Data Cleaning

Tijdens het voorbereiden van de dataset voor de analyse, stuitte ik op een aantal ontbrekende waarden. Om deze dataset bruikbaar te maken, heb ik de volgende stappen ondernomen.

1. Interpolatie
   Ik heb ervoor gekozen om de gaten in de temperatuurdata te vullen met lineaire interpolatie. Het klimaat verandert geleidelijk. Als ik de temperatuur van 1850 en 1852 weet, is een schatting voor 1851 op basis van die twee jaren veel logischer dan een gemiddelde van de hele eeuw te nemen. Door te interpoleren, blijven de natuurlijke schommelingen en trends in data beter zichtbaar.
2. Opschonen per land (`groupby`)
   In de code heb ik de data gegroepeerd per land zodat ik de lege cellen kon invullen. Je kunt de ontbrekende temperatuur in Canada niet invullen met gegevens uit Curacao. Door de functie `groupby('country')` te gebruiken, zorg ik ervoor dat gaten in de data van een specifiek land alleen wordt opgevuld met waarden uit hetzelfde land. Dit voorkomt dat de data onbetrouwbaar is.
3. Verwijderen van "ruis"
   Sommige rijen hadden geen landnaam, of een ongeldige datum (zoals `0000-00-00`). Deze rijen heb ik verwijderd, zonder te weten over welk land het gaat of uit welk jaar de meting komt, voegt de data niets toe aan het onderzoek en vervuilt het de uiteindelijke grafieken.
4. Bronbestanden behouden
   De opgeschoonde data heb ik opgeslagen in nieuwe tabellen in SQL (met de naam `cleaned_`)

Import & Database verbinding

```
import pandas as pd
import sqlalchemy as sa
import numpy as np

# Verbinding maken met de lokale XAMPP MariaDB database
engine = sa.create_engine("mysql+pymysql://root:@localhost/climate_watch")
print("Verbinding met database geslaagd.")
```

Cleaning

- Ik heb vijf verschillende tabellen met elk een andere structuur. In plaats van vijf keer dezelfde code te schrijven heb ik een dictionary gemaakt. Hierin vertel ik Python: als je de

stad-tabel pakt, moet je rekening houden met de kolommen `country` en `city` om de juiste locatie te bepalen.

- In de eerste regel van de `for`-loop haal ik de data op uit SQL en pas ik direct de eisen van de opdracht toe. Ik gebruik `pd.to_datetime` met `errors='coerce'` om van de datum-tekst een echt tijdsobject te maken. De 'ruis' verander ik hiermee in een leeg veld, zodat ik die rijen daarna met `dropna()` makkelijk kan verwijderen
- (`sort values`) Dit is belangrijk voor de interpolatie. Je kunt alleen een lijn trekken tussen twee punten als ze op de juiste volgorde staan. Ik sorteer de data eerst op locatie en daarna chronologisch op datum.
- `groupby` + `transform` . Ik gebruik `groupby` om de dataset op te splitsen in kleine groepjes. Binnen dat groepje pas ik `interpolate` toe. Mocht er dan aan het begin of eind van de reeks nog iets missen, gebruik ik de fallback `fillna(x.mean())` Dit vult de laatse gaten met het gemiddelde van dat specifieke land.

```
# Configuratie per tabel: welke kolommen bepalen de unieke locatie?
tables_config = {
    'global_temperatures': [],
    'global_temp_country': ['country'],
    'global_temp_state': ['country', 'state'],
    'global_temp_major_city': ['country', 'city'],
    'global_temp_city': ['country', 'city']
}

for tabel, groepering in tables_config.items():
    # Inladen en datum filteren
    df = pd.read_sql(f"SELECT * FROM {tabel}", engine)
    df['dt'] = pd.to_datetime(df['dt'], errors='coerce')
    df_subset = df[df['dt'].dt.year >= 1980].dropna(subset=['dt']).copy()

    # Sorteren voor tijdreeks-analyse
    sort_cols = groepering + ['dt'] if groepering else ['dt']
    df_subset = df_subset.sort_values(by=sort_cols)

    # Numerieke kolommen identificeren
    target_cols = [c for c in df_subset.select_dtypes(include=
['number']).columns if c not in ['id', 'latitude', 'longitude']]

    # Interpolatie toepassen (per groep indien van toepassing)
    if groepering:
        for col in target_cols:
            df_subset[col] = df_subset.groupby(groepering)[col].transform(
                lambda x: x.interpolate(method='linear',
limit_direction='both')
```

```
            )
            # Fallback: vul overgebleven gaten met het groepsgemiddelde
            df_subset[col] = df_subset.groupby(groepering)
[col].transform(lambda x: x.fillna(x.mean()))
    else:
        df_subset[target_cols] =
df_subset[target_cols].interpolate(method='linear', limit_direction='both')

    # Opslaan naar SQL
    df_subset.to_sql(f"cleaned_{tabel}", engine, if_exists='replace',
index=False)
    print(f"Tabel {tabel} succesvol verwerkt en opgeslagen.")
```

4 Opdrachten

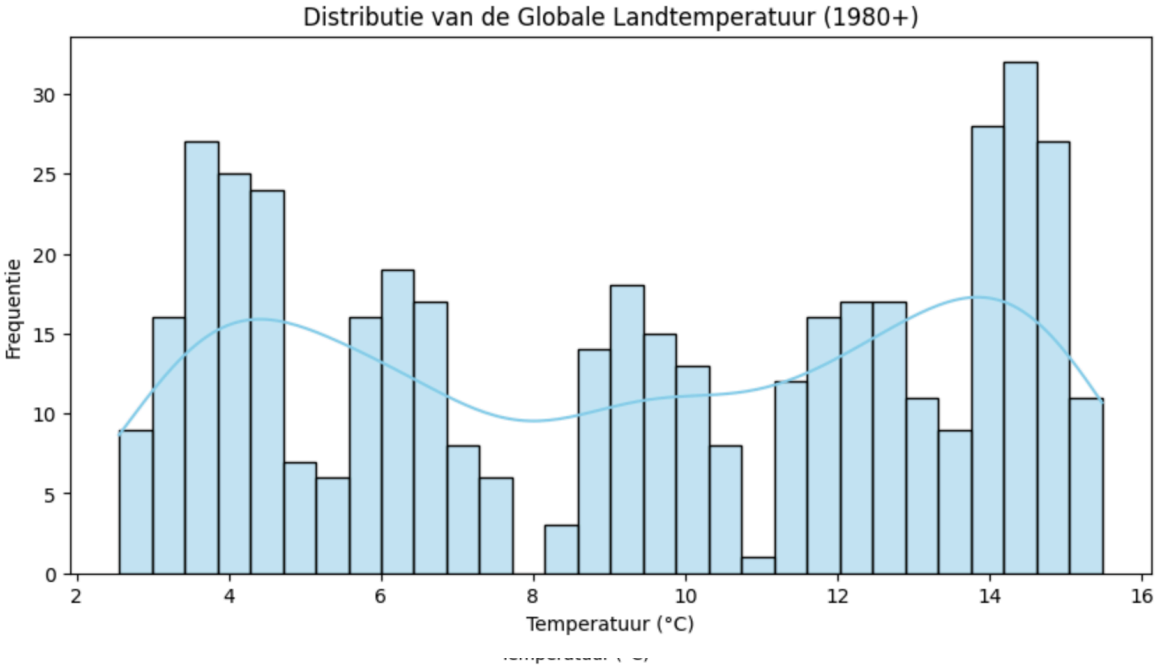
1. Exploration

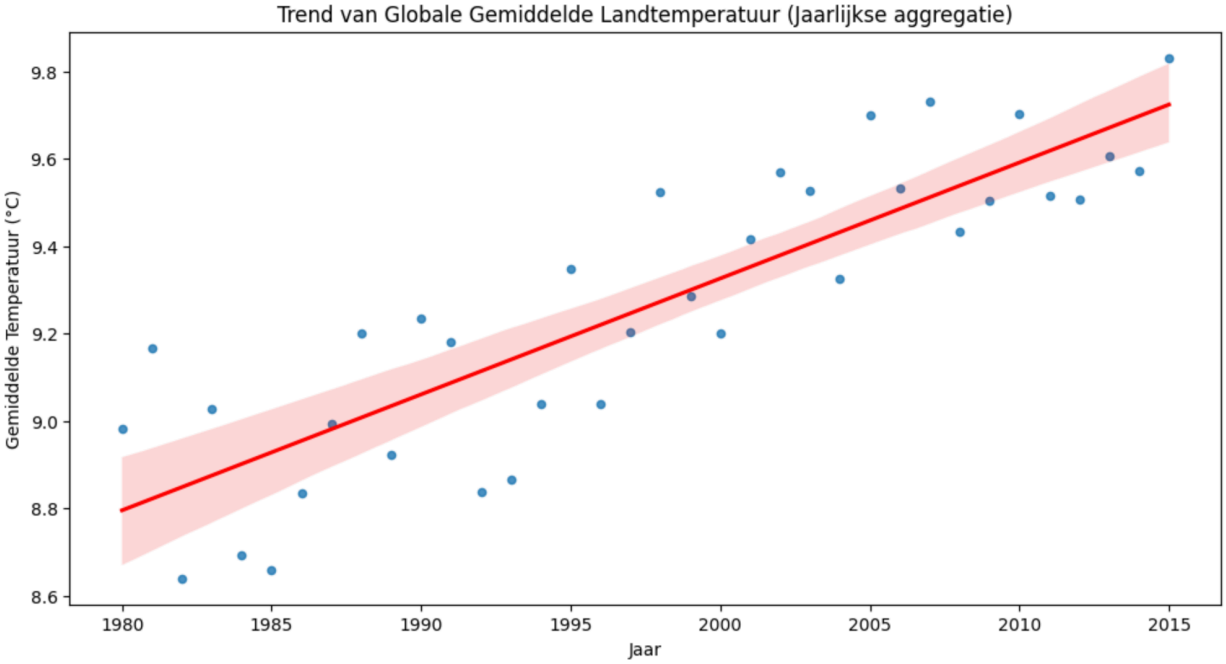
```
# Inladen van de opgeschoonde wereldwijde data
df_global_clean = pd.read_sql("SELECT * FROM cleaned_global_temperatures",
engine)

# Gebruik van describe() voor de algemene statistieken (Opdracht 4.1)
# Focus op de belangrijkste temperatuur-kolommen
stats = df_global_clean[['LandAverageTemperature', 'LandMaxTemperature',
'LandMinTemperature', 'LandAndOceanAverageTemperature']].describe()

# Toevoegen van de Mediaan (median) omdat describe() deze niet standaard
toont (behalve als 50%)
median = df_global_clean[['LandAverageTemperature', 'LandMaxTemperature',
'LandMinTemperature', 'LandAndOceanAverageTemperature']].median()
stats.loc['median'] = median

display(stats)
```

	LandAverageTemperature	LandMaxTemperature	LandMinTemperature	LandAndOceanAverageTemperature
count	432.000000	432.000000	432.000000	432.000000
mean	9.259451	14.964331	3.644563	15.655507
std	4.138503	4.252582	4.012910	1.229815
min	2.558000	8.090000	-2.853000	13.566000
25%	5.217000	10.710500	-0.296500	14.443500
50%	9.480500	15.207500	3.762000	15.732000
75%	13.224000	19.076250	7.500500	16.814250
max	15.482000	21.320000	9.715000	17.611000
median	9.480500	15.207500	3.762000	15.732000

2. Distributie van temperatuurwaarden over de jaren.
3. Trend in global average land temperatures over de tijd

```
import matplotlib.pyplot as plt
import seaborn as sns

# Zorg dat 'dt' een datetime object is
df_global_clean['dt'] = pd.to_datetime(df_global_clean['dt'])

# 1. Distributie van de temperatuur (Histogram)
plt.figure(figsize=(10, 5))
sns.histplot(df_global_clean['LandAverageTemperature'], bins=30, kde=True,
color='skyblue')
plt.title('Distributie van de Globale Landtemperatuur (1980+)')
plt.xlabel('Temperatuur (°C)')
plt.ylabel('Frequentie')
plt.show()

# 2. Jaarlijkse groepering (Opdracht 4.2)
# Groepering per jaar om de seizoensinvloeden te middelen
df_yearly = df_global_clean.groupby(df_global_clean['dt'].dt.year)
['LandAverageTemperature'].mean().reset_index()

# 3. Visualisatie van de trend (Opdracht 4.3)
plt.figure(figsize=(12, 6))
sns.regplot(data=df_yearly, x='dt', y='LandAverageTemperature', scatter_kws=
{'s':20}, line_kws={'color':'red'})
plt.title('Trend van Globale Gemiddelde Landtemperatuur (Jaarlijkse
aggregatie)')
plt.xlabel('Jaar')
plt.ylabel('Gemiddelde Temperatuur (°C)')
plt.show()
```


Distributie van de Globale Landtemperatuur (1980+)
Frequentie
Temperatuur (°C)
0
5
10
15
20
25
30
2
4
6
8
10
12
14
16


Trend van Globale Gemiddelde Landtemperatuur (Jaarlijkse aggregatie)
Gemiddelde Temperatuur (°C)
Jaar
8.6
8.8
9.0
9.2
9.4
9.6
9.8
1980
1985
1990
1995
2000
2005
2010
2015