```python
import math

from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(1337)

X_SIZE = 5
L = 100

BETA_LIST = (0.5, 1, 1, 1, 1, 1)

N_LIST = (
    [number for number in range(50, 100, 10)] +
    [number for number in range(100, 1100, 100)]
)

def generate_df(n: int, x_size: int = X_SIZE) -> np.array:
    return np.array([
        [np.random.normal(0, 1) for _ in range(x_size)]
        for _ in range(n)
    ])


def probability(x: list[float], b_list: list[float]) -> float:
    b_list = list(b_list)

    b0 = b_list.pop(0)

    exp_content = b0
    for i in range(len(x)):
        exp_content += b_list[i]*x[i]

    return 1 / (1 + math.exp(-exp_content))


def mse(beta_head: list[float], beta: list[float]) -> float:
    beta_head = np.array(beta_head)
    beta = np.array(beta)

    return np.mean((beta_head - beta)**2)
```

1. Fit logistic model and calculate the estimators of the coefficients β = (β1, β2, β3, β4, β5). Repeat the experiment L = 100 times and compute the MSE

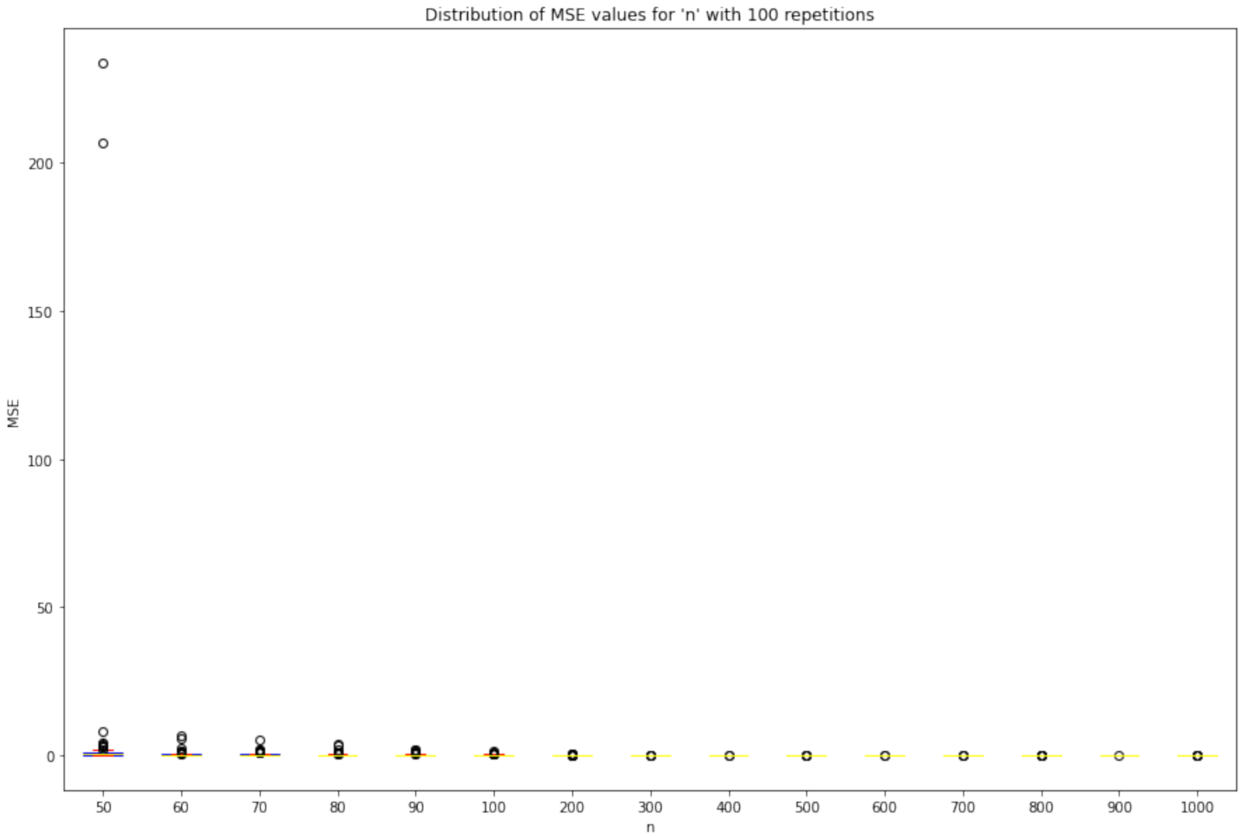
```python
results: dict[int, list[float]] = {}

for n in N_LIST:
```

```
    results[n] = []
    for _ in range(L):
        df = generate_df(n)

        p_list = [
            probability(x, BETA_LIST)
            for x in df
        ]

        y_list = [
            np.random.binomial(1, p)
            for p in p_list
        ]

        lf = LogisticRegression(penalty="l2", C=1000)
        lf.fit(df, y_list)

        beta_head = lf.coef_

        mse_value = mse(beta_head, BETA_LIST[1:])

        results[n].append(mse_value)

round_value = 3

for key in results:
    min_v = round(min(results[key]), round_value)
    avg_v = round(np.mean(results[key]), round_value)
    max_v = round(max(results[key]), round_value)

    print(f"{key:7} {min_v:7} {avg_v:7} {max_v:7}")
```

```
     50   0.014   5.128 233.857
     60   0.019   0.401   6.588
     70   0.012   0.352   5.122
     80   0.003   0.284   3.694
     90    0.01   0.206   2.006
    100   0.013   0.183    1.58
    200   0.007   0.074   0.589
    300   0.005   0.046   0.175
    400   0.004   0.026   0.083
    500   0.003   0.025   0.118
    600   0.002   0.017   0.072
    700   0.002   0.015   0.084
    800   0.001   0.013   0.064
    900   0.001   0.011   0.041
   1000   0.003   0.009   0.026
```

```
def draw(
        results: dict[int, list[float]],
        draw_first: bool = True,
```

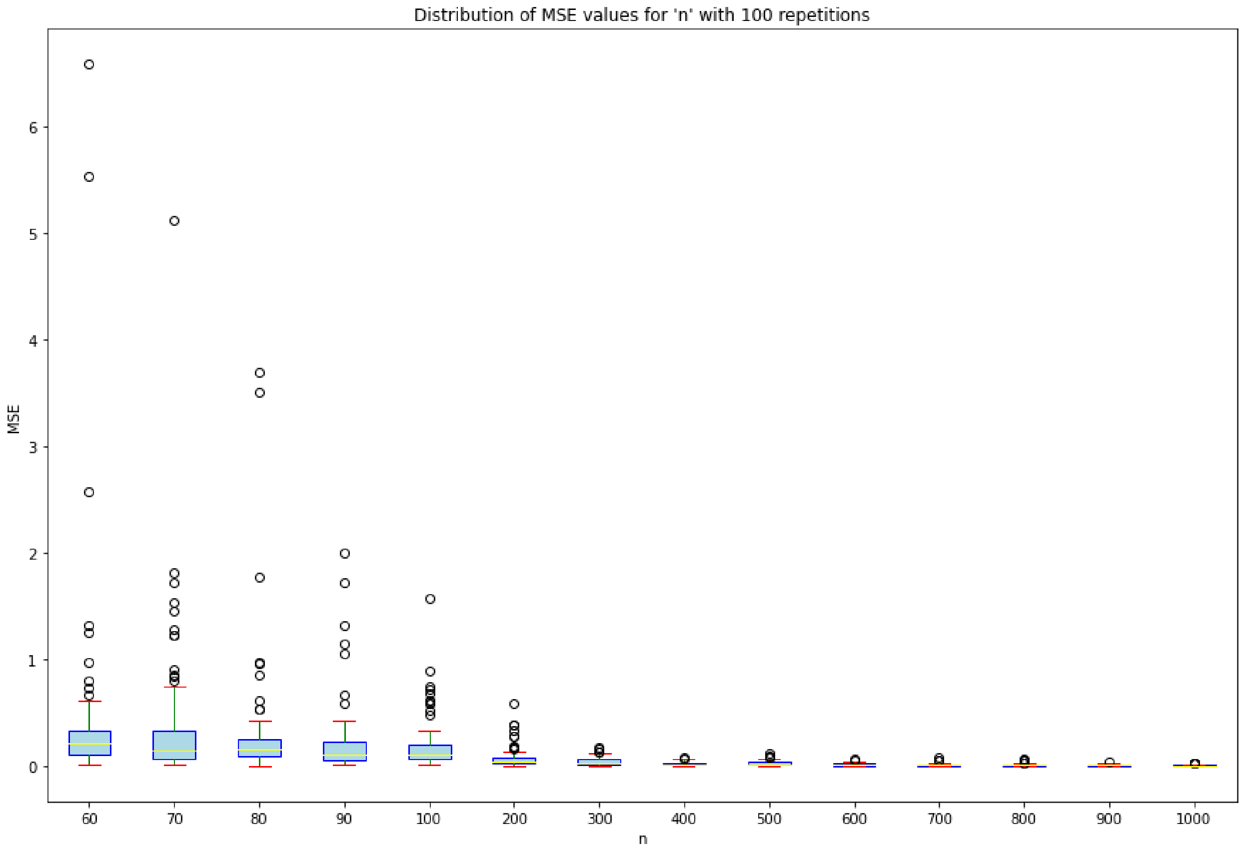
```
        max_value: float = None
        ) -> None:

    if max_value is None:
        max_value = float("inf")

    dict_to_plot = {
        key: [value for value in values if value <= max_value]
        for key, values in results.items()
    }

    if not draw_first:
        min_key = min(list(results.keys()))
        del dict_to_plot[min_key]

    labels, data = zip(*dict_to_plot.items())

    plt.figure(figsize=(15, 10))

    plt.boxplot(
        data, patch_artist=True, labels=labels,
        boxprops={"facecolor": "lightblue", "color": "blue"},
        whiskerprops={"color": "green"},
        capprops={"color": "red"},
        medianprops={"color": "yellow"}
        )

    plt.title("Distribution of MSE values for 'n' with 100
repetitions")
    plt.xlabel("n")
    plt.ylabel("MSE")

    plt.show()

draw(results)
```



As we can see, for small values (as 50) the MSE could be a really hight.

Let's remove it for greater readability of the chart.

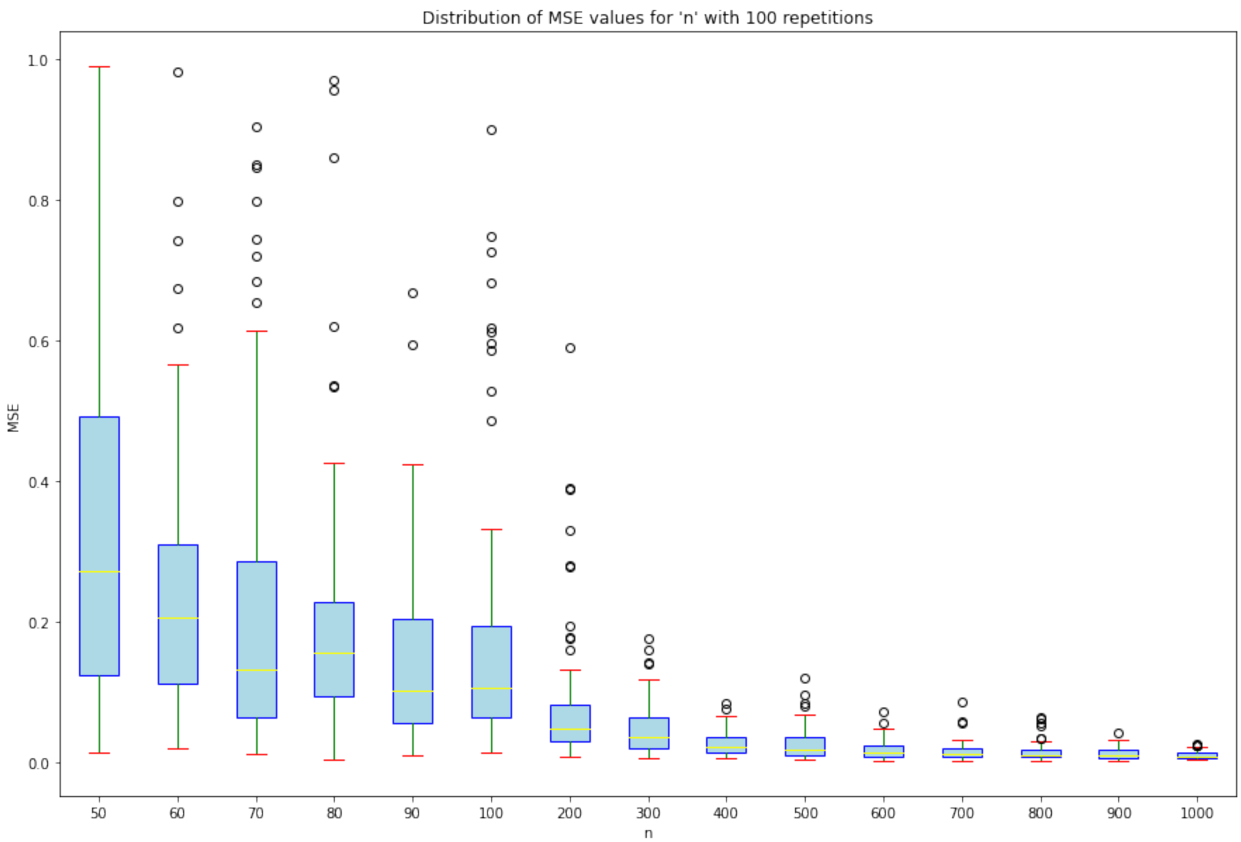
```
draw(results, draw_first=False)
```



Now it is better. However, there are a few outsiders in the data for smaller values.

Let's set the limit at Y axis, for example 1.

```
draw(results, max_value=1)
```



Now it is readable.

As we can see, the MSE is lower and lower with biggest 'n' number.

That means, that more samples means lower error. If we want to have a really good model, we need to have a lot of data for fitting

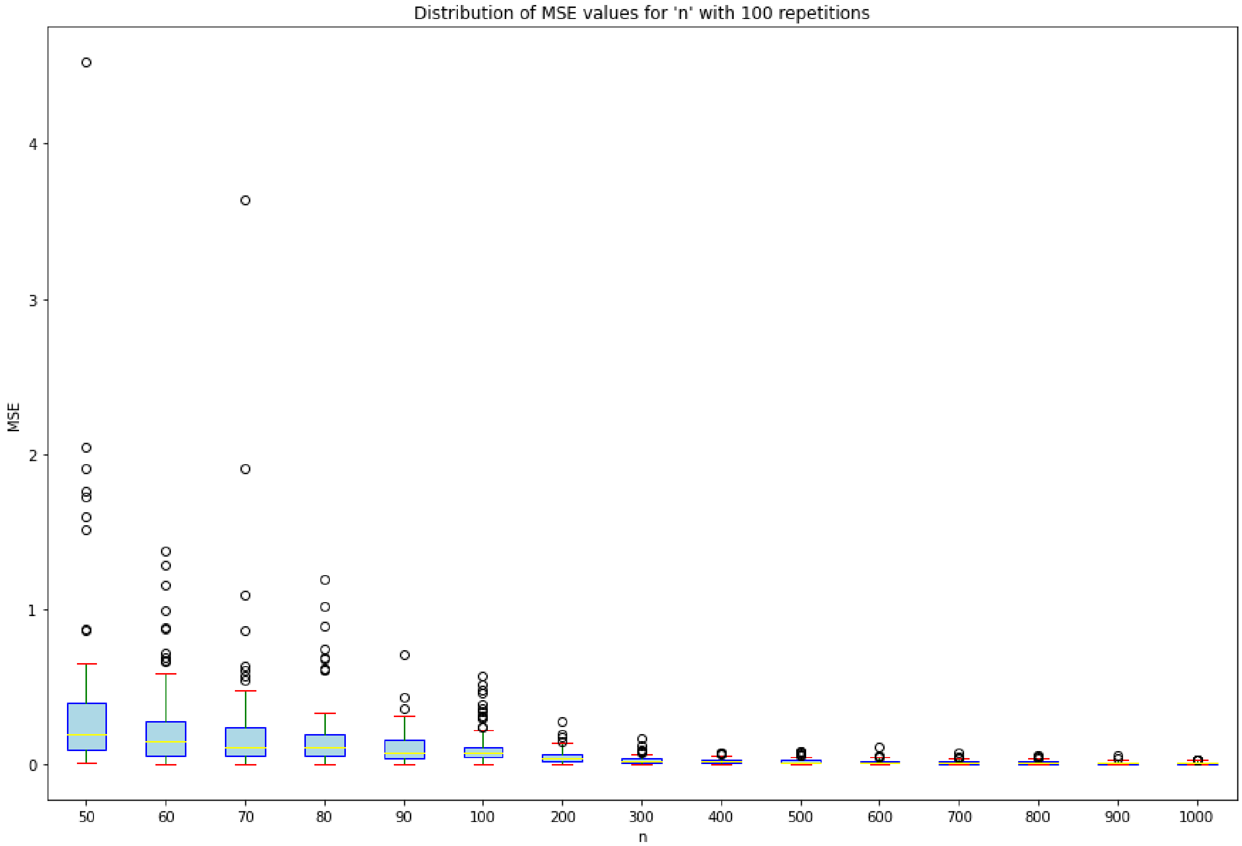
2. Using the same datasets, train the model based only on 3 variables: xi1, xi2, xi3 and draw thea nalogous curve showing how MSE for β = (β1, β2, β3) depends on n.

```
BETA_LIST = (0.5, 1, 1, 1)

new_results: dict[int, list[float]] = {}

for n in N_LIST:
    new_results[n] = []
    for _ in range(L):
        df = generate_df(n, 3)

        p_list = [
            probability(x, BETA_LIST)
            for x in df
        ]
```

```
        y_list = [
            np.random.binomial(1, p)
            for p in p_list
        ]

        lf = LogisticRegression(penalty="l2", C=1000)
        lf.fit(df, y_list)

        beta_head = lf.coef_

        mse_value = mse(beta_head, BETA_LIST[1:])

        new_results[n].append(mse_value)

round_value = 3

for key in new_results:
    min_v = round(min(new_results[key]), round_value)
    avg_v = round(np.mean(new_results[key]), round_value)
    max_v = round(max(new_results[key]), round_value)

    print(f"{key:7} {min_v:7} {avg_v:7} {max_v:7}")
```

```
     50   0.008   0.367   4.527
     60   0.002   0.238   1.378
     70   0.003   0.226   3.637
     80   0.004   0.175    1.19
     90   0.002    0.11   0.708
    100   0.005   0.109   0.567
    200   0.004   0.049   0.275
    300   0.001   0.027    0.17
    400   0.001   0.021   0.076
    500   0.001   0.019   0.086
    600     0.0   0.017   0.112
    700   0.001   0.012   0.076
    800   0.001   0.013   0.053
    900     0.0    0.01   0.054
   1000     0.0   0.009   0.032
```

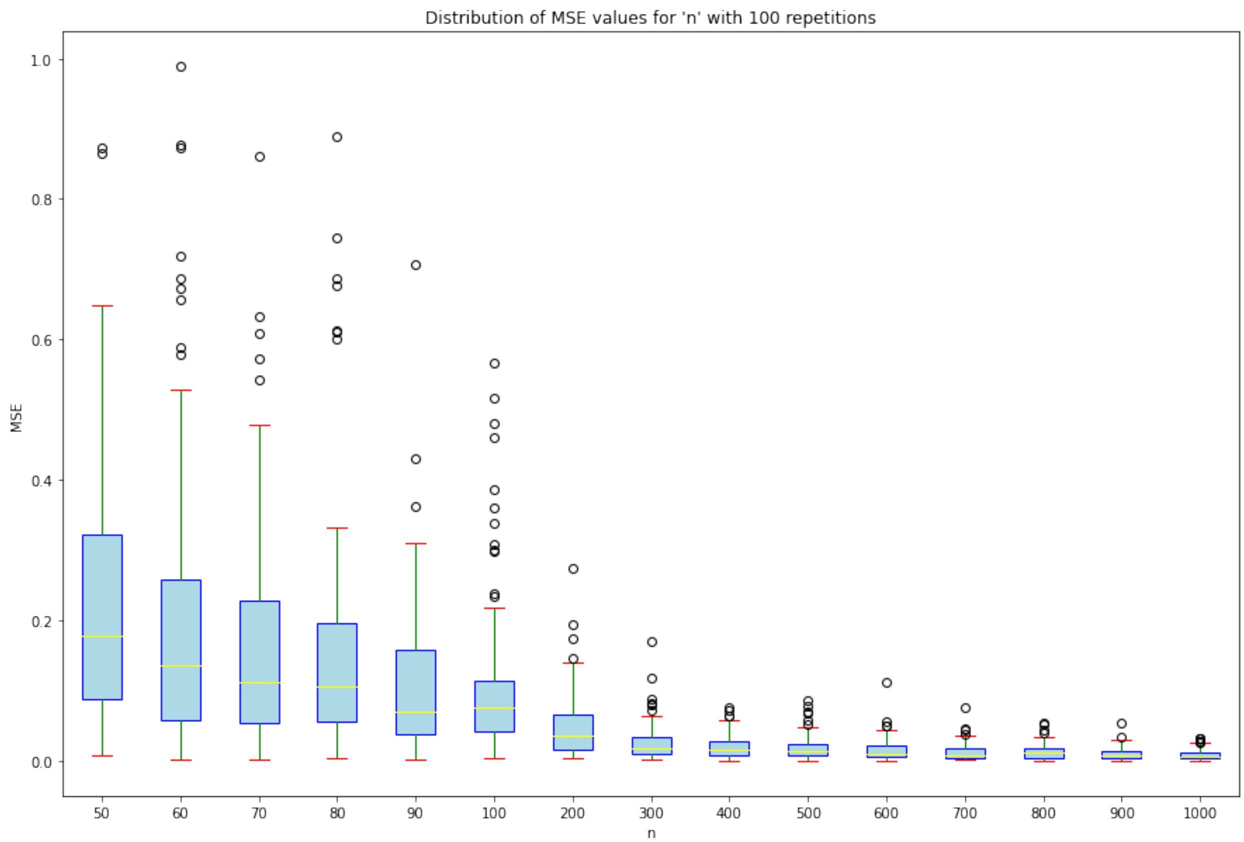
```
draw(new_results)
```



Now we have only three X variables.

There are still some outsiders, lets set Y limit to 1.

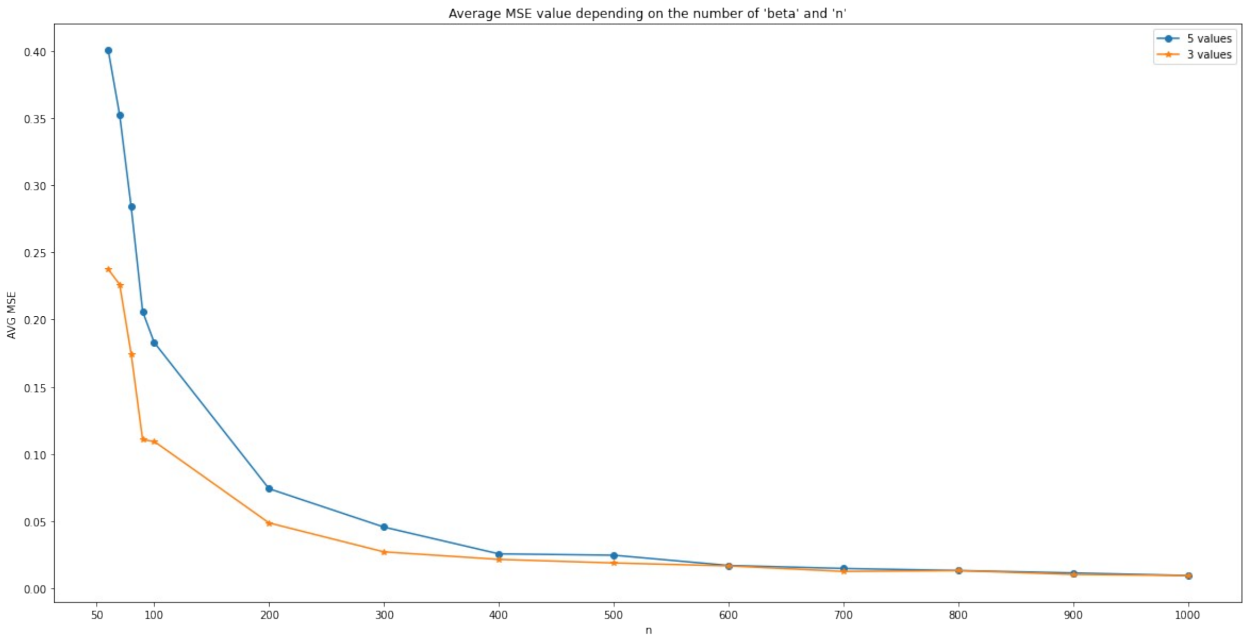
```
draw(new_results, max_value=1)
```



The plot looks really similar to the last one with five X variables.

Lets compare them correctly.

```
plt.figure(figsize=(20, 10))

averages = {label: sum(values) / len(values) for label, values in
results.items() if label != 50}
plt.plot(list(averages.keys()), list(averages.values()), marker="o",
label="5 values")

averages = {label: sum(values) / len(values) for label, values in
new_results.items() if label != 50}
plt.plot(list(averages.keys()), list(averages.values()), marker="*",
label="3 values")

plt.xticks([50] + list(range(100, 1100, 100)))
plt.title("Average MSE value depending on the number of 'beta' and
'n'")
plt.xlabel("n")
plt.ylabel("AVG MSE")
plt.legend()

plt.show()
```



You can see that at first the model learns easily for fewer variables - which sounds logical.

But with more samples loaded into the model, the number of variables becomes less important. The model trains just as well.