datetime('18:30:00').time()
primetime_end = pd.to_datetime('22:00:00').time()

df_main['is_primetime'] = df_main['startTime'].apply(
    lambda x: 1 if primetime_start <= x <= primetime_end else 0
)

# 4. Voeg is_weekend toe
df_main['is_weekend'] = df_main['weekday'].apply(lambda x: 1 if x >= 5 else 0)

# Optioneel: Toon resultaat
print(df_main[['dateDiff', 'startTime', 'weekday', 'month', 'is_primetime', 'is_weekend']].head(10))

```

Toelichting bij de features

1. weekday:

- Gebruikt Pandas dayofweek functie
- Waarden: 0 (maandag) t/m 6 (zondag)

2. month:

- Maandnummer van 1 (januari) t/m 12 (december)

3. is_primetime:

- Gebaseerd op startTime kolom
- 1 als programma tussen 18:30u en 22:00u start
- 0 voor andere tijden

4. is_weekend:

- Gebaseerd op weekday
- 1 voor zaterdag (5) en zondag (6)
- 0 voor werkdagen

Extra optimalisaties

1. Vectorized operations voor betere performance:

```

# Snellere implementatie voor grote datasets
df_main['is_primetime'] = ((df_main['startTime'] >= primetime_start) &
                           (df_main['startTime'] <= primetime_end)).astype(int)

```

2. Categorical features voor machine learning:

```
# Voor sommige modellen kan dit beter werken
df_main['weekday_name'] = df_main['dateDiff'].dt.day_name()
df_main['month_name'] = df_main['dateDiff'].dt.month_name()
```

3. Seizoensfeature toevoegen:

```
seasons = {
    1: 'Winter', 2: 'Winter', 3: 'Spring', 4: 'Spring', 5: 'Spring',
    6: 'Summer', 7: 'Summer', 8: 'Summer', 9: 'Autumn',
    10: 'Autumn', 11: 'Autumn', 12: 'Winter'
}
df_main['season'] = df_main['month'].map(seasons)
```

4. Programmaduur in minuten:

```
# Converteer rLength (HH:MM:SS) naar minuten
df_main['duration_min'] = pd.to_timedelta(df_main['rLength']).dt.total_seconds() / 60
```

Opslaan van het resultaat

```
df_main.to_csv('./Data/CleanData/kijkcijfers_met_features.csv', index=False, sep=';')
```

Voorbeeld 11: Features op basis van description (is_news, is_sports)

Prompt:

"De laatste feature die ik er nog aan wil toevoegen zijn features gebaseerd op de description. Ik wil de kolommen is_news, is_sports deze baseren zich op woorden als journaal, nieuws of wielrennen, voetbal, golf, boxen"

Antwoord AI:

Hier is de code om de is_news en is_sports features toe te voegen gebaseerd op trefwoorden in de description kolom:

```

import pandas as pd
import re

# Lijst van nieuwsgelateerde trefwoorden (kan uitgebreid worden)
news_keywords = [
    'journaal', 'nieuws', 'news', 'actualiteit', 'bulletin',
    'nieuwsuitzending', 'nieuwsvloot', 'nieuwsupdate'
]

# Lijst van sportgerelateerde trefwoorden (kan uitgebreid worden)
sports_keywords = [
    'voetbal', 'wielrennen', 'golf', 'boxen', 'tennis', 'sport',
    'match', 'wedstrijd', 'competitie', 'race', 'atletiek', 'zwemmen',
    'volleybal', 'basketbal', 'handbal', 'hockey', 'formule 1', 'f1'
]

# Maak regex patronen voor efficiënte matching
news_pattern = re.compile('|'.join(news_keywords), flags=re.IGNORECASE)
sports_pattern = re.compile('|'.join(sports_keywords), flags=re.IGNORECASE)

# Voeg features toe
df_main['is_news'] = df_main['description'].apply(
    lambda x: 1 if news_pattern.search(str(x)) else 0
)

df_main['is_sports'] = df_main['description'].apply(
    lambda x: 1 if sports_pattern.search(str(x)) else 0
)

# Optioneel: Voeg een 'program_type' kolom toe voor handmatige verificatie
def determine_program_type(desc):
    desc = str(desc).lower()
    if news_pattern.search(desc):
        return 'news'
    elif sports_pattern.search(desc):
        return 'sports'
    return 'other'

df_main['program_type'] = df_main['description'].apply(determine_program_type)

# Toon resultaten
print("\nAantal nieuwsprogramma's:", df_main['is_news'].sum())
print("Aantal sportprogramma's:", df_main['is_sports'].sum())
print("\nVoorbeeld van geclassificeerde programma's:")
print(df_main[['description', 'is_news', 'is_sports', 'program_type']].sample(10))

```

Belangrijke aanpassingen

1. Case-insensitive matching:

- Gebruik `re.IGNORECASE` zodat 'Journaal' en 'journaal' hetzelfde matchen

2. Handling van ontbrekende waarden:

- `str(x)` conversie zorgt dat NaN-waarden niet crashen

3. Uitbreidbare trefwoordenlijsten:

- Voeg eenvoudig nieuwe trefwoorden toe aan de lijsten

4. Verificatie kolom:

- De `program_type` kolom helpt bij het controleren van de classificatie

Voorbeeld van uitbreiding

```
# Uitbreiding van trefwoordenlijsten
news_keywords.extend(['nieuwsrubriek', 'nieuwsdienst'])
sports_keywords.extend(['atletiek', 'zwemmen', 'schaatsen'])

# Update de regex patronen
news_pattern = re.compile(''.join(news_keywords), flags=re.IGNORECASE)
sports_pattern = re.compile(''.join(sports_keywords), flags=re.IGNORECASE)
```

Optimalisatie voor grote datasets

Voor betere performance met grote datasets:

```
# Vectorized approach (sneller voor grote datasets)
df_main['is_news'] = df_main['description'].str.contains(
    ''.join(news_keywords),
    case=False,
    regex=True,
    na=False
).astype(int)

df_main['is_sports'] = df_main['description'].str.contains(
    ''.join(sports_keywords),
    case=False,
    regex=True,
    na=False
).astype(int)
```

Opslaan van het resultaat

```
df_main.to_csv('./Data/CleanData/kijkcijfers_met_features_v2.csv', index=False, sep=';')
```

Voorbeeld 12: Outlier detectie en distributieanalyse kijkcijfers

Prompt:

"Ik zie enkele fouten in de grafieken. Eerst en vooral klopt er iets niet aan de distributie van de kijkcijfers. De grafiek hangt sterk naar links. Ik denk dat er dus enkele outliers zijn, we gaan deze eruit halen"

Antwoord AI:

Hier is de verbeterde analyse met outlier-detectie en verwijdering voor de kijkcijfers (rateInK):

1. Outlier Detectie en Verwijdering

```
# Bereken de IQR voor rateInK
Q1 = df_final['rateInK'].quantile(0.25)
Q3 = df_final['rateInK'].quantile(0.75)
IQR = Q3 - Q1

# Definieer grenzen voor outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Maak een schone versie zonder outliers
df_clean = df_final[(df_final['rateInK'] >= lower_bound) &
                    (df_final['rateInK'] <= upper_bound)].copy()

print(f"Originele dataset: {len(df_final)} rijen")
print(f"Na outlier verwijdering: {len(df_clean)} rijen")
print(f"Verwijderd: {len(df_final) - len(df_clean)} rijen ({(len(df_final)-len(df_clean))/len(df_final)*100:.2f}%)")
```

2. Verbeterde Visualisaties

```

plt.figure(figsize=(18, 12))

# 1. Vergelijkende distributie voor/na outlier verwijdering
plt.subplot(2, 3, 1)
sns.kdeplot(df_final['rateInK'], label='Origineel', fill=True)
sns.kdeplot(df_clean['rateInK'], label='Zonder outliers', fill=True)
plt.title('Vergelijkende distributie van kijkcijfers')
plt.xlabel('Kijkcijfers (rateInK)')
plt.legend()

# 2. Boxplot voor/na outlier verwijdering
plt.subplot(2, 3, 2)
sns.boxplot(data=pd.DataFrame({
    'Origineel': df_final['rateInK'],
    'Zonder outliers': df_clean['rateInK']
}))
plt.title('Boxplot vergelijking')
plt.ylabel('Kijkcijfers (rateInK)')

# 3. Channel populariteit zonder outliers
plt.subplot(2, 3, 3)
channel_order = df_clean['channel'].value_counts().index
sns.barplot(y=df_clean['channel'].value_counts().index,
            x=df_clean['channel'].value_counts().values,
            order=channel_order)
plt.title('Aantal programma\'s per zender (zonder outliers)')
plt.xlabel('Aantal programma\'s')

# 4. Programmaduur vs kijkcijfers (schoon)
plt.subplot(2, 3, 4)
sns.scatterplot(x='duration_min', y='rateInK', hue='is_weekend',
               data=df_clean, alpha=0.6, palette='viridis')
plt.title('Relatie duur en kijkcijfers (zonder outliers)')
plt.xlabel('Duur (minuten)')
plt.ylabel('Kijkcijfers')

# 5. Weekdag patronen (schoon)
plt.subplot(2, 3, 5)
weekday_names = ['Ma', 'Di', 'Wo', 'Do', 'Vr', 'Za', 'Zo']
sns.boxplot(x='weekday', y='rateInK', data=df_clean,
            showfliers=False)
plt.xticks(range(7), weekday_names)
plt.title('Kijkcijfers per weekday (zonder outliers)')
plt.xlabel('Weekdag')
plt.ylabel('Kijkcijfers')

# 6. Programmatype analyse (schoon)
plt.subplot(2, 3, 6)
sns.boxplot(x='program_type', y='rateInK', data=df_clean,
            showfliers=False, order=['Nieuws', 'Sport', 'Film', 'Overig'])
plt.title('Kijkcijfers per programmatype (zonder outliers)')
plt.xlabel('Programmatype')
plt.ylabel('Kijkcijfers')

plt.tight_layout()
plt.show()

```

3. Aanvullende Analyse

```
# Tijdspatronen analyse zonder outliers
plt.figure(figsize=(12, 6))
df_clean['hour'] = pd.to_datetime(df_clean['startTime_fixed']).dt.hour
sns.lineplot(x='hour', y='rateInK', hue='is_weekend',
             data=df_clean, ci=None, estimator='median',
             palette='coolwarm')
plt.title('Mediane kijkcijfers per uur (zonder outliers)')
plt.xticks(range(0, 24))
plt.xlabel('Uur van de dag')
plt.ylabel('Mediane kijkcijfers')
plt.legend(title='Weekend', labels=['Weekdag', 'Weekend'])
plt.show()
```

Belangrijke Verbeteringen

1. Outlier behandeling:

- Gebruik van IQR-methode voor objectieve outlier-detectie
- Behoud van 93-97% van de data (typisch bij deze methode)
- Duidelijke visualisatie van het effect van outlier-verwijdering

2. Verbeterde visualisaties:

- Gebruik van mediane i.p.v. gemiddelde waarden waar relevant
- Betere leesbaarheid met weekdagnamen i.p.v. nummers
- Consistent kleurenschema in alle plots

3. Robuustere analyses:

- Boxplots zonder outliers voor betere interpretatie
- Scatterplots met transparantie voor betere densiteitsweergave
- Vergelijkende plots voor/na cleaning

Interessante Bevindingen om te Checken

1. Is er nog steeds een scheve distributie na outlier-verwijdering?
2. Welke zenders hebben de meeste outliers?
3. Hoe verhouden verschillende programmatypes zich na cleaning?
4. Zijn de tijdspatronen duidelijker zichtbaar?

Wil je dat ik me focus op specifieke aspecten van deze schonere dataset, of aanvullende analyses uitvoer?

Voorbeeld 13: Pipelines voor onehotencoding en standaardiseren

Prompt:

"Maak een onehotencoder pipeline. Deze gaan we later gebruiken om de channel categorie te onehotencoden en maak ook een standaardiseer pipeline. Later gaan we die gebruiken om de numerieke waarden te standaardiseren."

Antwoord AI:

Perfect! We kunnen hiervoor gebruik maken van `ColumnTransformer` en `Pipeline` uit `scikit-learn`. Ik stel hieronder twee aparte pipelines voor:

❑ 1. OneHotEncoder pipeline voor channel

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

# One-hot encode alleen de 'channel' kolom
onehot_pipeline = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

❑ 2. Standaardiseerpipeline voor numerieke kolommen


```
from sklearn.preprocessing import StandardScaler

# Standaardiseer alleen numerieke kolommen zoals 'duration_min', 'hour', etc.
standardize_pipeline = Pipeline(steps=[
    ('scaler', StandardScaler())
])
```

□ Later: combineren in een ColumnTransformer

Wanneer je klaar bent om alles samen te voegen (bv. vóór een regressiemodel), doe je dit:

```
from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer(transformers=[
    ('num', standardize_pipeline, ['duration_min', 'hour']),
    ('cat', onehot_pipeline, ['channel'])
], remainder='passthrough') # 'passthrough' houdt andere kolommen ongemoeid
```

En als je daarna bijvoorbeeld een `LinearRegression`-model wil trainen:

```
from sklearn.linear_model import LinearRegression

full_pipeline = Pipeline(steps=[
    ('preprocessing', preprocessor),
    ('model', LinearRegression())
])
```

Laat me weten:

- welke numerieke kolommen je wil standaardiseren,
- of je nog andere categorische kolommen hebt om te encoden,
- en welk model je eerst wil proberen (bv. `LinearRegression`, `RandomForestRegressor`, ...)

Dan help ik je met een volledig trainingsscript.

Voorbeeld 14: Starttime als feature verwerken (minutes since midnight, cyclisch)

Prompt:

"StartTime_fixed is een object en als ik probeer te trainen dan lukt dit niet is er een goeie manier om starttime toch te gebruiken als een feature? Ik heb hour al maar is het nuttig om minutes hier ook aan toe te voegen als aparte feature of kan ik deze beter samenvoegen tot 1 feature"

Antwoord AI:

Hier zijn drie goed werkende opties om `startTime_fixed` effectief te gebruiken als feature:

1. Aanbevolen Optie: Sinus/Cosinus Transformatie voor Cyclische Tijd

```
def encode_cyclical_time(df):
    # Converteer naar datetime en extraheer minuten sinds middernacht
    df['time_minutes'] = pd.to_datetime(df['startTime_fixed']).dt.hour * 60 + pd.to_datetime(df['startTime_fixed']).dt.minute

    # Sinus/cosinus transformatie
    df['time_sin'] = np.sin(2 * np.pi * df['time_minutes']/1440) # 1440 min = 24u
    df['time_cos'] = np.cos(2 * np.pi * df['time_minutes']/1440)

    return df.drop(['startTime_fixed', 'time_minutes'], axis=1)

df = encode_cyclical_time(df)
```

Voordelen:

- Behoudt de cyclische natuur van tijd (23:59 is dichtbij 00:00)

- Werkt uitstekend voor ML-modellen
- Eén feature-paar vervangt zowel `hour` als `minute`

2. Tijdsblokken Creatie

```
def create_time_blocks(df):
    time_bins = [0, 6, 9, 12, 15, 18, 21, 24]
    time_labels = ['nacht', 'ochtend', 'voormiddag', 'middag', 'namiddag', 'avond', 'late avond']

    df['time_block'] = pd.cut(
        pd.to_datetime(df['startTime_fixed']).dt.hour,
        bins=time_bins,
        labels=time_labels,
        right=False
    )

    # One-hot encoding toepassen
    return pd.get_dummies(df, columns=['time_block'])

df = create_time_blocks(df)
```

Voordelen:

- Menselijk interpreteerbaar
- Vermindert dimensionaliteit
- Goede baseline voor snelle experimenten

3. Numerieke Minuten sinds Middernacht

```
def minutes_since_midnight(df):
    df['total_minutes'] = pd.to_datetime(df['startTime_fixed']).dt.hour * 60 + pd.to_datetime(df['startTime_fixed']).dt.minute
    return df.drop('startTime_fixed', axis=1)

df = minutes_since_midnight(df)
```

Voordelen:

- Eenvoudigste numerieke representatie
- Bevat meer informatie dan alleen `hour`
- Lineaire modellen kunnen hier goed mee werken

Wat te doen met bestaande `hour` feature?

- **Houden** als je kiest voor optie 2 of 3
- **Verwijderen** als je kiest voor optie 1 (sinus/cosinus bevat al deze info)
- **Minuten apart toevoegen** heeft weinig toegevoegde waarde boven de minuten-sinds-middernacht aanpak

Voorbeeld Pipeline Integratie

```
def preprocessing_time_features(df):
    # Behoud bestaande hour feature voor backwards compatibility
    df = df.copy()

    # Optie 1 - Aanbevolen
    df = encode_cyclical_time(df)

    # OF Optie 2
    # df = create_time_blocks(df)

    # OF Optie 3
    # df = minutes_since_midnight(df)

    return df

# Gebruik in bestaande pipeline:
df = preprocessing_time_features(df)
```

Kies optie 1 voor de beste modelprestaties, optie 2 voor interpretatie, of optie 3 voor eenvoud. De sinus/cosinus aanpak is bijzonder effectief gebleken voor tijdsfeatures in ML-modellen.

Voorbeeld 15: Gridsearch opzetten voor regressiemodellen

Prompt:

"Ik heb verschillende regressiemodellen getest op de trainingsdata en wil nu met gridsearch op zoek gaan naar de beste modellen. Ik ga hieronder de code geven van de modellen die potentie hebben. Jij moet dan een gridsearch opstellen."

Antwoord AI:

Hier is een uitgebreide GridSearch setup voor elk van je modellen, waarbij ik focus op de belangrijkste hyperparameters voor optimalisatie:

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

# Definieer een functie om grid search uit te voeren en resultaten weer te geven
def run_grid_search(pipeline, param_grid, X_train, y_train, X_test, y_test):
    grid_search = GridSearchCV(
        estimator=pipeline,
        param_grid=param_grid,
        cv=5,
        scoring='neg_mean_squared_error',
        n_jobs=-1,
        verbose=1
    )
    grid_search.fit(X_train, y_train)

    print(f"Beste parameters: {grid_search.best_params_}")
    print(f"Beste RMSE (train): {(-grid_search.best_score_)**0.5:.2f}")

    # Evalueer op testset
    test_score = (-grid_search.score(X_test, y_test))*0.5
    print(f"RMSE (test): {test_score:.2f}")

    return grid_search.best_estimator_
```

1. Polynomiale Regressie

```

poly_param_grid = {
    'polynomialfeatures__degree': [1, 2, 3],
    'linearregression__fit_intercept': [True, False]
}

poly_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('polynomialfeatures', PolynomialFeatures()),
    ('linearregression', LinearRegression())
])

best_poly = run_grid_search(poly_pipeline, poly_param_grid, X_train, y_train, X_test, y_test)

```

2. Decision Tree

```

tree_param_grid = {
    'decisiontreeregressor__max_depth': [None, 5, 10, 20],
    'decisiontreeregressor__min_samples_split': [2, 5, 10],
    'decisiontreeregressor__min_samples_leaf': [1, 5, 10]
}

tree_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('decisiontreeregressor', DecisionTreeRegressor(random_state=42))
])

best_tree = run_grid_search(tree_pipeline, tree_param_grid, X_train, y_train, X_test, y_test)

```

3. Random Forest

```

rf_param_grid = {
    'randomforestregressor__n_estimators': [50, 100, 200],
    'randomforestregressor__max_depth': [None, 5, 10, 20],
    'randomforestregressor__min_samples_split': [2, 5, 10],
    'randomforestregressor__min_samples_leaf': [1, 2, 4]
}

rf_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('randomforestregressor', RandomForestRegressor(random_state=42))
])

best_rf = run_grid_search(rf_pipeline, rf_param_grid, X_train, y_train, X_test, y_test)

```

4. Gradient Boosting

```

gb_param_grid = {
    'gradientboostingregressor__n_estimators': [50, 100, 200],
    'gradientboostingregressor__learning_rate': [0.01, 0.1, 0.2],
    'gradientboostingregressor__max_depth': [3, 5, 7],
    'gradientboostingregressor__min_samples_split': [2, 5, 10]
}

gb_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('gradientboostingregressor', GradientBoostingRegressor(random_state=42))
])

best_gb = run_grid_search(gb_pipeline, gb_param_grid, X_train, y_train, X_test, y_test)

```

5. XGBoost

```
xgb_param_grid = {
    'xgbregressor__n_estimators': [50, 100, 200],
    'xgbregressor__learning_rate': [0.01, 0.1, 0.2],
    'xgbregressor__max_depth': [3, 5, 7],
    'xgbregressor__subsample': [0.8, 1.0],
    'xgbregressor__colsample_bytree': [0.8, 1.0]
}

xgb_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('xgbregressor', XGBRegressor(random_state=42))
])

best_xgb = run_grid_search(xgb_pipeline, xgb_param_grid, X_train, y_train, X_test, y_test)
```

Aanvullende tips

1. Parallelisatie: Alle grid searches gebruiken `n_jobs=-1` voor maximale parallelisatie

2. Scoring: Focus op RMSE, maar je kunt ook andere metrics toevoegen:

```
scoring = {
    'RMSE': 'neg_root_mean_squared_error',
    'R2': 'r2',
    'MAPE': 'neg_mean_absolute_percentage_error'
}
```

3. Early Stopping: Voor XGBoost en Gradient Boosting kun je early stopping toevoegen:

```
xgb_param_grid['xgbregressor__early_stopping_rounds'] = [10]
xgb_param_grid['xgbregressor__eval_set'] = [(X_test, y_test)]
```

4. RandomizedSearchCV: Voor grotere parameter spaces:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform
```

5. Model Evaluatie: Bewaar de beste modellen en vergelijk ze:

```
best_models = {
    'Polynomial': best_poly,
    'Decision Tree': best_tree,
    'Random Forest': best_rf,
    'Gradient Boosting': best_gb,
    'XGBoost': best_xgb
}

for name, model in best_models.items():
    y_pred = model.predict(X_test)
    rmse = root_mean_squared_error(y_test, y_pred)
    print(f"{name} - Test RMSE: {rmse:.2f}")
```

Voorbeeld 16: Visualisaties en Interpretatie

Prompt:

"Nu staat in de opdracht dit nog: Visualiseer en interpreteer je resultaten. – Zit je voornamelijk met overschatting of onderschatting? – Maak een rangschikking van de factoren die het meeste bijdragen aan de kijkcijfers. – Verklaar eventueel waarom bepaalde modellen beter presteren dan andere indien mogelijk.

Geef mij de code om het model te visualiseren en te interpreteren"

Antwoord AI:

Hier is een uitgebreide uitwerking van de gevraagde visualisaties en interpretaties:

```

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.inspection import permutation_importance

# Kies je model (vervang 'XGBoost' door het model van je keuze uit best_models)
geselecteerd_model = best_models['XGBoost']
model_naam = 'XGBoost' # Pas dit aan naar de naam van je gekozen model

# 1. Voorspellingen maken
y_pred = geselecteerd_model.predict(X_test)

# 2. Performance metrics berekenen
rmse = root_mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print(f"\nPerformance van {model_naam} model:")
print(f"- RMSE: {rmse:.2f}")
print(f"- R²: {r2:.2f}")
print(f"- MAPE: {mape:.2%}")

# 3. Visualisatie: Voorspelling vs Werkelijkheid
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--')
plt.xlabel('Werkelijke Kijkcijfers')
plt.ylabel('Voorspelde Kijkcijfers')
plt.title(f'{model_naam}: Voorspelling vs Werkelijkheid')
plt.grid(True)
plt.show()

# 4. Visualisatie: Voorspellingsfouten (residuals)
residuals = y_pred - y_test

plt.figure(figsize=(12, 6))
plt.hist(residuals, bins=50, edgecolor='black')
plt.axvline(x=0, color='r', linestyle='--')
plt.title(f'Verdeling van Voorspellingsfouten ({model_naam})')
plt.xlabel('Voorspellingsfout (Predicted - Actual)')
plt.ylabel('Frequentie')
plt.grid(True)
plt.show()

# 5. Overschatting vs Onderschatting analyse
overschat = np.sum(residuals > 0) / len(residuals) * 100
onderschat = np.sum(residuals < 0) / len(residuals) * 100
gelijk = np.sum(residuals == 0) / len(residuals) * 100

print(f"\nOverschatting vs Onderschatting analyse:")
print(f"- Percentage overschatting: {overschat:.1f}%")
print(f"- Percentage onderschatting: {onderschat:.1f}%")
print(f"- Perfecte voorspellingen: {gelijk:.1f}%")

# 6. Feature importance analyse
try:
    # Probeer eerst built-in feature importance
    if hasattr(geselecteerd_model.steps[-1][1], 'feature_importances_'):
        importances = geselecteerd_model.steps[-1][1].feature_importances_
        feature_names = geselecteerd_model.named_steps['preprocessing'].get_feature_names_out()
    else:
        # Gebruik permutation importance als built-in niet beschikbaar is
        result = permutation_importance(geselecteerd_model, X_test, y_test, n_repeats=10, random_state=42)
        importances = result.importances_mean

```

```

        feature_names = X_test.columns

importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(data=importance_df.head(10), x='Importance', y='Feature')
plt.title(f'Top 10 Belangrijkste Features ({model_naam})')
plt.tight_layout()
plt.show()

print("\nTop 5 belangrijkste features:")
print(importance_df.head(5).to_string(index=False))

except Exception as e:
    print(f"\nKon feature importance niet extraheren: {str(e)}")

# 7. Model interpretatie
print(f"\nInterpretatie van het {model_naam} model:")
print(f"1. Het model verklaart {r2:.0%} van de variantie in kijkcijfers (R²)")
print(f"2. De gemiddelde fout is {rmse:,.0f} kijkers (RMSE) of {mape:.0%} van de werkelijke waarde (MAPE)")
print(f"3. Het model heeft een tendens om kijkcijfers {'te overschatten' if overschat > onderschat else 'te onderschatten'}")
print(f"4. De belangrijkste features zijn hierboven weergegeven")

```

Hoe te gebruiken:

1. Vervang 'XGBoost' door de naam van het model dat je wilt analyseren (bijv. 'Random Forest')
2. De code werkt voor elk model in je `best_models` dictionary
3. Alle visualisaties en analyses worden automatisch gegenereerd

Wat de code bevat:

- Prestatiemetrics (RMSE, R², MAPE)
- Voorspelling vs werkelijkheid scatter plot
- Verdeling van voorspellingsfouten
- Overschatting/onderschatting analyse
- Feature importance ranking
- Duidelijke modelinterpretatie

Aanpassingen voor specifieke modellen:

- Voor lineaire modellen (zoals Polynomial): gebruik `coef_` in plaats van `feature_importances_`
- Voor Decision Trees: de feature importance is standaard beschikbaar
- Voor ensemble methoden: zowel built-in importance als permutation importance wordt ondersteund

Voorbeeld 17: Visualisaties en Interpretatie

Prompt:

(in bijlage de csv `input_voor_examen.csv`) "de file is een csv file dat ik moet gebruiken om voorspellingen te maken. Hieronder nog wat extra info over wat ik precies moet doen.

Beste student,

Zoals eerder aangegeven, zal je op het examen (gepland op 2, 3 en 4 juni) een bestand ontvangen volgens het formaat van het bestand in bijlage.

de datums in dit bestand (dat inhoudelijk individueel kan zijn) zullen niet meer dan enkele dagen in de toekomst liggen t.o.v. de examendatum je zorgt ervoor dat je een Python-applicatie (.py file) ter beschikking hebt die dit bestand kan inlezen en vervolgens predicties van de kijkcijfers kan doen, op basis van jouw opgeslagen model. Je kunt het bestand in bijlage dus gebruiken om je .py-script te testen, alleen zul je op het examen andere data krijgen. het is dus niet de bedoeling dat je tijdens het examen opnieuw gaat trainen om deze predicties te berekenen. er zullen in dit bestand geen tv-programma's in voorkomen die in het verleden nog nooit werden uitgezonden. deze predicties zullen niet gebruikt worden om je model te evalueren, maar enkel om na te gaan of je een werkend, opgeslagen model hebt. naast deze .py file dien je ook een .ipynb file in tegen zondagavond 1 juni om 22u00 op Chamilo (een opdracht zal nog aangemaakt worden), waarin je duidelijk aangeeft hoe je tot je optimale model gekomen bent."

Antwoord AI:

Hier is een complete Python script (`predict_kijkcijfers.py`) dat het opgeslagen model laadt en voorspellingen maakt voor het gegeven CSV-bestand:

```

import pandas as pd
import numpy as np
import re
from datetime import datetime
import pickle
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
import joblib

def preprocess_input_data(input_df, holidays_path='./Data/CleanData/holidays_clean.csv',
                          school_holidays_path='./Data/CleanData/school_holidays_clean.csv'):
    """
    Preprocess the input data in the same way as the training data was processed
    """
    # Load holidays data
    df_holidays = pd.read_csv(holidays_path, delimiter=';')
    df_holidays['date'] = pd.to_datetime(df_holidays['date'])

    df_school_holidays = pd.read_csv(school_holidays_path, delimiter=';')
    df_school_holidays['start_date'] = pd.to_datetime(df_school_holidays['start_date'])
    df_school_holidays['end_date'] = pd.to_datetime(df_school_holidays['end_date'])

    # Convert dates and times
    input_df['Datum'] = pd.to_datetime(input_df['Datum'], dayfirst=True)
    input_df['Start'] = pd.to_datetime(input_df['Start'], format='%H:%M:%S').dt.time

    # Convert duration to minutes
    def convert_duration(dur_str):
        h, m, s = map(int, dur_str.split(':'))
        return h * 60 + m + s / 60

    input_df['duration_min'] = input_df['Duur'].apply(convert_duration)

    # Create features
    input_df['weekday'] = input_df['Datum'].dt.dayofweek
    input_df['month'] = input_df['Datum'].dt.month
    input_df['hour'] = input_df['Start'].apply(lambda x: x.hour)

    # Minutes since midnight
    input_df['minutes_since_midnight'] = input_df['Start'].apply(lambda x: x.hour * 60 + x.minute)

    # Check if holiday
    def check_holiday(date):
        if date in df_holidays['date'].values:
            return 1
        for _, row in df_school_holidays.iterrows():
            if row['start_date'] <= date <= row['end_date']:
                return 1
        return 0

    input_df['is_holiday'] = input_df['Datum'].apply(check_holiday)
    input_df['is_weekend'] = input_df['weekday'].apply(lambda x: 1 if x >= 5 else 0)

    # Prime time feature (18:30 - 22:00)
    input_df['is_primetime'] = input_df['Start'].apply(
        lambda x: 1 if (18, 30) <= (x.hour, x.minute) <= (22, 0) else 0
    )

    # News program feature
    news_keywords = ['journaal', 'nieuws', 'sportweekend']
    news_pattern = re.compile('|'.join(news_keywords), flags=re.IGNORECASE)
    input_df['is_news'] = input_df['Programma'].apply(lambda x: 1 if news_pattern.search(str(x)) else 0)

    # Select and rename columns to match training data

```



```

processed_df = input_df[['Zender', 'minutes_since_midnight', 'duration_min',
                        'weekday', 'month', 'hour', 'is_primetype',
                        'is_weekend', 'is_holiday', 'is_news']]

processed_df = processed_df.rename(columns={'Zender': 'channel'})

return processed_df

def main():
    # Load the trained model
    try:
        model = joblib.load('best_model.pkl') # Change to your model filename
    except FileNotFoundError:
        print("Error: Model file 'best_model.pkl' not found. Please train the model first.")
        return

    # Read input data
    input_file = 'Input_voor_examen.csv'
    try:
        input_df = pd.read_csv(input_file, delimiter=';', encoding='utf-8-sig')
    except FileNotFoundError:
        print(f"Error: Input file '{input_file}' not found.")
        return

    # Preprocess the input data
    processed_data = preprocess_input_data(input_df)

    # Make predictions
    predictions = model.predict(processed_data)

    # Add predictions to original dataframe
    input_df['Voorspelde_Kijkcijfers'] = predictions.round().astype(int)

    # Save results
    output_file = 'voorspellingen_kijkcijfers.csv'
    input_df.to_csv(output_file, index=False, sep=';')

    print(f"Voorspellingen succesvol opgeslagen in {output_file}")
    print("\nEerste 5 voorspellingen:")
    print(input_df[['Programma', 'Zender', 'Datum', 'Start', 'Voorspelde_Kijkcijfers']].head())

if __name__ == "__main__":
    main()

```

Instructies voor gebruik:

1. **Bewaar dit script als** `predict_kijkcijfers.py`

2. **Zorg dat je deze bestanden hebt:**

- Je getrainde model (standaard zoekt het naar `best_model.pkl` - pas dit aan als je model een andere naam heeft)
- De input CSV (`Input_voor_examen.csv`)
- De holidays data (`./Data/CleanData/holidays_clean.csv` en `school_holidays_clean.csv`)

3. **Voer het script uit:**

```
python predict_kijkcijfers.py
```

4. **Output:**

- Het script maakt een `voorspellingen_kijkcijfers.csv` bestand met de voorspelde kijkcijfers
- Het toont ook de eerste 5 voorspellingen in de console

Belangrijke aanpassingen die je mogelijk moet doen:

1. **Model bestandsnaam:** Pas `best_model.pkl` aan naar de naam van je opgeslagen model

2. **Model type:** Als je een ander model dan XGBoost gebruikt, zorg dan dat de preprocessing compatibel is
3. **Pad naar holidays data:** Pas de paden aan als je bestanden ergens anders staan

Werking van het script:

1. **Preprocessing:** Het transformeert de input data op dezelfde manier als je trainingsdata
2. **Feature engineering:** Maakt alle features aan die je model verwacht
3. **Voorspelling:** Laadt het opgeslagen model en maakt voorspellingen
4. **Output:** Slaat de voorspellingen op in een nieuw CSV-bestand

Zorg ervoor dat je dit script test met het voorbeeld CSV-bestand voordat je het examen aflegt!

4. Evaluatie van de AI-tool

- **Nut:**

Ik vond Deepseek een heel nuttige tool die me heel hard geholpen heeft bij het maken van deze opdracht. Zeker als je niet eht weet hoe te beginnen is het heel handig om te gebruiken aangezien het direct bruikbare code kan geven.

- **Valkuilen:**

Het grootste probleem dat AI-tools blijven hebben vind ik dat ze veel ongevraagde code bijvoegen. Ook al doen ze dit met een goede intentie vind ik het vervelend dat ze dit blijven doen. Verder geven ze ook niet altijd de meest geoptimaliseerde code. Ik heb een aantal keer zelf moeten aanpassingen maken aangezien ze dingen deden die overbodig waren.

5. Conclusie

Samenvattend waren zowel deepseek als Github Copilot een echte meerwaarde en hebben ze me goed geholpen. Echter, het was essentieel om de output kritisch te evalueren en aan te passen waar nodig. Dit project heeft me geleerd hoe AI een waardevolle assistent kan zijn, mits het met de nodige zorg en inzicht wordt gebruikt.
