Введение в анализ данных

Домашнее задание **3.** Сбор и анализ данных.

Правила, прочитайте внимательно:

- Выполненную работу нужно отправить телеграм-боту `@miptstats_ds22_bot`. Для начала работы с ботом каждый раз отправляйте `/start`. Работы, присланные иным способом, не принимаются.
- Дедлайн см. в боте. После дедлайна работы не принимаются кроме случаев наличия уважительной причины.
- Прислать нужно ноутбук в формате **ipynb**.
- Телеграм не разрешает боту получать файлы более **20 Мб**. Если ваше решение весит больше, *заранее* разделите ноутбук на несколько.
- Будьте внимательны при работе со сбором данных. Ответственность за корректность ваших действий лежит на вас. Не нагружайте сервера, *делайте паузы между запросами*. Как следствие, начинайте выполнять задание заранее. **Если вас где-то забаннили и т.п., то это не является уважительной причиной продления дедлайна.**
- Выполнять задание необходимо полностью самостоятельно. **При обнаружении списывания все участники списывания будут сдавать устный зачет.**
- Решения, размещенные на каких-либо интернет-ресурсах, не принимаются. Кроме того, публикация решения в открытом доступе может быть приравнена к предоставлении возможности списать.
- Для выполнения задания используйте этот ноутбук в качестве основы, ничего не удаляя из него. Можно добавлять необходимое количество ячеек.
- Комментарии к решению пишите в **markdown**-ячейках.
- Выполнение задания (ход решения, выводы и пр.) должно быть осуществлено на русском языке.
- Если код будет не понятен проверяющему, оценка может быть снижена.
- Никакой код из данного задания при проверке запускаться не будет. *Если код студента не выполнен, недописан и т.д., то он не оценивается.*

Перед выполнением задания посмотрите презентацию по выполнению и оформлению домашних заданий с занятия 2.

Баллы за задание:

Легкая часть (достаточно на "хор"):

- Задача **1** — **50** баллов

Сложная часть (необходимо на "отл"):

- Задача **2** — **80** баллов
- Задача **3** — **100** баллов

In [47]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style='whitegrid', font_scale=1.3, palette='Set2')
%matplotlib inline
```

Легкая часть

Задача 1.

На занятии мы уже начинали работать с датасетом Титаник. Сейчас ваша задача — на основе простого анализа предоставленных данных получить некоторое приближенное правило определения, выжил пассажир или нет.

1. Подготовка данных

Загрузите данные с помощью `pandas`.

In [48]:

```
data = pd.read_csv('titanik.csv', index_col=0)
```

Разделите данные по строкам на две части случайным образом в соотношении **7:3**. Первую часть мы будем называть *обучающей*, а вторую — *тестовой*.

In [49]:

```
def train_test_split(df, frac=0.7):
    test = df.sample(frac=frac, axis=0)
    train = df.drop(index=test.index)
    return train, test

test_data, train_data = train_test_split(data)
```

Из каждой части оставим несколько признаков, с которыми мы будем работать, а также отдельно — целевой признак. Примените к обеим частям таблицы функцию ниже

In [50]:

```
features_columns = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']
target_column = 'Survived'  # Целевой признак

def get_features_and_target(data):
    all_data = data[[*features_columns, target_column]]
    features = data[features_columns]
    target = data[target_column]
    return features, target, all_data
```

In [51]:

```
categories, result, train = get_features_and_target(train_data)
```

2. Исследование

Внимание. Эта часть задачи должна выполняться *исключительно на обучающих данных*. За использование тестовых данных решение не будет оценено.

Проведите визуальный анализ данных чтобы понять, как различные признаки влияют на целевой. Исследовать можно не целиком обучающие данные, а разделить их на две части по одному из признаков, а далее рассматривать каждую часть отдельно.

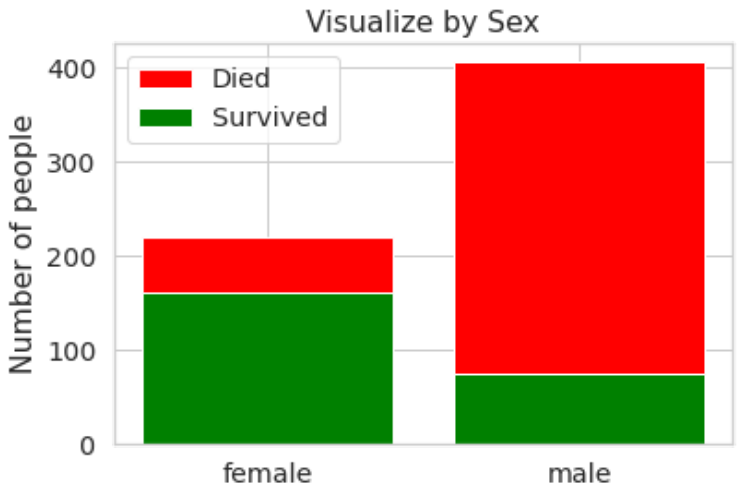
Не забывайте про информативность и эстетичность графиков.

In [52]:

```
def visualize(feature):
    plt.title('Visualize by ' + feature)
    survived = train[train['Survived'] == 1][feature].value_counts().sort_index()
    died = train[train['Survived'] == 0][feature].value_counts().sort_index()
    plt.ylabel("Number of people")
    all = survived + died
    plt.bar(all.index, all, color='red')
    plt.bar(survived.index, survived, color='green')
    plt.legend(['Died', 'Survived'])
```

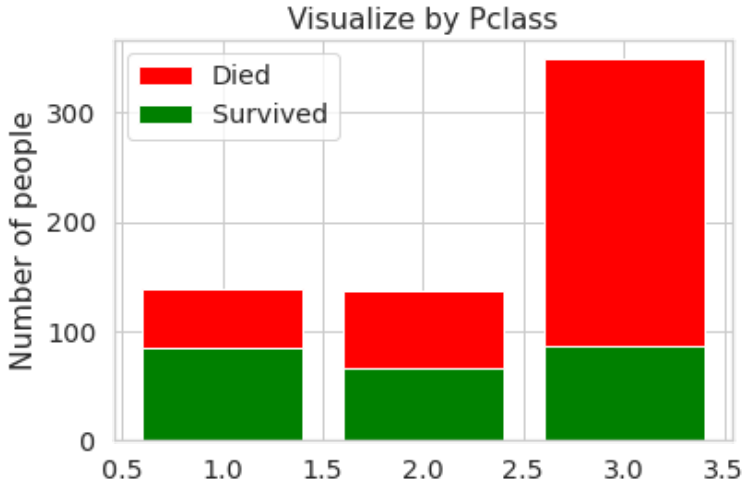
In [53]:

```
visualize('Sex')
```



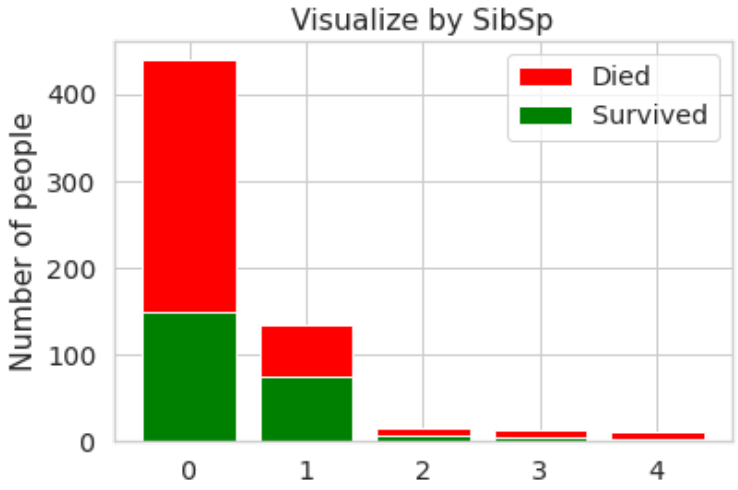
In [54]:

```
visualize('Pclass')
```



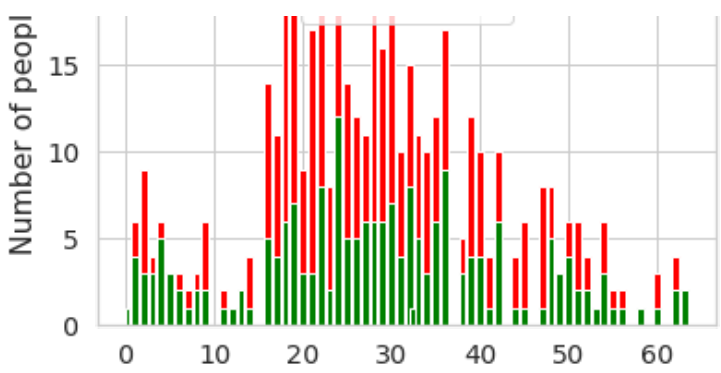
In [55]:

```
visualize('SibSp')
```



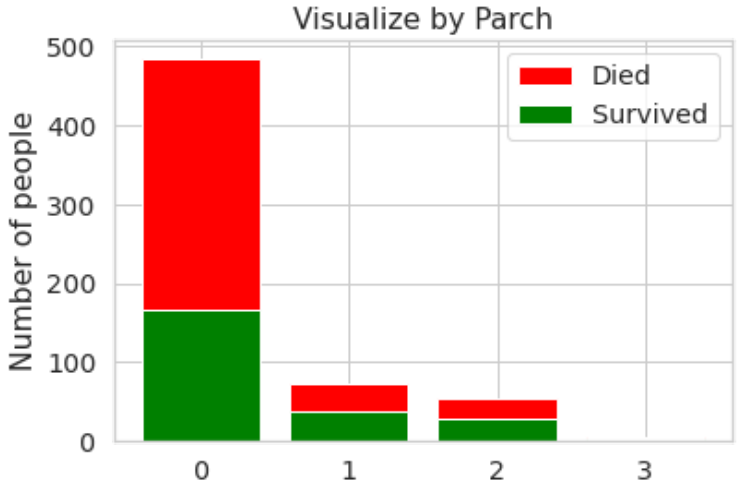
In [56]:

```
visualize('Age')
```



In [57]:

```
visualize('Parch')
```



3. Классификация

На основе проведенного исследования предложите два разных правила в виде решающего дерева, пример которого можете посмотреть в презентации с первой лекции (классификация котиков).

В данной задаче достаточно, если первое дерево будет иметь максимальную глубину **2**, а второе — глубину **1**, и при этом не является поддеревом первого. *Примечание*: дерево из одного листа имеет глубину **0**.

Дерево 1:

Посмотрим на график по параметру **"Sex"**. Нетрудно видеть, что среди мужчин процент погибших гораздо выше, чем процент выживших. Для женщин, наоборот, процент погибших оказался ниже. Таким образом, построим модель решающего дерева: каждый мужчина не выжил, а женщина выжила. **Дерево 2:**

Посмотрим на графики по параметрам **"Age"** и **"Pclass"**. Нетрудно видеть, что среди данных с **"Age"** от **20** до **40** высокий процент погибших. Также заметим, что высокий процент погибших среди людей **3** класса. Предположим, что люди погибли, если их возраст от **20** до **40**, а класс **1** или **2**. Иначе выжили.

Таким образом наше дерево решений будет иметь следующий вид:

- для людей с **"Age"** из **[20, 40]**, для людей из **3** класса предскажем **0**
- для людей с **"Age"** не из **[20, 40]** и **"Pclass"** из **{1, 2}** предскажем **1**
- для людей с **"Age"** из **[20, 40]** и **"Pclass"** не из **{1, 2}** предскажем **0**
- для людей с **"Age"** не из **[20, 40]** и **"Pclass"** из **{1, 2}** предскажем **1**

Реализуйте полученные деревья по шаблону

In [58]:

```
def detector1(x: pd.Series):
    if x['Sex'] == 'male':
        return 0
    else:
        return 1
```

```
def detector2(x: pd.DataFrame):
    if 20 < x['Age'] < 40:
        if x['Pclass'] == 1:
            return 1
        elif x['Pclass'] == 2:
            return 0
        else:
            return 0
    else:
        if x['Pclass'] == 1:
            return 1
        elif x['Pclass'] == 2:
            return 1
        else:
            return 0

def tree(features, detector):
    predicted = features.copy()

    predicted['Survived'] = predicted.apply(lambda x: detector(x), axis=1)
    return predicted
```

4. Качество

Вспомним, что у нас имеется тестовая часть выборки. Самое время ее использовать для того, чтобы оценить, насколько хорошими получились деревья. Предложите какой-нибудь критерий качества.

Замечание. Не стоит пытаться искать, какие критерии существуют. В данном случае легко предложить адекватный критерий. Стандартные критерии мы разберем позже, но для начала лучше подумать самостоятельно.

Критерий:

Определим критерий как количество верных предсказаний, деленное на количество всех предсказаний.

Реализуйте данный критерий по шаблону

In [59]:

```
def criteria(target, predicted):
    return 1 - ((predicted['Survived'] - target['Survived'])**2).mean()
```

Посчитайте качество полученных ранее деревьев

Примечание. Полученные значения не влияют на оценку по задание. Оценивается только корректность и обоснованность решения, а также графики и выводы.

In [60]:

```
criteria(test_data, tree(test_data, detector1))
```

Out[60]:

```
0.7865168539325843
```

In [61]:

```
criteria(test_data, tree(test_data, detector2))
```

Out[61]:

```
0.7078651685393258
```

5. Выводы

При анализе данных были визуализированы количество погибших и выживших в зависимости от значимых характеристик: **'Pclass', 'Sex', 'Age', 'SibSp', 'Parch'**. Исходя из наглядной оценки процентного соотношения

выживших при каждом параметре были построены **2** модели решающих деревьев. Глубины **1** и **2**. Был также предложен и реализован критерий качества. Им были оценены точности обеих моделей. Исследование показало, что точность модели глубины **1** оказалась выше, чем у модели глубины **2**. Так получилось, потому что параметр **"Sex"** оказался наиболее влиятельным, что также видно по соответствующему графику и соответствует реальности.

Сложная часть

Задача 2

Yelp — веб-сайт для поиска на местном рынке услуг, например ресторанов или парикмахерских, с возможностью добавлять и просматривать рейтинги и обзоры этих услуг. Для популярных бизнесов имеются сотни обзоров. Для обозревателей на сайте предусмотрены элементы социальной сети.

Вам предоставляется следующая информация о компаниях на **Yelp:**

Файл `yelp_business.csv`:

- `business_id` — уникальный идентификатор компании;
- `name` — имя компании;
- `address`, `city`, `state` — месторасположении компании;
- `latitude`, `longitude` — географические координаты;
- `categories` — категории услуг компании.

Файл `yelp_review.csv`, содержащий оценки пользователей:

- `business_id` — идентификатор компании, соответствующий файлу `yelp_business.csv`;
- `stars` — поставленная пользователем оценка от **1** до **5**.

В целях сокращения объема файла, текстовые отзывы пользователей не были включены.

Оригинальную версию датасета в формате `json` можно посмотреть по ссылке.

Что нужно сделать:

- Найти город с наибольшим количеством компаний;
- Для этого города определить районы с наиболее качественными услугами. Пример с несколько другой задачей.
- А также найти рестораны с наилучшими отзывами.

Город с наибольшим количеством компаний

Загрузите данные из файла `yelp_business.csv` с помощью функции `pd.read_csv`. Посмотрите на первые несколько строк с помощью метода `head`.

In [ ]:

```
...
```

Найдите пять городов, по которым присутствует информация о наибольшем количестве компаний. В таблице должен быть указан город (название) и количество компаний в этом городе.

Подсказка. Для выполнения стоит воспользоваться методами `groupby`, `count`, `sort_values`, `head`.

In [ ]:

```
...
```

Пусть `N` — город с наибольшим количеством компаний. Оставьте в таблице только записи, соответствующие городу `N`. Нанесите все эти компании на график, в котором по оси x отметьте долготу, а по оси y — широту.

In [ ]:

```
...
```

Сам город находится в сгустке точек. Есть какие-то компании, которые приписаны к этому городу, но находятся далеко от него. Избавьтесь от них, подобрав некоторые границы значений широты и долготы. Изобразите все компании на новом графике.

На этом графике должны выделяться некоторые улицы. Откройте карту города `N` и сравните ее с построенным графиком.

Замечание. Подгружать карту города в качестве фона графика мы научимся чуть позже.

In [ ]:

```
...
```

Оценки компаний

Для выполнения задания нужно посчитать среднюю оценку каждой компании, а также количество выставленных оценок.

Загрузите таблицу оценок `yelp_review.csv`.

In [ ]:

```
...
```

В подгруженной таблице оценок оставьте только компании города `N`. Для этого установите значения `business_id` в качестве индекса у таблицы оценок и воспользуйтесь методом `loc`.

Подсказка. Чтобы индекс снова сделать полем таблицы, можно воспользоваться методом `reset_index`.

In [ ]:

```
...
```

Теперь посчитайте среднюю оценку каждой компании, а также количество выставленных компании оценок.

Подсказка. Помочь в этом могут функции `groupby` и `aggregate([np.mean, np.size])`.

In [ ]:

```
...
```

Назовите колонки таблицы красивыми именами, изменив `<имя таблицы>.columns`, после чего напечатайте несколько строк полученной таблицы. Красивые имена — то, что будет понятно простому человеку при чтении ваших результатов. Как именно их назвать — задача аналитика, то есть в данном случае ваша :)

In [ ]:

```
...
```

Соедините две полученные ранее таблицы по компаниям города `N` в одну. Для этого сначала установите поле `business_id` в качестве индекса в обеих таблицах с помощью `set_index`. В одной из них это уже должно было быть сделано. В полученной таблице должны получится поля `latitude`, `longitude`, `categories`, `name`, `stars`, `count`.

Подсказка. Соединение таблиц можно выполнить с помощью `join`. Индексы у этих таблиц одинаковые, так что тип джойна не имеет значения.

In [ ]:

```
...
```

Изобразите все компании на графике, раскрасив точку в цвет, оттенок которого соответствует средней оценке компании. Прозрачность точки выставляйте не более 0.3.

In [ ]:

```
...
```

Чтобы получить районы города, то есть разделить город на "клетки", округлите значения широты и долготы, подобрав оптимальный размер района.

Подсказка. Например, можно сделать так `np.round(долгота*4, decimals=1)*0.25`.

In [ ]:

```
...
```

Для получения средней оценки компании по району постройте сводную таблицу при помощи `pd.pivot_table`, взяв в качестве индексов и колонок округленные широту и долготу, а в качестве значений — оценки. Аггрегирующей функцией является среднее.

Изобразите полученную таблицу при помощи `sns.heatmap`.

In [ ]:

```
...
```

Полученный график имеет ряд недостатков. Во-первых, не очень правильно судить о районе, если в нем мало компаний. Во-вторых, на графике цветовая гамма автоматически подстроилась под минимальное и максимальное значения оценки.

Почему эти недостатки могут быть существенными?

Ответ: <...>

Оставьте районы, в которых имеется информация о не менее 30 компаний. Постройте новый график районов, использовав параметры `vmin` и `vmax` у функции `sns.heatmap`.

In [ ]:

```
...
```

Сравните полученный график с предыдущим и сделайте вывод.

Вывод: <...>

Рестораны

Будем считать компанию рестораном, если в поле `categories` *содержится* слово `restaurant`. Обратите внимание, что в анализе данных часто нет четкого формата данных. Например, данное слово может быть написано как с большой буквы, так и с маленькой; может как разделяться `;`, так и не разделяться. При возникновении недопонимания стоит посмотреть данные.

Составьте таблицу, в которой будет информация о всех ресторанах города `N`, для которых имеется не менее 5 отзывов. Далее постройте **heatmap-**график районов, в котором каждому району сопоставьте среднюю оценку по ресторанам этого района. Рассматривайте только те районы, в которых есть не менее 10 ресторанов, для каждого из которых есть не менее 5 отзывов.

In [ ]:

```
...
```

Чем полезны ограничения снизу на количество отзывов для ресторана и количество ресторанов в районе?

Ответ: <...>

Кот Василий из города `N` очень придирчив к выбору ресторана. Он доверяет только ресторанам с высоким рейтингом, который основывается на большом количестве отзывов. Напечатайте в виде таблицы информацию 10 ресторанах с самым большим рейтингом в порядке убывания рейтинга. Для каждого из этих ресторанов должно быть не менее 50 отзывов. По каждому ресторану необходимо вывести следующую информации: название ресторана, средняя оценка, количество отзывов, географические координаты, категории.

In [ ]:

```
...
```

Нанесите на карту все рестораны со средней оценкой не менее 4.7, которая посчитана по не менее 50 отзывам. Отдельным цветом отметьте **10** ресторанов, которые вы получили ранее.

In [ ]:

```
...
```

Охарактеризуйте кота Василия, а также сделайте общий вывод по задаче.

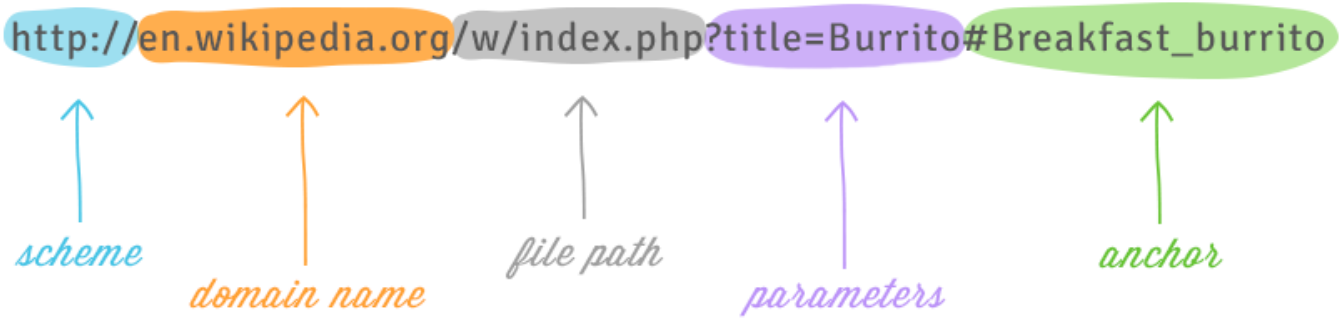
Вывод: <...>

Задача 3.

В данной задаче вам предстоит распарсить сайт, который вы выберите.

1. Каждая ссылка **(URL)** в интернете состоит из нескольких компонент:

- схема, *например,* `http` *или* `https`,
- хост, *наример,* `en.wikipedia.org`
- путь, по которому находится информация,
- параметры,
- якорь — указание фрагмента страницы.



Одна и та же ссылка может иметь несколько форм и вести на ту же самую страницу. Во время обхода сайта это надо учитывать и сохранять один уникальный урл для каждой страницы. Процедура, приводящая урл к каноничному виду, называется нормализацией. Процедуры, которые применяются к урлу для нормализации, описаны, например, на Википедии.

Установите пакет `url_normalize`, в котором есть готовая функция для нормализации.

pip install url_normalize

Прочитайте, как именно функция `url_normalize` меняет ссылку, и приведите примеры 4 урлов, которые нормализуются к одному и тому же. Примеры должны демонстрировать различные этапы процедуры нормализации. Одним из этих примеров может быть сам нормализованный урл.

In [ ]:

```
from url_normalize import url_normalize
```

In [ ]:

```
...
```

Изучите код, в котором скачиваются страницы сайта `simple.wikipedia.org` с занятия про парсинг данных и **скопируйте его в решение данной задачи**. Код, реализующий обход в ширину с фильтрацией и нормализацией урлов, приведён в виде функции.

In [ ]:

```
from urllib.parse import urlparse, urldefrag, urljoin
from urllib.request import urlopen
from bs4 import BeautifulSoup
from queue import Queue
import time
```

Код обхода в ширину:

In [ ]:

```
def load_web_pages(seed, max_downloads, filtration_function):
    '''Обходит web-страницы в ширину и загружает информацию о них.

    Принимает:
        seed (str) -- страница, с которой начинать обход.
        max_downloads (int) -- максимальное число загруженных страниц.
        filtration_function (str -> bool) -- функция, указывающая,
            стоит ли загружать страницу. Пример: is_wiki_article.

    Возвращает:
        pages_json (list) - список словарей с информацией о страницах.
    '''

    # Создаём список со страницами
    pages_json = []
```

```
    # Создаём очередь для обхода в ширину
    q = Queue()
    q.put(seed)

    already_visited = set()
    n_downloads = 0
    time_start = time.time()

    while not q.empty():
        # Нормализуем урл
        main_url = url_normalize(q.get())
        if main_url in already_visited:
            continue
        already_visited.add(main_url)
        html = download_from_the_internet(main_url)

        # Извлекаем ссылки из страницы
        children_links = extract_links_from_html(main_url, html)
        time.sleep(1)

        # Извлекаем текст страницы
        text_info = extract_text_info_from_html(html)

        # Добавляем запись в таблицу
        text_info['url'] = main_url
        pages_json.append(text_info)

        n_downloads += 1
        if n_downloads > max_downloads:
            break

        # Добавляем ещё не посещённые ссылки в очередь
        for child in children_links:
            if url_normalize(child) not in already_visited \
            and filtration_function(child):
                q.put(child)

    return pages_json
```

2. Выберите достаточно крупный сайт, который вам интересен, а также некоторую категорию страниц в нём. Поймите, с какой страницы сайта надо начать обход, чтобы обходить сайты данной категории.

*Пример: ищем статьи про **Data Science** на Википедии, начинаем со статьи **Data Dcience**.*

Укажите, что выбрали: <...>

Придумайте критерий, который по тексту из **HTML**-страницы будет определять, находится ли страница в определённой вами категории.

*Пример: статья на Википедии про **Data Science**, если в ней есть слово **"data"** или термины из статистики, теории вероятностей и анализа данных.*

Опишите критерий: <...>

Начав с выбранной страницы, скачайте не менее 500 страниц сайта. В качестве функции фильтрации возьмите функцию, которая отделяет страницы с выбранного сайта от других.

In [ ]:

```
...
```

Сделайте `pandas`-таблицу со следующей информацией:

- количество слов в тексте статьи;
- принадлежит ли страница выбранной категории.

Для последнего пункта можете воспользоваться функцией поиска слов по заданным префиксам, рассмотренной на занятии.

In [ ]:

```
...
```

Определите, какой процент скачанных страниц принадлежит выбранной категории?

In [ ]:

```
...
```

С помощью гистограмм сравните визуально распределения количества слов в статьях из выбранной категории и во всех остальных. Для этих двух распределений вычислите средние, медианы и дисперсии. Средние и медианы отобразите на графиках с гистограммами. Сделайте выводы.

In [ ]:

```
...
```

Постройте **boxplot**-графики для того, чтобы сравнить эти **2** распределения. Какой способ сравнения кажется вам более удобным?

In [ ]:

```
...
```

3. Визуализируйте скачанные страницы сайта. В качестве значений по осям возьмите количество слов на странице и количество ссылок на ней. Цветом обозначьте принадлежность выбранной вами категории.

In [ ]:

```
...
```

4. Предложите функцию фильтрации **web**-страниц, которую нужно подать в `load_web_pages`, чтобы среди скачанных сайтов был больше процент страниц из выбранной вами категории.

In [ ]:

```
...
```

Скачайте снова не менее 500 страниц, но уже с новой функцией фильтрации. Добавьте следующую информацию к каждой странице:

- глубина в обходе,
- время скачивания страницы.

Для замера времени можете воспользоваться функцией `time` из пакета `time`.

In [ ]:

```
...
```

Постройте гистограмму для времени скачивания страницы.

In [ ]:

```
...
```

Предположите, каким известным вам стандартным распределением оно может быть приближено?

In [ ]:

```
...
```

Выводы: <...>