H1 - Samples

```
# Data inlezen (kijken welke sep dat het is, meestal ; of ,) -> default is ,
data = pd.read_csv("data.csv", sep=";")

# Data bekijken
data.head() # eerste 5 rijen

# Properties van de data bekijken
# Print data information
print("Data information:")
print(data.info()) # prints column names and data types

# Print number of rows and columns
print("\nNumber of rows and columns:")
print("Rows:", len(data)) # prints number of rows
print("Columns:", len(data.columns)) # prints number of columns

# Print shape of data
print("\nShape of data:")
print(data.shape) # prints number of rows and columns

# Print data types
print("\nData types:")
print(data.dtypes) # prints data types of columns

# het aantal unieke waarden in een kolom
data["kolomnaam"].unique()

# hoeveel van elk datatype zit er in
# volledige dataset
data.dtypes.value_counts()
# per kolom
data["kolomnaam"].value_counts()

# indexen
data.index # geeft de indexen weer
data.set_index("kolomnaam", inplace=True) # zet de kolom als index
```

Kwalitatieve variabelen

```
# Kwalitatieve variabelen moeten omgezet worden naar een category
data["kolomnaam"] = data["kolomnaam"].astype("category")

# Kwalitatieve variabelen omzetten naar een category met een bepaalde volgorde -> ordinal variab
# bv. een rating van 1 tot 5
# maak een lijst aan met de volgorde
rating_order = ["1", "2", "3", "4", "5"]
# maak een CategoricalDtype aan met de volgorde
rating_type = CategoricalDtype(categories=rating_order, ordered=True)
# zet de kolom om naar een category met de volgorde
data["kolomnaam"] = data["kolomnaam"].astype(rating_type)
# deze volgorde wordt gebruikt bij het plotten van de data
```

Selecteren van data

```
# Selecteren van kolommen
data["kolomnaam"] # geeft de kolom terug
data.kolomnaam # geeft de kolom terug
data[["kolomnaam1", "kolomnaam2"]] # geeft de kolommen terug in een dataframe

# Selecteren van rijen
data.iloc[0] # geeft de eerste rij terug -> tellen vanaf 0
data.iloc[0:5] # geeft de eerste 5 rijen terug -> tellen vanaf 0 -> exclusief de laatste index

# Query's
data[data["kolomnaam"] == "waarde"] # geeft alle rijen terug waar de kolomnaam de waarde heeft
#of
data.query("kolomnaam == 'waarde'")

# query's voor bepaalde kolomen
data[(data["kolomnaam1"] == "waarde1") & (data["kolomnaam2"] == "waarde2")][["kolomnaam1", "kolc
```

Droppen van data

```
# Droppen van kolommen
data.drop("kolomnaam", axis="columns", inplace=True) # axis=1 -> kolom, axis=0 -> rij
of
data = data.drop("kolomnaam", axis="columns")

# veel lege waardes in een kolom?
data.dropna() #dropt elke rij waar er een lege waarde in zit -> niet aan te raden
data.dropna(how="all") #dropt elke rij waar alle waardes leeg zijn

# legen waardes vervangen door een waarde
data["kolomnaam"].fillna("waarde", inplace=True)
```

Creëren van nieuwe kolommen

```
# Creëren van nieuwe kolommen
data["nieuwecol"] = #iets van data of een berekening

# mappen van waardes
map_dict = {"waarde1": "nieuwewaarde1", "waarde2": "nieuwewaarde2"}
data["nieuwecol"] = data["kolomnaam"].map(map_dict)

# kan ook met functie
def functie(x):
    if x == "waarde1":
        return "nieuwewaarde1"
    elif x == "waarde2":
        return "nieuwewaarde2"
    else:
        return "waarde3"

data["nieuwecol"] = data["kolomnaam"].map(functie)
```

H2 - Analyse van 1 variabele

Kwalitatieve variabelen

```
# barchart
sns.catplot(x="kolomnaam", kind="count", data=data) # count -> telt het aantal waardes per categ
# of
sns.countplot(x="kolomnaam", data=data)

# centrality measures
data.mode() # geeft de modus terug -> meest voorkomende waarde
data["kolomnaam"].mode() # geeft de modus terug -> meest voorkomende waarde
data.describe() # geeft een overzicht van de data -> count, mean, std, min, max, 25%, 50%, 75%
```

Kwantitatieve variabelen

```
# histogram

# Boxplot -> geeft de 5 getallen weer -> min, 25%, 50%, 75%, max
sns.boxplot(data=data, x="kolomnaam") # x -> kolomnaam, y -> waarde
# of violinplot -> geeft de distributie weer
sns.violinplot(data=data, x="kolomnaam") # x -> kolomnaam, y -> waarde
# of kernel density plot (kde) -> geeft de distributie weer in 1 curve
sns.kdeplot(x = data["kolomnaam"])

# combineren histogram en density plot -> historgram met distributiecruve
sns.distplot(x = data["kolomnaam"], kde=True) # histogram + density plot

# centrality and dispersion measures
## Mean, st and friends
print(f"mean: {data['kolomnaam'].mean()}")
print(f"Standard deviation: {data['kolomnaam'].std()}") # Pay attention: n-1 in the denominator
print(f"Variance: {data['kolomnaam'].var()}") # Pay attention: n-1 in the denominator
print(f"skewness: {data['kolomnaam'].skew()}")
print(f"kurtosis: {data['kolomnaam'].kurtosis()}")

##median & friends
print(f"minimum: {data['kolomnaam'].min()}")
print(f"median: {data['kolomnaam'].median()}")
print(f"maximum: {data['kolomnaam'].max()}")

print(f"percentile 25%: {data['kolomnaam'].quantile(0.25)}")
print(f"percentile 50%: {data['kolomnaam'].quantile(0.5)}")
print(f"percentile 75%: {data['kolomnaam'].quantile(0.75)}")

print(f"iqr (interquartile range): {data['kolomnaam'].quantile(0.75) - data['kolomnaam'].quantil
print(f"range: {data['kolomnaam'].max() - data['kolomnaam'].min()}")

# of ge zijt slim en doet
data.describe()
```

Formule voor de standaard deviatie

```
# BIJ SAMPLE GEBRUIK JE N - 1
# BIJ POPULATIE GEBRUIK JE N
# dit omdat je zo een betere schatting hebt van de populatie

# Bij pandas word standaard de sample gebruikt
# Bij numpy word standaard de populatie gebruikt
print(f"Pandas uses ddof=1 by default: {data['col'].std()}") # ddof -> delta degrees of freedom
print(f"Numpy  uses ddof=0 by default: {np.std(data['col'])}")

#pandas
print(f"Standard deviation population: {data['col'].std(ddof=0)}")
print(f"Standard deviation sample    : {data['col'].std()}")

#numpy
print(f"Standard deviation population: {np.std(a)}")
print(f"Standard deviation sample    : {np.std(a, ddof=1)}")
```

H3

Discrete random variable -> een variabele die een beperkt aantal waardes kan aannemen
Continuous random variable -> een variabele die een oneindig aantal waardes kan aannemen

- Kans type 1 fout = alpha

Central Limit Theorem

- De som van een groot aantal onafhankelijke random variabelen is ongeveer normaal verdeeld
- Hoe groter de steekproef, hoe beter de benadering
- Hier is de sigma ALTIJD bij sample = **standaardafwijking / sqrt(n)**

The normal distribution

Plotting density function of a normal distribution

```
# STANDAARD NORMAL DISTRIBUTIE -> mean = 0, std = 1
# Take 100 values for the X-axis, between -4 and 4, evenly spaced
x = np.linspace(-4, +4, num=101)
y = stats.norm.pdf(x, 0, 1)
# Plot the probability density function (pdf) for these X-values
plt.plot(x, y)

# voor een normale distributie met mean = 5 en std = 1.5 -> de vorm van de grafiek is identiek g
m = 5    # Gemiddelde
s = 1.5  # Standaardafwijking
x = np.linspace(m - 4 * s, m + 4 * s, num=201)
plt.plot(x, stats.norm.pdf(x, loc=m, scale=s))
```

Plotting histogram of a sample with theoretical probability density

```
# Histogram of the sample
plt.hist(sample, bins=20, density=True, label="Histogram of the sample")
# of
sns.distplot(sample, kde=True, label="Histogram of the sample")
```

Probability distribution in the normal distribution

Student t-distribution in Python

Import scipy.stats

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 $P(X < x)$
stats.t.sf(x, df=d)	Right-tail probability $P(X > x)$
stats.t.isf(1-p, df=d)	p% of observations are expected to be lower than this value

Normal distribution in Python

Python functions

Import scipy.stats

For a normal distribution with mean m and standard deviation s:

Function	Purpose
stats.norm.pdf(x, loc=m, scale=s)	Probability density at x
stats.norm.cdf(x, loc=m, scale=s)	Left-tail probability $P(X < x)$
stats.norm.sf(x, loc=m, scale=s)	Right-tail probability $P(X > x)$
stats.norm.isf(1-p, loc=m, scale=s)	p% of observations are expected to be lower than result

More examples of probability calculations

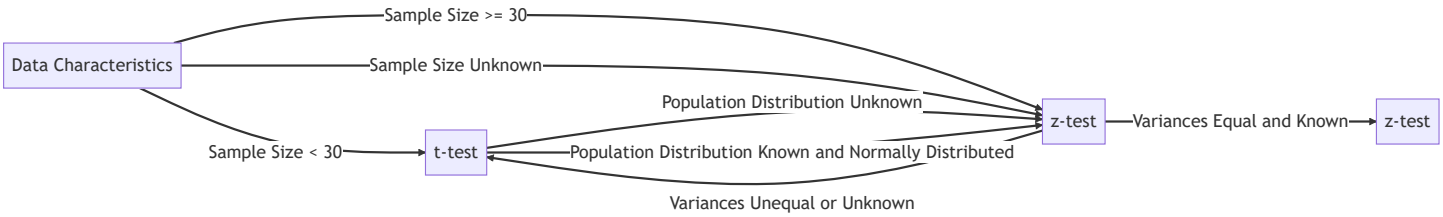
confidence intervals

- confidence interval -large sample -> een interval waarin de parameter met een bepaalde kans ligt

```
# 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))
```

- confidence interval -small sample -> students t test

```
# Step 1.
m = 5.2      # Sample mean
s = 1.5      # Sample (!) standard deviation
n = 15       # Sample size
alpha = .05  # 1 - alpha is the confidence level

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

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



Requirements z-test:

- Random sample
- Sample groot genoeg (n >= 30)
  - als normaal verdeeld is is sample size niet relevant
- normaal verdeeld

- populatie standaard deviatie is gekend

indien 1 van deze niet voldaan is gebruik je de t-test en deze normaal verdeeld is

Z-test

right-tailed

```
## RIGHT TAIL Z-TEST
#Step 1: formulate the null and alternative hypotheses
#- H0: μ = 100
#- H1: μ > 100

#Step 2: specify the significance level
# Properties of the sample:
n = 50      # sample size
sm = 20.2  # sample mean
s = 0.4    # population standard deviation (assumed to be known)
a = 0.05    # significance level (chosen by the researcher)
m0 = 20.0    # hypothetical population mean (H0)

# Plotting the sample distribution
# Gauss-curve plot:
# X-values
dist_x = np.linspace(m0 - 4 * s/np.sqrt(n), m0 + 4 * s/np.sqrt(n), num=201)
# Y-values for the Gauss curve
dist_y = stats.norm.pdf(dist_x, m0, s/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(m0, color="orange", lw=2)
# Show the sample mean with a red line
dplot.axvline(sm, color="red")

#Step 3: compute the test statistic (red line in the plot)
# Hier is dat 20.2

#Step 4:
## method 1
# Determine the $p$-value and reject $H_0$ if $p < \alpha$.
#The $p$-value is the probability, if the null hypothesis is true, to obtain
# a value for the test statistic that is at least as extreme as the
# observed value
p = stats.norm.sf(sm, loc=m0, scale=s/np.sqrt(n))
print("p-value: %.5f" % p)
if(p < a):
    print("p < a: reject H0")
else:
    print("p > a: do not reject H0")

## method 2
# An alternative method is to determine the critical region, i.e. the set of all values for the
```

```
# The boundary of that area is called the critical value $g$. To the left of it you can't reject
g = stats.norm.isf(a, loc = m0, scale = s / np.sqrt(n))
print("Critical value g ≃ %.3f" % g)
if (sm < g):
    print("sample mean = %.3f < g = %.3f: do not reject H0" % (sm, g))
else:
    print("sample mean = %.3f > g = %.3f: reject H0" % (sm, g))


# Step 5
# We can conclude that if we assume that  $H_0$  is true, the probability to draw a sample from
```

left-tailed

```
## LEFT TAIL Z-TEST
#Step 1: formulate the null and alternative hypotheses
#- H0: μ = 100
#- H1: μ < 100

#Step 2: specify the significance level
# Properties of the sample:
n = 50      # sample size
sm = 19.94  # sample mean
s = 0.4    # population standard deviation (assumed to be known)
a = 0.05    # significance level (chosen by the researcher)
m0 = 20.0    # hypothetical population mean (H0)

# Plotting the sample distribution
# Gauss-curve plot:
# X-values
dist_x = np.linspace(m0 - 4 * s/np.sqrt(n), m0 + 4 * s/np.sqrt(n), num=201)
# Y-values for the Gauss curve
dist_y = stats.norm.pdf(dist_x, m0, s/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(m0, color="orange", lw=2)
# Show the sample mean with a red line
dplot.axvline(sm, color="red")

#Step 3: compute the test statistic (red line in the plot)
# Hier is dat 19.94

#Step 4:
## method 1
# Determine the $p$-value and reject $H_0$ if $p < \alpha$.
#The $p$-value is the probability, if the null hypothesis is true, to obtain
#a value for the test statistic that is at least as extreme as the
# observed value
p = stats.norm.cdf(sm, loc=m0, scale=s/np.sqrt(n))
print("p-value: %.5f" % p)
if(p < a):
    print("p < a: reject H0")
else:
    print("p > a: do not reject H0")

## method 2
# An alternative method is to determine the critical region, i.e. the set of all values for the
# The boundary of that area is called the critical value $g$. To the right of it you can't rejec
g = stats.norm.isf(1-a, loc = m0, scale = s / np.sqrt(n))
print("Critical value g ≃ %.3f" % g)
```

```
if (sm > g):
    print("sample mean = %.3f > g = %.3f: do not reject H0" % (sm, g))
else:
    print("sample mean = %.3f < g = %.3f: reject H0" % (sm, g))


# Step 5
#  We can conclude that there is not enough evidence to reject the
#  null hypothesis.
```

two-tailed

```
## TWo tailed Z-TEST
#Step 1: formulate the null and alternative hypotheses
#- H0: μ = 100
#- H1: μ != 100

#Step 2: specify the significance level
# Properties of the sample:
n = 50      # sample size
sm = 19.94  # sample mean
s = 0.4    # population standard deviation (assumed to be known)
a = 0.05    # significance level (chosen by the researcher)
m0 = 20.0    # hypothetical population mean (H0)



#Step 3: compute the test statistic (red line in the plot)
# Hier is dat 19.94

#Step 4:
## method 1
# Calculate the $p$-value and reject $H_0$ if $p < \alpha/2$ (why do we divide by 2?).
p = stats.norm.cdf(sm, loc=m0, scale=s/np.sqrt(n))
print("p-value: %.5f" % p)
if(p < a):
    print("p < a: reject H0")
else:
    print("p > a: do not reject H0")

## method 2
# In this case, we have two critical values: $g_1$ on the left of the mean and $g_2$ on the righ
g1 = stats.norm.isf(1-a/2, loc = m0, scale = s / np.sqrt(n))
g2 = stats.norm.isf(a/2, loc = m0, scale = s / np.sqrt(n))

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

# Plotting the sample distribution
# Gauss-curve
# X-values
dist_x = np.linspace(m0 - 4 * s/np.sqrt(n), m0 + 4 * s/np.sqrt(n), num=201)
# Y-values
dist_y = stats.norm.pdf(dist_x, loc=m0, scale=s/np.sqrt(n))
fig, dplot = plt.subplots(1, 1)
# Plot
dplot.plot(dist_x, dist_y)
```

```
# Hypothetical population mean in orange
dplot.axvline(m0, color="orange", lw=2)
# Sample mean in red
dplot.axvline(sm, color="red")
acc_x = np.linspace(g1, g2, num=101)
acc_y = stats.norm.pdf(acc_x, loc=m0, scale=s/np.sqrt(n))
# Fill the acceptance region in light blue
dplot.fill_between(acc_x, 0, acc_y, color='lightblue')

# Step 5
#  So if we do not make a priori statement whether the actual population mean is either smaller
```

t-test

right-tailed

```
# Right tailed t test
#Step 1: formulate the null and alternative hypotheses
#- H0: μ = 100
#- H1: μ > 100

#Step 2: specify the significance level
# Properties of the sample:
n = 50      # sample size
sm = 20.2  # sample mean
s = 0.4    # sample standard deviation (assumed to be known)
a = 0.05    # significance level (chosen by the researcher)
m0 = 20.0    # hypothetical population mean (H0)

# Plotting the sample distribution
# Gauss-curve plot:
# X-values
dist_x = np.linspace(m0 - 4 * s/np.sqrt(n), m0 + 4 * s/np.sqrt(n), num=201)
# Y-values for the Gauss curve
dist_y = stats.t.pdf(dist_x, loc = m0,scale = s/np.sqrt(n), df = n-1)
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(m0, color="orange", lw=2)
# Show the sample mean with a red line
dplot.axvline(sm, color="red")

#Step 3: compute the test statistic (red line in the plot)
# Hier is dat: VUL IN

#Step 4:
## method 1
# Determine the $p$-value and reject $H_0$ if $p < \alpha$.
#The $p$-value is the probability, if the null hypothesis is true, to obtain
# a value for the test statistic that is at least as extreme as the
# observed value
p = stats.t.sf(sm, loc=m0, scale=s/np.sqrt(n), df=n-1)
print("p-value: %.5f" % p)
if(p < a):
    print("p < a: reject H0")
else:
    print("p > a: do not reject H0")

## method 2
# An alternative method is to determine the critical region, i.e. the set of all values for the
```

```
# The boundary of that area is called the critical value $g$. To the left of it you can't reject
g = stats.t.isf(a, loc = m0, scale = s / np.sqrt(n), df = n-1)
print("Critical value g ≃ %.3f" % g)
if (sm < g):
    print("sample mean = %.3f < g = %.3f: do not reject H0" % (sm, g))
else:
    print("sample mean = %.3f > g = %.3f: reject H0" % (sm, g))


# Step 5
# We can conclude that if we assume that  $H_0$  is true, the probability to draw a sample from
```

left-tailed

```
## LEFT TAIL t-TEST
#Step 1: formulate the null and alternative hypotheses
#- H0: μ = 100
#- H1: μ < 100

#Step 2: specify the significance level
# Properties of the sample:
n = 50      # sample size
sm = 19.94  # sample mean
s = 0.4    # sample standard deviation (assumed to be known)
a = 0.05    # significance level (chosen by the researcher)
m0 = 20.0    # hypothetical population mean (H0)

# Plotting the sample distribution
# Gauss-curve plot:
# X-values
dist_x = np.linspace(m0 - 4 * s/np.sqrt(n), m0 + 4 * s/np.sqrt(n), num=201)
# Y-values for the Gauss curve
dist_y = stats.t.pdf(dist_x, loc=  m0, scale= s/np.sqrt(n), df = n-1)
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(m0, color="orange", lw=2)
# Show the sample mean with a red line
dplot.axvline(sm, color="red")

#Step 3: compute the test statistic (red line in the plot)
# Hier is dat 19.94

#Step 4:
## method 1
# Determine the $p$-value and reject $H_0$ if $p < \alpha$.
#The $p$-value is the probability, if the null hypothesis is true, to obtain
#a value for the test statistic that is at least as extreme as the
# observed value
p = stats.t.cdf(sm, loc=m0, scale=s/np.sqrt(n), df=n-1)
print("p-value: %.5f" % p)
if(p < a):
    print("p < a: reject H0")
else:
    print("p > a: do not reject H0")

## method 2
# An alternative method is to determine the critical region, i.e. the set of all values for the
# The boundary of that area is called the critical value $g$. To the right of it you can't rejec
g = stats.t.isf(1-a, loc = m0, scale = s / np.sqrt(n), df=n-1)
print("Critical value g ≃ %.3f" % g)
```

```
if (sm > g):
    print("sample mean = %.3f > g = %.3f: do not reject H0" % (sm, g))
else:
    print("sample mean = %.3f < g = %.3f: reject H0" % (sm, g))


# Step 5
#  We can conclude that there is not enough evidence to reject the
#  null hypothesis.
```

two-tailed

```
## TWo tailed Z-TEST
#Step 1: formulate the null and alternative hypotheses
#- H0: μ = 100
#- H1: μ != 100

#Step 2: specify the significance level
# Properties of the sample:
n = 50      # sample size
sm = 19.94  # sample mean
s = 0.4    # sample standard deviation (assumed to be known)
a = 0.05    # significance level (chosen by the researcher)
m0 = 20.0    # hypothetical population mean (H0)


#Step 3: compute the test statistic (red line in the plot)
# Hier is dat 19.94

#Step 4:
## method 1
# Calculate the $p$-value and reject $H_0$ if $p < \alpha/2$ (why do we divide by 2?).
p = stats.t.cdf(sm, loc=m0, scale=s/np.sqrt(n), df=n-1)
print("p-value: %.5f" % p)
if(p < a):
    print("p < a: reject H0")
else:
    print("p > a: do not reject H0")

## method 2
# In this case, we have two critical values: $g_1$ on the left of the mean and $g_2$ on the righ
g1 = stats.t.isf(1-a/2, loc = m0, scale = s / np.sqrt(n), df = n-1)
g2 = stats.t.isf(a/2, loc = m0, scale = s / np.sqrt(n), df = n-1)

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

# Plotting the sample distribution
# Gauss-curve
# X-values
dist_x = np.linspace(m0 - 4 * s/np.sqrt(n), m0 + 4 * s/np.sqrt(n), num=201)
# Y-values
dist_y = stats.t.pdf(dist_x, loc=m0, 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(m0, color="orange", lw=2)
# Sample mean in red
dplot.axvline(sm, color="red")
acc_x = np.linspace(g1, g2, num=101)
acc_y = stats.t.pdf(acc_x, loc=m0, scale=s/np.sqrt(n),  df=n-1)
# Fill the acceptance region in light blue
dplot.fill_between(acc_x, 0, acc_y, color='lightblue')

# Step 5
#  So if we do not make a priori statement whether the actual population mean is either smaller
```

H4 -> 2 kwalitatieve variabelen

Contingency tables and visualisation techniques

```
# Contingency table -> oppassen met de margins -> als je de margins erbij zet dan krijg je een e
pd.crosstab(data.x, data.y, margins=True, margins_name="Total")
# Contingency table -> zonder de margins
pd.crosstab(data.x, data.y)
```

Clustered bar chart

```
# Clustered bar chart
# hue is de opsplitsing van de data
sns.catplot(x="x", hue="y", data=data, kind="count")
```

Stacked bar chart

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

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

Chi-squared and Cramér's V

Chi-squared test

1. Formulate the hypotheses:
    - : There is no association between the variables (the differences between observed and expected values are small)
    - : There is an association between the variables (the differences are large)
2. Choose significance level
3. Calculate the value of the test statistic in the sample (here:).
4. Use one of the following methods (based on the degrees of freedom):
    1. Determine critical value so
    2. Calculate the -value

5. Draw a conclusion based on the outcome:
    1. : do not reject ; : reject
    2. : do not reject ; : reject

```
observed = pd.crosstab(rlanders.Survey, rlanders.Gender)
chi2, p, df, expected = stats.chi2_contingency(observed)

print("Chi-squared : %.4f" % chi2)
print("Degrees of freedom: %d" % df)
print("P-value : %.4f" % p)


alpha = .05
dimensions = observed.shape
dof = (dimensions[0]-1) * (dimensions[1]-1)

print("Chi-squared        : %.4f" % chi_squared)
print("Degrees of freedom : %d" % dof)

# Calculate critical value
g = stats.chi2.isf(alpha, df = dof)
print("Critical value     : %.4f" % g)

# Calculate p-value
p = stats.chi2.sf(chi_squared, df=dof)
print("p-value            : %.4f" % p)
```

Cramér's V

- is a formula that normalises to a value between 0 and 1 that is independent of the table size.

Cramér'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

```
# Cramér's V
dof = min(observed.shape) - 1
cramers_v = np.sqrt(chi_squared / (dof * n))
print(cramers_v)
```

Goodness of fit test

- controleren of sample representatief is voor de populatie

1. Formulate the hypotheses:
    - : The sample is representative of the population, i.e. the frequency of each class within the sample corresponds well to that in the population.
    - : The sample is *not* representative of the population, i.e. the differences with the expected frequencies are too large.
2. Choose significance level
3. Calculate the value of the test statistic in the sample (here:).
4. Use one of the following methods (based on the degrees of freedom with the number of categories in the sample):
    1. Determine critical value so
    2. Calculate the -value
5. Draw a conclusion based on the outcome:
    1. : do not reject ; : reject
    2. : do not reject ; : reject

```
observed = np.array([127, 75, 98, 27, 73])
expected_p = np.array([.35, .17, .23, .08, .17])
expected = expected_p * sum(observed)
chi2, p = stats.chisquare(f_obs=observed, f_exp=expected)

print("χ² = %.4f" % chi2)
print("p = %.4f" % p)
```

Standardised residuals

- kijken of uw sample overrepresentatief is voor een bepaalde groep of niet
- na chi-squared test

```python
# Standardised residuals -> heb een functie gemaakt ervoor xd
# zorg dat expected_p in de dataframe zit
# zorg dat observed in de dataframe zit
# zorg dat expected in de dataframe zit
def calculate_stdres(contingency_table):
    """
    Calculates the standardized residuals for a contingency table.

    Args:
    contingency_table (pd.DataFrame): A contingency table with observed and expected frequencies

    Returns:
    pd.DataFrame: The contingency table with added column for standardized residuals.
    """
    # Calculate the standardized residuals
    contingency_table['stdres'] = (contingency_table['observed'] - contingency_table['expected']

    return contingency_table
```

Cochran's rule

- Chi-quadraat test enkel juiste resultaat als er voldoende data is
  - Contingency table -> 2x2
  - Alle expected values > 1
  - minstens 20% expected values > 5

H5 -> 1 kalitatieve variabele en 1 kwantitatieve variabelen

The t-test for independent samples (two-sample t-test)

- vergelijken van het gemiddelde van 2 groepen (niet perse even groot)
- gemiddelde van 2 verschillende groepen
- Groep met placebo en groep met medicijn

```
# alternative = 'less' -> one-tailed test
#  `alternative='less'` indicates that we want to test for the alternative hypothesis that the m
# alternative = 'two-sided' -> two-tailed test
# alternative = 'greater' -> one-tailed test
control = np.array([91, 87, 99, 77, 88, 91])
treatment = np.array([101, 110, 103, 93, 99, 104])

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

The t-test for paired samples (paired t-test)

- vergelijken van dingen op dezelfde groep bv
- Voorbeelden
- Voorbeeld zelfde auto met verschillende soorten benzine

```
    Before and after measurements: Paired samples are often used when you want to compare the measurements of t

    Matched pairs: Paired samples analysis is useful when you have a natural pairing or matching between the ob

    Repeated measures: Paired samples can be used when you have multiple measurements taken on the same subject
```

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

# Paired t-test with ttest_rel() -> vergeet niet alternative='less' of 'greater' of 'two-sided'
stats.ttest_rel(before, after, alternative='less')
```

Cohen's d

Effect size is another metric to express the magnitude of the difference between two groups. Several definitions of effect size exist, but one of the most commonly used is *Cohen's* .

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

cohen_d(before, after)
```

H6 -> 2 kwantitatieve variabelen

Visualisatie

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

regressie

```
from sklearn.linear_model import LinearRegression

x = data.x.values.reshape(-1,1)
y = data['y']

model = LinearRegression().fit(x, y)

print(f"Regression line: ŷ = {model.intercept_:.2f} + {model.coef_[0]:.2f} x")

# Predict y values corresponding to x
model.predict([[valueOpX]])[0]
```

covariantie + R + R^2

correlation coefficient and the coefficient of determination.

		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

		Explained variance	Linear relation
> .95	> .9	> 90%	exceptionally strong

```
cor = np.corrcoef(cats.Hwt, cats.Bwt)[0][1]
print(f"R = { cor }")
print(f"R² = {cor ** 2}")
```

H7 Time Series Analysis

Time whatttt?

- Analyse van data die geordend is in tijd over een bepaalde periode
  - bv Supermarkt: hoeveelheid verkochte producten per dag
  - bv Aandelen: prijs van een aandeel per dag
- Op deze data kunnen we dan voorspellingen maken
  - bv Supermarkt: hoeveelheid verkochte producten per dag in de toekomst
  - bv Aandelen: prijs van een aandeel per dag in de toekomst

Components of Time Series

- Trend: de algemene richting van de data
  - bv stijgend, dalend, constant
- Seasonality: herhaling van patronen in de data
  - bv. elke maandag is er een piek in de verkoop
- Noise: onregelmatigheden in de data
  - random variaties die niet te verklaren zijn door de trend of seasonality
- Cyclical: herhaling van patronen in de data die niet op een vaste tijdsinterval gebeuren

moving averages

simple moving average

```py
# simple moving average
data['SMA3'] = data['Close'].rolling(window=3).mean()
data['SMA5'] = data['Close'].rolling(window=5).mean()
data['SMA10'] = data['Close'].rolling(window=10).mean()

# Simple moving average with shift -> to predict the future je zet ze een rij naar onder op de p
data['SMA3_forecast'] = data['Close'].rolling(3).mean().shift(1)
data['SMA5_forecast'] = data['Close'].rolling(5).mean().shift(1)
data['SMA10_forecast'] = data['Close'].rolling(10).mean().shift(1)
```

weighted moving average -> recentere data krijgt meer gewicht

- exponential moving average
  If is close to 0, then 1 − is close to 1 and the weights decrease very slowly. In other words, observations from the distant past continue to have a large influence on the next forecast. This means that the graph of the forecasts will be relatively smooth, just as with a large span in the moving averages method. But if is close to 1, the weights decrease rapidly, and only very recent observations have much influence on the next forecast.

```
# exponential moving average
# alpha
data['EMA_0.1'] = data['Close'].ewm(alpha=.1, adjust=False).mean()
data['EMA_0.5'] = data['Close'].ewm(alpha=.5, adjust=False).mean()
```

Exponential smoothing

single exponential smoothing

-> exponential moving average (EMA)

-> geen trend of seasonality

```
# single exponential smoothing
# smoothing_level=0.1 -> alpha
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
data_ses = SimpleExpSmoothing(data['Close']).fit(smoothing_level=0.1, optimized=True)
data['SES'] = data_ses.fittedvalues
data.head()

data.plot(y=['Close',  'SES'], figsize=[10,5])

#Predicting the future
data_ses_fcast = data_ses.forecast(12)

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

double exponential smoothing

-> Holt's method

-> trend maar geen seasonality

```
# double exponential smoothing
from statsmodels.tsa.api import Holt

data_des = Holt(data['Close']).fit(smoothing_level=.1, smoothing_trend=.2)

data['DES'] = data_des.fittedvalues

data.plot(y=['Close',  'DES'], figsize=[10,5])

#Predicting the future
data_des_fcast = data_des.forecast(12)

data['number_of_heavily_wounded'].plot(marker='o', legend=True) # Observations
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')
```

triple exponential smoothing

-> Holt-Winters method

-> trend en seasonality

2 Soorten:

- additive
  - seasonality is constant

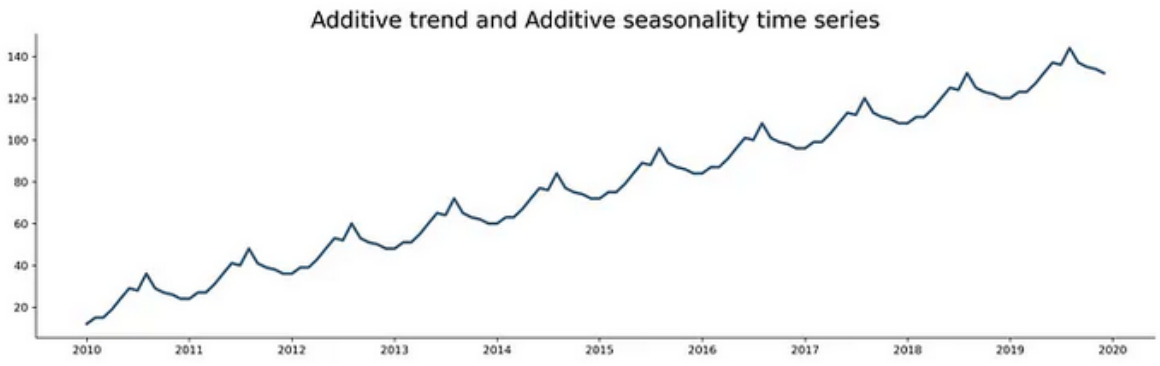


Image 3 — Additive trend and additive seasonality time series (image by author)

- multiplicative
  - seasonality is not constant

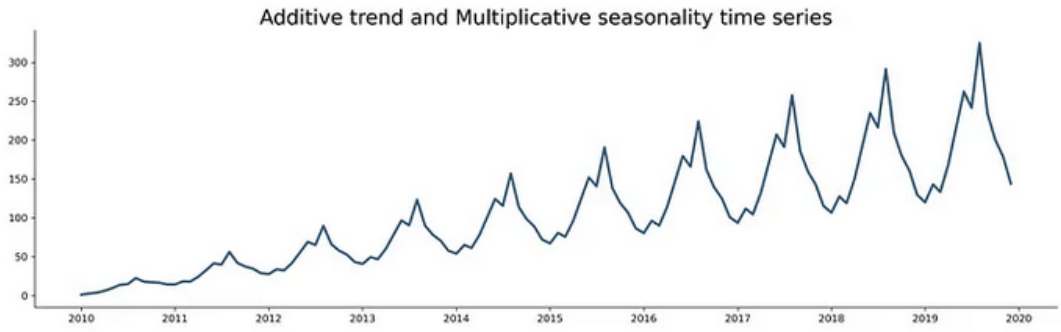


Image 4 — Additive trend and multiplicative seasonality time series (image by author)

```
# triple exponential smoothing
# je kunt de freq aanpassen naar bv D voor days of MS voor months
# additive
from statsmodels.tsa.holtwinters import ExponentialSmoothing

train = data.number_of_heavily_wounded
test = data.number_of_heavily_wounded

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

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

# multiplicative
from statsmodels.tsa.holtwinters import ExponentialSmoothing

train = data.number_of_heavily_wounded
test = data.number_of_heavily_wounded

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

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

#Predicting the future
model_predicted = model.forecast(12)

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

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

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

Model ìnternals

```
# Model internals: last estimate for level, trend and seasonal factors:
print(f'level: {wounded_hw.level[83]}')
print(f'trend: {wounded_hw.trend[83]}')
print(f'seasonal factor: {wounded_hw.season[72:84]}')
```