

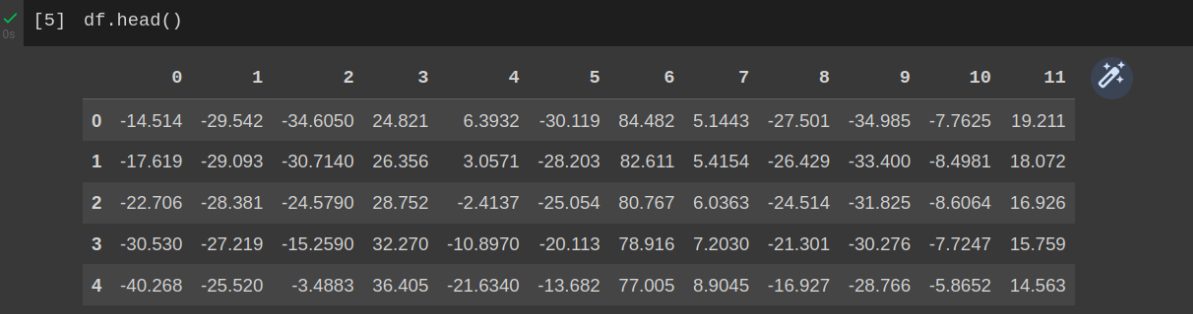
Київський національний університет імені Тараса  
Шевченка

Звіт  
до розрахункової роботи  
з дисципліни “Інтелектуальна обробка даних”

Виконав студент 4-го курсу  
факультету комп'ютерних наук та кібернетики  
спеціальності “комп'ютерні науки”  
групи ТК-41  
Некряч Владислав

Київ, 2022

Для виконання розрахункової роботи було обрано датасет A13.txt, що містить запис кардіограми людини по 12 каналах. Структура файлу: 1-й канал, 2-й канал, ..., 12-й канал (амплітуда у відносних одиницях)



The screenshot shows a Jupyter Notebook interface with a code cell containing `[5] df.head()`. Below the code, a table displays the first 5 rows of the DataFrame. The table has 12 columns, indexed 0 to 11, and 5 rows, indexed 0 to 4. The values are numerical, representing amplitude in relative units.

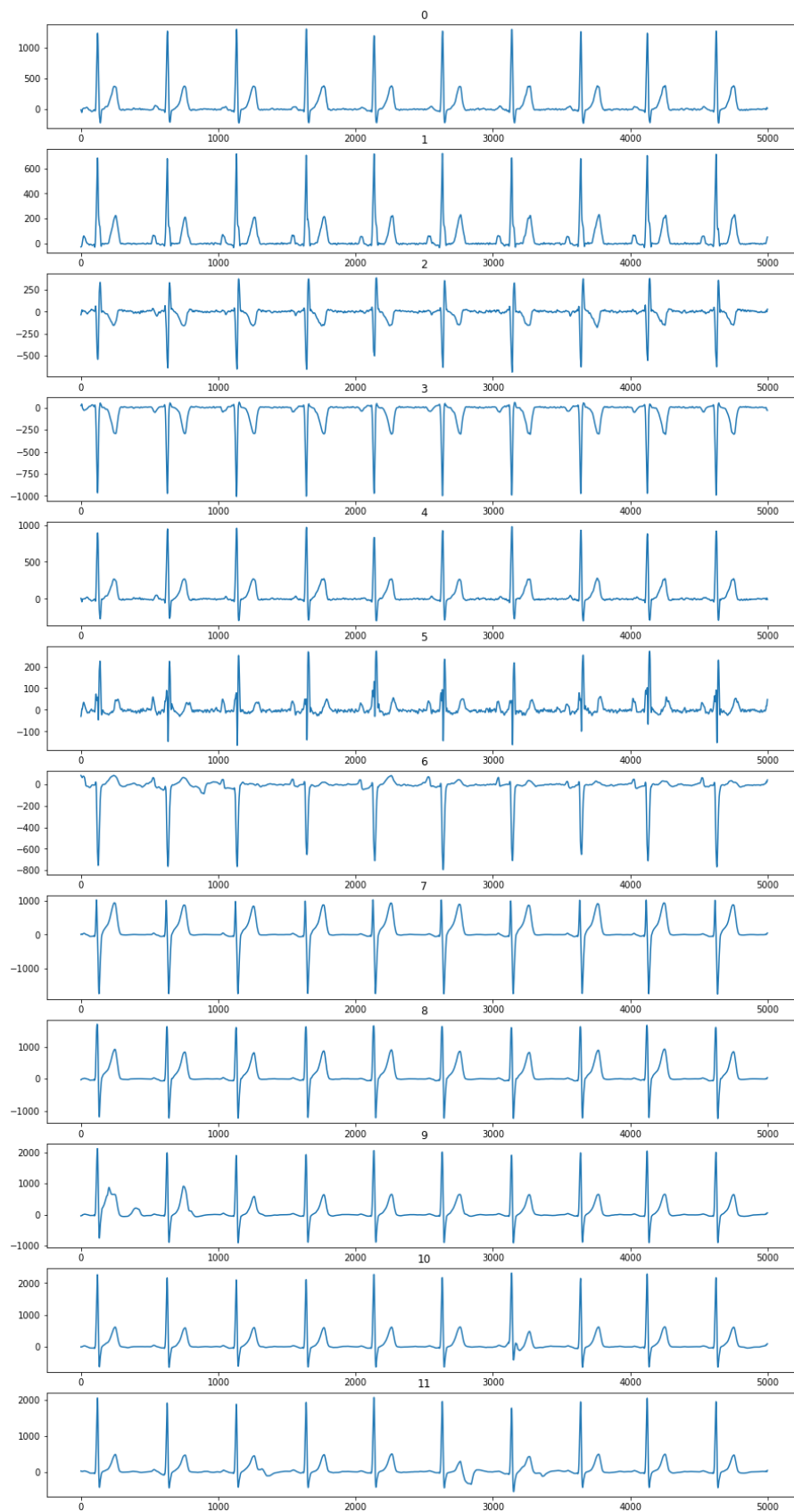
	0	1	2	3	4	5	6	7	8	9	10	11
0	-14.514	-29.542	-34.6050	24.821	6.3932	-30.119	84.482	5.1443	-27.501	-34.985	-7.7625	19.211
1	-17.619	-29.093	-30.7140	26.356	3.0571	-28.203	82.611	5.4154	-26.429	-33.400	-8.4981	18.072
2	-22.706	-28.381	-24.5790	28.752	-2.4137	-25.054	80.767	6.0363	-24.514	-31.825	-8.6064	16.926
3	-30.530	-27.219	-15.2590	32.270	-10.8970	-20.113	78.916	7.2030	-21.301	-30.276	-7.7247	15.759
4	-40.268	-25.520	-3.4883	36.405	-21.6340	-13.682	77.005	8.9045	-16.927	-28.766	-5.8652	14.563

Час запису:  $t = 10$  с.

Довжина запису:  $N = 5000$

Дискретність: 500 точок за 1 секунду

## 1. Візуалізація даних



На всіх діаграмах видно нерівномірність піків та нерівність

інтервалів серцебиття; це, скоріш за все, свідчить про патології  
серця.

## 2. Основні стат. параметри

```
df.describe()
```

	zero	one	two	three	four	five	six	seven	eight	nine	ten	eleven
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	64.476566	41.427255	-19.946345	-54.508454	43.532617	7.423669	-25.518986	69.248284	112.151698	102.830755	98.485079	72.621542
std	201.009121	105.999845	108.364117	151.174117	152.893707	39.773720	122.006665	382.291574	367.475396	348.732384	338.286075	290.575493
min	-232.100000	-35.751000	-688.790000	-1006.200000	-298.780000	-163.940000	-794.630000	-1775.700000	-1239.300000	-903.480000	-648.330000	-554.930000
25%	-10.390500	-5.918550	-12.992250	-36.002000	-8.726225	-8.478375	-12.694500	-8.769375	-9.426000	-22.502250	-11.800250	-18.912500
50%	-2.537050	-2.246200	1.419950	1.792200	-1.567650	-3.018950	-2.344600	-1.464750	-0.760165	0.868825	0.488675	3.252450
75%	33.587000	44.579250	10.244000	7.148300	20.866250	6.901000	7.862625	126.115000	126.687500	78.802250	49.213000	40.171500
max	1304.100000	721.180000	386.780000	61.918000	976.350000	272.030000	85.309000	1035.200000	1710.500000	2120.800000	2315.100000	2070.500000

Медіана:

```
df.median()
```

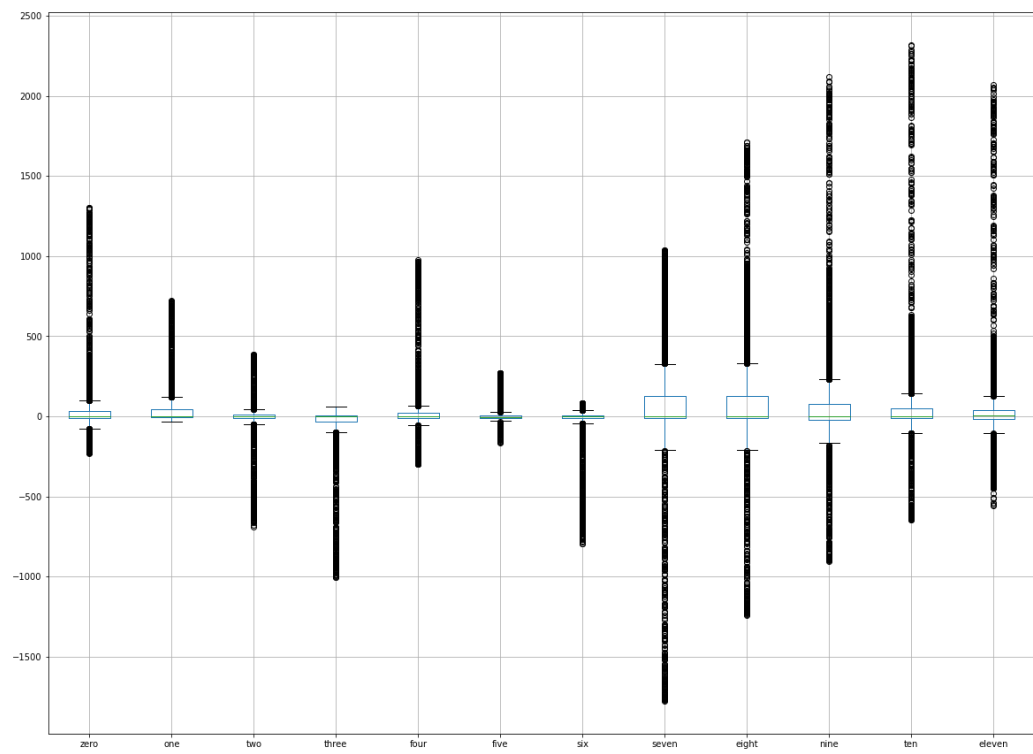
zero	-2.537950
one	-2.246200
two	1.419950
three	1.792200
four	-1.567650
five	-3.018950
six	-2.344600
seven	-1.464750
eight	-0.760165
nine	0.868825
ten	0.488675
eleven	3.252450
dtype:	float64

Дисперсія:

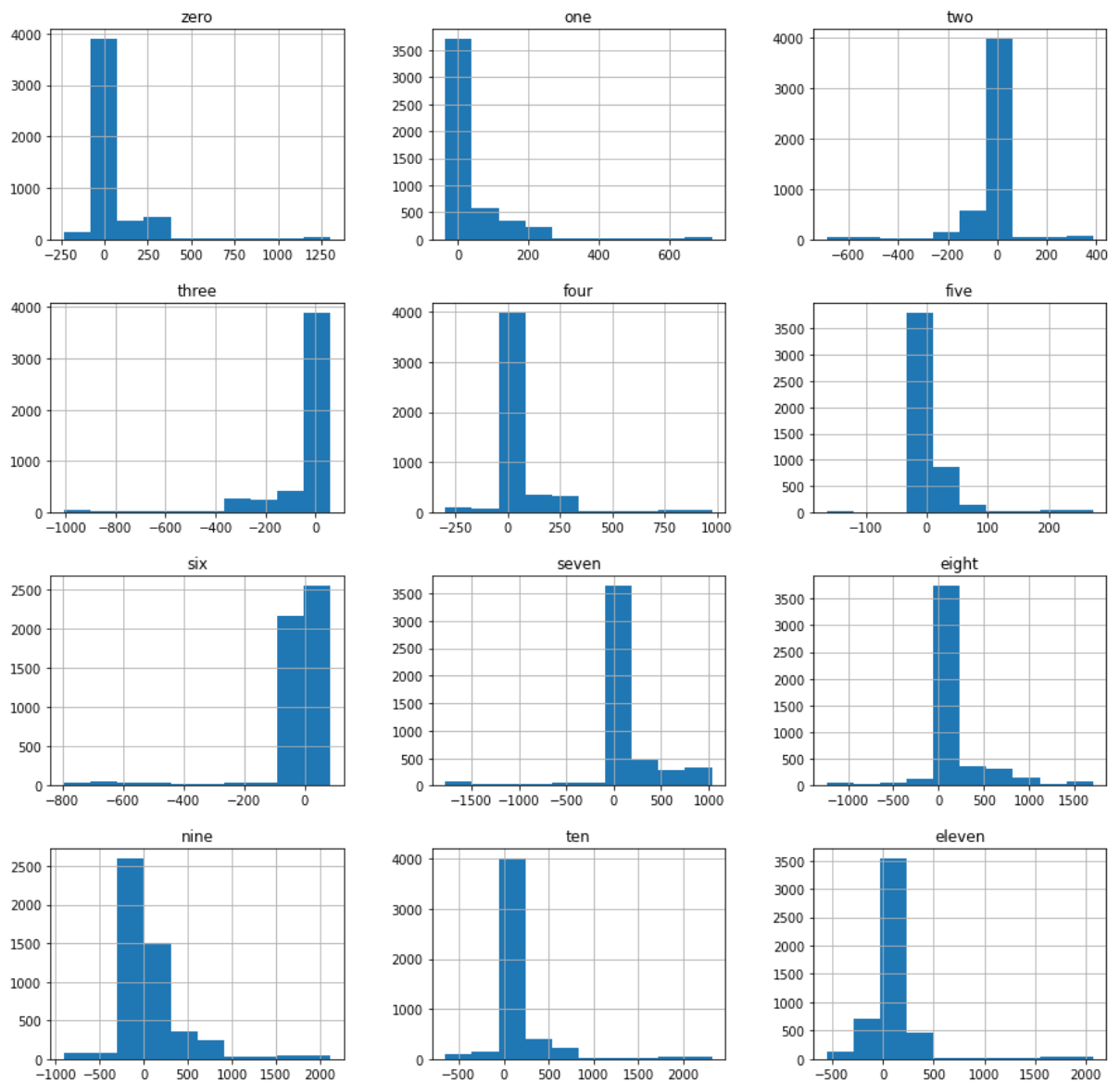
```
df.var(ddof=1)
```

zero	40404.666685
one	11235.967081
two	11742.781839
three	22853.613745
four	23376.485500
five	1581.948774
six	14885.626345
seven	146146.847630
eight	135038.166542
nine	121614.275392
ten	114437.468379
eleven	84434.116869
dtype:	float64

## Box-plot:



Гістограми для кожного каналу:



З наведених вище діаграм видно, що канал eight, ten та eleven (індексація починається з 0) мають нормальний розподіл.

# Нормалізовані дані за методом z-score:

```
[13] def z_score(df):
      df_std = df.copy()
      for column in df_std.columns:
          df_std[column] = (df_std[column] - df_std[column].mean()) / df_std[column].std()
      return df_std

[14] df_standardized = z_score(df)

      df_standardized.head()
```

	zero	one	two	three	four	five	six	seven	eight	nine	ten	eleven
0	-0.392970	-0.669522	-0.135272	0.524756	-0.242910	-0.943906	0.901598	-0.167683	-0.380033	-0.395191	-0.314076	-0.183810
1	-0.408417	-0.665286	-0.099366	0.534909	-0.264730	-0.895734	0.886263	-0.166974	-0.377116	-0.390646	-0.316251	-0.187729
2	-0.433724	-0.658569	-0.042751	0.550759	-0.300511	-0.816561	0.871149	-0.165350	-0.371904	-0.386129	-0.316571	-0.191673
3	-0.472648	-0.647607	0.043256	0.574030	-0.355996	-0.692333	0.855978	-0.162298	-0.363161	-0.381687	-0.313964	-0.195689
4	-0.521094	-0.631579	0.151877	0.601382	-0.426222	-0.530644	0.840315	-0.157848	-0.351258	-0.377357	-0.308468	-0.199805



### 3. Однофакторний аналіз

Проаналізуємо, чи датчики статистично відрізняються один від одного. Для цього використаємо модель ANOVA.

```
[15] df_standardized_long = df_standardized.melt(value_vars=list(df_standardized.columns), var_name='channel')
df_standardized_long
```

	channel	value
0	zero	-0.392970
1	zero	-0.408417
2	zero	-0.433724
3	zero	-0.472648
4	zero	-0.521094
...	...	...
59995	eleven	-0.236615
59996	eleven	-0.219366
59997	eleven	-0.192389
59998	eleven	-0.159547
59999	eleven	-0.124562

60000 rows x 2 columns

```
from statsmodels.formula.api import ols
import statsmodels.api as sm
import statistics

lm = ols('value ~ C(channel)', data=df_standardized_long).fit()
table = sm.stats.anova_lm(lm)
print(table)
```

	df	sum_sq	mean_sq	F	PR(>F)
C(channel)	11.0	1.500373e-28	1.363975e-29	1.363975e-29	1.0
Residual	59988.0	5.998800e+04	1.000000e+00	NaN	NaN

З наведеної таблиці робимо висновок, що між каналами немає статистичної різниці (F-statistic  $\sim 1.0$ ).

#### 4. Двофакторний аналіз

Перевіримо, чи комбінація часу та каналу впливає на кардіограму.

Розділяємо дані на відрізки по 2 секунди, і кожен відрізок помічаємо відповідним числом від 1 до 5.

```
[17] sample = np.concatenate((np.repeat(1, 1000), np.repeat(2, 1000), np.repeat(3, 1000), np.repeat(4, 1000), np.repeat(5, 1000)), axis=None)
df_standardized['sample'] = sample
```

```
[18] df_standardized
```

	zero	one	two	three	four	five	six	seven	eight	nine	ten	eleven	sample
0	-0.392970	-0.669522	-0.135272	0.524756	-0.242910	-0.943906	0.901598	-0.167683	-0.380033	-0.395191	-0.314076	-0.183810	1
1	-0.408417	-0.665286	-0.099366	0.534909	-0.264730	-0.895734	0.886263	-0.166974	-0.377116	-0.390646	-0.316251	-0.187729	1
2	-0.433724	-0.658569	-0.042751	0.550759	-0.300511	-0.816561	0.871149	-0.165350	-0.371904	-0.386129	-0.316571	-0.191673	1
3	-0.472648	-0.647607	0.043256	0.574030	-0.355996	-0.692333	0.855978	-0.162298	-0.363161	-0.381687	-0.313964	-0.195689	1
4	-0.521094	-0.631579	0.151877	0.601382	-0.426222	-0.530644	0.840315	-0.157848	-0.351258	-0.377357	-0.308468	-0.199805	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	-0.204884	-0.102352	0.194014	0.189116	-0.217711	0.381542	0.403896	-0.129993	-0.287623	-0.182039	-0.091098	-0.236615	5
4996	-0.205735	-0.051361	0.244703	0.172129	-0.236624	0.519950	0.433386	-0.118353	-0.271310	-0.170130	-0.076814	-0.219366	5
4997	-0.217620	-0.004002	0.312311	0.163860	-0.267400	0.681539	0.472851	-0.102653	-0.249747	-0.154602	-0.059060	-0.192389	5
4998	-0.235425	0.038215	0.385869	0.161320	-0.302900	0.849388	0.517668	-0.083822	-0.224948	-0.137185	-0.039508	-0.159547	5
4999	-0.254663	0.077196	0.458928	0.160798	-0.338418	1.015428	0.564281	-0.063311	-0.198872	-0.119200	-0.019407	-0.124562	5

Перегрупуємо дані:

```
channels_names = list(df_standardized.columns)
df_standardized_long = df_standardized.melt(id_vars=['sample'], value_vars=channels_names, var_name='channel')
df_standardized_long
```

sample	channel	value
0	zero	-0.392970
1	zero	-0.408417
2	zero	-0.433724
3	zero	-0.472648
4	zero	-0.521094
...	...	...
59995	eleven	-0.236615
59996	eleven	-0.219366
59997	eleven	-0.192389
59998	eleven	-0.159547
59999	eleven	-0.124562

Виконуємо двофакторний аналіз:

✓  
3s

▶

```
lm = ols('value ~ C(channel)*C(sample)', data=df_standardized_long).fit()
table = sm.stats.anova_lm(lm)
table
```

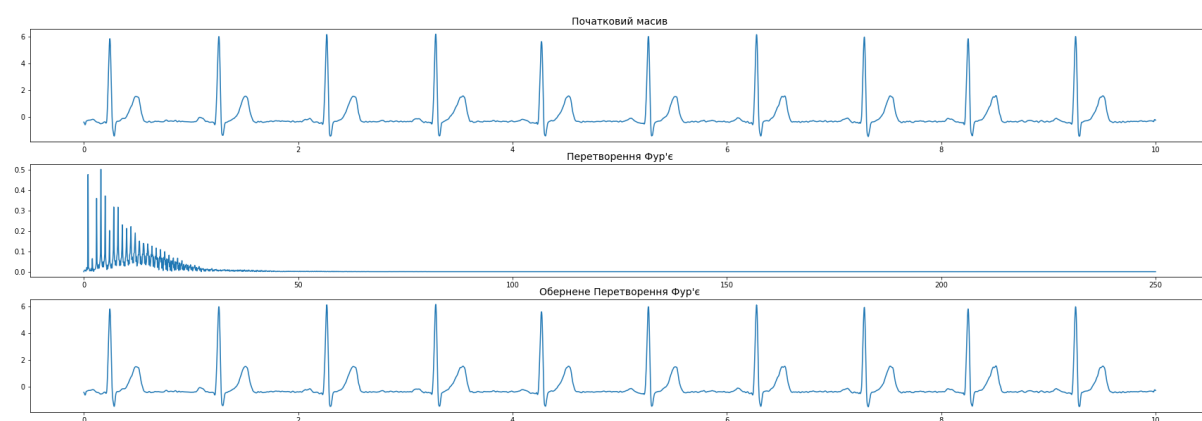
	df	sum_sq	mean_sq	F	PR(>F)
C(channel)	11.0	5.268409e-28	4.789463e-29	4.789357e-29	1.000000
C(sample)	4.0	6.554184e+00	1.638546e+00	1.638510e+00	0.161433
C(channel):C(sample)	44.0	4.012181e+01	9.118594e-01	9.118393e-01	0.638548
Residual	59940.0	5.994132e+04	1.000022e+00	NaN	NaN

Бачимо, що жодна з груп не має суттєвого впливу на кардіограму при  $p\text{-value} = 0.05$ . Тобто, не дивлячись на нерівності які були помітні при візуалізації даних, серцебиття є стабільним.

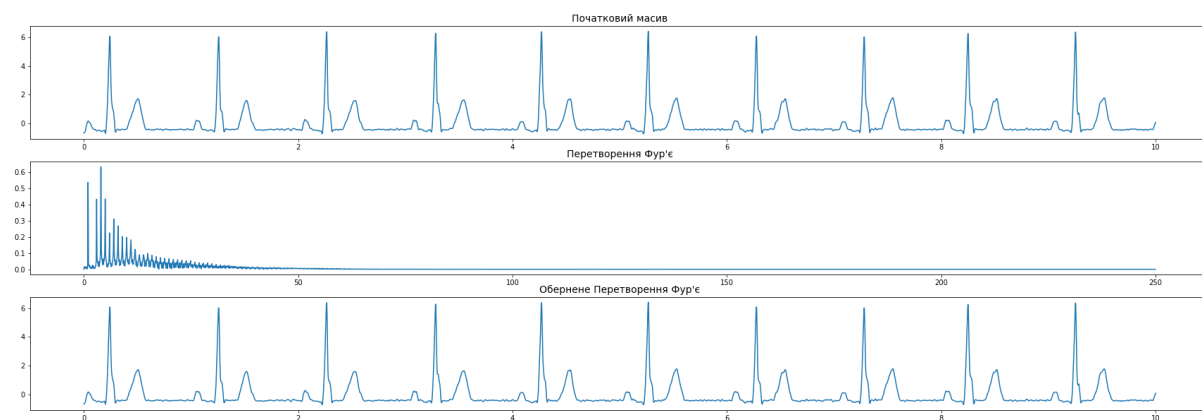
## 5. Перетворення Фур'є

```
✓ [21] N = 5000  
0s t = 10  
Fs = 500  
t_step = 1 / Fs  
f_step = Fs / N
```

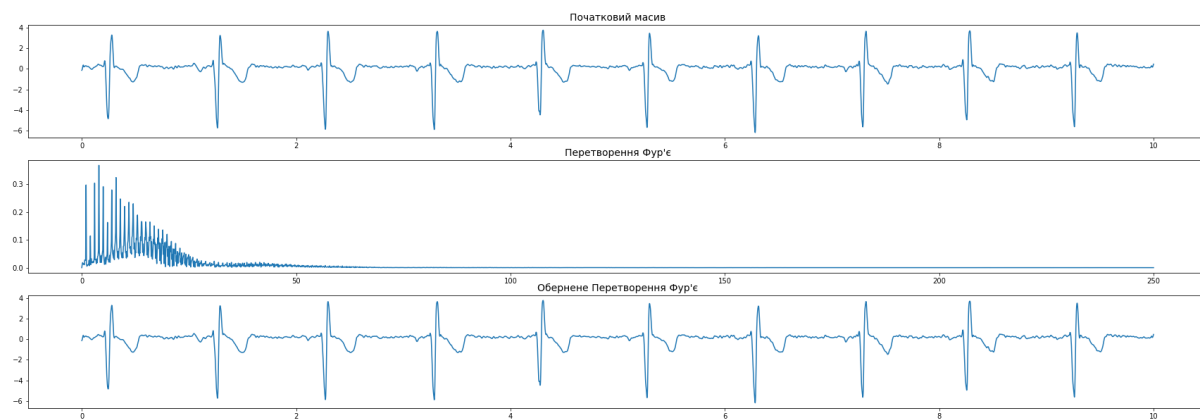
1:



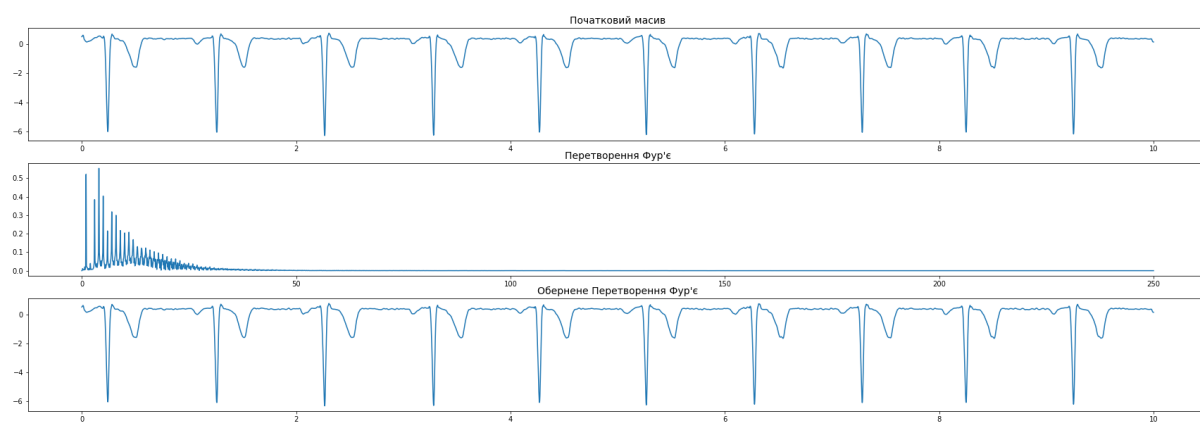
2:



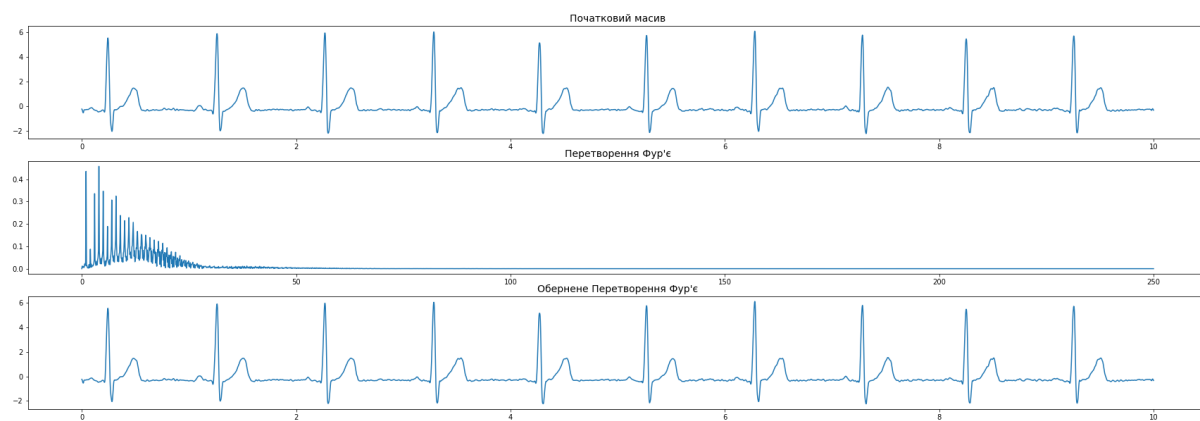
3:



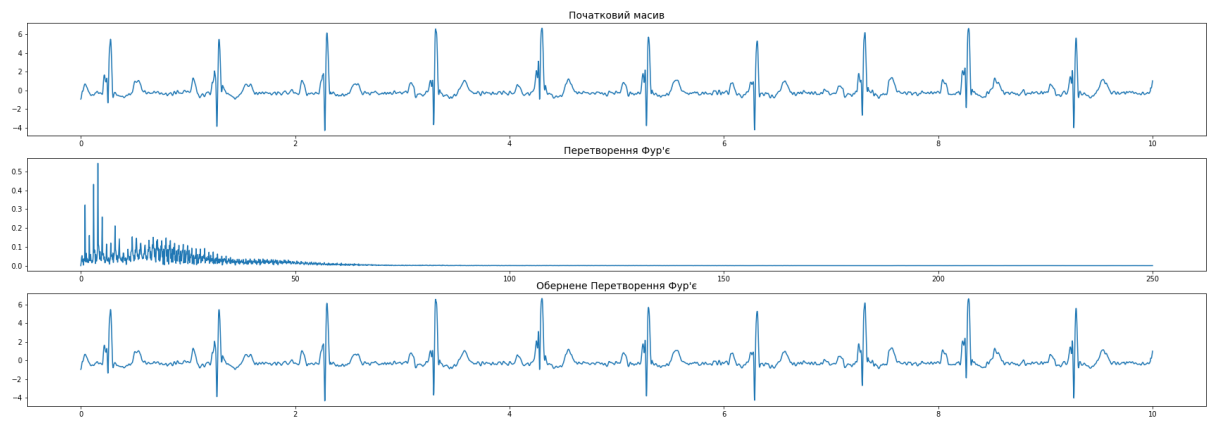
4:



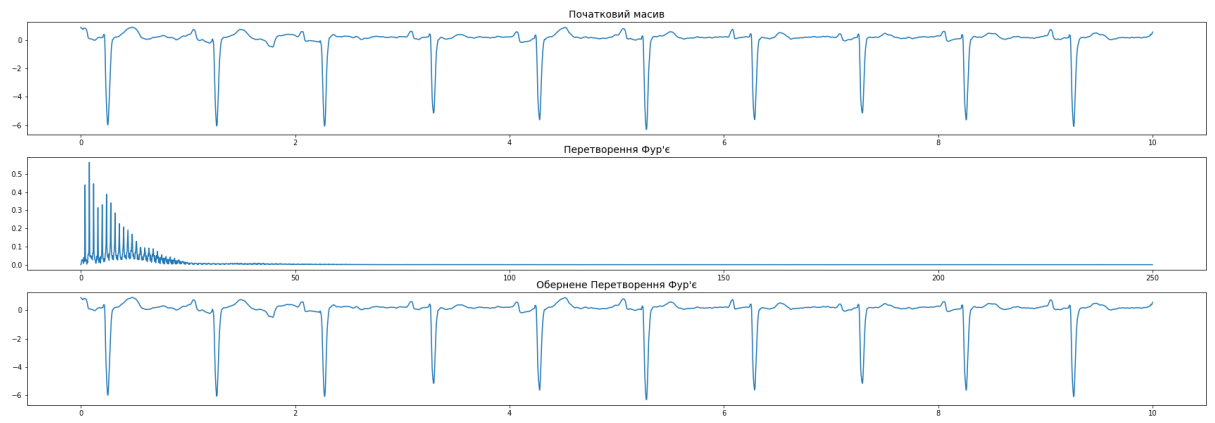
5:



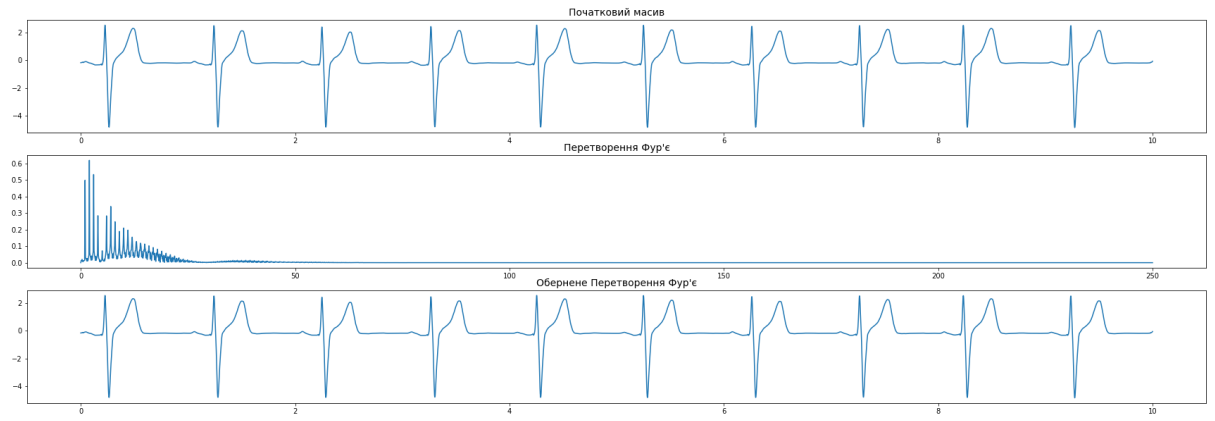
6:



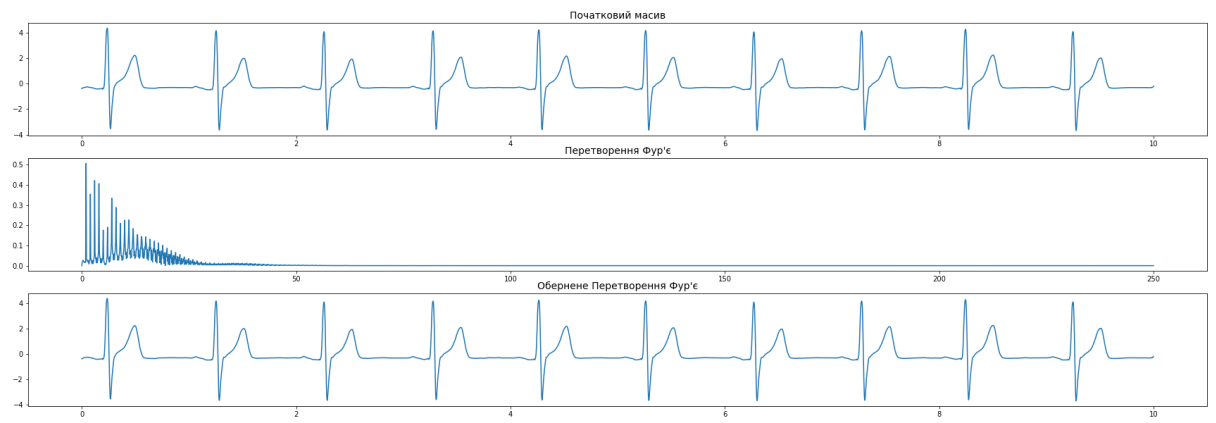
7:



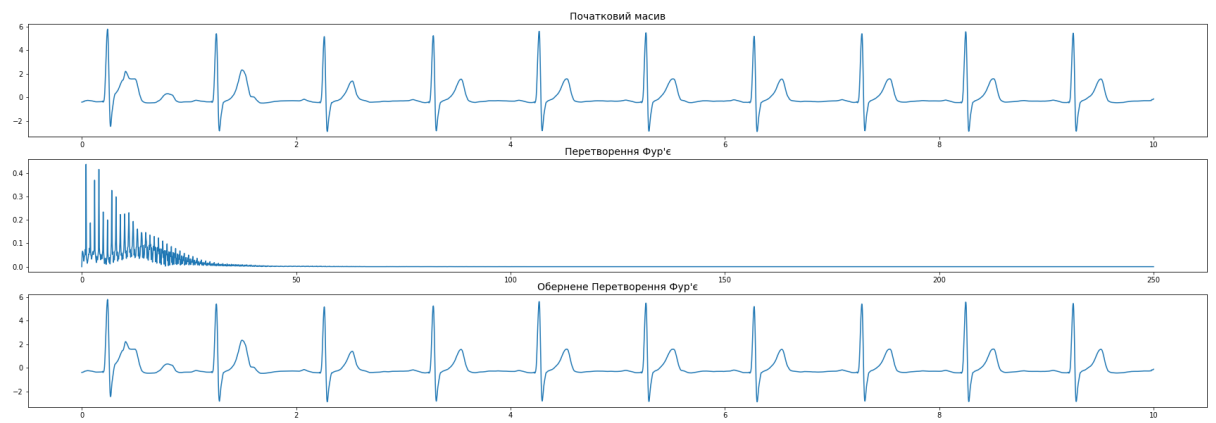
8:



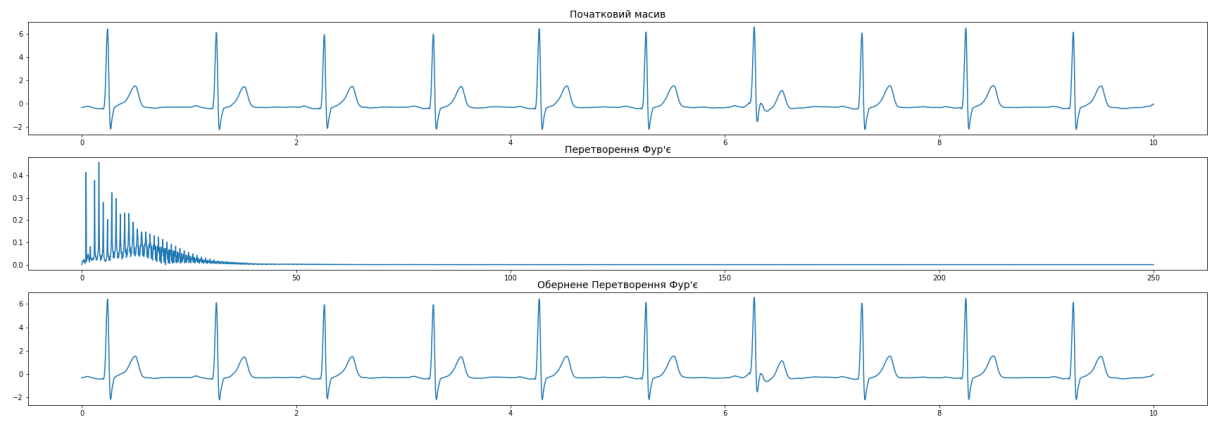
9:



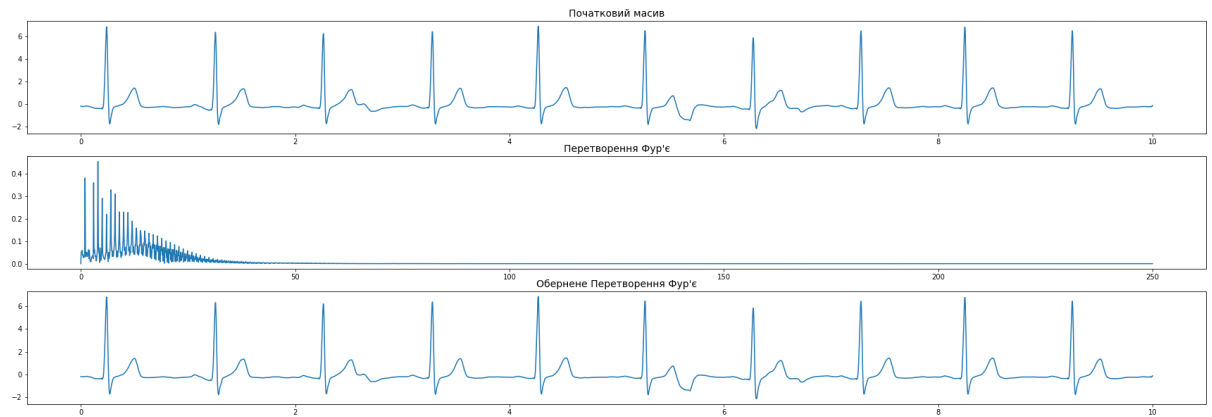
10:



11:



12:



З перетворень Фур'є бачимо, що велику роль відіграють низькі частоти. Скоріш за все, на цих частотах знаходяться “корисні” сигнали. Високі частоти не зашумлені - можливо для кардіограм використовувалося обладнання, котре прибирає шуми з сигналу. Початковий та результуючий масиви майже не відрізнити: похибка дуже мала і знаходиться в межах похибки під час роботи з дійсними числами.



## 6. Кореляційний аналіз

Кореляційна матриця:

	zero	one	two	three	four	five	six	seven	eight	nine	ten	eleven
zero	1.000000	0.924682	-0.926323	-0.989687	0.991254	-0.049926	-0.593749	0.311544	0.793519	0.882034	0.932605	0.937533
one	0.924682	1.000000	-0.717296	-0.968616	0.867153	0.328988	-0.641200	0.186380	0.685902	0.793849	0.875065	0.884732
two	-0.926323	-0.717296	1.000000	0.864342	-0.967452	0.416780	0.464709	-0.372705	-0.773311	-0.835575	-0.853158	-0.853824
three	-0.989687	-0.968616	0.864342	1.000000	-0.962534	-0.090348	0.622207	-0.271430	-0.768473	-0.864960	-0.927609	-0.934063
four	0.991254	0.867153	-0.967452	-0.962534	1.000000	-0.179372	-0.557084	0.341367	0.801689	0.881564	0.920343	0.923573
five	-0.049926	0.328988	0.416780	-0.090348	-0.179372	1.000000	-0.229523	-0.293527	-0.172301	-0.106125	-0.021720	-0.003194
six	-0.593749	-0.641200	0.464709	0.622207	-0.557084	-0.229523	1.000000	0.551170	-0.019049	-0.234700	-0.382262	-0.444091
seven	0.311544	0.186380	-0.372705	-0.271430	0.341367	-0.293527	0.551170	1.000000	0.810612	0.635428	0.510799	0.439328
eight	0.793519	0.685902	-0.773311	-0.768473	0.801689	-0.172301	-0.019049	0.810612	1.000000	0.954580	0.910675	0.870248
nine	0.882034	0.793849	-0.835575	-0.864960	0.881564	-0.106125	-0.234700	0.635428	0.954580	1.000000	0.965870	0.943713
ten	0.932605	0.875065	-0.853158	-0.927609	0.920343	-0.021720	-0.382262	0.510799	0.910675	0.965870	1.000000	0.976955
eleven	0.937533	0.884732	-0.853824	-0.934063	0.923573	-0.003194	-0.444091	0.439328	0.870248	0.943713	0.976955	1.000000

Для пошуку часткової кореляції та множинного коефіцієнту кореляції ми вибрали ознаки zero, one та four.

```
[54] def partial_corr(ab, bc, ac):  
      return (ab - ac * bc) / (((1-ac*ac) ** (1/2)) * ((1-bc*bc) ** (1/2)))  
  
      partial_corr(corr_matrix["zero"]["one"], corr_matrix["zero"]["four"], corr_matrix["one"]["four"])  
  
      0.99069843394116  
  
[56] def mult_corr_coef(ab, bc, ac):  
      return np.sqrt((ab*ab + ac*ac - 2*ab*ac*bc) / (1 - bc*bc))  
  
      mult_corr_coef(corr_matrix["zero"]["one"], corr_matrix["zero"]["four"], corr_matrix["one"]["four"])  
  
      0.9977008701016992
```

Як бачимо, кореляції близькі до 1.

Виходячи з цього та з таблиці кореляцій, робимо висновки, що незалежних параметрів в даних немає (більшість кореляцій близька до 0.8-0.9 за модулем, що свідчить про лінійний зв'язок).

## 7. Факторний аналіз

Проведемо тест Кайзера-Мейєра-Олкіна для визначення придатності даних щодо використання їх у факторному аналізі.

### 7. Factor analysis

```
[29]
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

from factor_analyzer import FactorAnalyzer

import matplotlib.pyplot as plt
from factor_analyzer.factor_analyzer import calculate_kmo

kmo_all, kmo_model = calculate_kmo(df)
kmo_model

0.8435104478982646
```

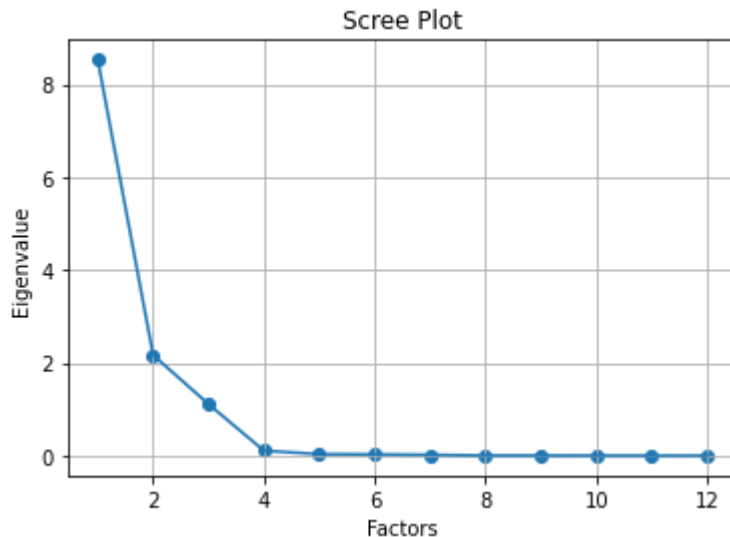
Маємо  $0.84 > 0.6$  -> можемо використовувати наявні дані.

```
fa = FactorAnalyzer(rotation=None)
fa.fit(df)
ev,v = fa.get_eigenvalues()

[31] ev

array([8.53293487e+00, 2.16658961e+00, 1.11041415e+00, 1.06583048e-01,
       3.25363271e-02, 2.62784216e-02, 1.74825113e-02, 3.06244673e-03,
       2.23901729e-03, 1.38858217e-03, 4.14589203e-04, 7.64317872e-05])
```

За критерієм Кайзера можемо вибрати 3 фактори, бо їх власні числа більше за 1.



Будуємо діаграму осипання. Бачимо, що після 3 компонент швидкість спадання графіку спадає.

Проведемо факторний аналіз.

```

%%
fa.set_params(n_factors=3, rotation = 'varimax')
fa.fit(df)
fa.loadings_

array([[ 0.99282164, -0.05859168, -0.05957387],
       [ 0.93589944, -0.11263927,  0.31377826],
       [-0.90248621,  0.00976093,  0.42305758],
       [-0.98931483,  0.07911277, -0.07904046],
       [ 0.97798433, -0.03979994, -0.18771632],
       [-0.00249897, -0.16511156,  0.98407165],
       [-0.55944793,  0.7967041 , -0.10276452],
       [ 0.35116314,  0.90980966, -0.14640239],
       [ 0.82836385,  0.55364309, -0.07928772],
       [ 0.91419292,  0.34650449, -0.04469842],
       [ 0.96185993,  0.20412991,  0.01770539],
       [ 0.96318203,  0.1290813 ,  0.02308944]])

[34] fa.get_factor_variance()

(array([8.42241442, 1.99872864, 1.33199069]),
 array([0.70186787, 0.16656072, 0.11099922]),
 array([0.70186787, 0.86842859, 0.97942781]))

```

Загальна описана доля дисперсії  $\sim 0.98$ , що є дуже непоганим результатом.

Також трансформуємо дані для подальшого використання в кластеризації.

```
[35] pca_df = pd.DataFrame(fa.transform(df.values))
pca_df
```

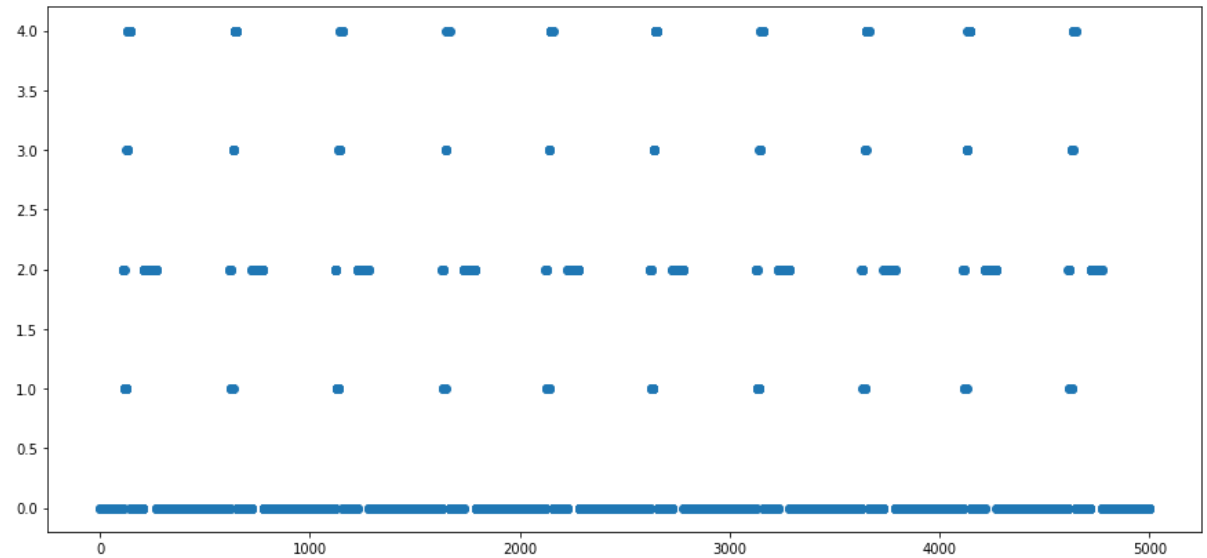
	0	1	2
0	-0.422592	0.031750	-0.972048
1	-0.428145	0.054296	-0.919720
2	-0.436213	0.094130	-0.831982
3	-0.447347	0.158629	-0.692615
4	-0.459861	0.241882	-0.510290
...	...	...	...
4995	-0.216727	-0.115848	0.365756
4996	-0.201828	-0.063041	0.515681
4997	-0.190459	0.010951	0.691416
4998	-0.181645	0.095123	0.873690
4999	-0.173843	0.181327	1.053416

5000 rows × 3 columns

## 8. Кластерний аналіз

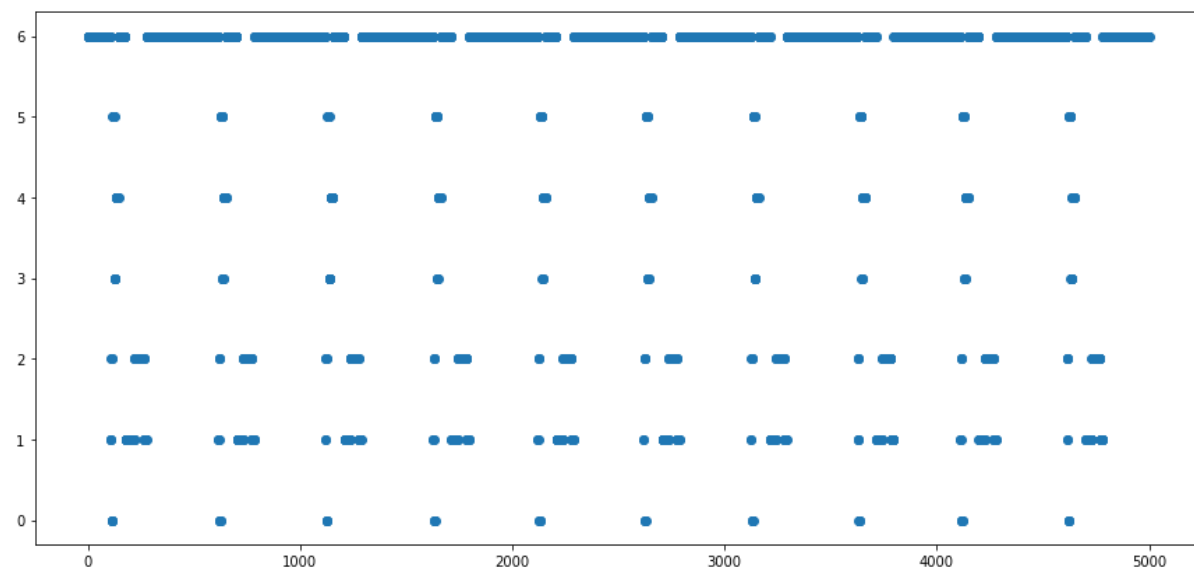
Кластруємо дані з допомогою алгоритму k-means на 5 та 7 кластерів.

5 кластерів:



cluster zero		
0	0	4084
2	2	618
4	4	138
1	1	110
3	3	50

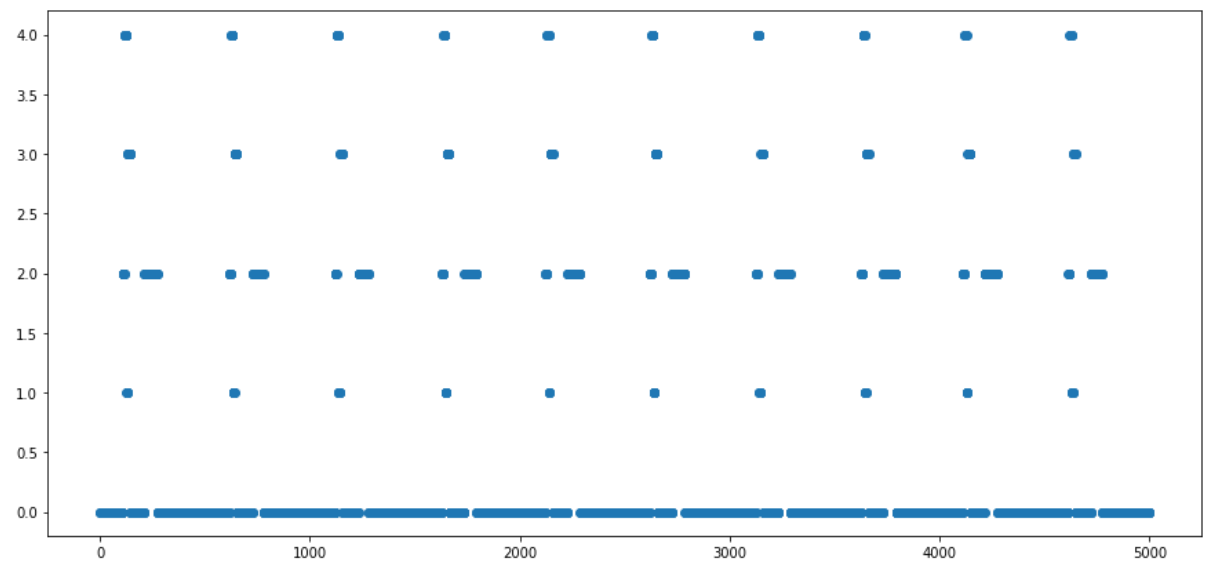
7 кластерів:



cluster zero		
6	6	3797
1	1	448
2	2	442
4	4	136
5	5	84
3	3	50
0	0	43

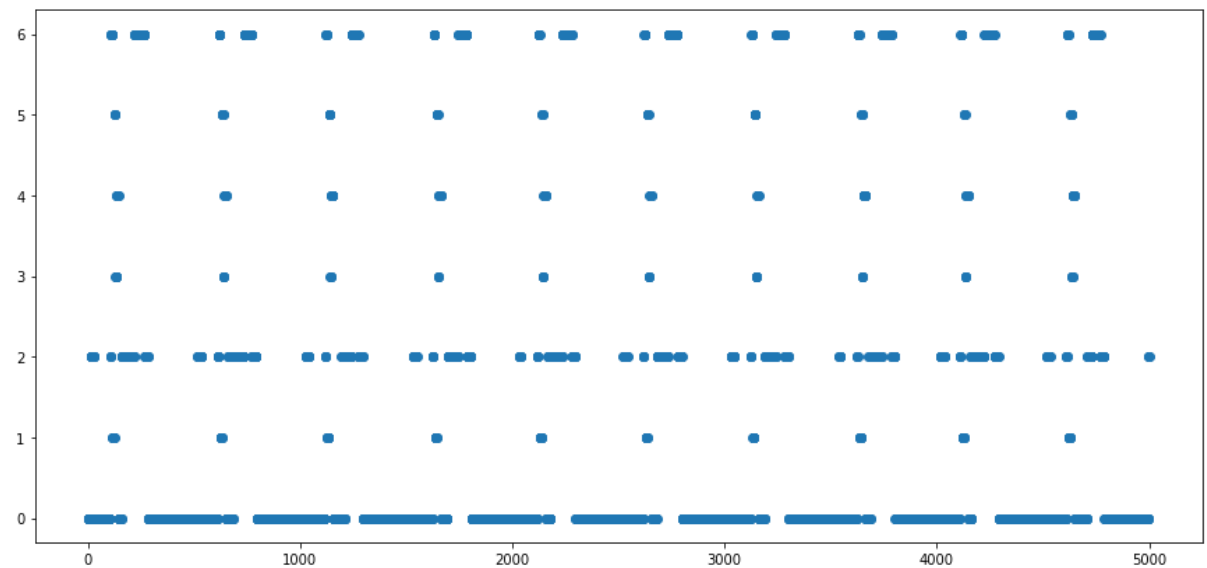
Повторюємо процедуру на даних, які ми отримали при факторному аналізі.

5 кластерів:



cluster zero		
0	0	4054
2	2	658
3	3	133
4	4	95
1	1	60

7 кластерів:



cluster	zero	count
0	0	3307
2	2	904
6	6	499
4	4	103
1	1	89
5	5	55
3	3	43

Бачимо, що для 5 кластерів кластеризація змінилася не дуже сильно, але для 7 кластерів алгоритм показав інші результати - приблизно 500 об'єктів змінили свій кластер з найбільшого за розміром на другий за розміром.