Gruppe 42, Statistikk Prosjekt 1a

EIRIK KALDHOL STRANDMAN

University of Agder, 2024
Faculty of Engineering and Science
Department of Engineering and Sciences

Oppgave 1

0.1 Oppgave Beskrivelse

Beliggenhetsmål: Begrunn ut alle tre beliggenhetsmål (typetall, median, gjennommsnitt), og avgjør hivlket som passer best for tilfellet.

0.1.1 (a)

Femunden FriFiskerForening (også kalt 4F) har følgende fordeling av medllemmer:

1. 24 er fra Drevsjø i Engerdal, med postnummer 2443
2. 6 er fra Ålesund, med postnummer 6020
3. 19 er fra Røros, med postnummer 7374
4. 1 er fra Bodø, med postnummer 8092

0.1.2 (b)

En klasse ingeniører med 50 sstudenter hadde 3 måneder etter endt utdannelse følgende fordeling av inntekten:

1. 5 stykker hadde ennå ikke funnet arbeid, og hadde en årsinntekt på 0 kroner.
2. 41 hadde årsinntekter på hhv, 340 000, 341 000, 342 000 ... opp til 380 000.
3. De fire siste tjente henholdsvis 613 00, 727 000, 958 000 og 70 millioner.

0.1.3 (c)

For en klasse på 100 var tilsvarende fordeling som følger:

1. 51 tjente ingenting.
2. Av de 49 restenrende tjente 24 stykker 312 000, mens 25 tjente 478 000.

0.2 Besvarelse

0.2.1 (a)

Typetallet er det hyppigeste tallet eller dataen. Siden datasettet er fire sorterte mengder, vet vi at den største mengden er typetallet. **Typetallet er da Drevsjø i Engderdal, med postnummer 2443**.

Median er den midderste verdien, vi starter med å sortere dataen i stigende rekkefølge:

1. 1 er fra Bodø, med postnummer 8092
2. 6 er fra Ålesund, med postnummer 6020
3. 19 er fra Røros, med postnummer 7374
4. 24 er fra Drevsjø i Engerdal, med postnummer 2443

Siden datasettet er et partall, blir medianen i teorien på indeks 2,5, men vi runder ned til 2. **Medianen er da Ålesund, med postnummer 6020.**

Gjennomsnitt er summen av målingene, delt på antallet, og er beskrevet med formelen:

$$S_x = \sum_{k=1}^{\infty} x_k \tag{1}$$

$$\bar{x} = \frac{S_x}{n} \tag{2}$$

- S_X = Sum av all målingene.
- k = Indeks til en måling.
- $\bar{x}$ = Snittet til datasettet
- n = Antall målinger eller mengde.

Gjennomsnittet blir da:

$$S_X = 24 + 6 + 19 + 1 = 50$$

$$\bar{x} = \frac{50}{4} = 12.5$$

Siden Røros er den nærmeste verdien, kan vi si at i snitt er meldemmer fra Røros.

I konklusjon, kan vi si at typetallet her beskriver best mengden medlemmer, siden de utgjør majoriteten.

0.2.2 (b)

I denne oppgaven er det et mer omfattende datasett. Får å gjøre livet enklere, bruker vi Python for å regne ut de ønskede verdiene. Først, setter vi opp en klasse for å largre datasett, filbaner og lignende.

Listing 1: Datasett klasse.

```
# Define data set class
class data_set:
  def __init__(self):
    self.path = None
    self.df_raw = None
    self.df_unique = None
    self.df_rising = None
    self.mode = None
    self.median = None
    self.n = None
    self.sum = None
    self.avg = None
```

Så importere vi dataen fra excel arket.

Listing 2: Importering av data fra Excel.

```
# Packages
import pandas as pd

# Engineer Class income
engineer_income = data_set()
engineer_income.path = "oblig_1a/data/ingeniør_inntekter.xlsx"
engineer_income.df_raw = pd.read_excel(engineer_income.path)
```

Her settes og en funksjon for å finne typetallet til datasettet. Siden typetallet er det mest gjenntagende tallet, må vi først sortere dataen etter unike verdier.

Listing 3: Datasett sorterting funksjon.

```
def count_unique_values(df, column_name, unique_values_colum_name, ...
    count_name):
    # Find unique values and their counts
    unique_values_counts = df[column_name].value_counts()

    # Create a new DataFrame with unique values and counts
    result_df = pd.DataFrame({
        unique_values_colum_name: unique_values_counts.index,
        count_name: unique_values_counts.values
    })

    return result_df
```

Koden i listing 3, tar inn et rå datasett og generer et nytt sortert datasett. Vi kan så mate den inn i den neste funksjonsen for å finne hvilke verdi som gjentar seg oftest:

Listing 4: Finner tallet som gjenntar seg oftest.

```
def get_largest_value(df, unique_values_colum_name, count_name):
    # Find the index of the largest count
    max_count_index = df[count_name].idxmax()
    # Locate value with the corresponding index
    mode = df.loc[max_count_index, unique_values_colum_name]

    return mode
```

Den komplette koden ser slik ut:

Listing 5: Oppgave 1b Typetall

```
###   Task 1b   ###
##  Import data  ##
# Engineer Class income
print("Engineering class income")
engineer_income = data_set()
engineer_income.path = "oblig_1a/data/ingeniør_inntekter.xlsx"
engineer_income.df_raw = pd.read_excel(engineer_income.path)
print("-----------------------------------------")
print(engineer_income.df_raw)
print("-----------------------------------------")


##  Find mode value  ##
# Create dataframe with unique value
engineer_income.df_unique = count_unique_values(engineer_income.df_raw, ...
    "Inntekt", "Unique_Values", "Counts")

# Locate mode value
mode_b_man = get_largest_value(engineer_income.df_unique, "Unique_Values", ...
    "Counts")

# Alternativly, you can use this function from Pandas:
engineer_income.mode = engineer_income.df_raw["Inntekt"].mode().iloc[0]

# Print out results
print(f"The mode is {mode_b_man}, found manually.")
print(f"The mode is {engineer_income.mode}, according to Panda.")
```

Fra koden over, får vi at typetallet for ingeniørklassen er en inntekt på 0 kr.

For å regne ut median verdien kan vi bruke følgende formler:

$$\tilde{x} = x_{\frac{n+1}{2}} \tag{3}$$

(dersom n er oddetall)

$$\tilde{x} = \frac{1}{2} \cdot (x_{\frac{n}{2}} + x_{\frac{n+2}{2}}) \tag{4}$$

(dersom n er partall)

- $\tilde{x}$ = Median verdi.
- x = Verdi i datasettet.
- n = Mengden data.

Listing 6: Finner median

```
def find_median(df, column):
   # Find the amount of data points.
   n = len(df[column]) # Get length of column

   # Check if "n" is a odd or even number
   if n % 2 == 0:
     # "n" is even
     median = (1/2) * ( df.at[(n / 2), column] + df.at[((n + 2) / 2), ...
         column]) # df.at[i, column] gets value at index i of the column.
   else:
     # "n" is odd
     median = df.at[((n + 1) / 2), column]

   return median
```

Her er den komplette koden, med sortering av datasettet.

Listing 7: Oppgave 1b median

```
## Find median value ##
# Sort data in ascending order
engineer_income.df_rising = ...
    engineer_income.df_raw.sort_values(by='Inntekt', ascending=True)

# Locate median
median_b_man = find_median(engineer_income.df_rising, "Inntekt")

# Alternativly, you can use this function from Pandas:
engineer_income.median = engineer_income.df_raw['Inntekt'].median()

# Print out results
print(f"The median is {median_b_man}, found manually.")
print(f"The median is {engineer_income.median}, according to Panda.")
```

Fra koden over, får vi at medianen for ingeniørklassen er en inntekt på 359 500 kr.

For å regne ut gjennomsnittet, er formel 2 brukt til å lage følgende Python funksjon:

Listing 8: Oppgave 1b Gjennomsnitt

```
## Find average value  ##
# Find the number of elements
engineer_income.n = len(engineer_income.df_raw["Inntekt"])

# Find sum of elements
engineer_income.sum = engineer_income.df_raw["Inntekt"].sum()

# Find mean value
avg_b_man = engineer_income.sum / engineer_income.n

# Alternativly, you can use this function from Pandas:
engineer_income.avg = engineer_income.df_raw["Inntekt"].mean()

# Print out results
print(f"The average is {avg_b_man}, found manually.")
print(f"The average is {engineer_income.avg}, according to Panda.")
```

Fra koden over, får vi at gjennomsnitt for ingeniørklassen er en inntekt på 1 741 169 kr.

I konklusjon, kan vi si at medianen her beskriver best lønnen til ingeniør klassen.

0.2.3 (c)

I denne oppgaven, bruker vi de innebygde statistikk verktøyene til Pandas, vi kan da rask og effektivt finne typetall, median og gjennomsnitt slikt:

Listing 9: Oppgave 1c

```
###   Task 1c   ###
# Import data
# Class income
print("Class income")
class_income = data_set()
class_income.path = "oblig_1a/data/klasse_inntekter.xlsx"
class_income.df_raw = pd.read_excel(class_income.path)
print("-----------------------------------------")
print(class_income.df_raw)
print("-----------------------------------------")

# Mode
class_income.mode = class_income.df_raw["Inntekt"].mode().iloc[0]

# Median
class_income.median = class_income.df_raw['Inntekt'].median()

# Mean
class_income.avg = class_income.df_raw["Inntekt"].mean()

# Print out results
print(f"The mode is {class_income.mode}")
print(f"The median is {class_income.median}")
print(f"The mean is {class_income.avg}")
```

Her er typetallet 0kr, medianen er 312 000kr og gjennomsnittet 263 333.33kr. Vi kan konkludere med at gjennomsnittet er best her.

Oppgave 8

0.3 Metode

0.3.1 Måleoppsett

I denne oppgaven, var 30 seigemenn av typen Laban og Brynhild målt. Seigemennene var holdt nede ved beinet, så strekt etter hode over en linjal til de slet. Oppsettet kan sees under i figur 1.

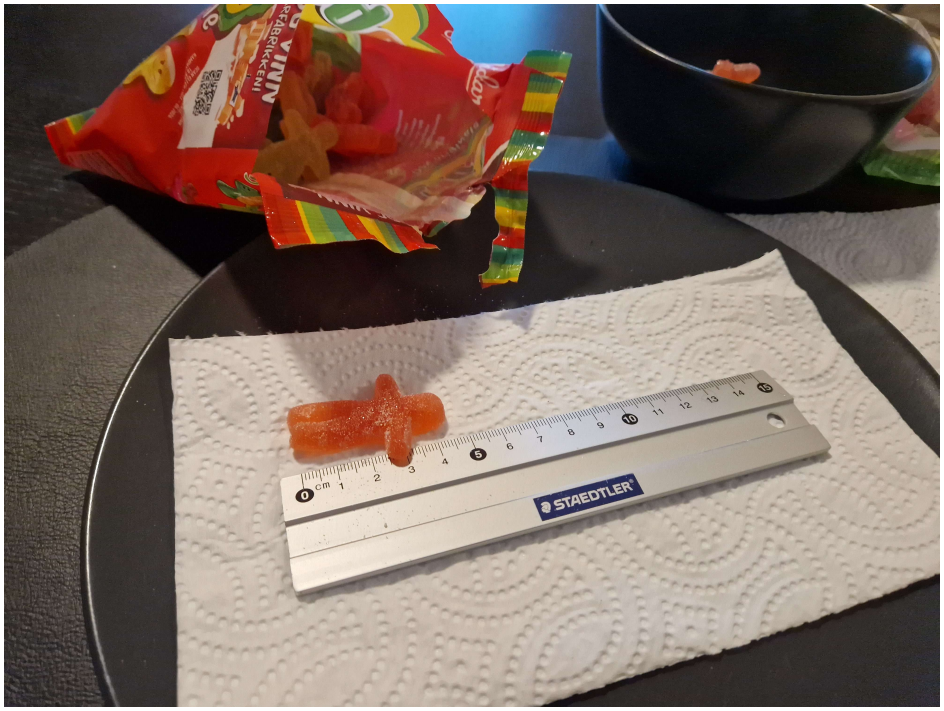

Figure 1: Måleoppsett for seigemen.

Målingen var målt til en presisjon på 5mm.

0.3.2 Statistikk Utregning

For data prosessering, var Python Pandas brukt, med følgende pakker:

Listing 10: Python pakker

```
# Packages
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.lines import Line2D
import os
```

Dataen var importert med følgende kode:

Listing 11: Import av data

```
###   Import data   ###
# Laban
laban = data_set()
laban.path = "oblig_1a/data/Laban42.csv"
laban.df_raw = pd.read_csv(laban.path)

# Brynhild
brynhild = data_set()
brynhild.path = "oblig_1a/data/Brynhild42.csv"
brynhild.df_raw = pd.read_csv(brynhild.path)
```

For utregning av typetall, median og gjennomsnitt var gjort med koden under:

Listing 12: Utregninger

```
###   Find mode, median, mean, standard deviation and population standard ...
    deviation   ###
# Mode
laban.mode = laban.df_raw["Midpunkt"].mode().iloc[0]
brynhild.mode = brynhild.df_raw["Midpunkt"].mode().iloc[0]

# Median
laban.median = laban.df_raw["Midpunkt"].median()
brynhild.median = brynhild.df_raw["Midpunkt"].median()

# Mean
laban.avg = laban.df_raw["Midpunkt"].mean()
brynhild.avg = brynhild.df_raw["Midpunkt"].mean()

# Standard deviation
laban.std = laban.df_raw["Midpunkt"].std()
brynhild.std = brynhild.df_raw["Midpunkt"].std()
print("Standard deviation:")
print(laban.std)

# Population standard deviation
laban.std_pop = laban.df_raw["Midpunkt"].std(ddof=0)
brynhild.std_pop = brynhild.df_raw["Midpunkt"].std(ddof=0)
```

For generering av tabeller, var følgende kode brukt:

Listing 13: Generering av tabeller.

```
###   Tables   ###
## Laban
# Create a frequency table for the "Midpunkt" column
laban.freq = laban.df_raw['Midpunkt'].value_counts()

# Create cumulative frequency table for "Midpunkt" colum
laban.cum_freq = ...
    laban.df_raw['Midpunkt'].value_counts().sort_index(ascending=False).cumsum()

# Set table up in correct format
laban_sorted = laban.df_raw.groupby(['Lower', 'Upper', 'Midpunkt', ...
    'Bredde']).size().reset_index(name='Antall')

# Sort the DataFrame by 'Frequency' in ascending order
laban_sorted = laban_sorted.sort_values('Midpunkt', ascending=False)

# Add a cumulative frequency column
laban_sorted['Kumulativt Antall'] = laban_sorted['Antall'].cumsum()


## Brynhild
# Create a frequency table for the "Midpunkt" column
brynhild.freq = brynhild.df_raw['Midpunkt'].value_counts()

# Create cumulative frequency table for "Midpunkt" colum
brynhild.cum_freq = ...
    brynhild.df_raw['Midpunkt'].value_counts().sort_index(ascending=False).cumsum()

# Set table up in correct format
brynhild_sorted = brynhild.df_raw.groupby(['Lower', 'Upper', 'Midpunkt', ...
    'Bredde']).size().reset_index(name='Antall')

# Sort the DataFrame by 'Antall' in ascending order
brynhild_sorted = brynhild_sorted.sort_values('Midpunkt', ascending=False)

# Add a cumulative frequency column
brynhild_sorted['Kumulativt Antall'] = brynhild_sorted['Antall'].cumsum()
```

0.4 Resultat

Type	Typetall	Median	Gjennomsnitt	Standard Avvik	Populasjon Standard Avvik
Laban	110.00	115.00	115.50	7.580	7.460
Brynhild	125.00	125.00	124.17	5.73	5.64

Table 1: Resultat for typetall, median, gjennomsnitt, standard avvik og populasjonsavvik.

	Lower	Upper	Midpunkt	Bredde	Antall	Kumulativt Antall
6	129.750000	130.250000	130	0.500000	2	2
5	124.750000	125.250000	125	0.500000	3	5
4	119.750000	120.250000	120	0.500000	8	13
3	114.750000	115.250000	115	0.500000	5	18
2	109.750000	110.250000	110	0.500000	8	26
1	104.750000	105.250000	105	0.500000	3	29
0	99.750000	100.250000	100	0.500000	1	30

Table 2: Frekvens og kumulative frekvenstabell for Laban. [mm]

	Lower	Upper	Midpunkt	Bredde	Antall	Kumulativt Antall
5	134.750000	135.250000	135	0.500000	1	1
4	129.750000	130.250000	130	0.500000	8	9
3	124.750000	125.250000	125	0.500000	11	20
2	119.750000	120.250000	120	0.500000	6	26
1	114.750000	115.250000	115	0.500000	3	29
0	109.750000	110.250000	110	0.500000	1	30

Table 3: Frekvens og kumulative frekvenstabell for Brynhild. [mm]

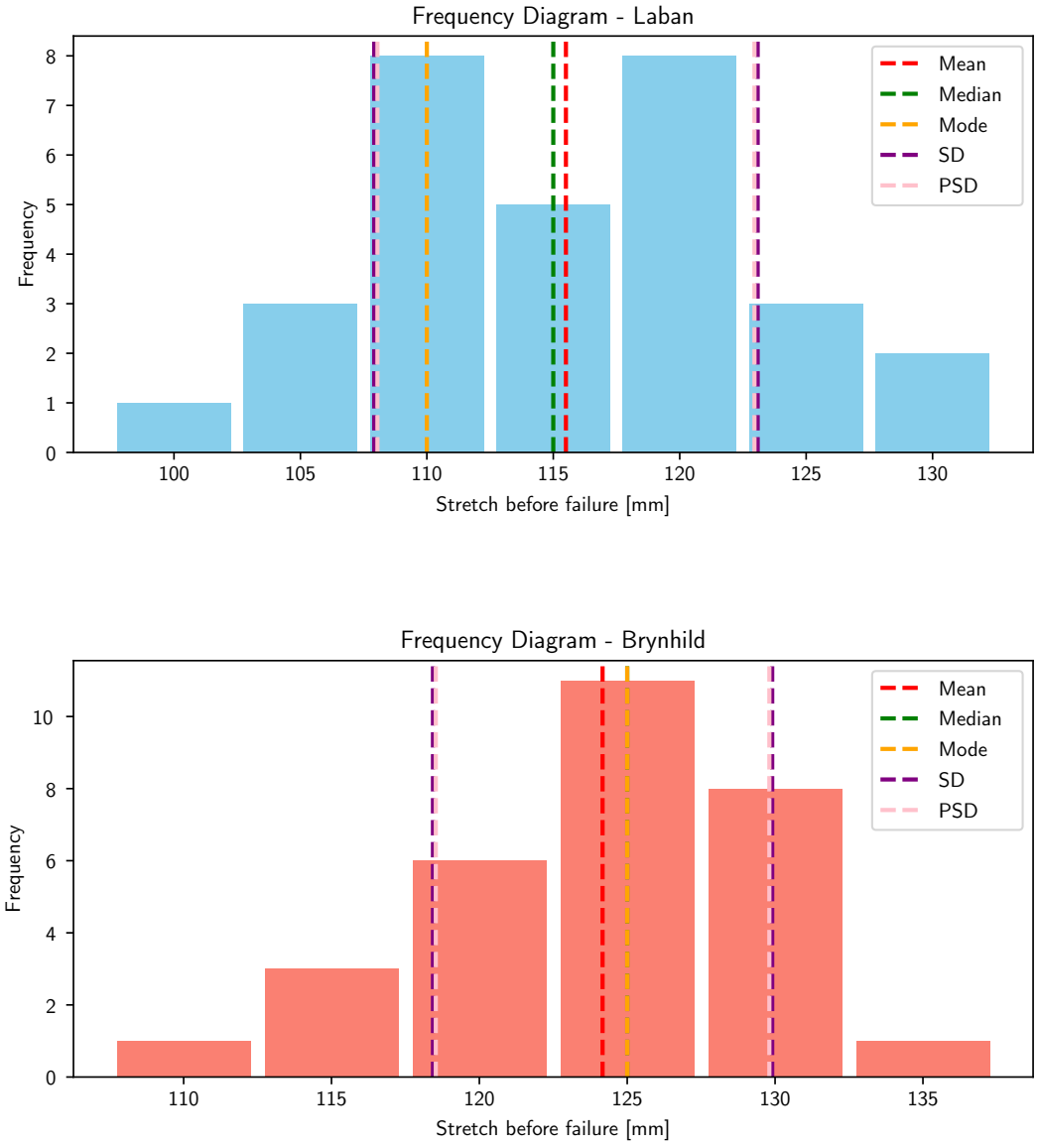

Figure 2: Frekvens diagram for Laban og Brynhild.

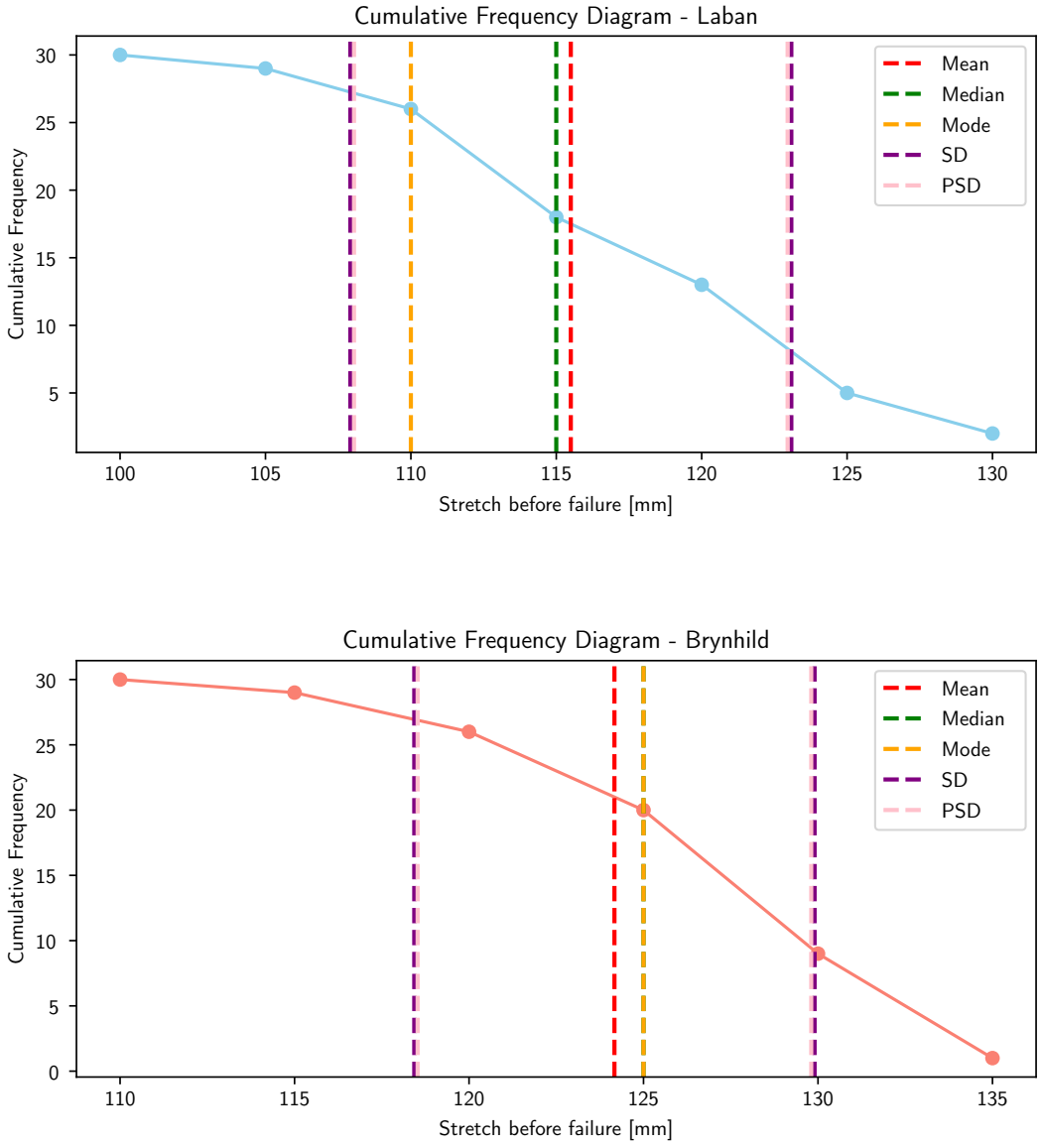

Figure 3: Kumulativ frekvens diagram for Laban og Brynhild.

Appendix A

Kode

Listing A.1: Komplett kode

```
###   Set-Up   ###
# Packages
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.lines import Line2D
import os


# Example using seaborn color palette
colors = sns.color_palette('pastel')


# Functions
def find_median(df, column):
   # Find the amount of data points.
   n = len(df[column]) # Get length of column

   # Check if "n" is a odd or even number
   if n % 2 == 0:
     # "n" is even
     median = (1/2) * ( df.at[(n / 2), column] + df.at[((n + 2) / 2), ...
         column]) # df.at[i, column] gets value at index i of the column.
   else:
     # "n" is odd
     median = df.at[((n + 1) / 2), column]

   return median

def count_unique_values(df, column_name, unique_values_colum_name, ...
    count_name):
    # Find unique values and their counts
    unique_values_counts = df[column_name].value_counts()

    # Create a new DataFrame with unique values and counts
    result_df = pd.DataFrame({
        unique_values_colum_name: unique_values_counts.index,
        count_name: unique_values_counts.values
    })

    return result_df

def get_largest_value(df, unique_values_colum_name, count_name):
    # Find the index of the largest count
```

```
    max_count_index = df[count_name].idxmax()
    # Locate value with the corresponding index
    mode = df.loc[max_count_index, unique_values_colum_name]

    return mode


# Define data set class
class data_set:
  def __init__(self):
    self.path = None
    self.df_raw = None
    self.df_unique = None
    self.df_rising = None
    self.mode = None
    self.median = None
    self.n = None
    self.sum = None
    self.avg = None
    self.freq = None
    self.cum_freq = None
    self.std = None
    self.std_pop = None




# Initialize classes and import data
# FriFiskerForening (4F)
FriFiskerForening = data_set()
FriFiskerForening.path = "oblig_1a/data/FriFiskerForening.xlsx"
FriFiskerForening.df_raw = pd.read_excel(FriFiskerForening.path)

# Laban
laban = data_set()
laban.path = "oblig_1a/data/Laban42.xlsx"
laban.df_raw = pd.read_excel(laban.path)

# Brynhild
brynhild = data_set()
brynhild.path = "oblig_1a/data/Brynhild42.xlsx"
brynhild.df_raw = pd.read_excel(brynhild.path)



####    Task 1    ####
###   Task 1b   ###
##  Import data  ##
# Engineer Class income
print("Engineering class income")
engineer_income = data_set()
engineer_income.path = "oblig_1a/data/ingeniør_inntekter.xlsx"
engineer_income.df_raw = pd.read_excel(engineer_income.path)
print("-----------------------------------------")
print(engineer_income.df_raw)
print("-----------------------------------------")


##  Find mode value  ##
# Create dataframe with unique value
```

```
engineer_income.df_unique = count_unique_values(engineer_income.df_raw, ...
    "Inntekt", "Unique_Values", "Counts")

# Locate mode value
mode_b_man = get_largest_value(engineer_income.df_unique, "Unique_Values", ...
    "Counts")

# Alternativly, you can use this function from Pandas:
engineer_income.mode = engineer_income.df_raw["Inntekt"].mode().iloc[0]

# Print out results
print(f"The mode is {mode_b_man}, found manually.")
print(f"The mode is {engineer_income.mode}, according to Panda.")


## Find median value ##
# Sort data in ascending order
engineer_income.df_rising = ...
    engineer_income.df_raw.sort_values(by='Inntekt', ascending=True)

# Locate median
median_b_man = find_median(engineer_income.df_rising, "Inntekt")

# Alternativly, you can use this function from Pandas:
engineer_income.median = engineer_income.df_raw['Inntekt'].median()

# Print out results
print(f"The median is {median_b_man}, found manually.")
print(f"The median is {engineer_income.median}, according to Panda.")


## Find average value  ##
# Find the number of elements
engineer_income.n = len(engineer_income.df_raw["Inntekt"])

# Find sum of elements
engineer_income.sum = engineer_income.df_raw["Inntekt"].sum()

# Find mean value
avg_b_man = engineer_income.sum / engineer_income.n

# Alternativly, you can use this function from Pandas:
engineer_income.avg = engineer_income.df_raw["Inntekt"].mean()

# Print out results
print(f"The average is {avg_b_man}, found manually.")
print(f"The average is {engineer_income.avg}, according to Panda.")
print("")
print("")




###   Task 1c   ###
# Import data
# Class income
print("Class income")
class_income = data_set()
class_income.path = "oblig_1a/data/klasse_inntekter.xlsx"
class_income.df_raw = pd.read_excel(class_income.path)
print("-----------------------------------------")
```

```
print(class_income.df_raw)
print("-----------------------------------------")

# Mode
class_income.mode = class_income.df_raw["Inntekt"].mode().iloc[0]

# Median
class_income.median = class_income.df_raw['Inntekt'].median()

# Mean
class_income.avg = class_income.df_raw["Inntekt"].mean()

# Print out results
print(f"The mode is {class_income.mode}")
print(f"The median is {class_income.median}")
print(f"The mean is {class_income.avg}")
```

```
####    Task 8    ####
###   Import data   ###
# Laban
laban = data_set()
laban.path = "oblig_1a/data/Laban42.csv"
laban.df_raw = pd.read_csv(laban.path)

# Brynhild
brynhild = data_set()
brynhild.path = "oblig_1a/data/Brynhild42.csv"
brynhild.df_raw = pd.read_csv(brynhild.path)



###   Find mode, median, mean, standard deviation and population standard ...
    deviation   ###
# Mode
laban.mode = laban.df_raw["Midpunkt"].mode().iloc[0]
brynhild.mode = brynhild.df_raw["Midpunkt"].mode().iloc[0]
print("Mode:")
print(brynhild.mode)

# Median
laban.median = laban.df_raw["Midpunkt"].median()
brynhild.median = brynhild.df_raw["Midpunkt"].median()
print("Median:")
print(brynhild.median)

# Mean
laban.avg = laban.df_raw["Midpunkt"].mean()
brynhild.avg = brynhild.df_raw["Midpunkt"].mean()
```

```
print("Avg:")
print(brynhild.avg)

# Standard deviation
laban.std = laban.df_raw["Midpunkt"].std()
brynhild.std = brynhild.df_raw["Midpunkt"].std()
print("Standard deviation:")
print(brynhild.std)

# Population standard deviation
laban.std_pop = laban.df_raw["Midpunkt"].std(ddof=0)
brynhild.std_pop = brynhild.df_raw["Midpunkt"].std(ddof=0)
print("Population standard deviation:")
print(brynhild.std_pop)



###   Tables   ###
## Laban
# Create a frequency table for the "Midpunkt" column
laban.freq = laban.df_raw['Midpunkt'].value_counts()

# Create cumulative frequency table for "Midpunkt" colum
laban.cum_freq = ...
    laban.df_raw['Midpunkt'].value_counts().sort_index(ascending=False).cumsum()

# Set table up in correct format
laban_sorted = laban.df_raw.groupby(['Lower', 'Upper', 'Midpunkt', ...
    'Bredde']).size().reset_index(name='Antall')

# Sort the DataFrame by 'Frequency' in ascending order
laban_sorted = laban_sorted.sort_values('Midpunkt', ascending=False)

# Add a cumulative frequency column
laban_sorted['Kumulativt Antall'] = laban_sorted['Antall'].cumsum()


## Brynhild
# Create a frequency table for the "Midpunkt" column
brynhild.freq = brynhild.df_raw['Midpunkt'].value_counts()

# Create cumulative frequency table for "Midpunkt" colum
brynhild.cum_freq = ...
    brynhild.df_raw['Midpunkt'].value_counts().sort_index(ascending=False).cumsum()

# Set table up in correct format
brynhild_sorted = brynhild.df_raw.groupby(['Lower', 'Upper', 'Midpunkt', ...
    'Bredde']).size().reset_index(name='Antall')

# Sort the DataFrame by 'Antall' in ascending order
brynhild_sorted = brynhild_sorted.sort_values('Midpunkt', ascending=False)

# Add a cumulative frequency column
brynhild_sorted['Kumulativt Antall'] = brynhild_sorted['Antall'].cumsum()




###   Plotting   ###
## Set-Up  ##
# Create the first figure for frequency diagrams
```

```
fig1, axs1 = plt.subplots(2, 1, figsize=(8.27, 11.69))

# LABAN
# First subplot: Frequency Diagram - Laban
axs1[0].bar(laban.freq.index, laban.freq, color='skyblue', width=4.5)
axs1[0].set_title('Frequency Diagram - Laban')
axs1[0].set_xlabel('Stretch before failure [mm]')
axs1[0].set_ylabel('Frequency')

# First subplot: Frequency Diagram - Laban
mean_line = axs1[0].axvline(laban.avg, color='red', linestyle='dashed', ...
    linewidth=2)
median_line = axs1[0].axvline(laban.median, color='green', ...
    linestyle='dashed', linewidth=2)
mode_line = axs1[0].axvline(laban.mode, color='orange', ...
    linestyle='dashed', linewidth=2)
std_line = axs1[0].axvline(laban.avg + laban.std, color='purple', ...
    linestyle='dashed', linewidth=2)
std_neg_line = axs1[0].axvline(laban.avg - laban.std, color='purple', ...
    linestyle='dashed', linewidth=2)
std_pop_line = axs1[0].axvline(laban.avg + laban.std_pop, color='Pink', ...
    linestyle='dashed', linewidth=2)
std_pop_neg_line = axs1[0].axvline(laban.avg - laban.std_pop, ...
    color='Pink', linestyle='dashed', linewidth=2)


# Add a legend
axs1[0].legend([mean_line, median_line, mode_line, std_line, ...
    std_pop_line], ['Mean', 'Median', 'Mode', 'SD', 'PSD'], loc='upper right')

# Brynhild
# Second subplot: Frequency Diagram - Brynhild
axs1[1].bar(brynhild.freq.index, brynhild.freq, color='salmon', width=4.5)
axs1[1].set_title('Frequency Diagram - Brynhild')
axs1[1].set_xlabel('Stretch before failure [mm]')
axs1[1].set_ylabel('Frequency')

# First subplot: Frequency Diagram - Brynhild
mean_line = axs1[1].axvline(brynhild.avg, color='red', linestyle='dashed', ...
    linewidth=2)
median_line = axs1[1].axvline(brynhild.median, color='green', ...
    linestyle='dashed', linewidth=2)
mode_line = axs1[1].axvline(brynhild.mode, color='orange', ...
    linestyle='dashed', linewidth=2)
std_line = axs1[1].axvline(brynhild.avg + brynhild.std, color='purple', ...
    linestyle='dashed', linewidth=2)
std_neg_line = axs1[1].axvline(brynhild.avg - brynhild.std, ...
    color='purple', linestyle='dashed', linewidth=2)
std_pop_line = axs1[1].axvline(brynhild.avg + brynhild.std_pop, ...
    color='Pink', linestyle='dashed', linewidth=2)
std_pop_neg_line = axs1[1].axvline(brynhild.avg - brynhild.std_pop, ...
    color='Pink', linestyle='dashed', linewidth=2)

# Add a legend
axs1[1].legend([mean_line, median_line, mode_line, std_line, ...
    std_pop_line], ['Mean', 'Median', 'Mode', 'SD', 'PSD'], loc='upper right')

# Increase the vertical space between the subplots
plt.subplots_adjust(hspace=0.5)

# Create the second figure for cumulative frequency diagrams
```

```
fig2, axs2 = plt.subplots(2, 1, figsize=(8.27, 11.69))

# LABAN
# First subplot: Cumulative Frequency Diagram - Laban
axs2[0].plot(laban.cum_freq.index, laban.cum_freq, marker='o', ...
    color='skyblue')
axs2[0].set_title('Cumulative Frequency Diagram - Laban')
axs2[0].set_xlabel('Stretch before failure [mm]')
axs2[0].set_ylabel('Cumulative Frequency')

# Second subplot: Cumulative Frequency Diagram - Laban
mean_line = axs2[0].axvline(laban.avg, color='red', linestyle='dashed', ...
    linewidth=2)
median_line = axs2[0].axvline(laban.median, color='green', ...
    linestyle='dashed', linewidth=2)
mode_line = axs2[0].axvline(laban.mode, color='orange', ...
    linestyle='dashed', linewidth=2)
std_line = axs2[0].axvline(laban.avg + laban.std, color='purple', ...
    linestyle='dashed', linewidth=2)
std_neg_line = axs2[0].axvline(laban.avg - laban.std, color='purple', ...
    linestyle='dashed', linewidth=2)
std_pop_line = axs2[0].axvline(laban.avg + laban.std_pop, color='Pink', ...
    linestyle='dashed', linewidth=2)
std_pop_neg_line = axs2[0].axvline(laban.avg - laban.std_pop, ...
    color='Pink', linestyle='dashed', linewidth=2)

# Add a legend
axs2[0].legend([mean_line, median_line, mode_line, std_line, ...
    std_pop_line], ['Mean', 'Median', 'Mode', 'SD', 'PSD'], loc='upper right')

# Brynhild
# Second subplot: Cumulative Frequency Diagram - Brynhild
axs2[1].plot(brynhild.cum_freq.index, brynhild.cum_freq, marker='o', ...
    color='salmon')
axs2[1].set_title('Cumulative Frequency Diagram - Brynhild')
axs2[1].set_xlabel('Stretch before failure [mm]')
axs2[1].set_ylabel('Cumulative Frequency')

# Second subplot: Cumulative Frequency Diagram - Brynhild
mean_line = axs2[1].axvline(brynhild.avg, color='red', linestyle='dashed', ...
    linewidth=2)
median_line = axs2[1].axvline(brynhild.median, color='green', ...
    linestyle='dashed', linewidth=2)
mode_line = axs2[1].axvline(brynhild.mode, color='orange', ...
    linestyle='dashed', linewidth=2)
std_line = axs2[1].axvline(brynhild.avg + brynhild.std, color='purple', ...
    linestyle='dashed', linewidth=2)
std_neg_line = axs2[1].axvline(brynhild.avg - brynhild.std, ...
    color='purple', linestyle='dashed', linewidth=2)
std_pop_line = axs2[1].axvline(brynhild.avg + brynhild.std_pop, ...
    color='Pink', linestyle='dashed', linewidth=2)
std_pop_neg_line = axs2[1].axvline(brynhild.avg - brynhild.std_pop, ...
    color='Pink', linestyle='dashed', linewidth=2)

# Add a legend
axs2[1].legend([mean_line, median_line, mode_line, std_line, ...
    std_pop_line], ['Mean', 'Median', 'Mode', 'SD', 'PSD'], loc='upper right')

# Increase the vertical space between the subplots
plt.subplots_adjust(hspace=0.5)
```

```
# Show the plots
#plt.show()


#### Save Files  ####
##  Save figures  ##
# Define the relative directory where you want to save the files
directory = "oblig_1a/figures/"

# Create the directory if it doesn't exist
os.makedirs(directory, exist_ok=True)


# Define the name of the first file
filename1 = 'freq.pgf'

# Construct the full path to the first file
full_path1 = os.path.join(directory, filename1)

# Save the first figure to this location
fig1.savefig(full_path1)


# Define the name of the second file
filename2 = 'cum_freq.pgf'

# Construct the full path to the second file
full_path2 = os.path.join(directory, filename2)

# Save the second figure to this location
fig2.savefig(full_path2)




## Save tables ##
# Specify the directory
directory_table = "oblig_1a/tables/"




###   Export Tables to LaTeX   ###
## Laban
# Save table in latex format
filepath = directory_table + "laban_table"
with open(filepath, 'w') as f:
  f.write(laban_sorted.to_latex())


## Brynhild
# Save table in latex format
filepath = directory_table + "brynhild_table"
with open(filepath, 'w') as f:
  f.write(brynhild_sorted.to_latex())
```