week4_notebook

September 26, 2025

1 Week 4 - Data organising with Pandas

Author: Johanne Sejrskild
Date: 22.09.2025

Student: Réka Forgó (au737257@uni.au.dk)

Date of latest changes: 26.09.2025

Good afternoon
Today we are going to work with loops, condions and using ´pandas´to manipulate data. The green excercises will be highly linked to what you livecoded with Anna. If you find them challenging use yesterdays work as a help or ask. If you want to challenge yourself, try and do them all without using any help. In the yellow excercises we will do some data manipulation challenges using pandas. And we will skip the red tasks today as we have a lot on the program

Structure of the notebook:
Green excercises

Data wrangling on the iris dataset

Yellow excercises

Music sales challenge

Space mission challenge

Supervillan challenge

Start with the first excercise, and then continue in order. Feel free to work together, and see how far you can get.
The important thing is to learn, not to solve all the challenges!

---

[2]:
```
# Before we start we need to import the necessary packages
#%pip install pandas
#%pip install lxml
#%pip install scikit-learn

import pandas as pd
import requests # We might need this package to get some data from the web
```

```
from sklearn import datasets
```

Green excercises

1.1 Data organisation using a dataset about flowers

[3]:
```
flower = datasets.load_iris()

# convert to DataFrame
df = pd.DataFrame(flower.data, columns=flower.feature_names)

df.head()
```

[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Lets take a look at the data frame

[4]:
```
# There are some commands in the library pandas that can give you a quick␣
 ↪overview of the data :)

df.head()      # first 5 rows, if you put a number into the paranthesis you can␣
 ↪decide how many rows
df.tail()      # last 5 rows
df.info()      # summary of columns and types
df.describe()  # quick statistics (for numbers)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	\
count	150.000000	150.000000	150.000000	
mean	5.843333	3.057333	3.758000	
std	0.828066	0.435866	1.765298	
min	4.300000	2.000000	1.000000	

```
25%             5.100000          2.800000          1.600000
50%             5.800000          3.000000          4.350000
75%             6.400000          3.300000          5.100000
max             7.900000          4.400000          6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
75%            1.800000
max            2.500000
```

Selecting columns and rows

Try to run the cell below and figure out which output is linked to the code

[5]:
```
# If you want to select a specific column you can select it using the name:
print(df['sepal length (cm)'].head())

# If you would like to print one row, you can use the index of the row:
print(df.iloc[0])

# if you want to select a few rows of only a few columns you can also use␣
 ↪indexing:
print(df.iloc[0:3 , 0:2])  # first three rows, first two columns

# And if you want to select specific data, you can specify a single row and␣
 ↪column:
print(df.iloc[2,0])  # second row, first column

# Or use the column name:
print(df.loc[2, "sepal length (cm)"]  )
```

```
0    5.1
1    4.9
2    4.7
3    4.6
4    5.0
Name: sepal length (cm), dtype: float64
sepal length (cm)    5.1
sepal width (cm)     3.5
petal length (cm)    1.4
petal width (cm)     0.2
Name: 0, dtype: float64
   sepal length (cm)  sepal width (cm)
0                5.1               3.5
```

```
1                4.9               3.0
2                4.7               3.2
4.7
4.7
```

Subsetting data

Subsetting is the process of retrieving just the parts of large files which are of interest for a specific purpose.

This will come in handy for your projects when you have to work with potentially large data files

[6]:
```
# Let's try to select some data using conditionals

# Here we select all rows where the sepal length is larger than or equal to 5 cm
lengt_above_five = df[df["sepal length (cm)"] >= 5]
lengt_above_five.head()
```

[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
7	5.0	3.4	1.5	0.2
10	5.4	3.7	1.5	0.2

[7]:
```
# Here we select all rows where the sepal length is larger than or equal to 5␣
 ↪cm and the sepal width is less than 2,5 cm
length_and_width = df[(df["sepal length (cm)"] >= 5) & (df["sepal width (cm)"]␣
 ↪< 3.5)]
length_and_width.head()
```

[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
7	5.0	3.4	1.5	0.2
20	5.4	3.4	1.7	0.2
23	5.1	3.3	1.7	0.5
25	5.0	3.0	1.6	0.2
26	5.0	3.4	1.6	0.4

[8]:
```
# Excercise - Find the longest petal length and the median petal length
# and subset the flowers that are between the median and one centimeter shorter␣
 ↪than max length

max_length = df["petal length (cm)"].max()
median_length = df["petal length (cm)"].median()
subset = df[(df["petal length (cm)"] > median_length) & (df["petal length␣
 ↪(cm)"] < max_length - 1)]
print(f"Max length: {max_length}, Median length: {median_length}")
print(subset.head())
```

```
Max length: 6.9, Median length: 4.35
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
```

50	7.0	3.2	4.7	1.4
51	6.4	3.2	4.5	1.5
52	6.9	3.1	4.9	1.5
54	6.5	2.8	4.6	1.5
55	5.7	2.8	4.5	1.3

Sorting data

We can choose to sort our data in order of something of interest.

[9]:
```
# we could sort the data by a specific column in both ascending and descending order
df_sorted = df.sort_values(by="sepal length (cm)", ascending=False) # change direction by True/False so if you want ascending order set it to True
df_sorted.head()
```

[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
131	7.9	3.8	6.4	2.0
135	7.7	3.0	6.1	2.3
122	7.7	2.8	6.7	2.0
117	7.7	3.8	6.7	2.2
118	7.7	2.6	6.9	2.3

[10]:
```
# Excercise - sort the data by petal width in ascending order and select the 10 flowers with the smallest petal width

sorted_df = df.sort_values(by="petal width (cm)", ascending=True)
subset_10 = sorted_df.head(10)

print(subset_10)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
32	5.2	4.1	1.5	0.1
13	4.3	3.0	1.1	0.1
37	4.9	3.6	1.4	0.1
9	4.9	3.1	1.5	0.1
12	4.8	3.0	1.4	0.1
0	5.1	3.5	1.4	0.2
27	5.2	3.5	1.5	0.2
28	5.2	3.4	1.4	0.2
29	4.7	3.2	1.6	0.2
30	4.8	3.1	1.6	0.2

Flipping

Should you work with time seires data and would like to mirror (flip) your data, you can do this using pandas

[11]:
```
print(df.head(5))
```

```
reversed_df = df.iloc[::-1]   # Flipping the dataframe horisontally (reverse␣
 ↪rows)

print(reversed_df.head(5))
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
149                5.9               3.0                5.1               1.8
148                6.2               3.4                5.4               2.3
147                6.5               3.0                5.2               2.0
146                6.3               2.5                5.0               1.9
145                6.7               3.0                5.2               2.3
```

Joining Sometimes we have multiple dataframes we woudl like to add together. Maybe you have been subsetting parts of an old dataframe to substract important information and would now like join them so you can begin your analysis.

[12]:
```
# Joining a bit of the iris data with a new dataframe (we will make up some␣
 ↪data here)
first_10 = df.iloc[0:10, :]  # selecting the first 10 rows of the iris data
new_data = {"color": ["red", "blue", "green", "yellow", "purple", "red",␣
 ↪"blue", "green", "yellow", "purple"],
            "height": [80, 80, 70, 100, 90, 80, 80, 70, 100, 90]}
# Right now new_df is a dictionary, we need to convert it to a dataframe
new_df = pd.DataFrame(new_data)

#Now we join the two dataframes
joined = first_10.join(new_df, how='left') # There are 4 different types of how:
 ↪ outer, inner, left, right.

joined
```

[12]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
5                5.4               3.9                1.7               0.4
6                4.6               3.4                1.4               0.3
7                5.0               3.4                1.5               0.2
8                4.4               2.9                1.4               0.2
9                4.9               3.1                1.5               0.1
```

```
    color  height
0     red      80
1    blue      80
2   green      70
3  yellow     100
4  purple      90
5     red      80
6    blue      80
7   green      70
8  yellow     100
9  purple      90
```

Different types of how to join two data frames
This is important if your dataframes do not have the same amount of rows

left → all rows from the left DataFrame (default).

right → all rows from the right DataFrame.

inner → only rows with matching index values in both.

outer → all rows from both, fill missing with NaN.

[13]:
```
# Excercise - Which types of join (the 'how=') will work in the example above?
 ↪Try them out and see what happens

# Answer: All four types of join will work in this example because both
 ↪Dataframes have the same row indices (0 to 9).

joined_left = first_10.join(new_df, how='left')
print("left:", joined_left.shape)
display(joined_left.head())

joined_inner = first_10.join(new_df, how='inner')
print("inner:", joined_inner.shape)
display(joined_inner.head())

joined_right = first_10.join(new_df, how='right')
print("right:", joined_right.shape)
display(joined_right.head())

joined_outer = first_10.join(new_df, how='outer')
print("outer:", joined_outer.shape)
display(joined_outer.head())
```

```
left: (10, 6)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
```

```
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

    color  height
0     red      80
1    blue      80
2   green      70
3  yellow     100
4  purple      90
inner: (10, 6)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

    color  height
0     red      80
1    blue      80
2   green      70
3  yellow     100
4  purple      90
right: (10, 6)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

    color  height
0     red      80
1    blue      80
2   green      70
3  yellow     100
4  purple      90
outer: (10, 6)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

```
    color  height
0     red      80
1    blue      80
2   green      70
3  yellow     100
4  purple      90
```

[ ]:
```
# Excercise 2 - Add a row to one of the dataframes and see what happens when you join them again
first11 = df.iloc[0:11, :]

joined_with_extra = first11.join(new_df, how='left')

joined_with_extra

right_joined_with_extra = first11.join(new_df, how='right')
right_joined_with_extra
```

[ ]:
```
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                 5.1               3.5                1.4               0.2
1                 4.9               3.0                1.4               0.2
2                 4.7               3.2                1.3               0.2
3                 4.6               3.1                1.5               0.2
4                 5.0               3.6                1.4               0.2
5                 5.4               3.9                1.7               0.4
6                 4.6               3.4                1.4               0.3
7                 5.0               3.4                1.5               0.2
8                 4.4               2.9                1.4               0.2
9                 4.9               3.1                1.5               0.1
10                5.4               3.7                1.5               0.2

     color  height
0      red    80.0
1     blue    80.0
2    green    70.0
3   yellow   100.0
4   purple    90.0
5      red    80.0
6     blue    80.0
7    green    70.0
8   yellow   100.0
9   purple    90.0
10     NaN     NaN
```

Concatenating

You can also join two dataframes bu simply gluing them together.

[21]:
```
# We just made a subset of the original dataframe called 'first_10' now we find␣
 ↪the last 10 and glue them together
last_10 = df.iloc[-10:, :]    # selecting the last 10 rows using one of the␣
 ↪methods we learned above


# Now we concatenate the two dataframes together
concatenated = pd.concat( [first_10, last_10], axis=0)  # axis=0 means we␣
 ↪concatenate rows, axis=1 would concatenate columns
concatenated
```

[21]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
140	6.7	3.1	5.6	2.4
141	6.9	3.1	5.1	2.3
142	5.8	2.7	5.1	1.9
143	6.8	3.2	5.9	2.3
144	6.7	3.3	5.7	2.5
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

Now you have played around with some of the basics manipulation in pandas! Now lets jump into some challenges

Yellow excercises

OBS: To ensure you can go back in 3 months time and read you code and understand the logics behind it it needs to be well commented.
So, while you solve the yellow excercises ensure that you add some meaningful comments about the logics and coding choices.

:))

The Yellow excercises is borrowed from last years couse and written by Ethan Weed

Music sales challenge

Write a script that:

1. Combines the tables of best-selling physical singles and best-selling digital singles on the Wikipedia page "List_of_best-selling_singles"
2. Outputs the artist and single name for the year you were born. If there is no entry for that year, take the closest year after you were born.
3. Outputs the artist and single name for the year you were 15 years old.

[22]:
```
# Starter code
#musicdata = pd.read_html("https://en.wikipedia.org/wiki/
 ↪List_of_best-selling_singles")
url_music = "https://en.wikipedia.org/wiki/List_of_best-selling_singles"

# Add a User-Agent header so Wikipedia doesn't block it
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"}
response = requests.get(url_music, headers=headers)

# Pass the HTML text to pandas
musicdata = pd.read_html(response.text)


#Extracting physical and digital singles from the musicdata
physical_singles = musicdata[0]
digital_singles = musicdata[3]

physical_singles['Type'] = 'Physical'
digital_singles['Type'] = 'Digital'

# Combining the two tables
combined_singles = pd.concat([physical_singles, digital_singles])
combined_singles.head()
```

```
/var/folders/0y/tt7d7wj97hbf8tnc9hnp1_f00000gn/T/ipykernel_86483/1297234199.py:1
0: FutureWarning: Passing literal html to 'read_html' is deprecated and will be
removed in a future version. To read from a literal string, wrap it in a
'StringIO' object.
  musicdata = pd.read_html(response.text)
```

[22]:

	Artist	Single	Released	Sales (in millions)	Source	Type
0	Bing Crosby	"White Christmas"	1942	50	[1]	Physical
1	Elton John	"Something About the Way You Look Tonight"/"Ca…	1997	33	[1]	Physical
2	Bing Crosby	"Silent Night"	1935	30	[2]	Physical
3	Tino Rossi	"Petit Papa Noël"	1946	30	[3]	Physical
4	Bill Haley & His Comets	"Rock Around the Clock"				

```
4      1954                  25  [4][5]  Physical
```

[ ]:
```
# Print the artist and single from the year you were 15 years old.

subset_15 = combined_singles[combined_singles["Released"]== 2018]
subset_15
```

[ ]:
```
                            Artist                                Single  \
16                      Katy Perry                          "Firework"
21  Shakira featuring Freshlyground  "Waka Waka (This Time for Africa)"

    Released Sales (in millions) Source     Type
16      2010                  17   [59]  Digital
21      2010                  15   [61]  Digital
```

[ ]:
```
#this is just for fun and because I'm curious
subset_artist = combined_singles[combined_singles["Artist"]== "Taylor Swift"]
subset_artist
```

[ ]:
```
          Artist        Single  Released Sales (in millions) Source     Type
15  Taylor Swift  "Love Story"      2008               18[a]   [58]  Digital
```

1.2 Space challenge

1. Make a single dataframe that combines the space missions from the 1950's to the 2020's
2. Write a script that returns the year with the most launches
3. Write a script that returns the most common month for launches
4. Write a script that ranks the months from most launches to fewest launches

[73]:
```
# Starter code.
url_space =  "https://en.wikipedia.org/wiki/
 ↪Timeline_of_Solar_System_exploration"

# Add a User-Agent header so Wikipedia doesn't block it
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"}
response = requests.get(url_space, headers=headers)

# Pass the HTML text to pandas
spacedata = pd.read_html(response.text)

# combine all tables into data frame
combined_space = pd.concat(spacedata, ignore_index = True)

# Dropping column we dont need
combined_space = combined_space.iloc[:, 0:3]
combined_space.head()
```

```
/var/folders/0y/tt7d7wj97hbf8tnc9hnp1_f00000gn/T/ipykernel_86483/4030390085.py:9
```

```
: FutureWarning: Passing literal html to 'read_html' is deprecated and will be
removed in a future version. To read from a literal string, wrap it in a
'StringIO' object.
  spacedata = pd.read_html(response.text)
```

[73]:
```
   Mission name      Launch date  \
0    Sputnik 1   4 October 1957
1    Sputnik 2  3 November 1957
2   Explorer 1  1 February 1958
3   Vanguard 1    17 March 1958
4       Luna 1   2 January 1959

                                        Description
0                                First Earth orbiter
1  Earth orbiter, first animal in orbit, a dog na…
2  Earth orbiter; discovered Van Allen radiation …
3  Earth orbiter; oldest spacecraft still in Eart…
4  First lunar flyby (attempted lunar impact?); f…
```

[ ]:
```
## The year with the most launches

#need to clean data
combined_space['Launch Year'] = pd.to_datetime(combined_space['Launch date']).
 ↪dt.year




# Counting the number of launches per year
launch_counts = combined_space['Launch Year'].value_counts()
most_launches_year = launch_counts.idxmax()
most_launches_count = launch_counts.max()
combined_space.head()

print(f"The year with the most launches is {most_launches_year} with␣
 ↪{most_launches_count} launches.")

#there is some missing data in the launch date column, and that's why the year␣
 ↪is a float.
```

```
The year with the most launches is 1965.0 with 12 launches.
```

[81]:
```
# The month with the most launches
months = {
    1: "January", 2: "February", 3: "March", 4: "April",
    5: "May", 6: "June", 7: "July", 8: "August",
    9: "September", 10: "October", 11: "November", 12: "December"
}
```

```
combined_space['Launch Month'] = pd.to_datetime(combined_space['Launch date']).
 ↪dt.month
launch_month_counts = combined_space['Launch Month'].value_counts()
most_launches_month = launch_month_counts.idxmax()
most_launches_month_count = launch_month_counts.max()
print(f"The month with the most launches is {months[most_launches_month]} with␣
 ↪{most_launches_month_count} launches.")
```

```
The month with the most launches is November with 30 launches.
```

[83]:
```
# Ranking of months with the most to the fewest launches
# Sorting the counts in descending order
sorted_month_counts=launch_month_counts.sort_values(ascending=False)
print("Ranking of months with the most to the fewest launches:")
for month, count in sorted_month_counts.items():
    print(f"{months[month]}:{count} launches")
```

```
Ranking of months with the most to the fewest launches:
November:30 launches
August:27 launches
September:25 launches
October:24 launches
January:21 launches
July:21 launches
December:19 launches
February:18 launches
May:18 launches
March:15 launches
June:14 launches
April:13 launches
```

1.3 Supervillain challenge

1. Write a script that combines the tables showing supervillain debuts from the 30's through the 2010's
2. Write a script that ranks each decade in terms of how many supervillains debuted in that decade
3. Write a script that ranks the different comics companies in terms of how many supervillains they have, and display the results in a nice table (pandas dataframe)

[94]:
```
#supervillandata = pd.read_html("https://en.wikipedia.org/wiki/
 ↪List_of_comic_book_supervillain_debuts")

url_villan = "https://en.wikipedia.org/wiki/
 ↪List_of_comic_book_supervillain_debuts"

# Add a User-Agent header so Wikipedia doesn't block it
```

```
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"}
response = requests.get(url_villan, headers=headers)

# Pass the HTML text to pandas
supervillandata = pd.read_html(response.text)

# combine all tables into data frame
df_supervillan = pd.concat(supervillandata, ignore_index = True)
df_supervillan[:10]
```

```
/var/folders/0y/tt7d7wj97hbf8tnc9hnp1_f00000gn/T/ipykernel_86483/4250549170.py:1
0: FutureWarning: Passing literal html to 'read_html' is deprecated and will be
removed in a future version. To read from a literal string, wrap it in a
'StringIO' object.
  supervillandata = pd.read_html(response.text)
```

[94]:

	0	1	Character / Team	\
0	NaN	This article has multiple issues. Please help …	NaN	
1	NaN	This article includes a list of general refere…	NaN	
2	NaN	This article needs additional citations for ve…	NaN	
3	NaN	This article includes a list of general refere…	NaN	
4	NaN	This article needs additional citations for ve…	NaN	
5	NaN	NaN	Ultra-Humanite	
6	NaN	NaN	Dr. Death	
7	NaN	NaN	The Monk	
8	NaN	NaN	The Claw	
9	NaN	NaN	Hath-Set	

	Year Debuted	Company	Creator/s	\
0	NaN	NaN	NaN	
1	NaN	NaN	NaN	
2	NaN	NaN	NaN	
3	NaN	NaN	NaN	
4	NaN	NaN	NaN	
5	1939 (June)	DC	Jerry Siegel, Joe Shuster	
6	1939 (July)	DC	Bob Kane, Bill Finger	
7	1939 (September)	DC	Bob Kane, Bill Finger	
8	1939 (December)	Lev Gleason Publications	Jack Cole	
9	1940 (January)	DC	Gardner Fox, Dennis Neville	

	First Appearance
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
5	Action Comics (vol. 1) #13

```
6  Detective Comics (vol. 1) #29
7  Detective Comics (vol. 1) #31
8        Silver Streak Comics #1
9                Flash Comics #1
```

[ ]:
```
# 1. Write a script that combines the tables showing supervillain debuts from␣
 ↪the 30's through the 2010's

#remove rows if year debuted is Na-s
clean = df_supervillan.dropna(subset=['Year Debuted']).copy()


# 2. Which decade had the most supervillain debuts?
#make year deputed clean, splitting the year from the rest of the string
clean['Year'] = clean['Year Debuted'].str.extract(r'(\d{4})')

clean['Year'] = pd.to_numeric(clean['Year'])


#find min and max value of year
min_year = decade_counts['Year'].min()
max_year = decade_counts['Year'].max()
print(f"The minimum year is {min_year} and the maximum year is {max_year}")


#group by decade
clean['Decade'] = (clean['Year'] // 10) * 10

clean.head()
```

```
The minimum year is 1939 and the maximum year is 2019
```

[ ]:
```
     0    1 Character / Team      Year Debuted                    Company  \
5 NaN  NaN   Ultra-Humanite       1939 (June)                         DC
6 NaN  NaN        Dr. Death       1939 (July)                         DC
7 NaN  NaN         The Monk  1939 (September)                         DC
8 NaN  NaN         The Claw   1939 (December)  Lev Gleason Publications
9 NaN  NaN         Hath-Set    1940 (January)                         DC

                    Creator/s              First Appearance  Year  Decade
5    Jerry Siegel, Joe Shuster     Action Comics (vol. 1) #13  1939    1930
6        Bob Kane, Bill Finger  Detective Comics (vol. 1) #29  1939    1930
7        Bob Kane, Bill Finger  Detective Comics (vol. 1) #31  1939    1930
8                   Jack Cole        Silver Streak Comics #1  1939    1930
9  Gardner Fox, Dennis Neville                Flash Comics #1  1940    1940
```

[ ]:
```
# 2. Write a script that ranks each decade in terms of how many supervillains␣
 ↪debuted in that decade

def decade_with_most_debuts(df):
    decade_counts = df['Decade'].value_counts().sort_index()
    most_debuts_decade = decade_counts.idxmax()
    most_debuts_count = decade_counts.max()
    return most_debuts_decade, most_debuts_count, decade_counts

most_debuts_decade, most_debuts_count, decade_counts =␣
 ↪decade_with_most_debuts(clean)
print(f"The decade with the most supervillain debuts is the␣
 ↪{most_debuts_decade}s with {most_debuts_count} debuts.")
#print in order
print("Ranking of decades by supervillain debuts:")
for decade, count in decade_counts.sort_values(ascending=False).items():
    print(f"{decade}s: {count} ")
```

```
The decade with the most supervillain debuts is the 1960s with 228 debuts.
Ranking of decades by supervillain debuts:
1960s: 228
1970s: 97
1980s: 91
1990s: 84
2000s: 50
1940s: 47
1950s: 26
2010s: 14
1930s: 4
```

[119]:
```
# 3. Write a script that ranks the different comics companies in terms of how␣
 ↪many supervillains they have, and display the results

company_counts = clean['Company'].value_counts()
print("Ranking of comic companies by number of supervillains:")
for company, count in company_counts.items():
    print(f"{company}: {count} ")
```

```
Ranking of comic companies by number of supervillains:
DC: 337
Marvel: 270
Fawcett Comics/DC: 6
Image: 5
Dark Horse: 5
Marvel/Timely: 4
Disney/Hyperion: 4
Eternity: 3
Lev Gleason Publications: 1
```

```
Comico: 1
Mirage: 1
Image Comics: 1
```