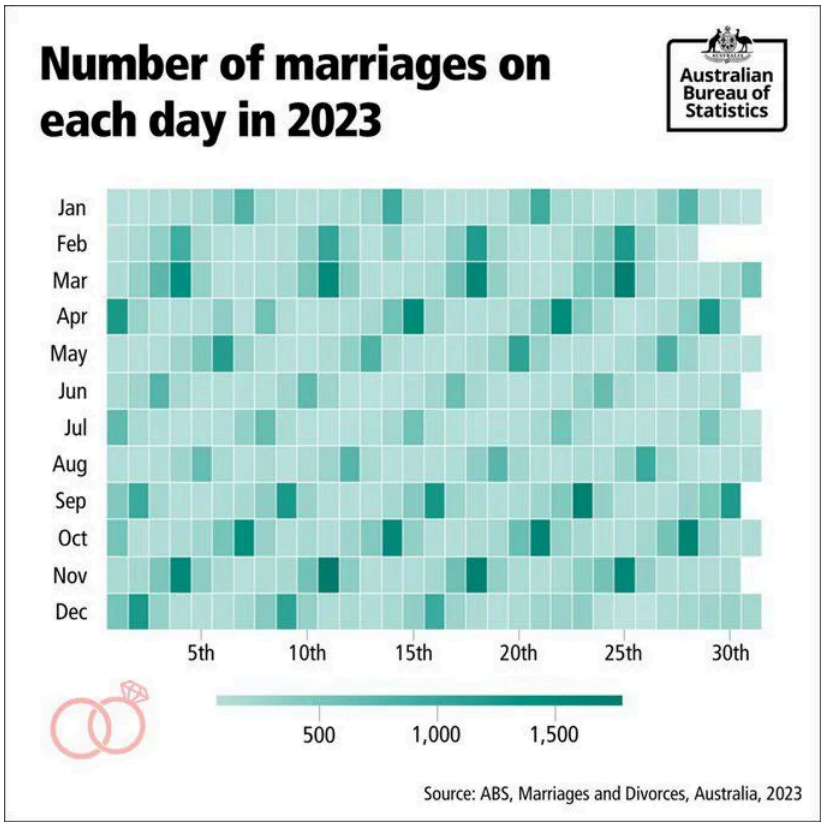
Seminar 2



Analyzing the data

- This image posted by the Australian Bureau of Statistics is a sort of heatmap that shows the numer of marriages per day in 2023.
- It captures the evolution of a time dependent value, so we are dealing with sequential data.

Drawbacks of representation

- you can't tell that the measured value reaches a spike every Sunday unless you manually check the number of days between spikes
- the aestetics makes the plot look noisy, viewer may be more distracted by the diagonal stripes formed by the hot points
- The color map choice makes it hard to tell the difference between adjacent lower values close to each other

In [1]:
```
import numpy as np, matplotlib.pyplot as plt, pandas as pd
```

Reproducing the pattern

- Since I don't have access to the actual data, I'll generate synthetic data that mimics the behavior of the chosen plot.
- The goal: generate spikes around Sundays

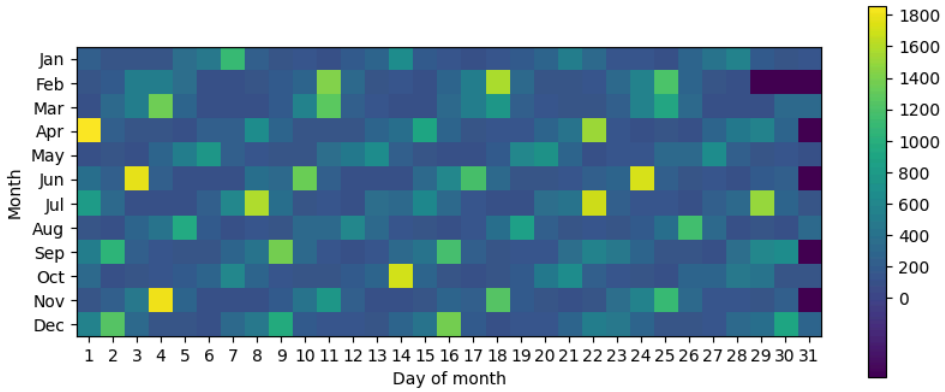
In [57]:
```
x = np.array(range(365))
y = np.zeros_like(x)
s = np.zeros_like(x)

for i in range(len(x)):
    noise_val = np.random.randint(400,600)
    s[i] = np.exp(3-min(i%7,4-i%7))//20
    spike_val = s[i]*(200+np.random.randint(800))
    y[i] = (noise_val + spike_val)/4
y[:10], s[:14]
```

Out[57]:
```
(array([ 212,  127,  121,  118,  358,  471, 1095,  230,  132,  143]),
 array([1, 0, 0, 0, 1, 2, 7, 1, 0, 0, 0, 1, 2, 7]))
```

In [69]:
```
days_of_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
m, d = 0, 0

month_of_day = np.zeros_like(x)

heatmap = -500*np.ones((12, 31))

for i in range(len(x)):
    month_of_day[i] = m
    heatmap[m,d] = y[i]
    d += 1
    if d==days_of_month[m]:
        d = 0
        m += 1
```

Simulating the roughly equivalent vizualization that the chart employs:

In [77]:
```
plt.figure(figsize=(10,4))
plt.imshow(heatmap)
plt.xlabel('Day of month')
plt.xticks(range(31), np.arange(1,32,1))
plt.ylabel('Month')
plt.yticks(range(12),['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'])
plt.colorbar(ticks=np.arange(0, 1900, 200))
plt.show()
```

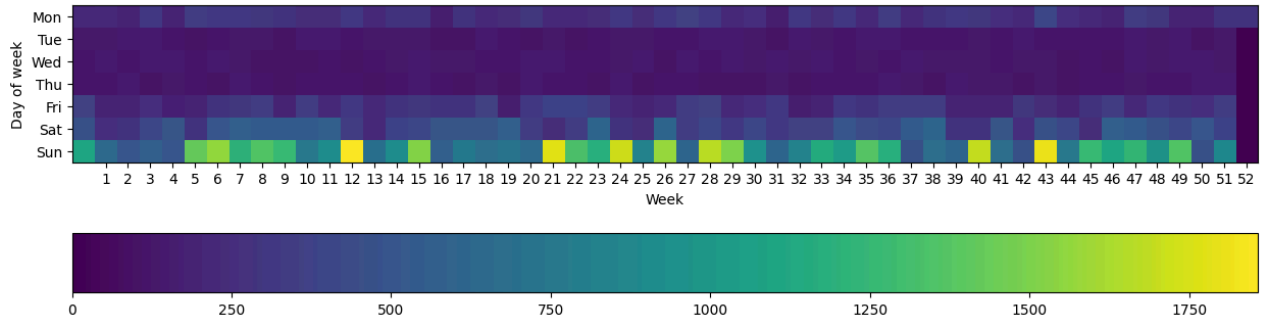


Suggesting fixes

First we can literally highlight the pattern by aligning the heatmap to match the days of the week

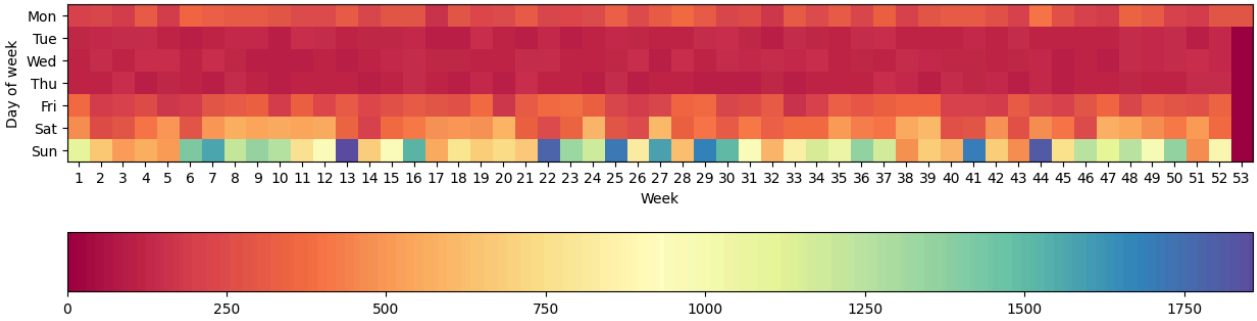
In [78]:
```
heatmap7 = np.zeros((7, 53))
for i in range(len(x)):
    heatmap7[i%7, i//7] = y[i]
```

In [84]:
```
plt.figure(figsize=(15,6))
plt.imshow(heatmap7)
plt.colorbar(orientation='horizontal', location='bottom')
plt.xticks(np.arange(1,53,1))
plt.yticks(range(7), ['Mon','Tue','Wed','Thu','Fri','Sat','Sun'])
plt.xlabel("Week")
plt.ylabel("Day of week")
plt.show()
```



Now we can clearly see the intended trend of marriages occuring on Sundays. Let's fix the closely valued adjacent points. Let's use a divergent color map

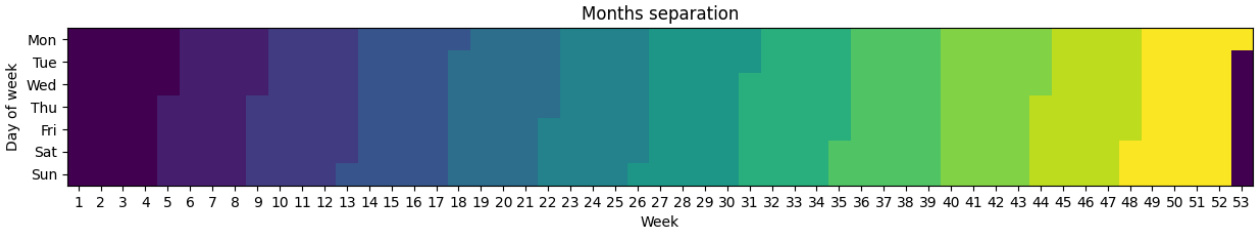
In [119]:
```
plt.figure(figsize=(15,6))
plt.imshow(heatmap7, cmap=plt.get_cmap('Spectral'))
plt.colorbar(orientation='horizontal', location='bottom')
plt.xticks(range(53), np.arange(1,54,1))
plt.yticks(range(7), ['Mon','Tue','Wed','Thu','Fri','Sat','Sun'])
plt.xlabel("Week")
plt.ylabel("Day of week")
plt.show()
```



This plot completely ignores the months, which were highlighted in the original representation.

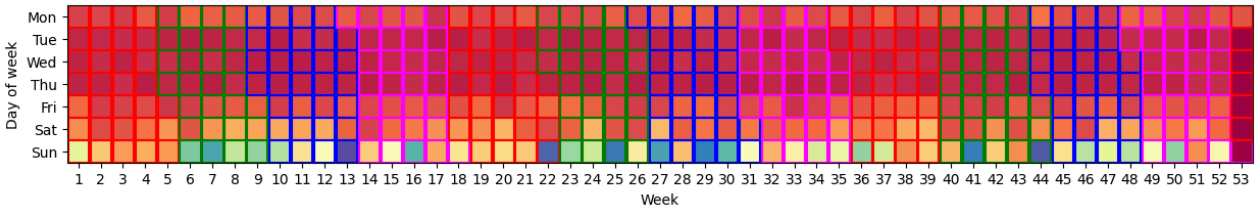
In [88]:
```
heatmapM = np.zeros((7, 53))
for i in range(len(x)):
    heatmapM[i%7, i//7] = month_of_day[i]
```

In [120]:
```
plt.figure(figsize=(15,6))
plt.imshow(heatmapM)
plt.xticks(range(53), np.arange(1,54,1))
plt.yticks(range(7), ['Mon','Tue','Wed','Thu','Fri','Sat','Sun'])
plt.xlabel("Week")
plt.ylabel("Day of week")
plt.title('Months separation')
plt.show()
```



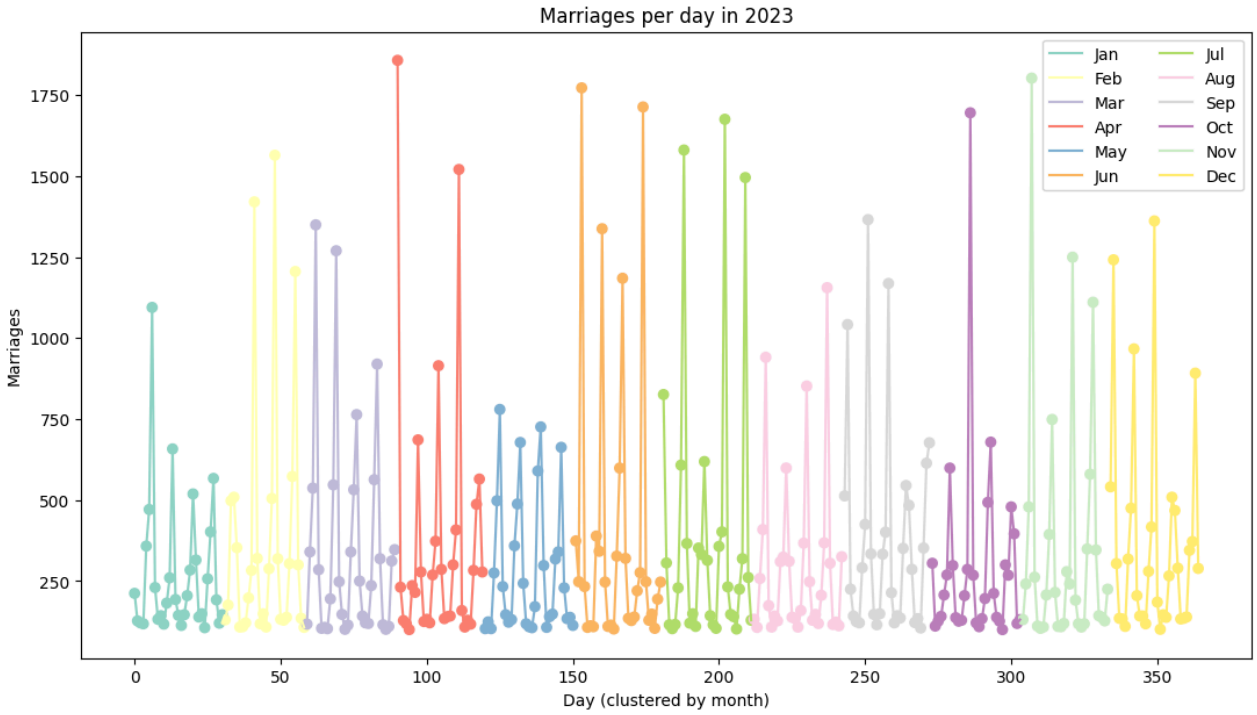
And now, combine the two heatmaps

In [158]:
```
fig, ax = plt.subplots(figsize=(15,10))
plt.imshow(heatmap7, cmap=plt.get_cmap('Spectral'))

month_cmap = ['red', 'green', 'blue', 'magenta']

for i in range(7):
    for j in range(53):
        ax.add_patch(plt.Rectangle((j/53+1e-3, i/7+1e-3), 1/53-0.0015, 1/7-0.0015, lw=1.5, ls="-",
                                   color=month_cmap[int(heatmapM[i,j])%4], fc="none", transform=ax.transAxes))


plt.xticks(range(53), np.arange(1,54,1))
plt.yticks(range(7), ['Mon','Tue','Wed','Thu','Fri','Sat','Sun'])
plt.xlabel("Week")
plt.ylabel("Day of week")
plt.show()
```



Also classical sequential approach:

In [162]:
```
plt.figure(figsize=(13,7))

M = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

for m in range(12):
    indices = []
    for i in range(len(month_of_day)):
        if month_of_day[i]==m:
            indices.append(i)
    plt.plot(x[indices], y[indices], c=plt.get_cmap("Set3")(m), label=M[m])

plt.scatter(x, y, c=plt.get_cmap("Set3")(month_of_day))

plt.xlabel("Day (clustered by month)")
plt.ylabel("Marriages")
plt.title("Marriages per day in 2023")
plt.legend(ncol=2)

plt.show()
```



Now it's quite obvious that the spikes are equidistant and they are 4 per week. It gives an insight into the general pattern, but we can't pinpoint Sunday here as being the day with the most marriages. This version, however, better reveals what happens over the weeks, which couldn't be seen in the original visualization. This way can also reveal other revelant info in case of a real dataset (like most marriages happening in summer, or less marriages near the end of the year etc.)

In [168]:
```
m_per_week = np.zeros(7)
m_count = np.zeros(7)
for i in range(len(y)):
    m_per_week[i%7] += y[i]
    m_count[i%7] += 1

avg_per_week = m_per_week / m_count

plt.title("Average marriages on each day of week in 2023")
plt.bar(['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'], avg_per_week)
plt.xlabel("Day of week")
plt.ylabel("Average marriages count")
plt.show()
```