Будем работать с датасетом физической активности, которую собирает приложение о туристических маршрутах.

В этом датасете собраны данные 30 человек, выполняющих различные действия со смартфоном на поясе. Данные записывались с помощью датчиков (акселерометра и гироскопа) в этом смартфоне. Были зафиксированы: "3-осевое линейное ускорение" (tAcc-XYZ) и "3-осевая угловая скорость" (tGyro-XYZ).

Все сигналы отсортированы по времени, пропущены через фильтр шума. Все наблюдения были разделены на окна по 128 сигналов в каждом, причем окна пересекаются на 50% (грубо говоря, с 1 по 128 сигнал в 1 окно, с 64 до 192 сигнала - во 2 окно и так далее). Линейное ускорение было разделено на две составляющие (с помощью низкочастотного фильтра): ускорение самого тела, а также гравитационная составляющая. Будем называть их ускорением тела и гравитационным ускорением. К получившимся окнам применяли различные функции для получения следующих показателей: max min mad (median) mean std skewness (коэффициент асимметричности распределения признака в окне) etc. (подробнее, на сколько это возможно, указано на страничке датасета) Величина (magnitude) сигнала также определялась через евклидову норму всех значений из вектора для одного окна. К окнам было применено преобразование Фурье, чтобы получить частоты. Из частот были извелченыте же показатели, плюс новые: bandsEnergy (энергия частотного интервала) Были посчитаны углы между векторами соответствующих окон. Для ускорений были посчитаны производные - рывки, к которым применялись те же функции. Итог: в датасете каждый признак - это какая-то операция над вектором из 128 чисел, которые соответствуют замерам определенного человека занятого какой-то активностью на протяжении 2.56 секунд.

Как читать названия признаков? Примеры:

tBodyAccMag-mean() - это вектор ускорений тела (без гравитационной составляющей этого ускорения), из которого взяли евклидову норму, а затем усреднили, чтобы получить скаляр fBodyAcc-bandsEnergy()-1,8.2 - это вектор ускорения тела, к которому применили преобразование Фурье, преобразовав его в вектор частот, а затем посчитали на этом векторе энергию данного частотного интервала в определенном диапазоне частот (в данном случае 1 - 8.2) angle(X,gravityMean) - это угол между осью X и вектором усредненных значений гравитационного ускорения из соответствующего окна Вероятно, сначала покажется сложным, но мы разберемся с некоторыми признаками в отдельности и вы сможете понять их смысл, уже исходя из фактических соотношений.

Задание №1. Проведите предобработку (удаление дубликатов, удаление/замена пропущенных данных, удаление выбросов, замена типа) и анализ данных (визуализация, определение корреляций, построение корреляционной матрицы, группирование и агрегирование, оценка существования зависимостей между переменными).

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5
6 import warnings
7 warnings.filterwarnings("ignore")
8
```

```
1 train = pd.read_csv("train.csv")
2 test = pd.read_csv("test.csv")
```

```
1 train.head()
```

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBody
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.98
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.97
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.96
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.98
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.97

5 rows × 563 columns

```
1 train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7352 entries, 0 to 7351
Columns: 563 entries, tBodyAcc-mean()-X to Activity
dtypes: float64(561), int64(1), object(1)
memory usage: 31.6+ MB
```

Т.к в датасете 563 признака, а нам столько для демонтрации знаний основ машинного обучения не нужно, я выбрал наиболее интересные руками.

```
cols_subset = [
    'tBodyAccMag-mean()',  # усредненная величина ускорения тела
    'tBodyGyroJerk-mad()-X',  # медианная величина рывка тела по оси X
    'tGravityAcc-min()-X',  # минимум гравитационной составляющей ускорения по оси X
    'tBodyAcc-max()-X',  # максимальная величина ускорения тела по оси X
    'fBodyAcc-bandsEnergy()-1,8.2',  # энергия ускорения тела в интервале частоты
    'angle(X,gravityMean)',  # угол между осью X и усредненной гравитационной составляющей ускорения
    'angle(Y,gravityMean)',  # угол между осью Y и усредненной гравитационной составляющей ускорения
    'angle(Z,gravityMean)',  # угол между осью Z и усредненной гравитационной составляющей ускорения,
    'fBodyAcc-skewness()-X',  # асимметричность частоты ускорения тела по оси X
    'subject',  # номер испытуемого
    'Activity',  # название вида деятельности (целевая переменная)
]
```

В задаче кластеризации (обучение без учителя) мы будем разбивать на кластеры по Activity, а в задаче регрессии я постараюсь предсказать максимальную величину ускорения тела по оси X, возможно мы будем использовать эту модель для заполнения пропусков в случае какой либо ошибки в дальнейшем в нашем проекте по туристическим маршрутам, думаю это будет хоть сколько то полезно.

```
train = train[cols_subset]
test = test[cols_subset]
```

Проведем предобработку и анализ и визуализацию данных для выполнения пунка номер 1.

```
# Посмотрим на пропуски
train.isnull().sum()
```

```
tBodyAccMag-mean()              0
tBodyGyroJerk-mad()-X           0
tGravityAcc-min()-X             0
tBodyAcc-max()-X                0
fBodyAcc-bandsEnergy()-1,8.2    0
angle(X,gravityMean)            0
angle(Y,gravityMean)            0
angle(Z,gravityMean)            0
fBodyAcc-skewness()-X           0
subject                         0
Activity                        0
dtype: int64
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7352 entries, 0 to 7351
Data columns (total 11 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   tBodyAccMag-mean()             7352 non-null   float64
 1   tBodyGyroJerk-mad()-X          7352 non-null   float64
 2   tGravityAcc-min()-X            7352 non-null   float64
 3   tBodyAcc-max()-X               7352 non-null   float64
 4   fBodyAcc-bandsEnergy()-1,8.2   7352 non-null   float64
 5   angle(X,gravityMean)           7352 non-null   float64
 6   angle(Y,gravityMean)           7352 non-null   float64
 7   angle(Z,gravityMean)           7352 non-null   float64
 8   fBodyAcc-skewness()-X          7352 non-null   float64
 9   subject                        7352 non-null   int64
 10  Activity                       7352 non-null   object
dtypes: float64(9), int64(1), object(1)
memory usage: 631.9+ KB
```

их нету

```
train[train.duplicated()].sum().sum() # посмотрим количество дублей
```

```
0.0
```

их нету

```
# проведем те же процедуры для датасета test и train
test.isnull().sum().sum()
test[test.duplicated()].sum().sum()
```

```
0.0
```

посмотрим основные характеристики датасета

```
1 train.describe()
```

	tBodyAccMag-mean()	tBodyGyroJerk-mad()-X	tGravityAcc-min()-X	tBodyAcc-max()-X	fBodyAcc-bandsEnergy()-1,8.2
count	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000
mean	-0.543884	-0.727735	0.678505	-0.468604	-0.847963
std	0.477653	0.315832	0.508656	0.544547	0.233633
min	-1.000000	-0.999889	-1.000000	-1.000000	-1.000000
25%	-0.983282	-0.991595	0.804270	-0.936219	-0.999480
50%	-0.883371	-0.950268	0.926693	-0.881637	-0.986582
75%	-0.106892	-0.474625	0.965095	-0.017129	-0.764508
max	1.000000	1.000000	1.000000	1.000000	1.000000

```
1 train.median()
```

```
<ipython-input-16-4f4a9c1154f5>:1: FutureWarning: The default value of numeric_only in DataFrame.median is deprecated. In a future v
  train.median()
tBodyAccMag-mean()              -0.883371
tBodyGyroJerk-mad()-X           -0.950268
tGravityAcc-min()-X              0.926693
tBodyAcc-max()-X                -0.881637
fBodyAcc-bandsEnergy()-1,8.2    -0.986582
angle(X,gravityMean)            -0.709417
angle(Y,gravityMean)             0.182071
angle(Z,gravityMean)             0.003181
fBodyAcc-skewness()-X           -0.163271
subject                         19.000000
dtype: float64
```

```
1 test.describe()
```

	tBodyAccMag-mean()	tBodyGyroJerk-mad()-X	tGravityAcc-min()-X	tBodyAcc-max()-X	fBodyAcc-bandsEnergy()-1,8.2
count	2947.000000	2947.000000	2947.000000	2947.000000	2947.000000
mean	-0.559043	-0.735661	0.696878	-0.462063	-0.880361
std	0.439539	0.282074	0.502873	0.523916	0.160496
min	-0.998936	-1.000000	-0.756774	-0.952357	-0.999989
25%	-0.978433	-0.989078	0.836787	-0.934447	-0.999330
50%	-0.861830	-0.914698	0.934930	-0.852659	-0.979336
75%	-0.149596	-0.486423	0.970871	-0.009965	-0.799113
max	0.479547	0.117606	0.996148	0.786436	0.048730

```
1 test.median()
```

```
<ipython-input-18-b51c85c92acd>:1: FutureWarning: The default value of numeric_only in DataFrame.median is deprecated. In a future v
  test.median()
tBodyAccMag-mean()              -0.861830
tBodyGyroJerk-mad()-X           -0.914698
tGravityAcc-min()-X              0.934930
tBodyAcc-max()-X                -0.852659
fBodyAcc-bandsEnergy()-1,8.2    -0.979336
angle(X,gravityMean)            -0.729648
angle(Y,gravityMean)             0.181563
angle(Z,gravityMean)            -0.010671
fBodyAcc-skewness()-X           -0.202050
subject                         12.000000
dtype: float64
```

из интересного - есть отрицательные числа.

Проверим данные на сбалансированность

```
1 train["Activity"].value_counts()
```

```
LAYING                1407
STANDING              1374
SITTING               1286
WALKING               1226
WALKING_UPSTAIRS      1073
WALKING_DOWNSTAIRS     986
Name: Activity, dtype: int64
```

```
1 maxx = train["Activity"].value_counts().max()
2 minn = train["Activity"].value_counts().min()
3 100 - (minn*100)/maxx
```

```
29.92181947405828
```

т к < чем 30% -> данные в train сбалансированы

```
1 test["Activity"].value_counts()
```

```
LAYING                537
STANDING              532
WALKING               496
SITTING               491
WALKING_UPSTAIRS      471
WALKING_DOWNSTAIRS    420
Name: Activity, dtype: int64
```

```
1
2 maxx = test["Activity"].value_counts().max()
3 minn = test["Activity"].value_counts().min()
4 100 - (minn*100)/maxx
```

```
21.787709497206706
```

Посчитаем корреляцию всех признаков:

```
1 cors_train = train
2 dat = ["STANDING", "LAYING", "WALKING", "SITTING", "WALKING_DOWNSTAIRS", "WALKING_UPSTAIRS"]
3 for t in dat:
4   cors_train[str(t)] = 0
5   cors_train.loc[cors_train["Activity"] == str(t), str(t)] = 1
6
7 cors_train.corr()
```

```
<ipython-input-24-88042b5051bb>:7: FutureWarning: The default value of numeric_only i
  cors_train.corr()
```

	tBodyAccMag-mean()	tBodyGyroJerk-mad()-X	tGravityAcc-min()-X	tBodyAcc-max()-X	bandsEne
tBodyAccMag-mean()	1.000000	0.899080	0.369962	0.959827	
tBodyGyroJerk-mad()-X	0.899080	1.000000	0.380118	0.879900	
tGravityAcc-min()-X	0.369962	0.380118	1.000000	0.380404	
tBodyAcc-max()-X	0.959827	0.879900	0.380404	1.000000	
fBodyAcc-bandsEnergy()-1,8.2	0.697832	0.507939	0.239088	0.613383	
angle(X,gravityMean)	-0.370849	-0.382246	-0.988663	-0.384192	
angle(Y,gravityMean)	0.495561	0.478567	0.797116	0.480229	
angle(Z,gravityMean)	0.426283	0.380967	0.667415	0.405023	
fBodyAcc-skewness()-X	0.379796	0.266002	-0.261676	0.335708	
subject	-0.073652	-0.110321	-0.042371	-0.055633	
STANDING	-0.414460	-0.370699	0.262354	-0.399861	
LAYING	-0.396643	-0.386730	-0.973167	-0.394745	
WALKING	0.358825	0.531695	0.234820	0.298526	
SITTING	-0.396389	-0.379403	0.187055	-0.387297	
WALKING_DOWNSTAIRS	0.561344	0.445048	0.192409	0.658369	
WALKING_UPSTAIRS	0.405434	0.257524	0.159741	0.347429	

```
1 train.corr()
2
```

```
<ipython-input-25-f4d37163d5c7>:1: FutureWarning: The default value of numeric_only i
  train.corr()
```

	tBodyAccMag-mean()	tBodyGyroJerk-mad()-X	tGravityAcc-min()-X	tBodyAcc-max()-X	bandsEne
tBodyAccMag-mean()	1.000000	0.899080	0.369962	0.959827	
tBodyGyroJerk-mad()-X	0.899080	1.000000	0.380118	0.879900	
tGravityAcc-min()-X	0.369962	0.380118	1.000000	0.380404	
tBodyAcc-max()-X	0.959827	0.879900	0.380404	1.000000	
fBodyAcc-bandsEnergy()-1,8.2	0.697832	0.507939	0.239088	0.613383	
angle(X,gravityMean)	-0.370849	-0.382246	-0.988663	-0.384192	
angle(Y,gravityMean)	0.495561	0.478567	0.797116	0.480229	
angle(Z,gravityMean)	0.426283	0.380967	0.667415	0.405023	
fBodyAcc-skewness()-X	0.379796	0.266002	-0.261676	0.335708	
subject	-0.073652	-0.110321	-0.042371	-0.055633	
STANDING	-0.414460	-0.370699	0.262354	-0.399861	
LAYING	-0.396643	-0.386730	-0.973167	-0.394745	
WALKING	0.358825	0.531695	0.234820	0.298526	
SITTING	-0.396389	-0.379403	0.187055	-0.387297	
WALKING_DOWNSTAIRS	0.561344	0.445048	0.192409	0.658369	
WALKING_UPSTAIRS	0.405434	0.257524	0.159741	0.347429	

```
1 test.corr()
```

```
<ipython-input-26-c06d3b70ad5e>:1: FutureWarning: The default value of numeric_only i
  test.corr()
```

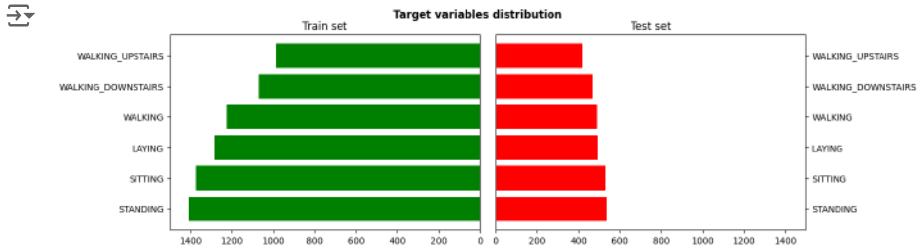
	tBodyAccMag-mean()	tBodyGyroJerk-mad()-X	tGravityAcc-min()-X	tBodyAcc-max()-X	bandsEnerg
tBodyAccMag-mean()	1.000000	0.934773	0.407084	0.965114	
tBodyGyroJerk-mad()-X	0.934773	1.000000	0.393773	0.898628	
tGravityAcc-min()-X	0.407084	0.393773	1.000000	0.386764	
tBodyAcc-max()-X	0.965114	0.898628	0.386764	1.000000	
fBodyAcc-bandsEnergy()-1,8.2	0.767875	0.625414	0.301643	0.707129	
angle(X,gravityMean)	-0.400871	-0.387274	-0.989043	-0.384454	
angle(Y,gravityMean)	0.439051	0.423288	0.697006	0.410477	
angle(Z,gravityMean)	0.425011	0.368411	0.633881	0.398008	
fBodyAcc-skewness()-X	0.365670	0.278765	-0.237864	0.338543	
subject	-0.057359	-0.079811	0.004927	-0.071786	

Коэффициенты корреляции говорят о том, что зависимость между признаками есть, или она отрицательная или положительная. Однако, есть и коэффициенты, которые стремятся к 0, значит есть признаки, зависимости между которыми зависимости никакой нет

Теперь перейдем к визуализации данных и построении графиков.

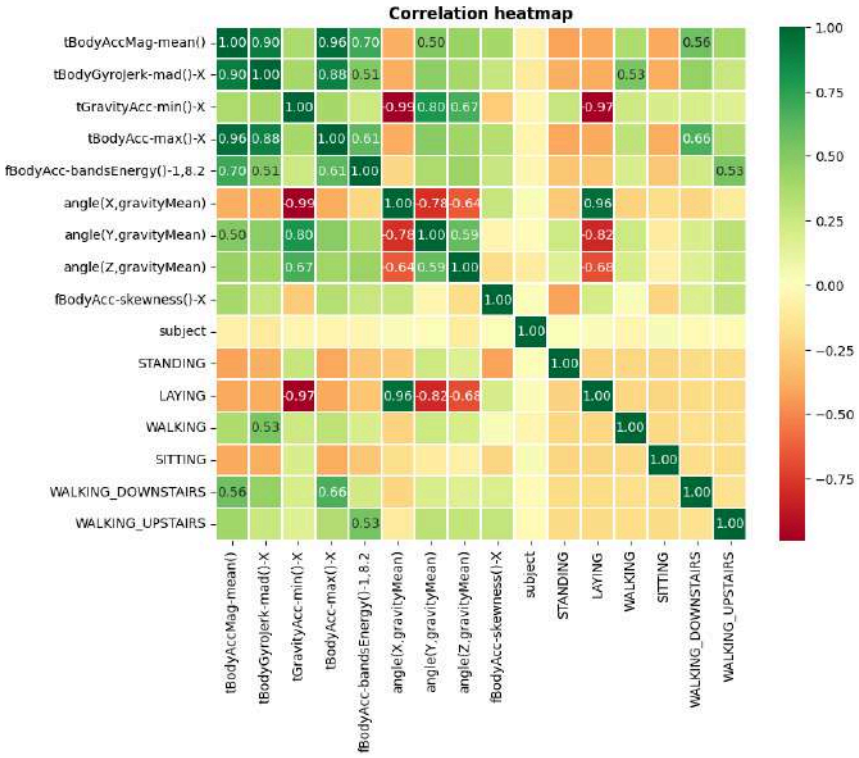
Начнем с наглядного графика, показывающего , что наши данные сбалансированы

```
# для test
ind_test = test["Activity"].unique()
ind2_test = np.array(test["Activity"].value_counts())
#ind_test = np.delete(ind_test, 6)

#Для train
ind_train = train["Activity"].unique()
ind2_train = np.array(train["Activity"].value_counts())
#ind_train = np.delete(ind_train, 6)

# Построение графиков
fig, axs = plt.subplots(nrows = 1, ncols = 2, figsize = (13, 4))

# Строим гистограммы для train и set
axs[0].barh(ind_train, ind2_train, label = "Train set", color = "green")
axs[1].barh(ind_test, ind2_test, label = "Test set", color = "red")

# Устанавливаем лимиты на оси X, чтобы цена деления графиков была одинаковая
axs[1].set_xlim(0, 1500)
axs[0].set_xlim(0, 1500)

# Отзеркаливает первый график, чтобы оси у у обоих графиков стояли рядом
axs[0].invert_xaxis()

axs[0].set_title('Train set')
axs[1].set_title('Test set')

ax = plt.gca() # Получаем текущие оси
#axs[1].yaxis.set_label_position("right")

axs[1].yaxis.tick_right() # Перемещает метки оси второго графика вправо
#plt.xlim(0, 1400)
#ax.spines['left'].set_position('center')
fig.suptitle('Target variables distribution', fontweight='bold')

plt.subplots_adjust(wspace=0.05) # Меняем расстояние между графиками

plt.show()

```



Визуализируем матрицу корреляций для наглядности и чтобы показать, что мы умеем это делать.

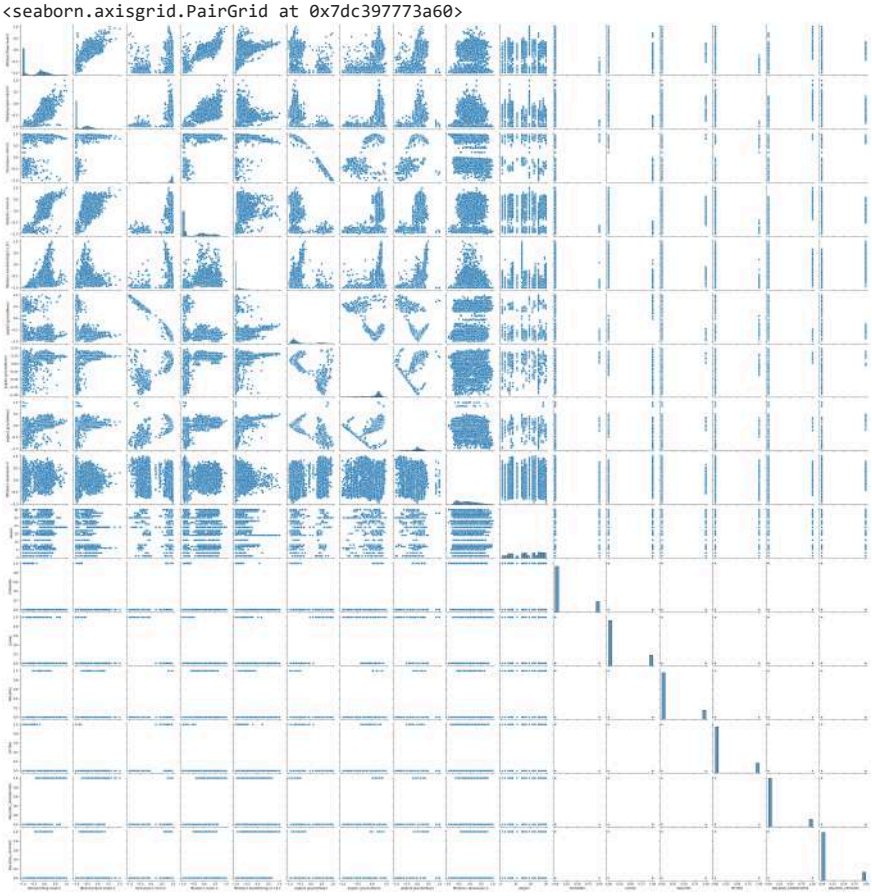
```
import math
cors_train = train
dat = ["STANDING", "LAYING", "WALKING", "SITTING", "WALKING_DOWNSTAIRS", "WALKING_UPSTAIRS"]
for t in dat:
  cors_train[str(t)] = 0
  cors_train.loc[cors_train["Activity"] == str(t), str(t)] = 1

fig, ax = plt.subplots(figsize = (9, 7))
ax = sns.heatmap(cors_train.corr(), cmap ="RdYlGn" , annot = True, fmt = "0.2f", linewidth = .3)
#plt.figure(figsize = (12, 8))
plt.title('Correlation heatmap', fontweight = "bold")
for t in ax.texts:
  if float(t.get_text())>=0.5 or float(t.get_text())<=-0.5:
    t.set_text(t.get_text())
  else:
    t.set_text("")

```



Визуализируем вообще весь train с помощью пэйрплота и выведем все графики разбросы

```
1 sns.pairplot(train)
```

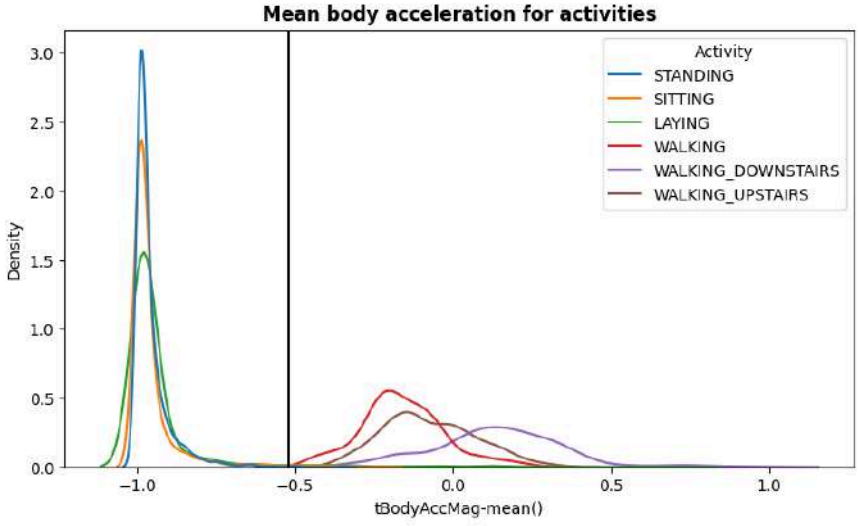
```
<seaborn.axisgrid.PairGrid at 0x7dc397773a60>
```



особо ничего это нам не дает, так как ничего не понятно и не особо видно, но для галочки - сделали. Теперь давайте попробуем отобразить плотность распределения tBodyAccMag-mean() для каждой из активностей и попробуем разделить линией две группы (активных и неактивных)

```
1 fig, ax = plt.subplots(figsize =(9, 5))
2 sns.kdeplot(train, x='tBodyAccMag-mean()', hue='Activity')
3 plt.axvline(x = -0.52, color = "black")
4 plt.title("Mean body acceleration for activities", fontweight = "bold")
```

```
Text(0.5, 1.0, 'Mean body acceleration for activities')
```

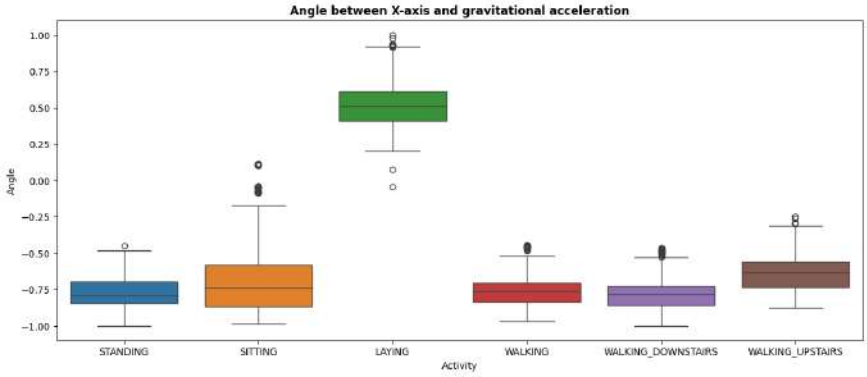


Как видим - наша гипотеза про ускорение и вид деятельности подтвердилась графиком.

Попробуем порисовать ящики с усами по фиче с названием angle(x,gravityMean) с разделением на разные виды активности.

```
fig, ax = plt.subplots(figsize = (15, 6))
sns.boxplot(train, y = "angle(X,gravityMean)", x = "Activity", hue = "Activity")
plt.ylabel("Angle")
plt.title("Angle between X-axis and gravitational acceleration", fontweight = "bold")
```

```
Text(0.5, 1.0, 'Angle between X-axis and gravitational acceleration')
```



Круто! Можем сделать еще один вывод о том, что когда мы лежим - угол между осью X и усредненной графитационной составляющей равен положительному значению - это круто и логично.

Отлично! Я показал свои способности в выдвижении гипотез, рисовании графиков , предобработке данных - теперь перейдем непосредственно к задачам машинного обучения.

Задание №2. Выберите целевую переменную и решите задачу регрессии, рассмотрев минимум 5 моделей и определив минимум 3 метрики работы моделей. Выберите лучшую модель. Для лучшей модели определите значимость признаков. Отбросив половину худших признаков, определите метрики работы модели без этих признаков. Подберите оптимальные параметры лучшей модели и определите метрики работы модели при этих значениях. Оцените качество работы лучшей моделей на основе кросс-валидации. Постройте кривые обучения (потерь) и оцените недообучение и переобучение.

Рассмотрим 5 моделей:DecisionTreeRegressor,MLPRegressor, SVR, RandomForestRegressor,LinearRegression. В возьмем 3-4 метрики ,такие как на примере mae,mse,r2. За целевую переменную возьмем например tBodyAcc-max()-X

```
y_train = train["tBodyAcc-max()-X"]
y_test = test["tBodyAcc-max()-X"]
```

```
X_train = train[["tBodyAccMag-mean()","tBodyGyroJerk-mad()-X","tGravityAcc-min()-X","fBodyAcc-bandsEnergy()-1,8.2","angle(X,gravityMean)","angl
X_test = test[["tBodyAccMag-mean()","tBodyGyroJerk-mad()-X","tGravityAcc-min()-X","fBodyAcc-bandsEnergy()-1,8.2","angle(X,gravityMean)","angle(
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.tree import DecisionTreeRegressor
```

```python
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(),
    "Support Vector Machine": SVR(),
    "Neural Network": MLPRegressor(),
    "Decision Tree": DecisionTreeRegressor()
}

for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Вычисление метрик
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Вывод результатов
    print(f"Metrics for {model_name}:")
    print(f"Mean Squared Error: {mse}")
    print(f"Mean Absolute Error: {mae}")
    print(f"R-squared: {r2}\n")
```

```
Metrics for Linear Regression:
Mean Squared Error: 0.01880522857894746
Mean Absolute Error: 0.08991588550570391
R-squared: 0.9314665647520397

Metrics for Random Forest:
Mean Squared Error: 0.018524369269894717
Mean Absolute Error: 0.08085463203919903
R-squared: 0.932490123343202

Metrics for Support Vector Machine:
Mean Squared Error: 0.016543034003709982
Mean Absolute Error: 0.0907217802175504
R-squared: 0.9397108657872256

Metrics for Neural Network:
Mean Squared Error: 0.015428155178408362
Mean Absolute Error: 0.07662513287241064
R-squared: 0.9437739100337958

Metrics for Decision Tree:
Mean Squared Error: 0.04206732567112615
Mean Absolute Error: 0.1167520211712148
R-squared: 0.8466905984240716
```

Лучшей моделью стала нейронка

```python
model = MLPRegressor()
model.fit(X_train, y_train)
```

```
▾ MLPRegressor
MLPRegressor()
```

```python
print("Weights of the coefficients for MLPRegressor:")
for i, coef in enumerate(model.coefs_):
    print(f"Layer {i} weights shape: {coef.shape}")
    print(coef)
```

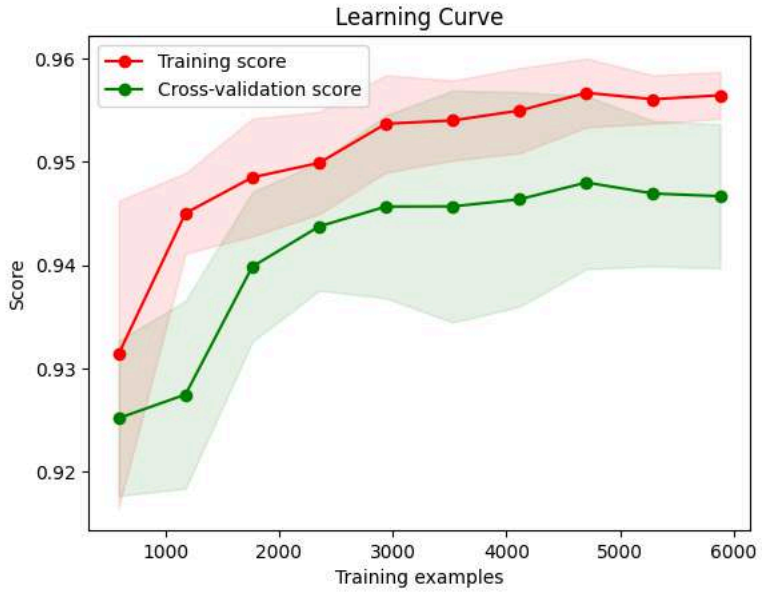
```
Weights of the coefficients for MLPRegressor:
Layer 0 weights shape: (8, 100)
[[ 1.95351408e-01  2.69804643e-02 -2.48136103e-14 -2.01419517e-01
  -1.08211928e-01 -1.36953382e-01  3.34078281e-01  7.52035062e-02
  -2.37241463e-02 -8.26752194e-02  1.51132761e-01  1.13677296e-01
   1.03725524e-01  2.89674438e-01  4.26583848e-01 -1.14559593e-01
  -6.61586543e-02  2.19097457e-01  1.57757326e-01  8.01908644e-02
  -1.06001946e-01  8.28741327e-02  4.02514938e-02  1.07755892e-01
   1.86464207e-02 -2.04790538e-01  2.35169356e-01 -1.93385770e-01
  -9.26135951e-02  6.03507427e-01  9.80558940e-02  1.31612436e-01
  -1.60803409e-02 -3.23521593e-02 -2.17907950e-01  3.13933403e-01
   5.09578330e-01  2.54644271e-02 -2.41778781e-01  3.14530310e-02
   4.37377576e-01 -2.25630452e-01 -7.48220988e-02  1.17550120e-01
  -7.55380242e-02  3.53160278e-01  1.70728323e-07  1.84097286e-01
  -7.94225765e-02 -1.98305676e-01 -2.67973988e-01 -4.29942078e-03
   6.24708914e-02 -7.71236660e-02  1.44183204e-01 -9.08377606e-02
  -6.01784905e-02 -1.48475227e-01 -1.67637524e-03  1.64048203e-01
   6.56285699e-02  1.62947662e-01  6.46235043e-03  2.05349187e-01
  -2.64954921e-01 -1.35642680e-01  1.85579145e-01 -1.54857655e-01
  -1.10470899e-02  7.23951862e-02 -2.72646007e-01 -7.62548009e-02
  -2.47033545e-01  1.36400619e-01 -2.21502156e-01 -2.55494826e-01
   3.97570667e-01  9.83208326e-02 -1.86150141e-01  3.19975560e-02
   2.27339896e-01 -1.33415855e-01 -9.38308804e-02  1.29815490e-01
```

```
 -2.19828171e-01  4.57026747e-01  3.13343232e-01  6.38754258e-11
  4.75666176e-02 -6.88724262e-02 -3.07159900e-01  2.87464178e-01
 -7.46148844e-03  2.91261003e-01  1.19874090e-01  2.78411379e-01
  7.19757059e-02  2.66447596e-01 -1.12806051e-01  1.54069817e-01]
[-1.47395546e-01  7.67032572e-02  2.53280785e-09  9.64032963e-02
  4.74148720e-02  1.70402447e-01  8.18970100e-02 -1.64131356e-01
 -1.21732597e-01 -2.46221406e-01 -1.99936314e-01 -7.30255379e-02
 -1.89902762e-01  1.67436828e-01  1.52984241e-01 -2.23895878e-01
  6.89185305e-02 -1.52742530e-01 -1.38021528e-01 -1.80237598e-01
  2.23023954e-01 -1.16980292e-01  2.06254160e-01 -2.22109609e-01
 -1.32845480e-01  2.45195182e-01  1.25228929e-02  1.23978738e-01
  1.25236997e-02 -2.19041626e-02  1.01594473e-02 -3.00265377e-01
  5.80899228e-03 -1.08605412e-01  2.11447035e-01  3.80985397e-02
  2.60767502e-02  1.87963724e-01  9.23684050e-02  1.71789849e-01
 -1.80101438e-01 -4.44856392e-02 -2.57752898e-01 -1.58756298e-01
 -4.99587128e-02 -6.81866961e-02  3.27436281e-07  1.82756582e-02
  4.02391432e-01  5.83892055e-03  1.68511168e-01 -1.20386752e-01
 -1.07864170e-01 -1.64538914e-01  1.42704034e-01  1.22309497e-01
  2.04166530e-01 -4.85044872e-02  9.43321453e-02  1.05651671e-01
  2.03188727e-01  3.46789417e-02 -1.75708064e-01 -1.90825980e-01
 -2.77619657e-01  1.51868573e-01  1.48175437e-01 -2.16874963e-01
  5.14209673e-02 -8.78099257e-02 -1.64242217e-02  2.68273712e-02
  1.75679157e-01  4.98482869e-03  7.66991288e-02  8.39472410e-02
  1.88885607e-02  3.06989806e-02  1.09857904e-01 -2.33909930e-02
  8.96874216e-02 -2.03959599e-01  3.24895029e-02  1.37477375e-01
 -2.36758379e-01  7.82094567e-02  1.57804100e-03  3.11398143e-06
 -9.12894639e-02 -5.10499826e-02  1.13586962e-01  7.26237945e-03
 -4.15361808e-02  9.20796220e-02  2.25229878e-01  7.58351604e-02
 -2.04018940e-01  5.66274509e-02 -1.01325228e-01  9.41963744e-02]
[-9.12511423e-02 -6.13958425e-02  3.95979830e-08 -2.34374572e-01
 -1.77848799e-01 -1.90643286e-01  8.28760991e-02 -8.24239573e-02
  2.21732938e-01  7.88327939e-02  6.14866032e-02 -1.00678252e-01
  1.27423870e-01  6.41768011e-02  2.77491482e-02 -8.43585337e-02
  2.25559663e-01 -9.98519043e-03  1.59325221e-01 -2.15224258e-01
  8.60296835e-02 -2.04481423e-01 -1.45144675e-01  4.12797269e-02
```

К сожалению мы не можем вывести веса каждой фичи если это нейронка, так как нейронка стала нашей лучшей моделью машинного обучения, поэтому выше я вывел просто веса коэфов для каждого слоя нейронной сети.

Далее построим кривую обучения и оценим переобучение.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve, cross_val_score

# Предположим, что у вас уже есть обученная модель MLPRegressor с именем model
model = MLPRegressor()

# Оценка обучения и переобучения с помощью кросс-валидации
train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_train, train_sizes=np.linspace(0.1, 1.0, 10), cv=5)

# Вычисление средних значений и стандартных отклонений для обучающих и тестовых оценок
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

# Построение кривой обучения
plt.figure()
plt.title('Learning Curve')
plt.xlabel("Training examples")
plt.ylabel("Score")
plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")
plt.legend(loc="best")
plt.show()
```



Как видим - модель обучена хорошо

Задание 3. Выберите целевую переменную и решите задачу классификации, рассмотрев минимум 5 моделей и определив 4 метрики работы моделей. Выберите лучшую модель. Подберите оптимальные параметры лучшей модели и определите метрики работы модели при этих значениях.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

```
X_train = train[["tBodyAcc-max()-X","tBodyAccMag-mean()","tBodyGyroJerk-mad()-X","tGravityAcc-min()-X","fBodyAcc-bandsEnergy()-1,8.2","angle(X,
X_test = test[["tBodyAcc-max()-X","tBodyAccMag-mean()","tBodyGyroJerk-mad()-X","tGravityAcc-min()-X","fBodyAcc-bandsEnergy()-1,8.2","angle(X,gr
```

```
y_train = train["Activity"]
y_test = test["Activity"]
```

```
models = {
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Logistic Regression': LogisticRegression(),
    'SVM': SVC(),
    'KNN': KNeighborsClassifier()
}

for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"Модель: {model_name} - Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1 Score: {f1}")
```

```
Модель: Decision Tree - Accuracy: 0.7560230743128605, Precision: 0.7595474735966729, Recall: 0.7560230743128605, F1 Score: 0.755484
Модель: Random Forest - Accuracy: 0.7967424499491008, Precision: 0.8016304324235994, Recall: 0.7967424499491008, F1 Score: 0.797007
Модель: Logistic Regression - Accuracy: 0.837122497455039, Precision: 0.8391760035078879, Recall: 0.837122497455039, F1 Score: 0.83
Модель: SVM - Accuracy: 0.846284356973193, Precision: 0.8489661767058383, Recall: 0.846284356973193, F1 Score: 0.8458173304484603
Модель: KNN - Accuracy: 0.832711231761113, Precision: 0.836370556401396, Recall: 0.832711231761113, F1 Score: 0.8327381652501196
```

Как видим - лучшая модель SVM , подберем оптимальные параметры с помощью GridSearch

```
from sklearn.model_selection import GridSearchCV

# Определение набора параметров, которые вы хотите оптимизировать
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'linear']}

# Создание модели SVM
svm = SVC()

# Создание объекта GridSearchCV для настройки параметров с помощью перекрестной проверки
grid_search = GridSearchCV(svm, param_grid, cv=5)

# Подгонка модели к данным
grid_search.fit(X_train, y_train)

# Вывод лучших параметров
print(grid_search.best_params_)
```

```
{'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
```

```
model = SVC(C = 100, gamma = 0.1, kernel = 'rbf')
model.fit(X_train, y_train)
print(f"Модель: {model_name} - Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1 Score: {f1}")
```

```
Модель: KNN - Accuracy: 0.832711231761113, Precision: 0.836370556401396, Recall: 0.832711231761113, F1 Score: 0.8327381652501196
```

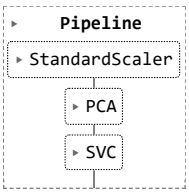
Как видим - стоковые гиперпараметры - самые лучшие.

Задание 4. Решите задачу уменьшения размерности (неконтролируемое обучение). Используя модели из п.2 или п.3, оцените как уменьшение размерности повлияло на метрики работы модели.

Теперь решим задачу уменьшения размерности, хотя я бы этого не делал, но для того чтобы показать что я это умею - сделаю.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
```

```
pca = PCA(n_components=6)  # Оставим тут 6 компонент

# Создание объекта StandardScaler (стандартизация)
scaler = StandardScaler()

# Создание модели SVC с подобранными параметрами
svc = SVC(C=100, gamma=0.1, kernel='rbf')

# Создание конвейера, который включает стандартизацию, уменьшение размерности и классификацию
model = make_pipeline(scaler, pca, svc)

# Обучение модели на обучающем наборе данных
model.fit(X_train, y_train)
```




```
y_pred = model.predict(X_test)

# Вычисление метрик качества
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Вывод метрик
print(f"Метрики качества:")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
Метрики качества:
Accuracy: 0.7678995588734306
Precision: 0.7724329317457954
Recall: 0.7678995588734306
F1 Score: 0.767617066847966
```

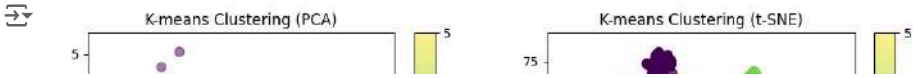
Как видим , из за того что мы лишились некоторых фиче, метрики упали - это нормально и этого стоило ожидать. Для пункта 2 будет аналогичная картина, так как , когда фичей всего 8-9 при уменьшения размерности до 6 этого стоит ожидать.

Задание 5. Решите задачу кластеризации, используя минимум 3 модели. Определите метрики кластеризации.

```
X_activity = X_train
```

```
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.preprocessing import StandardScaler

# Предположим, что у вас есть матрица признаков X_activity для переменной Activity

# Стандартизация данных
scaler = StandardScaler()
X_activity_scaled = scaler.fit_transform(X_activity)

# Кластеризация с помощью K-means
kmeans = KMeans(n_clusters=6, random_state=42)
kmeans_labels = kmeans.fit_predict(X_activity_scaled)

# Кластеризация с помощью DBSCAN
dbscan = DBSCAN(eps=1, min_samples=5)
dbscan_labels = dbscan.fit_predict(X_activity_scaled)

# Кластеризация с помощью Agglomerative Clustering
agg_clustering = AgglomerativeClustering(n_clusters=6)
agg_labels = agg_clustering.fit_predict(X_activity_scaled)
```

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Предположим, что у вас уже есть метки кластеров kmeans_labels, dbscan_labels и agg_labels

# Применение метода PCA для понижения размерности до 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_activity_scaled)

# Применение метода t-SNE для понижения размерности до 2D
tsne = TSNE(n_components=2)
X_tsne = tsne.fit_transform(X_activity_scaled)

# Визуализация кластеризации для K-means
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis', s=50, alpha=0.5)
plt.title('K-means Clustering (PCA)')
plt.colorbar()

plt.subplot(1, 2, 2)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=kmeans_labels, cmap='viridis', s=50, alpha=0.5)
plt.title('K-means Clustering (t-SNE)')
plt.colorbar()
plt.show()
```



Как то вот так ,выведем метрики...


```
from sklearn import metrics

# Для K-means
silhouette_kmeans = metrics.silhouette_score(X_activity_scaled, kmeans_labels)
calinski_harabasz_kmeans = metrics.calinski_harabasz_score(X_activity_scaled, kmeans_labels)
davies_bouldin_kmeans = metrics.davies_bouldin_score(X_activity_scaled, kmeans_labels)

# Для DBSCAN
silhouette_dbscan = metrics.silhouette_score(X_activity_scaled, dbscan_labels)
calinski_harabasz_dbscan = metrics.calinski_harabasz_score(X_activity_scaled, dbscan_labels)
davies_bouldin_dbscan = metrics.davies_bouldin_score(X_activity_scaled, dbscan_labels)

# Для Agglomerative Clustering
silhouette_agg = metrics.silhouette_score(X_activity_scaled, agg_labels)
calinski_harabasz_agg = metrics.calinski_harabasz_score(X_activity_scaled, agg_labels)
davies_bouldin_agg = metrics.davies_bouldin_score(X_activity_scaled, agg_labels)

# Вывод метрик
print("Silhouette Score:")
print(f"K-means: {silhouette_kmeans}, DBSCAN: {silhouette_dbscan}, Agglomerative Clustering: {silhouette_agg}")
print("\nCalinski Harabasz Score:")
print(f"K-means: {calinski_harabasz_kmeans}, DBSCAN: {calinski_harabasz_dbscan}, Agglomerative Clustering: {calinski_harabasz_agg}")
print("\nDavies Bouldin Score:")
print(f"K-means: {davies_bouldin_kmeans}, DBSCAN: {davies_bouldin_dbscan}, Agglomerative Clustering: {davies_bouldin_agg}")
```

```
Silhouette Score:
K-means: 0.4001276049335574, DBSCAN: 0.09950739854079525, Agglomerative Clustering: 0.3806965605480144

Calinski Harabasz Score:
K-means: 5881.281788302316, DBSCAN: 52.34594149746244, Agglomerative Clustering: 5490.4689927150175

Davies Bouldin Score:
K-means: 1.200587516028374, DBSCAN: 1.2546086948008657, Agglomerative Clustering: 1.2489069595912257
```

ИТОГ

В качестве потенциального внедрения в мой проект, я могу предложить внедрить данное проектное задание в приложение по составлению туристических маршрутов. Мы можем предсказывать на основе этих моделей сложность маршрута по ускорению человека или количеству зафиксированных активностей. Также мы можем предсказывать это и для разных групп людей, имеющих разный уровень подготовки. В приложении будет внедрён критерий эксперной оценки маршрута.