Backtesting of 99%/10-day Value at Risk

To determine the VaR breaches, we first need to calculate the following:

- the daily returns,
- the rolling 21-day standard deviations of the daily returns,
- the forward-realized 10-day returns,
- the 10-day-scaled standard deviations,
- the VaR.

Using `pandas`, let us import the `.csv` file and create `dataframe` with this information.

```
df = pd.read_csv('data.csv')
df["returns"] = np.log(df["SP500"]/df["SP500"].shift(1))
df["fwd 10 day returns"] = np.log(df["SP500"].shift(-10)/df["SP500"])
df["deviations"] = df["returns"].rolling(21).std()
df["10 day deviation"] = np.sqrt(10 * df["deviations"]**2)
df["VaR"] = norm.ppf(0.01)*df["10 day deviation"]
```

Note that the VaR is taken to be negative, since we calulate returns, but we want to quantify loses. We show the output of `df.dropna().head()` below.

	Date	SP500	returns	fwd 10 day returns	deviations	10 day deviation	VaR
21	21/02/2013	1502.420044	-0.006323	0.027468	0.005969	0.018874	-0.043908
22	22/02/2013	1515.599976	0.008734	0.023205	0.006243	0.019742	-0.045926
23	25/02/2013	1487.849976	-0.018479	0.044928	0.007513	0.023759	-0.055272
24	26/02/2013	1496.939941	0.006091	0.036431	0.007540	0.023842	-0.055465
25	27/02/2013	1515.989990	0.012646	0.025098	0.008028	0.025386	-0.059057

To quantify the VaR breaches, we check whether the forward realized 10-day returns are smaller than the VaR. We create a `"Breach"` column which contains a `1` if there is a breach and `0` otherwise.

```
df["Breach"] = np.where( (df["fwd 10 day returns"] < 0)
& (df["fwd 10 day returns"] < df["VaR"]), 1, 0)
```

Below are some examples of breaches obtained by printing `df[df["Breach"]==1]`.

	Date	SP500	returns	fwd 10 day returns	deviations	10 day deviation	VaR	Breach
135	05/08/2013	1707.140015	-0.001481	-0.036435	0.004627	0.014632	-0.034040	1
141	13/08/2013	1694.160034	0.002772	-0.038313	0.004093	0.012943	-0.030110	1
250	17/01/2014	1838.699951	-0.003903	-0.054088	0.006204	0.019620	-0.045643	1
251	21/01/2014	1843.800049	0.002770	-0.049246	0.005185	0.016396	-0.038144	1
252	22/01/2014	1844.859985	0.000575	-0.051851	0.005175	0.016365	-0.038071	1
379	24/07/2014	1987.979980	0.000488	-0.040241	0.005082	0.016071	-0.037388	1
417	17/09/2014	2001.569946	0.001295	-0.028074	0.003601	0.011387	-0.026491	1
418	18/09/2014	2011.359985	0.004879	-0.032948	0.003595	0.011367	-0.026444	1
425	29/09/2014	1977.800049	-0.002550	-0.053515	0.006015	0.019022	-0.044251	1
426	30/09/2014	1972.290039	-0.002790	-0.049148	0.005971	0.018883	-0.043929	1
468	28/11/2014	2067.560059	-0.002546	-0.032058	0.003463	0.010951	-0.025476	1
469	01/12/2014	2053.439941	-0.006853	-0.031568	0.003824	0.012094	-0.028134	1
470	02/12/2014	2066.550049	0.006364	-0.046457	0.003243	0.010254	-0.023855	1

We can add up the 1s to count the VaR breaches. Dividing by the length of the full dataframe yields the percentage of VaR breaches in the period of time under study.

```
var_breach_count = np.sum(df["Breach"])
var_breach_pct = var_breach_count/len(df.dropna())
print(f"VaR breach counts: {var_breach_count} \n
VaR breach percentage: {var_breach_pct}")
```

```
>> VaR breach counts: 25
VaR breach percentage: 0.020508613617719443
```

We have found:

$$\boxed{\text{VaR breaches} = 25}$$

$$\boxed{\text{Breach percentage} = 2.05\%}$$

We proceed to calculate consecutive VaR breaches, we leverage "Breach" column we created before to generate a "Consecutive" column. The element of this column are computed by multiplying the value of the "Breach" column at time t by the value at time $t+1$. This means it will take the value 1 if and only if there are consecutive breaches.

```
df["Consecutive"] = df["Breach"]*df["Breach"].shift(-1)
```

Indeed, if we print the table again, we observe that the behaviour is as expected.

	Date	SP500	returns	fwd 10 day returns	deviations	10 day deviation	VaR	Breach	Consecutive
135	05/08/2013	1707.140015	-0.001481	-0.036435	0.004627	0.014632	-0.034040	1	0.0
141	13/08/2013	1694.160034	0.002772	-0.038313	0.004093	0.012943	-0.030110	1	0.0
250	17/01/2014	1838.699951	-0.003903	-0.054088	0.006204	0.019620	-0.045643	1	1.0
251	21/01/2014	1843.800049	0.002770	-0.049246	0.005185	0.016396	-0.038144	1	1.0
252	22/01/2014	1844.859985	0.000575	-0.051851	0.005175	0.016365	-0.038071	1	0.0
379	24/07/2014	1987.979980	0.000488	-0.040241	0.005082	0.016071	-0.037388	1	0.0
417	17/09/2014	2001.569946	0.001295	-0.028074	0.003601	0.011387	-0.026491	1	1.0
418	18/09/2014	2011.359985	0.004879	-0.032948	0.003595	0.011367	-0.026444	1	0.0
425	29/09/2014	1977.800049	-0.002550	-0.053515	0.006015	0.019022	-0.044251	1	1.0
426	30/09/2014	1972.290039	-0.002790	-0.049148	0.005971	0.018883	-0.043929	1	0.0
468	28/11/2014	2067.560059	-0.002546	-0.032058	0.003463	0.010951	-0.025476	1	1.0
469	01/12/2014	2053.439941	-0.006853	-0.031568	0.003824	0.012094	-0.028134	1	1.0
470	02/12/2014	2066.550049	0.006364	-0.046457	0.003243	0.010254	-0.023855	1	1.0
471	03/12/2014	2074.330078	0.003758	-0.030067	0.003278	0.010366	-0.024116	1	0.0

We then carry out a similar calculation as the previous one to find the count and percentage.

```
consecutive_count = np.sum(df["Consecutive"])
consecutive_pct = consecutive_count/len(df.dropna())
print(f"Consecutive breach counts: {consecutive_count}
\nVaR Consecutive breach percentage: {consecutive_pct}")
```

```
>> Consecutive breach counts: 14.0
VaR Consecutive breach percentage: 0.011484823625922888
```

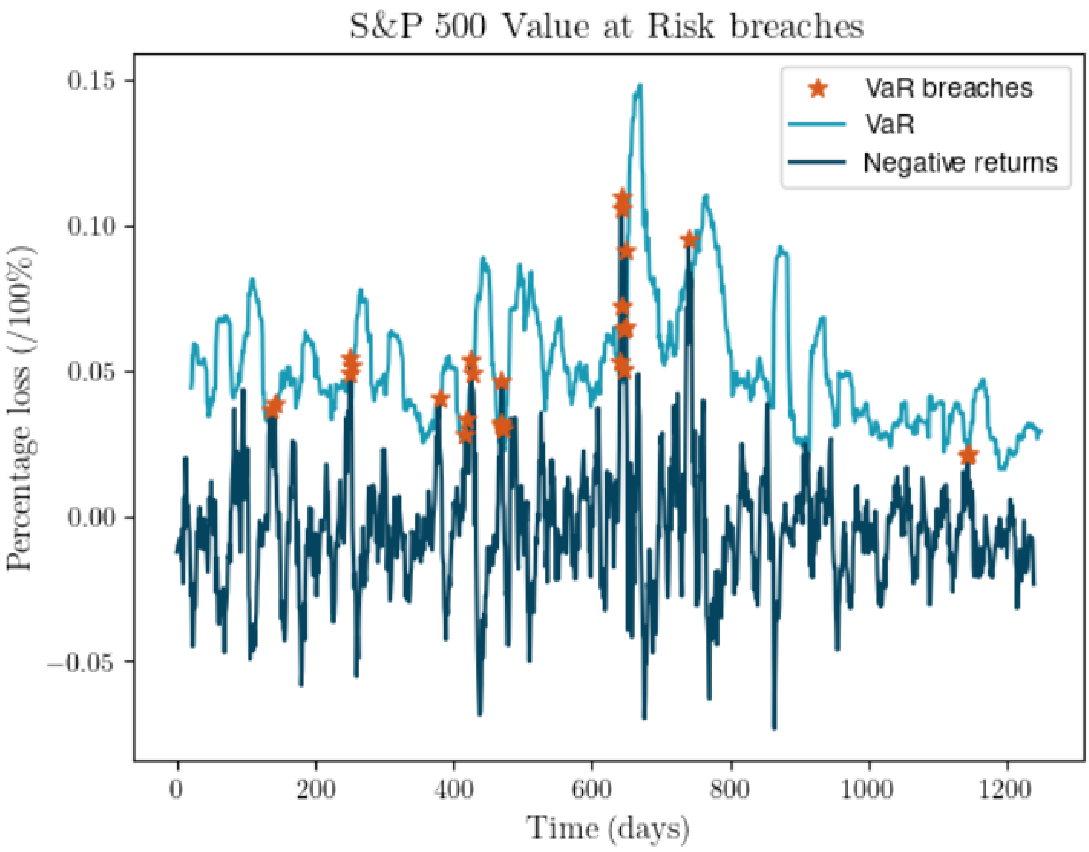
We have found:

$$\boxed{\text{Consecutive breaches} = 14}$$

$$\boxed{\text{Consecutive breach percentage} = 1.15\%}$$

The following `matplotlib.pyplot` script produces a plot that shows the percentages losses, value at risk, and breaches.

```
x = np.arange(len(df))
y_1 = -1*df["VaR"]
y_2 = -1*df["fwd 10 day returns"]
x_breaches = df[df["Breach"] ==1].index
y_breaches = df[df["Breach"]==1]["fwd 10 day returns"]*(-1)
plt.scatter(x_breaches, y_breaches, marker = '*',
zorder = 10, s=50, c = "#d8581c", label = "VaR breaches")
plt.plot(x, y_1, c = '#189ab4', label = "VaR")
plt.plot(x, y_2, c = '#05445e', label = "Negative returns")
plt.legend(loc='upper right')
plt.xlabel(r"$\textrm{Time (days)}$", fontsize = 12)
plt.ylabel(r"$\textrm{Percentage loss (/100\%)}$", fontsize = 12)
plt.title(r"$\textrm{S\&P 500 Value at Risk breaches}$", fontsize = 14)
```



Backtesting using EWMA

We create another dataframe `ewma_df` and define the returns and 10-day-forward returns in the same way as before.

```
ewma_df = pd.read_csv('data.csv')
ewma_df["returns"] = np.log(ewma_df["SP500"]/ewma_df["SP500"].shift(1))
ewma_df["fwd 10 day returns"] = np.log(ewma_df["SP500"].shift(-10)/ewma_df["SP500"])
```

For the standard deviations, however, we use EWMA with $\lambda = 0.72$. Since the first return on the table is `nan`, we fill that standard deviation with a zero and use the standard deviation `sigma` for the whole set for the *second* row in the table. The subsequent elements are filled using the EWMA formula recursively, as shown in the code snippet below.

```
sigma = ewma_df["returns"].std()
```

```
lambda_ = 0.72
deviations = np.zeros(len(ewma_df))
deviations[1] = sigma
for i in range(len(ewma_df)-2):

    deviations[i+2] = np.sqrt(lambda_*deviations[i+1]**2
    + (1-lambda_)*ewma_df["returns"][i+1]**2)

ewma_df["deviations"] = deviations
```

Having calculated the standard deviation, we can rescale them to calculate σ_{10D} and VaR_{10D}.

```
ewma_df["10 day deviation"] = np.sqrt(10 * ewma_df["deviations"]**2)
ewma_df["VaR"] = norm.ppf(0.01)*ewma_df["10 day deviation"]
```

We use the same procedure as before to find the amount of VaR breaches.

```
ewma_df["Breach"] = np.where( (ewma_df["fwd 10 day returns"] < 0) &
(ewma_df["fwd 10 day returns"] < ewma_df["VaR"]), 1, 0)
ewma_breach_count = np.sum(ewma_df["Breach"])
ewma_breach_pct = var_breach_count/len(ewma_df.dropna())
print(f"VaR breach counts: {ewma_breach_count} \nVaR breach percentage: {ewma_breach_pct}")
```

```
>> VaR breach counts: 32
VaR breach percentage: 0.020177562550443905
```

We have found

$$\boxed{\text{VaR breaches} = 32}$$

$$\boxed{\text{Breach percentage} = 2.01\%}$$

We note that even though the number of breaches found with this method is higher, the percentage is smaller. The reason is that, since we did not calculate the standard deviation using rolling windows, fewer data points needed to be discarded. Indeed, we were now able to assign a standard deviation to the first data points.

The code to compute this is almost identical as in Question 6.

```
ewma_df["Consecutive"] = ewma_df["Breach"]*ewma_df["Breach"].shift(-1)
ewma_consecutive_count = np.sum(ewma_df["Consecutive"])
ewma_consecutive_pct = consecutive_count/len(ewma_df.dropna())
print(f"Consecutive breach counts: {ewma_consecutive_count}
 \nVaR Consecutive breach percentage: {ewma_consecutive_pct}")
```

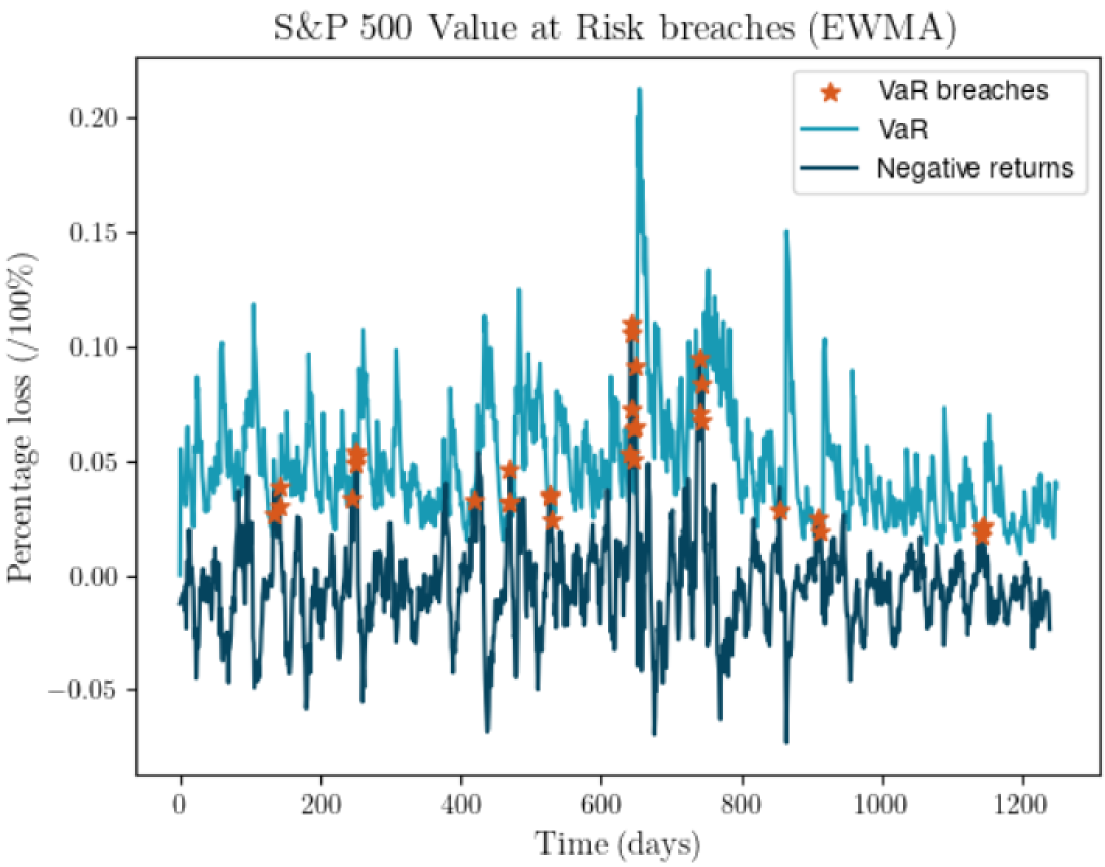
```
Consecutive breach counts: 17.0
VaR Consecutive breach percentage: 0.011299435028248588
```

We have found

$$\boxed{\text{Consecutive VaR breaches} = 17}$$

$$\boxed{\text{Consecutive breach percentage} = 1.13\%}$$

The same `pyplot` script as in Question 6 changing all instances of `df` with `ewma_df` produces the plot shown below.


S&P 500 Value at Risk breaches (EWMA)
VaR breaches
VaR
Negative returns
Percentage loss (/100%)
0.20
0.15
0.10
0.05
0.00
−0.05
0
200
400
600
800
1000
1200
Time (days)