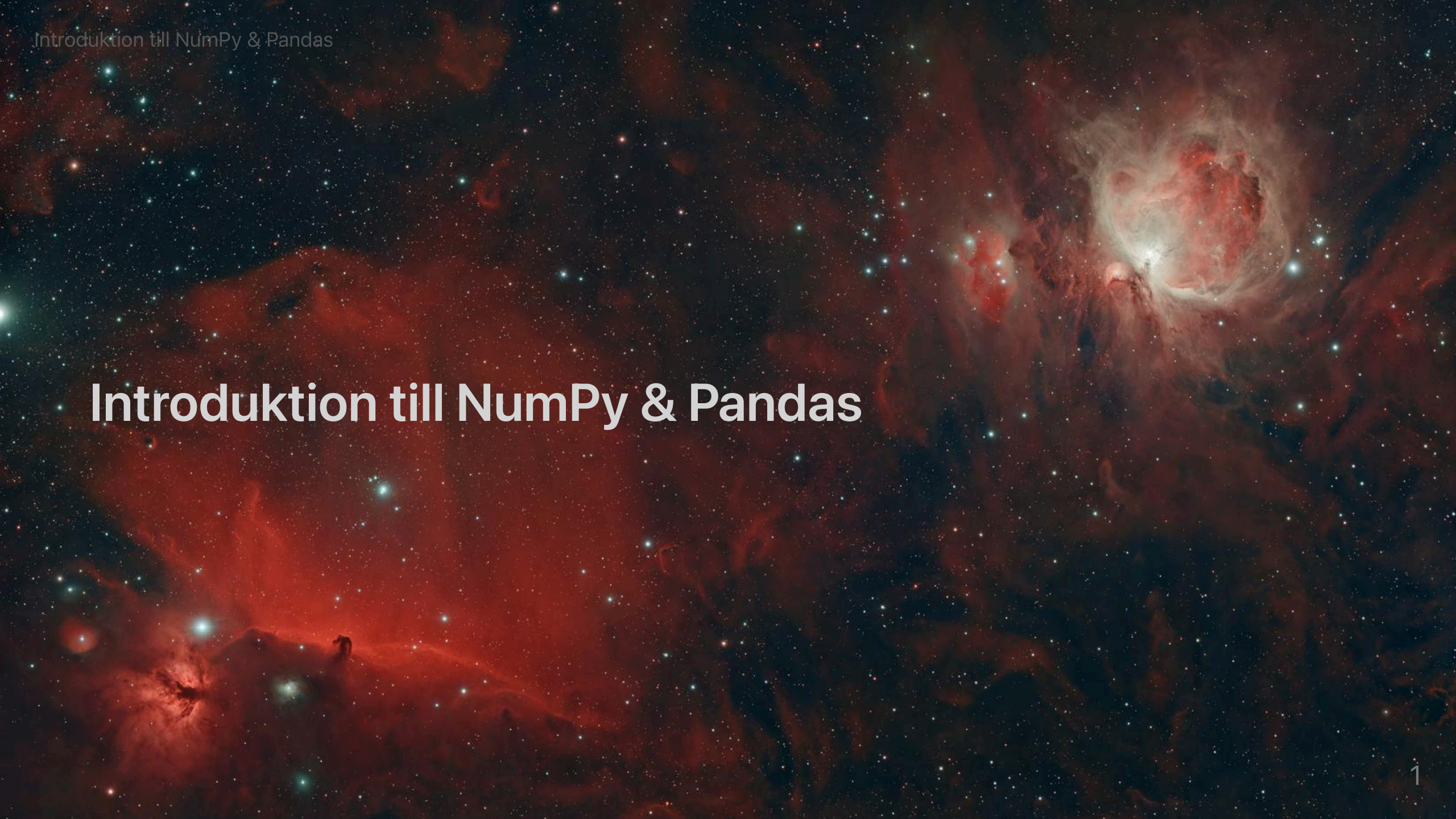


Introduktion till NumPy & Pandas

NumPy

- "Numerical Python"
- Arbetar med numerisk data i Python
- Främst *vektorer* och *matriser*
- Grundbult för annan datahantering, ex.vis Pandas

Vad är en vektor?

- En vektor är en endimensionell samling av tal.
- Representerar både storlek (magnitud) och riktning.
- Vanligtvis uttryckt som en lista, t.ex. `[1, 2, 3]`.
- Används inom matematik, fysik och datavetenskap för att beskriva positioner, krafter med mera.
- Exempelvis aktiekurs, försäljning, temperatur eller lufttryck över tid.

Vad är en matris?

- En matris är en tvådimensionell samling av tal ordnade i rader och kolumner.
- Exempel på en 2x3-matris: `[[1, 2, 3], [4, 5, 6]]`
- Exempelvis tabell, Excel-ark, bild/video, etc.

Import av biblioteket

```
import numpy
```

eller, bättre

```
import numpy as np
```

Enkla matematiska vektor-operationer

```
myvector = [1,2,3]
mean_value = np.mean(myvector)            # 2
max_value = np.max(myvector)              # 3
max_value_index = np.argmax(myvector)     # 2 (obs. python-indexering)
sum_value = np.sum(myvector)              # 6
```

Andra vektor-operationer

Listupprepning

```
mult = myvector*2                    # [1, 2, 3, 1, 2, 3]
```

Kanske bättre (beroende på vad som eftersträvas): numeriska operationer på *elementnivå*.

```
myvector = [1,2,3]
myvector = np.array(myvector)
mult = myvector*2                    # [2, 4, 6]
```

NumPy-arrayer är optimerade för numeriska beräkningar och mycket snabbare än Python-listor vid stora datamängder.

Numerisk operation utan NumPy

```
vector = [1, 2, 3]
scalar = 2
result = [x * scalar for x in vector]
```

Vilket sätt är snabbast?

```
import numpy as np
import time

size = 10**7                                # lista med 10 miljoner element
python_list = list(range(size))
scalar = 2
```

Python-lista

```
start_time = time.time()
python_result = [x * scalar for x in python_list]
python_duration = time.time() - start_time
print(f"Python-lista: {python_duration:.4f} seconds")   # 0.2511 s
```

NumPy-array

```
numpy_array = np.array(python_list)     # <-- obs. tidskrävande!
start_time = time.time()
numpy_result = numpy_array * scalar
numpy_duration = time.time() - start_time
print(f"NumPy-array: {numpy_duration:.4f} seconds")     # 0.0082 s
```

- NumPy-array-multiplikationen är ca 30 gånger snabbare här.
- Dock - gå till botten med *alla* potentiella tidstjuvar.
- Här är `np.array(python_list)` en tidskrävande operation.
    - `np.arange(size)` är mycket snabbare.

Skapa NumPy-array

```
size = 10
myarray = np.zeros(size)

# Fyll array:en med värden
for ii in range(size):
    myarray[ii] = ii**2     # [ 0.  1.  4.  9. 16. 25. 36. 49. 64. 81.]

# Funkar så här också:
mylist = []
for ii in range(size):
        mylist.append(ii**2)
mylist = np.array(mylist)

# eller (klart smidigast i just detta fall)
myfastarray = np.arange(size)**2
```

Sortering

```
np.random.seed(42)
nn = 10
random_array = np.random.rand(nn)
sorted_array = np.sort(random_array)

print("Slumpmässig array:", random_array)
print("Sorterad array:", sorted_array)
```

Utmatning:

```
 Slumpmässig array: [0.37454012 0.95071431 0.73199394 0.59865848 0.15601864 0.15599452
0.05808361 0.86617615 0.60111501 0.70807258]
 Sorterad array: [0.05808361 0.15599452 0.15601864 0.37454012 0.59865848 0.60111501
0.70807258 0.73199394 0.86617615 0.95071431]
```

Sortering, forts.

```
sorted_indices = np.argsort(random_array)
print("Index för sorterad array:", sorted_indices)
```

Utmatning:

```
Index för sorterad array: [6 5 4 0 3 8 9 2 7 1]
```

Matriser

- **Varför matriser?**
  - Effektiv lagring och beräkning av stora datamängder.
  - Grundläggande verktyg inom vetenskaplig beräkning, statistik och maskininlärning.
- **Exempel på en enkel matris (2x2):**

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

- **Skapa en NumPy-array i matris-form:**

```
import numpy as np
matrix = np.array([[1, 2], [3, 4]])   # 2x2-matrix
print(matrix)
```

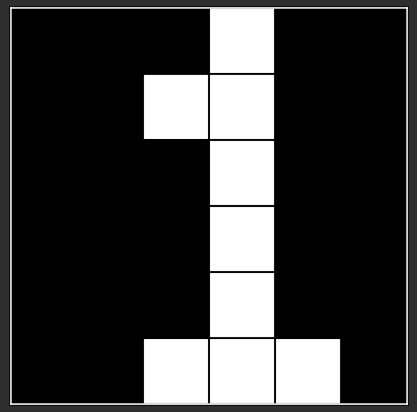
- Utmatning

```
[[1 2]
[3 4]]
```

- **Annat matris-exempel**

```
import numpy as np
matrix = np.array([
  [0, 0, 0, 1, 0, 0],
  [0, 0, 1, 1, 0, 0],
  [0, 0, 0, 1, 0, 0],
  [0, 0, 0, 1, 0, 0],
  [0, 0, 0, 1, 0, 0],
  [0, 0, 1, 1, 1, 0]])
```

Matrisplot (detaljer nästa gång)



Grundläggande operationer:

- Addition: `matrix + 1` lägger till 1 till varje element.
- Multiplikation: `matrix * 2` multiplicerar varje element med 2.
- Matrismultiplikation: `matrix.dot(matrix)` eller `np.dot(matrix, matrix)` för att beräkna produkt av matriser.

Varför detta är användbart?

- Effektiv beräkning: NumPy är optimerat i C, vilket ger snabbare operationer.
- Vektoriserade operationer: Arbeta med hela datamängder direkt utan loopar.

Tillämpningar av NumPy-matriser

- Dataanalys, statistik och maskininlärning.
- Grafiska beräkningar, fysikaliska simuleringar och optimeringsproblem.
- Koppling till Pandas för bl. a. tabellhantering
- Matriser i NumPy $\Rightarrow$ solid grund inom dataanalys i Python.

Sammanfattning, NumPy

- Kraftfullt bibliotek i Python för numeriska beräkningar
- Stödjer högpresterande operationer på stora datastrukturer
- Olika matematiska funktioner för beräkningar direkt på arrayer
- Kärnverktyg inom vetenskaplig beräkning, dataanalys och maskininlärning

Pandas

- "Python Data Analysis Library"
- Strukturerad data – Hantera tabeller med `DataFrame`
- Kraftfull manipulation – Filtrera, gruppera och transformera data enkelt.
- Flexibel import/export – Stöd för CSV, Excel, SQL, JSON m.m.

Vi börjar med ett exempel:

```
import pandas as pd

data = {
  'Namn': ['Alice', 'Bob', 'Charlie'],
  'Ålder': [38, 29, 56],
  'Stad': ['Stockholm', 'Göteborg', 'Malmö']
}

print(f'data:' {data})

df = pd.DataFrame(data)

print(f'\nDataFrame: {df}')
```

Utmatning:

```
data:
{'Namn': ['Alice', 'Bob', 'Charlie'], 'Ålder': [38, 29, 56],
'Stad': ['Stockholm', 'Göteborg', 'Malmö']}

DataFrame:
      Namn  Ålder       Stad
0    Alice     38  Stockholm
1      Bob     29   Göteborg
2  Charlie     56      Malmö
```

Exempelanvändning

Plocka ut en specifik kolumn:

```
print(df['Namn'].values)
```

Utmatning:

```
['Alice' 'Bob' 'Charlie']
```

Medelålder?

```
print(df['Ålder'].mean())                # 41
```

eller

```
print(np.mean(df['Ålder'].values))       # 41
```

Hur många bor i viss stad?

```
print(df['Stad'].value_counts())
```

Utmatning:

```
Stad
Stockholm    1
Göteborg     1
Malmö        1
```

Hur många bor i viss stad, sorterat?

```
print(df['Stad'].value_counts().sort_values(ascending=False))
```

(Samma utmatning igen i detta enkla exempel)

Spara DataFrame till Excel-fil

```
df.to_excel('excelfil.xlsx', index=False)
```

Läs in Excel-fil

```
df = pd.read_excel('excelfil.xlsx')
```

Större dataset

- `video_games_sales.csv` (CSV = comma-separated values. Relativt enkel filstruktur)
- ca 16000 TV-spel
- Namn, plattform, år, utvecklare, och mer (16 kolumner)

Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_Score	Critic_Count	User_Score	User_Count	Developer
Wii Sports	Wii	2006	Sports	Nintendo	41.36	28.96	3.77	8.45	82.53	76	51	8	322	Nintendo
Super Mario Bros.	NES	1985	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24					
Mario Kart Wii	Wii	2008	Racing	Nintendo	15.68	12.76	3.79	3.29	35.52	82	73	8.3	709	Nintendo
Wii Sports Resort	Wii	2009	Sports	Nintendo	15.61	10.93	3.28	2.95	32.77	80	73	8	192	Nintendo
Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	Nintendo	11.27	8.89	10.22	1	31.37					
Tetris	GB	1989	Puzzle	Nintendo	23.2	2.26	4.22	0.58	30.26					
New Super Mario Bros.	DS	2006	Platform	Nintendo	11.28	9.14	6.5	2.88	29.8	89	65	8.5	431	Nintendo
Wii Play	Wii	2006	Misc	Nintendo	13.96	9.18	2.93	2.84	28.92	58	41	6.6	129	Nintendo
New Super Mario Bros. Wii	Wii	2009	Platform	Nintendo	14.44	6.94	4.7	2.24	28.32	87	80	8.4	594	Nintendo
Duck Hunt	NES	1984	Shooter	Nintendo	26.93	0.63	0.28	0.47	28.31					
Nintendogs	DS	2005	Simulation	Nintendo	9.05	10.95	1.93	2.74	24.67					
Mario Kart DS	DS	2005	Racing	Nintendo	9.71	7.47	4.13	1.9	23.21	91	64	8.6	464	Nintendo
Pokemon Gold/Pokemon Silver	GB	1999	Role-Playing	Nintendo	9	6.18	7.2	0.71	23.1					
Wii Fit	Wii	2007	Sports	Nintendo	8.92	8.03	3.6	2.15	22.7	80	63	7.7	146	Nintendo
Kinect Adventures!	X360	2010	Misc	Microsoft Game Studios	15	4.89	0.24	1.69	21.81	61	45	6.3	106	Good Science Studio
Wii Fit Plus	Wii	2009	Sports	Nintendo	9.01	8.49	2.53	1.77	21.79	80	33	7.4	52	Nintendo
Grand Theft Auto V	PS3	2013	Action	Take-Two Interactive	7.02	9.09	0.98	3.96	21.04	97	50	8.2	3994	Rockstar North
Grand Theft Auto: San Andreas	PS2	2004	Action	Take-Two Interactive	9.43	0.4	0.41	10.57	20.81	95	80	9	1588	Rockstar North
Super Mario World	SNES	1990	Platform	Nintendo	12.78	3.75	3.54	0.55	20.61					
Brain Age: Train Your Brain in Minutes a Day	DS	2005	Misc	Nintendo	4.74	9.2	4.16	2.04	20.15	77	58	7.9	50	Nintendo
Pokemon Diamond/Pokemon Pearl	DS	2006	Role-Playing	Nintendo	6.38	4.46	6.04	1.36	18.25					
Super Mario Land	GB	1989	Platform	Nintendo	10.83	2.71	4.18	0.42	18.14					
Super Mario Bros. 3	NES	1988	Platform	Nintendo	9.54	3.44	3.84	0.46	17.28					
Grand Theft Auto V	X360	2013	Action	Take-Two Interactive	9.66	5.14	0.06	1.41	16.27	97	58	8.1	3711	Rockstar North
Grand Theft Auto: Vice City	PS2	2002	Action	Take-Two Interactive	8.41	5.49	0.47	1.78	16.15	95	62	8.7	730	Rockstar North
Pokemon Ruby/Pokemon Sapphire	GBA	2002	Role-Playing	Nintendo	6.06	3.9	5.38	0.5	15.85					
Brain Age 2: More Training in Minutes a Day	DS	2005	Puzzle	Nintendo	3.43	5.35	5.32	1.18	15.29	77	37	7.1	19	Nintendo
Pokemon Black/Pokemon White	DS	2010	Role-Playing	Nintendo	5.51	3.17	5.65	0.8	15.14					

Läs in filen

```
df = pd.read_csv('video_games_sales.csv')
```

Skriv ut de fem vanligaste genrerna

```
top_5_genres = df['Genre'].value_counts().head(5)
```

Utmatning:

```
Genre
Action          3370
Sports          2348
Misc            1750
Role-Playing    1500
Shooter         1323
```

Lägg till kolumn

```
df['Random_Number'] = np.random.rand(len(df))
print(df['Random_Number'])
```

Utmatning av den kolumnen

```
0         0.669603
1         0.199000
2         0.254762
3         0.730626
4         0.735303
           ...
16714     0.283945
16715     0.239567
16716     0.077229
16717     0.952326
16718     0.197140
```

Lägg till rad

```
new_row = {
  'Name': 'Zombie Panda',
  'Platform': 'PC',
  'Year_of_Release': 2025,
  'Genre': 'Action',
  'Publisher': 'LNU',
  'NA_Sales': 0.5,
  'EU_Sales': 0.4,
  'JP_Sales': 0.1,
  'Other_Sales': 0.2,
  'Global_Sales': 1.2,
  'Critic_Score': 85,
  'Critic_Count': 40,
  'User_Score': 8.5,
  'User_Count': 2000,
  'Developer': 'Indie Studio',
  'Rating': 'E',
  'Random_Number': np.random.rand()
}
```

Lägg till raden i df och skriv ut den

```
df = pd.concat([df, pd.DataFrame([new_row])], ignore_index=True)
print(df.tail(1))
```

Utmatning (trunkerad)

```
       Name          Platform  Year_of_Release  ...  Developer     Rating  Random_Number
16719  Zombie Panda  PC        2025             ...  LNU           E       0.710553
```

Dataanalys

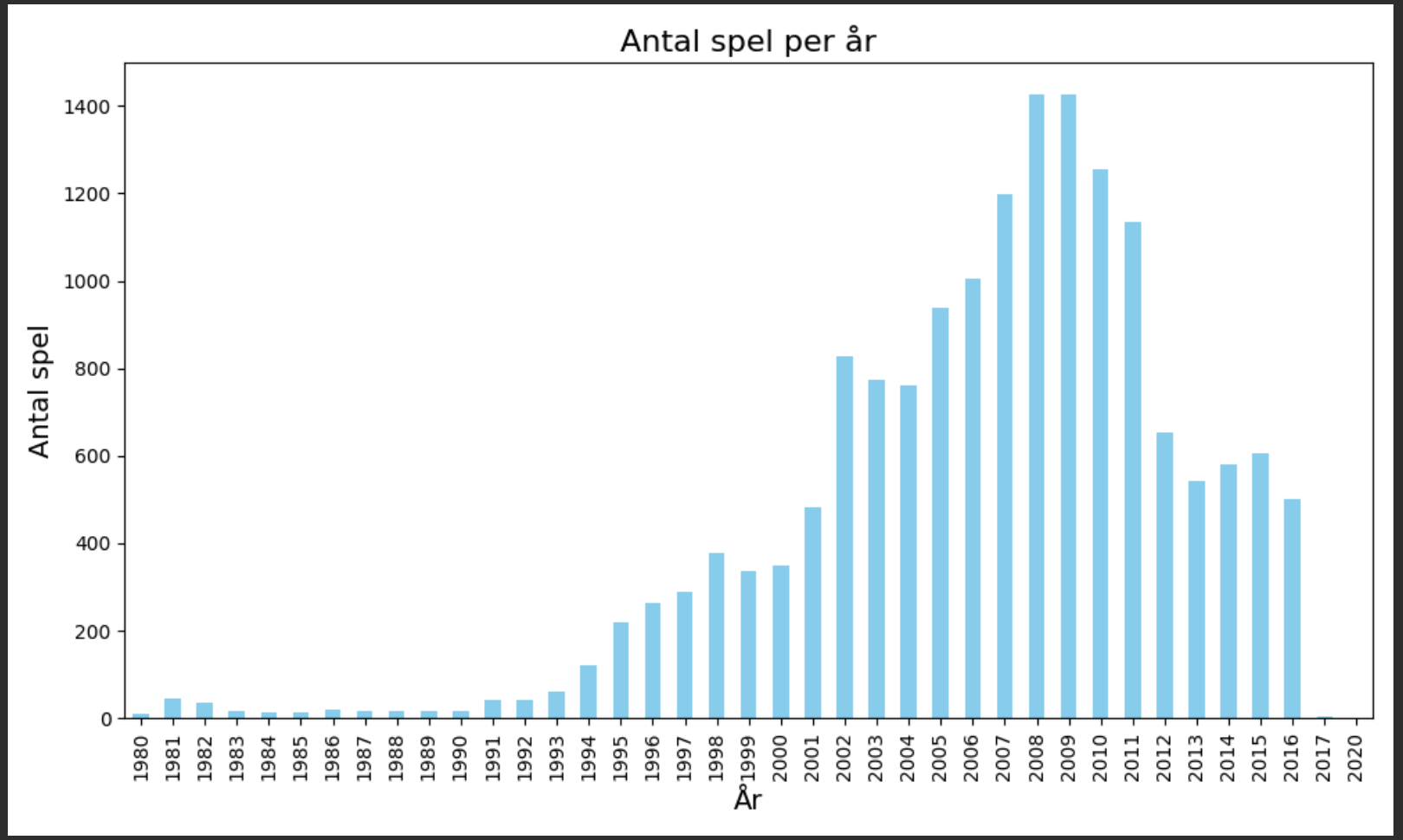
Se utvecklingen av antalet spelsläpp per år

```
games_per_year = df['Year_of_Release'].value_counts().sort_index()
```

Utmatning:

```
Year_of_Release
1980       9
1981      46
1982      36
1983      17
1984      14
1985      14
...      ...
```

Plotta ett histogram över utvecklingen (plot-detaljer nästa föreläsning):



Sammanfattning, Pandas

- Flexibla datalagringsstrukturer som DataFrames.
- Enkelt att manipulera, analysera och visualisera data i tabellform.
- Funktioner för import och export av data mellan olika format som CSV och Excel.
- Bred användning inom dataanalys och maskininlärning för stora dataset
- Dvs, har ni *mycket* och *bra* data i xlsx-format (eller motsv.), funkar Pandas som en utmärkt brygga mellan er data och en maskininlärningsmodell