01_Chan

May 9, 2025

[6]:
```
import numpy as np
import pandas as pd

# Set parameters
firms = 100
periods = 40
bankrupt_firms = 20
```

[7]:
```
# Model periods in which a firm would drop of the dataset:
def distribute_bankruptcies(periods, bankrupt_firms):
    if bankrupt_firms == 0:
        return np.zeros(periods, dtype=int)

    if bankrupt_firms <= periods:
        # Assign 1 bankruptcy to `bankrupt_firms` randomly chosen periods
        bankrupts = np.zeros(periods, dtype=int)
        chosen = np.random.choice(periods, bankrupt_firms, replace=False)
        bankrupts[chosen] = 1
    else:
        # Randomly divide bankrupt_firms into `periods` bins
        breaks = np.sort(np.random.choice(range(1, bankrupt_firms), periods -␣
 ↪1, replace=False))
        bankrupts = np.diff([0] + breaks.tolist() + [bankrupt_firms])

    return bankrupts
```

[9]:
```
bankrupts = distribute_bankruptcies(periods, bankrupt_firms)
# Number of Nan cells in each column
removed = np.cumsum(bankrupts)

print(bankrupts)
print(removed)
```

```
[0 0 1 1 0 1 0 1 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 1 0 1 0 0 1 1 1
 0 1 1]
[ 0  0  1  2  2  3  3  4  4  4  5  6  6  6  6  6  7  7  8  9 10 11 11 11
 11 12 13 13 13 14 14 15 15 15 16 17 18 18 19 20]
```

[45]:
```
# Generate the dataset

data = np.empty((firms, periods))
data[:] = np.nan  # initialize everything as NaN

for col in range(periods):
    n_valid = firms - removed[col]
    n_ones = int(np.rint(n_valid / 2)) #balances round up and round down
    n_zeros = n_valid - n_ones
    values = np.array([1] * n_ones + [0] * n_zeros)
    np.random.shuffle(values)
    data[:n_valid, col] = values

# Step 4: Convert to DataFrame for clarity (optional)
df = pd.DataFrame(data)

df.head()
```

[45]:

	0	1	2	3	4	5	6	7	8	9	…	30	31	32	33	\
0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	…	1.0	0.0	1.0	0.0	
1	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	…	0.0	1.0	1.0	0.0	
2	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	…	0.0	1.0	1.0	0.0	
3	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	…	0.0	0.0	1.0	0.0	
4	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	…	1.0	0.0	1.0	0.0	

	34	35	36	37	38	39
0	0.0	1.0	1.0	1.0	1.0	0.0
1	1.0	1.0	0.0	0.0	0.0	1.0
2	1.0	0.0	1.0	1.0	1.0	0.0
3	1.0	0.0	0.0	1.0	0.0	1.0
4	0.0	1.0	1.0	0.0	0.0	0.0

[5 rows x 40 columns]

[46]:
```
df.tail()
```

[46]:

	0	1	2	3	4	5	6	7	8	9	…	30	31	32	33	34	\
95	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	…	NaN	NaN	NaN	NaN	NaN	
96	0.0	0.0	0.0	1.0	0.0	0.0	1.0	NaN	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
97	1.0	1.0	0.0	1.0	1.0	NaN	NaN	NaN	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
98	0.0	1.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
99	0.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	

	35	36	37	38	39
95	NaN	NaN	NaN	NaN	NaN
96	NaN	NaN	NaN	NaN	NaN
97	NaN	NaN	NaN	NaN	NaN

```
98 NaN NaN NaN NaN NaN
99 NaN NaN NaN NaN NaN

[5 rows x 40 columns]
```

[23]:
```python
means = df.mean(skipna=True)
one_period_avg = means.mean()
print(one_period_avg)
```

```
0.4992929967292753
```

[47]:
```python
def mean_consecutive_ones(df, window_size=2):
    """
    For each window of size `window_size`, count how many firms have 1s in all␣
 ↪periods
    within that window, divided by the number of firms that are alive during␣
 ↪that window
    (i.e., firms that are not NaN in all the periods within the window).

    Parameters:
    df (pd.DataFrame): Input DataFrame.
    window_size (int): The window size for consecutive 1s.

    Returns:
    float
    """
    consecutive_counts = []

    for t in range(df.shape[1] - window_size + 1):
        # Define the window slice
        window = df.iloc[:, t:t + window_size]

        # Check if all values in the window are 1
        window_condition = (window == 1).all(axis=1)

        # Count the firms that are alive (i.e., not NaN) in all periods of the␣
 ↪window
        alive_firms = window.notna().all(axis=1).sum()

        # Count the number of firms that have 1 in all periods of the window
        count = window_condition.sum()

        # Divide by the number of firms alive during the window
        if alive_firms > 0:
            normalized_count = count / alive_firms
        else:
            normalized_count = 0  # To avoid division by zero
```

```
        consecutive_counts.append(normalized_count)

    return np.mean(consecutive_counts)
```

[49]:
```
for i in range(10):
    print(i+1,'\t', np.round(mean_consecutive_ones(df,i+1)*100,2), '%')
```

```
1       49.93 %
2       24.95 %
3       12.45 %
4       6.03 %
5       3.06 %
6       1.61 %
7       0.91 %
8       0.5 %
9       0.3 %
10      0.19 %
```

[ ]:
```

```