```python
from IPython.display import Image

PATH = "https://raw.githubusercontent.com/SIIIKOR/sad2/main/projekt2/"
IMG_PATH = PATH + "/BN.png"
TASK_1_IMG_PATH = PATH + "/TASK_1.jpg"
TASK_9_IMG_PATH = PATH + "/TASK_9.jpg"
Image(url=IMG_PATH)
```

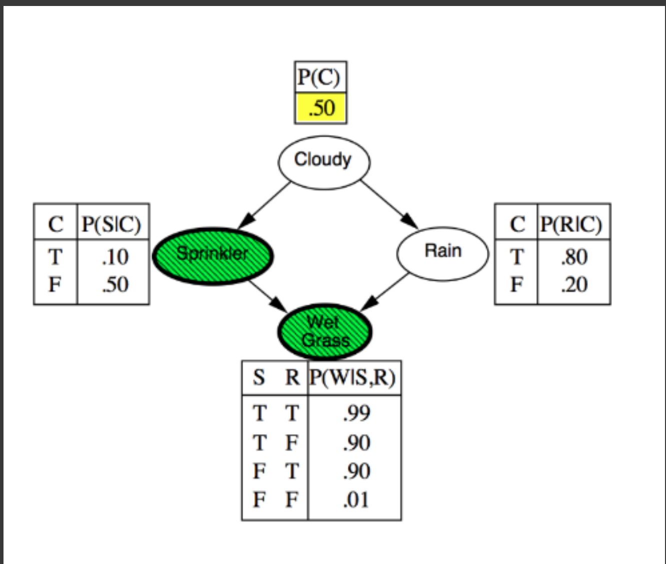


Figure 1: The *Rain Bayesian network*

```python
from random import choices, random
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

TASK 1

```python
Image(url=TASK_1_IMG_PATH, width=3200//3, height=1809//3)
```

$$P(C=T \mid R=T, S=T, W=T) = P(C=T \mid R=T, S=T) = \dots$$

$$P(C=T \mid R=F, S=T, W=T) = P(C=T \mid R=F, S=T) = \frac{P(C=T)\, P(R=F \mid C=T)\, P(S= \dots}{P(C=T)\, P(R=F \mid C=T)\, P(S=T \mid C=T \dots + P(C=F)\, P(R=F \mid C=F)\, P(S=T \mid C= \dots}$$

$$P(R=T \mid C=T, S=T, W=T) = \frac{P(R=T \mid C=T)\, P(W=T \mid R=T, S=T)}{P(R=T \mid C=T) \cdot P(W=T \mid R=T, S=T) + P(R=F \mid C=T) \cdot P(W=T \mid R=F, S=T)}$$

$$P(R=T \mid C=F, S=T, W=T) = \frac{P(R=T \mid C=F)\, P(W=T \mid R=T, S=T)}{P(R=T \mid C=F)\, P(W=T \mid R=T, S=T) + P(R=F \mid C=F)\, P(W=T \mid R=F, S=T)}$$

```python
P_ONE = 0.444               # P(C=T | R=T, S=T, W=T)
P_TWO = 0.048               # P(C=T | R=F, S=T, W=T)
P_THREE = 0.815             # P(R=T | C=T, S=T, W=T)
P_FOUR = 0.216              # P(R=T | C=F, S=T, W=T)
```

```

CLOUDY_DIST = (
    #C=FALSE    C=TRUE
    (1-P_TWO,   P_TWO),      #R=FALSE
    (1-P_ONE,   P_ONE)       #R=TRUE
)
RAIN_DIST = (
    #R=FALSE    R=TRUE
    (1-P_FOUR,  P_FOUR),     #C=FALSE
    (1-P_THREE, P_THREE)     #C=TRUE
)

DISTS = (CLOUDY_DIST, RAIN_DIST)

CHOICES = (0, 1)
```

TASK 2

```
def gibbs_sample(n):
    samples = np.empty((n, 2), dtype=int)
    samples[0, 0] = 0                                                                 # cloudy
    samples[0, 1] = 1                                                                 # rain

    for i in range(1, n):                                                             # for each time step excluding
        idx_var = int(random() > 0.5)                                                 # Randomly select index of the
        idx_other_var = 1 - idx_var                                                   # Index of the other variable

        x_other_var_prev = samples[i-1, idx_other_var]                                # Value of the other variable
        x_var = choices(CHOICES, weights=DISTS[idx_var][idx_other_var])[0]           # Sample chosen variable value
        x_other_var = choices(CHOICES, weights=DISTS[idx_other_var][x_var])[0]       # Sample other variable value

        samples[i, idx_var] = x_var
        samples[i, idx_other_var] = x_other_var

    return samples
```

TASK 3

```
samples = gibbs_sample(100)

print("P(R=T|S=T,W=T) =", samples[:, 1].mean())
```

```
P(R=T|S=T,W=T) = 0.37
```

TASK 4

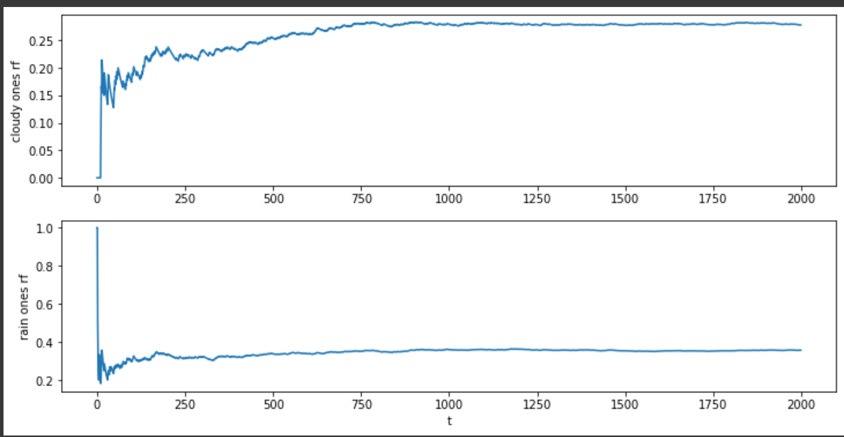
```
samples_one = gibbs_sample(50_000)
samples_two = gibbs_sample(50_000)
```

TASK 5

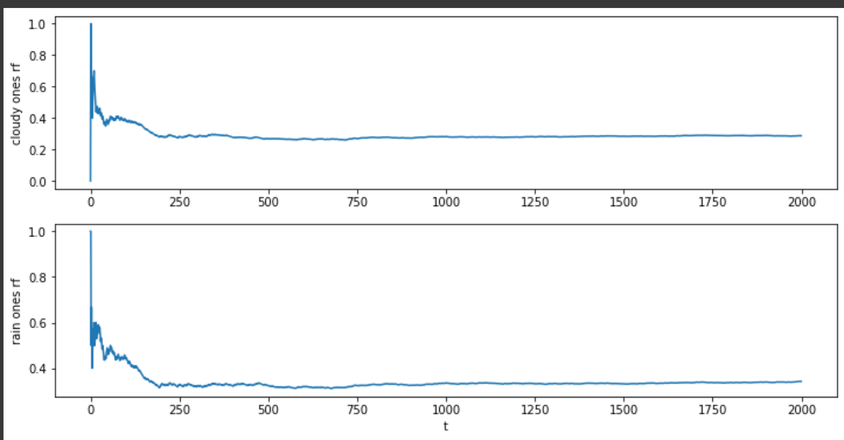
```
def get_zero_count(ones_count, idx):
    return (idx+1) - ones_count

def full_time_relative_counts(samples):
    counts = np.empty((len(samples), 2), dtype=int) #cloudy rain one counts
    counts[0, 0] = 0                                # initial values
    counts[0, 1] = 0

    for i, (x_cloudy, x_rain) in enumerate(samples):
        x_cloudy_freq = counts[i-1, 0] + x_cloudy if i != 0 else x_cloudy
        x_rain_freq = counts[i-1, 1] + x_rain if i != 0 else x_rain
        counts[i, 0] = x_cloudy_freq
        counts[i, 1] = x_rain_freq

    return counts

def counts_to_rf(samples):
```

```
    rf = np.empty((len(samples), 2), dtype=float)
    for i, val in enumerate(samples):
        rf[i] = val / i if i != 0 else val
    return rf

def plot_freqs(samples):
    fig, axs = plt.subplots(2, figsize=(12, 6))
    x = np.arange(len(samples))

    axs[0].plot(x, samples[:, 0])
    axs[0].set_ylabel("cloudy ones rf")
    axs[1].plot(x, samples[:, 1])
    axs[1].set_xlabel('t')
    axs[1].set_ylabel("rain ones rf")

    fig.show()

samples_one_rf = counts_to_rf(full_time_relative_counts(samples_one))
samples_two_rf = counts_to_rf(full_time_relative_counts(samples_two))
```

```
plot_freqs(samples_one_rf[:2000])
```




```
plot_freqs(samples_two_rf[:2000])
```



burn in time (takie najmniejsze t, że wykres zdaje się zbiegać) dla niektórych generacji może być tak małe jak t=100 a dla innych może wymagać nawet t=400.

Mimo tego, że może wymagać tylko 400, czasem zdarza się tak, że po pewnym czasie wartości zaczynają w miare liniowo dośc powoli spadać lub wzrastać. W związku z tym decyduje się wziąć większe t, takie, po którym nie zauważe tych wzrostów lub spadków.

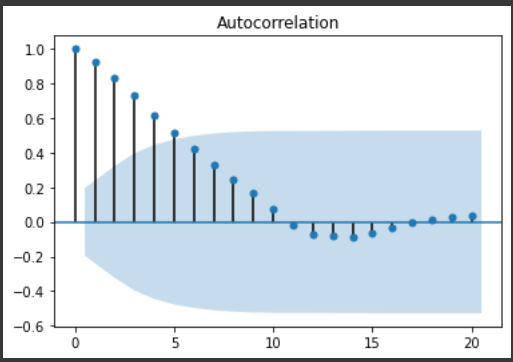
W związku z tym wydaje mi się, że w miare bezpieczny burn in time to t = 2000

TASK 6

Zredukowałem ilość sampli do pokazania na wykresie, inaczej z wykresu nie jestem w stanie nic odczytać.

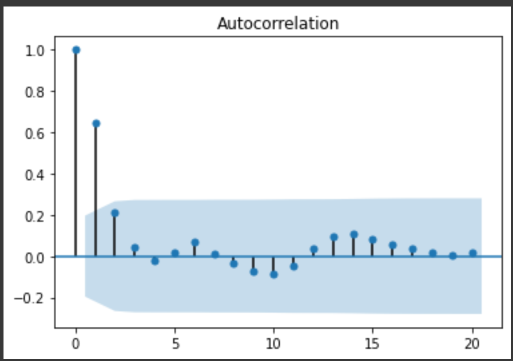
Cloudy

```
sm.graphics.tsa.plot_acf(samples_one_rf[:100, 0])
plt.show()
```



Rain

```
sm.graphics.tsa.plot_acf(samples_one_rf[:100, 1])
plt.show()
```



Możemy zauważyć, że korelacja na początku jest dodatnia. W przypadku cloudy spada do zera po interval=10 natomiast w przyadku rain jest zerowa tak szybko jak już po interval=4 lub interval=5. Dalej dla większych wartości oscyluje wokół zera.

Wybrałbym interval=10 tak aby dla obu było ok.

TASK 7

```
def gibbs_sample_optimised(n, burn_in=2000, interval=10):
    samples = np.empty((burn_in + interval*n, 2), dtype=int)                # we will have to make interva
    samples[0, 0] = 0                                                        # cloudy
    samples[0, 1] = 1                                                        # rain

    for i in range(1, burn_in + n*interval):                                 # for each time step excluding
        idx_var = int(random() > 0.5)                                        # Randomly select index of the
        idx_other_var = 1 - idx_var                                          # Index of the other variable

        x_other_var_prev = samples[i-1, idx_other_var]                       # Value of the other variable
        x_var = choices(CHOICES, weights=DISTS[idx_var][idx_other_var])[0]   # Sample chosen variable value
        x_other_var = choices(CHOICES, weights=DISTS[idx_other_var][x_var])[0]  # Sample other variable value

        samples[i, idx_var] = x_var
        samples[i, idx_other_var] = x_other_var

    return samples[burn_in::interval]                                        # take only interval'th elemen
```

TASK 8

```
samples = gibbs_sample_optimised(100)

print("P(R=T|S=T,W=T) =", samples[:, 1].mean())
```

```
P(R=T|S=T,W=T) = 0.37
```

TEST

```python
var = 1
t = 100
T = 1000
base_version_samples = np.array([gibbs_sample(t)[:, var].mean() for _ in range(T)])
improved_version_samples = np.array([gibbs_sample_optimised(t)[:, var].mean() for _ in range(T)])
```

```python
print("base gibbs mean: ", base_version_samples.mean())
print("improved gibbs mean: ", improved_version_samples.mean())
print()
print("base gibbs variance: ", base_version_samples.var())
print("improved gibbs variance: ", improved_version_samples.var())
```

```
base gibbs mean:  0.35534
improved gibbs mean:  0.34837

base gibbs variance:  0.0022648843999999997
improved gibbs variance:  0.0022212431
```

W obu przypadkach mean jest prawie taki sam (powinien być identyczny). Natomiast wersja z brun in i spłaszczaniem ma mniejszą wariancje, czyli ogólnie jej wyniki są bliżej rzeczywistej wartości niż w przypadku wersji bazowej.

Pokazuje to, że mój burn in i interval poprawiają jakość samplowania, ale nie jakoś szczególnie bardzo.

TASK 9

```python
Image(url=TASK_9_IMG_PATH, width=3200//3, height=1809//3)
```

$$P(R=T|S=T,W=T) = \frac{P(R=T,S=T,W=T)}{P(S=T,W=T)} \approx \frac{0.17}{0.5}$$

$$P(R=T,S=T,W=T) = P(R=T|C=T)\,P(S=T|C=T)\,P($$
$$P(R=T|C=F)\,P(S=T|C=F)\,P(W=$$

$$\approx 0{,}178$$

$$P(S=T,W=T) = P(R=F|C=T)\,P(S=T|C=T)\,P(W=$$
$$P(R=F|C=F)\,P(S=T|C=F)\,P(W=$$
$$P(R=T,S=T,W=T)$$

$$\approx 0.556$$

Otrzymywane przeze mnie wyniki nie są identyczne, ale są bardzo zbliżone do 0.32

BONUS

```python
def gelman_rubin_test(x):
    n = len(x)
    W = x.var(axis=1).mean()
    B = x.mean(axis=1).var()
    sigma_hat_squared = (n-1)*W/n + B/n
    return sigma_hat_squared
```

```
x = np.array([gibbs_sample_optimised(100)[:, 1] for _ in range(25)])
sigma_hat_squared = gelman_rubin_test(x)

for i, chain in enumerate(x):
    print(f"{i}'th chain variance: {round(chain.var(), 3)}")

print("varaince across all chains", round(sigma_hat_squared, 3))
```

```
0'th chain variance: 0.218
1'th chain variance: 0.218
2'th chain variance: 0.206
3'th chain variance: 0.21
4'th chain variance: 0.24
5'th chain variance: 0.218
6'th chain variance: 0.192
7'th chain variance: 0.24
8'th chain variance: 0.218
9'th chain variance: 0.24
10'th chain variance: 0.227
11'th chain variance: 0.206
12'th chain variance: 0.236
13'th chain variance: 0.214
14'th chain variance: 0.24
15'th chain variance: 0.214
16'th chain variance: 0.21
17'th chain variance: 0.236
18'th chain variance: 0.218
19'th chain variance: 0.218
20'th chain variance: 0.242
21'th chain variance: 0.228
22'th chain variance: 0.23
23'th chain variance: 0.227
24'th chain variance: 0.236
varaince across all chains 0.214
```

Wyniki nie są identyczne, ale są dość zbliżone.