```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
```

Ogólny opis danych

Główny wektor

Główny wektor danych (X_test i X_train) jest stworzony na podstawie danych żyroskopowych pobranych z telefonów, podczas gdy człowiek wykonywał pewne czynności (chodzi, wchodzi po schodach, schodzi po schodach, siedzi, stoi, leży). Orginalnym celem w tym zadaniu było przewidzenie na podstawie tych danych jaką aktywność, ale skoro my zajmujemy się klasteryzacją to nie będziemy korzystać z pliku y_train/y_test.

Do EDA połączymy zbiór treningowy i testowy w jeden.

```
def read_uci_data(path, colnames_path=None):
    colnames = None
    if colnames_path is not None:
        colnames = open("/data/workspace_files/UCI HAR Dataset/" + colnames_path).read().split("\n")[
    return pd.read_table(
        "/data/workspace_files/UCI HAR Dataset/" + path,
        sep="[ ]+",
        header=None,
        engine="python",
        names=colnames
    )
```

```
X_train = read_uci_data("train/X_train.txt", colnames_path="features.txt")
X_test = read_uci_data("test/X_test.txt", colnames_path="features.txt")
X = X_train.append(X_test)
```

```
X.shape
```

```
(10299, 561)
```

Ramka jest dosyć duża - 561 kolumn wygląda groźnie, ale większość to są różnorakie statystyki. Dodatkowo wiele z nich opisuje to prawie to samo (na przykład std, mad i iqr, które wszystkie opisują rozrzut), więc przyjrzymy się tylko niektórym.

Dane opisują zjawiska dziejące się w trzech wymiarach - więc prawie każda cecha ma trzy warianty.

Niektóre statystyki opisują sygnały po transformacie Fouriera, czyli są w świecie częstotliwości, a nie czasów.

```
X
```

	1 tBodyAcc-mean()-X	2 tBodyAcc-mean()-Y	3 tBodyAcc-mean()-Z	4 tBodyAcc-std()-X	5 tBodyAcc-std()-Y	6 tBodyAcc-std()-Z	7 tBodyAcc-mad()-X	8 tBodyAcc-mad()-Y	9 tBodyAcc-mad()-Z	10 tBodyAcc-max()-X	...	552 fBodyBodyGyr meanFreq()
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.074323
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.158075
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	0.414503
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	0.404573
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	0.087753
...	...	...	...	...	...	...	...	...	...	...	...	...
2942	0.310155	-0.053391	-0.099109	-0.287866	-0.140589	-0.215088	-0.356083	-0.148775	-0.232057	0.185361	...	0.074472
2943	0.363385	-0.039214	-0.105915	-0.305388	0.028148	-0.196373	-0.373540	-0.030036	-0.270237	0.185361	...	0.101859
2944	0.349966	0.030077	-0.115788	-0.329638	-0.042143	-0.250181	-0.388017	-0.133257	-0.347029	0.007471	...	-0.066249
2945	0.237594	0.018467	-0.096499	-0.323114	-0.229775	-0.207574	-0.392380	-0.279610	-0.289477	0.007471	...	-0.046467
2946	0.153627	-0.018437	-0.137018	-0.330046	-0.195253	-0.164339	-0.430974	-0.218295	-0.229933	-0.111527	...	-0.010386

10299 rows × 561 columns

Dane wyglądają podejrzanie: wszystkie są w zakresie od -1 do 1, więc prawdopodobnie zostały już przeskalowane.

```
X.describe().iloc[:, :9]
```

	1 tBodyAcc-mean()-X	2 tBodyAcc-mean()-Y	3 tBodyAcc-mean()-Z	4 tBodyAcc-std()-X	5 tBodyAcc-std()-Y	6 tBodyAcc-std()-Z	7 tBodyAcc-mad()-X	8 tBodyAcc-mad()-Y	9 tBodyAcc-mad()-Z
count	10299.000000	10299.000000	10299.000000	10299.000000	10299.000000	10299.000000	10299.000000	10299.000000	10299.000000
mean	0.274347	-0.017743	-0.108925	-0.607784	-0.510191	-0.613064	-0.633593	-0.525697	-0.614989
std	0.067628	0.037128	0.053033	0.438694	0.500240	0.403657	0.413333	0.484201	0.399034
min	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	0.262625	-0.024902	-0.121019	-0.992360	-0.976990	-0.979137	-0.993293	-0.977017	-0.979064
50%	0.277174	-0.017162	-0.108596	-0.943030	-0.835032	-0.850773	-0.948244	-0.843670	-0.845068
75%	0.288354	-0.010625	-0.097589	-0.250293	-0.057336	-0.278737	-0.302033	-0.087405	-0.288149
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Dane są faktycznie przeskalowane - każda wartość w wektorze cech jest w zakresie [-1, 1]. Nie ma żadnych zmiennych kategorycznych, zostało tylko zabrać się do pracy.

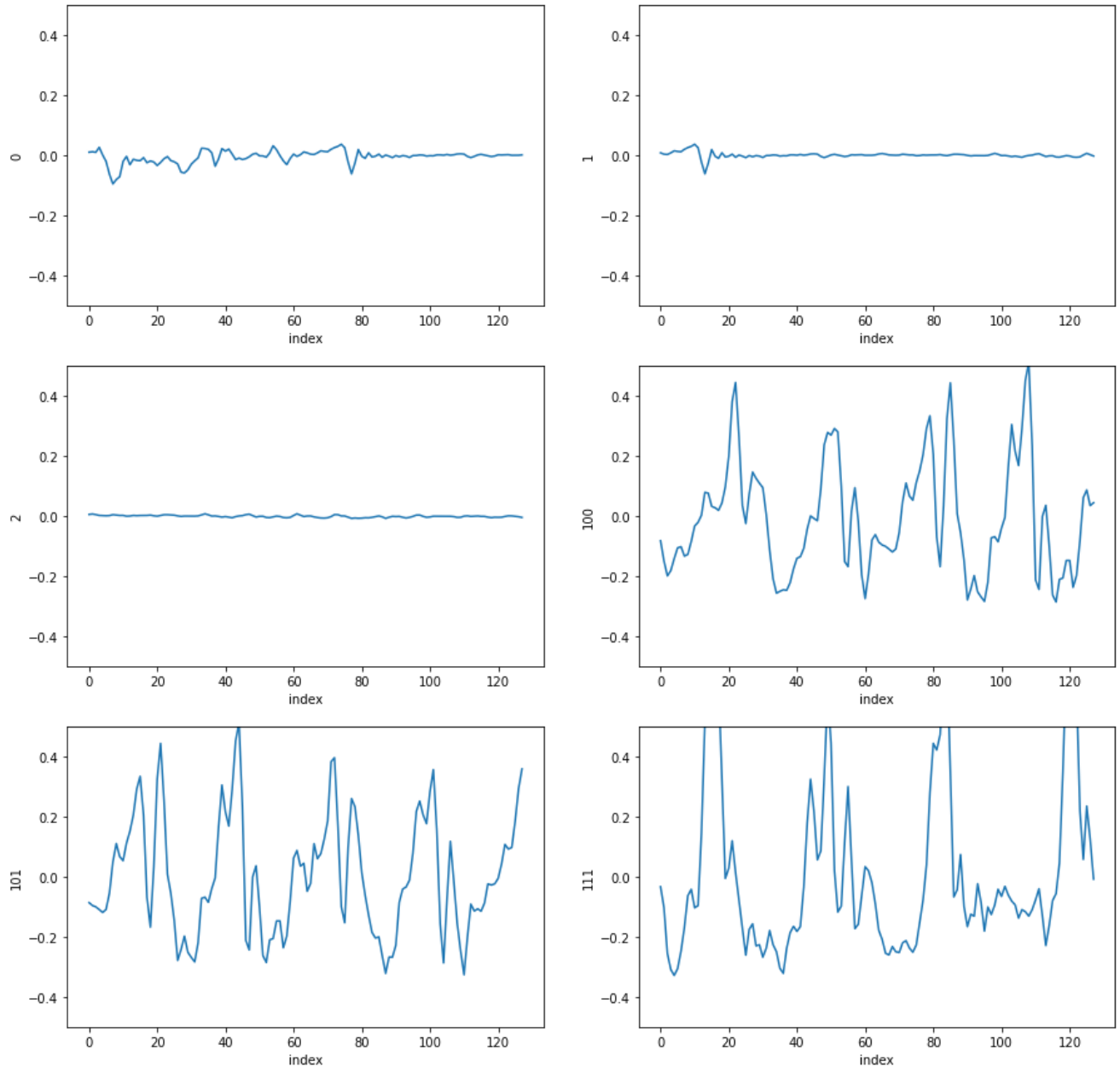
Serie czasowe

Poza wektorem cech X w zbiorze są jeszcze dane sygnałów, na podstawie których te statystyki są stworzone. Warto może się przyjrzeć jak one wyglądają.

Jest ich tyle samo co w ramce X, a rysowanie 10 000 wykresów jest raczej zbyt ambitną wizualizacją. Dlatego wybraliśmy kilka, które porównamy pomiędzy sobą.

```
body_acc_x_test = read_uci_data("test/Inertial Signals/body_acc_x_test.txt")
```

```
def plot_timeseries(data, row, **kwargs):
    sns.lineplot(
        data=data.iloc[row, :].reset_index(),
        x="index",
        y=row,
        **kwargs
    )


def plot_timeseries_facet(data, *which, y_lim=(-0.5, 0.5), cols=2):
    _fig, _axs = plt.subplots((len(which)+1)//cols, cols)

    for i, el in enumerate(which):
        plot_timeseries(data, el, ax=_axs[i//cols, i%cols])
        _axs[i//cols,i%cols].set_ylim(y_lim[0], y_lim[1])
    return _fig


fig = plot_timeseries_facet(body_acc_x_test, 0, 1, 2, 100, 101, 111)
fig.set_size_inches(15,15)
plt.show()
```



Jest definitywna różnica pomiędzy różnymi sygnałami - niektóre są idealnie płaskie, inne mają dużo skoków, a niektóre tylko raz skoczą.

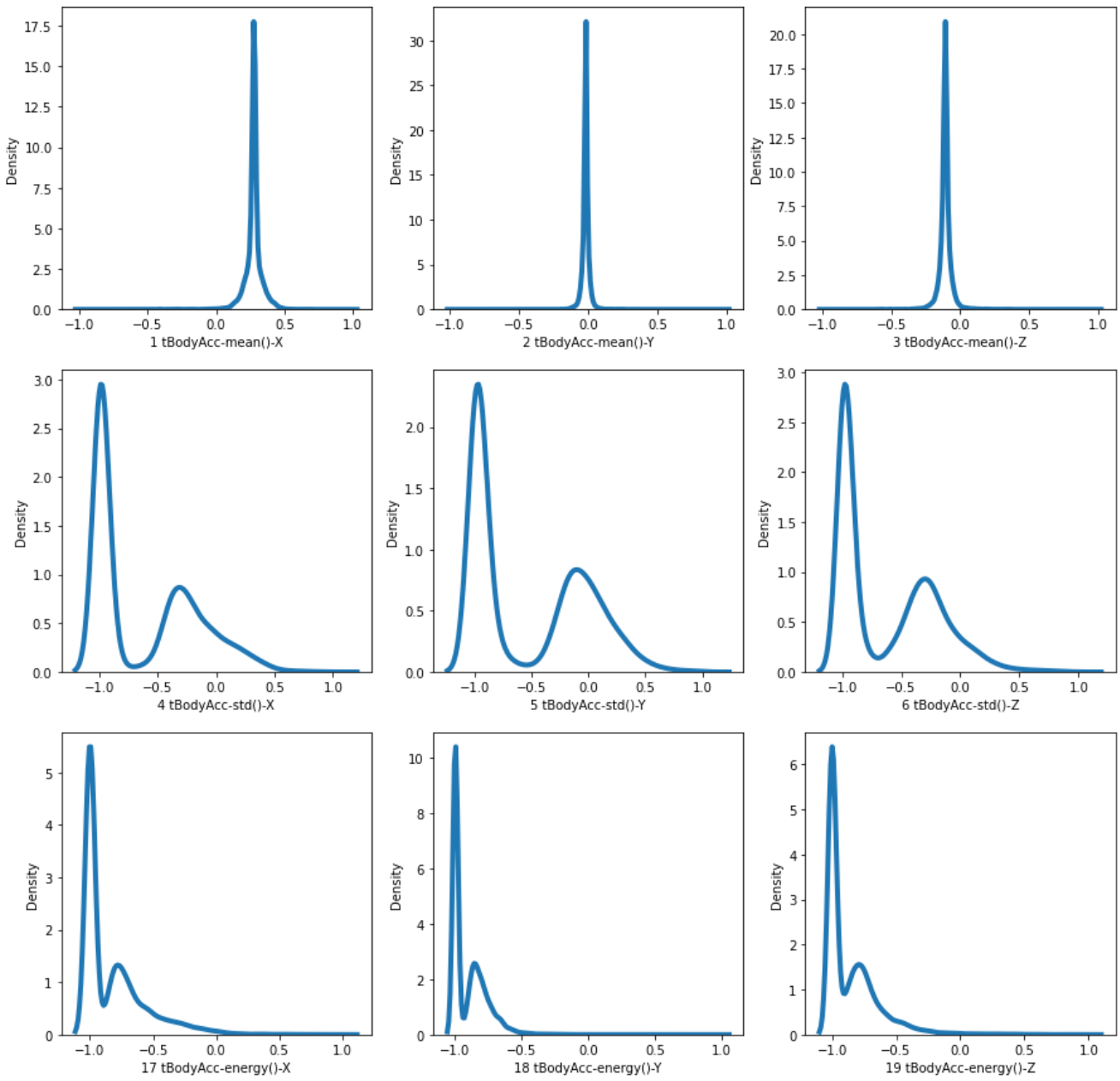
Dystrybucje

Wniknijmy w to co mogą pokazać nam rozkłady poszczególnych zmiennych, oczywiście również ograniczymy się do zademonstrowania kilku dystrybucji, gdyż przedstawienie wszystkich 561 byłboby wizualnie nieczytelne i redundantne.

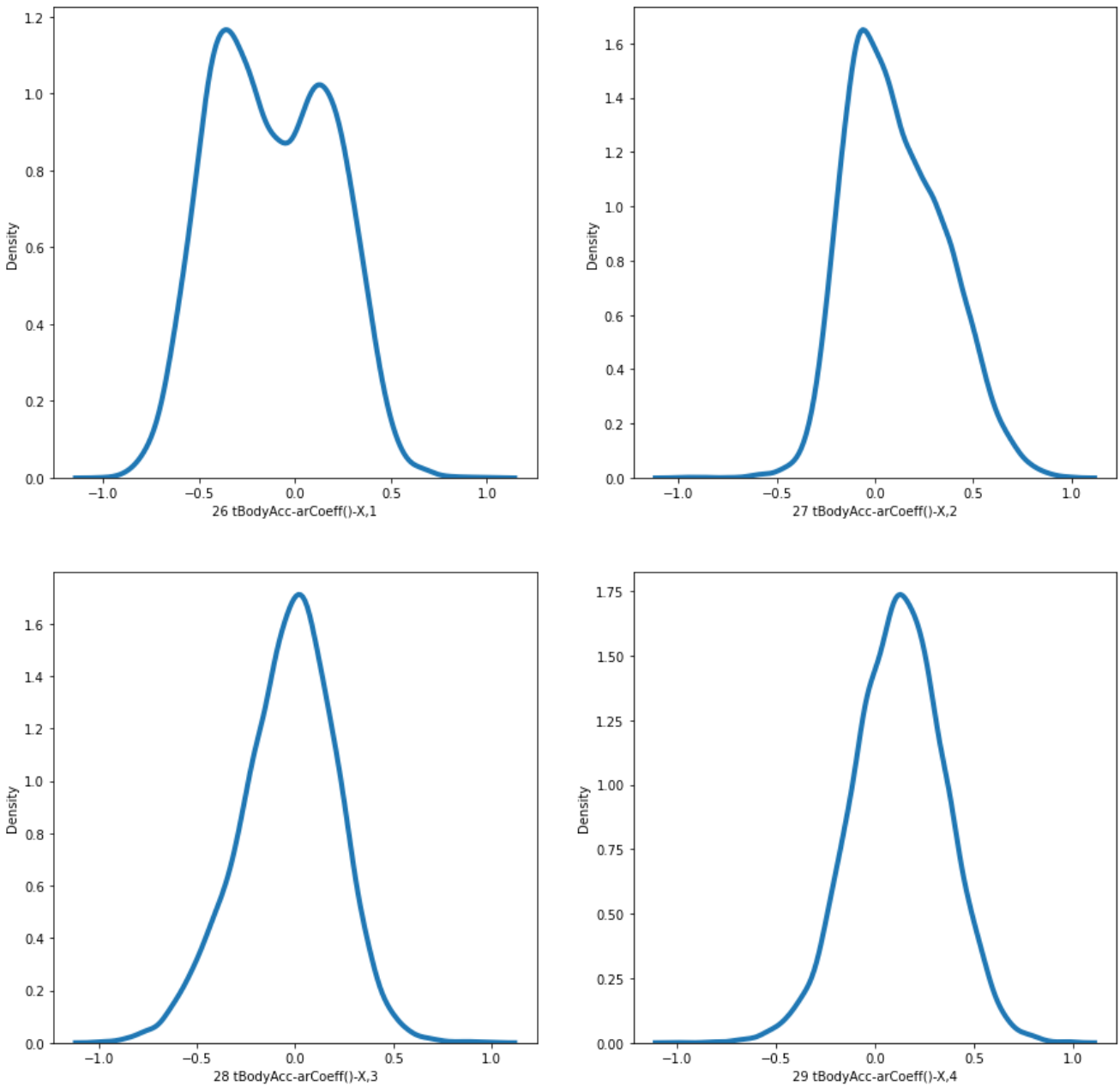
```
import statistics as stat
def get_statistical_data(data):
    try:
        stat.mode(data)
    except ValueError:
        print("Mean:", stat.mean(data),
          "\nMedian:", stat.median(data),
          "\nMode: Provided sample consists of unique values, hence there is no mode",
          "\nVariance:", stat.variance(data)
         )
        return
    print("Mean:", stat.mean(data),
          "\nMedian:", stat.median(data),
          "\nMode:", stat.mode(data),
          "\nVariance:", stat.variance(data)
         )
```

```
def plot_distribution(data,col, **kwargs):
    sns.distplot(
        data.iloc[:,col], kde=True,hist = False,
        bins=int(180/5), color = 'C0',
        norm_hist = True,
        hist_kws={'edgecolor':'black'},
        kde_kws={'linewidth': 4},
        **kwargs
    )

import warnings
warnings.filterwarnings("ignore")
```

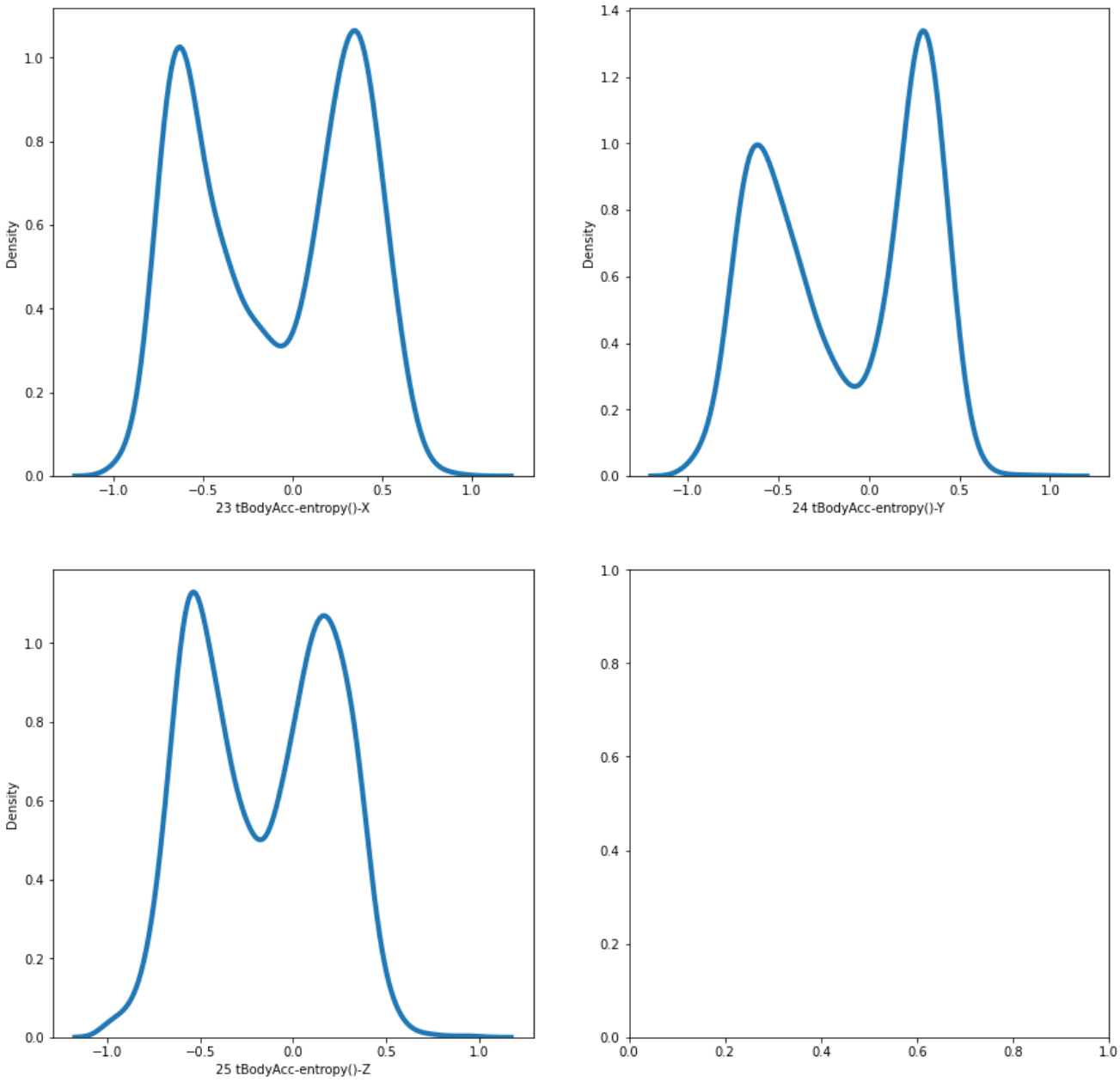
```
def plot_distribution_facet(data, *which, cols=2): # masz tutaj funkcje do facet gridów
    _fig, _axs = plt.subplots((len(which)+1)//cols, cols)

    for i, el in enumerate(which):
        plot_distribution(data, el, ax=_axs[i//cols, i%cols])
        #_axs[i//cols,i%cols].set_ylim(y_lim[0], y_lim[1])
    return _fig

fig_dist = plot_distribution_facet(X, *([i for i in range(6)] + [16, 17, 18]), cols=3)
fig_dist.set_size_inches(15,15)
plt.show()
```



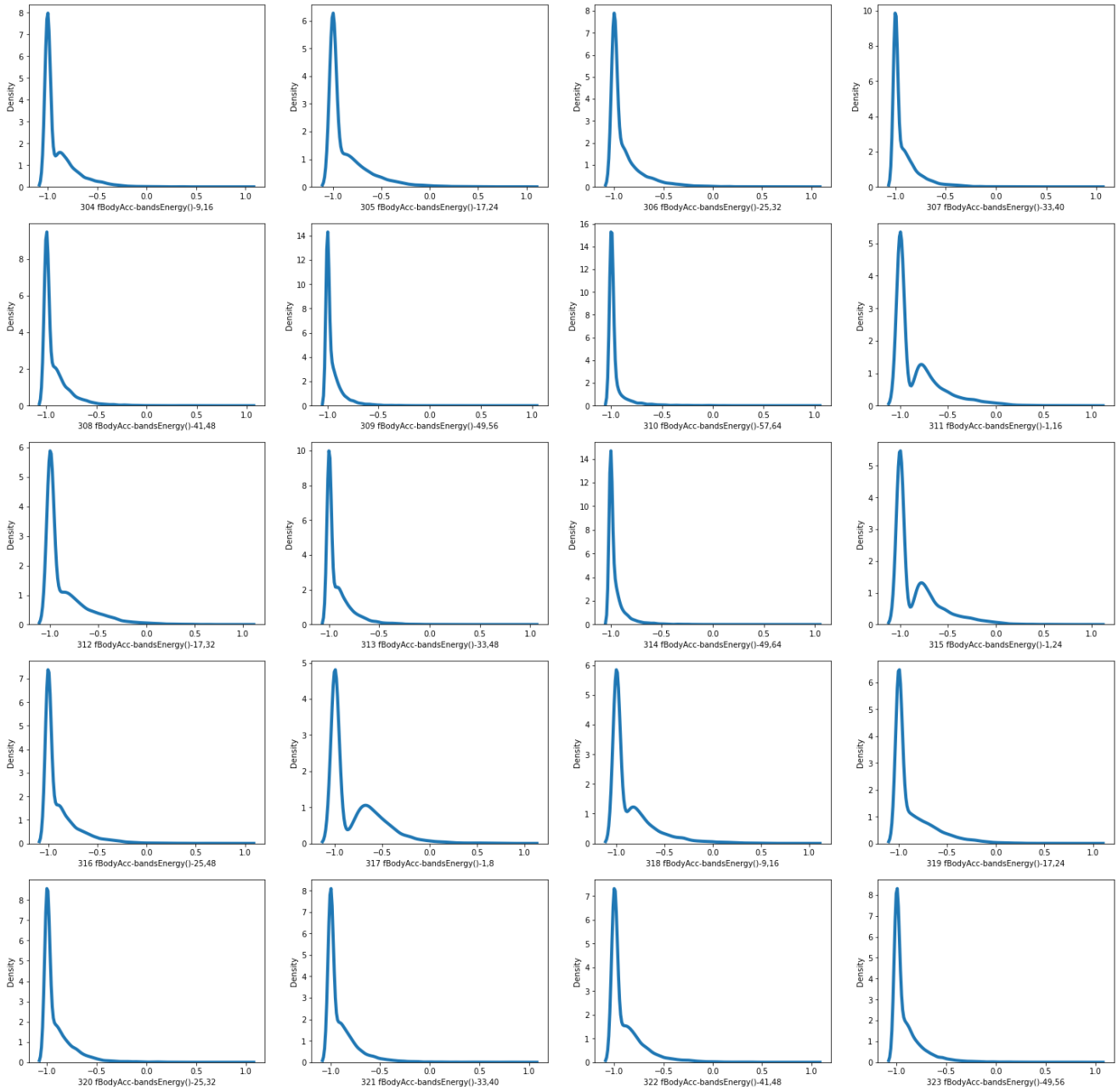

```
i = 26
fig_dist2 = plot_distribution_facet(X, i-1, i, i+1, i+2, cols=2)
fig_dist2.set_size_inches(15,15)
plt.show()
```




```
i = 23
fig_dist2 = plot_distribution_facet(X, i-1, i, i+1, cols=2)
fig_dist2.set_size_inches(15,15)
plt.show()
```




```
fig_dist2 = plot_distribution_facet(X, *[i for i in range(303, 323)], cols=4)
fig_dist2.set_size_inches(25,25)
plt.show()
```



Wybór cech

Wiemy, iż mamy do czynienia z dość sporą ramką danych, w szczególności charakteryzuje się nieprzeciętną ilością kolumn. Jednakże, jak już wcześniej zauważyliśmy, większość kolumn odpowiada pewnym statystykom z danych obserwacji, i jak wiemy, takie obiekty bywają silnie zależne od siebie, a zatem w przyszłej analizie mogą okazać się zbędne.
Dokonajmy "strzyżenia" naszej ramki, i zachowajmy jedynie kolumny, które, jak narazie naszym zdaniem, wnoszą jak najwięcej unikalnych danych do analizy.

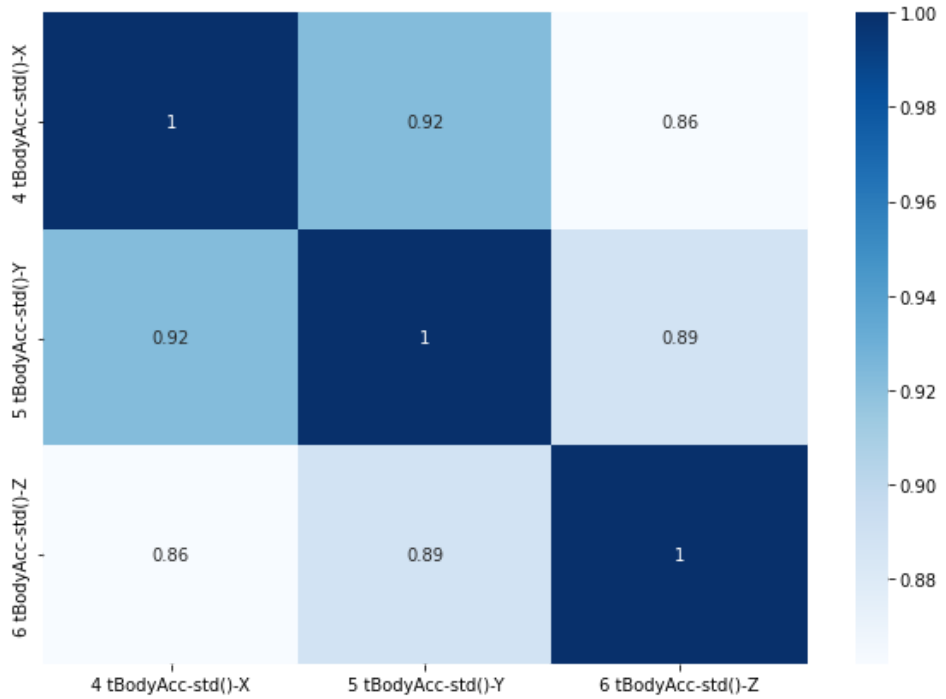
```
colnames = open("/data/workspace_files/UCI HAR Dataset/features.txt").read().split("\n")[:-1]
to_keep = ["mean", "std", "correlation" "energy", "entropy", "angle", "skewness", "kurtosis"] # <----
keep_colnames = []
for colname in colnames:
    if colname[0] == "f":
        pass
    elif any(list(map(lambda x: x in colname, to_keep))):
        keep_colnames.append(colname)

X_trimmed = X[keep_colnames]
X_trimmed.shape
# nadal dużo...
```

```
(10299, 145)
```

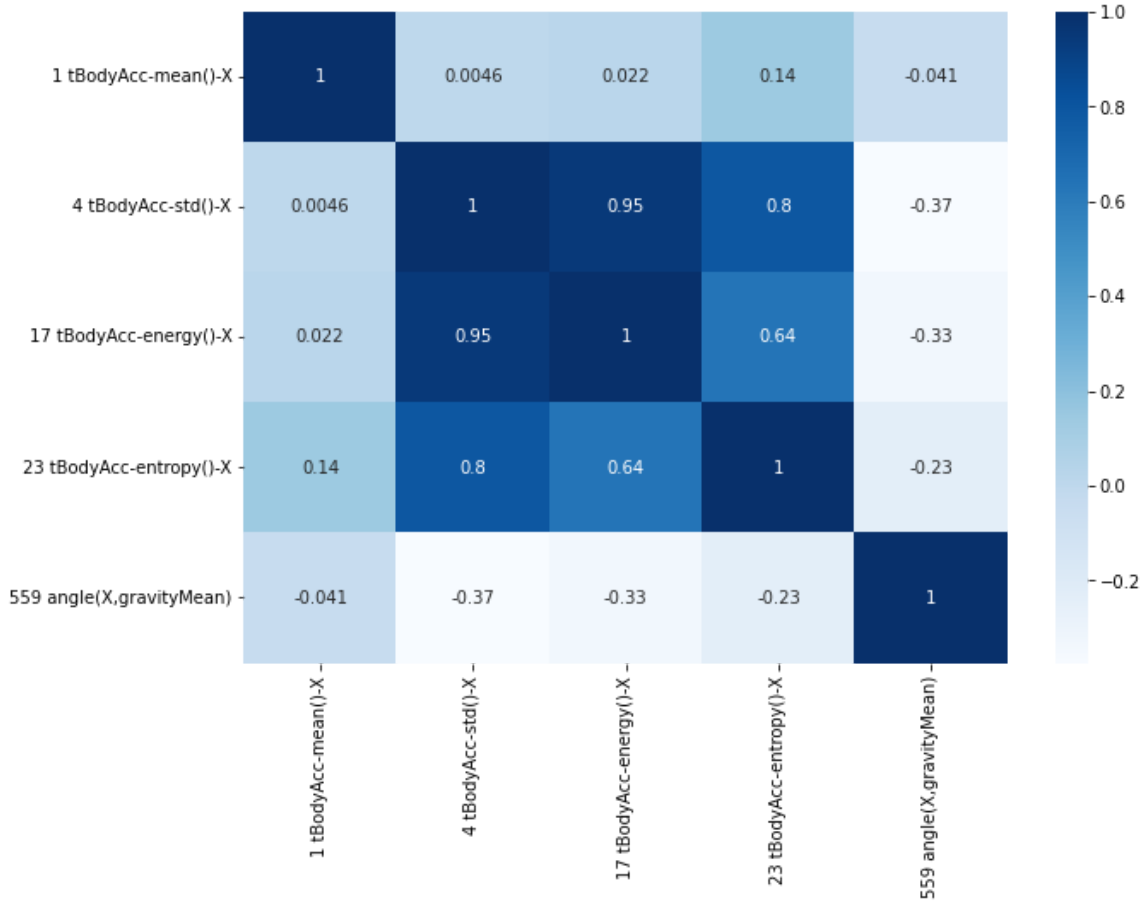
Zależności

W tej sekcji wniknijmy jak dane oddziaływują między sobą, domniemamy a priori, iż dla dużej większości kolumn korelacja będzie na wysokim poziomie ze względu na wspólną geneze danych.

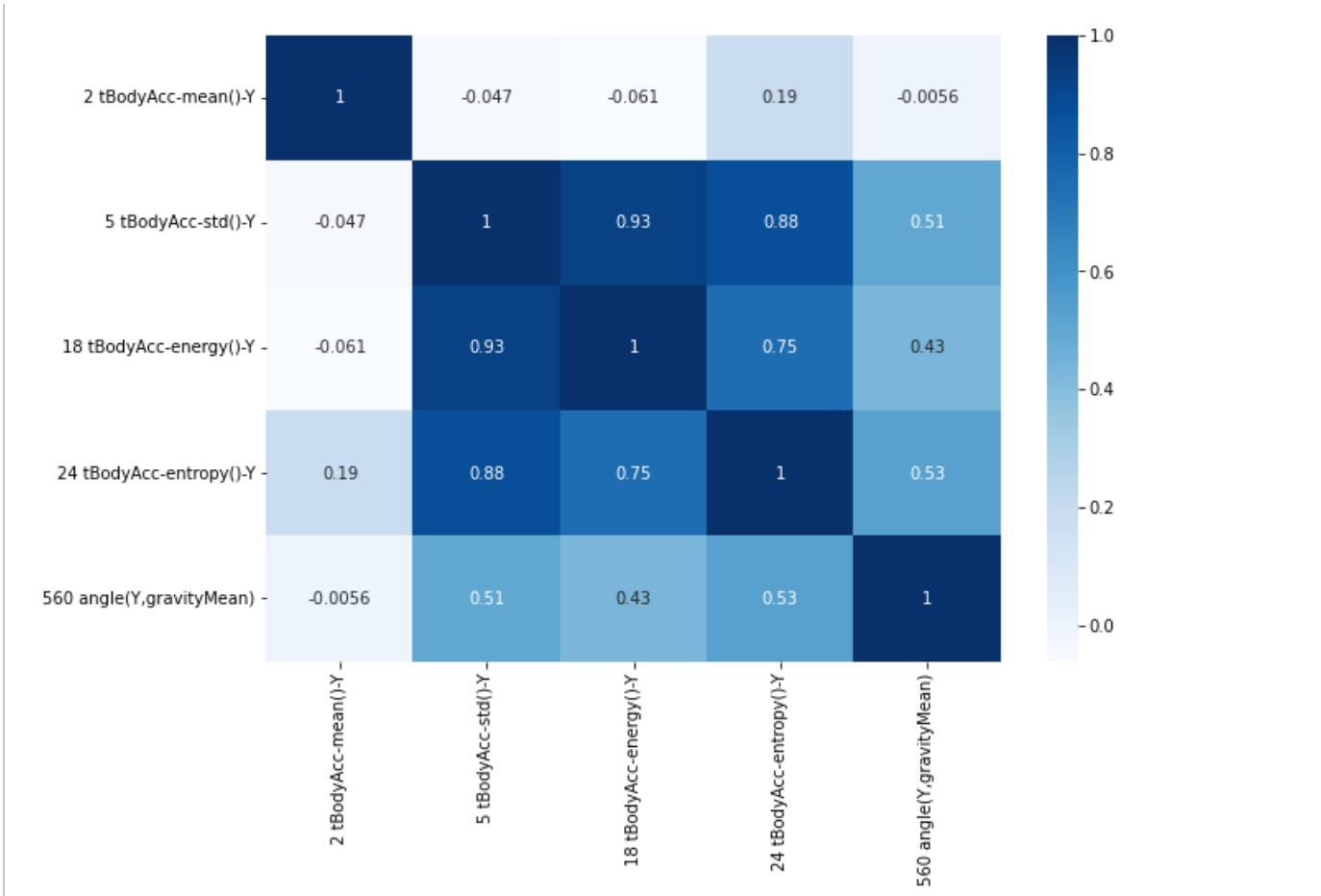
```
def corr_map(data, columns):    #columns to lista nazw lub numerow kolumn
    for i in range(0, len(columns)):
        if type(columns[i]) is int:
            columns[i] = data.columns[columns[i]]
    plt.figure(figsize=(10, 7))
    corr = data[columns].corr()
    sns.heatmap(corr, annot=True, cmap = "Blues")
    plt.yticks(va='center')

#corr_map(X, ['1 tBodyAcc-mean()-X', '2 tBodyAcc-mean()-Y', '3 tBodyAcc-mean()-Z'])
#corr_map(X, ['1 tBodyAcc-mean()-X', '4 tBodyAcc-std()-X'])
corr_map(X, ['4 tBodyAcc-std()-X', 4, 5])
plt.show()
```



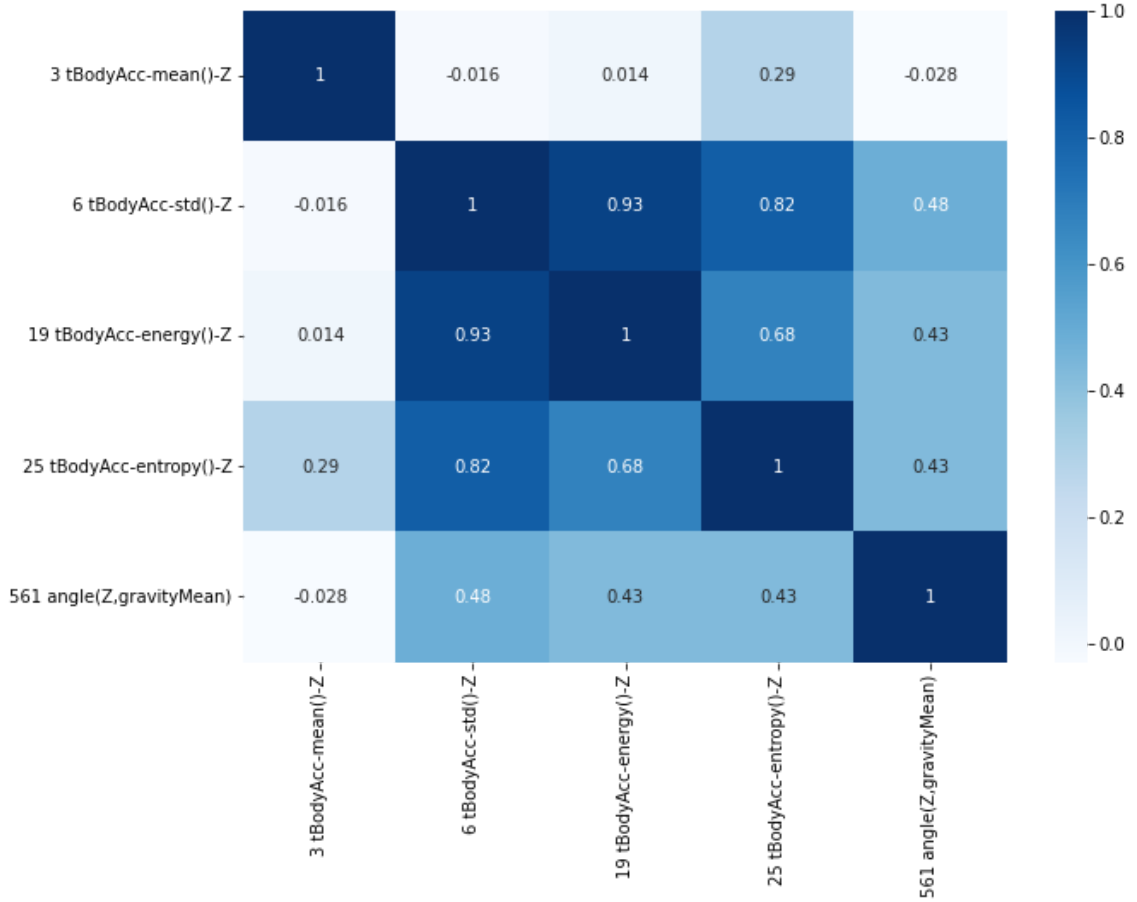
```
# ------------------------------------------------
# tutaj myślę spokojnie możemy napisać coś o korelacji pomiędzy X Y Z
# narysować jeszcze pomiędzy mean/std/energy/entropy jak wygląda, dla np X
corr_map(X,['1 tBodyAcc-mean()-X','4 tBodyAcc-std()-X','17 tBodyAcc-energy()-X','23 tBodyAcc-entropy(
#Zmienne dla X, niżej dałem dla Y oraz dla Z, bo zaskazująco X okazało się specjalne a Y i Z dość zre
#Można o tym wspomnieć
```




```
corr_map(X,['2 tBodyAcc-mean()-Y','5 tBodyAcc-std()-Y','18 tBodyAcc-energy()-Y','24 tBodyAcc-entropy(
```




```
corr_map(X,['3 tBodyAcc-mean()-Z','6 tBodyAcc-std()-Z','19 tBodyAcc-energy()-Z','25 tBodyAcc-entropy(
```



Dokonajmy teraz eksperymentu i zbadajmy ilość par kolumn, dla których współczynnik korelacji przekracza 95% oraz zobaczmy najwyższe wyniki. Pomimo tak wysokiego wymagania dotyczącego korelacji i tak spodziewamy się dość pokaźnej ilości wyników ze względu na źródło pochodzenia danych. Może uda nam się zaobserwować coś interesującego?

```
columns = X.columns
name_1 = []
name_2 = []
corr_val = []
for i in range(0, len(columns)):
    for j in range(i+1, len(columns)):
        corr = X[columns[i]].corr(X[columns[j]])
        if corr > 0.95:
            name_1.append(columns[i])
            name_2.append(columns[j])
            corr_val.append(corr)

data = {'Variable_1':name_1,'Variable_2':name_2,'Correlation value':corr_val}
corr_data_frame = pd.DataFrame(data)
corr_data_frame.sort_values(by = 'Correlation value')
print(corr_data_frame.head(5))
corr_data_frame.shape
```

```
           Variable_1              Variable_2  Correlation value
0  4 tBodyAcc-std()-X       7 tBodyAcc-mad()-X           0.998662
1  4 tBodyAcc-std()-X      10 tBodyAcc-max()-X           0.981226
2  4 tBodyAcc-std()-X        16 tBodyAcc-sma()           0.974977
3  4 tBodyAcc-std()-X      20 tBodyAcc-iqr()-X           0.980537
4  4 tBodyAcc-std()-X  84 tBodyAccJerk-std()-X           0.972063

(2127, 3)
```

Spróbujmy dokonać niemożliwego i postarać się wykonać heat-mapę dla całej naszej ramki.
Oczywiście dla naszego przypadku nie będzie to takie proste, dlatego wynonajmy pierw kilka operacji.

```
from PIL import Image

x = np.array([
    [-1.00, -0.75, -0.50],
    [-0.25,  0.00,  0.25],
    [ 0.50,  0.75,  1.00],
    ])
x= x*96 + 128
cmap = plt.get_cmap("seismic")
# (cmap(x.flatten().astype("uint8")) * 255).reshape(3, 3, 4).astype(np.uint8)
```

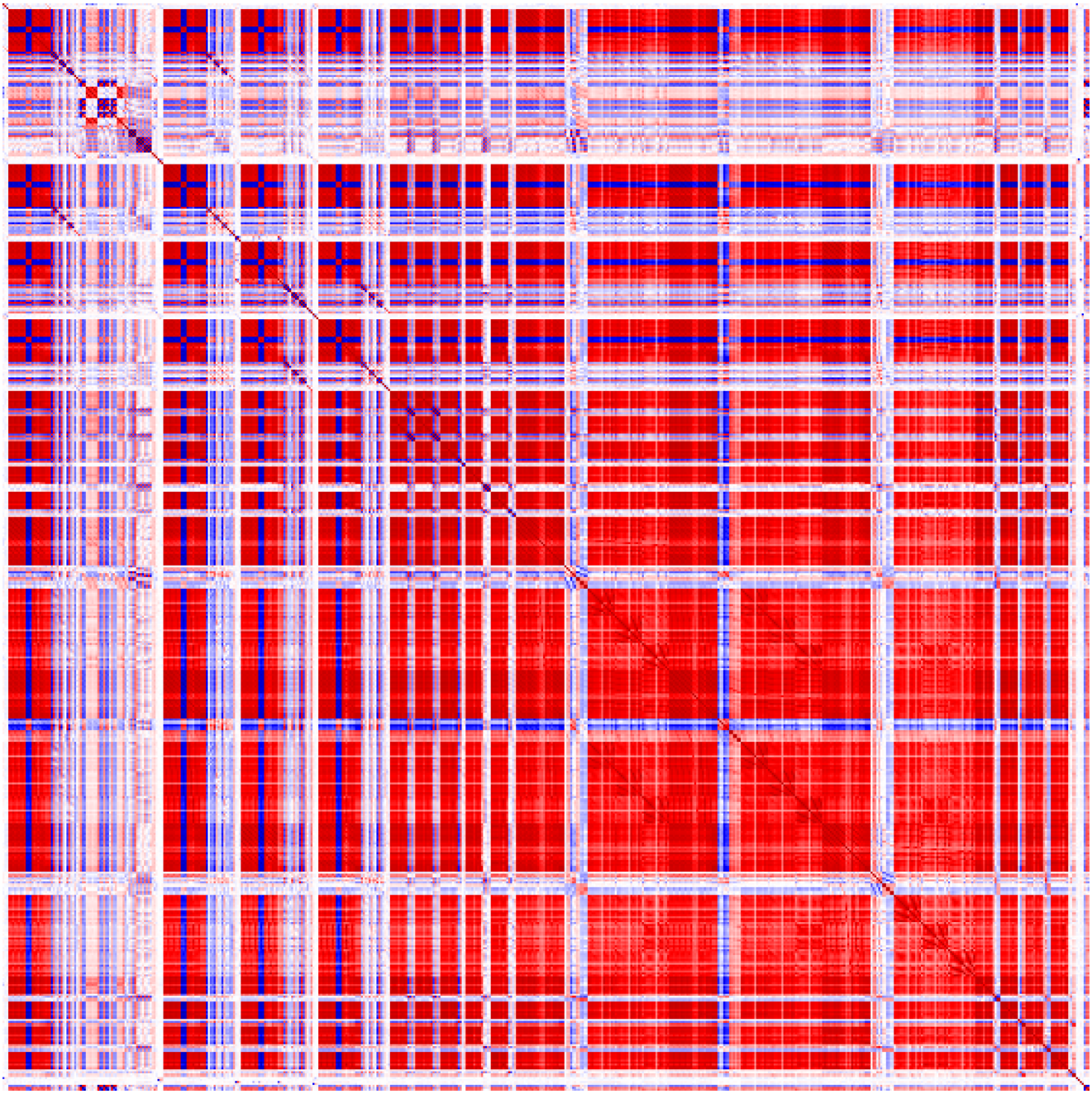
```
X_corr = X.corr()
```

```
X_corr= X_corr*96 + 128
```

```
X_corr.values
```

```
array([[224.        , 140.29150544, 105.89103297, ..., 124.06194533,
        131.26912398, 130.94302316],
       [140.29150544, 224.        , 125.1313423 , ..., 127.27871249,
        127.46086757, 126.44165913],
       [105.89103297, 125.1313423 , 224.        , ..., 128.30867595,
        126.75333708, 125.27305647],
       ...,
       [124.06194533, 127.27871249, 128.30867595, ..., 224.        ,
         56.16805786,  67.01786449],
       [131.26912398, 127.46086757, 126.75333708, ...,  56.16805786,
        224.        , 180.37896673],
       [130.94302316, 126.44165913, 125.27305647, ...,  67.01786449,
        180.37896673, 224.        ]])
```

```
i= Image.fromarray(
    (cmap(X_corr.values.flatten().astype("uint8")) * 255).reshape(561, 561, 4).astype(np.uint8),
    "RGBA"
)
i.resize((561*2, 561*2), 0) # cechy 26-80 to jest to centrum
```

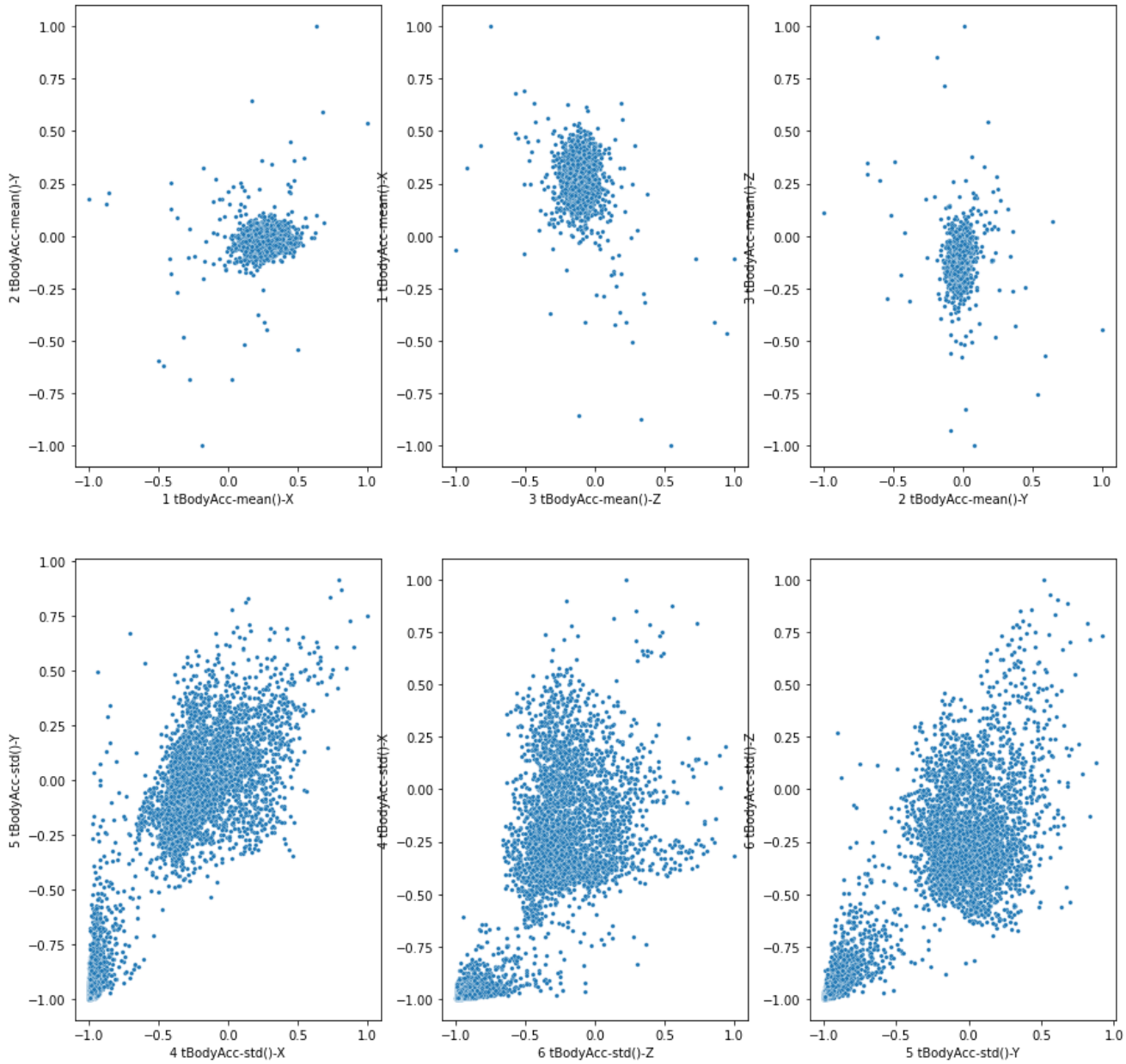


Teraz przyjrzyjmy się nieco inaczej zwizualizowanyn danych w postaci scatter plotów.
Jak już wcześniej zauważyliśmy w rozkładach, dla naszych danych występują pewne "grupy", ale nie
dla wszystkich zmiennych. Przyjrzymy się kilku scatter plotom, ale najpierw kilka newralgicznych dlań funkcji.

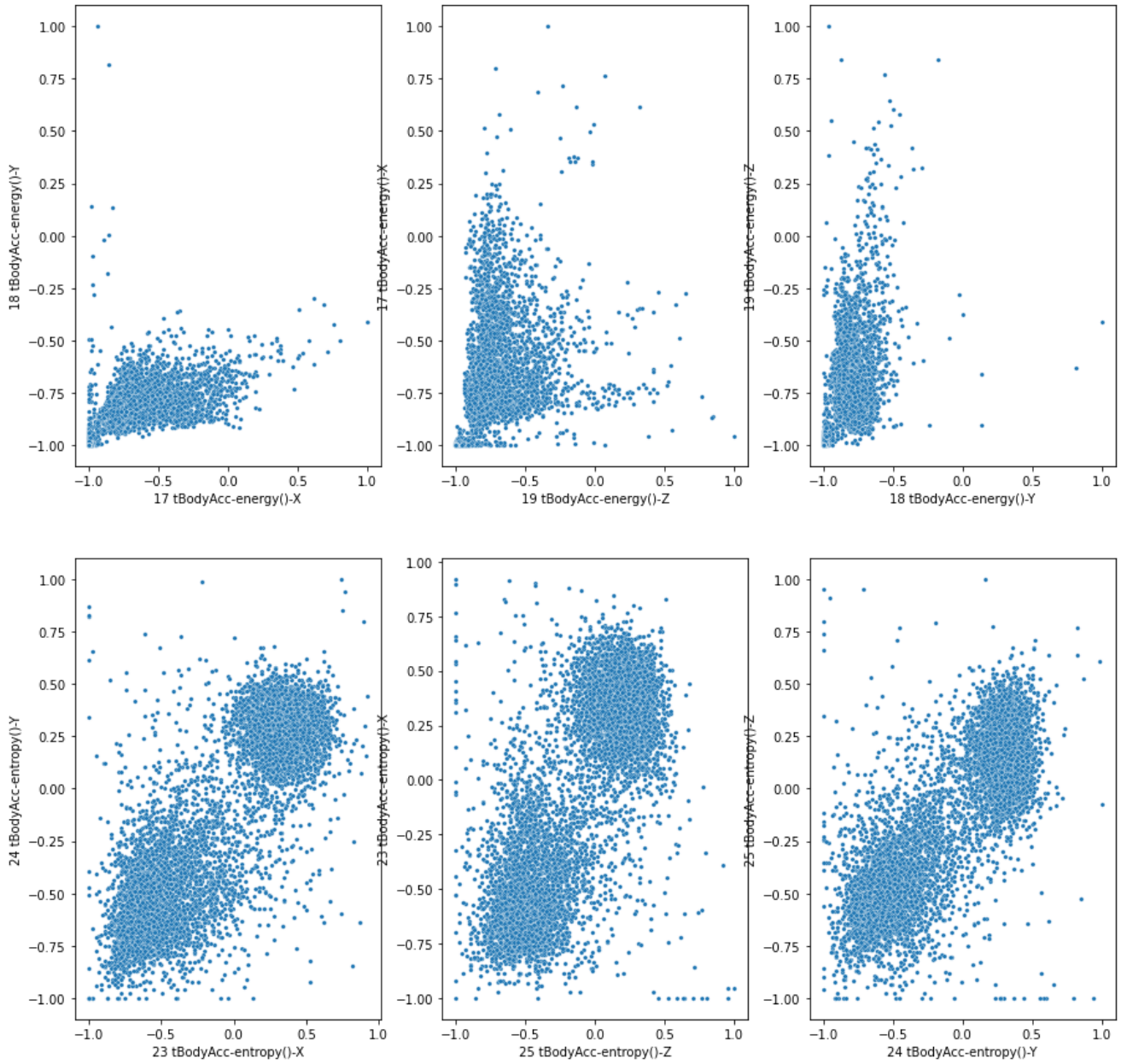
```
def scatter_plot(data, column1, column2):
    if type(column1) is int:
        column1 = data.columns[column1]
    if type(column2) is int:
        column2 = data.columns[column2]
    plt.figure(figsize=(10, 7))
    plt.scatter(x=data[column1], y=data[column2], s=12)
    plt.xlabel(column1)
    plt.ylabel(column2)
    plt.grid()
```

```python
#funkcja facet grid dla scatterów, musiałem też zmodyfikować funkcję do scatterów bo plt nie reaguje
def scatter_plot_2(data, column1, column2,**kwargs):
    if type(column1) is int:
        column1 = data.columns[column1]
    if type(column2) is int:
        column2 = data.columns[column2]
    sns.scatterplot(x=data[column1], y=data[column2], s=12,**kwargs)


def plot_scatter_facet(data, *which, cols=2): #which to tuple 2 miejscowa zawierająca index lub nazwę
    _fig, _axs = plt.subplots((len(which)+1)//cols, cols)

    for i, el in enumerate(which):
        scatter_plot_2(data, column1 = el[0],column2 = el[1],ax=_axs[i//cols, i%cols])
    return _fig
```

```python
#Przykład dla średnich → nic nam nie mówią, są wręcz wycentrowane w jednym klastrze
fig = plot_scatter_facet(X_train,(0,1),(2,0),(1,2),(3,4),(5,3),(4,5),cols=3)
fig.set_size_inches(15,15)
plt.show()
```



```
#Tutaj dla energy i entropy
fig = plot_scatter_facet(X_train,(16,17),(18,16),(17,18),(22,23),(24,22),(23,24),cols=3)
fig.set_size_inches(15,15)
plt.show()
```



Konkluzje, wyniki

Mamy do czynienia z ogromną ramką danych, które są między sobą dość silnie zależne, ma mapie korelacji mogliśmy zaobserwować, iż duża większość par miała współczynnik korelacji wyższy niż 0.9.
Analiza dystrybucji oraz scatter plotów wykazała istnienie penwych klastrów danych, dane formują się w pewne grupy, zatem nawet bez analizy labeli jesteśmy w stanie stwierdzić istnienie pewnego targetu.
Zmienna mean jest bardzo dobrze wycentrowana, a zatem nie mówi wnosi nam wiele informacji w naszej analizie.
Odchylenie standardowe natomiast dla poszczególnych zmiennych charakteryzuje się klasterowością.
Dane zostały poddane uprzedniej obróbce, każda dana jest z przedziału od -1 do 1 oraz nie występują ich braki, jest to bardzo użyteczna informacja do dalszej części projektu.