Департамент образования и науки города Москвы
Государственное автономное образовательное учреждение
высшего образования города Москвы
«Московский городской педагогический университет»
Институт цифрового образования
Департамент информатики управления и технологий

Инструменты хранения и анализа больших данных

Смоляков Руслан Игоревич БД-241м

Практическая работа 2.3. Анализ и визуализация больших данных. Машинное обучение на больших данных с использованием Apache Spark MLlib

Вариант 23

Направление подготовки/специальность
38.04.05 - Бизнес-информатика
Бизнес-аналитика и большие данные
(очная форма обучения)

Руководитель дисциплины:
Босенко Т.М., доцент департамента
информатики, управления и технологий,
кандидат технических наук

Москва
2025

Введение

Цель и задачи работы:

- **Познакомиться с понятием «большие данные» и способами их обработки;**
- **Познакомиться с инструментом Apache Spark и возможностями, которые он предоставляет для обработки больших данных.**
- **Получить навыки выполнения разведочного анализа данных использованием pyspark.**

Индивидуальные задания:

Вариант – 23

Задание 1 (Интерпретация) - Оцените названия столбцов в исходном датасете (раздел 2). Насколько они понятны для бизнес-пользователя? Есть ли столбцы, требующие переименования или дополнительного описания?

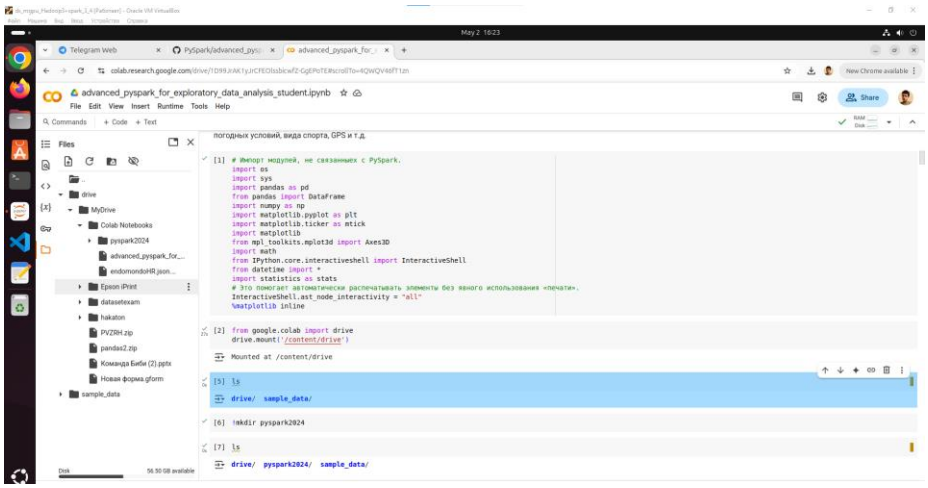
Задание 2 (Интерпретация) - В разделе 6 рассчитывается процент пользователей, занимающихся более чем 1 видом спорта. Как бы вы интерпретировали этот показатель для бизнеса? Это высокий или низкий процент?

Задание 3 (Интерпретация) - Если бы у вас были данные о погоде во время тренировки, как бы вы могли их использовать совместно с данными о duration или heart_rate?

Задание 4 (Практика PySpark/Python) - Напишите код PySpark, чтобы посчитать количество строк в датафрейме df до и после удаления дубликатов по столбцу id (уникальный ID тренировки).

Задание 5 (MLlib Концепция) - Почему удаление дубликатов важно перед обучением моделей ML в Spark MLlib? Как дубликаты могут повлиять на качество модели и оценку ее производительности?

Основная часть:
Работаю в google collab на ВМ



Скачал блокнот, импортировал все необходимые модули, дал доступ к своему диску, создал директорию pyspark2024.

```
[8] cd pyspark2024
/content/pyspark2024

!unzip "/content/drive/MyDrive/Colab Notebooks/endomondoHR.json.zip"
Archive:  /content/drive/MyDrive/Colab Notebooks/endomondoHR.json.zip
  inflating: endomondoHR.json

[10] ls
endomondoHR.json

[3] ! pip install pyspark
Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.5)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)
```

Перешел в новую директорию, разархивировал файл с данными, предварительно закинул его на диск в папку «Colab Notebooks»

```
# Импорт модулей, связанных с PySpark.
import pyspark
from pyspark.rdd import RDD
from pyspark.sql import Row
from pyspark.sql import DataFrame
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
from pyspark.sql import functions
from pyspark.sql.functions import lit, desc, col, size, array_contains\
, isnan, udf, hour, array_min, array_max, countDistinct
from pyspark.sql.types import *

MAX_MEMORY = '15G'
# Инициализировать сеанс Spark.
conf = pyspark.SparkConf().setMaster("local[*]") \
        .set('spark.executor.heartbeatInterval', 10000) \
        .set('spark.network.timeout', 10000) \
        .set("spark.core.connection.ack.wait.timeout", "3600") \
        .set("spark.executor.memory", MAX_MEMORY) \
        .set("spark.driver.memory", MAX_MEMORY)
def init_spark():
    spark = SparkSession \
        .builder \
        .appName("Pyspark guide") \
        .config(conf=conf) \
        .getOrCreate()
    return spark

spark = init_spark()
filename_data = 'endomondoHR.json' # Загрузите данные в текущий каталог Colab.
# Загрузите основной набор данных в фрейм данных pyspark.
df = spark.read.json(filename_data, mode="DROPMALFORMED")
print('Data frame type: ' + str(type(df)))

Data frame type: <class 'pyspark.sql.dataframe.DataFrame'>
```

Запустили pyspark.

Приступаем к обзору набора данных:

```
print('Обзор данных')
df.printSchema()
print('Обзор столбцов')
pd.DataFrame(df.dtypes, columns = ['Column Name','Data type'])
```

```
Обзор данных
root
 |-- altitude: array (nullable = true)
 |    |-- element: double (containsNull = true)
 |-- gender: string (nullable = true)
 |-- heart_rate: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- id: long (nullable = true)
 |-- latitude: array (nullable = true)
 |    |-- element: double (containsNull = true)
 |-- longitude: array (nullable = true)
 |    |-- element: double (containsNull = true)
 |-- speed: array (nullable = true)
 |    |-- element: double (containsNull = true)
 |-- sport: string (nullable = true)
 |-- timestamp: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- url: string (nullable = true)
 |-- userId: long (nullable = true)
```

Обзор столбцов

	Column Name	Data type
0	altitude	array<double>
1	gender	string
2	heart_rate	array<bigint>
3	id	bigint
4	latitude	array<double>
5	longitude	array<double>
6	speed	array<double>
7	sport	string
8	timestamp	array<bigint>
9	url	string
10	userId	bigint

Произвели обзор данных и столбцов.

```
print('Описание фрейма данных (только строковые и числовые столбцы):')
df.describe().toPandas()

print(f'Общее количество {df.count()} строк, печатаем несколько первых строк:')
df.limit(2).toPandas()
```

Описание фрейма данных (только строковые и числовые столбцы):

	summary	gender	id	sport	url	userId
0	count	253020	253020	253020	253020	253020
1	mean	None	3.566244412926132E8	None	None	4619648.939783417
2	stddev	None	1.574845634895318E8	None	None	3932877.7296880507
3	min	female	99296	aerobics	https://www.endomondo.com/users/10014612/worko...	69
4	max	unknown	674008008	yoga	https://www.endomondo.com/users/9991401/workou...	15481421

Общее количество 253020 строк, печатаем несколько первых строк:

	altitude	gender	heart_rate	id	latitude	longitude	speed	sport	timestamp	url	userId
0	[41.6, 40.6, 40.6, 38.4, 37.0, 34.0, 34.0, 34....	male	[100, 111, 120, 119, 120, 116, 125, 128, 131, ...	396826535	[60.173348765820265, 60.173239801079035, 60.17...	[24.64977040886879, 24.65014273300767, 24.6509...	[6.8652, 16.4736, 19.1988, 20.4804, 31.3956, 3...	bike	[1408898746, 1408898754, 1408898765, 140889877...	https://www.endomondo.com/users/10921915/worko...	10921915
1	[38.4, 39.0, 39.0, 38.2, 36.8, 36.8, 36.8, 35....	male	[100, 105, 111, 110, 108, 115, 126, 130, 132, ...	392337038	[60.173247596248984, 60.17320962622762, 60.172...	[24.649855233728886, 24.65015547350049, 24.650...	[9.0792, 13.284, 15.9336, 10.9476, 16.1676, 30...	bike	[1408221682, 1408221687, 1408221699, 140822170...	https://www.endomondo.com/users/10921915/worko...	10921915

Описание фрейма данных и вывод кол-ва строк + первые 2 строки.

Можно приступать к «очистке» данных. Сначала будем искать пропуски и нули:

```
print('Обзор столбцов')
pd.DataFrame(df.dtypes, columns = ['Column Name','Data type'])
```

Обзор столбцов

	Column Name	Data type
0	altitude	array<double>
1	gender	string
2	heart_rate	array<bigint>
3	id	bigint
4	latitude	array<double>
5	longitude	array<double>
6	speed	array<double>
7	sport	string
8	timestamp	array<bigint>
9	url	string
10	userId	bigint

Еще раз обзор столбоцов.

Для строковых столбцов проверяем наличие None и null.

Для числовых столбцов проверяем наличие нулей и NaN.

Для столбцов типа массив проверяем, содержит ли массив нули или NaN

```
string_columns = ['gender', 'sport', 'url']
numeric_columns = ['id','userId']
array_columns = ['altitude', 'heart_rate', 'latitude', 'longitude', 'speed', 'timestamp']
missing_values = {}
for index, column in enumerate(df.columns):
    if column in string_columns:    # проверить столбцы строк со значениями None и Null
#         missing_count = df.filter(col(column).eqNullSafe(None) | col(column).isNull()).count()
#         missing_values.update({column: missing_count})
        missing_count = df.filter(col(column).eqNullSafe(None) | col(column).isNull()).count()
        missing_values.update({column:missing_count})
    if column in numeric_columns:  # check zeroes, None, NaN
        missing_count = df.where(col(column).isin([0,None,np.nan])).count()
        missing_values.update({column:missing_count})
    if column in array_columns:  # check zeros and NaN
        missing_count = df.filter(array_contains(df[column], 0) | array_contains(df[column], np.nan)).count()
        missing_values.update({column:missing_count})
missing_df = pd.DataFrame.from_dict([missing_values])
missing_df
```

	altitude	gender	heart_rate	id	latitude	longitude	speed	sport	timestamp	url	userId
0	40848	0	1280	0	113	113	7741	0	0	0	0

✓ 15m 41s completed at 5:44 PM

(15 минут выполнялся, думал что зависло все)

Увидели количество пропущенных значених.

```
#  создаем новый столбец для подсчета количества временных меток, записанных для каждой строки/тренировки, с именем столбца «PerWorkoutRecordCount».
df = df.withColumn('PerWorkoutRecordCount', size(col('timestamp')))


# Эта часть написана как функция, которую можно будет использовать позже.
def user_activity_workout_summarize(df):
    user_count = format(df.select('userId').distinct().count(), ',d')
    workout_count = format(df.select('id').distinct().count(), ',d')
    activity_count = str(df.select('sport').distinct().count())
    sum_temp = df.agg(functions.sum('PerWorkoutRecordCount')).toPandas()
    total_records_count = format(sum_temp['sum(PerWorkoutRecordCount)'][0],',d')
    columns=['Users count', 'Activity types count','Workouts count', 'Total records count']
    data = [[user_count], [activity_count], [workout_count], [total_records_count]]
    sum_dict = {column: data[i] for i, column in enumerate(columns)}
    sum_df = pd.DataFrame.from_dict(sum_dict)[columns]
    gender_user_count = df.select('gender','userId').distinct().groupBy('gender').count().toPandas()
    gender_activities_count = df.groupBy('gender').count().toPandas()
    gender_user_activity_count = gender_user_count.join(
        gender_activities_count.set_index('gender'), on='gender'
        , how='inner', lsuffix='_gu'
    )
    gender_user_activity_count.columns = ['Gender', '# of users', 'Activities (workouts) count']

    return sum_df, gender_user_activity_count

sum_dfs= user_activity_workout_summarize(df)
print('\nОбщая сводка набора данных о пользователях, тренировках и количестве записей (предварительная фильтрация):')
sum_dfs[0]
```

```
Общая сводка набора данных о пользователях, тренировках и количестве записей (предварительная фильтрация):
```

	Users count	Activity types count	Workouts count	Total records count
0	1,104	49	253,020	111,541,956

Общее кол-во записей более 110 миллионов.

```
print('Количество тренировок с менее чем 50 записями и статистической сводкой:')
removed_df = df.select('PerWorkoutRecordCount').where(df.PerWorkoutRecordCount < 50) \
               .toPandas().describe().astype(int)
removed_df.rename(columns = {'PerWorkoutRecordCount': 'PerWorkoutRecordCount <50'}, inplace=True)
removed_df.T
```

```
Количество тренировок с менее чем 50 записями и статистической сводкой:
```

	count	mean	std	min	25%	50%	75%	max
PerWorkoutRecordCount <50	5541	23	14	1	11	22	36	49

Переходим к ленивой оценке:
Ленивая оценка расширяет возможности Apache Spark за счет сокращения времени выполнения операций RDD.

Оцениваем лучшие типы тренировок и выводим их

```
[45] ranked_sport_users_df = df.select(df.sport, df.userId) \
        .distinct() \
        .groupBy(df.sport) \
        .count() \
        .orderBy("count", ascending=False)

    # Топ-5 типов тренировок
    highest_sport_users_df = ranked_sport_users_df.limit(5).toPandas()
    # Переименование имени столбца: 'count' -> Users count Количество пользователей.
    highest_sport_users_df.rename(columns = {'count':'Users count'}, inplace = True)
    # Подсчет общего количества пользователей. Результат потребуется для подсчета процентного соотношения.
    total_sports_users = ranked_sport_users_df.groupBy().sum().collect()[0][0]
```

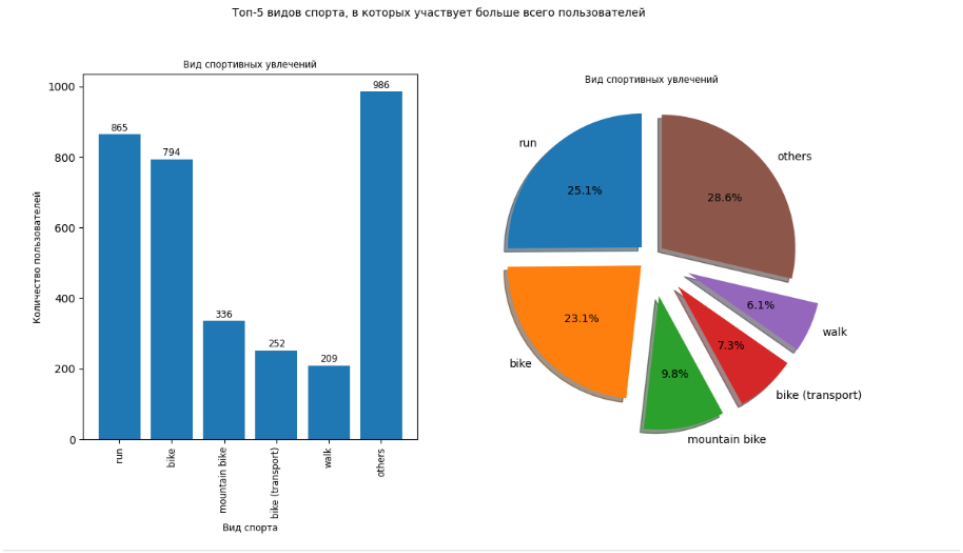
```
ranked_sport_users_df.collect()[:5]
```

```
[Row(sport='run', count=865),
 Row(sport='bike', count=794),
 Row(sport='mountain bike', count=336),
 Row(sport='bike (transport)', count=252),
 Row(sport='walk', count=209)]
```

Приступаем к исследовательскому анализу данных:

```python
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# Расчет процентов и добавление группы "others"
highest_sport_users_df_renamed = highest_sport_users_df.copy()
highest_sport_users_df_renamed['percentage'] = highest_sport_users_df['Users count'] / total_sports_users * 100

others = {
    'sport': 'others',
    'Users count': total_sports_users - sum(highest_sport_users_df_renamed['Users count']),
    'percentage': 100 - sum(highest_sport_users_df_renamed['percentage'])
}

others_df = pd.DataFrame([others])
highest_sport_users_df_renamed = pd.concat([highest_sport_users_df_renamed, others_df], ignore_index=True)

print('Топ-5 видов спорта, в которых участвует больше всего пользователей:')
print(highest_sport_users_df_renamed)

# Построение графиков
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

# Барчарт
axs[0].bar(x=highest_sport_users_df_renamed['sport'], height=highest_sport_users_df_renamed['Users count'])
axs[0].set_title('Вид спортивных увлечений', fontsize='small')
axs[0].set_xlabel('Вид спорта', fontsize='small')
axs[0].set_ylabel('Количество пользователей', fontsize='small')

# Установка фиксированного локатора для меток оси X
axs[0].xaxis.set_major_locator(ticker.FixedLocator(range(len(highest_sport_users_df_renamed))))
axs[0].set_xticklabels(highest_sport_users_df_renamed['sport'], rotation='vertical', fontsize='small')

# Добавление значений над барами
for i, v in enumerate(highest_sport_users_df_renamed['Users count']):
    axs[0].text(i, v + 10, str(v), ha='center', fontsize='small')

# Пайчарт
explode = (0.1, 0.1, 0.3, 0.3, 0.3, 0.1)
axs[1].pie(
    x=highest_sport_users_df_renamed['percentage'],
    labels=highest_sport_users_df_renamed['sport'],
    autopct='%1.1f%%', shadow=True, explode=explode, startangle=90, radius=1
)
axs[1].set_title('Вид спортивных увлечений', fontsize='small')

# Общий заголовок
fig.text(0.5, 1.02, 'Топ-5 видов спорта, в которых участвует больше всего пользователей', ha='center', va='top', transform=fig.transFigure)

# Сохранение графика
plt.savefig('top_sports_analysis.png', bbox_inches='tight', dpi=300)

# Отображение графика
plt.show();
```

Составили 2 диаграммы с лучшими видами тренировок:




```python
# Просмотр данных в зависимости от пола
activities_by_gender = df.groupBy('sport', 'gender').count().toPandas()
activities_by_gender[:5]
```

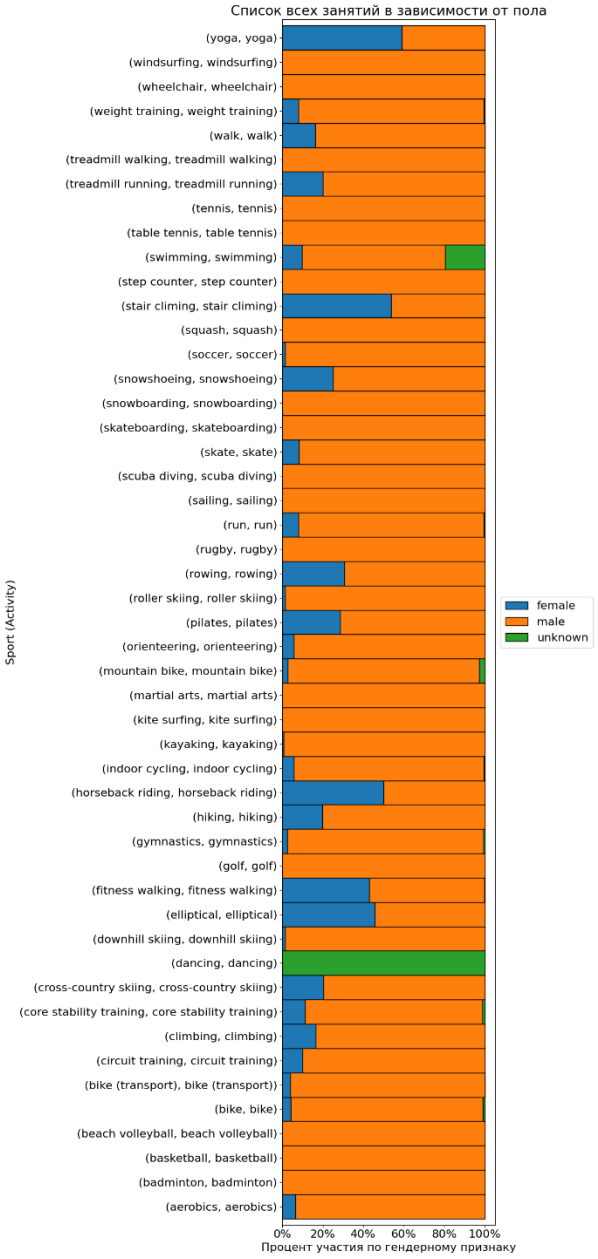
	sport	gender	count
0	hiking	female	71
1	core stability training	male	1103
2	run	male	107882
3	kayaking	male	253
4	mountain bike	male	12782

Просмотрели данные в зависимости от пола

Требуется изменить форму приведенной выше таблицы, чтобы разбить столбец пол, для последующей визуализации.

Определим принцип работы с категориями:

- метод DataFrame.stack(): "сводит" уровень (возможно, иерархический) меток столбцов, возвращая а DataFrame с новым, самым внутренним уровнем меток строк;
- метод DataFrame.unstack(): (обратная операция DataFrame.stack()) "сводит" уровень индекса строки (возможно, иерархический) к оси столбца, создавая измененный DataFrame с новым, самым внутренним уровнем меток столбцов.

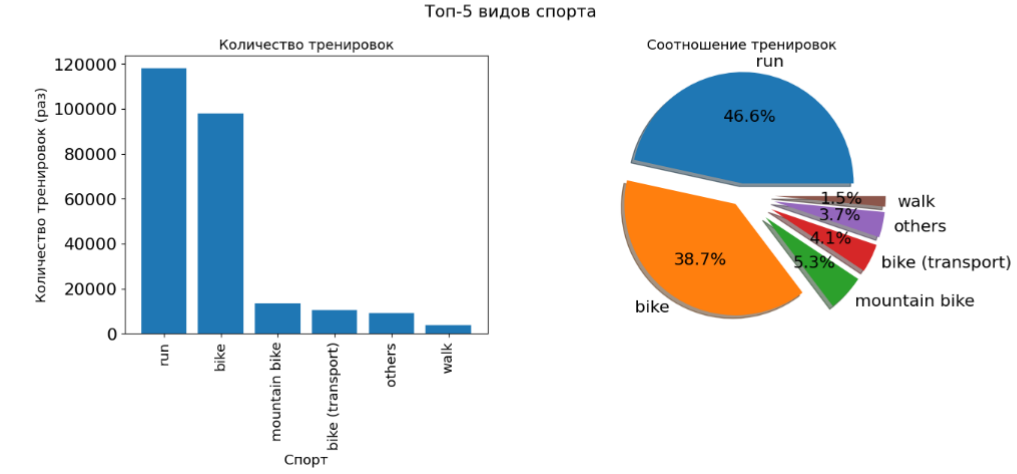


Смотрим верхнее парето из 5 видов спорта, в которых больше всего участников.

```
# Pivot the data by gender and sport
activities_by_gender_df = activities_by_gender.pivot_table(
    index="sport", columns="gender", values='count', fill_value=0) \
    .reset_index().rename_axis(None, axis=1)

# Add total count and percentage
activities_by_gender_df['total'] = activities_by_gender_df['male'] + activities_by_gender_df['female'] + activities_by_gender_df['unknown']
activities_by_gender_df['percentage'] = activities_by_gender_df['total'] / sum(activities_by_gender_df['total']) * 100

# Get top 5 activities by percentage
top_activities_by_gender_df = activities_by_gender_df.sort_values(by='percentage', ascending=False).head(5)

# Create the 'others' group
others = {'sport': 'others'}
for column in ['female', 'male', 'unknown', 'total', 'percentage']:
    value = sum(activities_by_gender_df[column]) - sum(top_activities_by_gender_df[column])
    others.update({column: value})

# Convert 'others' to a DataFrame
others_df = pd.DataFrame([others])

# Use pd.concat to add 'others' row to the DataFrame
top_activities_by_gender_df = pd.concat([top_activities_by_gender_df, others_df], ignore_index=True)

# Sort by percentage again
top_activities_by_gender_df = top_activities_by_gender_df.sort_values(by='percentage', ascending=False)

# Display the final DataFrame
print(top_activities_by_gender_df)

# Plotting
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=plt.figaspect(0.35))

# Bar plot for total count of activities
axs[0].bar(x=top_activities_by_gender_df['sport'], height=top_activities_by_gender_df['total'])
axs[0].set_title('Количество тренировок', fontsize='small')
axs[0].set_xlabel('Спорт', fontsize='small')
axs[0].set_ylabel('Количество тренировок (раз)', fontsize='small')
axs[0].set_xticklabels(top_activities_by_gender_df['sport'], rotation='vertical', fontsize='small')

# Pie chart for percentage distribution
explode = (0.1, 0.1, 0.3, 0.3, 0.3, 0.3)
axs[1].pie(
    x=top_activities_by_gender_df['percentage'],
    labels=top_activities_by_gender_df['sport'],
    autopct='%1.1f%%', shadow=True, explode=explode, radius=1
)
axs[1].set_title('Соотношение тренировок', fontsize='small');

# Add text to the figure
fig.text(0.5, 1.02, 'Топ-5 видов спорта', ha='center', va='top', transform=fig.transFigure)

# Show the plot
plt.show();
```

Вывод:

```
             sport  female      male  unknown     total  percentage
0              run  9360.0  107882.0    660.0  117902.0   46.597897
1             bike  4172.0   92966.0    863.0   98001.0   38.732511
2    mountain bike   353.0   12782.0    375.0   13510.0    5.339499
3  bike (transport)   414.0   10030.0      1.0   10445.0    4.128132
5           others  1034.0    8255.0     42.0    9331.0    3.687851
4             walk   626.0    3204.0      1.0    3831.0    1.514110
<ipython-input-41-89ad68df0949>:39: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
  axs[0].set_xticklabels(top_activities_by_gender_df['sport'], rotation='vertical', fontsize='small')
```



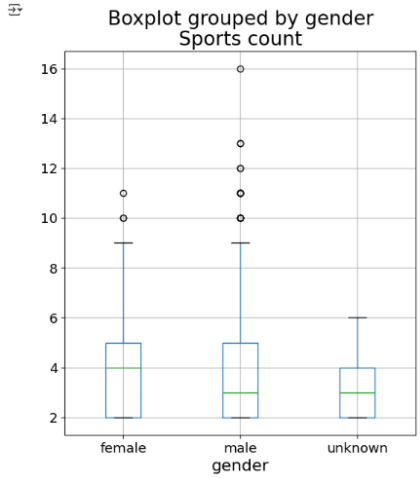
Узнаем сколько людей участвовало более чем в одном виде спорта.

```
[42] min_number_of_sports = 1

     sport_df = df \
         .select(df.userId, df.gender, df.sport) \
         .distinct() \
         .groupBy(df.userId, df.gender) \
         .count()

     user_more_sports_df = sport_df \
                         .filter(sport_df["count"] > min_number_of_sports) \
                         .orderBy("count", ascending = False) \
                         .toPandas()
     user_more_sports_df.rename(columns = {'count':'Sports count'}, inplace = True)
     user_more_sports_df.describe().astype(int).T
```

	count	mean	std	min	25%	50%	75%	max
userId	822	4860464	3953412	69	1609606	3730685	7554937	15481421
Sports count	822	3	2	2	2	3	5	16

Разобьем по полу

```
plot = user_more_sports_df.boxplot(column='Sports count', by='gender', fontsize='small', figsize=(6,7))
```



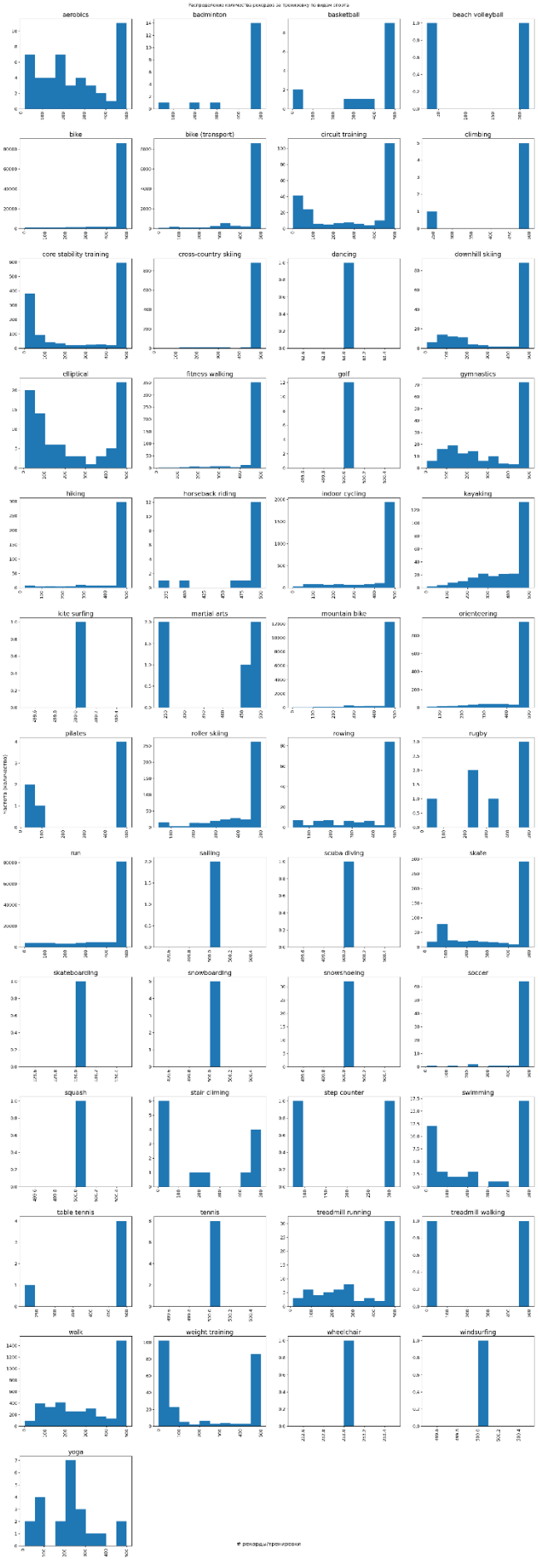
Выбросы есть, но в целом разница между мужчинами и женщинами не очень большая.

Для более детального наблюдения мы разбили количество рекордов по каждому виду деятельности по каждому отдельному виду спорта.

Исходя из распределения, максимальное количество записей за тренировку составляет 500, но не все тренировки и виды спорта достигают этого числа.

```
print('\nГрафик распределения тренировок по видам деятельности:')
plot_size_x, plot_size_y = 5, 5
figsize_x, figsize_y = plot_size_x * 4 + 3, plot_size_y * 13 + 1
figsize=(figsize_x, figsize_y)
fig = plt.figure(figsize=figsize) #
grid_size = (13,4)
ax = plt.subplot2grid(grid_size, (0,0), colspan=1, rowspan=1)
#fig, ax = plt.subplots()
PerWorkoutRecordCount_dist = df.select('PerWorkoutRecordCount', 'sport').toPandas().hist(
    column='PerWorkoutRecordCount', bins=10, sharex = False, grid=True
    , xlabelsize='small', ylabelsize='small', by='sport', ax = ax
    , layout = grid_size, figsize=figsize
    )
a = fig.tight_layout()
title = fig.text(0.5, 1, 'Распределение количества рекордов за тренировку по видам спорта', ha='center'
         , fontsize='small', transform=fig.transFigure);
xlabel = fig.text(
    0.5, 0.01, '# рекорды/тренировки', va='bottom', ha='center', transform=fig.transFigure
)
ylabel = fig.text(0.01, 0.5, 'Частота (количество)', va='center', rotation='vertical');
```

Вывод:


aerobics
badminton
basketball
beach volleyball
bike
bike (transport)
circuit training
climbing
core stability training
cross-country skiing
dancing
downhill skiing
elliptical
fitness walking
golf
gymnastics
hiking
horseback riding
indoor cycling
kayaking
kite surfing
martial arts
mountain bike
orienteering
pilates
roller skiing
rowing
rugby
run
sailing
scuba diving
skate
skateboarding
snowboarding
snowshoeing
soccer
squash
stair climing
step counter
swimming
table tennis
tennis
treadmill running
treadmill walking
walk
weight training
wheelchair
windsurfing
yoga

Отфильтруем по тем, у кого больше 10 тренировок:

```
# Отфильтровать df с минимум 10 записями (поскольку мы предполагаем, что любой user_id с записью менее 10 не будет иметь смысла)
qualified_df = df \
    .select(df.sport, df.userId, df.gender) \
    .groupBy(df.sport, df.userId, df.gender) \
    .count()
qualified_df = qualified_df.filter(qualified_df["count"] >= 10) \
    .orderBy("count", ascending = False)
```

```
print('Количество пользователей, имеющих более 10 тренировок:')
qualified_pd_df = qualified_df.select("userId", "gender").distinct() \
    .groupBy(qualified_df.gender).count().toPandas()
qualified_pd_df.rename(columns={'count': 'Users count'}, inplace=True)
qualified_pd_df
qualified_users_count = sum(qualified_pd_df['Users count'])
total_users_count = df.select('userId').distinct().count()
qualified_percentage = round((qualified_users_count / total_users_count),2) * 100
print('\nТаким образом, есть {} / {} пользователей, соответствующих 10 критериям исторических записей, что {:.2f}%' \
      .format(qualified_users_count, total_users_count, qualified_percentage)
     )
```

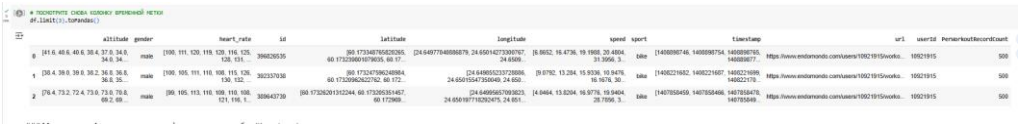
```
Количество пользователей, имеющих более 10 тренировок:
```

	gender	Users count
0	unknown	13
1	female	88
2	male	886

```
Таким образом, есть 987 / 1104 пользователей, соответствующих 10 критериям исторических записей, что 89.00%
```

Переходим к пользовательским функциям:

Сначала смотрим колонку временной метки:



Далее создаем 4 вспомогательные функции для столбца 'timestamp', как описано выше, а затем преобразуем их в UDF.

```
# Преобразование столбца метки времени в Datetime.Datetime, чтобы позже использовать его для функции .withColumn.
def to_time(timestamp_list):
    # преобразовать в дату и время и минус 7 часов из-за разницы во временном окне Endomondo с временем utc в качестве описания набора данных
    return [datetime.fromtimestamp(t) - timedelta(hours=7) for t in timestamp_list]

# Регистрация вспомогательной функции to_time в структуре UDF pyspark
udf_to_time = udf(to_time, ArrayType(elementType=TimestampType()))

# Вспомогательная функция для получения продолжительности (в минутах) списка значений даты и времени, которая будет позже использована для функции withColumn.
def get_duration(datetime_list):
    time_dif = max(datetime_list) - min(datetime_list)
    return time_dif.seconds/60

# Регистрация вспомогательной функции get_duration как пользовательской функции в pyspark.
udf_get_duration = udf(get_duration, FloatType())

# Вспомогательная функция для получения времени начала тренировки из списка даты и времени, которая будет позже использована для функции withColumn.
def get_start_time(datetime_list):
    return min(datetime_list)

# Регистрация вспомогательной функции get_start_time как пользовательской функции в pyspark
udf_get_start_time = udf(get_start_time, TimestampType())

# Вспомогательная функция для получения списка интервалов во время тренировки
def get_interval(datetime_list):
    if len(datetime_list) == 1:
        return [0]
    else:
        interval_list = []
        for i in range(0, len(datetime_list)-1):
            interval = (datetime_list[i+1] - datetime_list[i]).seconds
            interval_list.append(interval)
        return interval_list

# Регистрация вспомогательной функции get_interval как пользовательской функции в pyspark.
udf_get_interval = udf(get_interval, ArrayType(elementType=IntegerType()))

# Создание нового столбца date_time для преобразования метки времени в формат даты и времени Python для последующего использования.
df = df.withColumn('date_time',
    udf_to_time('timestamp'))

# Создание столбца «workout_start_time», чтобы получить время начала каждой тренировки/строки:
df = df.withColumn('workout_start_time', hour(udf_get_start_time('date_time')))

# Создание столбца продолжительности из только что созданного столбца date_time, используя функцию udf udf_get_duration, определенную выше.
df = df.withColumn('duration', udf_get_duration('date_time'))

# Создание столбца интервала из столбца date_time, используя функцию udf udf_get_interval, определенную выше.
df = df.withColumn('interval', udf_get_interval('date_time'))

print('Новые столбцы (''date_time'', ''workout_start_time'' in hour\
, ''duration'' in minutes & ''interval'' in seconds)\n, first 5 rows:')
df.select('timestamp','date_time', 'workout_start_time', 'duration', 'interval').limit(5).toPandas()
```

Вывод:

Новые столбцы (date_time, workout_start_time in hour, duration in minutes & interval in seconds)
, first 5 rows:

	timestamp	date_time	workout_start_time	duration	interval
0	[1408898746, 1408898754, 1408898765, 140889877...	[2014-08-24 09:45:46, 2014-08-24 09:45:54, 201...	9	126.483330	[8, 11, 13, 16, 6, 23, 16, 23, 29, 23, 24, 24,...
1	[1408221682, 1408221687, 1408221699, 140822170...	[2014-08-16 13:41:22, 2014-08-16 13:41:27, 201...	13	74.000000	[5, 12, 8, 4, 5, 6, 4, 4, 5, 13, 7, 17, 4, 10,...
2	[1407858459, 1407858466, 1407858478, 140785849...	[2014-08-12 08:47:39, 2014-08-12 08:47:46, 201...	8	112.483330	[7, 12, 12, 16, 6, 7, 16, 11, 19, 13, 17, 11, ...
3	[1407432042, 1407432048, 1407432056, 140743206...	[2014-08-07 10:20:42, 2014-08-07 10:20:48, 201...	10	75.316666	[6, 8, 9, 4, 13, 4, 3, 4, 13, 10, 13, 13, 7, 1...
4	[1406909434, 1406909436, 1406909437, 140690943...	[2014-08-01 09:10:34, 2014-08-01 09:10:36, 201...	9	22.616667	[2, 1, 1, 3, 2, 1, 4, 2, 1, 2, 1, 1, 2, 4, 4, ...

Смотрим на продолжительность каждой тренировки в минутах.

```
[51] print('\nСтатистика длительности (в минутах):')
     df.select('duration').toPandas().describe().T
```

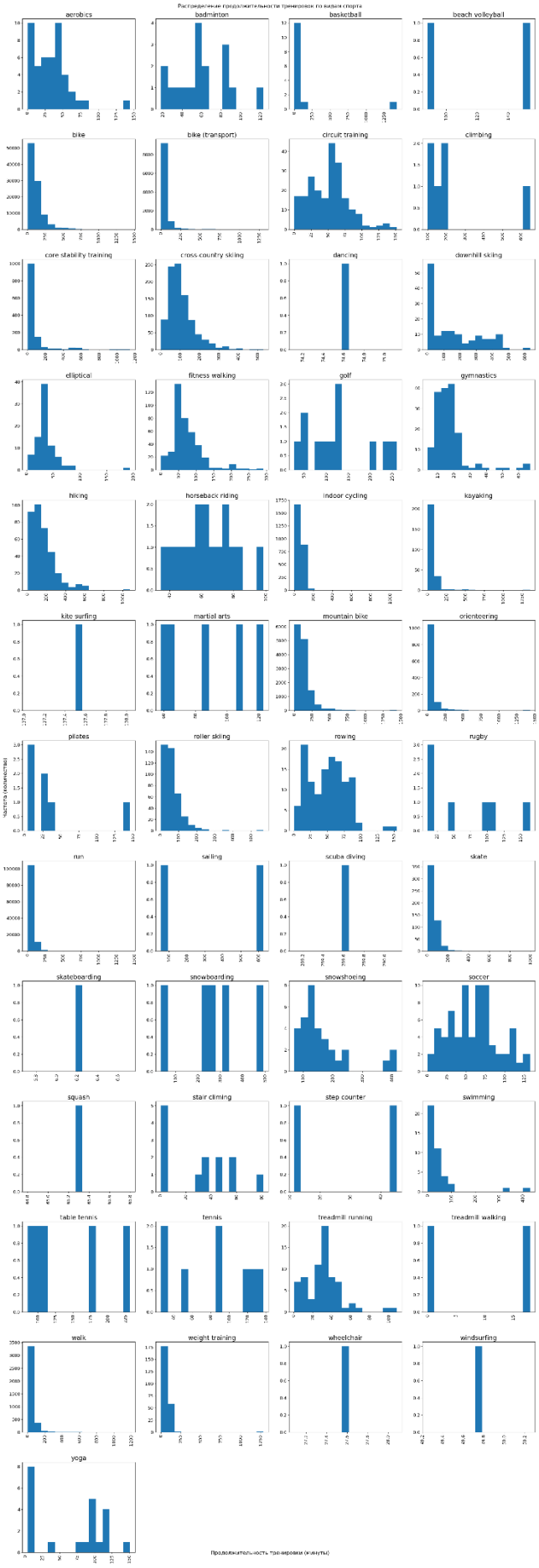
Статистика длительности (в минутах):

	count	mean	std	min	25%	50%	75%	max
duration	253020.0	84.486427	98.089287	0.0	32.166668	58.916668	102.73333	1439.966675

Строим график продолжительности:

```
print('\nПостроение графика распределения продолжительности по видам спорта:')
plot_size_x, plot_size_y = 5, 5
figsize_x, figsize_y = plot_size_x * 4 +3, plot_size_y * 13 + 1
figsize = (figsize_x, figsize_y)
fig = plt.figure(figsize=figsize) #
grid_size = (13,4)
ax = plt.subplot2grid(grid_size, (0,0), colspan=1, rowspan=1)

duration_dist = df.select('duration', 'sport').toPandas().hist(
    column='duration', by='sport', bins=15, sharex = False, grid=True
    , xlabelsize='small', ylabelsize='small' , ax = ax
    , layout = grid_size, figsize=figsize
    )
a = fig.tight_layout()
title = fig.text(0.5, 1, 'Распределение продолжительности тренировок по видам спорта'
             , ha='center', va='center', transform=fig.transFigure
            )
xlabel = fig.text(0.5, 0.01, 'Продолжительность тренировки (минуты)'
             , ha='center', va='center', transform=fig.transFigure)
ylabel = fig.text(0.01, 0.5, 'Частота (количество)', va='center', rotation='vertical');
```

Графики:


Распределение продолжительности тренировок по видам спорта
aerobics
badminton
basketball
beach volleyball
bike
bike (transport)
circuit training
climbing
core stability training
cross-country skiing
dancing
downhill skiing
elliptical
fitness walking
golf
gymnastics
hiking
horseback riding
indoor cycling
kayaking
kite surfing
martial arts
mountain bike
orienteering
pilates
roller skiing
rowing
rugby
run
sailing
scuba diving
skate
skateboarding
snowboarding
snowshoeing
soccer
squash
stair climbing
step counter
swimming
table tennis
tennis
treadmill running
treadmill walking
walk
weight training
wheelchair
windsurfing
yoga
Частота (количество)
Продолжительность тренировки (минуты)

Переходим к следующему этапу: **Преобразование объектов строк в устойчивый распределенный набор данных Spark**

Рассчитаем некоторые основные статистические данные (мин/макс/среднее/среднее/стандартное отклонение и 4 квантиля 25/50/75/95) в pySpark, преобразуем в Rdd и построим их на графике.

```
# Вспомогательная функция для расчета статистики имени столбца из кортежа x (спорт, список записей столбца) tuple x of (sport, records list of the column)
#, статистика для расчета также предоставляется в качестве входных данных
def calculate_stats(x,column_name, stat_list):
    sport, records_list = x
    stat_dict = {'sport': sport}
    if 'min' in stat_list:
        min_stat = min(records_list)
        stat_dict.update({'min ' + column_name : min_stat})
    if 'max' in stat_list:
        max_stat = max(records_list)
        stat_dict.update({'max ' + column_name: max_stat})
    if 'mean' in stat_list:
        average_stat = stats.mean(records_list)
        stat_dict.update({'mean ' + column_name: average_stat})
    if 'stdev' in stat_list:
        std_stat = stats.stdev(records_list)
        stat_dict.update({'stdev ' + column_name: std_stat})
    if '50th percentile' in stat_list:
        median_stat = stats.median(records_list)
        stat_dict.update({'50th percentile ' + column_name: median_stat})
    if '25th percentile' in stat_list:
        percentile_25th_stat = np.percentile(records_list, 25)
        stat_dict.update({'25th percentile ' + column_name: percentile_25th_stat})
    if '75th percentile' in stat_list:
        percentile_75th_stat = np.percentile(records_list, 75)
        stat_dict.update({'75th percentile ' + column_name: percentile_75th_stat})
    if '95th percentile' in stat_list:
        percentile_95th_stat = np.percentile(records_list, 95)
        stat_dict.update({'95th percentile ' + column_name: percentile_95th_stat})
    return stat_dict

def to_list(a):
    return a

def extend(a, b):
    a.extend(b)
    return a

def retrieve_array_column_stat_df(df, column_name, stat_list):
    # Преобразование спорт и «column_name» в RDD, чтобы легко рассчитать статистику интервалов по видам спорта.
    sport_record_rdd = df.select('sport', column_name).rdd \
    .map(tuple).combineByKey(to_list, extend, extend).persist()

    # Вычислить статистику входного столбца, вызвав функцию Calcul_stats, определенную выше.
    record_statistic_df = pd.DataFrame(sport_record_rdd.map(
        lambda x: calculate_stats(x, column_name,stat_list)).collect()
                                       )
    # Установка правильного порядка столбцов данных.
    columns_order = ['sport'] + [stat + ' ' + column_name for stat in stat_list]
    # Изменение порядка столбцов
    return record_statistic_df[columns_order]

stat_list = ['min', '25th percentile', 'mean', '50th percentile',
                    '75th percentile', '95th percentile', 'max', 'stdev']
interval_statistic_df = retrieve_array_column_stat_df(df, column_name='interval', stat_list=stat_list)
print('\nПросмотр статистики интервалов в секундах (по виду спорта)' )
interval_statistic_df
```

Вывод:

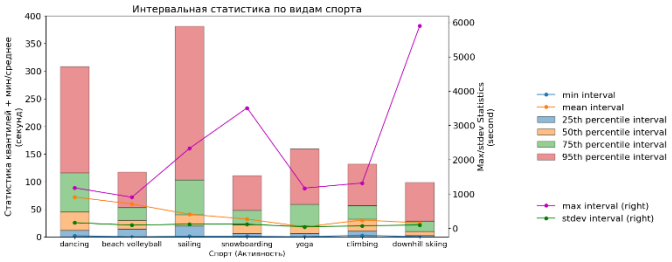
Просмотр статистики интервалов в секундах (по виду спорта)

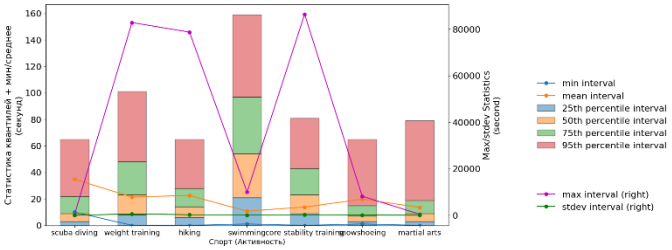
	sport	min interval	25th percentile interval	mean interval	50th percentile interval	75th percentile interval	95th percentile interval	max interval	stdev interval
0	squash	2	6.0	7.851703	8.0	10.00	13.00	18	2.900293
1	circuit training	0	3.0	9.675286	5.0	9.00	24.00	6029	47.256595
2	climbing	3	14.0	30.470308	16.0	23.00	64.65	1318	69.604276
3	martial arts	0	6.0	13.563431	8.0	14.00	37.00	446	22.455242
4	cross-country skiing	0	6.0	15.107559	10.0	16.00	29.00	86399	426.429807
5	beach volleyball	0	12.0	59.516393	33.0	71.25	191.85	903	89.604732
6	soccer	0	4.0	7.579923	6.0	9.00	14.00	3584	33.544463
7	treadmill running	0	2.0	5.513624	4.0	6.00	13.00	683	10.635236
8	gymnastics	0	1.0	3.214098	2.0	3.00	8.00	2837	16.358282
9	dancing	2	20.0	72.209677	20.0	63.00	278.35	1178	159.687927
10	bike	0	5.0	15.750739	10.0	17.00	34.00	86399	276.575945
11	mountain bike	0	6.0	16.040844	11.0	18.00	34.00	86399	243.660599
12	swimming	1	3.0	10.661234	4.0	8.00	50.00	9962	107.103196
13	tennis	1	5.0	9.976954	10.0	13.00	22.00	58	6.459012
14	core stability training	0	3.0	13.778445	6.0	13.00	43.00	86399	258.837691
15	stair climing	0	3.0	8.304059	6.0	9.00	18.00	1862	36.519847
16	rowing	0	4.0	7.445080	6.0	9.00	14.00	2453	15.077587
17	kayaking	0	5.0	11.859714	7.0	11.00	22.00	86342	391.203691
18	yoga	0	3.0	18.114438	7.0	19.00	70.00	1170	39.695865
19	step counter	1	7.0	8.877285	10.0	12.00	13.00	16	3.591450
20	downhill skiing	0	6.0	25.353402	16.0	26.00	63.00	5906	95.364198
21	fitness walking	0	5.0	9.265518	8.0	11.00	21.00	3403	19.103858
22	scuba diving	10	21.0	34.823647	33.0	43.00	62.00	123	16.067158
23	badminton	0	4.0	8.181660	7.0	10.00	17.00	1677	21.833185
24	weight training	0	3.0	21.347607	6.0	10.00	60.00	82806	595.522372
25	snowshoeing	1	9.0	19.690256	14.0	20.00	38.00	8197	106.664846
26	walk	0	3.0	7.912633	5.0	8.00	18.00	86399	194.586603
27	horseback riding	0	4.0	7.932153	7.0	10.00	18.00	291	7.298296
28	rugby	1	3.0	11.860455	5.0	12.00	36.00	1744	44.593851
29	wheelchair	1	4.0	6.567460	6.0	8.00	13.00	17	3.769163

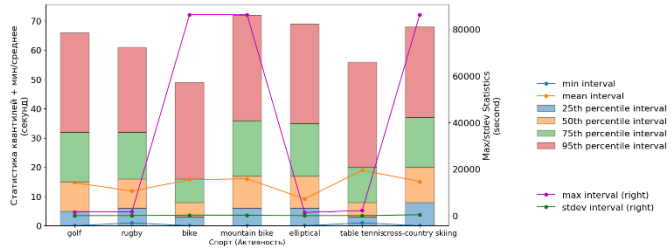
Отображение в виде столбцов и линейных диаграмм:

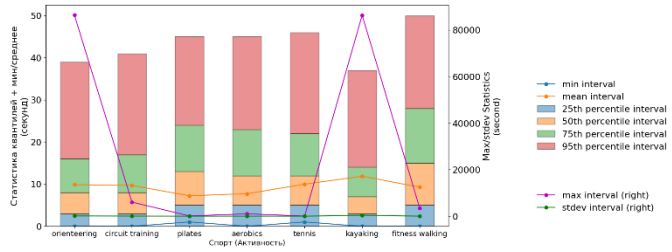
```
import numpy as np
import matplotlib.pyplot as plt

print('\nОбобщенная статистика интервальных видов спорта:')

bar_columns = ['25th percentile interval', '50th percentile interval', '75th percentile interval', '95th percentile interval']
line_columns1 = ['min interval', 'mean interval']
line_columns2 = ['max interval', 'stdev interval']

interval_statistic_df = interval_statistic_df.sort_values(by='95th percentile interval', ascending=False)

figsize = (13, 59)
fig, axs = plt.subplots(nrows=7, figsize=figsize)

d = axs[0].set_title('Интервальная статистика по видам спорта', fontsize=18)

for i in range(7):
    interval_statistic_sub_df = interval_statistic_df.iloc[i*7:i*7+7]

    # Plot the bar chart for the quantiles
    plot1 = interval_statistic_sub_df[['sport'] + bar_columns].groupby(['sport']).agg(np.mean).plot(
        kind='bar', stacked=True, grid=False, alpha=0.5, edgecolor='black', ax=axs[i]
    )

    # Plot the line chart for min and mean intervals
    plot2 = interval_statistic_sub_df[['sport'] + line_columns1].plot(x='sport', ax=axs[i], marker='o')

    # Create a secondary y-axis for max and stdev intervals
    ax2 = axs[i].twinx()
    plot3 = interval_statistic_sub_df[['sport'] + line_columns2].plot(x='sport', ax=ax2, marker='o', color=['m', 'g'])

    # Legends for the plots
    a = axs[i].legend(loc='center left', fontsize=16, bbox_to_anchor=(1.2, 0.5), frameon=False)
    a = ax2.legend(labels=['max interval (right)', 'stdev interval (right)'], loc="center left", fontsize=16, bbox_to_anchor=(1.2, 0.11), frameon=False)

    # Formatting for x-ticks and labels
    b = axs[i].set_xticklabels(interval_statistic_sub_df['sport'], rotation='horizontal', fontsize='small')
    c = axs[i].set_xlabel('Спорт (Активность)', fontsize='small')
    d = axs[i].set_ylabel('Статистика квантилей + мин/среднее\n(секунд)', fontsize=16)
    e = ax2.set_ylabel('Max/stdev Statistics\n(second)', fontsize=16)

    # Set font size for y-axis ticks
    for tick in axs[i].yaxis.get_major_ticks():
        tick.label1.set_fontsize(16)  # Use label1 for primary y-axis labels
    ax2.tick_params(axis='y', labelsize=16)

    # Make sure x-tick labels are visible for all subplots
    b = plt.setp([a.get_xticklabels() for a in fig.axes[:-1]], visible=True)

# Adjust the layout for better visualization
plt.subplots_adjust(hspace=0.2)

# Show the plot
plt.show();
```

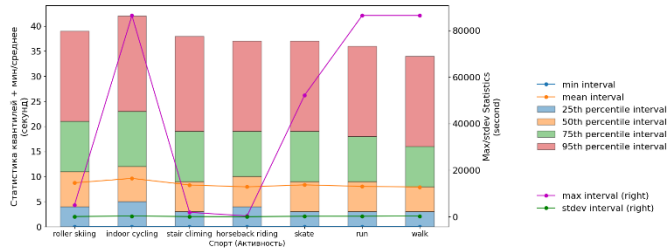
Диаграммы:

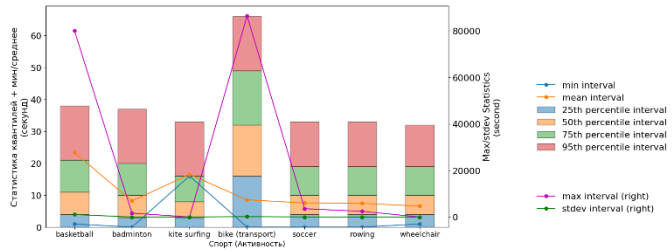

Интервальная статистика по видам спорта
Статистика квантилей + мин/среднее (секунд)
Max/stdev Statistics (second)
Спорт (Активность)
dancing
beach volleyball
sailing
snowboarding
yoga
climbing
downhill skiing
min interval
mean interval
25th percentile interval
50th percentile interval
75th percentile interval
95th percentile interval
max interval (right)
stdev interval (right)

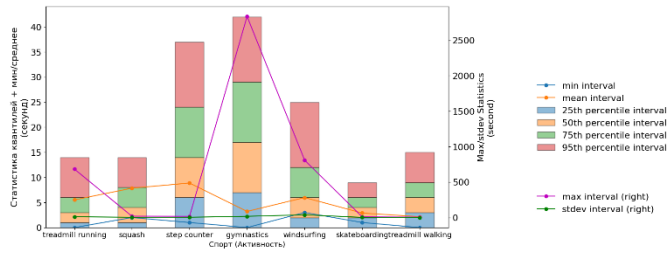

Статистика квантилей + мин/среднее (секунд)
Max/stdev Statistics (second)
Спорт (Активность)
scuba diving
weight training
hiking
swimming
core stability training
snowshoeing
martial arts
min interval
mean interval
25th percentile interval
50th percentile interval
75th percentile interval
95th percentile interval
max interval (right)
stdev interval (right)


Статистика квантилей + мин/среднее (секунд)
Max/stdev Statistics (second)
Спорт (Активность)
golf
rugby
bike
mountain bike
elliptical
table tennis
cross-country skiing
min interval
mean interval
25th percentile interval
50th percentile interval
75th percentile interval
95th percentile interval
max interval (right)
stdev interval (right)


Статистика квантилей + мин/среднее (секунд)
Max/stdev Statistics (second)
Спорт (Активность)
orienteering
circuit training
pilates
aerobics
tennis
kayaking
fitness walking
min interval
mean interval
25th percentile interval
50th percentile interval
75th percentile interval
95th percentile interval
max interval (right)
stdev interval (right)


Статистика квантилей + мин/среднее (секунд)
Max/stdev Statistics (second)
Спорт (Активность)
roller skiing
indoor cycling
stair climing
horseback riding
skate
run
walk
min interval
mean interval
25th percentile interval
50th percentile interval
75th percentile interval
95th percentile interval
max interval (right)
stdev interval (right)


Статистика квантилей + мин/среднее (секунд)
Max/stdev Statistics (second)
Спорт (Активность)
basketball
badminton
kite surfing
bike (transport)
soccer
rowing
wheelchair
min interval
mean interval
25th percentile interval
50th percentile interval
75th percentile interval
95th percentile interval
max interval (right)
stdev interval (right)

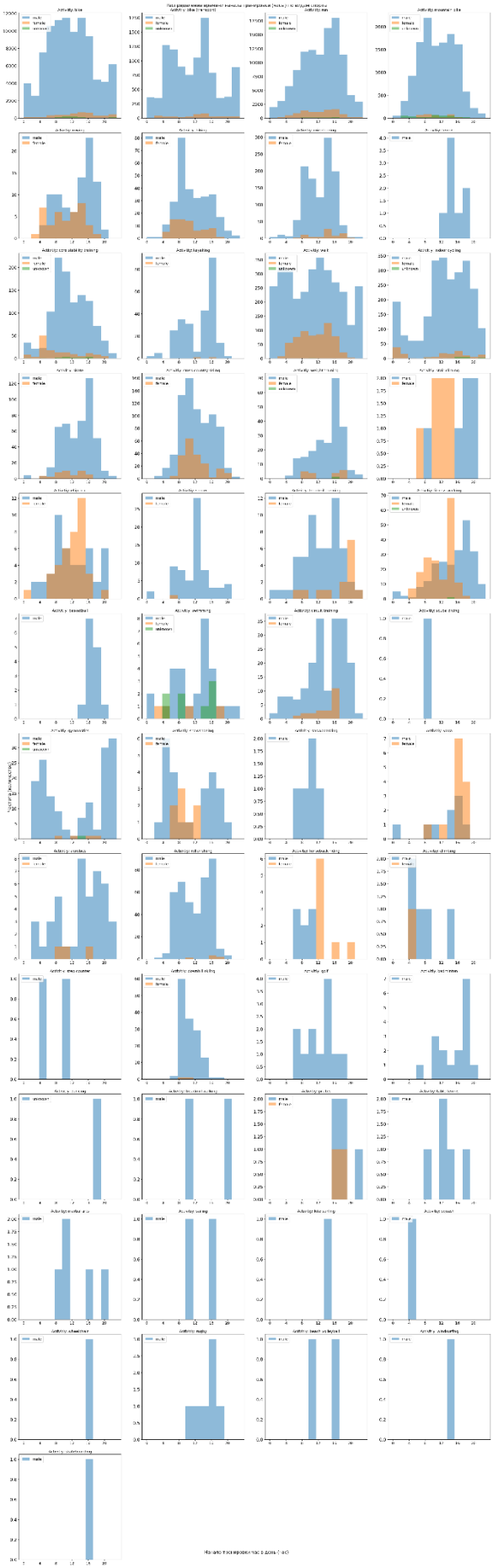

Статистика квантилей + мин/среднее (секунд)
Max/stdev Statistics (second)
Спорт (Активность)
treadmill running
squash
step counter
gymnastics
windsurfing
skateboarding
treadmill walking
min interval
mean interval
25th percentile interval
50th percentile interval
75th percentile interval
95th percentile interval
max interval (right)
stdev interval (right)

Используем гистограмму, чтобы посмотреть на распределение часов начала тренировок, сгруппированных по видам спорта и с разбивкой по полу. Мы делим день на интервалы по 2 часа, всего получается 12 частей.

```
[56] # Получите таблицу пола, вида спорта и времени начала тренировки для построения графика.
     start_time_df = df.select('gender', 'sport','workout_start_time').toPandas()
```

```
activities = start_time_df['sport'].unique()
plot_size_x, plot_size_y = 5, 5
figsize_x, figsize_y = (plot_size_x + 0.5) * 4 +3, (plot_size_y + 1) * 13 + 1


nrows, ncols = 13, 4
a = fig.subplots_adjust(hspace = 1, wspace = 1)
fig, axs = plt.subplots(nrows=nrows, ncols=ncols, figsize=(figsize_x, figsize_y))
print('\nГрафик распределения времени начала тренировки по видам спорта с разбивкой по полу:')
a = plt.setp(axs, xticks=[0, 4, 8, 12, 16, 20])
for index, sport in enumerate(activities):
    row_index, col_index = divmod(index, ncols)
    male_start_time_list = start_time_df[(start_time_df.sport == sport) &
                                         (start_time_df.gender == 'male')]['workout_start_time']
    female_start_time_list = start_time_df[(start_time_df.sport == sport) &
                                         (start_time_df.gender == 'female')]['workout_start_time']
    unknown_start_time_list = start_time_df[(start_time_df.sport == sport) &
                                         (start_time_df.gender == 'unknown')]['workout_start_time']
    if len(male_start_time_list) > 0:
        male_dist = axs[row_index, col_index].hist(male_start_time_list,
                                    bins = 12, alpha=0.5, label='male', range=(0, 23))
    if len(female_start_time_list) > 0:
        female_dist = axs[row_index, col_index].hist(female_start_time_list,
                                    bins = 12, alpha=0.5, label='female', range=(0, 23))
    if len(unknown_start_time_list) > 0:
        unknown_dist = axs[row_index, col_index].hist(unknown_start_time_list,
                                    bins = 12, alpha=0.5, label = 'unknown', range=(0, 23))
    b= axs[row_index, col_index].set_title('Activitiy: ' + sport, fontsize='small')
    a = axs[row_index, col_index].legend(loc="upper left", fontsize='small')
    a = plt.setp(axs[row_index, col_index].get_xticklabels(), fontsize='small')

for i in range(1,4):
    x = axs[12, i].set_visible(False)
a = fig.tight_layout()
z = fig.text(0.5, 1, 'Распределение времени начала тренировки (часы) по видам спорта'
             , ha='center', va='top', transform=fig.transFigure)
y = fig.text(0.5, 0.01, 'Начало тренировки час в день (час)'
             , ha='center', va='bottom', transform=fig.transFigure)
z = fig.text(0.02, 0.5, 'Частота (количество)', va='center', rotation='vertical');
```

Графики:



Смотрим глубже на информацию на уровне строки:

```
[58] stat_list = ['min', '25th percentile', 'mean', '95th percentile', 'max', 'stdev']
     heart_rate_statistic_df = retrieve_array_column_stat_df(df, column_name='heart_rate', stat_list=stat_list)
```

Из-за огромного количества пользователей и количества тренировок мы случайным образом выбрали до x количества пользователей каждого пола (например, 5) и до y тренировок по каждому типу активности (например, 10).

```
# Вспомогательная функция, которая помогает отбирать данные
def sampling_data(max_users_per_gender, max_workouts_per_sport):
    '''
          max_users_per_gender: максимальное количество пользователей, выбираемых случайным образом для каждого пола.
          max_workouts_per_sport: максимальное количество занятий, которые можно выбрать для каждого вида спорта.
          (виды спорта, существующие у выбранных пользователей)
    '''
    # Получение уникального списка идентификаторов пользователя и пола для выборки.
    users_genders = df.select('userId', 'gender').distinct().toPandas()
    #Выбор до трех идентификаторов пользователей каждого пола из уникального списка идентификаторов пользователей.
    random_x_users_per_gender = users_genders.groupby('gender')['userId'].apply(
                lambda s: s.sample(min(len(s), max_users_per_gender))
    )

    # Применение фильтра к pyspark dataframe для выборки.
    samples_by_gender = df.where(df.userId.isin(list(random_x_users_per_gender)))

    # Генерация уникальных идентификаторов занятий и списков видов спорта из выборочного набора данных.
    workout_sports = samples_by_gender.select('id', 'sport').distinct().toPandas()
    # выбор до 10 идентификаторов занятий для каждого вида спорта.
    random_y_workouts_per_sport = workout_sports.groupby('sport')['id'].apply(
        lambda s: s.sample(min(len(s), max_workouts_per_sport))
    )

    # фильтр к выборочному набору данных, чтобы продолжить сокращать количество тренировок для каждого типа активности.
    samples_by_gender_and_sport = samples_by_gender.where(df.id.isin(list(random_y_workouts_per_sport)))
    return samples_by_gender_and_sport
```

```
# Использование 2 переменных для определения критериев выборки:
# максимальное количество пользователей по полу и максимальное количество тренировок по виду спорта
max_users_per_gender, max_workouts_per_sport = 20, 15

# Сбор набора выборочных данных в Pandas для использования с функциями графика.
pd_df = sampling_data(max_users_per_gender, max_workouts_per_sport).toPandas()
print('\nОбзор выборочных данных (только строковые и числовые столбцы)):')
pd_df.describe()
```

```
Обзор выборочных данных (только строковые и числовые столбцы)):
```

	id	userId	PerWorkoutRecordCount	workout_start_time	duration
count	1.920000e+02	1.920000e+02	192.000000	192.000000	192.000000
mean	3.358936e+08	6.161184e+06	417.671875	12.322917	76.101562
std	1.214009e+08	3.290179e+06	139.638357	4.739264	92.111412
min	3.265483e+07	8.011800e+04	6.000000	0.000000	0.266667
25%	2.926944e+08	4.439016e+06	368.000000	9.000000	25.633333
50%	3.259460e+08	5.211418e+06	500.000000	12.000000	50.158333
75%	4.125635e+08	8.899244e+06	500.000000	16.000000	82.604168
max	6.247964e+08	1.418553e+07	500.000000	23.000000	553.633362

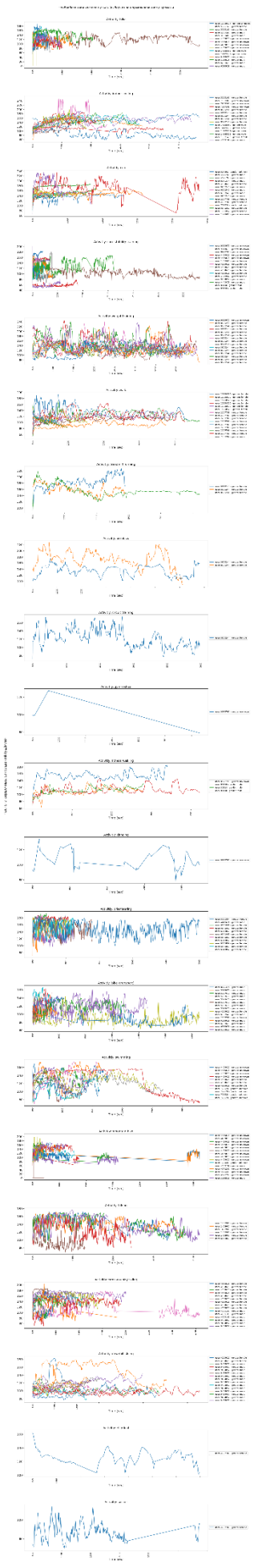
Нормализуем время для всех тренировок, рассчитав продолжительность (в секундах) каждой записи временной метки из первой записи тренировки (первый элемент datetime списка в этой тренировке).
Затем отображаем частоту сердечных сокращений в зависимости от этого нормализованного времени, группируя по видам спорта.

```
# лямбда-функция для объединения списка списков в один большой список
flattern = lambda l: set([item for sublist in l for item in sublist])

normalized_datetime_list = []
for index,data_row in pd_df.iterrows():
    min_date_time = min(data_row['date_time'])
    normalized_datetime_list.append(
        [(date_time - min_date_time).seconds for date_time in data_row['date_time']]
    )

pd_df['normalized_date_time'] = normalized_datetime_list

print('New normalized datetime (first 7 rows):')
pd_df.head(7)[['userId', 'sport', 'date_time','normalized_date_time']]

print('\nПостроение необработанного пульс (выборка) по нормированному времени:')

sport_list = pd_df['sport'].unique()
# Динамическое определение длины фигуры зависит от длины спортивного списка.
fig, axs = plt.subplots(len(sport_list), figsize=(15, 6*len(sport_list)))
subplot_adj = fig.subplots_adjust(hspace = 0.6)
plot_setp = plt.setp(axs, yticks=range(0,250,20))

for sport_index, sport in enumerate(sport_list):
    workout = pd_df[pd_df.sport == sport]
    max_time = max(flattern(workout.normalized_date_time))
    for workout_index, data_row in workout.iterrows():
        label = 'user: ' + str(data_row['userId']) + ' - gender: ' + data_row['gender']
        plot_i = axs[sport_index].plot(
            data_row['normalized_date_time'], data_row['heart_rate'], label=label
        )
    title_i = axs[sport_index].set_title('Activitiy: ' + sport, fontsize='small')
    xlabel_i = axs[sport_index].set_xlabel('Time (sec)', fontsize='small')
    xsticklabels_i = axs[sport_index].set_xticklabels(
        range(0, max_time, 500),rotation = 'vertical', fontsize=9
    )
    ysticklabels_i = axs[sport_index].set_yticklabels(range(0,250,20),fontsize='small')
    legend_i = axs[sport_index].legend(
        loc='center left', bbox_to_anchor=(1.0, 0.5), prop={'size': 9}
    )

x_label = fig.text(0.04, 0.5, 'Частота сердечных сокращений (уд/мин)', va='center', rotation='vertical')
chart_title = fig.text(0.5, 1.3, 'Необработанная частота пульса (выборка) по нормализованному времени',
           ha='center', va='center', fontsize='small', transform=axs[0].transAxes)
```

Вывод и графики:

New normalized datetime (first 7 rows):

	userId	sport	date_time	normalized_date_time
0	11889307	bike	[2013-10-01 11:42:14, 2013-10-01 11:42:18, 201...	[0, 4, 5, 6, 9, 11, 14, 16, 17, 22, 29, 32, 34...
1	5325166	indoor cycling	[2014-12-21 14:22:16, 2014-12-21 14:22:17, 201...	[0, 1, 6, 10, 41, 93, 96, 102, 103, 108, 146, ...
2	5325166	bike	[2014-08-28 19:01:28, 2014-08-28 19:01:30, 201...	[0, 2, 6, 9, 12, 15, 19, 25, 28, 31, 34, 37, 4...
3	5325166	bike	[2014-02-23 13:54:10, 2014-02-23 13:54:24, 201...	[0, 14, 29, 49, 74, 94, 107, 128, 181, 204, 21...
4	897592	run	[2014-04-29 15:55:08, 2014-04-29 15:55:12, 201...	[0, 4, 7, 13, 18, 23, 34, 43, 49, 61, 72, 80, ...
5	9613679	core stability training	[2013-05-07 10:12:32, 2013-05-07 10:12:34, 201...	[0, 2, 4, 5, 6, 8, 13, 14, 16, 17, 19, 22, 23,...
6	9613679	weight training	[2013-04-29 17:30:42, 2013-04-29 17:30:44, 201...	[0, 2, 5, 12, 22, 31, 40, 43, 53, 62, 72, 78, ...



Перемещения во время тренировки

У нас будет некоторая визуализация в трех столбцах с информацией о смещении/геометрии (longitude,latitude и altitude). Поскольку расположение каждого пользователя и тренировки отличается друг от друга, мы отображаем только несколько отдельных тренировок на 3D-графиках, чтобы просмотреть маршрут тренировки.

```
[62] pd_df_small = sampling_data(max_users_per_gender=2, max_workouts_per_sport=2).toPandas()
     print('Sampled data (2 user, 2 workouts per sport):')
     pd_df_small[['userId', 'gender','sport','id', 'workout_start_time'
                  ,'PerWorkoutRecordCount', 'duration', 'longitude', 'latitude', 'altitude']].describe()
```

```
Sampled data (2 user, 2 workouts per sport):
```

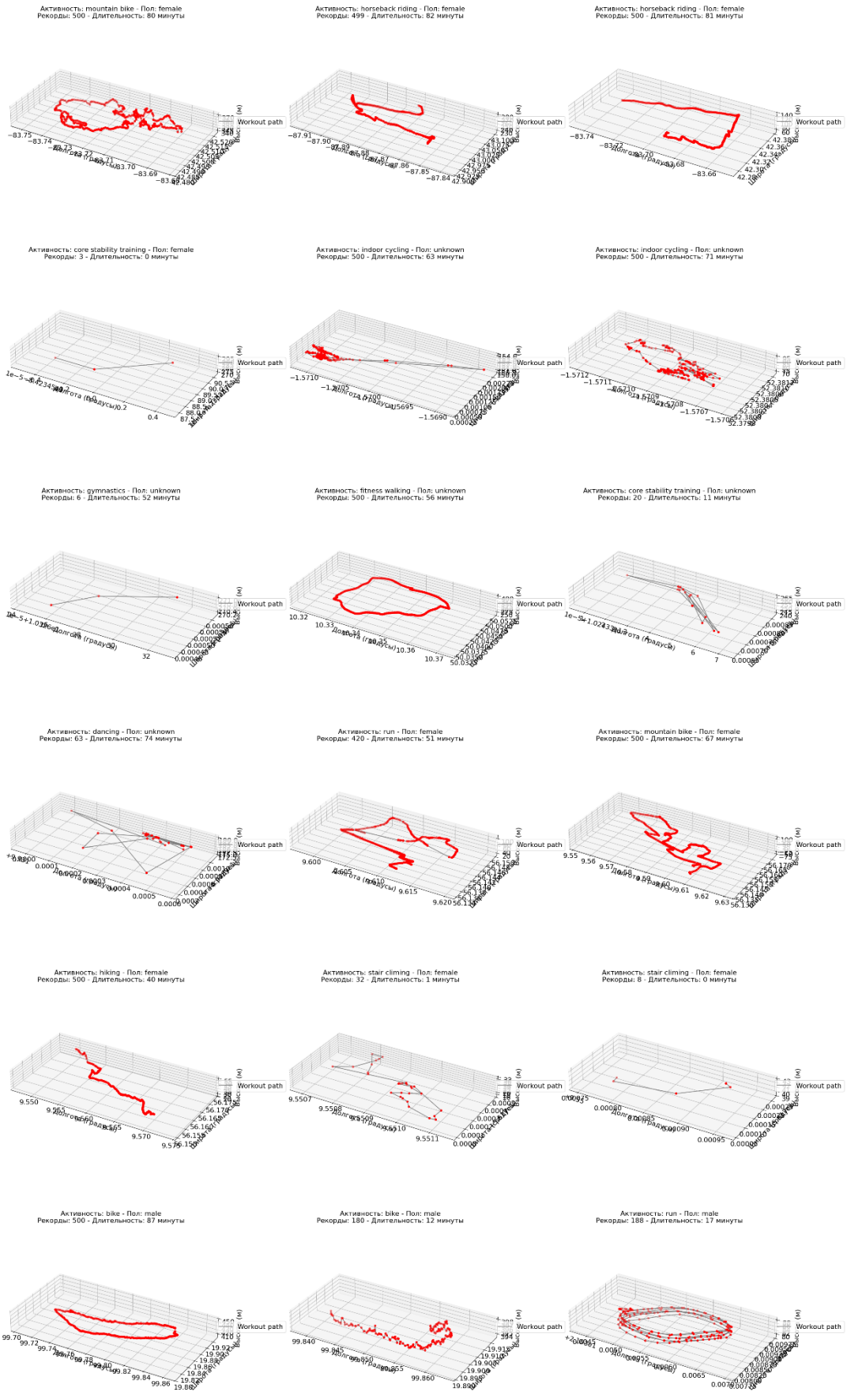
	userId	id	workout_start_time	PerWorkoutRecordCount	duration
count	1.800000e+01	1.800000e+01	18.000000	18.000000	18.000000
mean	6.936093e+06	3.349032e+08	12.777778	301.055556	47.405560
std	3.826614e+06	1.087948e+08	4.672266	225.861351	31.784660
min	1.878236e+06	8.207335e+07	5.000000	3.000000	0.050000
25%	3.014251e+06	2.920013e+08	9.250000	39.750000	13.766666
50%	7.189799e+06	3.259566e+08	12.000000	459.500000	54.650000
75%	8.972795e+06	3.510963e+08	16.750000	500.000000	73.908335
max	1.411184e+07	5.539021e+08	20.000000	500.000000	87.199997

```python
import matplotlib.pyplot as plt
import numpy as np
import math
import matplotlib.ticker as mtick
import numpy as np  # To use numpy functions for standard deviation

def get_fixed_mins_maxs(mins, maxs):
    deltas = (maxs - mins) / 12.
    mins = mins + deltas / 4.
    maxs = maxs - deltas / 4.
    return [mins, maxs]

workout_count = pd_df_small.shape[0]
ncols = 3
nrows = math.ceil(workout_count / ncols)

fig = plt.figure(figsize=(8 * (ncols + 0.5), 8 * nrows))
fig.subplots_adjust(hspace=0.2, wspace=0.5)

print('Построение траекторий тренировки в виде 3D для каждой тренировки:')
for row_index, row in pd_df_small.iterrows():
    min_long = min(row['longitude']) - np.std(row['longitude'])  # Use numpy.std()
    max_long = max(row['longitude']) + np.std(row['longitude'])  # Use numpy.std()
    minmax_long = get_fixed_mins_maxs(min_long, max_long)

    min_lat = min(row['latitude']) - np.std(row['latitude'])  # Use numpy.std()
    max_lat = max(row['latitude']) + np.std(row['latitude'])  # Use numpy.std()
    minmax_lat = get_fixed_mins_maxs(min_lat, max_lat)

    min_alt = min(row['altitude']) - np.std(row['altitude'])  # Use numpy.std()
    max_alt = max(row['altitude']) + np.std(row['altitude'])  # Use numpy.std()
    minmax_alt = get_fixed_mins_maxs(min_alt, max_alt)

    ax = fig.add_subplot(nrows, ncols, row_index + 1, projection='3d')

    title = 'Активность: ' + row['sport'] + ' - Пол: ' + row['gender'] + \
            '\nРекорды: ' + str(int(row['PerWorkoutRecordCount'])) + \
            ' - Длительность: ' + str(int(row['duration'])) + ' минуты'
    ax.set_title(title, fontsize=16)

    # Scatter plot for points
    scatter = ax.scatter(row['longitude'], row['latitude'], row['altitude'], c='r', marker='o')

    # Plot the workout path in 3D
    plot = ax.plot3D(row['longitude'], row['latitude'], row['altitude'], c='gray', label='Workout path')

    # Labels for the axes
    ax.set_xlabel('Долгота (градусы)', fontsize=16)
    ax.set_ylabel('Широта (градусы)', fontsize=16)
    ax.set_zlabel('Высота (м)', fontsize=16, rotation=0)
```

```python
    # Set font size for ticks (accessing `label1` for the primary ticks)
    for t in ax.xaxis.get_major_ticks():
        t.label1.set_fontsize(16)
    for t in ax.yaxis.get_major_ticks():
        t.label1.set_fontsize(16)
    for t in ax.zaxis.get_major_ticks():
        t.label1.set_fontsize(16)

    # Add legends
    ax.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))

    # Set limits for each axis
    ax.set_xlim(minmax_long)
    ax.set_ylim(minmax_lat)
    if minmax_alt[0] != minmax_alt[1]:
        ax.set_zlim(minmax_alt)

    # Set aspect ratio for the plot (adjust these values as needed)
    ax.set_box_aspect([4, 2, 0.5])  # aspect ratio for x, y, z axes

    # Remove spines
    for spine in ax.spines.values():
        spine.set_visible(False)

# Add a global title
fig.text(0.5, 1.02, "Маршрут тренировки (долгота/широта/высота)", ha='center', va='top', fontsize=18)

plt.tight_layout()
plt.show();
```

Графики:

Маршрут тренировки (долгота/широта/высота)



Индивидуальные задания:

Вариант – 23

Задание 1 (Интерпретация) - Оцените названия столбцов в исходном датасете (раздел 2). Насколько они понятны для бизнес-пользователя? Есть ли столбцы, требующие переименования или дополнительного описания?

Задание 2 (Интерпретация) - В разделе 6 рассчитывается процент пользователей, занимающихся более чем 1 видом спорта. Как бы вы интерпретировали этот показатель для бизнеса? Это высокий или низкий процент?

Задание 3 (Интерпретация) - Если бы у вас были данные о погоде во время тренировки, как бы вы могли их использовать совместно с данными о duration или heart_rate?

Задание 4 (Практика PySpark/Python) - Напишите код PySpark, чтобы посчитать количество строк в датафрейме df до и после удаления дубликатов по столбцу id (уникальный ID тренировки).

Задание 5 (MLlib Концепция) - Почему удаление дубликатов важно перед обучением моделей ML в Spark MLlib? Как дубликаты могут повлиять на качество модели и оценку ее производительности?

Задание 1.

Названия столбцов:

Обзор столбцов

	Column Name	Data type
0	altitude	array<double>
1	gender	string
2	heart_rate	array<bigint>
3	id	bigint
4	latitude	array<double>
5	longitude	array<double>
6	speed	array<double>
7	sport	string
8	timestamp	array<bigint>
9	url	string
10	userId	bigint

Понятные:

- gender – пол пользователя. Понятное название.
- sport – вид спорта. Интуитивно ясно.
- url – ссылка на запись или ресурс. Общепринятое обозначение.
- userId – уникальный идентификатор пользователя. Часто используется в бизнес-аналитике.

Требуют доп информации:

- id – слишком общее название.
- altitude
- latitude
- longitude
- speed
- timestamp
- heart_rate.

Все эти поля являются массивами, обозначения полей понятны, но следует добавить единицы измерения, например «altitude_m» или «heart_rate_bpm».

Задание 2.

Общее количество пользователей (из раздела 2):

Общая сводка набора данных о пользователях, тренировках и количестве записей (предварительная фильтрация):

	Users count	Activity types count	Workouts count	Total records count
0	1,104	49	253,020	111,541,956

Количество пользователей, участвующих более чем в одном виде спорта:

	count	mean	std	min	25%	50%	75%	max
userId	822	4860464	3953412	69	1609606	3730685	7554937	15481421
Sports count	822	3	2	2	2	3	5	16

Рассчитаем процентное соотношение:
Процент пользователей= 822/1104 × 100 ≈ 74.45%

Это высокий процент , который указывает на то, что большинство пользователей платформы имеют разнообразные интересы и используют сервис для разных видов физической активности.

Задание 3.

Приведу по 3 примера:

duration:

1. Анализ влияния дождя на продолжительность тренировок:
    - Понять, как осадки влияют на желание пользователей заниматься на улице.
    - Например: "Во время дождя средняя продолжительность бега снижается на 20%".
2. Сравнение сезонной активности (весна vs лето vs зима):
    - Определить, в какие сезоны пользователи тренируются дольше.
    - Это помогает планировать маркетинговые кампании и рекомендации.
3. Прогнозирование продолжительности тренировки на основе прогноза погоды:
    - Строить модель, которая предсказывает, сколько времени пользователь может потратить на тренировку завтра, исходя из погоды.

heart_rate:

1. **Оценка влияния температуры на пульс во время тренировки:**
    - **Выявить, увеличивается ли пульс при высокой температуре даже при одинаковой нагрузке.**
    - **Например: "При температуре выше 30°C пульс растёт на 10% по сравнению с комфортными условиями".**
2. **Изучение влияния влажности на восстановление после тренировки:**
    - **Проверить, как влажный воздух влияет на восстановление пульса после окончания активности.**
3. **Персонализированные уведомления о возможном перегреве или переохлаждении:**
    - **На основе погоды и текущего пульса отправлять пользователям рекомендации:**
      "Сегодня очень жарко. Ваш пульс повышен — возможно, стоит снизить интенсивность."

Задание 4.

Напишите код PySpark, чтобы посчитать количество строк в датафрейме df до и после удаления дубликатов по столбцу id (уникальный ID тренировки).

```
# 1. Количество строк до удаления дубликатов
count_before = df.count()
print(f"Количество строк до удаления дубликатов: {count_before}")

# 2. Удаление дубликатов по столбцу 'id'
df_deduplicated = df.dropDuplicates(['id'])

# 3. Количество строк после удаления дубликатов
count_after = df_deduplicated.count()
print(f"Количество строк после удаления дубликатов: {count_after}")

# 4. Разница
print(f"Удалено дубликатов: {count_before - count_after}")
```

```
Количество строк до удаления дубликатов: 253020
Количество строк после удаления дубликатов: 253020
Удалено дубликатов: 0
```

На всякий случай проверил строки с дубликатами отдельно

```
[67] from pyspark.sql import Window
     from pyspark.sql import functions as F

     # Создаем окно по id
     window_spec = Window.partitionBy("id").orderBy("id")

     # Добавляем колонку с количеством повторений
     df_with_count = df.withColumn("count", F.count("id").over(window_spec))

     # Фильтруем только те строки, где id встречается более одного раза
     duplicates_df = df_with_count.filter(F.col("count") > 1)

     # Выводим количество дублирующихся записей
     duplicate_count = duplicates_df.count()
     print(f"Количество строк с дубликатами: {duplicate_count}")
```

59 сек.

```
Количество строк с дубликатами: 0
```

Задание 5.

Удаление дубликатов важно, потому что они искажают распределение данных, ведут к переобучению модели и завышенной оценке качества на тесте. Дубликаты нарушают независимость наблюдений, что делает результаты модели некорректными и менее обобщающими.

Вывод:

Работа демонстрирует практическое применение инструментов Apache Spark и PySpark для обработки, анализа и подготовки больших данных с последующей подготовкой к задачам машинного обучения. Полученные результаты могут быть использованы для улучшения пользовательского опыта, персонализации рекомендаций и анализа физической активности.