Inhoud

Opdrachten Dag 1

Opdracht 1.1 - Functie Beschrijvende Statistiek

Van een reeks getallen kunnen een aantal beschrijvende kenmerken worden afgeleid. Bijvoorbeeld het aantal getallen, het hoogste getal, het laagste en de gemiddelde van de getallen.

Stap 1: Maak een lijst met willekeurige gehele getallen.
Tip: **getallen = [55, 86, 98, 13, 51, 46, 9, 60, 22, 69, 56, 17, 29, 86, 26, 88]**

Stap 2: Maak een functie, bv. analyseer(), die als argument een lijst aanneemt.
Tip: **def analyseer(getallen):**

Stap 3: Maak een lege dictionary. Deze wordt in de volgende stap gevuld met verschillende statistische kenmerken.
Tip: **d = {}**

Stap 4: Bepaal in de functie de volgende berekeningen en neemt de berekende waarden op in de dictionary.
- aantal: Het aantal getallen
- minimum: Het laagste getal
- maximum: Het hoogste getal
- gemiddelde: Het gemiddelde van de getallen
- mediaan: Het middelste getal als de getallen gesorteerd zijn.
- eerste_kwartiel: Het getal op een vierde van de gesorteerde getallen
- derde_kwartiel: Het getal op drie vierde van de gesorteerde getallen

Tip: b.v. **d['aantal'] = len(getallen)**

Stap 5: Geef de dictionary terug als uitkomst van de functie.
Tip: **return d**

Stap 6: Test de functie door de functie aan te roepen de lijst van willekeurige getallen en het resultaat te printen.

★ Stap 7: Bepaal ook de interkwartielafstand: IQR = Q3 - Q1. Voeg dit in de functie ook toe als kenmerk aan de dictionary.

★ Stap 8: Op basis van het interkwartielafstand kunnen de grenzen voor outliers worden bepaald. Outliers liggen 1.5 * IQR beneden het eerste kwartiel en boven het derde kwartiel. Voeg ook deze grenzen toe aan de kenmerken die functie bepaald.

★ Stap 9: Geef ook een list van outliers terug.

Opdracht 1.2 - Een CSV bestand inlezen

Er is een CSV bestand klaar gezet in de directory datasets: ca-500.csv. Dit bestand bevat fictieve persoonsgegevens van 500 mensen die in Canada wonen. Maak gebruik van het csv bibliotheek om dit bestand te lezen.

Stap 1: Import de bibliotheek csv.
Tip: **import csv**.

Stap 1: Open het bestand met de functie open(). Maak ook gebruik van een context manager d.m.v. de keyword **with**.
Tip:
filename = '../datasets/ca-500.csv'
with open(filename) as f:

Stap 2: Instantieer een DictReader op basis van de file.
Tip: **reader = csv.DictReader(f)**

Stap 3: Print om te kijken of dat goed is gegaan de namen van de kolommen.
Tip: **print(reader.fieldnames)**

Stap 4: Loop nu in een lus door alle regels van het bestand.
Tip: **for row in reader:**

Stap 5: Print in de lus alleen de velden first_name, last_name, city en email.
Tip: **print(row['first_name'], row['last_name'], row['city'], row['email'])**

Stap 6: Maak hier een mooi lijstje van.
Tip: **print('{:20} {:20} {:20} {}'.format(…))**

Stap 7: Print nu alleen de mensen die in Montreal wonen.
Tip: **if row['city'] == 'Montreal':**

★ Stap 8: Print een overzicht met het aantal mensen per stad.
Tip:
creëer een lege dictionary **cities = {}**
Tel in een loop het aantal mensen met**:**
cities[row['city']] = cities.get(row['city'], 0) + 1

★ Stap 9: Print een overzicht met het aantal mensen per email provider/domain
Tip: Vergelijkbaar als de vorige stap maar haal de provider uit het e-mail adres met string methods: **p = row['email'].find('@')** en **provider = row['email'][p+1:]**

Opdracht 1.3 - Data ophalen van een API

Er zijn heel veel gegevens op het internet beschikbaar via zogenaamde API's. Ook weergegevens. Deze zijn bijvoorbeeld op te halen van **openweathermap.org**. Check hiervoor de API-documentatie op de site https://openweathermap.org/current. In deze opdracht geven we antwoord op de vraag: "Wat is de temperatuur in Amsterdam op dit moment?".

Stap 1: Voor het bevragen van openweathermap.org is een apikey nodig. Dit zou u zelf gratis kunnen aanvragen maar u kan ook gebruik maken van de key: **d1526a9039658a6f76950cff21823aff**.

Stap 2: Stel het URL samen.
Tip:
url = 'http://api.openweathermap.org/data/2.5/weather'
url += '?appid=d1526a9039658a6f76950cff21823aff'

Stap 3: Geef in de querystring ook de gewenste units mee.
Tip: **url += '&units=metric'**

Step 4: Voeg eventueel nog meer opties toe aan het URL.
Tip: **url += '&mode=json** en **&lang=nl'**

Step 5: Geef tenslotte een plaats in de wereld op.
Tip: **url += '&q=Amsterdam'**

Stap 6: Maak gebruik van de bibliotheek requests.
Tip: **import requests** en **response = requests.get(url)**

Stap 7: De response bevat JSON. Dit is een veel gebruikte standaard voor het verzenden van data. JSON kan worden gedecodeerd met de methode json(). Het resultaat is een Python datastructuur die bestaat uit dictionaries en lists.
Tip: **data = response.json()**

Stap 7: Gebruik tenslotte list en dictionary indexing om de temperatuur op te halen.
Tip: **data['main']['temp']**

★ Stap 8: Print het resultaat. Bijvoorbeeld: Het is 9 graden in Amsterdam.

★ Stap 9: Haal ook de beschrijving uit het ontvangen data en voeg dat toe aan de uitvoer.
Tip: **weather = d['weather'][0]['description']**

Opdrachten Dag 2

Opdracht 2.1 - Random data genereren

Het genereren van random data kan nuttig zijn als test data. Numpy biedt hiervoor verschillende mogelijkheden.

Stap 1: Importeer numpy met alias np: **import numpy as np**

Stap 2: Genereer een numpy array met 10 willekeurige gehele getalen tussen 1 en 100.
Tip: Gebruik de functie **np.random.randint(1, 100, 10)**

Stap 3: Genereer een numpy array met 1000 willekeurige getallen met een normale verdeling rond gemiddelde van 10 en een spreiding van 5.
Tip: **s = np.random.randn(1000) * 5 + 10**
of **s = np.random.normal(10, 5, 1000)**

Stap 4: Maak een histogram van de gegenereerde getallen.
Tip:
Importeer hiervoor matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
en gebruik:
plt.hist(s)

Stap 5: Pas het aantal bins aan naar 20 en maak opnieuw een histogram.
Tip: **plt.hist(s, bins=20)**

Stap 6: De bin grenzen kunnen ook worden bepaald door het meegeven van een lijst van waarden. Maak een lijst met waarden van -10 tot 40 met een stapgrootte van 2 en gebruik dit om de bins in de histogram te bepalen.
Tip: **bins = np.arange(-20, 42, 2)** en **plt.hist(s, bins=bins)**

★ Stap 7: Maak een histogram van een uniforme verdeling van waarden tussen 0 en 10.
Tip: Een uniforme verdeling met **s =np.random.uniform(0, 10, 10000)**

★ Stap 8: Maak een binominale verdeling van het aantal keer kop wordt gegooid als er met 6 munten wordt gegooid met een 50-50 kans op kop of munt en dit 1000 keer wordt herhaald. Maak hier een staafdiagram van.
Tip: **s = np.random.binomial(6, 0.5, 1000)**
en **values, counts = np.unique(s, return_counts=True)**
en **plt.bar(values, counts)**

Opdracht 2.2 - Ruis toevoegen aan een model

Soms is het handig om ruis aan een functie toe te voegen. Bij voorbeeld om test data te genereren voor machine learning algoritmes. In deze opdracht definiëren we eerst een model in de vorm van een functie. In Python kan dit goed met een lambda expressie. Vervolgens worden random input waarden gegenereerd waarvan de uitvoerwaarden van het model worden bepaald.

Stap 1: Importeer numpy en matplotlib.
Tip: **import numpy as np** en **import matplotlib.pyplot as plt**

Stap 2: Definieer de vergelijking van het model. Een lineaire functie heeft bijvoorbeeld de vergelijking y= a.x + b.
Tip: definieer twee variabelen **a = 2** en **b = 5** en vervolgens de functie
model = lambda x: a * x + b

Stap 3: Genereer nu een aantal waarden x-waarden. Bijvoorbeeld 80. Kies hiervoor een laagste en hoogste waarde voor x. Bijvoorbeeld 0 en 10. Bepaal de bijbehorende y-waarden van het model.
Tip: **xmin = 0**, **x_max = 10** en **n = 80**
x_model = np.linspace(x_min, x_max, n)
y_model = model(x_model)

Stap 4: Voor de gegenereerde data willen we wat minder evenredig verdeelde x-waarden. Definieer hiervoor een aantal (bv. weer 100) uniform verdeelde x-waarden tussen dezelfde grenzen 0 en 10.
Tip: en **x_data = np.random.uniform(x_min, x_max, n)**

Stap 5: Genereer nu de ruis. Dit zijn net zoveel normaal verdeelde waarden met een gemiddelde van 0 en een te kiezen spreiding (bv. 3)
Tip: **spead = 3** en **ruis = np.random.randn(n) * spread**

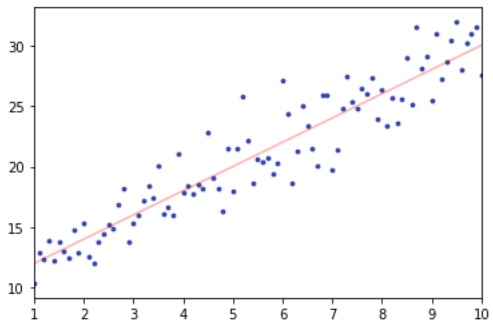
Stap 6: Maak nu een array met de uitgangswaarden. Dit is de uitgangswaarde van het model voor een bepaalde x plus ruis.
Tip: **y_data = model(x_data) + ruis**

Stap 7: Plot nu zowel het model als de gegenereerde data in één plot. Gebruik voor het weergeven van de gegenereerde data een scatterplot.
Tip: **plt.plot(x_model, y_model, color='red', linewidth=1, alpha=0.8)**
en **plt.scatter(x_data, y_data)**

★ Stap 8: Herhaal bovenstaande stappen voor andere modellen.
Tip: sinusvormig: model = **lambda x: a * np.sin(c*x) + b**
kwadratisch: model = **lambda x: a*x**2 + b*x + c**
exponetieel: model = **lambda x: a*b**x + c**

Opdracht 2.3 - Lineaire regressie

Lineaire regressie vormt de basis van vele statistische modellen. We gebruiken hiervoor de fictieve data die u zelf in de vorige opdracht heeft gegenereerd. Bepaal de coëfficiënten van de lineaire regressie lijn door een verzameling punten. Anders gezegd: bepaal de coëfficiënten (a en b) van de lijn die het best past bij de punten met vergelijking y = a.x + b.



Stap 1: Herhaal de stappen 1 t/m 7 van de vorige oefening.

Stap 2: Numpy heeft een functie **polyfit** waarmee u een best-fit polynoom kunt bepalen. Gebruik deze functie met de x en y waarden als argumenten. Door 1 als graad op te geven wordt een rechte lijn gebruikt. De return waarde is een tuple met de coördinaten van de vergelijking.
Tip: **coefs = np.polyfit(x_data, y_data, 1)**

Stap 3: Plot ook deze lijn nu in de plot.
Tip: **regression = np.poly1d(coefs)**
en **plt.plot(x_model, regression (x_model))**

Stap 4: Varieer met het originele model, het aantal data punten, de hoeveelheid ruis en met de graad van het regressie model en bestudeer het resultaat.

★ Stap 5: SciPy heeft ook een methode om lineaire regressie uit te voeren en de coördinaten van het lineaire regressie model te vinden. Gebruik deze functie en plot het resultaat. Verifieer dat het hetzelfde is het resultaat van de vorige stap.
Tip: **slope, intercept, *_ = scipy.stats.linregress(x_data, y_data)**
en **regression = lambda x: x * slope + intercept**

★ Stap 6: SciPy heeft tevens allerlei methoden beschikbaar om allerlei soorten optimalisatie problemen aan te kunnen pakken. Lineaire regressie kan ook gezien worden als het minimaliseren van de afwijkingen ten opzichte van een lineair model. Hiervoor kan bijvoorbeeld de methode optimize.curve_fit() worden gebruikt. Deze methode kan ook worden gebruikt als het model niet lineair is. Bepaal met deze methode ook de coëfficiënten van het lineaire model.
Tip: **model_to_optimize = lambda x, a, b: a * x + b**
parameters, *_ = optimize.curve_fit(model_to_optimize, x_data, y_data)
en **slope, intercept = parameters**

Opdrachten Dag 3

Opdracht 3.1 - Pandas

Deel 1: Lees data vanuit een CSV bestand naar een dataframe

Stap 1: Importeer pandas
Tip: **import pandas as pd**

Stap 2: Lees het ca-500.csv bestand dat we eerder hebben gebruikt met de functie read_csv.
Tip: **df = pd.read_csv('../datasets/ca-500.csv')**

Stap 3: Exploreer de dataframe. Toon de eerste 5 rijen? Hoeveel kolommen en rijen zijn er? Wat zijn de data types van de kolommen?

Deel 2: Selecteer kolommen uit een dataframe

Stap 1: Selecteer de kolommen 'first_name', 'last_name', 'city' en 'email'.
Tip: **df[['first_name', 'last_name', 'city', 'email']]**

Deel 3: Maak een nieuw kolom in een dataframe

Stap 1: Maak een kolom 'name' als een samenvoeging van de 'first_name' en de 'last_name'
Tip: **df['name'] = df['first_name'] + ' ' + df['last_name']**

Deel 4 - Filter rows

Stap 1: Selecteer alleen de rijen van mensen die in Montreal wonen
Tip: **df[df['city'] == 'Montreal']**

Stap 2: Selecteer alleen de rijen van mensen die in Montreal of Vancouver wonen
Tip: **df[df['city'].isin(('Montreal', 'Vancouver'))]**
of: **df[(df['city']=='Montreal') | (df['city']=='Vancouver')]**

Deel 5 - Sorteer de rijen

Stap 1: Sorteer de rijen op last name
Tip: **df.sort_values(by='last_name')**

Stap 2: Sorteerd op city en dan op last name
Tip: **df.sort_values(by=['city', 'last_name'])**

Deel 6 - Data Cleaning

Stap 1: Zoek regels zonder email adres
Tip: **df[df['email'].isna()]**

Stap 2: Verwijder de regels zonder email adres uit de dataframe
Tip: **df.dropna(subset=['email'], inplace = True)**

Stap 3: Controleer of er rijen voorkomen die identiek zijn
Tip: **df[df.duplicated()]**

Stap 4: Controleer of er personen zijn die vaker voorkomen. Controleer de betrokken rijen.
Tip: **df.loc[df.duplicated(['last_name', 'first_name','email'])]**
en **df[df['last_name'].isin(duplicates['last_name'])] .sort_values('last_name')**

Stap 5: Ontdubbel de dataframe door het terugbrengen van deze rijen met dubbele gegevens tot één rij. Behoud de laatste rij.
Tip: **df.drop_duplicates(['last_name', 'first_name','email'], keep = 'last', inplace=True)**

Deel 7 - Aggregatie

Stap 1: Toon een gesorteerde lijst van steden die in het bestand voorkomen. Hoeveel verschillende steden komen in het bestand voor?
Tip: **sorted(df['city'].unique())**

Stap 2: Geef een lijst van steden met daarin hoeveel mensen er wonen. Wat zijn de top 10 steden met de meeste mensen?
Tip: **df['city'].value_counts().head(10)**

Stap 3: Geef een lijst van provincies met daarbij het aantal mensen die in de provincie wonen.
Tip: **df[' province'].value_counts()**

Stap 4: En maak hiervan een (horizontale) bar plot.
Tip: **df['province'].value_counts(ascending=True).plot(kind='barh')**

Opdracht 3.2 - KNMI daggegevens

De KNMI stelt vele datasets beschikbaar m.b.t. het weer en klimaat in Nederland. Onder andere biedt de KNMI gegevens over het weer per dag van 35 verschillende weerstations stations op https://www.knmi.nl/nederland-nu/klimatologie/daggegevens. Vergelijk de temperatuurgegevens van het afgelopen jaar van twee verschillende weerstations in Nederland. Waar is het kouder? Laat met pandas zien of tussen deze stations sprake is van een temperatuurverschil.

Stap 1: Importeer pandas, numpy en matplotlib zoals al eerder gedaan.

Stap 2: Ga naar de bovengenoemde site en download de gegevens van twee verschillende stations. Het is handig om twee station te kiezen die enigszins uit elkaar liggen. Pak de verkregen zip-bestanden uit en plaats de bestanden in de datasets directory. Bijvoorbeeld: De Bilt (260) en Hoogeveen (279).

Stap 3: Bekijk de bestanden met een teksteditor. Bestudeer de opbouw van het bestand. Op welke regel beginnen de daggegevens? Op welke rij staan de kolomheaders?

Stap 4: Begin met het opgeven de bestandsnamen.
Tip: **filename1 = '../datasets/etmgeg_260.txt'**
en **filename2 = '../datasets/etmgeg_279.txt'**

Stap 5: Maak een list met de kolomnamen. Kopieer hiervoor de regel uit een van de bestanden met de kolomnamen en plak dit in een string. Gebruik vervolgens split om hier een list van te maken.
Tip: **names = 'STN,YYYYMMDD,DDVEC,FHVEC, FG, FHX, FHXH, FHN, FHNH, FXX, FXXH, TG, TN, TNH, TX, TXH, T10N,T10NH, SQ, SP, Q, DR, RH, RHX, RHXH, PG, PX, PXH, PN, PNH, VVN, VVNH, VVX, VVXH, NG, UG, UX, UXH, UN, UNH, EV24'**
en **column_names = names.replace(' ', '').split(',')**

Stap 6: Lees de gegevens in twee Pandas dataframes met read_csv(). Gebruik het skip argument om de regels aan het begin van het bestand over te slaan en gebruik de names list om de kolom namen aan te geven.
Tip: **df1 = pd.read_csv(filename1, skiprows=49, header=None, names= column_names, dtype=str)**
en df2 net zo met filename2.

Stap 7: Maak een selectie van de gewenste rijen op basis van het jaar (bv. 2020). Selecteer enkele kolommen om mee te werken.
Tip:
year = 2020
columns_of_interest = ['STN', 'YYYYMMDD', 'TG', 'TN', 'TX']
df1_selection = df1.loc[df1['YYYYMMDD'].str.startswith(str(year)), columns_of_interest]
df2_selection = df2.loc[df2['YYYYMMDD'].str.startswith(str(year)), columns_of_interest]

Stap 8: En voeg beide dataframes onder elkaar samen met **concat()**.
Tip: **df = pd.concat([df1_selection, df2_selection])**

Stap 9: De temperatuurwaarden zijn gegeven in 0.1 graden celcius. Converteer de temperaturen naar numerieke waarden en vervolgens naar graden celcius door de waarden te delen door 10. Plaats in nieuwe kolomen.
Tip: **df[['GEMIDDELDE','MINIMUM','MAXIMUM']] = df[['TG','TN','TX']].astype('int') * 0.1**

Stap 10: Converteer het datumveld YYYYMMDD naar een datetime waarde in een nieuw kolom, bv. DATUM. Gebruik hiervoor pd.to_datetime().
Tip: **df['DATUM'] = pd.to_datetime(df['YYYYMMDD'], format='%Y%m%d')**

Stap 11: Converteer het STN kolom naar de stationnamen. Gebruik hiervoor de methode map met een dict.
Tip:
df['STATION'] = df['STN'].astype('int').map({260: 'De Bilt', 279: 'Hoogeveen'})

Stap 12: Drop de oorspronkelijke kolommen. Die zijn niet meer nodig.
Tip:
df = df[['DATUM', 'STATION', 'GEMIDDELDE', 'MINIMUM', 'MAXIMUM']]

Stap 13: Unstack de dataframe naar een vorm met de datum als rij index en de verschillende temperatuurmetingen in de kolommen zoals hieronder weergegeven.

	GEMIDDELDE		MINIMUM		MAXIMUM	
STATION	De Bilt	Hoogeveen	De Bilt	Hoogeveen	De Bilt	Hoogeveen
DATUM						
2020-01-01	0.8	0.2	-0.2	-0.7	1.8	1.8
2020-01-02	3.9	1.8	1.3	-0.8	7.4	6.0
2020-01-03	7.6	6.7	4.3	3.3	9.9	9.1
2020-01-04	6.7	5.3	4.0	1.8	8.1	8.1
2020-01-05	6.9	5.4	5.9	1.2	7.6	7.0
...	...	...	...	...	...	...
2020-12-27	5.0	4.4	3.2	3.4	5.9	5.3
2020-12-28	3.6	3.5	1.3	1.4	5.4	5.8
2020-12-29	3.7	3.2	2.5	2.3	5.8	4.2
2020-12-30	4.2	3.1	2.4	1.2	6.8	5.8
2020-12-31	2.1	1.9	-2.1	-1.6	4.0	3.8

366 rows × 6 columns

Tip: Zet de index eerst op DATUM en STATION en voer daarop **unstack()** uit. Maak hiervan weer een nieuwe dataframe.
df_unstacked = df.set_index(['DATUM', 'STATION']).unstack()
Tip: Dit laatste kan ook met **pivot()** worden bereikt.
df_unstacked = df.pivot(index = 'DATUM', columns = 'STATION', values = ['GEMIDDELDE','MINIMUM','MAXIMUM'])

Stap 14: Plot de beide temperaturen in één plot. Is er een conclusie te trekken?
Tip: **df_ unstacked[('GEMIDDELDE','De Bilt')].plot()**
en **df_ unstacked[(''GEMIDDELDE'','Hoogeveen')].plot()**

Step 15: Toon de gemiddelden van de kolommen met **df_unstacked.describe()**

Stap 16: Maak boxplots van de kolommen. Kan hieruit het een en ander worden afgeleid m.b.t. het temperatuur verschil?
Tip: **df_unstacked.boxplot()**

Stap 17: Maak een scatterplot van de twee temperaturen en plot ook een lijn waarbij de temperaturen identiek zouden zijn. Is er een conclusie te trekken?
Tip:
plt.scatter(df_unstacked[('GEMIDDELDE', 'De Bilt')], df_unstacked[('GEMIDDELDE', 'Hoogeveen')])
en **plt.plot([0, 28], [0, 28], color='r')**

Stap 18: Maak een extra series met het verschil tussen de temperaturen.
Tip: **diff = df_unstacked[('GEMIDDELDE','Hoogeveen')] - df_unstacked[('GEMIDDELDE', 'De Bilt')]**

Stap 19: Toon een histogram van het temperatuurverschil. Is er een conclusie te trekken?
Tip: **diff.hist(bins=20)**

Stap 20: Plot de temperatuurverschillen. Plot ook een nullijn.
Tip: **diff.plot()** en **plt.axhline(0, color='red', linewidth=1)**

Stap 21: Bepaal het aantal dagen dat het kouder is in Hoogeveen dan in De Bilt.
Tip: **n_kouder = np.sum(diff<0)**

★ Stap 22: Tot nu toe trekken we conclusies op basis van visualisaties. Met een statistische test kan een conclusie met een zeker waarschijnlijkheid worden aangenomen of verworpen. Hiervoor wordt een Student t-test. Als de p-waarde kleiner is dan 0.01 dan kan met 99% zekerheid worden gezegd dat het kouder is in Hoogeveen.
Tip:
from scipy import stats
t, p = stats.ttest_1samp(diff, 0)

Nu we toch temperatuurgegevens van De Bilt hebben vanaf 1900 tot nu kunnen we ook kijken naar de temperatuurstijging in de loop der jaren.

★ Stap 23 Lees hiervoor opnieuw de gegevens van De Bilt in en converteer een naar bruikbare dataframe. Herhaal hiervoor dus stappen 5, 6 en 7.
Tip:
data = pd.read_csv(filename1, …)
df = data[['YYYYMMDD', 'TG']]
df[['GEMIDDELDE']] = df[['TG']].astype('int') * 0.1
df['DATUM'] = pd.to_datetime(df['YYYYMMDD'], format='%Y%m%d')

★ Stap 24: Zet de datum als index en selecteer alleen de rijen tussen 1-1-1900 en het einde van vorig jaar.
Tip:
df.set_index('DATUM', inplace=True)
df = df.truncate(before='1900-01-01', after='2020-12-31')[['GEMIDDELDE']]

★ Stap 25: Pandas kan goed met timeseries omgaan. Zo kan een timeserie eenvoudig nnaar een andere tijdsperiode worden omgezet. Zet de daggegevens om in jaargemiddelden.
Tip: **yearly = df.resample('Y').mean()**

★ Stap 26: Plot de jaargemiddelden.
Tip: **plt.plot(yearly['TG'])**

★ Stap 27: Tenslotte kunnen we op basis van de gegevens een best-fit curve bepalen en ook in de plot weergeven.
Tip:
coefs = np.polyfit(yearly.index.year, yearly['GEMIDDELDE'], 5)
model = np.poly1d(coefs)
plt.plot(yearly['GEMIDDELDE'])
plt.plot(yearly.index, model(yearly.index.year))

Workshop Bring Your Own Data - Deel 1

In deze training wordt gewerkt met een workshop waarin u aan de gang gaan met uw eigen data. Mocht u geen eigen data hebben zoek dan op internet naar een dataset waarmee uw wilt werken.

In de workshop worden de fases van de Data Science Pipeline gevolgd:

1. Data ingestion
2. Data exploratie
3. Data cleaning
4. Data visualisatie
5. Data analyse / modeling

Begin met het formuleren van een vraag die uw op basis van de data tracht te beantwoorden.

In het eerste deel van de workshop voert u de stappen die vandaag aan de orde zijn geweest uit op uw eigen data.

Stap 1: Voer de benodigde stappen uit om de data in een Pandas dataframe te krijgen.

Stap 2: Verken de data. Geef een overzicht weer van het aantal en de types van de variabelen. Geef het aantal observaties weer en eventueel wat beschrijvende statistiek.

Stap 3: Onderzoek of er sprake is van ontbrekende gegevens. Neem als dit het geval is de nodige maatregelen om de data op te schonen.

Stap 4: Doe hetzelfde voor eventuele duplicaten.

Stap 5: Doe ook hetzelfde voor eventuele outliers.

Opdrachten Dag 4

Opdracht 4.1 - Matplotlib gallery

Op de site van matplotlib is er een gallery met allerlei voorbeelden. De URL hiervan is https://matplotlib.org/stable/gallery.

Stap 1: Ga naar de site en onderzoek de verschillende soorten plots.

Stap 2: Kies een plot die je interessant vind. Klik op de plot zodat de achterliggende python code wordt getoond. Kopieer de code naar een notebook en produceer hetzelfde plot.

Opdracht 4.2 - Verschillende plots maken

Op basis van het iris dataset kunnen allerlei plots worden gemaakt om de data te visualiseren.

Stap 1: Lees het iris dataset in een dataframe. Het bestand **iris.csv** is te vinden in de datasets directory.
Tip:
filename = '../datasets/iris.csv'
df = pd.read_csv(filename)

Stap 2: Exploreer de dataset. Hoeveel rijen bevat de dataset? Hoeveel kolommen? Wat voor datatypes hebben de kolommen?

Stap 3: Welke waarden zijn er in de 'species' kolom? Hoeveel rijen zijn er van ieder 'species'?
Tip: **df['species'].value_counts()**

Stap 4: Geef in een barplot het aantal van de verschillende species weer. Doe dit zowel verticaal als horizontaal. Plaats een titel bij de plot en labels bij de beide assen. Pas de plots verder aan naar eigen inzicht. Specificeer onder andere de kleur van de drie verschillende bars.
Tip: **species_count.plot.bar()** en **species_count.plot.barh()**

Stap 5: Maak een plot met vier subplots naast elkaar. Plot in de subplots histogrammen van de kolommen 'sepal_length','sepal_width','petal_length' en 'petal_width'. Plaats bij ieder subplot een overeenkomende title.
Tip: **fig, axs = plt.subplots(1, 4)** en **axs[0].hist(df['sepal_length'])**

Stap 6: Maak van dezelfde kolommen nu een boxplot. En ook een violinplot.
Tip: **df.boxplot()**
of **plt.violinplot([df['sepal_length'], …])**

Stap 7: Geef van de numerieke kolommen ook de beschrijvende statistiek en kijk of dit overeenkomt met de voorgaande boxplots.
Tip: **df.describe()**

Stap 8: Maak scatterplot van de 'sepal_length' versus de 'pedal_width'.
Tip: **df.plot.scatter(x='sepal_length', y='petal_width')**

Seaborn is een bibliotheek met allerlei voorgedefinieerde plots voor data analyse. Vaak makkelijker dan met MatPlotLib. Voorbeelden zijn te vinden in de examples gallery https://seaborn.pydata.org/examples/.

★ Stap 9: Importeer seaborn met **import seaborn as sns**

★ Stap 10: Maak met Seaborn een scatterplot waarin de verschillende species een andere kleur hebben.
Tip: **sns.lmplot(x='…', y='…', hue='species', fit_reg=False, data=df)**

★ Stap 11: Maak met Seaborn een pairplot waarin de scatterplots van iedere combinatie van de kolommen wordt weergeggeven.
Tip: **sns.pairplot(data=df, hue='species')**

Opdracht 4.3 - Geopandas

Geografisch gerelateerde data kan worden weergeven op een kaart met de package geopandas.

Stap 1: Installeer de package geopandas als dit nog niet is gedaan met **conda install geopandas** of **pip install geopandas**. Installeer ook de package descartes.

Stap 2: Geopandas bevat standaard een wereld kaart met een aantal kengetallen. Volg enkele stappen op https://geopandas.org/mapping.html om een wereldkaart te tonen en om hiermee te laten zien dat geopandas correct werkt.

Stap 3: Er zijn vele shape bestanden beschikbaar die door geopandas kunnen worden gebruikt om kaarten weer te geven. Het CBS heeft bijvoorbeeld een bestand voor buurten, wijken en gemeentes die te vinden is op https://www.cbs.nl/nl-nl/dossier/nederland-regionaal/geografische-data/wijk-en-buurtkaart-2019.
Download het bestand WijkBuurtkaart_2019_v2up en plaats dit in het datasets directory.

Stap 4: Lees de wijk- en buurtkaart in met geopandas. Dit kan ook direct uit het zip-bestand. Importeer hiervoor de package zipfile
Tips:
filename = '../datasets/WijkBuurtkaart_2019_v2up.zip'

shapefile = 'shape 2011 versie 3.0/gemeente_2019_v2up.shp'
df = geopandas.read_file('zip://' + filename + '!' + shapefile)

Stap 5: Bestudeer de gegevens die beschikbaar zijn in de dataframe. Bekijk hierbij ook de toelichting in de bijgegeven toelichting op de site.

Stap 6: Plot de kaart van Nederland. Laat hierbij de WATER regio's buiten beschouwing.
Tip: **nederland[nederland['WATER']=='NEE'].plot()**

Stap 7: Kies een grootheid en geef die in de kaart weer. Bijvoorbeeld de bevolkingsdichtheid per gemeente.
Tip:
df[df['BEV_DICHTH']>0].plot(column='BEV_DICHTH');

Stap 8: Geef ook de verhouding tussen het aantal mannen en vrouwen per gemeente weer op de kaart. Wat valt op?
Tip:
man_vrouw = nederland.loc[nederland['AANT_VROUW'] > 0,
['AANT_MAN', 'AANT_VROUW', 'geometry']]
man_vrouw['MAN_VROUW'] = man_vrouw['AANT_MAN'] /
man_vrouw['AANT_VROUW']
man_vrouw.plot(column='MAN_VROUW', legend=True);

Workshop Bring Your Own Data - Deel 2

In het tweede deel van de workshop gaat u uw eigen data visualiseren m.b.v. verschillende soorten plots. Gebruik naar eigen inzicht: histogrammen, barcharts, boxplots of scatterplots. Geografische plots zijn uiteraard ook mogelijk. Uw kunt gebruik maken Matplotlib, Seaborn of Geopandas.

Opdrachten Dag 5

Opdracht 5.1 - Data preparatie

Voordat data kan worden gebruikt voor machine learning is het noodzakelijk enkele voorbereidingen te treffen.

Stap 1: Importeer de benodigde bibliotheken:
Tip:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

Stap 2: Lees de iris dataset in een dataframe.
Tip:
filename = '../datasets/iris.csv'
df = pd.read_csv(filename)

Stap 3: Exploreer de dataframe. Hoeveel verschillende species zijn er? En hoeveel rijen per specie?

Stap 4: Splits de data in een training set en een test set.
Tip:
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(df, test_size=0.3)

Stap 5: Verdeel de data in features en een target.
Tip:
feature_names = df.columns [:-1]
target_name = df.columns[-1]
df_train_features = df_train[feature_names]
df_train_target = df_train[target_name]
df_test_features = df_test[feature_names]
df_test_target = df_test[target_name]

Stap 6: Scale de features met de standard scaler.
Tip:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df_train_features)
train_features_scaled = scaler.transform(df_train_features)
test_features_scaled = scaler.transform(df_test_features)

Stap 7: Het resultaat van de scaler is een numpy ndarray. Maak hiervan weer een dataframe. Let er op dat de indeces correct worden meegegeven. Geef de

kolommen een toevoeging '_scaled'. Doe dit voor zowel de training features als de test features.
Tip:

```
df_train_features_scaled = pd.DataFrame(
                train_features_scaled,
                columns = [col + '_scaled' for col in list(feature_names)],
                index = df_train_features.index)
```

Stap 8: Maak een Covariantie Matrix en een Correlatie Matrix van de training features. Maak hier ook een heatmap plot van.
Tip:

```
cov = df_train_features.cov()
corr = df_train_features.corr()
corr.style.background_gradient(cmap='coolwarm', axis=None).set_precision(2)
```

Stap 9: Voer een Principle Component Analyse (PCA) op de training features. Doe dit met zowel 3 als met 2 componenten. Maak hiervan ook een nieuwe dataframes net als bij stap 6. Bestudeer het resultaat met een correlatie matrix.
Tip:

```
from sklearn.decomposition import PCA
pca3 = PCA(n_components = 3)
pca3.fit(train_features_scaled)
train_features_pca3 = pca3.transform(train_features_scaled)
test_features_pca3 = pca3.transform(test_features_scaled)
```

Stap 10: Voeg nu alle kolommen samen van de training dataset samen inclusief het target column. Doe dat ook voor de test dataset.
Tip:

```
df_train = pd.concat([df_train_features,
                      df_train_features_scaled,
                      df_train_features_pca3,
                      df_train_target], axis=1)
```

Stap 11: Sla beide dataset op in een CSV bestand. Handig zo later gebruik.
Tip:

```
df_train.to_csv('../datasets/iris_training_dataset.csv')
df_test.to_csv('../datasets/iris_test_dataset.csv')
```

Opdracht 5.2 - Classificatie

Machine Learning draait vaak om het classificeren van nieuwe observaties op basis van een model die is getraind met bestaande en bekende observaties. We maken opnieuw gebruik van de iris dataset.

Stap 1: Ga verder met de voorbereidde dataset uit de vorige opdracht. Stel het eerst kolom in als index van de dataframe.
Tip:
df_train = pd.read_csv('../datasets/iris_training_dataset.csv', index_col=0)
df_test = pd.read_csv('../datasets/iris_test_dataset.csv', index_col=0)

Stap 2: Bepaal de features die voor de classificatie zullen worden gebruikt. Maak hiervan twee datasets voor zowel de training dataset als de test dataset.
Tip:
feature_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
target_name = 'species'
df_train_features = df_train[feature_names]
df_train_target = df_train[target_name]
df_test_features = df_test[feature_names]
df_test_target = df_test[target_name]

Stap 3: Maak een k-Nearest Neighbor model op basis van de training data.
Tip:
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors = 3)
knn_classifier.fit(df_train_features, df_train_target)

Stap 4: Pas het verkregen model toe op de test data.
Tip: **df_test['3NN_predicted'] = knn_classifier.predict(df_test_features)**

Stap 5: Bepaal een aantal evaluation metrics zoals de confusion matrix, accuracy, precision, recall en F1
Tip:
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(df_test['species'], df_test['3NN_predicted'])

Stap 5: Voer de stappen 3 t/m 5 uit met een ander Naïeve Bayeniaanse Classificatie algoritme.
Tip:
from sklearn.naive_bayes import GaussianNB
nb_classifier = GaussianNB()
nb_classifier.fit(df_train_features, df_train_target)
df_test['NB_predicted']= nb_classifier.predict(df_test_features)

Stap 6: Voer de stappen 3 t/m 5 uit met een ander Decision Tree Classificatie algoritme.

Stap 7: Voer de stappen 3 t/m 5 uit met een ander Support Vector Machine Classificatie algoritme.

Stap 8: Vergelijk de resultaten van de verschillende classificaties uit de vorige stappen.

Workshop Bring Your Own Data - Deel 3

In het derde deel van de workshop komt uw eerdere werk samen. Formuleer een vraag die uw op basis van de data tracht te beantwoorden. Stel een story samen waarin u uw bevindingen uit de doeken doet. Onderbouw uw bevindingen met statistiek en grafieken.

De workshop wordt afgesloten met een presentatie van de verschillende stories.

Extra Opdracht World Bank data

1. Import pandas
2. Read the data from the file **wb_gdp_cap_vs_life_exp.csv** in a dataframe
3. Explore the dataset. Which columns are available
4. Create a scatter plot of the **gdp_cap** versus the **life_exp**.
5. Add a title
   plt.title('World Development in 2007')
6. Add labels for the axis
   plt.xlabel('GDP per Capita [in USD]')
   plt.ylabel('Life Expectancy [in years]')
7. Change the x axis to a logarithmic scale with **plt.xscale('log')**
8. Add ticks on the x axis with **plt.xticks([1000,10000,100000], ['1k','10k','100k'])**
9. Map the continent column to a new '**color**' column. Use a dictionary like:
   continent_to_color_mapping = {
   'Asia':'red',
   'Europe':'limegreen',
   'Africa':'deepskyblue',
   'Americas':'gold',
   'Oceania':'black'}
   and map this to a new column with: **color =**
   df['cont'].map(continent_to_color_mapping)
10. Add color to the scatter plot with de **color** argument.
11. Set the transparency to 0.8 with the **alpha** argument.
12. Create a new '**size**' column from the 'population' column with:
    size = df['population'] * 0.000004
13. Add a grid with **plt.grid(True)**
14. Can you make any conclusions?