A3: Data Preprocessing and Knowledge Discovery (20 points)

DUE: Sunday, 29 October 2023, 23:59

General Task Description

This Jupyter notebook guides you through crucial steps in data preprocessing, which you will need for your IVDA projects. This includes handling missing values, grouping, mapping, normalizing data, and other key skills. The lecture content you'll need to help you through this assignment can be found in the L08 ppt file posted on OLAT. This Jupyter notebook is meant to help you along a self-study of that lecture material, which you'll begin during the usual lecture period on Thursday, October 18th, 14-16h. Please use this time to ensure that you're properly set up for working on the tasks presented. The TAs will be available to support you in this during the lecture period.

If you have never used a Jupyter Notebook before, don't fret! Start-up instructions can be found here:

- General, comprehensive walkthrough: https://www.dataquest.io/blog/jupyter-notebook-tutorial/ https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/
- Jupyter Notebooks for VS Code users: https://code.visualstudio.com/docs/datascience/jupyter-notebooks

The dataset we'll be wrangling today is from a publicly available questionnaire. The link to the questionnaire, which includes documentation on every question in your dataset, can be found here: https://data.mendeley.com/datasets/88y3nffs82/5/files/c5f122b0-0380-42e7-9930-3d85ee083a06.

Point Distribution:

This assignment is worth 20 points in total. Your assigned tasks and questions can be found in **bold text** in the code skeleton below. Some tasks require a written response in addition to your code contribution. Generally, the points are allocated as follows, with a more detailed breakdown presented in the tasks themselves:

- Task 1 - Handling duplicates & mapping Data (3 points)
- Task 2 - Handling and treating missing values (4.5 points)

- Task 3 - Grouping data & colour coding (2.5 points)
- Task 4 - Normalizing data & colour coding (3 points)
- Task 5 - KDD (7 points)

Submission:

Please submit a PDF export of your completed Jupyter notebook, with all the cells' outputs visible, **especially the visualiations and written responses**. You will be graded on the outcome of your code and on your written responses, not the code itself.

In your submission, please make sure you keep in mind the following:

- **Plots:** Always add titles, legends, axes descriptions, etc. It is recommended you use the *plotly* library.
- **Written responses:** Please answer as concisely as possible, while still using full sentences.
- **Allowed packages:** Your required packages are included in the code skeleton below. **Please do not import additional packages into this notebook**. Also, please do not use the *pandas-profiling* function. Your goal is to learn the abilities and limits of the tools presented below, to make you knowledgeable about the iterative process of data wrangling. However, you're welcome to use it to confirm your findings, if you so desire (not mandatory).
- Selected solutions will be discussed after the deadline.

In [209…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
import plotly.express as px
import plotly.graph_objects as go
```

Import the data from the url: https://raw.githubusercontent.com/akkuebler/ivda_dataprocessing/main/preppedStudentCovidData.csv' This is also provided to you in the A3 folder you've downloaded from OLAT, in case you have difficulties importing from a link.

Note: you will want to have the responding student's number as your dataframe's index

In [210…
```
url ='https://raw.githubusercontent.com/akkuebler/ivda_dataprocessing/main/preppedStudentCovidData.csv'
df = pd.read_csv (url, index_col=0)
```

Display the first 5 rows of your dataset.

In [211…
```
df[:5]
```

Out[211]:

	Q1	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10a	Q10b	...	Q38h_2	Q38i_2	Q38j_2	Q38k_2	Q38l_2	Q38m_2	Q38n_2	Q38o_2	Q38p_2
29349	Turkey	1.0	1.0	1.0	1.0	24.0	1.0	2.0	1.0	1.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
23415	Portugal	1.0	1.0	1.0	3.0	NaN	1.0	2.0	4.0	5.0	...	1.0	3.0	5.0	4.0	3.0	5.0	1.0	2.0	2.0
5568	Croatia	1.0	1.0	1.0	2.0	20.0	2.0	2.0	3.0	3.0	...	5.0	1.0	1.0	4.0	2.0	2.0	1.0	1.0	2.0
10176	Hungary	1.0	1.0	1.0	3.0	21.0	1.0	2.0	3.0	3.0	...	5.0	1.0	3.0	4.0	4.0	3.0	1.0	3.0	1.0
2226	Bangladesh	1.0	1.0	2.0	1.0	NaN	1.0	2.0	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 161 columns

Task 1 - Handling duplicates & mapping data (3 points)

Task 1.1

Make yourself familiar with the data's shape and size

In [212…
```
print("The dataset contains {} data records and {} features.".format(df.shape[0], df.shape[1]))
```

```
The dataset contains 10000 data records and 161 features.
```

Read through all the tasks in this notebook, and then create a list of all the questions in the questionaire which are relevant for your assigned data analysis tasks.

Note: the answer to each question is stored in a column. In our context of A3, we refer to those columns as *attributes* or *features*

Task 1.2

Give basic summary statistics for the questions you've identified as relevant, and state which of these questions has the most missing data. (0.5 points)

In [213…

```
# summary statistics
relevant_questions = ['Q1', 'Q7', 'Q4', 'Q17', 'Q5']
df_relevant = df[relevant_questions]

df_25 = df.filter(like='Q25')
df_26 = df.filter(like='Q26')

df_relevant = pd.concat([df_relevant, df_25, df_26], axis=1)
df_relevant
```

Out[213]:

	Q1	Q7	Q4	Q17	Q5	Q25a	Q25b	Q25c	Q25d	Q25e	...	Q26a	Q26b	Q26c	Q26d	Q26e	Q26f	Q26g	Q26h	Q26i	Q26j
29349	Turkey	24.0	1.0	NaN	1.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
23415	Portugal	NaN	1.0	4.0	1.0	4.0	5.0	5.0	5.0	4.0	...	5.0	5.0	4.0	3.0	5.0	5.0	3.0	4.0	5.0	3.0
5568	Croatia	20.0	1.0	5.0	1.0	3.0	3.0	2.0	4.0	4.0	...	1.0	3.0	5.0	5.0	5.0	3.0	5.0	2.0	4.0	1.0
10176	Hungary	21.0	1.0	1.0	1.0	1.0	3.0	3.0	4.0	1.0	...	1.0	1.0	5.0	5.0	2.0	1.0	1.0	1.0	2.0	3.0
2226	Bangladesh	NaN	1.0	NaN	2.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12150	Indonesia	20.0	1.0	4.0	1.0	3.0	2.0	2.0	5.0	3.0	...	3.0	3.0	5.0	5.0	5.0	2.0	5.0	5.0	5.0	5.0
3969	Chile	35.0	2.0	NaN	1.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
29241	Turkey	26.0	1.0	3.0	2.0	3.0	4.0	3.0	1.0	1.0	...	1.0	1.0	1.0	4.0	1.0	1.0	2.0	3.0	3.0	4.0
15925	Malaysia	47.0	1.0	NaN	2.0	3.0	4.0	4.0	3.0	1.0	...	1.0	1.0	2.0	2.0	2.0	2.0	NaN	2.0	2.0	2.0
13276	Italy	23.0	1.0	4.0	1.0	3.0	4.0	3.0	4.0	3.0	...	1.0	1.0	2.0	1.0	1.0	3.0	1.0	1.0	1.0	1.0

10000 rows × 25 columns

In [214...
```
df_relevant.to_csv('relevant.csv')
```

In [215...
```
df_relevant.describe()
```

Out[215]:

	Q7	Q4	Q17	Q5	Q25a	Q25b	Q25c	Q25d	Q25e	Q25f	...	
count	8024.000000	9493.000000	6655.000000	9430.000000	6817.000000	6798.000000	6800.000000	6799.000000	6799.000000	6804.000000	...	6719.00
mean	23.546735	1.115559	3.396093	1.242842	2.853601	3.129303	2.688676	3.340197	2.960141	3.270870	...	2.58
std	5.677028	0.319712	1.227374	0.523890	1.067821	1.121784	1.155630	1.066495	1.083197	1.180578	...	1.24
min	18.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.00
25%	20.000000	1.000000	3.000000	1.000000	2.000000	2.000000	2.000000	3.000000	2.000000	2.000000	...	2.00
50%	22.000000	1.000000	4.000000	1.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	...	2.00
75%	24.000000	1.000000	4.000000	1.000000	4.000000	4.000000	3.000000	4.000000	4.000000	4.000000	...	3.00
max	70.000000	2.000000	5.000000	3.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	...	5.00

8 rows × 24 columns

In [216…

```
# Which question has the most missing data?
sum_nan = df_relevant.isna().sum()
print(sum_nan.idxmax())
```

```
Q17
```

Task 1.3

When working with data, we should be concerned not just by missing data, but also by duplicated rows. **Check your dataset of relevant questions (from the above step) for duplicates, and delete all but the first entry.**

Hint: Be aware that the dataset contains duplicate indices. The indices function as a unique student number for every student.

In [217…

```
# Check for duplicates
df_relevant = (df_relevant.reset_index()
        .drop_duplicates(keep='first')
        .set_index('index'))
#df_relevant.drop_duplicates(keep='first', inplace=True, ignore_index=False)
df_relevant
```

Out[217]:

index	Q1	Q7	Q4	Q17	Q5	Q25a	Q25b	Q25c	Q25d	Q25e	...	Q26a	Q26b	Q26c	Q26d	Q26e	Q26f	Q26g	Q26h	Q26i	Q26j
29349	Turkey	24.0	1.0	NaN	1.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
23415	Portugal	NaN	1.0	4.0	1.0	4.0	5.0	5.0	5.0	4.0	...	5.0	5.0	4.0	3.0	5.0	5.0	3.0	4.0	5.0	3.0
5568	Croatia	20.0	1.0	5.0	1.0	3.0	3.0	2.0	4.0	4.0	...	1.0	3.0	5.0	5.0	5.0	3.0	5.0	2.0	4.0	1.0
10176	Hungary	21.0	1.0	1.0	1.0	1.0	3.0	3.0	4.0	1.0	...	1.0	1.0	5.0	5.0	2.0	1.0	1.0	1.0	2.0	3.0
2226	Bangladesh	NaN	1.0	NaN	2.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12150	Indonesia	20.0	1.0	4.0	1.0	3.0	2.0	2.0	5.0	3.0	...	3.0	3.0	5.0	5.0	5.0	2.0	5.0	5.0	5.0	5.0
3969	Chile	35.0	2.0	NaN	1.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
29241	Turkey	26.0	1.0	3.0	2.0	3.0	4.0	3.0	1.0	1.0	...	1.0	1.0	1.0	4.0	1.0	1.0	2.0	3.0	3.0	4.0
15925	Malaysia	47.0	1.0	NaN	2.0	3.0	4.0	4.0	3.0	1.0	...	1.0	1.0	2.0	2.0	2.0	2.0	NaN	2.0	2.0	2.0
13276	Italy	23.0	1.0	4.0	1.0	3.0	4.0	3.0	4.0	3.0	...	1.0	1.0	2.0	1.0	1.0	3.0	1.0	1.0	1.0	1.0

8438 rows × 25 columns

Why can you not simply use `df.drop_duplicates(keep ='first')` ? What would happen in this case? (Max. 50 words). (0.5 points)

In [219…

```
print("After handling duplicates, the dataset now contains {} data records and {} features.".format(df_relevant.shape[0], df_r
```

```
After handling duplicates, the dataset now contains 8438 data records and 25 features.
```

Since we set the student id as index, then the drop_duplicates will move all the rows that are exactly the same without considering the index. Then we would remove same answers that could have been given by different students.

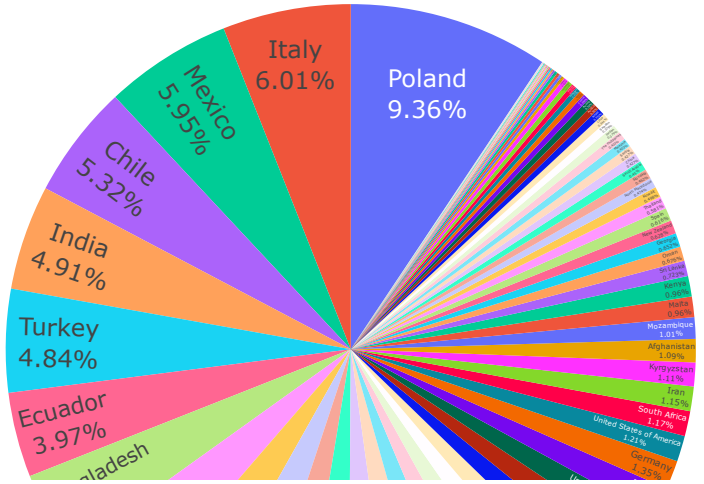
Task 1.4

Create a pie chart visualization describing the ratio of responses from each country in your dataset. For now, do not perform any additional clean-up of your data. (0.5 points)

In [220…

```
c = list(df_relevant["Q1"])
def counting_as_dict(c):
    countries = dict()
    for country in c:
        if country not in countries:
            countries[country] = 1
        else:
            countries[country] += 1
    return countries



def plot_pie(df):
    # Created the plot from a data frame, since I was not able to modify the legends of the hoover data otherwise.
    fig = px.pie(df, values='Responses', names='Country', title='Responses from each country')
    fig.update_traces(textposition='inside', textinfo='percent+label')
    fig.show()

country_dict = counting_as_dict(c)

data_list = [(key, value) for key, value in country_dict.items()]
country_df = pd.DataFrame(data_list, columns=['Country', 'Responses'])
plot_pie(country_df)
```

Responses from each country



As you can see, the plot is very cluttered! To reduce this clutter, we could instead create and plot with a set of logical groupings-- in the case of our data here, these groupings could be continents.

Map the countries to the respective continents using the provided `continent.csv` file, and then plot it again. You can find this csv either in your A3 assignment folder from OLAT, or at https://raw.githubusercontent.com/akkuebler/ivda_dataprocessing/main/continents.csv (0.5 points)

Hint: Turn the `continents` dataframe into a dictionary you can use for mapping

In [221…
```
url2 ='https://raw.githubusercontent.com/akkuebler/ivda_dataprocessing/main/continents.csv'
continents = pd.read_csv (url2, index_col=0)
continents
```

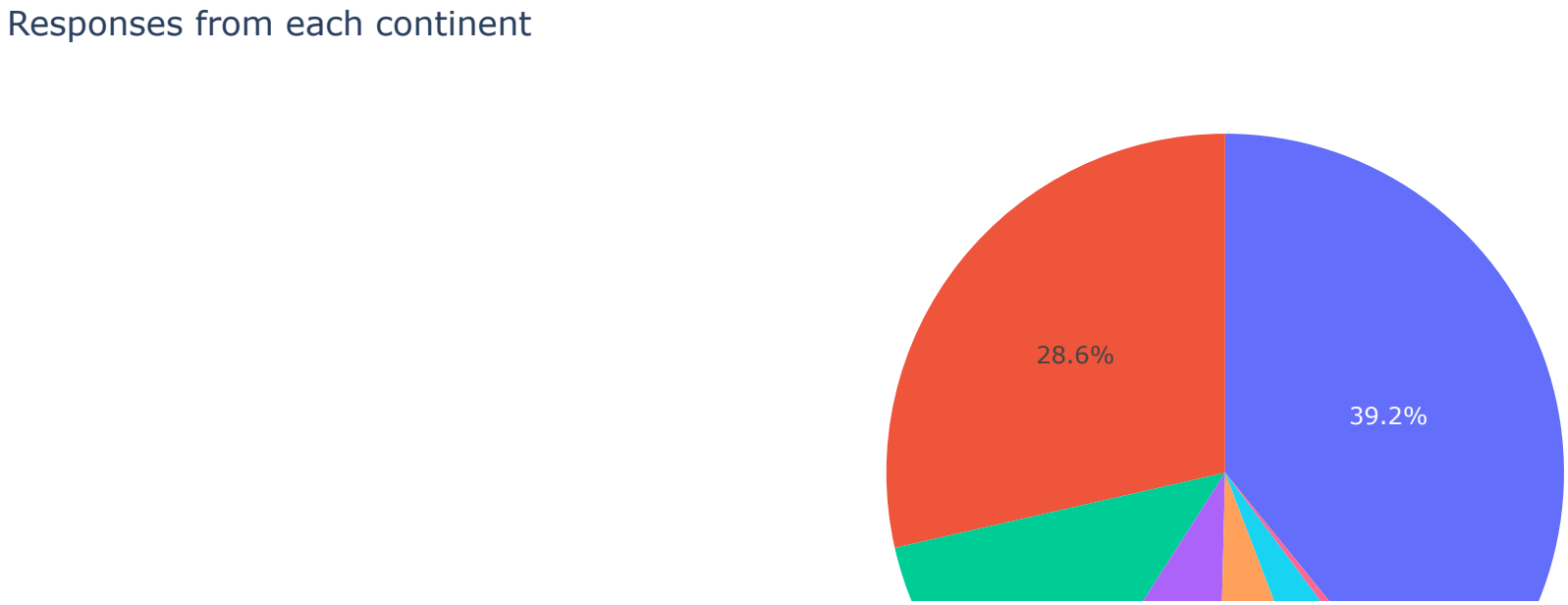
Out[221]:

country	continent
Algeria	Africa
Angola	Africa
Benin	Africa
Botswana	Africa
Burkina Faso	Africa
...	...
Paraguay	South America
Peru	South America
Suriname	South America
Uruguay	South America
Venezuela	South America

234 rows × 1 columns

In [222…
```
continents_dict = continents.to_dict()['continent']
```

In [223…
```
# Mapping
def add_continent(country):
    # The countries of the dataset do not match entirely with the countries in the continent dataset.
    # Example: 'The Netherlands' and 'Netherlands'. That's why unknown continent exist.
    try:
        return continents_dict[country]
    except:
        return "Unknown"

country_df['Continent'] = country_df['Country'].map(lambda x: add_continent(x))

def plot_pie2(df):
    # Created the plot from a data frame, since I was not able to modify the legends of the hoover data otherwise.
    fig = px.pie(df, values='Responses', names='Continent', title='Responses from each continent')
    #fig.update_traces(textposition='inside', textinfo='percent+label')
    fig.show()

#country_dict = counting_as_dict(c)
plot_pie2(country_df)
```



In [153…
```
country_df.to_csv('temp.csv')
```

Inspect the pie chart and the dataframe: Why do you see some 4-5% missing values? Where do these rows come from? (Max. 50 words). (0.5 points)

Reason 1: The names of the countries not match in both datasets. Example: 'The Netherlands' and 'Netherlands'.

Reason 2: Some cells in Q1 were NaN. NaN cells do not have a continent.

In the visualization this values were assigned an 'Unknown' continent in the Data Frame. It is a bad practice that the parts of a pie chart don't sum up a whole (100%)

Apply your findings from the question above to further reduce the amount of missing data. Plot the pie chart again. (0.5 points)

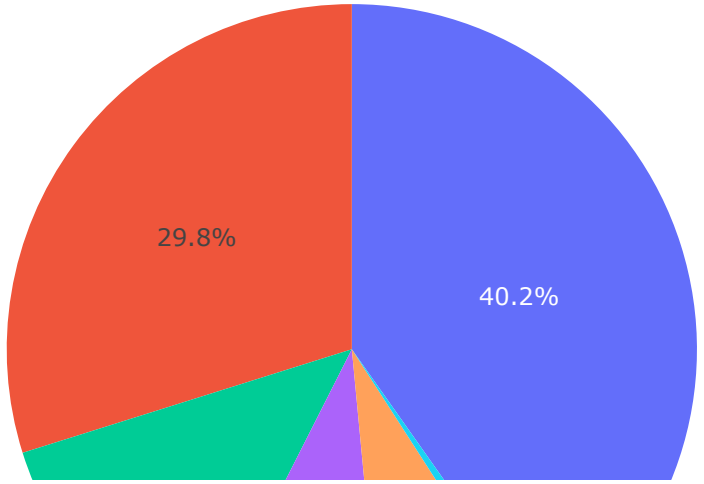
In [224…

```
country_df.dropna(subset=['Country'], inplace=True)

country_df['Country']=country_df['Country'].str.replace("United States of America", "United States")
country_df['Country']=country_df['Country'].str.replace("The Philippines", "Philippines")
country_df['Country']=country_df['Country'].str.replace("Phillipines", "Philippines")
country_df['Country']=country_df['Country'].str.replace("Democratic Republic of Congo", "DR Congo")
country_df['Country']=country_df['Country'].str.replace("The Netherlands", "Netherlands")
country_df['Country']=country_df['Country'].str.replace("Palestinian State", "Palestine")
```

In [225…

```
# Mapping
def add_continent(country):
    return continents_dict[country]


country_df['Continent'] = country_df['Country'].map(lambda x: add_continent(x))

def plot_pie2(df):
    # Created the plot from a data frame, since I was not able to modify the legends of the hoover data otherwise.
    fig = px.pie(df, values='Responses', names='Continent', title='Responses from each continent')
    #fig.update_traces(textposition='inside', textinfo='percent+label')
    fig.show()

#country_dict = counting_as_dict(c)
plot_pie2(country_df)
```

Responses from each continent



Please note: from the perspective of visuzalization theory, pie charts are not ideal! This is because humans do not do well when asked to estimate quantities within (and across) pie charts with spatial disarray. As Edward Tufte once wrote:

> "... the only worse design than a pie chart is several of them."
>
> Tufte. (2015). The visual display of quantitative information (Second edition, ninth printing). Graphics Press.

In our context, we use pie charts only for the purpose of visualizing the progress we make in our grouping of the data. Since we do not set out to express some subtle findings to a reader, our use case is sufficiently simple for a pie chart. However, please not use pie charts in your upcoming group projects, or reports.

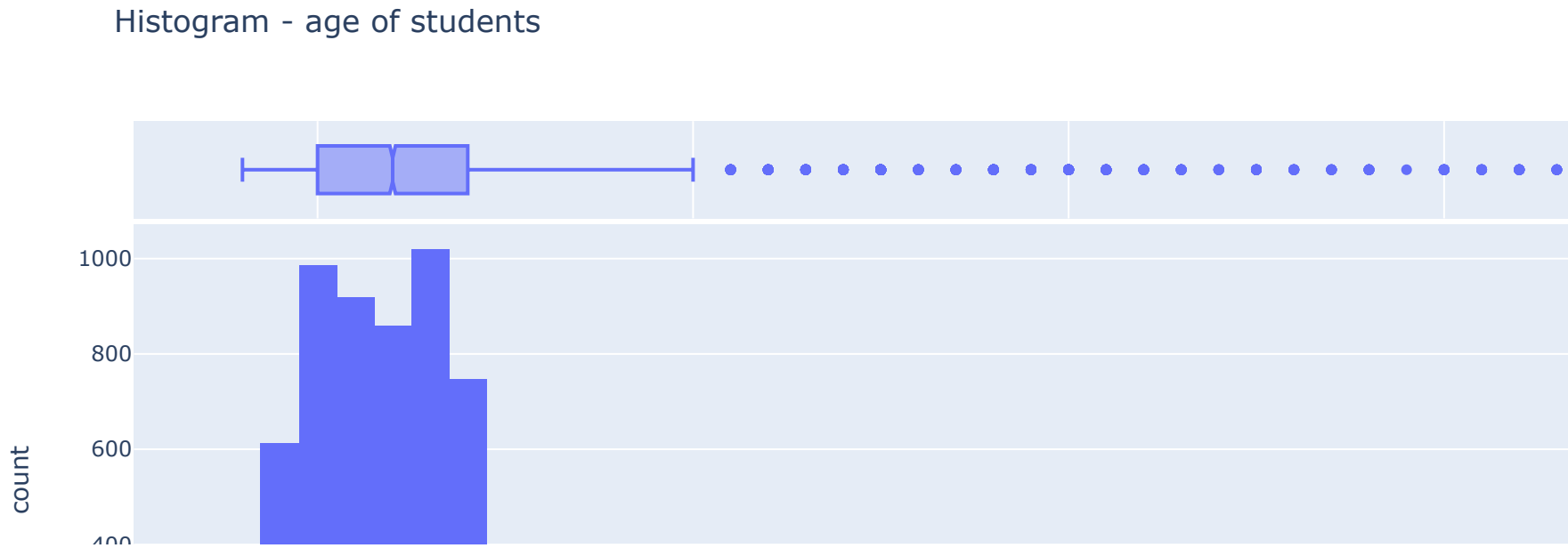
Task 2 - Handling and treating missing values (4.5 points)

Task 2.1

Visualize the age of the students in your dataset with an appropriate boxplot and histogram. With regards to the visualizations' treatment of missing data, what issues do you encounter here? (Max. 50 words) (1 point)

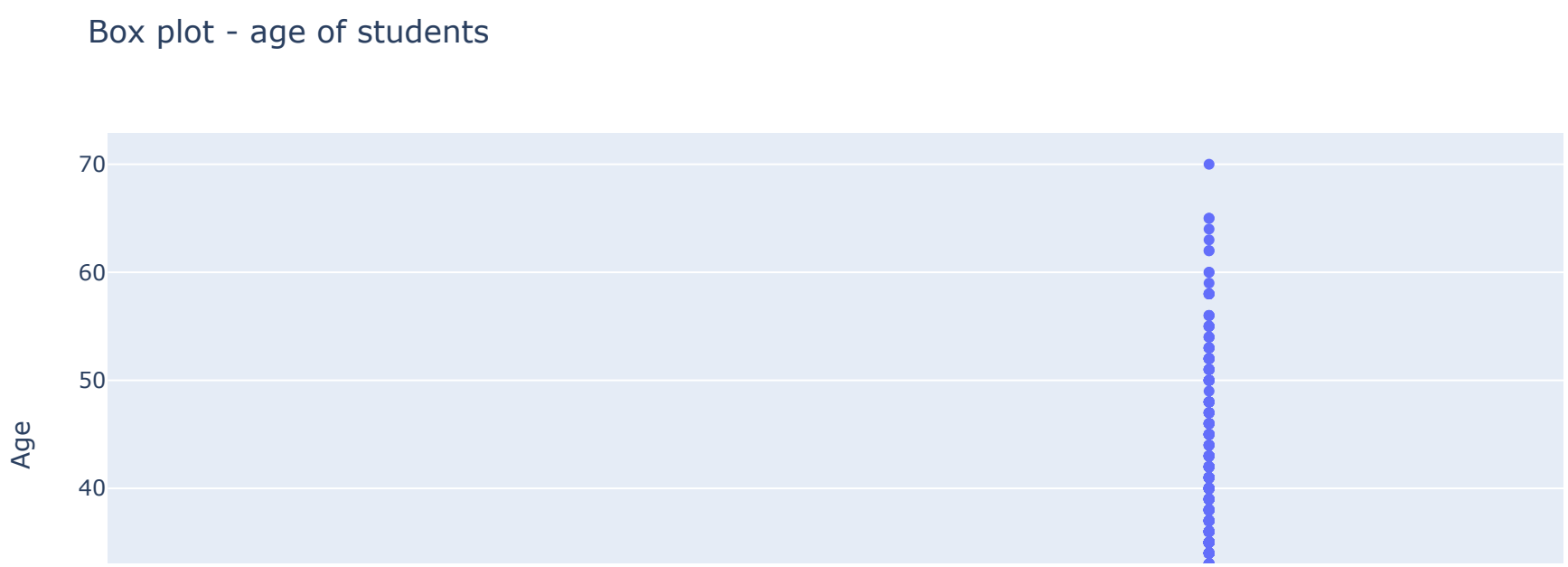
In [228…

```
# Visualize
fig = px.histogram(df_relevant, x="Q7", title="Histogram - age of students", marginal='box', labels={
                     "Q7": "Age",
                 })
fig.show()
```



In [227…

```
fig = px.box(df_relevant, y="Q7", title="Box plot - age of students", labels={
                     "Q7": "Age",
                 })
fig.show()
```



Visualize the age of the students in your dataset with an appropriate boxplot and histogram. With regards to the visualizations' treatment of missing data, what issues do you encounter here? (Max. 50 words) (1 point)

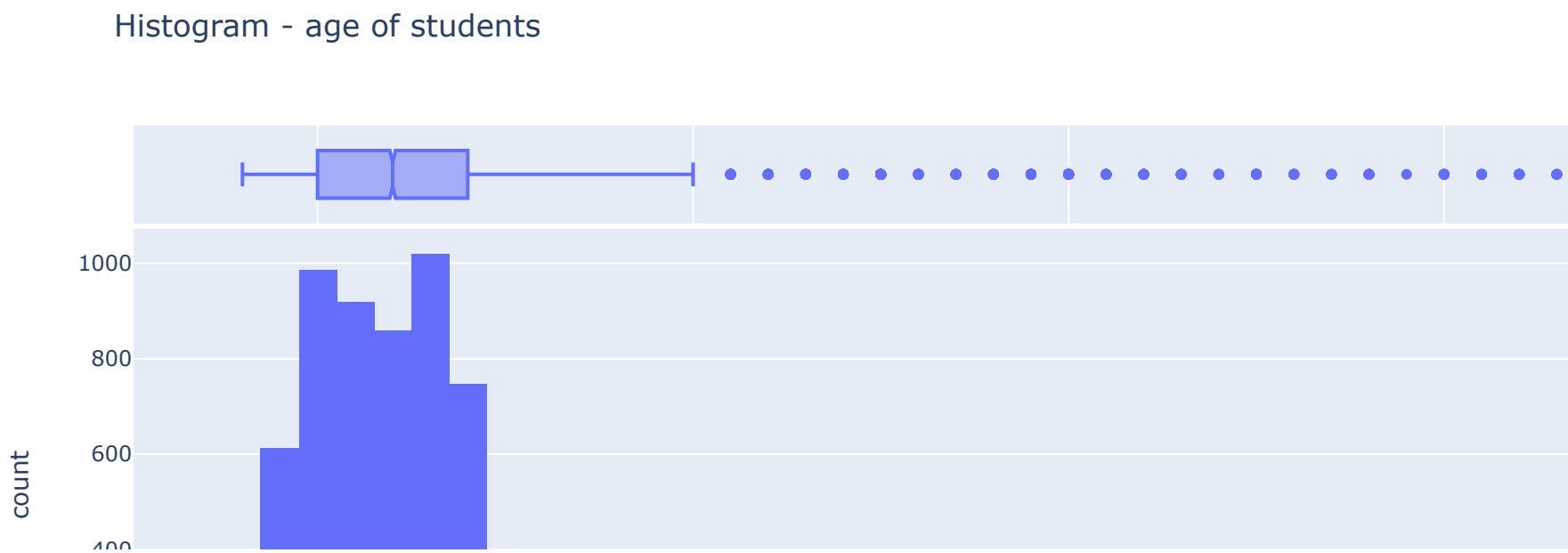
In this case plotting without removing missing values and with missing values return exactly the same plots. Looks like plotly automatically ignored NaN values before plotting. Depending what we are analyzing, this could be negative, because we don't know the amount of students that refused to answer that question.

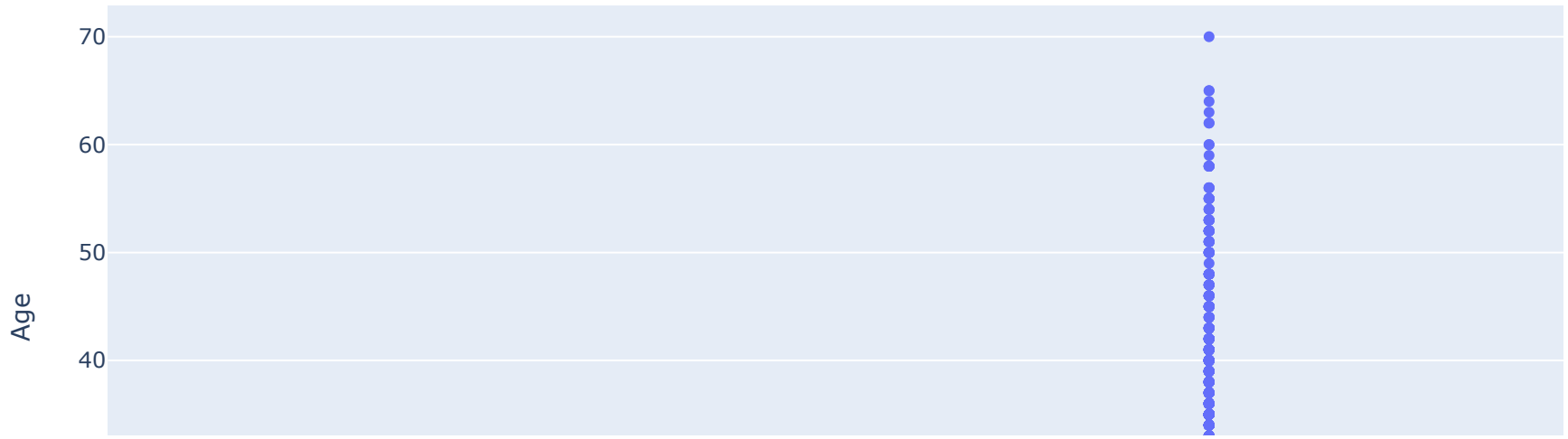
Now, filter out missing data using `np.isnan()`, and run your visualization code again. (0.5 points)

Hint: If you are working on your local machine, you might want use `from plotly.offline import plot` together with `plot(fig)`. This creates a locally-stored HTML that is then opened within your web browser.

In [229…

```
# Filter data using np.isnan
Q7_no_nan = df_relevant.dropna(subset=['Q7'])

# Visualize
fig1 = px.histogram(Q7_no_nan, x="Q7", title="Histogram - age of students", marginal='box', labels={
                     "Q7": "Age",
                 })
fig1.show()

fig2 = px.box(Q7_no_nan, y="Q7", title="Box plot - age of students", labels={
                     "Q7": "Age",
                 })
fig2.show()
```


Histogram - age of students
count
1000
800
600
400


Box plot - age of students
Age
70
60
50
40

Task 2.2

Inspect your dataset again. We still have to deal with some missing data!

Now you must come up with a strategy to reduce your (reduced) dataset's missing values, so that you can continue your work on a nice, clean dataset.

Hint: In general, there are 3 different ways to handle missing data:

1. Ignoring on purpose (as we did above)
2. Deleting entire entries (rows) or features (columns) with missing data, or deleting entire features when missing data reaches a certain threshold (i.e. ignoring them in your analysis)
3. Imputing missing data with the feature's mean or median, or treat a missing value as seperate category (when the feature is categorical).

But remember: All of the described ways above are data type- and context-dependent!

Task 2.2.1

Drop rows where all data is missing. Note that, since our index is the unique student number, we should NOT reset the index here. (0.5 points)

In [230…

```
print("The dataset before dropping the rows contains {} data records and {} features.".format(df_relevant.shape[0], df_relevan

# drop rows with missing data
df_relevant.dropna(how='all', inplace=True)

print("The dataset now contains {} data records and {} features.".format(df_relevant.shape[0], df_relevant.shape[1]))

df_relevant
```

```
The dataset before dropping the rows contains 8438 data records and 25 features.
The dataset now contains 8381 data records and 25 features.
```

Out[230]:

index	Q1	Q7	Q4	Q17	Q5	Q25a	Q25b	Q25c	Q25d	Q25e	...	Q26a	Q26b	Q26c	Q26d	Q26e	Q26f	Q26g	Q26h	Q26i	Q26j
29349	Turkey	24.0	1.0	NaN	1.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
23415	Portugal	NaN	1.0	4.0	1.0	4.0	5.0	5.0	5.0	4.0	...	5.0	5.0	4.0	3.0	5.0	5.0	3.0	4.0	5.0	3.0
5568	Croatia	20.0	1.0	5.0	1.0	3.0	3.0	2.0	4.0	4.0	...	1.0	3.0	5.0	5.0	5.0	3.0	5.0	2.0	4.0	1.0
10176	Hungary	21.0	1.0	1.0	1.0	1.0	3.0	3.0	4.0	1.0	...	1.0	1.0	5.0	5.0	2.0	1.0	1.0	1.0	2.0	3.0
2226	Bangladesh	NaN	1.0	NaN	2.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12150	Indonesia	20.0	1.0	4.0	1.0	3.0	2.0	2.0	5.0	3.0	...	3.0	3.0	5.0	5.0	5.0	2.0	5.0	5.0	5.0	5.0
3969	Chile	35.0	2.0	NaN	1.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
29241	Turkey	26.0	1.0	3.0	2.0	3.0	4.0	3.0	1.0	1.0	...	1.0	1.0	1.0	4.0	1.0	1.0	2.0	3.0	3.0	4.0
15925	Malaysia	47.0	1.0	NaN	2.0	3.0	4.0	4.0	3.0	1.0	...	1.0	1.0	2.0	2.0	2.0	2.0	NaN	2.0	2.0	2.0
13276	Italy	23.0	1.0	4.0	1.0	3.0	4.0	3.0	4.0	3.0	...	1.0	1.0	2.0	1.0	1.0	3.0	1.0	1.0	1.0	1.0

8381 rows × 25 columns

Task 2.2.2

This task is specifically about Q7 (Age).

Impute the missing values with the feature mean, and visualize again the age of the students in your dataset with an appropriate boxplot and histogram. (0.5 points)

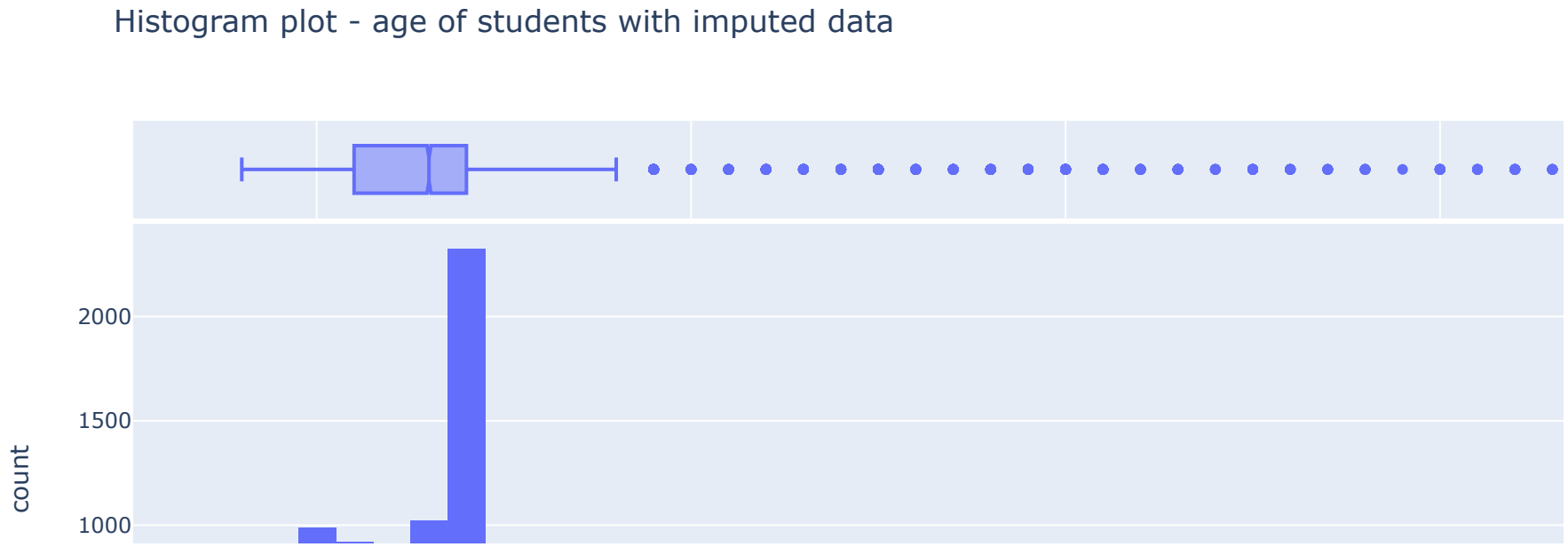
In [478…

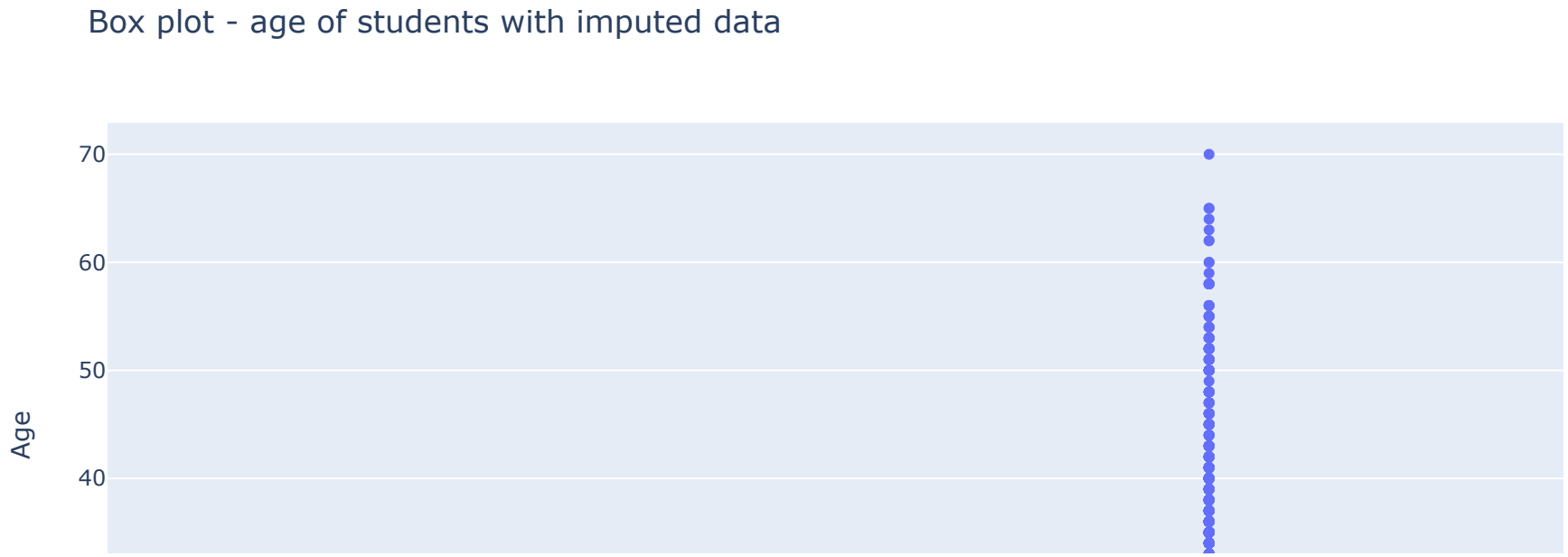
```
print("Original Data : \n", df_relevant.Q7.describe(), "\n")


# impute missing values
df_imputed = df_relevant['Q7'].replace(np.nan, df_relevant["Q7"].mean())
print("Imputed Data : \n", df_imputed.describe())

# It is not relevant to describe if we drip NaNs or not since they are ignored, therefore is expected to obtain the same stati

# Visualize
# fig1.add_vline(x=df_relevant["Q7"].mean(), line_dash = 'dash', line_color = 'firebrick', annotation_text="average")
# fig2.add_hline(y=df_relevant["Q7"].mean(), line_dash = 'dash', line_color = 'firebrick', annotation_text="average")
# fig1.show()
# fig2.show()
```

```
Original Data :
 count    6805.000000
mean       23.583688
std         5.680701
min        18.000000
25%        20.000000
50%        22.000000
75%        24.000000
max        70.000000
Name: Q7, dtype: float64

Imputed Data :
 count    8381.000000
mean       23.583688
std         5.118729
min        18.000000
25%        21.000000
50%        23.000000
75%        24.000000
max        70.000000
Name: Q7, dtype: float64
```

In [233…

```
# Visualize
fig3 = px.histogram(df_imputed, x="Q7", title="Histogram plot - age of students with imputed data", marginal='box', nbins=70,
                    "Q7": "Age",
                })
fig3.show()

fig4 = px.box(df_imputed, y="Q7",title="Box plot - age of students with imputed data", labels={
                    "Q7": "Age",
                })
fig4.show()
```


Histogram plot - age of students with imputed data
count
2000
1500
1000



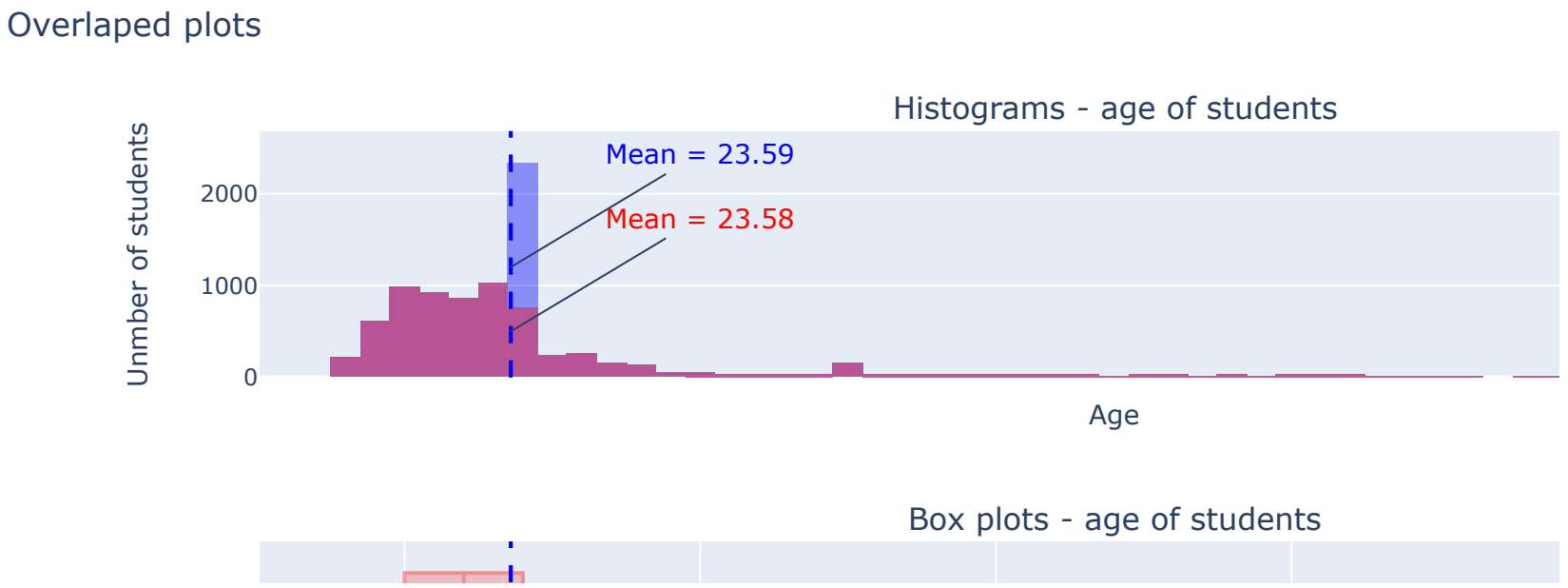
Show the graphs with the imputed and ignored datasets in one plot object. Model your visualization after the example below, and annotate each approach's mean, so that easy comparison between approaches is possible. In total, there should be 2 histograms and 2 boxplots (1 points)

IMAGE

In [259…

```
import plotly.subplots as sp

fig = sp.make_subplots(rows=2, cols=1, shared_xaxes=True, subplot_titles=["Histograms - age of students", "Box plots - age of
#fig = go.Figure()
trace1 = go.Histogram(x=df_imputed, nbinsx=70, marker_color="blue", name="Imputed missing fields, histogram")
trace2 = go.Histogram(x=df_relevant["Q7"], nbinsx=70, marker_color="red", name="Ignored missing fields, histogram")
trace3 = go.Box(x=df_imputed, marker_color="blue", name="Imputed missing fields, box plot")
trace4 = go.Box(x=df_relevant["Q7"], marker_color="red", name="Ignored missing fields, box plot")


fig.add_trace(trace1)
fig.add_trace(trace2)
fig.append_trace(trace3, row=2, col=1)
fig.append_trace(trace4, row=2, col=1)

fig.add_vline(x=df_relevant["Q7"].mean(), line_dash = 'dash', line_color = 'firebrick')
fig.add_vline(x=df_imputed.mean()+0.01, line_dash = 'dash', line_color = 'blue')

fig.add_annotation(text="Mean = {}".format(np.round(df_relevant["Q7"].mean(),2)),
                   x=df_relevant["Q7"].mean(),
                   y=500,
                   ax=100,
                   ay=-60,
                   align="left",
                   font=dict(
                       size=14,
                       color="red"
                   )
               )

fig.add_annotation(text="Mean = {}".format(np.round(df_relevant["Q7"].mean()+0.01,2)),
                   x=df_imputed.mean(),
                   y=1200,
                   ax=100,
                   ay=-60,
                   align="left",
                   font=dict(
                       size=14,
                       color="blue"
                   ))

fig.update_layout(title_text="Overlaped plots", xaxis_title="Age", yaxis_title="Unmber of students", barmode="overlay")
```

```
fig.update_traces(opacity=0.4)
fig.show()
```



Shortly describe the graph. What can you infer from it? Do the boxplots and histograms differ? If so, explain why. (Max. 100 words). (0.5 points)

The bars show the distribution of ages of different students. Most of the students are young people between 20-25 years old. Which is expected since we are speaking of students enrolled in higher education. There are a few students who are older, but it is common in most universities that they are a minority group.

By imputing the missing data it is possible to infer that the most common value are the missing values. The histogram and the box plot vary when values are imputed as well as the median, q1, and q3 values. It is important to note that imputed values not necessary describe the dataset since those people decided not to answer and could be much older or younger than the mean.

Task 2.2.3

This task is about Q4 (Student Status).

As this is categorical data, you will form a new category for the missing data and name it '*Unanswered*'. Then, create a plot for each age group stated below, describing the proportion of students who have attended school full-time and part-time. (0.5 points)

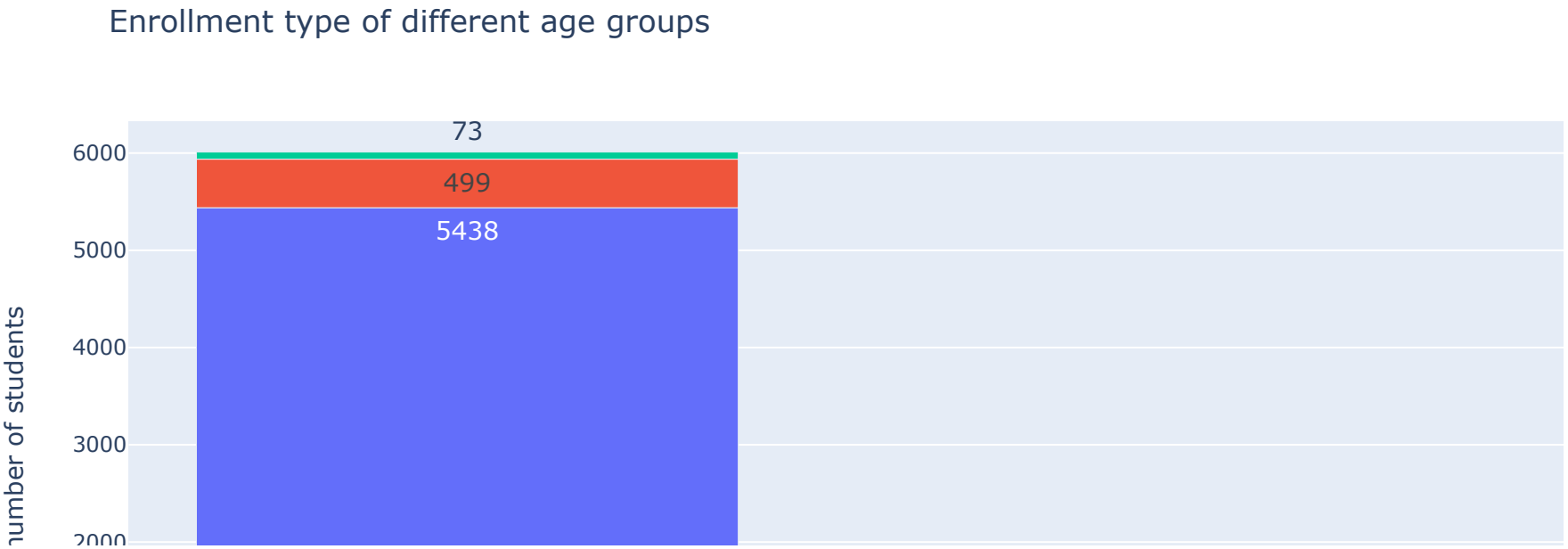
Age groups to include:

- 18-28
- 29-38
- 39+

In [336…

```
# Form new category named "Unanswered"
df_enrollmentType = df_relevant[["Q4", "Q7"]].copy()
df_enrollmentType["Q4"] = df_enrollmentType["Q4"].fillna("Unanswered")

# Map floats to categories to create groups (number and str are not compatible)
df_enrollmentType = df_enrollmentType.replace({"Q4":{1.0:"Full-time", 2.0:"Part-time"}})

# Since we are grouping by ages, I am removing the NaN in age column.
df_enrollmentType.dropna(subset=["Q7"], inplace=True)
df_enrollmentType["Q7"] = df_enrollmentType["Q7"].astype(float)


# Visualize
groups = df_enrollmentType.groupby(["Q4", pd.cut(df_enrollmentType.Q7,[18,28,39,80], include_lowest=True, right=False)]).count
groups.index.names = ["enrollment-type", "age"]
groups = groups.reset_index()
groups['age'] = groups['age'].astype(str)
groups = groups.replace({"age":{"[18, 28)":"18-27","[28, 39)":"28-38", "[39, 80)":"39+"}})
sums_age = groups[['age', 'Q7']].groupby(by='age').sum()
#print(sums_age.loc['18-27']['Q7'])

percentage_col=[]
for index, row in groups.iterrows():
    percentage_col.append(np.round((row['Q7']/sums_age.loc[row['age']]["Q7"])*100,2))
groups['percentage'] = percentage_col

fig = px.bar(groups, x="age", y="Q7", color="enrollment-type", text_auto=True, hover_data=['percentage'], title="Enrollment ty
fig.update_traces(textfont_size=14, textposition="outside", cliponaxis=False)
fig.show()
```

```
C:\Users\jorge\AppData\Local\Temp\ipykernel_21260\3255625547.py:14: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to
retain current behavior or observed=True to adopt the future default and silence this warning.
```


Enrollment type of different age groups
73
499
5438
6000
5000
4000
3000
2000
number of students

Task 3 - Grouping data & colour coding (2.5 points)

In this task, we will visualize students' perception of how their workload has changed since in-person classes were cancelled. Group the data depending on whether the student is a full-time or a part-time student. The associated questions in the questionaire are Q17 and Q4.

First, take care of the missing data present in this set of responses, and describe your strategy below (Max. 50 words) (0.5 points)

In [335…

```
# Handle missing data
df_3 = df_relevant[['Q4','Q17']].copy()
df_3["Q4"] = df_3["Q4"].fillna("Unanswered")
df_3 = df_3.replace({"Q4":{1.0:"Full-time", 2.0:"Part-time"}})
df_3.dropna(subset=["Q17"], inplace=True)
df_3
```

Out[335]:

index	Q4	Q17
23415	Full-time	4.0
5568	Full-time	5.0
10176	Full-time	1.0
2589	Full-time	3.0
2765	Full-time	4.0
...	...	...
17410	Full-time	5.0
21430	Part-time	4.0
12150	Full-time	4.0
29241	Full-time	3.0
13276	Full-time	4.0

5910 rows × 2 columns

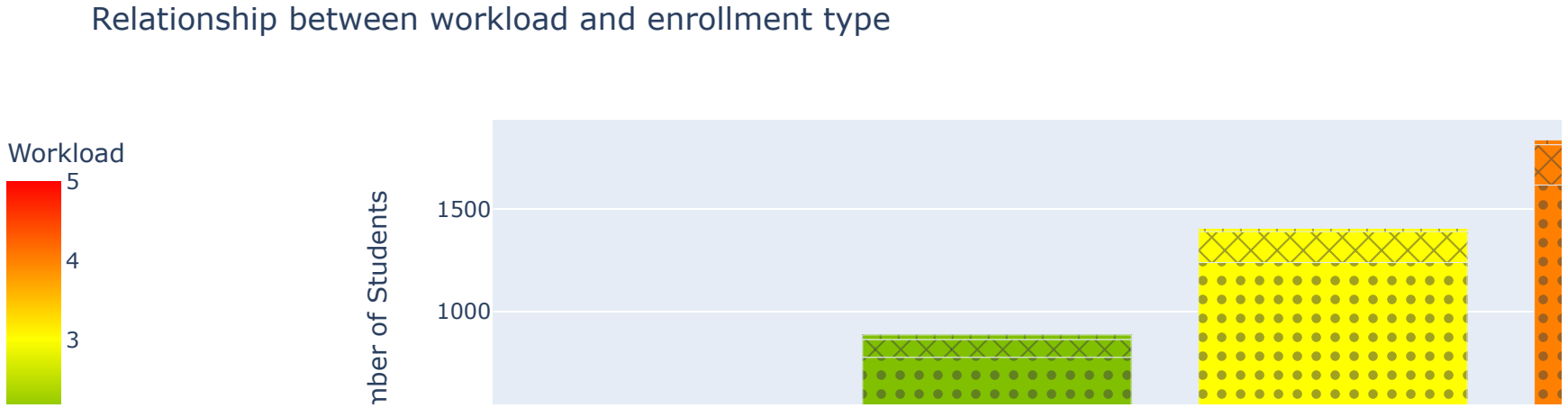
First the column Q4 was renamed into categories Full-Time, Part-Time. Since there are NaN values, I renamed them as Unanswered.

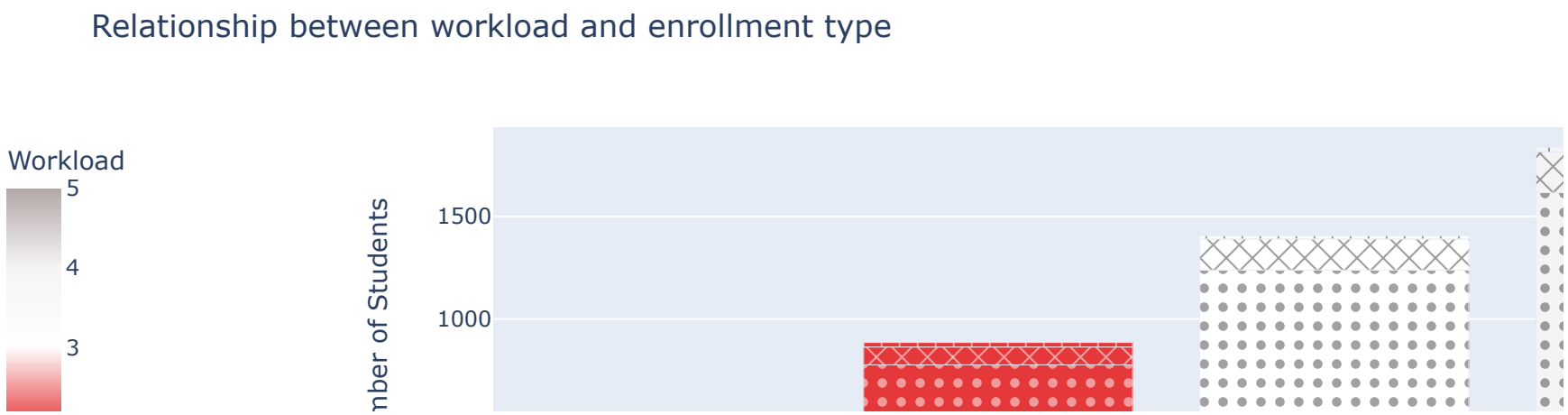
Finally, I dropped all the rows with NaNs in Q17, since missing information in this field is not relevant

Use an appropriate stacked bar chart as visualization. Plot it two times, once with a two-sided gradient color scheme in red/green, and the other time using the following colours: https://coolors.co/a4161a-e5383b-ffffff-f5f3f4-b1a7a6. (1.5 points)

In [360…

```
# Visualizations
groups_3 = df_3.groupby(by=['Q17','Q4']).size().reset_index()
fig = px.bar(groups_3, x='Q17', y=0, color='Q17', title="Relationship between workload and enrollment type",
             color_continuous_scale=['green', 'yellow','red'],
             hover_data=[0, 'Q17', 'Q4'],
             pattern_shape="Q4", pattern_shape_sequence=[".", "x", "+"],
             labels={'Q17':'Workload', 'Q4':'Enrollment Type', '0': 'Number of Students'}, height=400)
fig.update_layout(coloraxis_colorbar_x=-0.3)
fig.show()

fig2 = px.bar(groups_3, x='Q17', y=0, color='Q17', title="Relationship between workload and enrollment type",
             color_continuous_scale=["#A4161A","#E5383B","#FFFFFF","#F5F3F4","#B1A7A6"],
             hover_data=[0, 'Q17', 'Q4'],
             pattern_shape="Q4", pattern_shape_sequence=[".", "x", "+"],
             labels={'Q17':'Workload', 'Q4':'Enrollment Type', '0': 'Number of Students'}, height=400)
fig2.update_layout(coloraxis_colorbar_x=-0.3)
fig2.show()
```





Discuss the pros and cons of the second colour scheme, and give an example of when it may not be appropriate to use. (Max. 50 words) (0.5 points)

The second scheme has the advantage that the colors among the color map are easy to distinguish. The disadvantages are that this color scheme is not intuitive. I normally associate red as more difficult. Here the color with highest intensity is in the lowest workload, reason why I prefer the first color scheme. This is an example where it is not appropiate to use, since it is not intuitive and the color encoding is not the best option for this task.

Task 4 - Normalizing data & colour coding (3 points)

In this task, you will build your own "emotional satisfaction score" (ESS) using the data from Q25.

To do this, you will group the data into positive and negative feelings, sum up the relevant features, and then normalize the results to a scale of 0 to 1. Use the value for the given categorical data item in Q25 as a measure of "emotional intensity" (i.e. 'Never'= 1 and 'Always'= 5). Our "emotional satisfaction score" attempts to balance out positive and negative emotions, and so you will calculate it using the formula:

```
ESS = (positive emotions score sum) - (negative emotions score sum)
```

Afterwards, use an appropriate colour-coded plot to show differences between Bachelor, Masters, and PhD students, using responses from Q5.

Plot: 2 points

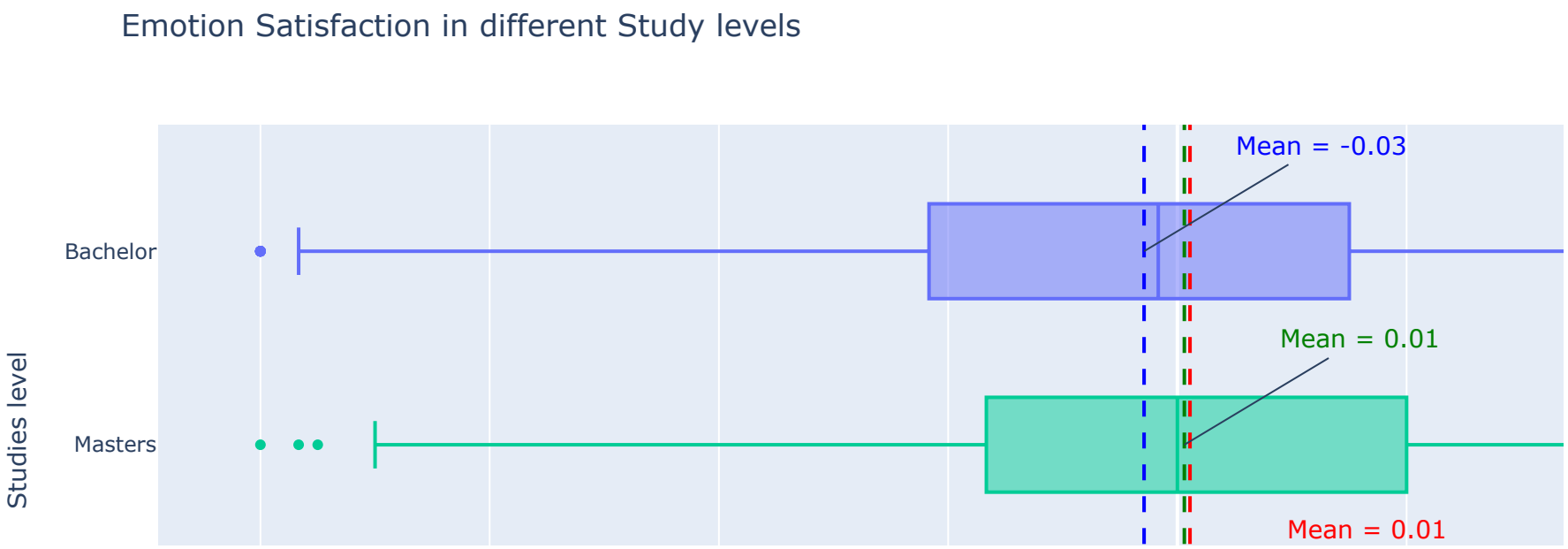
In [409…

```
df_4 = df_relevant.filter(like='Q25').copy()
#df_4 = pd.concat([df_relevant['Q5'], df_4], axis=1)

print("The dataset contains {} data records and {} features.".format(df_4.shape[0], df_4.shape[1]))

# Handling missing data

# Remove only when all the row is missing, since there are rows that are only missing a column. I don't think it is fair to
# remove all the row for just one value. Instead I replaced for 0 so it doesn't affect the ESS
df_4.dropna(how='all', inplace=True)
df_4.replace(np.nan, 0, inplace=True)

print("The dataframe after removing rows with NaN value in the specified columns contains {} data records and {} features.".fo

# Summarize specific columns

emotions={
    'Joyful':'positive',
    'Hopeful':'positive',
    'Proud':'positive',
    'Frustrated':'negative',
    'Angry':'negative',
    'Anxious':'negative',
    'Ashamed':'negative',
    'Relieved':'positive',
    'Hopeless':'negative',
    'Bored':'negative'
}

df_4.columns = emotions.keys()

positive_tags = [k for k, v in emotions.items() if v=='positive']
negative_tags = [k for k, v in emotions.items() if v=='negative']


df_4['positive'] = df_4[positive_tags].sum(axis=1)
df_4['negative'] = df_4[negative_tags].sum(axis=1)

# Normalize new columns
df_4['positive'] = ((df_4['positive']-df_4['positive'].min())/(df_4['positive'].max()-df_4['positive'].min()))
df_4['negative'] = ((df_4['negative']-df_4['negative'].min())/(df_4['negative'].max()-df_4['negative'].min()))
```

```
# Calculate final emotional satisfaction score
df_4['ESS'] = df_4['positive']-df_4['negative']

# Group by Q5, calculate average
q5 = df_relevant['Q5']
q5 = q5.replace({1.0:'Bachelor', 2.0:'Masters', 3.0:'PhD'})

df_4 = pd.concat([df_4, q5], join='inner', axis=1)

means = dict(df_4.groupby(by='Q5').mean('ESS').ESS)

# Visualization
fig = px.box(df_4, x='ESS', y='Q5', color='Q5',
             labels={
                 'Q5':'Studies level',
                 'ESS': 'Emotional Satisfaction Score'
             }, title='Emotion Satisfaction in different Study levels')
# Use a better order
fig.update_yaxes(categoryorder='array', categoryarray=['PhD', 'Masters', 'Bachelor'])
fig.add_vline(x=means['Bachelor'], line_dash='dash', line_color='blue')
fig.add_vline(x=means['Masters'], line_dash='dash', line_color='green')
fig.add_vline(x=means['PhD'], line_dash='dash', line_color='red')

fig.add_annotation(text="Mean = {}".format(np.round(means["Bachelor"],2)),
                   x=means["Bachelor"],
                   y=2,
                   ax=100,
                   ay=-60,
                   align="left",
                   font=dict(
                       size=14,
                       color="blue"
                   )
              )

fig.add_annotation(text="Mean = {}".format(np.round(means["Masters"],2)),
                   x=means["Masters"],
                   y=1,
                   ax=100,
                   ay=-60,
                   align="left",
                   font=dict(
                       size=14,
```

```
                    color="green"
                )
            )

fig.add_annotation(text="Mean = {}".format(np.round(means["PhD"],2)),
                   x=means["PhD"],
                   y=0.02,
                   ax=100,
                   ay=-60,
                   align="left",
                   font=dict(
                       size=14,
                       color="red"
                   )
            )

fig.show()
#
```

```
The dataset contains 8381 data records and 10 features.
The dataframe after removing rows with NaN value in the specified columns contains 6272 data records and 10 features.
```



Discuss your visualized results. First, describe how you treated missing data here. Second, comment on whether our initial assumption, that the negative emotions experienced by students are ultimately balanced out by their positive emotions, applies to any of the groups we visualized. If so, for which group is this the case? (Max. 100 words) (1 point)

To manage missing data, first I extracted all relevant Q25 columns. Then I dropped the rows that had only all columns missing. I figured out that it was not justified to ignore a response of a student just because he didn't answer a field. There are cases in which only one field is missing. To be able to calculate the score, I replaced the remaining NaNs for 0 so they wouldn't affect the calculations.

Regarding the assumption, according to my visualization positive emotions balance out negative emotions since the means in all categories are really close to 0. The Bachelor level is the furthest away from 0, however it is still close to 0.

Task 5 - KDD (7 points)

Taking into account the KDD process you recently learned in the lecture, we now focus on data mining. Imagine the EU wants to help the group with the worst ESS.

Use an unsupervised learning approach to help universities target their interventions appropriately. An outline of the steps you must cover is presented below:

1. Consider the answers from Q26, about worries on personal circumstances, and filter for European countries only, using the steps you've practiced above. Do not forget to also filter for the group with the lowest ESS as an outcome of Task 4.
2. Use an appropriate clustering model (K-Means, Hierarchical Clustering, DBSCAN, etc.). You can find documentation on how to implement this models at: https://scikit-learn.org/stable/unsupervised_learning.html#unsupervised-learning
3. **Visualize the results of the KDD process. Justify your visualization encoding choices using the presented course methodologies (Max. 150 words, see slides and textbook). (3 points)**
4. **Interpretation/Evaluation: Describe your KDD approach, focusing on answering these topics (Max. 250 words) (4 points):**
    - Can the students can be grouped into distinct clusters based on their worries about personal circumstances? If you've found that this is the case, what are the characteristics of these clusters?
    - What patterns can you detect in your model's results?
    - Can we identify segments of students who are particularly at high risk, due to pervasive worries across all categories?
    - What recommendation would you give to the universities, based on your visualized results?

In [446…

```
# Select columns of interest
df_5 = df_relevant.filter(like='Q26')

# Filter European countries
df_5 = pd.concat([df_relevant['Q1'], df_5], join='inner', axis=1)
european_countries = {k: v for k, v in continents_dict.items() if v== "Europe"}
df_5['Q1']=df_5['Q1'].str.replace("The Netherlands", "Netherlands")
def is_europe(x):
    if x in european_countries.keys():
        return 'Yes'
    else:
        return np.nan

df_5['Europe'] = df_5['Q1'].map(lambda x: is_europe(x))
df_5.dropna(subset=['Europe'], inplace=True)

df_5.drop('Europe', axis=1, inplace=True)

# Filter lowest ESS (Bachelor) using the Data Frame used to calculate the value.
df_5 = pd.concat([df_5, df_4['Q5']], join='inner', axis=1)
df_5['Q5'].replace('Masters', np.nan, inplace=True)
df_5['Q5'].replace('PhD', np.nan, inplace=True)
df_5.dropna(subset=['Q5'], inplace=True)

# Remove missing data
df_5 = df_5.dropna()
# We no longer need Q1 and Q5
df_5.drop(['Q1', 'Q5'], axis=1, inplace=True)
df_5.to_csv('temp.csv')
```

In [441…

```
from sklearn.cluster import KMeans

k_means = KMeans(init="k-means++", n_clusters=5, n_init=10)
results = k_means.fit(df_5.values)
results
```

Out[441]:

```
▾            KMeans

KMeans(n_clusters=5, n_init=10)
```

In [479…

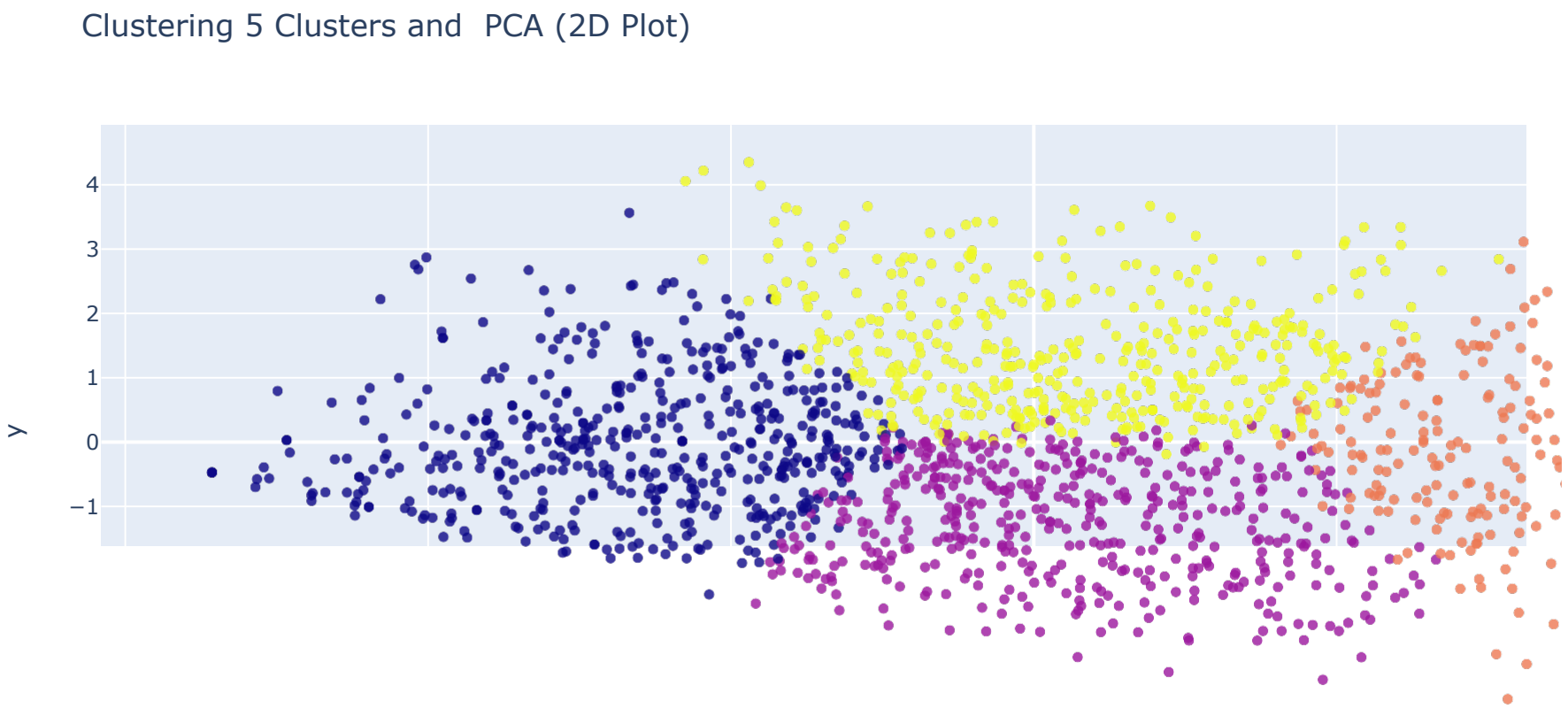
```
# Since the number of columns is 10 and I can not create a visualization that is understandable
# for every user in more than 3D I am using PCA to 3D because reduce 10 dims to 2 may imply loosing
# information
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
components = pca.fit_transform(df_5.values)

df_5_3d = df_5.copy()
df_5_3d['Cluster'] = results.labels_
df_5_3d['PCA1'] = components[:,0]
df_5_3d['PCA2'] = components[:,1]
df_5_3d['PCA3'] = components[:,2]

fig_3D = px.scatter_3d(df_5_3d, x='PCA1', y='PCA2', z='PCA3', color='Cluster', opacity=0.8,
                       hover_data=['Q26a','Q26b','Q26c','Q26d','Q26e','Q26f','Q26g','Q26h','Q26i','Q26j'],
                       labels={
                           'Q26a':'Personal physical health',
                           'Q26b': 'Personal mental health',
                           'Q26c': 'Studying issues (lectures, seminars, practical work)',
                           'Q26d': 'Future education',
                           'Q26e': 'Personal finances',
                           'Q26f': 'Family and relationship',
                           'Q26g': 'Professional career in the future',
                           'Q26h': 'COVID19 or similar pandemic crisis in the future',
                           'Q26i': 'Leisure activities',
                           'Q26j': 'Traveling abroad'
                       },
                       size_max=1,title='Clustering 5 Clusters and  PCA (3D Plot)')

fig_3D.show()
```

Clustering 5 Clusters and PCA (3D Plot)

Clear plot in 2D (above is just demostrative)

In [477…

```
# Try 2 D since it is difficult to inderstand and reduce to 4 clusters
from sklearn.cluster import KMeans

k_means = KMeans(init="k-means++", n_clusters=4, n_init=10)
results = k_means.fit(df_5.values)

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
components = pca.fit_transform(df_5.values)

df_5_2d = df_5.copy()
df_5_2d['Cluster'] = results.labels_
df_5_2d['PCA1'] = components[:,0]
df_5_2d['PCA2'] = components[:,1]

fig_2D = px.scatter(df_5_2d, x='PCA1', y='PCA2', color='Cluster', opacity=0.8,
                        hover_data=['Q26a','Q26b','Q26c','Q26d','Q26e','Q26f','Q26g','Q26h','Q26i','Q26j'],
                        labels={
                            'Q26a':'Physical health',
                            'Q26b': 'Mental health',
                            'Q26c': 'Studying issues',
                            'Q26d': 'Future education',
                            'Q26e': 'Finances',
                            'Q26f': 'Family and relationship',
                            'Q26g': 'Future career',
                            'Q26h': 'Pandemic crisis',
                            'Q26i': 'Leisure activities',
                            'Q26j': 'Traveling abroad',
                            'PCA1' : 'x',
                            'PCA2' : 'y'
                        },
                        size_max=1,title='Clustering 5 Clusters and  PCA (2D Plot)')

fig_2D.show()
```


Clustering 5 Clusters and PCA (2D Plot)
y
4
3
2
1
0
−1

QUESTION 1

First I obtained the columns of interest from the original data frame and using the continents dictionary, I removed all the non european countries. Then, using the data frame from the last task, I just kept the ones of Bachelor level since is the lowest ESS. Additionally, it was required to remove all the missing values to be able to apply an unsupervised ML technique. Finally, the not more required columns as country and education level were removed since they are not needed for clustering.

Then sklearn was used to train a KMeans clustering technique selecting 4 clusters. After testing with different values, it was decided that 4 clusters could represent fine different groups in 2D. To plot the data it is necessary to reduce the dimensionality since humans are not good visualizing more than 3D. To achieve this purpose PCA was used. It was decided that a 2D representation is clearer.

Finally, for the plotting, colors were decided taking into account the obtained Cluster. It is hard to tell how x and y were determined by PCA, however the user can see the original Q26 questions and anwers by using hovering.

#

QUESTION 2

Interpreting results from a clustering algorithm and from PCA could be tricky without having knowledge on how the algorithms work. Additionally, machines are much more capable than us to detect numerical patterns. However, by including hover and visualize the original information, it is possible to detect certain paterns.

For example the cluster in orange color is composed by students that tend to be very worried about different topics. It was observed that at least 3 categories have the answer 5 (Worried all the time). On the opposite side the blue cluster is composed by more relaxed students. Most of the answers have values 1 or 2, which means they don't tend to worry.

The clusters yellow and purple could be classified as moderately worried students. A pattern is that the lower the value in y is, the more worried they are about Leisure time, traveling abroad, possible pandemics, etc. In general the last questions. And in the opposite tendency, the higher the y value, the more worried they are about the questions in the first columns (mental and phyisical health, among others).

Based on this results it is easier to visualize that students in the orange cluster are in biggest risk and the students in the blue cluster are 'safer' so to say, which also help us to discard students that don't need immediate help. Given concrete recommendations to universities is difficult due to the wide spectrum of categories where students are worried. However, I would recommend to hire experts as psychologist and start working with students that are in the 'high-risk' cluster.

In [2]:
```
import plotly
plotly.offline.init_notebook_mode()
```

In [ ]:
```

```