

# Cursus Robbe Magerman Data Science and AI

Robbe Magerman

04/06/2025

---

# Inhoudstafel

---

<b>1</b>	<b>Hoofdstuk 0: Basisgegevens</b>	<b>4</b>
<b>2</b>	<b>Hoofdstuk 1: Samples</b>	<b>5</b>
2.1	Theorie . . . . .	5
2.2	Measurement levels (NORI): . . . . .	5
2.2.1	Kwalitatief . . . . .	5
2.2.2	Quantitatief . . . . .	5
2.2.3	Kwalitatieve measurements . . . . .	5
2.2.4	Quantitatieve measurements . . . . .	5
2.3	Populatie . . . . .	5
2.4	Sample . . . . .	5
2.5	Hoe kies je een goede steekproef . . . . .	6
2.6	Sampling errors: . . . . .	6
2.7	Een dataset inlezen met basismethoden . . . . .	6
2.8	Qualitatieve en ordinale variabelen . . . . .	7
2.9	Data selecteren . . . . .	7
2.9.1	Tonen van specifieke kolommen: . . . . .	7
2.9.2	Filteren van kolommen . . . . .	8
2.9.3	Kolommen verwijderen . . . . .	8
2.9.4	Andere nuttige functies . . . . .	8
<b>3</b>	<b>Hoofdstuk 2: Analysis 1 var</b>	<b>10</b>
3.1	Measure of Central Tendency . . . . .	10
3.2	Measures of Dispersion . . . . .	10
3.3	Kenmerken Standaarddeviatie . . . . .	10
3.4	Qualitatieve variabelen . . . . .	11
3.5	Quantitative variables . . . . .	11
3.6	Centrality and Dispersion Measures . . . . .	12
3.7	Formula for Standard Deviation . . . . .	12
<b>4</b>	<b>Hoofdstuk 3: Central Limit Testing</b>	<b>14</b>
4.1	3.02 Confidence Intervals . . . . .	14
4.1.1	Z-score . . . . .	14
4.1.2	T-score . . . . .	15
4.1.3	Berekening zonder alpha . . . . .	15
4.2	3.03 Hypothesis testing . . . . .	15
4.2.1	Z-test . . . . .	15
4.2.2	Student's T-test . . . . .	19
<b>5</b>	<b>Hoofdstuk 4: Bivariate Qual</b>	<b>21</b>
5.1	Crosstab . . . . .	21
5.2	Chi-kwadraad en Cramr's V . . . . .	21
5.2.1	Data visualasations . . . . .	21
5.2.2	Chi kwadraad . . . . .	21
5.3	Goodsness of fit test . . . . .	22
5.4	Chi-squared toets voor onafhankelijkheid . . . . .	23
5.5	Cochran's rule . . . . .	23
<b>6</b>	<b>Hoofdstuk 5: Bivariate Qual Quant</b>	<b>24</b>
6.1	Boxplot . . . . .	24
6.2	Violinplot . . . . .	24
6.3	Density/kdeplot . . . . .	24
6.4	Barplot met errors . . . . .	24
6.5	2-sample t-test for independent samples . . . . .	24
6.6	The t-test for paired samples . . . . .	25
6.7	Effect Size . . . . .	25

---

6.8	Cohens D . . . . .	25
<b>7</b>	<b>Hoofdstuk 6: Regession Analysis</b>	<b>26</b>
7.1	Visualisation . . . . .	26
7.2	Method of least squares introduction . . . . .	26
7.3	Plotting Regessionline . . . . .	27
7.4	Covariantie . . . . .	27
7.5	Coeffecient of determination . . . . .	28
<b>8</b>	<b>Hoofdstuk 7: Time Series</b>	<b>29</b>
8.1	Componenten van Tijdreeksgegevens . . . . .	29
8.2	Werken met een date als index . . . . .	30
8.3	Time Series Models . . . . .	31
8.3.1	Tabellen . . . . .	31
8.3.2	Maken van voorspellingen . . . . .	31
8.4	Moving average . . . . .	31
8.4.1	Simple Moving Average . . . . .	32
8.4.2	Exponential Moving Average (EMA) of Simple Exponential Smoothing (SES) . . . . .	32
8.4.3	Double exponetional Smoothing (Holt's methode) . . . . .	33
8.4.4	Triple exponential Smoothing (Holt-Winters methode) . . . . .	33
8.5	Decomposing a Time Series . . . . .	34
8.6	Samenvatting . . . . .	34
<b>9</b>	<b>Samenvatting Data Science en AI</b>	<b>36</b>
9.1	Confidence intervals . . . . .	36
9.1.1	Confidence interval for a large samples mean . . . . .	36
9.1.2	Confidence Interval for a small samples mean . . . . .	36
9.2	Z-tests . . . . .	37
9.2.1	Right tailed Z-test . . . . .	37
9.2.2	Left tailed Z-test . . . . .	37
9.2.3	Two tailed Z-test . . . . .	38
9.3	Student's T-test . . . . .	38
9.4	Chi-squared en Cramr's V . . . . .	39
9.5	Goodness of fit test . . . . .	39
9.6	T-test for independent variables . . . . .	40
9.7	T-test for paired samples . . . . .	41
9.8	Cohen's d . . . . .	41
9.9	Simple Moving Avarage . . . . .	41
9.10	Exponential Moving Avarage . . . . .	42
9.11	Double exponential smoothing . . . . .	42
9.12	Triple Exponential Smoothing . . . . .	43

# 1 Hoofdstuk 0: Basisgegevens

---

- Toevoegen van alle mogelijke package imports die we nodig gaan hebben

```
# Package imports for data science and time series analysis in particular
import numpy as np                                # "Scientific computing"
import pandas as pd                                # Dataframe
import scipy.stats as stats                        # Statistical tests

from pandas.api.types import CategoricalDtype

import matplotlib.pyplot as plt                    # Basic visualisation
import seaborn as sns                              # Advanced dataviz

from sklearn.linear_model import LinearRegression # Linear regression model

# Time series analysis
from datetime import datetime
from sklearn.metrics import mean_absolute_error, mean_squared_error
from statsmodels.tsa.api import Holt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.seasonal import seasonal_decompose
import math
```

## 2 Hoofdstuk 1: Samples

### 2.1 Theorie

### 2.2 Measurement levels (NORI):

#### 2.2.1 Kwalitatief

- Beschrijven eigenschappen of categorieën.
- Niet numeriek (of als het wel cijfers zijn, betekenen ze niks wiskundig).
- Voorbeelden:
  - Geslacht (man/vrouw)
  - Land (Belgi, Frankrijk)
  - Kleur (rood, blauw)

#### 2.2.2 Quantitatief

- Zijn echte getallen met een betekenis.
- Je kunt ermee rekenen (optellen, gemiddelde nemen, verhoudingen, enz.).
- Voorbeelden:
  - Leeftijd in jaren
  - Temperatuur in graden
  - Lengte in meter

#### 2.2.3 Kwalitatieve measurements

- Nominaal (geen volgorde):
  - Categorieën zonder rangorde.
  - Voorbeelden: geslacht, kleur, land.
- Ordinaal (wel volgorde):
  - Categorieën met rangorde.
  - Maar: de afstand tussen rangen is niet bekend of niet gelijk.
  - Voorbeelden:
    - Opleidingsniveau (middelbaar < bachelor < master)
    - Militaire rang

#### 2.2.4 Quantitatieve measurements

- Interval:
  - Gelijke verschillen tussen waarden.
  - Geen absoluut nulpunt: 0 heeft een betekenis (niet “niets”, maar juist wel).
  - Je kan geen verhoudingen maken (20C is niet twee keer zo warm als 10C).
  - Voorbeelden: temperatuur in C of F
- Ratio (verhouding):
  - Gelijke verschillen + absoluut nulpunt (0 betekent “niets”).
  - Je kan verhoudingen berekenen.
  - Voorbeelden: lengte in meter, gewicht in kg, energie in joule.
- Voorbeeld voor de duidelijkheid:
  - 20 meter is wel 1/3 langer dan 15 meter => ratio
  - 20C is NIET 1/3 warmer dan 15C => interval (omdat 0C niet “geen warmte” is)

### 2.3 Populatie

- Populatie = iedereen of alles dat je wil onderzoeken.
- **Voorbeeld:** alle studenten aan HoGent, alle auto's in België, alle klanten van een winkel.
- Het is vaak te veel om allemaal tegelijk te onderzoeken, daarom pikken we een sample er uit

### 2.4 Sample

- Steekproef = een klein groepje uit de populatie.

- Je kiest een deel van de populatie om gegevens van te verzamelen.
- **Voorbeeld:** je wil weten hoeveel uur studenten studeren per week. Je vraagt het aan 100 studenten (dat is je steekproef) in plaats van alle 10.000.

## 2.5 Hoe kies je een goede steekproef

- **Random steekproef (willekeurig)**
  - Iedereen uit de populatie heeft evenveel kans om gekozen te worden.
  - Zo vermijd je vooroordelen of vertekeningen.
  - Voorbeeld: je trekt 100 studenten at random uit de hele studentenlijst van HoGent.
- **Niet-random steekproef (niet willekeurig)**
  - Je kiest mensen die gemakkelijk bereikbaar zijn.
  - Dit heet ook wel een gemakssteekproef.
  - Voorbeeld: je vraagt gewoon aan je klasgenoten of vrienden.
  - Probleem: deze groep is misschien niet representatief, dus je resultaten kunnen vertekend zijn.

## 2.6 Sampling errors:

- Dit zijn fouten die ontstaan omdat je niet de volledige populatie gebruikt, maar slechts een steekproef:
- **Sampling errors:**
  - Toevallige steekproeffouten (Accidental sampling errors)
    - Fouten door **puur toeval**.
    - Je hebt niemand iets verkeerd gedaan, maar het resultaat is toch niet representatief.
    - Voorbeeld: je kiest willekeurig 10 studenten, maar het zijn toevallig allemaal jongens -> je steekproef is toevallig scheef.
  - Systematische steekproeffouten (Systematic sampling errors)
    - Fouten die ontstaan omdat de manier waarop je mensen kiest vooroordeel inbouwt (per ongeluk mensen uitsluiten).
    - Je sluit onbewust een deel van de populatie uit.
    - Voorbeelden:
      - Online enquete -> mensen zonder internet kunnen niet meedoen.
      - Straatinterview -> alleen mensen die toevallig op straat zijn worden bevraagd.
      - Vrijwillige enquete -> alleen mensen die zin hebben om mee te doen reageren (die zijn vaak meer betrokken).
- **Systeematische errors:**
- Fouten die niet te maken hebben met de steekproef op zich, maar met de manier waarop je data verzamelt, meet of verwerkt.
  - Perongeluk (toevallig) (Accidental systematic errors)
    - Fouten door vergissingen tijdens het invullen of verwerken van data.
    - Voorbeeld: iemand kruist per ongeluk het verkeerde vakje aan in de enquete.
  - Systematisch (structureel) (Systematic system errors)
    - Fouten die altijd opnieuw gebeuren omdat er iets fundamenteel mis is met de meetmethode.
    - Voorbeelden:
      - Je gebruikt een verkeerd afgestelde weegschaal -> alle gewichten zijn fout.
      - Mensen liegen bewust (bv. zeggen dat ze minder roken dan echt).
      - Het feit dat je iets meet, verandert het gedrag van mensen (bv. iemand gedraagt zich anders omdat ze weten dat ze geobserveerd worden).

## 2.7 Een dataset inlezen met basismethoden

- Data inlezen van github:

```
titanic = pd.read_csv('https://raw.githubusercontent.com/DataRepo2019/Data-
files/master/titanic.csv')
titanic.head() # of titanic.tail(10)
```

- Properties aflezen van je dataset (aantal rijen, aantal kolommen, datatypes ...)

```
# How many rows does the DataFrame have?
print(f"Number of rows: {len(titanic)}")

# How many columns?
print(f"Number of columns: {len(titanic.columns)}")

# Namen van de kolommen inlezen
print(titanic.columns)
# How many rows and columns, i.e. the shape
print(f"The shape of the Data Frame is: {titanic.shape}")

# General information about the DataFrame
print("*"*50)
titanic.info()

# Give the data type of each column.
print("*"*50)
print(titanic.dtypes)

# How many columns of each data type are there?
print("*"*50)
print(titanic.dtypes.value_counts())
```

- Sommige datasets hebben geen index, daarom kunnen we dat gemakkelijk zetten met: `set_index()`:

```
titanic.set_index(['PassengerId'])
```

## 2.8 Qualitatieve en ordinale variabelen

- Een kolom omzetten naar een kwalitatieve variabele:

```
# Describe the variable Survived -> is considered to be quantitative
print(titanic.Survived.describe())

# Convert to a categorical variable
titanic.Survived = titanic.Survived.astype('category')

# Ask to describe once more -> now it is considered to be qualitative
print(titanic.Survived.describe())
```

- Een kolom omzetten naar een ordinale variabele:

```
print(titanic.Embarked.unique())

embarked_type = CategoricalDtype(categories=['S', 'C', 'Q'], ordered=True)
titanic.Embarked = titanic.Embarked.astype(embarked_type)
titanic.Embarked.describe()

sns.countplot(data=titanic, x='Embarked')
```

## 2.9 Data selecteren

### 2.9.1 Tonen van specifieke kolommen:

```
titanic.Age # of titanic['Age']
```

```
# Select adjacent columns
titanic.iloc[:, 2:4]

# meerdere kolommen tegelijk
titanic[['Name', 'Age', 'Cabin']]

# Observation with row number 5 (counting from zero)
print(titanic.iloc[5])

# De eerste 4 kolommen
titanic.iloc[0:4]
```

### 2.9.2 Filteren van kolommen

```
# Volledige dataset tonen met deze volwaarde
titanic[titanic.Age < 18]

# Volledige dataset tonen met deze volwaarde maar enkel Embarked
titanic[titanic.Age < 18].Embarked

# Volledige dataset tonen met deze volwaarde maar enkel Age en Embarked
titanic[titanic['Age'] < 18][['Age', 'Embarked']]

# Filteren via een query
titanic.query("(Sex=='male') and (Age < 18)")
```

### 2.9.3 Kolommen verwijderen

```
titanic = titanic.drop("PassengerId", axis="columns")
```

### 2.9.4 Andere nuttige functies

```
# Toont de datatypes van alle kolommen
titanic.info()

# Verwijdert elke rij met minstens een Null waarde
titanic.dropna() # how="all" verwijdert enkel met rijen met alleen maar Null
                 waarden

# Toont hoeveel rijen elke kolom ingevuld heeft staan
print(titanic.count())

# Rijen vervangen door nieuwe rijen
avg_age = titanic['Age'].mean()
titanic = titanic.fillna(value={'Age' : avg_age})

# Een nieuwe kolommen maken van een al-bestaande kolom
titanic['Age_Times_Two'] = titanic['Age'] * 2

# Unieke waarde tonen
titanic['Age'].unique()

# Waardes vervangen via een set
mapping_dict = {'male' : 0, 'female' : 1}
titanic['Sex'] = titanic['Sex'].map(mapping_dict)
```



```
# Waardes vervangen via mapping
def age_to_category(age):
    if age < 12:
        return "child"
    if age < 18:
        return "teen"
    return "adult"

titanic['AgeCategory'] = titanic['Age'].map(age_to_category)
titanic.head()
```

## 3 Hoofdstuk 2: Analysis 1 var

### 3.1 Measure of Central Tendency

- Mean/average:
  - Som van alle waarden delen door aantal waarden
- Median
  - Middelste waarde van alle waarden
  - Even waarden = middelste twee waarden / 2
- Mode
  - De waarde die het meeste voorkomt

### 3.2 Measures of Dispersion

- **Range**
  - De absolute waarde van het verschil tussen de hoogste en kleinste waarde
- **Quartilen** (Q1, Q2 en Q3)
  - Bereken eerst de mediaan (Q2)
  - Bereken dan tussen het begin van de waarden en Q2 ook de mediaan
  - Hetzelfde voor Q2 tot het einde
- **Interkwartiele afstand**
  - Het verschil tussen Q3 en Q1
- **Variantie**
  - We gaan de array [2, 4, 8, 16, 30]
  - Eerst berekenen we het gemiddelde = 12
  - Nu berekenen hoe ver elke waarde ligt van het gemiddelde ( $X_i - \text{gemiddelde}$ )
    - $2 - 12 = -10$
    - $4 - 12 = -8$
    - $8 - 12 = -4$
    - $16 - 12 = 4$
    - $30 - 12 = 16$
    - Dan nemen we de som van al die waarden en nemen we daar de macht van
- **Standaarddeviatie**
  - Dit is simpelweg de vierkantswortel van de variantie

### 3.3 Kenmerken Standaarddeviatie

- Nooit negatief (minimum is 0, wat betekent dat alle waarden exact gelijk zijn.).
- Outliers verhogen de standaarddeviatie aanzienlijk
- Eenheid van de output is delfde eenheid als de oorspronkelijke data.
- Hoe interpreteer je de standaarddeviatie samen met het gemiddelde? -> Ze tonen samen het centrum en de spreiding van je data.

Measurement Level	Center	Spread Distribution
Qualitative	Mode	—
Quantitative	Average/Mean Median	Variance, Standard Deviation Range, Interquartile Range

Figure 1: alt text

Measurement level	Chart type
Qualitative	Bar chart
Quantitative	Boxplot Histogram Density plot

Figure 2: alt text

### 3.4 Qualitatieve variabelen

- Data visualiseren van kwalitatieve variabelen tonen we dat via catplots:

```
# Bar chart in Seaborn: catplot() with 'kind = "count"'
sns.catplot(data=tips, kind='count', y='day');
```

- De ‘mode’ kan je weer weergeven via deze codes:

```
tips.mode() # or tips[nameField].mode()
```

- ‘Describe’ zal van alle kwalitatieve variabelen enkele statistieken weergeven:

```
tips.describe()
```

### 3.5 Quantitative variables

- Data visualiseren van kwalitatieve variabelen tonen we dat via boxplots:

```
sns.boxplot(data=tips, x='tip');
```

- Dit kan ook met een staafdiagram (histogram):

```
sns.displot(data=tips, x='tip'); # je kan ook bins aanpassen hier
```

- Alsook met een kdeplot:

```
sns.kdeplot(data=tips, x='tip');
```

- We kunnen ook de vorige twee plotten combineren in n:

```
sns.displot(data=tips, x='tip', kde=True);
```

- En uiteindelijk een violplot, dit is een combinatie van de box en density plot.

```
sns.violinplot(data=tips, x='tip');
```

### 3.6 Centrality and Dispersion Measures

```
# Centrality and dispersion measures
# Mean, standard deviation & friends
print(f"Mean: {tips.tip.mean()}") # Gemiddelde
print(f"Standard deviation: {tips.tip.std()}") # Standaarddeviatie
print(f"Variance: {tips.tip.var()}") # Variantie
print(f"Skewness: {tips.tip.skew()}") # Scheefheid
print(f"Kurtosis: {tips.tip.kurtosis()}") # Kurtosis (spitsheid)

# Median & co
print(f"Minimum: {tips.tip.min()}")
print(f"Median: {tips.tip.median()}")
print(f"Maximum: {tips.tip.max()}")
percentiles = [0.0, 0.25, 0.5, 0.75, 1.0]
print(f"Percentiles {percentiles}\n{tips.tip.quantile(percentiles)}")
)
print(f"Inter Quartile Range: {stats.iqr(tips.tip)}")
print(f"Range : {tips.tip.max() - tips.tip.min()}")
```

### 3.7 Formula for Standard Deviation

- Dit zijn alle hulpmethoden die worden gebruikt voor het berekenen hiervan:

```
# Variance of the population
def pop_var(x):
    n = len(x)
    mean = sum(x) / n
    return 1/n * sum((x - mean) ** 2)

# Standard deviation of the population
def pop_sd(x):
    return np.sqrt(pop_var(x))

# Variance of a sample
def sample_var(x):
    n = len(x)
    mean = sum(x) / n
    return 1/(n-1) * sum((x - mean) ** 2)

# Standard deviation of the sample
def sample_sd(x):
    return np.sqrt(sample_var(x))

# If we calculate the variance in both ways, we see a difference:
a = np.array([4, 8, 6, 5, 3, 2, 8, 9, 2, 5])

print(f"Population variance: {pop_var(a)}")
print(f"Sample variance: {sample_var(a)}")
print()
print(f"Population standard deviation: {pop_sd(a)}")
print(f"Sample standard deviation: {sample_sd(a)}")
```

- Wat doet dit:

```
population = np.arange(0, 101)

population_mean = sum(population) / len(population)
```

```
print(f'Population mean: {population_mean}')

population_variance = pop_var(population)
print(f'Population variance: {population_variance}')

population_stdev = pop_sd(population)
print(f'Population standard deviation: {population_stdev}')
```

- Now for the actual experiment: we repeatedly take a random sample from this population and look at the result of both formulas for the variance. We will choose the formula that comes closest to the true variance of the population over a large number of iterations:

```
sample_size = 15          # the size of the sample
num_samples = 5000        # number of times we will take a sample

# We are going to calculate the average of the result with each formula.
# In order to do this, we keep track of the sum of these results and
# divide afterwards by the number of samples
sum_pop = 0
sum_smp = 0

for _ in range(num_samples):
    sample = np.random.choice(population, size=sample_size)
    sum_pop = sum_pop + pop_sd(sample)
    sum_smp = sum_smp + sample_sd(sample)

print(f"Average result with formula population      : {sum_pop/num_samples}")
print(f"Average result with formula sample         : {sum_smp/num_samples}")
print(f"The true standard deviation of the population: {population_stdev}")
```

• :

```
a = np.array([1,2,3,4,5])

print(f"Population variance with numpy : {np.var(a)}")
print(f"Population variance own function: {pop_var(a)}")
print()
print(f"Sample variance with numpy : {np.var(a, ddof=1)}")
print(f"Sample variance own function: {sample_var(a)}")
print()
print(f"Standard deviation population: {np.std(a)}")
print(f"Standard deviation sample : {np.std(a, ddof=1)}")
```

## 4 Hoofdstuk 3: Central Limit Testing

```
mu = 63      # Mean score History
sigma = 8    # Standard deviation History
x = np.linspace(mu - 4 * sigma, mu + 4 * sigma, num=201)
y = stats.norm.pdf(x, mu, sigma)
plt.plot(x, y)
plt.axvline(75, color="green"); # Show a green line for Alexandra's score for History
```

### 4.1 3.02 Confidence Intervals

- Confidence interval is om te berekenen of de steekproef data in de populatieparameter ligt
- Een confidence interval (CI) geeft een schatting van waar het werkelijke populatiegemiddelde zich bevindt, op basis van een steekproef.
- Bijvoorbeeld: een 95% CI van [321.3, 327.9] betekent dat we 95% zeker zijn dat het echte gemiddelde ergens in dat interval zit.

Situatie	Gebruik	Verdeling	Reden
- <b>Kleine steekproef</b> ( $n < 30$ ) en onbekend	t-verdeling	stats.t	De t-verdeling compenseert de extra onzekerheid bij kleine samples.
- <b>Grote steekproef</b> ( $n \geq 30$ ) of bekend	z-verdeling	stats.norm	Bij grote steekproeven geldt de Centrale Limietstelling, dus gebruik je de normale verdeling.
- Populatiestandaardafwijking bekend, ongeacht n	z-verdeling	stats.norm	Want er is geen extra onzekerheid van schatting van .

For a  $t$ -distribution with  $df$  degrees of freedom: ( $df$  = degrees of freedom)

Function	Purpose
stats.t.pdf(x, df=d)	Probability density for $x$
stats.t.cdf(x, df=d)	Left-tail probability ( $< x$ )
stats.t.sf(x, df=d)	Right-tail probability ( $> x$ )
stats.t.isf(1-p, df=d)	p% of observations are expected to be lower than this value

#### 4.1.1 Z-score

- Calculate a 95% confidence interval (big sample)

```
import math
# Step 1.
m = 324.6      # Sample mean
s = 2.5        # Population standard deviation
n = 45         # Sample size
alpha = .05    # 1 - alpha is the confidence level

# Step 2.
z = stats.norm.isf(alpha/2)
print("z-score: %.5f" % z)
```

```
# Step 3.
lo = m - z * s / np.sqrt(n)
hi = m + z * s / np.sqrt(n)
print("Confidence interval: [%.4f, %.4f]" % (lo, hi))
```

#### 4.1.2 T-score

- Calculate a x% confidence interval (small sample)

```
import math

# Step 1.
m = 324.6          # Sample mean
s = 2.5            # Population standard deviation
n = 15             # Sample size
alpha = .05        # 1 - alpha is the confidence level

# Step 2.
t = stats.t.isf(alpha/2, df = n - 1)
print("t-score: %.5f" % t)

# Step 3.
lo = m - t * s / np.sqrt(n)
hi = m + t * s / np.sqrt(n)
print("Confidence interval: [%.4f, %.4f]" % (lo, hi))
```

#### 4.1.3 Berekening zonder alpha

Betrouwbaarheidsniveau	(significantieniveau)	/2	z-waarde
90%	0.10	0.05	<b>1.645</b>
95%	0.05	0.025	<b>1.960</b>
99%	0.01	0.005	<b>2.576</b>

## 4.2 3.03 Hypothesis testing

### 4.2.1 Z-test

#### 4.2.1.1 Kenmerken Z-test

- We gebruiken de Z-test wanneer:
  - The sample needs to be random
  - The sample size needs to be sufficiently large ( $n \geq 30$ )
  - The test statistic needs to have a normal distribution
  - The standard deviation of the population,  $\sigma$ , is known
- Het moment dat de standaarddeviatie bekend is, gebruiken we dit, anders is het de T-test

#### 4.2.1.2 Right tailed Z-test

- We gebruiken de right tailed Z-test wanneer het gemiddelde van de populatie kleiner is dan het gemiddelde van de sample

```
# Properties of the sample:
import math
n = 50          # Sample size
mu = 34.2       # (Hypothetical) population mean
```

```

sigma = 2.6          # Population standard deviation (assumed to be known)
m_sample = 33.5      # Sample mean
alpha = 0.05         # Significance level
s_sample = sigma / math.sqrt(n)

# X-values
dist_x = np.linspace(mu - 4 * s_sample, mu + 4 * s_sample, num=201)
# Y-values for drawing the Gauss curve
dist_y = stats.norm.pdf(dist_x, mu, s_sample)
# Plot the Gauss-curve
plt.plot(dist_x, dist_y)

# Fill the area left of x
# plt.fill_between(dist_x, 0, dist_y, where=(dist_x <= x), color='lightblue')
# Show the mean with an orange line
plt.axvline(m_sample, color="blue");

```

- How to write a conclusion:

```

**Step 1.** Formulate the hypotheses:

- $H_0$:  $\mu = 351.2$  (the machine is working correctly)
- $H_1$:  $\mu > 351.2$  (the machine is not working correctly)

```

- P-value berekenen

```

p = stats.norm.sf(m_sample, loc=mu, scale=sigma/np.sqrt(n))

print("p-value: %.5f" % p)
if(p < alpha):
    print("p < alpha: reject H0")
else:
    print("p > alpha: do not reject H0")

```

- Critical value berekenen:

```

# Critical value according to the formula:
# g = mu + stats.norm.isf(alpha) * sigma / np.sqrt(n)
# Making use of the loc/scale parameters of isf:
g = stats.norm.isf(alpha, loc=mu, scale=sigma / np.sqrt(n))
print("Critical value g %.3f" % g)
if (m_sample < g):
    print("sample mean = %.3f < g = %.3f: do not reject H0" % (m_sample, g))
else:
    print("sample mean = %.3f > g = %.3f: reject H0" % (m_sample, g))

```

- Dit wordt geplot:

```

# Gauss-curve plot:
# X-values
dist_x = np.linspace(mu - 4 * sigma/np.sqrt(n), mu + 4 * sigma/np.sqrt(n), num
=201)
# Y-values for the Gauss curve
dist_y = stats.norm.pdf(dist_x, mu, sigma/np.sqrt(n))
fig, dplot = plt.subplots(1, 1)
# Plot the Gauss-curve
dplot.plot(dist_x, dist_y)

```



```
# Show the hypothetical population mean with an orange line
dplot.axvline(mu, color="orange", lw=2)
# Show the sample mean with a red line
dplot.axvline(m_sample, color="red")
# Fill the acceptance area in light blue
dplot.fill_between(dist_x, 0, dist_y, where=dist_x <= g, color='lightblue');
```

#### 4.2.1.3 Left tailed Z-test

- We gebruiken de left tailed Z-test wanneer het gemiddelde van de populatie groter is dan het gemiddelde van de sample

```
import math
n = 50                # Sample size
mu = 34.2             # (Hypothetical) population mean
sigma = 2.6           # Population standard deviation (assumed to be known)
m_sample = 33.5       # Sample mean
alpha = 0.05          # Significance level
s_sample = sigma / math.sqrt(n)
```

```
# Pay attention! Use cdf() instead of sf()! Why?
p = stats.norm.cdf(m_sample, loc=mu, scale=sigma/np.sqrt(n))

print("p-value: %.5f" % p)
if(p < alpha):
    print("p < a, reject H0")
else:
    print("p > a, do not reject H0")
```

- Critical value berekenen:

```
# Critical value according to the formula in the lecture slides
# g = m0 - stats.norm.isf(a) * s / np.sqrt(n)
# Making use of the loc/scale parameters of isf:
g = stats.norm.isf(1-alpha, loc=mu, scale=(sigma / np.sqrt(n)))
print("Critical value g %.3f" % g)
if (m_sample > g):
    print("sample mean = %.3f > g = %.3f: do not reject H0" % (m_sample, g))
else:
    print("sample mean = %.3f < g = %.3f: reject H0" % (m_sample, g))
```

- Dit plotten we:

```
# Gauss-curve:
# X-values
dist_x = np.linspace(mu - 4 * sigma/np.sqrt(n), mu + 4 * sigma/np.sqrt(n), num
                    =201)
# Y-values
dist_y = stats.norm.pdf(dist_x, mu, sigma/np.sqrt(n))
fig, dplot = plt.subplots(1, 1)
# Plot
dplot.plot(dist_x, dist_y)
# Hypothetical population mean in orange
dplot.axvline(mu, color="orange", lw=2)
# Sample mean in red
dplot.axvline(m_sample, color="red")
# Fill the acceptance region in light blue
```

```
dplot.fill_between(dist_x, 0, dist_y, where=dist_x >= g, color='lightblue');
```

#### 4.2.1.4 Two tailed Z-test

```
import math
n = 50                # Sample size
mu = 34.2             # (Hypothetical) population mean
sigma = 2.6           # Population standard deviation (assumed to be known)
m_sample = 33.5       # Sample mean
alpha = 0.05          # Significance level
s_sample = sigma / math.sqrt(n)
```

- Wanneer we niet willen weten of iets groter of kleiner, maar enkel willen weten of de sample 'close enough' is, gebruiken we deze test.
- P-waarde berekenen:

```
p = stats.norm.sf(m_sample, loc=mu, scale=sigma/np.sqrt(n))
print("p-waarde: %.5f" % p)
if(p < alpha/2):
    print("p < alpha/2, reject H0")
else:
    print("p > alpha/2, do not reject H0")
```

- De twee critical values berekenen:

```
g1 = mu - stats.norm.isf(alpha/2) * sigma / np.sqrt(n)
g2 = mu + stats.norm.isf(alpha/2) * sigma / np.sqrt(n)

print("Acceptance region [g1, g2]   [%.3f, %.3f]" % (g1,g2))
if (g1 < m_sample and m_sample < g2):
    print("Sample mean = %.3f is inside acceptance region: do not reject H0" %
          m_sample)
else:
    print("Sample mean = %.3f is outside acceptance region: reject H0" %
          m_sample)
```

- Dit plotten we:

```
# Gauss-curve
# X-values
dist_x = np.linspace(mu - 4 * sigma/np.sqrt(n), mu + 4 * sigma/np.sqrt(n), num
                    =201)
# Y-values
dist_y = stats.norm.pdf(dist_x, loc=mu, scale=sigma/np.sqrt(n))
fig, dplot = plt.subplots(1, 1)
# Plot
dplot.plot(dist_x, dist_y)
# Hypothetical population mean in orange
dplot.axvline(mu, color="orange", lw=2)
# Sample mean in red
dplot.axvline(m_sample, color="red")
acc_x = np.linspace(g1, g2, num=101)
acc_y = stats.norm.pdf(acc_x, loc=mu, scale=sigma/np.sqrt(n))
# Fill the acceptance region in light blue
dplot.fill_between(acc_x, 0, acc_y, color='lightblue');
```

## 4.2.2 Student's T-test

- Eigen

```
accidents = [9.75, 9.33, 9.57, 9.26, 9.45, 9.72]
n = len(accidents)                # Sample size
mu = 9.15                         # (Hypothetical) population mean
s = 32                           # Population standard deviation (
    assumed to be known)
m_sample = np.mean(accidents)     # Sample mean
alpha = 0.05                     # Significance level (chosen by the
    researcher)

print(f"Sample mean: {m_sample}")
```

- P-value berekenen:

```
p = stats.t.cdf(m_sample, loc = mu, scale = s / math.sqrt(n), df = n - 1)
print(f"p-value: {p}")
if(p < alpha):
    print("p < alpha: reject H0")
else:
    print("p > alpha: do not reject H0")
```

- Critical value berekenen:

```
g = stats.t.isf(1-alpha, loc = mu, scale = s / math.sqrt(n), df = n - 1)
print("Critical value g %.3f" % g)
if (m_sample > g):
    print("sample mean = %.3f > g = %.3f: do not reject H0" % (m_sample, g))
else:
    print("sample mean = %.3f < g = %.3f: reject H0" % (m_sample, g))
```

- Dit plotten we:

```
# Gauss-curve
# X-values
dist_x = np.linspace(mu - 4 * s, mu + 4 * s, num=201)
# Y-values
dist_y = stats.t.pdf(dist_x, loc=mu, scale=s/np.sqrt(n), df=n-1)
fig, dplot = plt.subplots(1, 1)
# Plot
dplot.plot(dist_x, dist_y)
# Hypothetical population mean in orange
dplot.axvline(mu, color="orange", lw=2)
# Sample mean in red
dplot.axvline(m_sample, color="red")
# Fill the acceptance region in light blue
dplot.fill_between(dist_x, 0, dist_y, where=dist_x >= g, color='lightblue');
```

- Het kan ook korter met enkel deze functie:

```
study_times = [18,22,19,21,23,20,17,22,19,18,21,20,16,24,19]
alpha = 0.05
mu = 20

t_stat, p_val = stats.ttest_1samp(a=study_times, popmean=mu, alternative='less
')
```

```
print("Sample mean      : %.3f" % np.mean(study_times))
print("t-score          : %.3f" % t_stat)
print(f"p-value         : {p_val}")
```

## 5 Hoofdstuk 4: Bivariate Qual

### 5.1 Crosstab

- Hier is hoe je een crosstab kan maken met een legende:

```
df.Gender = df.Gender.astype('category')
age_scale = CategoricalDtype(categories=['Less than 20', 'Between 20 and 40', '
    Between 40 and 60', 'Over 60'], ordered=True)
df.Age = df.Age.astype(age_scale)
observed = pd.crosstab(df.Preference, df.Age)
observed.plot(kind='barh', stacked=True)
```

### 5.2 Chi-kwadraad en Cramr's V

#### 5.2.1 Data visualasations

- Bar chart:

```
# Contingency table without the margins
observed = pd.crosstab(rlanders.Gender, rlanders.Survey)

# Horizontally oriented stacked bar chart
observed.plot(kind='barh', stacked=True);
```

- Bar chart zonder marges:

```
# Contingency table without the margins
observed_p = pd.crosstab(rlanders.Gender, rlanders.Survey, normalize='index')

# Horizontally oriented stacked bar chart
observed_p.plot(kind='barh', stacked=True);
```

#### 5.2.2 Chi kwadraad

- Informatie weergeven van de kolommen

```
observed = pd.crosstab(rlanders.Survey, rlanders.Gender)
row_sums = observed.sum(axis=1)
col_sums = observed.sum()
n = row_sums.sum()

print(row_sums)
print(col_sums)
print(f'Number of observations: {n}')
```

- Antwoorden weergeven per soort dat jij aanduidde?

```
expected = np.outer(row_sums, col_sums) / n
expected
```

- Controle of de margins nog kloppen (meestal niet nodig):

```
exp_row_sums = np.sum(expected, axis=1)
exp_col_sums = np.sum(expected, axis=0)

print(f'Row totals : {exp_row_sums}')
```

```
print(f'Column totals: {exp_col_sums}')
print(f'Observations : {exp_col_sums.sum()}')
```

- Het verschil zien van de expected en observed:

```
expected - observed
```

- Dit controleert iets in verband met de verschillen met expected en observed?

```
diffs = (expected - observed)**2 / expected
print(diffs)
```

- Chi-squared eindelijk berekenen:

```
chi_squared = diffs.values.sum()
print(' %.3f' %chi_squared)
```

- Cramr' V berekenen:

```
dof = min(observed.shape) - 1
cramers_v = np.sqrt(chi_squared / (dof * n))
print(cramers_v)

# of op deze manier:
# crammers_v = stats.contingency.association(observed, method='cramer')
```

- Het geven van een conclusie of er een sterk verband is of niet kan via deze tabel:

Cramr's V	Interpretation
:---:	:---
0	No association
<=0.1	Weak association
<=0.25	Moderate association
<=0.50	Strong association
<=0.75	Very strong association
1	Complete association

### 5.3 Goodness of fit test

- Een Goodness-of-Fit test kijkt of de waargenomen data overeenkomt met wat je zou verwachten op basis van een kansverdeling (kijken of de steeproef representatief is).
- Bijvoorbeeld:
  - Stel je gooit een dobbelsteen 60 keer. Je verwacht dat elk getal (1 t/m 6) ongeveer 10 keer voorkomt. Maar je telt: 8, 12, 9, 11, 7, 13. Is dat verschil normaal toeval, of past het niet bij een eerlijke dobbelsteen?

```
types = ['mutant', 'human', 'alien', 'god', 'demon']
observed = np.array([ 127, 75, 98, 27, 73])
expected_p = np.array([ .35, .17, .23, .08, .17])
```

- Om aan het resultaat te komen moeten we dit allemaal gaan uitvoeren:

```
alpha = 0.05 # Significance level
n = sum(observed) # Sample size
k = len(observed) # Number of categories
dof = k - 1 # Degrees of freedom
expected = expected_p * n # Expected absolute frequencies in the sample
g = stats.chi2.isf(alpha, df=dof) # Critical value
```

```
# Goodness-of-fit-test in Python:
chi2, p = stats.chisquare(f_obs=observed, f_exp=expected)

print("Significance level  alpha = %.2f" % alpha)
print("Sample size        n = %d" % n)
print("k = %d; df = %d" % (k, dof))
print("Chi-squared        x squared = %.4f" % chi2)
print("Critical value      g = %.4f" % g)
print("p-value            p = %.4f" % p)
```

- Hier plotten we alles even

```
# Plot of the case:
# x-values:
x = np.linspace(0, 15, num=100)
# probability density of the chi-squared distribution with 4 degrees of freedom
y = stats.chi2.pdf(x, df=dof)
# the number q for which the right tail probability is exactly 5%:
q = stats.chi2.isf(alpha, df=dof)

fig, tplot = plt.subplots(1, 1)
tplot.plot(x, y) # probability density
tplot.fill_between(x, y, where=x>=q, # critical area
                  color='lightblue')
tplot.axvline(q) # critical value
tplot.axvline(chi2, color='orange'); # chi-squared
```

## 5.4 Chi-squared toets voor onafhankelijkheid

- Hiermee test je of twee categorische variabelen onafhankelijk zijn.
- Voorbeeld: Is er een verband tussen geslacht en voorkeur?

```
# Maak een kruistabel (contingentietabel)
contingency = pd.crosstab(df['Gender'], df['Preference'])

# Chi-kwadraattoets
chi2, p, dof, expected = stats.chi2_contingency(contingency)

print("x squared = %.4f" % chi2)
print("df = %d" % dof)
print("p = %.4f" % p)
```

## 5.5 Cochran's rule

Een chi-squared test kan enkel voldoen als:

- All expected values must be at least 1
- At most 20% of the expected values may be smaller than 5

## 6 Hoofdstuk 5: Bivariate Qual Quant

- Als je wilt weten of een numeriek verschil bestaat tussen twee of meer categorien gebruik je dit.
- Bijvoorbeeld: “Geven mannen gemiddeld meer fooi dan vrouwen?”

### 6.1 Boxplot

- Toont spreiding, mediaan en outliers per groep.
- Prima manier om verschil tussen groepen visueel te vergelijken.

```
tips = sns.load_dataset("tips")
sns.boxplot(data=tips, x='tip', y='sex');
```

### 6.2 Violinplot

- Combineert boxplot met een density plot.
- Je ziet ook hoe de data verdeeld is.

```
sns.violinplot(data=tips, x='tip', y='sex');
```

### 6.3 Density/kdeplot

- Laat de kansverdeling van de variabele tip zien, gescheiden per geslacht.
- Handig om verschillen in vorm, spreiding of centrum te ontdekken.

```
sns.kdeplot(data=tips, hue='sex', x='tip');
```

### 6.4 Barplot met errors

- Toont het gemiddelde per groep + standaarddeviatie als foutmarge.
- Goed voor een snelle visuele indruk.

```
sns.barplot(data=tips, x='sex', y='tip', errorbar='sd');
```

### 6.5 2-sample t-test for independent samples

- Wordt gebruikt om te testen of er een significant verschil is tussen de gemiddelden van twee onafhankelijke groepen.
- Dit gebruik je wanneer je twee niet-gerelateerde groepen vergelijkt, dus twee totaal verschillende steekproeven.
- Voorbeeld:
  - Je test een medicijn op 10 mensen en vergelijkt hun reactietijd met 10 andere mensen die geen medicijn kregen.
  - Dit zijn onafhankelijke (non-related) steekproeven.

```
stats.ttest_ind(a=control, b=treatment, alternative='less', equal_var=False)
```

- Dit kan ook

```
stats.ttest_ind(alternative='two-sided', a=tips.tip[tips.sex == 'Male'], b=tips.tip[tips.sex == 'Female']);
```

Waarde	Betekenis
"two-sided"	Er is een verschil (je weet niet of het groter of kleiner is)



Waarde	Betekenis
"less"	De eerste groep ( <b>a</b> ) heeft een <b>lager gemiddelde</b> dan groep <b>b</b>
"greater"	De eerste groep ( <b>a</b> ) heeft een <b>hoger gemiddelde</b> dan groep <b>b</b>

## 6.6 The t-test for paired samples

- Gebruik je wanneer je twee metingen hebt van dezelfde groep personen/objecten.
- Dit gebruik je wanneer je dezelfde mensen/objecten twee keer meet, onder verschillende condities.
- Voorbeeld:
  - Je laat dezelfde 10 mensen eerst een medicijn nemen en meet hun reactietijd. Daarna geef je ze geen medicijn en meet je opnieuw.
  - Dit zijn afhankelijke (paired) metingen.
- Uitgewerkt voorbeeld: Zelfde auto's getest met gewone benzine vs. additieven.

```
# Measurements
regular = np.array([16, 20, 21, 22, 23, 22, 27, 25, 27, 28])
additives = np.array([19, 22, 24, 24, 25, 25, 26, 26, 28, 32])

# Visualization
sns.boxplot(
    data={'Regular': regular, 'Additives': additives},
    orient='h');
```

```
# Paired t-test with ttest_rel()
p = stats.ttest_rel(regular, additives, alternative='less')
```

## 6.7 Effect Size

- Een test kan statistisch significant zijn maar het verschil kan praktisch onbelangrijk zijn.
- Daarom gebruiken we een effectgrootte zoals Cohen's

## 6.8 Cohens D

- Maat voor hoe groot het verschil tussen twee gemiddelden is, in standaarddeviaties.

```
def cohen_d(a, b):
    na = len(a)
    nb = len(b)
    pooled_sd = np.sqrt( ((na-1) * np.var(a, ddof=1) +
                          (nb-1) * np.var(b, ddof=1)) / (na + nb - 2) )
    return (np.mean(b) - np.mean(a)) / pooled_sd
```

```
# Effect size of additives in gasoline:
cohen_d(regular, additives)
```

- Betekenis resultaten | Cohen's d | Betekenis | | ——— | ————— | | ≤0.2 | Klein effect | | ≤0.5 | Gemiddeld effect | | ≤0.8+ | Groot effect |

## 7 Hoofdstuk 6: Regression Analysis

### 7.1 Visualisation

- Voor we een regressiemodel toepassen, is het belangrijk om eerst visueel te kijken of er een (lineair) verband is tussen twee variabelen.
- Hiervoor gebruiken we een scatterplot met `relplot()` uit Seaborn.
- Dit tekent ook een rechte

```
sns.relplot(data=penguins, x='flipper_length_mm', y='body_mass_g');
```

- Door extra variabelen zoals species (soort) en sex (geslacht) toe te voegen, kan je groepen visueel onderscheiden:

```
sns.relplot(data=penguins,
            x='flipper_length_mm', y='body_mass_g',
            hue='species', style='sex');
```

- Ook extra dimensies zoals grootte kan je tonen:

```
sns.relplot(data=penguins,
            x='bill_depth_mm', y='bill_length_mm',
            size='body_mass_g',
            hue='species', style='sex');
```

- Samenvoegen van 2 plots:

```
fig, axs = plt.subplots(1, 2, figsize=(20, 5))

sns.regplot(ax=axs[0], data=movies, x=movies['7-day Gross'], y=movies['Total US Gross'])
axs[0].set_title("7-day gross vs Total US Gross")

sns.regplot(ax=axs[1], data=movies, x=movies['14-day Gross'], y=movies['Total US Gross'])
axs[1].set_title("14-day gross vs Total US Gross")
```

### 7.2 Method of least squares introduction

$$\beta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

- Dit zijn de formules die we hiervoor moeten bekomen
- Dit wordt gedaan met de volgende code:

```
least_squares = pd.DataFrame({
    'x': male_chinstrap.flipper_length_mm,
    'y': male_chinstrap.body_mass_g
})
mx = least_squares.x.mean()
my = least_squares.y.mean()
```

```

least_squares['(x-x mean)'] = least_squares['x'] - mx
least_squares['(y-y mean)'] = least_squares['y'] - my

least_squares['(x-x mean)(y-y mean)'] = least_squares['(x-x mean)'] *
    least_squares['(y-y mean)']
least_squares['(x-x mean) to the power of 2'] = least_squares['(x-x mean)'] **
    2
least_squares

```

### 7.3 Plotting Regressionline

- Hier worden de regressielijn getekent
- Dankzij de regressielijn kunnen we ook voorspellingen doen

```

farms_x = farms['year']
farms_y = farms['average_area_per_farm_(ha)']

# b en richtingscoefficient
a, b = np.polyfit(farms_x, farms_y, 1)
print(f'Regression line: y = {a} * x + {b}')

print(f'De gemiddelde oppervlakte in 2035: {a * 2035 + b}')

```

- wat dit ook mag zijn:

```

# Numerator and denominator of the formula for b_0:
num = sum(least_squares['(x-x mean)(y-y mean)'])
denom = sum(least_squares['(x-x mean)to the power of 2'])
beta1 = num/denom
beta0 = my - beta1 * mx

print(f"beta_1 = {num:.4f} / {denom:.4f} = {beta1:.4f}")
print(f"beta_0 = {my:.4f} - {beta1:.4f} * {mx:.4f} = {beta0:.4f}")
print(f"y = {beta0:.4f} + {beta1:.4f} x")

```

- De grafiek kan ook zo:

```

# sns.lmplot(data=male_chinstrap, x='flipper_length_mm', y='body_mass_g');
sns.regplot(data=male_chinstrap, x='flipper_length_mm', y='body_mass_g');

```

### 7.4 Covariantie

- Covariantie meet de samenhang tussen twee variabelen. Het zegt iets over de richting van het verband (positief/negatief), maar niet over de sterkte.
- Covariantie berekenen:

```

np.cov(
    male_chinstrap.flipper_length_mm,
    male_chinstrap.body_mass_g,
    ddof=1)[0][1]

```

\*De correlatie (Pearson's R) is een gestandaardiseerde maat voor samenhang (tussen -1 en +1):

```

cor = np.corrcoef(
    male_chinstrap.flipper_length_mm,

```

```
male_chinstrap.body_mass_g)[0][1]
print(f"R is approximicly {cor:.4f}")
```

Cov > 0: increasing

Cov is ongeveer 0: no relationship

Cov < 0: decreasing

## 7.5 Coefficient of determination

- R tot de 2e macht vertelt hoeveel procent van de variantie in y verklaard wordt door x (verklaarde variantie).
- Je berekent het eenvoudig met:

```
cor ** 2
```

$abs(R)$	$R^2$	Explained variance	Linear relation
< .3	< .1	< 10%	very weak
.3 - .5	.1 - .25	10% - 25%	weak
.5 - .7	.25 - .5	25% - 50%	moderate
.7 - .85	.5 - .75	50% - 75%	strong
.85 - .95	.75 - .9	75% - 90%	very strong
> .95	> .9	> 90%	exceptionally strong

- Bijvoorbeeld: Bijvoorbeeld als  $R = 0.87$  R tot de tweede is ongeveer 0.76, wat betekent dat 76% van de spreiding in y verklaard wordt door x.

$abs(R)$	$R^2$	Explained variance	Linear relation
< .3	< .1	< 10%	very weak
.3 - .5	.1 - .25	10% - 25%	weak
.5 - .7	.25 - .5	25% - 50%	moderate
.7 - .85	.5 - .75	50% - 75%	strong
.85 - .95	.75 - .9	75% - 90%	very strong
> .95	> .9	> 90%	exceptionally strong

Figure 3: alt text

## 8 Hoofdstuk 7: Time Series

### 8.1 Componenten van Tijdreeksgegevens

- Een tijdreeks is een reeks waarnemingen die op opeenvolgende tijdstippen zijn verzameld (bijvoorbeeld maandelijks of jaarlijks). Een tijdreeks bestaat meestal uit vier componenten:
  - **Trendcomponent:** De algemene richting van de data op lange termijn (bijv. een stijgende of dalende lijn).
  - **Seizoenscomponent:** Veel tijdreeksen vertonen een seizoenspatroon, zoals jaarlijkse herhalingen in verkoopcijfers. Dit patroon is doorgaans voorspelbaar van jaar tot jaar. Regelmatige patronen die zich voordoen op vaste tijdsintervallen, zoals jaarlijks of maandelijks. Voorbeeld: hogere ijsverkoop in de zomer
  - **Cyclische component:** Dit verwijst naar stijgingen en dalingen die geen vaste periode hebben, zoals economische recessies en herstelperiodes. Het verschil met seizoenspatronen is dat cyclische patronen onregelmatig zijn qua duur, terwijl seizoenspatronen gekoppeld zijn aan de kalender en een vaste periode hebben.
  - **Willekeurige component (of ruis):** Dit is het onvoorspelbare deel dat de onregelmatige, zigzag-uitstraling van de meeste tijdreeksgrafieken veroorzaakt. Het omvat inherente willekeur, onvoorspelbare “schokken” en menselijk gedrag. Onverklaarbare of toevallige variaties. Dit is het “chaotische” deel dat overblijft als de andere drie componenten zijn verwijderd.

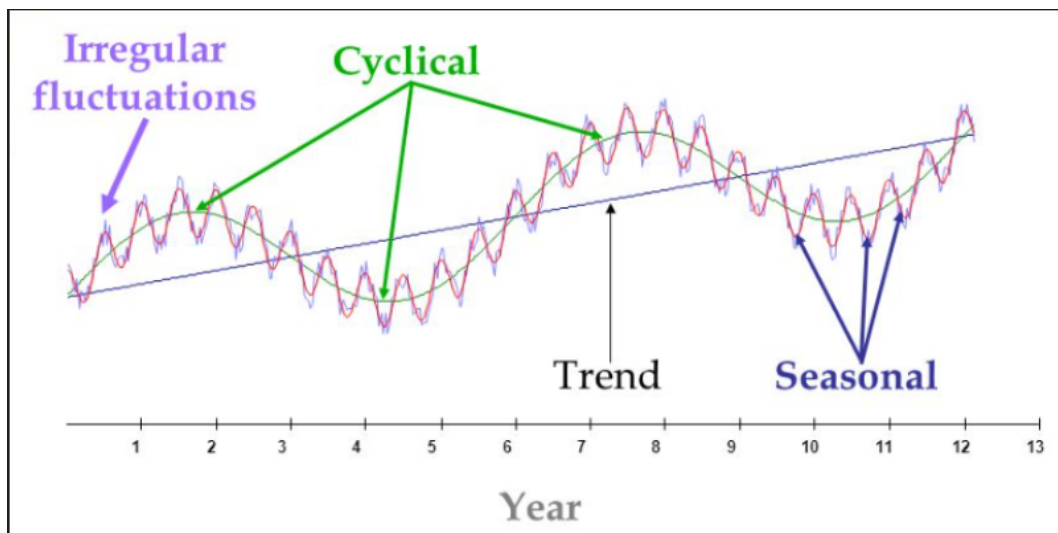


Figure 4: alt text

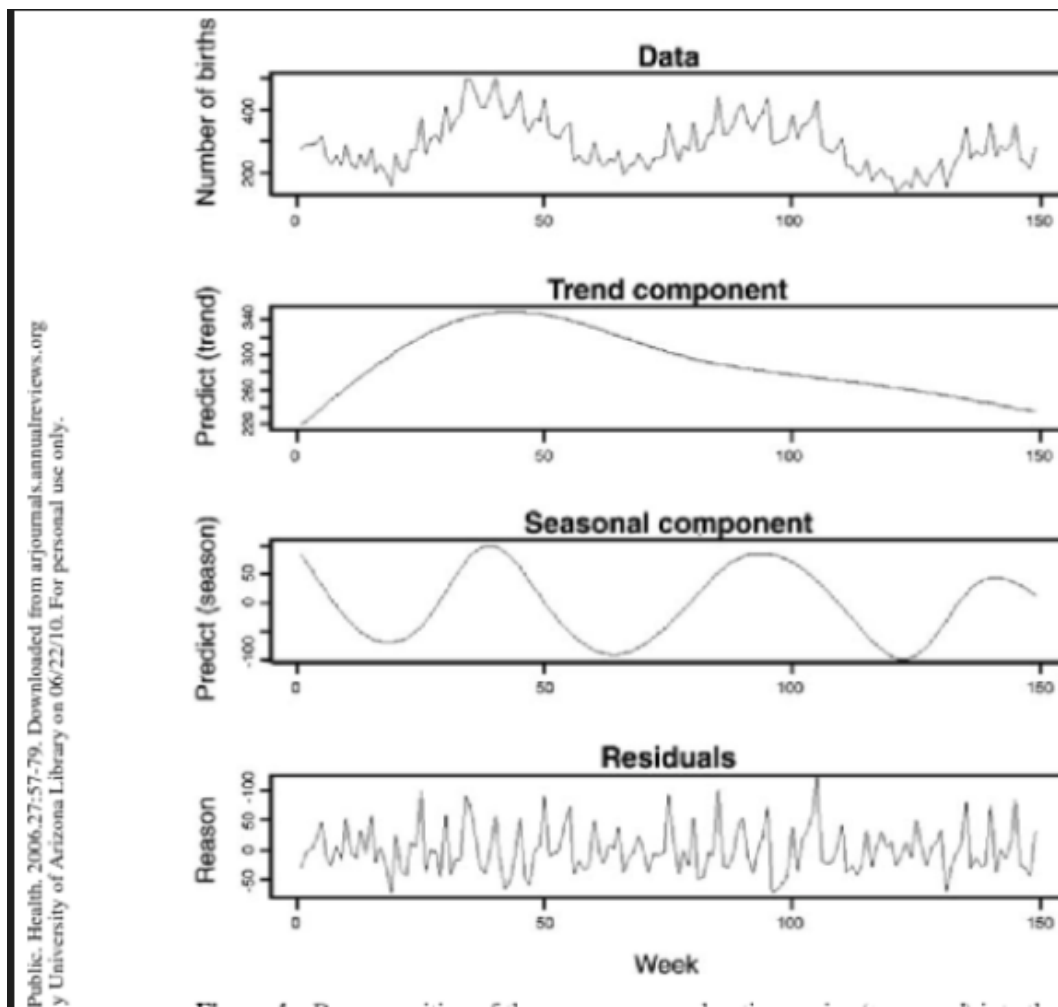


Figure 5: alt text

## 8.2 Werken met een date als index

- Bij tijdreeksen is het cruciaal om een datumkolom als index te gebruiken. Zo kan pandas automatisch met tijdstructuren werken.
- Het kan zowel op deze manier

```
# Read the data from CSV, parse dates and set the index
wounded_data = pd.read_csv(
    'https://raw.githubusercontent.com/HoGentTIN/dsai-en-labs/main/data/
    number_of_heavily_wounded_car_accidents_BE.txt',
    delimiter = "\t",
    parse_dates=['date']).set_index(['date'])

# Geef expliciet aan dat het maandelijks is (MS = Month Start)
wounded_data.index = pd.DatetimeIndex(wounded_data.index, freq='MS')

wounded_data.head()
```

- Als deze manier via datetime:

```
dfemployees['recording_date'] = pd.to_datetime(dfemployees['recording_date'])
dfemployees = dfemployees.set_index('recording_date')
```

## 8.3 Time Series Models

### 8.3.1 Tabellen

- Om te bepalen welk model het is, moeten we de data even plotten
- Dit kan je op twee manieren doen:

```
wounded_ts = pd.Series(
    data=np.array(wounded_data['number_of_heavily_wounded']),
    index=wounded_data.index
)
wounded_ts.plot(marker='o');
```

```
wounded_data.plot( y='number_of_heavily_wounded', figsize=[10,5]);
```

### 8.3.2 Maken van voorspellingen

- We plotten eerst de volledige data is

```
number_of_samples = len(wounded_data['number_of_heavily_wounded'])

a, b = np.polyfit(np.arange(0, number_of_samples), wounded_data['
    number_of_heavily_wounded'].values, 1)

# plot the time series and the regression line
plt.plot(np.arange(0, number_of_samples), wounded_data['
    number_of_heavily_wounded'])
plt.axline((0, b), slope=a)
```

- Voorspellingen maken met deze regressielijn:

```
# voorspelling januari 2024
print(f'voorspelling aantal zwaargewonden in januari 2024 = {a *
    number_of_samples + b}')

# februari 2024 (+1 maand)
print(f'voorspelling aantal zwaargewonden in februari 2024 = {a * (
    number_of_samples + 1) + b}')

# maand 2024 (+2 maand)
print(f'voorspelling aantal zwaargewonden in maart 2024 = {a * (
    number_of_samples + 2) + b}')
```

- Regressielijn met seaborn:

```
sns.regplot(
    x=np.arange(0, number_of_samples),
    y=wounded_data['number_of_heavily_wounded'],
    line_kws={'color': 'b'},
    scatter_kws={'color': 'g'});
```

## 8.4 Moving average

- Hulpplot om te tonen wat de trend en seasonal component van de plot is:

```
from statsmodels.tsa.seasonal import seasonal_decompose

# remove other columns (e.g. SES, DES, ...)
```

```
data = wounded_data[['number_of_heavily_wounded']]

wounded_decomposed = seasonal_decompose(data, model='additive')
wounded_decomposed.plot();
```

#### 8.4.1 Simple Moving Average

- Met deze functies kijken we zoveel waarden in de toekomst (SMA = Simple Moving Average)
- Indien je None waarden hebt voeg dan de parameter `min_periods=1` toe in `rolling`.
- Voortschrijdend gemiddelde over een vast aantal vorige maanden.

```
wounded_data['SMA3'] = wounded_data['number_of_heavily_wounded'].rolling(3).
    mean()
wounded_data['SMA5'] = wounded_data['number_of_heavily_wounded'].rolling(5).
    mean()
wounded_data['SMA10'] = wounded_data['number_of_heavily_wounded'].rolling(10).
    mean()
wounded_data
```

- En dit plotten we zo:

```
wounded_data.plot(
    y=['number_of_heavily_wounded', 'SMA3', 'SMA5', 'SMA10'],
    figsize=[10,5]);
```

#### 8.4.2 Exponential Moving Average (EMA) of Simple Exponential Smoothing (SES)

- Pandas heeft een ingebouwde functie om dit te berekenen
- Hier krijgt de meest recente waarneming meer gewicht dan oudere.

```
wounded_data['EMA_0.1'] = wounded_data['number_of_heavily_wounded'].ewm(alpha
    =.1, adjust=False).mean()
wounded_data['EMA_0.5'] = wounded_data['number_of_heavily_wounded'].ewm(alpha
    =.5, adjust=False).mean()

wounded_data.plot(y=['number_of_heavily_wounded', 'EMA_0.1', 'EMA_0.5'],
    figsize=[10,5]);
```

- SES berekenen met Holt Winters (zelfde effect, meer controle)

```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing

# Train the model
data_ses = SimpleExpSmoothing(wounded_data['number_of_heavily_wounded']).fit(
    smoothing_level=0.1, optimized=False)

# Add fitted values to the dataframe
wounded_data['SES_predicted_level'] = data_ses.level
wounded_data.head(20)
```

- En dan platten we weer:

```
wounded_data.plot(y=['number_of_heavily_wounded', 'SES'], figsize=[10,5]);
```

- Hierdoor kunnen we voorspellingen doen via `forecast`:



```
data_ses_fcast = data_ses.forecast(12)

wounded_data.plot(y=['number_of_heavily_wounded', 'SES'], figsize=[10,5])
data_ses_fcast.plot(marker='.', legend=True, label='Forecast');
```

### 8.4.3 Double exponential Smoothing (Holt's methode)

- Voegt een trendcomponent toe aan SES:

```
from statsmodels.tsa.api import Holt

data_des = Holt(wounded_data['number_of_heavily_wounded']).fit(
    smoothing_level=.1,
    smoothing_trend=.2,
    optimized=False)

wounded_data['DES'] = data_des.level
wounded_data.tail()
```

- Dit plotten we weer

```
wounded_data.plot(y=['number_of_heavily_wounded', 'DES'], figsize=[10,5]);
```

- Dit kan wederom worden geforcast:

```
data_des_fcast = data_des.forecast(12)

# Plot observations, fitted values and forecast
wounded_data['number_of_heavily_wounded'].plot(marker='o', legend=True)
wounded_data['DES'].plot(legend=True, label='DES fitted values', figsize
    =[10,5]);
data_ses_fcast.plot(marker='.', legend=True, label='Forecast SES');
data_des_fcast.plot(marker='.', legend=True, label='Forecast DES');

wounded_data['DES_prediction_next_month'] = data_des.level + data_des.trend #
    predicted value for next month => m = 1
wounded_data['DES_fittedvalues'] = data_des.fittedvalues
wounded_data.tail()
```

### 8.4.4 Triple exponential Smoothing (Holt-Winters methode)

- Voegt ook seizoenspatronen toe:

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

train = wounded_data.number_of_heavily_wounded[:-12]
test = wounded_data.number_of_heavily_wounded[-12:]

wounded_hw = ExponentialSmoothing(train, trend='add', seasonal='add',
    seasonal_periods=12, freq='MS').fit()

train.plot(legend=True, label='train')
test.plot(legend=True, label='test')
(wounded_hw.level + wounded_hw.season).plot(legend=True, label='fitted');
```

```
wounded_predicted = wounded_hw.forecast(12)

train.plot(legend=True, label='train')
wounded_hw.fittedvalues.plot(legend=True, label='fitted')

test.plot(legend=True, label='test')
wounded_predicted.plot(legend=True, label='predicted')

plt.title('Train, test, fitted & predicted values using Holt-Winters');
```

```
test.plot(legend=True, label='test');
wounded_predicted.plot(legend=True, label='predicted');
plt.title('Test data vs. predicted values');
```

```
# Predicted value by forecast():
print(wounded_predicted.iloc[0])

# Computed manually from model internals:
(wounded_hw.level.iloc[-1] + wounded_hw.trend.iloc[-1]) + wounded_hw.season.
    iloc[-12]
```

- Evaluatie:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error

print(f'MAE = {mean_absolute_error(test, wounded_predicted)}')
print(f'MSE = {mean_squared_error(test, wounded_predicted)}')
print(f'Square root MSE = {np.sqrt(mean_squared_error(test, wounded_predicted))}')
print(f'stdev = {wounded_data.number_of_heavily_wounded.std()}')
```

## 8.5 Decomposing a Time Series

- Om elke component (trend, seizoen, residu) apart te analyseren:

```
from statsmodels.tsa.seasonal import seasonal_decompose

# remove other columns (e.g. SES, DES, ...)
data = wounded_data[['number_of_heavily_wounded']]

wounded_decomposed = seasonal_decompose(data, model='additive')
wounded_decomposed.plot();
```

## 8.6 Samenvatting

Methode	Wat het doet	Sterkte
SMA	Gemiddelde over vaste venster	Simpel, maar gevoelig voor outliers
EMA / SES	Meer gewicht aan recente waarden	Sneller reagerend op veranderingen
Holt (DES)	Trend erbij	Voorspelt stijging/daling
Holt-Winters	Trend + seizoen	Beste voor complexe patronen

Vraag	Antwoord	Model(s) die passen
Is er een <b>duidelijke trend</b> ?	Ja	Holt (Double Exponential Smoothing), Regressie
Is er een <b>seizoenspatroon</b> ?	Ja	Holt-Winters (Triple Exponential Smoothing)
Zijn er <b>geen duidelijke patronen</b> ?	Ja	Simple/Exponential Moving Average, SES
Is de data <b>sterk fluctuerend</b> ?	Ja	EMA of SES (geeft gewicht aan recente waarden)
Is het <b>ruis met kleine schommelingen</b> ?	Ja	SMA of SES (voor gladstrijken)

## 9 Samenvatting Data Science en AI

### 9.1 Confidence intervals

- Confidence interval is om te berekenen of de steekproef data in de populatieparameter ligt
- Een confidence interval (CI) geeft een schatting van waar het werkelijke populatiegemiddelde zich bevindt, op basis van een steekproef.
- Bijvoorbeeld: een 95% CI van [321.3, 327.9] betekent dat we 95% zeker zijn dat het echte gemiddelde ergens in dat interval zit.

Situatie	Gebruik	Verdeling	Reden
- <b>Kleine steekproef</b> ( $n < 30$ ) en onbekend	t-verdel ing	stats.t	De t-verdeling compenseert de extra onzekerheid bij kleine samples.
- <b>Grote steekproef</b> ( $n \geq 30$ ) of bekend	z-verdel ing	stats.norm	Bij grote steekproeven geldt de Centrale Limietstelling, dus gebruik je de normale verdeling.
- Populatiestandaardafwijking <b>bekend</b> , ongeacht $n$	z-verdel ing	stats.norm	Want er is geen extra onzekerheid van schatting van .

#### 9.1.1 Confidence interval for a large samples mean

- Berekenen van de confidence interval voor grote samples

```
import math
# Step 1.
m = 324.6          # Sample mean
s = 2.5            # Population standard deviation
n = 45             # Sample size
alpha = .05        # 1 - alpha is the confidence level

# Step 2.
z = stats.norm.isf(alpha/2)
print("z-score: %.5f" % z)

# Step 3.
lo = m - z * s / np.sqrt(n)
hi = m + z * s / np.sqrt(n)
print("Confidence interval: [%.4f, %.4f]" % (lo, hi))
```

#### 9.1.2 Confidence Interval for a small samples mean

- Berekenen van de confidence interval voor kleine samples

```
import math
# Step 1.
m = 324.6          # Sample mean
s = 2.5            # Population standard deviation
n = 15             # Sample size
alpha = .05        # 1 - alpha is the confidence level

# Step 2.
t = stats.t.isf(alpha/2, df = n - 1)
```

```
print("t-score: %.5f" % t)

# Step 3.
lo = m - t * s / np.sqrt(n)
hi = m + t * s / np.sqrt(n)
print("Confidence interval: [%.4f, %.4f]" % (lo, hi))
```

## 9.2 Z-tests

- Je gebruikt dit wanneer de standaardvariatie **WEL** vermeld is of wanneer de sample size  $\geq 30$

Function	Purpose
<code>stats.t.pdf(x, df=d)</code>	Probability density for $x$
<code>stats.t.cdf(x, df=d)</code>	Left-tail probability ( $< x$ )
<code>stats.t.sf(x, df=d)</code>	Right-tail probability ( $> x$ )
<code>stats.t.isf(1-p, df=d)</code>	p% of observations are expected to be lower than this value

### 9.2.1 Right tailed Z-test

- Dit gebruik je wanneer het **gemiddelde** van de *populatie* **KLEINER** is dan het **gemiddelde** van de *sample*
- Variabelen:

```
n = 40          # Sample size
mu = 351.2      # (Hypothetical) population mean
sigma = 4.1     # Population standard deviation (assumed to be known)
m_sample = 352.8 # Sample mean
s_sample = sigma / math.sqrt(n)
```

- p-waarde:

```
alpha = 0.05 # Significance level chosen by the researcher
p = stats.norm.sf(m_sample, loc=mu, scale=sigma/np.sqrt(n))

print("p-value: %.5f" % p)
if(p < alpha):
    print("p < alpha: reject H0")
else:
    print("p > alpha: do not reject H0")
```

### 9.2.2 Left tailed Z-test

- Dit gebruik je wanneer het **gemiddelde** van de *populatie* **GROTER** is dan het **gemiddelde** van de *sample*
- Variabelen:

```
n = 40          # Sample size
mu = 351.2      # (Hypothetical) population mean
sigma = 4.1     # Population standard deviation (assumed to be known)
m_sample = 352.8 # Sample mean
alpha = 0.05    # Significance level
s_sample = sigma / math.sqrt(n)
```

- p-waarde:

```
p = stats.norm.cdf(m_sample, loc=mu, scale=sigma/np.sqrt(n))

print("p-value: %.5f" % p)
if(p < alpha):
    print("p < a, reject H0")
else:
    print("p > a, do not reject H0")
```

### 9.2.3 Two tailed Z-test

- Je kijkt niet specifiek of iets kleiner of groter is, maar je kijkt of er gewoon een verschil is
- Variabelen:

```
accidents = [4, 3, 4, 0, 5, 2, 3, 0, 4, 4, 6, 0, 1, 4, 0, 3, 0, 5]
n = len(accidents)
mu = 3                                # Populatiegemiddelde ()
s = 1.73                             # Populatiestandaarddeviatie () d.w.z. de
    spreiding van het aantal ongevallen in de populatie
    # s is te berekenen via: np.std(accidents, ddof
    =1)
m_sample = np.mean(accidents)        # (sample mean)
alpha = 0.05                         # Significantieniveau ()
```

- p-waarde:

```
p = stats.norm.sf(m_sample, loc=mu, scale=sigma/np.sqrt(n))
print("p-waarde: %.5f" % p)
if(p < alpha/2):
    print("p < alpha/2, reject H0")
else:
    print("p > alpha/2, do not reject H0")
```

## 9.3 Student's T-test

- Je gebruikt dit wanneer de sample < 30 en standaarddeviatie van POPULATIE is niet gekend
- Variabelen:

```
accidents = [4, 3, 4, 0, 5, 2, 3, 0, 4, 4, 6, 0, 1, 4, 0, 3, 0, 5]
n = len(accidents)
mu = 3
s = 1.73
m_sample = np.mean(accidents)
alpha = 0.05
```

- p-waarde:

```
p = stats.t.cdf(m_sample, loc = mu, scale = s / math.sqrt(n), df = n - 1)
print(f"p-value: {p}")
if(p < alpha):
    print("p < alpha: reject H0")
else:
    print("p > alpha: do not reject H0")
```

- Verkorte notatie voor deze specifieke test:

```
t_stat, p_val = stats.ttest_1samp(a=study_times, popmean=mu, alternative='less')
print("Sample mean      : %.3f" % np.mean(study_times))
print("t-score         : %.3f" % t_stat)
print(f"p-value        : {p_val}")
```

## 9.4 Chi-squared en Cramr's V

- Je gebruikt dit wanneer je wilt vergelijken of er een verband is tussen 2 of meerdere 'attributen' van een dataset.
- Variabelen:

```
observed = pd.crosstab(rlanders.Survey, rlanders.Gender)
row_sums = observed.sum(axis=1)
col_sums = observed.sum()
n = row_sums.sum()

print(row_sums)
print(col_sums)
print(f'Number of observations: {n}')
```

- Expected variabelen:

```
expected = np.outer(row_sums, col_sums) / n
exp_row_sums = np.sum(expected, axis=1)
exp_col_sums = np.sum(expected, axis=0)

print(f'Row totals      : {exp_row_sums}')
print(f'Column totals: {exp_col_sums}')
print(f'Observations    : {exp_col_sums.sum()}')
```

- Chi-squared zelf berekenen:

```
chi_squared = diffs.values.sum()
print(' %.3f' % chi_squared)
```

- Cramr's V berekenen en interpreteren:

```
dof = min(observed.shape) - 1
cramers_v = np.sqrt(chi_squared / (dof * n))
print(cramers_v)

# Of de verkorte notatie
# stats.contingency.association(observed, method='cramer')
```

- Interpretatie Cramr's V: | Cramr's V | Interpretation | | :— | :— | | 0 | No association | | 0.1 | Weak association | | 0.25 | Moderate association | | 0.50 | Strong association | | 0.75 | Very strong association | | 1 | Complete association |

## 9.5 Goodness of fit test

- Hier kijk je of een bepaald attribuut zijn soorten een bepaald percentage aantal voorkomt
- Variabelen:

```
types = ['mutant', 'human', 'alien', 'god', 'demon']
observed = np.array([ 127, 75, 98, 27, 73])
expected_p = np.array([ .35, .17, .23, .08, .17])
```

- Vragen beantwoorden:

```
alpha = 0.05 # Significance level
n = sum(observed) # Sample size
k = len(observed) # Number of categories
dof = k - 1 # Degrees of freedom
expected = expected_p * n # Expected absolute frequencies in the sample
g = stats.chi2.isf(alpha, df=dof) # Critical value

# Goodness-of-fit-test in Python:
chi2, p = stats.chisquare(f_obs=observed, f_exp=expected)

print("Significance level = %.2f" % alpha)
print("Sample size n = %d" % n)
print("k = %d; df = %d" % (k, dof))
print("Chi-squared = %.4f" % chi2)
print("Critical value g = %.4f" % g)
print("p-value p = %.4f" % p)
```

- Kortere notatie:

```
alpha=0.01 # significance level
dof=len(families)-1 # degrees of freedom
g = stats.chi2.isf(alpha, df=dof) # Critical value
# Perform Chi-squared test, calculate and p
chi2, p = stats.chisquare(f_obs=families.observed, f_exp=families.expected)

print("Chi-squared = %.4f" % chi2)
print("Critical value g = %.4f" % g)
print("p-value p = %f" % p)
```

## 9.6 T-test for independent variables

- Bij de T-testen **vergelijk** je altijd twee kolommen
- In deze test ga je kijken naar gegevensverzameling van **groep 1** en daarna gegevens van een **compleet andere groep** met een *ander experiment*
- Variabelen:

```
# Sample:
control = np.array([91, 87, 99, 77, 88, 91])
treatment = np.array([101, 110, 103, 93, 99, 104])
# Visualization:
sns.boxplot(
    data=pd.DataFrame({'control': control, 'treatment': treatment}),
    orient='h');
```

- T-test berekenen:

```
stats.ttest_ind(a=control, b=treatment, alternative='less', equal_var=False)
```



## 9.7 T-test for paired samples

- Bij de T-testen **vergelijk** je altijd twee kolommen
- In deze test ga je kijken naar gegevensverzameling van **groep 1** en daarna gegevens van **dezelfde groep** voor nog een *ander experiment*
- Variabelen berekenen:

```
# Measurements
regular = np.array([16, 20, 21, 22, 23, 22, 27, 25, 27, 28])
additives = np.array([19, 22, 24, 24, 25, 25, 26, 26, 28, 32])
# Visualization
sns.boxplot(
    data={'Regular': regular, 'Additives': additives},
    orient='h');
```

- T-test berekenen:

```
stats.ttest_rel(regular, additives, alternative='less')
```

## 9.8 Cohen's d

- Dit trekt de gemiddelde waarden van de beide groepen van de T-test for paired samples af en geeft het verschil weer.
- Functie:

```
def cohen_d(a, b):
    na = len(a)
    nb = len(b)
    pooled_sd = np.sqrt( ((na-1) * np.var(a, ddof=1) +
                          (nb-1) * np.var(b, ddof=1)) / (na + nb - 2) )
    return (np.mean(b) - np.mean(a)) / pooled_sd

cohen_d(regular, additives)
```

## 9.9 Simple Moving Avarage

- Wanneer je bv. je dataset werkt met maanden kan je dat via **Simple Moving Avarage** via de `rolling`-methode enkel maanden vooruit trekken
- Velden aan dataset toevoegen:

```
wounded_data['SMA3'] = wounded_data['number_of_heavily_wounded'].rolling(3).
    mean()
wounded_data['SMA5'] = wounded_data['number_of_heavily_wounded'].rolling(5).
    mean()
wounded_data['SMA10'] = wounded_data['number_of_heavily_wounded'].rolling(10).
    mean()
wounded_data
```

- Dit kan je dan plotten:

```
wounded_data.plot(
    y=['number_of_heavily_wounded', 'SMA3', 'SMA5', 'SMA10'],
    figsize=[10,5]);
```

## 9.10 Exponential Moving Avarage

- Velden aan dataset toevoegen:

```
wounded_data['EMA_0.1'] = wounded_data['number_of_heavily_wounded'].ewm(alpha
    =.1, adjust=False).mean()
wounded_data['EMA_0.5'] = wounded_data['number_of_heavily_wounded'].ewm(alpha
    =.5, adjust=False).mean()

wounded_data.plot(y=['number_of_heavily_wounded', 'EMA_0.1', 'EMA_0.5'],
    figsize=[10,5]);
```

- Data klaarmaken/trainen om uiteindelijk te forecasten:

```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing

# Train the model
data_ses = SimpleExpSmoothing(wounded_data['number_of_heavily_wounded']).fit(
    smoothing_level=0.1, optimized=False)

# Add fitted values to the dataframe
wounded_data['SES_predicted_level'] = data_ses.level
wounded_data.head(20)
```

- Forecasten:

```
data_ses_fcast = data_ses.forecast(12)

wounded_data.plot(y=['number_of_heavily_wounded', 'SES'], figsize=[10,5])
data_ses_fcast.plot(marker='.', legend=True, label='Forecast');
```

- Waarde van de volgorde rij voorspellen

```
wounded_data['SES_predicted_value'] = data_ses.fittedvalues
wounded_data.tail()
```

## 9.11 Double exponential smoothing

- Dit gebruik je wanneer **duidelijke trend** zich voordoet in de plot
- Data maken/trainen dat je gaat forecasten:

```
from statsmodels.tsa.api import Holt

data_des = Holt(wounded_data['number_of_heavily_wounded']).fit(
    smoothing_level=.1,
    smoothing_trend=.2,
    optimized=False)

wounded_data['DES'] = data_des.level
wounded_data.tail()
```

- Forecasten uitvoeren:

```
data_des_fcast = data_des.forecast(12)

# Plot observations, fitted values and forecast
wounded_data['number_of_heavily_wounded'].plot(marker='o', legend=True)
```

```
wounded_data['DES'].plot(legend=True, label='DES fitted values', figsize
    =[10,5]);
data_ses_fcast.plot(marker='.', legend=True, label='Forecast SES');
data_des_fcast.plot(marker='.', legend=True, label='Forecast DES');
```

## 9.12 Triple Exponential Smoothing

- Dit gebruik je zodra er een **duidelijke trend** zich voordoet in de plot alsook alsook een **seasonal trend**
- Data klaarmaken/trainen om te forcasten:

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

train = wounded_data.number_of_heavily_wounded[:-12]
test = wounded_data.number_of_heavily_wounded[-12:]

wounded_hw = ExponentialSmoothing(train, trend='add', seasonal='add',
    seasonal_periods=12, freq='MS').fit()

train.plot(legend=True, label='train')
test.plot(legend=True, label='test')
(wounded_hw.level + wounded_hw.season).plot(legend=True, label='fitted');
```

- Forcasten uitvoeren:

```
wounded_predicted = wounded_hw.forecast(12)

train.plot(legend=True, label='train')
wounded_hw.fittedvalues.plot(legend=True, label='fitted')

test.plot(legend=True, label='test')
wounded_predicted.plot(legend=True, label='predicted')

plt.title('Train, test, fitted & predicted values using Holt-Winters');

# Enkel het verschil tussen data en forcast zien
# test.plot(legend=True, label='test');
# wounded_predicted.plot(legend=True, label='predicted');
# plt.title('Test data vs. predicted values');
```

- Overige methoden:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error

print(f'MAE = {mean_absolute_error(test,wounded_predicted)}')
print(f'MSE = {mean_squared_error(test,wounded_predicted)}')
print(f'Vierkantswortel MSE = {np.sqrt(mean_squared_error(test,
    wounded_predicted))}')
print(f'stdev = {wounded_data.number_of_heavily_wounded.std()}')
```