Практическая работа

Проведение экспериментов с имитационной моделью в среде Python / Simpy / Scipy / Streamlit

Библиотека Simpy обеспечивает поддержку описания и запуска дискретно-событийных моделей на Python. В библиотеке Simpy нет полноценной графической среды для построения, выполнения и составления отчетов по результатам моделирования, однако она предоставляет необходимые базовые компоненты моделирования. Для обеспечения построения графиков и визуализации модельного процесса можно в описание модели подключить известные фреймворки Python, такие как Matplotlib, Tkinter, Streamlit.

Для примера опишем процесс организации пропуска потока посетителей с входной очередью на масштабное мероприятие (концерт, ярмарка, парк аттракционов). Другими похожими процессами систем массового обслуживания, которые следуют аналогичной схеме, могут быть супермаркет, кинотеатр, железнодорожный вокзал и т.п.

В этом примере будем моделировать ситуацию входного потока, который полностью обслуживается общественным транспортом (рисунок 1): автобус на регулярной основе будет высаживать несколько посетителей, которым затем нужно будет отсканировать билеты перед входом на мероприятие. Предположим, у одних посетителей будут заранее приобретённые билеты (бейджи), в то время как другим нужно будет сначала подойти к кассам, чтобы купить билеты. Дополнительно усложним модель учетом варианта возможного прихода к кассам группами людей, имитируя семейную/групповую покупку билетов. Однако далее на контрольном пункте каждому человеку нужно будет сканировать свой билет персонально.

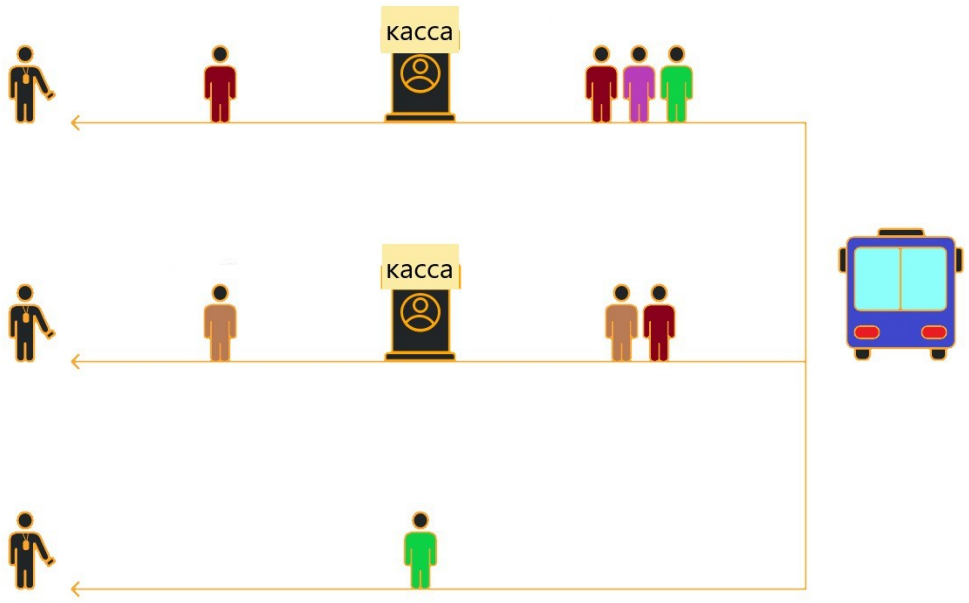


Рисунок 1. Схема движения потока участников мероприятия

Чтобы смоделировать этот процесс, нужно решить, как представить различные события, используя различные виды распределения вероятностей.

Настройки, которые отразим в нашей реализации, включают такие исходные данные:

- Автобусы прибывают в среднем 1 раз в 3 минуты;
- В каждом автобусе будут находиться от 35 до 125 посетителей, что моделируется с использованием нормального распределения (μ = 80, σ = 15);
- Посетители могут формировать группы от 1 до 4 человек, что смоделируем используя нормальное распределение (μ = 2,5, σ = 0,5), и округляя это значение до ближайшего целого числа;
- Предположим, что для 50% посетителей потребуется приобрести билеты в кассах, еще 50% придут с билетами, уже купленными заранее (онлайн);
- Посетителям требуется пройти от автобуса к кассе за время, которое составляет в среднем 1 минуту (нормальное распределение, μ = 1, σ = 0,25);
- Посетителям требуется пройти от кассы до контроля билетов за время, которое составляет около 0,5 минуты (нормальное распределение, μ = 0,5, σ = 0,1);
- Посетители по прибытии всегда выбирают самую короткую очередь, и для каждой очереди есть один продавец или контролёр;
- Для покупки билетов в кассе требуется около 1 минуты (нормальное распределение, μ = 1, σ = 0,2);
- Сканирование у контролёра занимает около 0.4 минут (нормальное распределение, μ = 0,4, σ = 0,1).

Параметрами, представляющими интерес для анализа, являются количество касс продажи билетов (SELLER_LINES) и количество контролёров билетов (SCANNER_LINES).

Для подготовки модели нам потребуется Python с дополнительными фреймворками simpy, matplotlib, scipy, pandas, streamlit. Библиотеки random, collections, math обычно входят в дистрибутив Python.

Фаза 1

Начнём разработку модели с определения базового набора процессов и ресурсов.

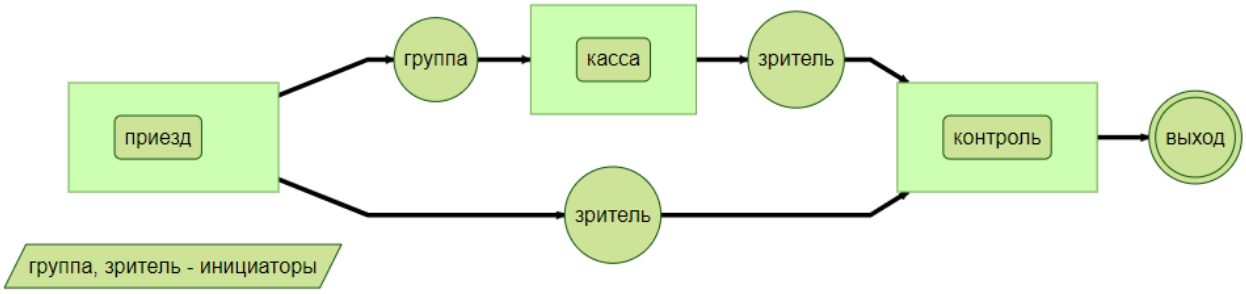


Рисунок 2. Блочная параметрическая схема логической модели

Объявляем ресурсы процесса на основе компонента ресурса из Simpy:

```
seller_lines = [simpy.Resource(env, capacity = SELLERS_PER_LINE)
for _ in range(SELLER_LINES)]
scanner_lines = [simpy.Resource(env, capacity = SCANNERS_PER_LINE)
for _ in range(SCANNER_LINES)]
```

Запишем процессы -

```
Процесс def bus_arrival() – прибытие автобуса
Процесс def purchasing_customer()  - приход клиента в кассу
Процесс def scanning_customer() – приход клиента на контроль
```

Начнем формирование модели с добавления нижерасположенного кода в программу модели.

Для этого начните новый файл (.py) в своей системе разработки (VS Code / IDLE / Anaconda).

```
# -*- coding: utf-8 -*-
import random as rd
import simpy
from collections import defaultdict

# -------------------------
#  конфигурация процесса
# -------------------------

BUS_ARRIVAL_MEAN = 3
BUS_OCCUPANCY_MEAN = 80
BUS_OCCUPANCY_STD = 15

PURCHASE_RATIO_MEAN = 0.5
PURCHASE_GROUP_SIZE_MEAN = 2.5
PURCHASE_GROUP_SIZE_STD = 0.5

TIME_TO_WALK_TO_SELLERS_MEAN = 1
TIME_TO_WALK_TO_SELLERS_STD = 0.25
TIME_TO_WALK_TO_SCANNERS_MEAN = 0.5
TIME_TO_WALK_TO_SCANNERS_STD = 0.1

SELLER_LINES = 6
SELLERS_PER_LINE = 1
SELLER_MEAN = 1
SELLER_STD = 0.2

SCANNER_LINES = 6
SCANNERS_PER_LINE = 2
SCANNER_MEAN = 0.4
SCANNER_STD = 0.1

seller_lines, scanner_lines, event_log = [],[],[]
arrivals = defaultdict(lambda: 0)
seller_waits = defaultdict(lambda: [])
scan_waits = defaultdict(lambda: [])
```

```
rd.seed(4210)

# -------------------------
# Общие параметры для сбора статистики
class Globals:
    # предварительно запишем времена прибытия автобусов и кол-во пассажиров,
    # чтобы точно воспроизводить случайности в экспериментах;
    # предусмотрим 50 событий приезда автобусов
    ARRIVALS = [rd.expovariate(1 / BUS_ARRIVAL_MEAN) for _ in range(50)]
    ON_BOARD = [abs(int(rd.gauss(BUS_OCCUPANCY_MEAN, BUS_OCCUPANCY_STD))) for _ in range(50)]
    ARRIVAL_ORIGIN = ARRIVALS.copy()  # сохраним оригинальные списки для
    ON_BOARD_ORIGIN= ON_BOARD.copy()  # повторного использования в эксперименте
    # словари статистических данных для аналитики
    seller_queues = {v:[] for v in range(SELLER_LINES)}


# -------------------------
#  Основные процессы модели
# -------------------------
def pick_shortest(lines):
    """
      определяем самую короткую очередь к ресурсам модели -
      функция возвращает кортеж, где 0й элемент - SimPy resource,
      а 1й элемент - номер ресурса (начиная с 1, а не с 0) //
      номер очереди выбирается случайно после перемешивания shuffle, чтобы не всегда начинать с 1ой
    """
    shuffled = list(zip(range(len(lines)), lines)) # list of tuples (i, line)
    rd.shuffle(shuffled)
    shortest = shuffled[0][0]
    for i, line in shuffled:
        if len(line.queue) < len(lines[shortest].queue):
            shortest = i
            break
    return (lines[shortest], shortest + 1)

def bus_arrival(env, seller_lines, scanner_lines):
```

```
    """
      моделируем приезд автобуса через BUS_ARRIVAL_MEAN минут,
      который привозит BUS_OCCUPANCY_MEAN людей;
      это первое событие в модели, после которого срабатывают все другие события
    """
    next_bus_id, next_person_id = 0, 0
    # уникальные ID для автобуса и людей будут нужны для визуализации
    while True:
        next_bus = Globals.ARRIVALS.pop()
        on_board = Globals.ON_BOARD.pop()
        # ждать следующий автобус
        yield env.timeout(next_bus)
        # вызов register_bus_arrival() для записи в логи
        clientIDs = list(range(next_person_id, next_person_id + on_board))
        next_person_id += on_board
        next_bus_id += 1
        #
        while len(clientIDs) > 0:
            group_size = min(round(abs(rd.gauss(PURCHASE_GROUP_SIZE_MEAN, PURCHASE_GROUP_SIZE_STD))), len(clientIDs))
            people_processed = clientIDs[-group_size:] # получить последние элементы из группы
            clientIDs = clientIDs[:-group_size] # оставить id тем кто еще остался
            # кто-то идет в кассу, а кто-то уже с билетом идет на контроль
            if rd.random() > PURCHASE_RATIO_MEAN:
                env.process(scanning_customer(env, people_processed, scanner_lines,
                            TIME_TO_WALK_TO_SELLERS_MEAN + TIME_TO_WALK_TO_SCANNERS_MEAN,
                            TIME_TO_WALK_TO_SELLERS_STD + TIME_TO_WALK_TO_SCANNERS_STD))
            else:
                env.process(purchasing_customer(env, people_processed, seller_lines, scanner_lines))

def purchasing_customer(env, people_processed, seller_lines, scanner_lines):
    # подойти к кассе
    walk_begin = env.now
    sigm3l = TIME_TO_WALK_TO_SELLERS_MEAN - TIME_TO_WALK_TO_SELLERS_STD*3
    if sigm3l < 0: sigm3l =0   # настройка левой границы треугольного распределения
    sigm3h = TIME_TO_WALK_TO_SELLERS_MEAN + TIME_TO_WALK_TO_SELLERS_STD*3
    yield env.timeout(rd.triangular(sigm3l, sigm3h, TIME_TO_WALK_TO_SELLERS_MEAN))
```

```
    walk_end = env.now
    # встать в очередь
    queue_begin = env.now
    # клиент всегда выбирает самую короткую очередь
    seller_line = pick_shortest(seller_lines)
    # ждем начала обслуживания
    with seller_line[0].request() as req:
        # подождать в очереди
        yield req
        queue_end = env.now
        # покупка билета
        sale_begin = env.now
        sigm3l = SELLER_MEAN - SELLER_STD*3
        if sigm3l < 0: sigm3l = 0   # настройка левой границы треугольного распределения
        sigm3h = SELLER_MEAN + SELLER_STD*3
        yield env.timeout(rd.triangular(sigm3l,sigm3h,SELLER_MEAN))
        sale_end = env.now
        # вызов register_group_moving_() для записи в логи
        env.process(scanning_customer(env, people_processed, scanner_lines, TIME_TO_WALK_TO_SCANNERS_MEAN,
TIME_TO_WALK_TO_SCANNERS_STD))

def scanning_customer(env, people_processed, scanner_lines, walk_duration, walk_std):
    # подойти на контроль билетов
    walk_begin = env.now
    yield env.timeout(abs(rd.gauss(walk_duration, walk_std)))
    walk_end = env.now
    # клиент всегда выбирает самую короткую очередь
    queue_begin = env.now
    scanner_line = pick_shortest(scanner_lines)
    with scanner_line[0].request() as req:
        # подождать в очереди
        yield req
        queue_end = env.now
        # контроль билета у каждого клиента
        for person in people_processed:
            scan_begin = env.now
```

```
            yield env.timeout(abs(rd.gauss(SCANNER_MEAN, SCANNER_STD)))
            scan_end = env.now
            # вызов register_visitor_moving_() для записи в логи

# основная функция для запуска модели
def model_env():
  global seller_lines, scanner_lines
  env = simpy.Environment()
  seller_lines = [simpy.Resource(env, capacity = SELLERS_PER_LINE) for _ in range(SELLER_LINES)]
  scanner_lines = [simpy.Resource(env, capacity = SCANNERS_PER_LINE) for _ in range(SCANNER_LINES)]
  env.process(bus_arrival(env, seller_lines, scanner_lines))
  env.run(until = 45)
  print("OK!",env.now)

model_env()
```

Запустим такую модель для проверки.

Если есть сообщение в консоли -

OK! 45

- значит, модель работает, но результатов не показывает и не собирает. Это настроим позже.

Фаза 2

Теперь добавим функции сбора статистики для наблюдения результатов.
Добавим в функцию model_env()

```
event_log =[]
```

перед строкой env = simpy.Environment().

Подготовим специальный мониторинговый процесс сбора статистики по очередям.
Вставим код этой функции, например, перед def bus_arrival().

```
# специальная функция для сбора данных
def monitor(ev):
   global seller_lines
   while True:
      # запомним текущую длину очередей
      for i in range(len(seller_lines)):
         Globals.seller_queues[i].append(len(seller_lines[i].queue))
      yield ev.timeout(1.0)
```

Активируем этот процесс перед запуском модельной системы (курсивом записан уже написанный нами ранее код):

env.process(bus_arrival(env, seller_lines, scanner_lines))

```
env.process(monitor(env))
```

env.run(until = 45)

Добавим нижерасположенный код функций после строк с `class Globals` перед разделом кода `Основные процессы модели.`

```
def avg_wait(raw_waits):
    waits = [ w for i in raw_waits.values() for w in i ]
    ret =round(sum(waits)/len(waits), 1) if len(waits) > 0 else 0
    return ret

def register_bus_arrival(time, bus_id, people_created):
    arrivals[int(time)] += len(people_created)
    print(f"Автобус {bus_id+1} приехал в {round(time, 2)} с {len(people_created)} чел")
    event_log.append({
        "event": "BUS_ARRIVAL", "time": round(time, 2), "busId": bus_id+1, "peopleCreated": people_created
    })

def register_group_moving_from_bus_to_seller(people,walk_begin,walk_end,seller_line,queue_begin,queue_end,sale_begin,sale_end):
    waitq = queue_end - queue_begin
    service_time = sale_end - sale_begin
    seller_waits[int(queue_end)].append(waitq)
    print(f"Группа {len(people)} чел ждала {round(waitq,2)} мин в очереди_{seller_line}, обслужилась за {round(service_time,2)} мин")
    event_log.append({
        "event": "WAIT_IN_SELLER_LINE", "people": people, "sellerLine": seller_line,
        "time": round(queue_begin, 2), "duration": round(waitq, 2)
    })

def register_visitor_moving_to_scanner(person, walk_begin, walk_end, scanner_line, queue_begin, queue_end, scan_begin, scan_end):
    waitq = queue_end - queue_begin
    service_time = scan_end - scan_begin
    scan_waits[int(queue_end)].append(waitq)
    print(f"Клиент на контроле ждал {round(waitq,2)} мин в очереди_{scanner_line}, обслуживался {round(service_time,2)} мин")
    event_log.append({
        "event": "WAIT_IN_SCANNER_LINE", "person": person, "scannerLine": scanner_line,
        "time": round(queue_begin, 2), "duration": round(waitq, 2)
    })
```

Используем эти функции для регистрации событий в списки логов (журналирование).
В функции bus_arrival() дополним выделенную здесь строку (курсивом записан уже написанный нами ранее код):

yield env.timeout(next_bus)

 # автобус прибыл, определяем Id для прибывших клиентов для записи в логи

clientIDs = list(range(next_person_id, next_person_id + on_board))

```
register_bus_arrival(env.now, next_bus_id, people_ids)
```

next_person_id += on_board

В функции purchasing_customer() дополним выделенную строку здесь (курсивом записан уже написанный нами ранее код):

yield env.timeout(rd.gauss(SELLER_MEAN, SELLER_STD))

sale_end = env.now

```
register_group_moving_from_bus_to_seller(people_processed, walk_begin, walk_end, seller_line[1],queue_begin,queue_end,sale_begin,sale_end)
```

env.process(scanning_customer(env, people_processed, scanner_lines, TIME_TO_WALK_TO_SCANNERS_MEAN, TIME_TO_WALK_TO_SCANNERS_STD))

В функции scanning_customer() дополним выделенную строку здесь (курсивом дан написанный ранее код):

scan_begin = env.now

yield env.timeout(abs(rd.gauss(SCANNER_MEAN, SCANNER_STD)))

 # контроль билетов пройден

scan_end = env.now

```
register_visitor_moving_to_scanner(person, walk_begin, walk_end, scanner_line[1], queue_begin, queue_end, scan_begin, scan_end)
```

Запустим такую модель для проверки работы.

После запуска модели должен наблюдаться вывод на консоль журнала событий (рисунок 3).

```
Группа клиентов 1 чел ждала 1.99 мин в очереди_6, обслужилась за 1.1 мин
Клиент на контроле ждал 0.76 мин в очереди_3, обслуживался 0.06 мин
Клиент на контроле ждал 1.15 мин в очереди_4, обслуживался 0.08 мин
Клиент на контроле ждал 0.76 мин в очереди_3, обслуживался 0.05 мин
Клиент на контроле ждал 0.95 мин в очереди_2, обслуживался 0.08 мин
Группа клиентов 2 чел ждала 1.66 мин в очереди_2, обслужилась за 1.26 мин
Клиент на контроле ждал 0.92 мин в очереди_1, обслуживался 0.2 мин
Автобус 11 приехал в 33.57 с 74 чел
Клиент на контроле ждал 0.95 мин в очереди_2, обслуживался 0.09 мин
Группа клиентов 2 чел ждала 2.09 мин в очереди_3, обслужилась за 0.9 мин
Клиент на контроле ждал 1.15 мин в очереди_4, обслуживался 0.12 мин
Клиент на контроле ждал 0.92 мин в очереди_1, обслуживался 0.06 мин
Группа клиентов 1 чел ждала 2.19 мин в очереди_5, обслужилась за 0.8 мин
Клиент на контроле ждал 1.24 мин в очереди_4, обслуживался 0.05 мин
Клиент на контроле ждал 0.77 мин в очереди_3, обслуживался 0.16 мин
Клиент на контроле ждал 0.85 мин в очереди_2, обслуживался 0.09 мин
Клиент на контроле ждал 0.77 мин в очереди_3, обслуживался 0.07 мин
Клиент на контроле ждал 1.24 мин в очереди_4, обслуживался 0.1 мин
Клиент на контроле ждал 1.11 мин в очереди_1, обслуживался 0.15 мин
Клиент на контроле ждал 0.85 мин в очереди_2, обслуживался 0.1 мин
Клиент на контроле ждал 1.17 мин в очереди_4, обслуживался 0.07 мин
Группа клиентов 2 чел ждала 1.92 мин в очереди_4, обслужилась за 1.26 мин
Клиент на контроле ждал 0.85 мин в очереди_2, обслуживался 0.05 мин
Клиент на контроле ждал 1.11 мин в очереди_1, обслуживался 0.08 мин
Клиент на контроле ждал 0.77 мин в очереди_3, обслуживался 0.17 мин
Клиент на контроле ждал 1.1 мин в очереди_3, обслуживался 0.04 мин
Клиент на контроле ждал 1.11 мин в очереди_1, обслуживался 0.09 мин
Клиент на контроле ждал 1.17 мин в очереди_4, обслуживался 0.13 мин
```

Рисунок 3. Журнал событий в консоли

Фаза 3

Теперь для проведения расчетных экспериментов добавим возможность обмена параметрами между моделью и программой управления экспериментом.

В области объявления глобальных переменных добавим:

```
ОКНО = True   # флаг визуализации журнала в окне консоли
```

(замечание: в идентификаторе ОКНО все символы латинские :)

Подготовим передачу параметров через аргументы вызова функции модели.

```
def model_env(config=None):
```

Далее добавим ниже строки объявлений глобальных переменных:

```
    global seller_lines, scanner_lines, event_log
    global SELLER_LINES, SCANNER_LINES, ОКНО
```

В самой функции после строки env = simpy.Environment() добавим такой код:

```
ОКНО = False  # надо выводить сообщения в консоль?
finitime = 45
   if config:
      SELLER_LINES = config["num_cashiers"]
      SCANNER_LINES = config["sel_scanners"]
      rd.seed(config["seed_select"])
      finitime = config["simulation_time"]
```

Изменим запуск модели так:

```
 env.run(until = finitime)
```

Добавим возвращаемый результат из функции модели:

```
   resq = {f'Q{v}':Globals.seller_queues[v] for v in range(SELLER_LINES)}
   return event_log, resq
```

Созданные списки ARRIVALS и ON_BOARD после их создания в коде сохраним для воспроизведения при экспериментировании. Для этого в функции `model_env` добавим код перед строкой

```
seller_lines =[simpy.Resource(env, capacity =SELLERS_PER_LINE) for _ in
range(SELLER_LINES)]
```

строки

```
   Globals.seller_queues = {v:[] for v in range(SELLER_LINES)}
   Globals.ARRIVALS=Globals.ARRIVAL_ORIGIN.copy()
   Globals.ON_BOARD=Globals.ON_BOARD_ORIGIN.copy()
```

В коде **всех функций** регистрации событий скорректируем выполнение **<u>всех</u>** операторов *print*() с учетом флага ОКНО в виде:

if ОКНО: print(f"Автобус {bus_id+1} приехал в {round(time, 2)} с {len(people_created)} чел")

Запустите для проверки модель как с флагом ОКНО = True, так и ОКНО = False ---

print(model_env())

Сохраним файл с моделью под названием **exper_tickets.py**

Модель в основном готова для разовых экспериментов.
Для проверки запустите модель с разными начальными значениями ГПСЧ random.seed(), попробуйте 3-4 варианта.
Убедитесь в различии получаемых результатов, сохраните их в отчет о выполненной работе.

Фаза 4

Спроектируем графический интерфейс для наблюдения результатов модели на основе фреймворка Streamlit.
Streamlit – фреймворк Python с открытым исходным кодом, который популярен для проектов в области машинного обучения и анализа данных. Он позволяет публиковать веб-приложения в общем доступе, и использует встроенный веб-сервер с возможностью развертывания как локально, так и в контейнере docker.
Установка Streamlit делается, например, с помощью команды:

pip install streamlit

После установки и запуска такого приложения нужно в браузере перейти по локальному URL localhost:8501, чтобы увидеть приложение Streamlit в действии.

Streamlit позволяет отображать в клиентском браузере данные несколькими командами:

st.title() - для установки заголовка;
st.text() - для записи описания для графика;
st.markdown() - для отображения текста в виде markdown;
st.latex() - для отображения математических выражений на панели мониторинга;
st.write() - помогает отображать все возможные детали, например, график, фрейм данных, функции, модель и т.д.;
st.sidebar() - для отображения данных на боковой панели;
st.dataframe() - для отображения фрейма данных Pandas;
st.map() - для отображения карты и т.п.

С веб-приложением обычно работают так:

- открываем для изменений файл Python в редакторе кода (Visual Studio Code, Notepad++, SublimeText и пр.)
- наблюдаем за изменениями в браузере, расположенном в соседнем окне, обновляя страничку браузера с веб-модулем приложения.

Создадим файл **app.py** и вставим в него такой код:

```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt

# Импорт нашей модели simpy
import exper_tickets as mdl

# Функция для запуска симуляции и передачи данных в реальном времени
def run_simulation (params):
    """Запускает моделирование с использованием параметров """
    results = []
    # Запуск модели
    results = mdl.model_env(config=params)
    # возврат результата
    return results
```

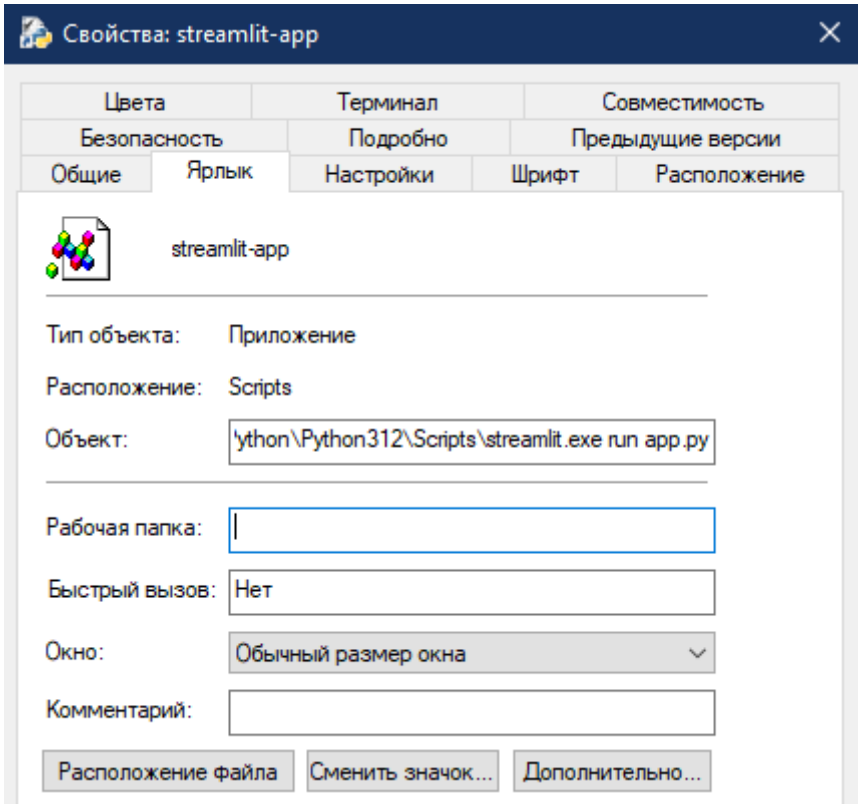
```
   st.set_page_config(
      page_title="модель_тест",
      page_icon=":shark:",
      layout="centered",
      initial_sidebar_state="expanded"
   )

# Интерфейс Streamlit
st.title(":hearts: Моделирование сервиса")

with st.sidebar:
  with st.form(key="input_form"):
    # Настройки параметров модели
    st.header("Параметры модели :clubs:")
    num_cashiers = st.slider("Количество кассиров", 3, 10, 1)
    cashier_processing_time = st.slider(
      "Среднее время обслуживания в кассе (сек)", 30, 100, 10)
    sel_scanners = st.selectbox("Количество контролёров", (4,5,6,7,8,9))
    queue_timeout = st.slider("Среднее время обслуживания на контроле (сек)",15,50,5)
    p_range = ('Magadan', 'Vorkuta', 'UlanUdae')
    seed_select = st.radio('seed', p_range)
    simulation_time = st.slider("Время моделирования (мин)", 10, 90, 5)
    st.session_state.pr_click = st.form_submit_button(label=":spades: Запуск модели")

# Подготовка параметров запуска модели
params = {
    "num_cashiers": num_cashiers,
    "cashier_processing_time": cashier_processing_time,
    "sel_scanners": sel_scanners,
    "queue_timeout": queue_timeout,
    "seed_select": seed_select,
    "simulation_time": simulation_time,
}

# Кнопка для запуска
if st.session_state.pr_click:
    # Контейнеры для графиков и метрик
    progress_placeholder = st.empty()

    # Запуск модели!
    with st.spinner("♺ Запуск модели..."):
      events, sell_queues = run_simulation(params)
      df1 = pd.DataFrame(events)
      df2 = pd.DataFrame(sell_queues)

      graph1_placeholder = st.empty()
      graph2_placeholder = st.empty()
      st.success("Моделирование завершено!", icon=" ")
```

Для запуска приложения Streamlit в ОС Windows используем команду в окне терминала:

C:\Users\USER\AppData\Local\Programs\Python\Python312\Scripts\streamlit.exe run app.py

- здесь вместо USER будет имя вашего пользователя Windows, и вместо Python312 будет название вашей версии Python.

Для удобства запуска приложения модели в Windows нужно сделать файл-ярлык streamlit-app.lnk



Теперь запустите модель с помощью этого ярлыка для проверки работы приложения.

Фаза 5

Теперь подготовим модель ***exper_tickets.py*** для использования в статистическом расчетном эксперименте, настроив сбор данных для статистического расчёта с помощью фреймворка `scipy`.
Добавим такой новый код после всего ранее созданного кода модели:

```
from scipy import stats
import math

def stat_experimentA():
    global SELLER_LINES, SCANNER_LINES, OKHO, seller_waits, scan_waits
    # выключаем вывод в консоль
    Tmp = OKHO; OKHO = False
    # эксперимент_1_
```

```
    Level = 4  # кол-во уровней фактора
    numReplica = 11  # кол-во реплик на каждом уровне
    print("_Запуск эксперимента_")
    #_1_
    experiment_ds ={o:[] for o in range(Level)}
    # создадим словарь списков результатов моделирования
    for f in range(Level):
        SELLER_LINES = 4+f
        SCANNER_LINES = 6
        print(f'_уровень ++ {f+1} ++ SELLER_LINES= {SELLER_LINES}')
        for i in range(numReplica):
            # сброс списков статистики
            arrivals = defaultdict(lambda: 0)
            seller_waits = defaultdict(lambda: [])
            scan_waits = defaultdict(lambda: [])
            Globals.seller_queues = {v:[] for v in range(SELLER_LINES)}
            # установка нового случ.зерна
            rd.seed(732 + i*numReplica + f)
            model_env()  # запуск модели
            if OKHO: print(f'+ {f+1} + реплика_{i+1} +')
            # сохраняем результат прогона
            experiment_ds[f].append(avg_wait(seller_waits)+avg_wait(scan_waits))
    OKHO = Tmp
if OKHO: print (experiment_ds)
```

Для проверки запустите код ***exper_tickets.py*** через новый эксперимент, но предварительно закомментируйте вызов простого прогона модели:

```
# print(model_env())
OKHO=True
# запуск стат.эксперимента
stat_experimentA()
```

Сейчас должен отработать цикл сбора статистики без визуализации работы, но с показом по завершении работы состояния словаря **experiment_ds**.

На втором шаге настройки эксперимента добавим блок кода обработки собранной статистики. Для этого будем использовать статистическую библиотеку ***scipy.stats***.

```
    #_2_
    print('\n_Сводная описательная статистика_\n фактор SELLER_LINES принимал
значения [4,5,6,7]\n')

    statDescript ={
    'N':[0]*Level,'smm':[0]*Level,'sm':[0]*Level,'sv':[0]*Level,
    'ss':[0]*Level,'sk':[0]*Level,'D_KS':[0]*Level,'pval_KS':[0]*Level,
    'F':[0]*Level,'pval_F':[0]*Level }
    for z in range(Level):
        nn, (smin, smax), smn, sv, ss, sk = stats.describe(experiment_ds[z])
        statDescript['N'][z]=nn     # размер выборки
        statDescript['smm'][z]=(round(smin,3), round(smax,3))
        statDescript['sm'][z]=round(smn,3)   # среднее выборочное
```

```
        statDescript['sv'][z]=round(sv,3)    # дисперсия выборки
        statDescript['ss'][z]=round(ss,3)    # skew/скос
        statDescript['sk'][z]=round(sk,3)    # kurtosis/эксцесс
        if OKHO: print(f'Датасет{z+1}::{nn}: среднее= {round(smn,3)} | дисперсия
= {round(sv,3)} | min={round(smin,3)} | max={round(smax,3)}')
        # нормализуем данные датасета перед тестом на нормальность
        zx = (experiment_ds[z] - smn) / math.sqrt(sv)
        # применим тест Колмогорова-Смирнова на нормальность распределения
        dks, pval = stats.kstest(zx, 'norm')
        print (f'Датасет{z+1}: KS-статистика: D={round(dks,4)} | p-value =
{round(pval,4)}\n')
        statDescript['D_KS'][z]=round(dks,4)
        statDescript['pval_KS'][z]=round(pval,4)
print(statDescript)
```

Запустите ***exper_tickets.py*** через вызов эксперимента stat_experimentA().
Сейчас должен отработать цикл сбора статистики без визуализации работы, но с показом по завершении работы состояния словаря *experiment_ds* и набора данных сводной описательной статистики по собранным наблюдениям.

На третьем шаге настройки эксперимента добавим код расчёта по методу однофакторного дисперсионного анализа ANOVA[i], который может показать степень влияния изменчивости фактора на результат. Фактором является количество касс, результатом – время ожидания в очередях.

```
    #_3_
    print("\n_Однофакторный дисперсионный анализ ANOVA (alpha= 0.05)_\n")
    F, pval = stats.f_oneway(experiment_ds[0], experiment_ds[1],
experiment_ds[2], experiment_ds[3])
    print (f'значение критерия F: {round(F,4)} | p-value: {round(pval,4)}')
    statDescript['F'][0]=round(F,3)
    statDescript['pval_F'][0]=round(pval,4)

    return statDescript
```

Модель с экспериментальным исследованием подготовлена.
Запустим ***exper_tickets.py*** с этим экспериментом:

```
  print(stat_experimentA())
```

Задание для самостоятельной работы - уточните по рассчитанной дисперсии количество прогонов (реплик) для обеспечения качества оценки на уровне 95% с точностью 0,5.
Внесите изменения в количество прогонов и повторите обработку данных по уточнённому расчётному эксперименту.

Фаза 6

Вернёмся к приложению моделирования и добавим построение графиков с использованием библиотеки *matplotlib* в файле приложения **app.py**. В этом случае традиционно используют фреймворк *pandas,* добавляющий удобные возможности обработки данных.
Найдите этот фрагмент кода и измените на такой код с исправлением блока запуска модели:

```
    # Запуск модели!
    with st.spinner("♺ Запуск модели..." ):
      events, sell_queues = run_simulation(params)
      df1 = pd.DataFrame(events)
      df2 = pd.DataFrame(sell_queues)

      graph1_placeholder = st.empty()
      graph2_placeholder = st.empty()
      st.success("Моделирование завершено!", icon=" ")

    fig, ax = plt.subplots(figsize=(8,6), layout='constrained', facecolor='0.7')
    for sc in df2.columns:
       ax.step(df2.index, df2[sc], label='Kacca_'+sc)
    ax.legend(loc='upper left')
    # Обновление графика
    graph1_placeholder.scatter_chart(df1[["person"]])
    graph2_placeholder.pyplot(fig)

    # Итоговые метрики
    st.write("Итоговые метрики:")
    st.write(df1)
    st.dataframe(df2)


if statest :
   with st.spinner('Выполняется...'):
      results = mdl.stat_experimentA()
      t_df = pd.DataFrame(results)
      st.write(t_df)
      ff = t_df[['F'][0]]
      fp = t_df[['pval_F'][0]]

   st.metric("критерий F-тест", ff[0], fp[0])

   if fp[0]>0.05:
      st.markdown("**Гипотеза H0 не отклоняется!**")
   else:
      st.markdown("**Гипотеза H0 отклоняется!**")
   st.success(" Статистический анализ завершен!", icon=" ")
```

Задание для самостоятельной работы - настройте и проведите **свой** эксперимент stat_experiment_B по оценке влияния другого параметра модели SCANNER_LINES на тот же результат процесса. Сравните с результатами первого эксперимента, сделайте письменно выводы в файле отчета.

Фаза 7

Для построения приложения с возможностью расширения вариантов экспериментирования изменим код приложения streamlit в форме многостраничного приложения.

Сделайте в своей рабочей папке, где находится код программы модели Python, вложенную папку ***pages*** для кода страниц приложения, и служебную папку ***.streamlit*** (с точкой в начале!).

Создайте в своей рабочей папке файл стартовой страницы **start.py**.
Впишите в него такой код:

```
# -*- coding: utf-8 -*-
import streamlit as st
import graphviz

# Настройка интерфейса Streamlit -первый оператор!
st.set_page_config(
   page_title="модель_A",
   page_icon=" ",
   layout="wide",
   initial_sidebar_state="expanded"
)

st.title(":hearts: Моделирование прохода на мероприятие")
st.write("Моделирование процессов организации пропуска потока посетителей")
st.sidebar.page_link("pages/Main_model.py", label="Базовая модель")
st.sidebar.page_link("pages/Experiment_1.py", label="Эксперимент A")
st.divider();
st.write("Блочная схема процесса")
dot_string = """
 digraph MG {
 rankdir="LR"
 A [label="Приезд", shape=rectangle, style=filled, fillcolor=lightblue];
 B [label="Касса", shape=rectangle, style=filled, fillcolor=lightblue];
 C [label="Контроль", shape=rectangle, style=filled, fillcolor=lightblue];
 D [label="выход", shape=circle, style=filled, fillcolor=white];
 K [label="группа", shape=ellipse, style=filled, fillcolor=cyan];
 P [label="персона", shape=ellipse, style=filled, fillcolor=cyan];
 H [label="персона", shape=ellipse, style=filled, fillcolor=cyan];

 A -> K [color=blue, style=bold];
 K -> B [color=blue, style=bold];
 B -> P [color=blue, style=bold];
 P -> C [color=blue, style=bold];
 C -> D;
 A -> H [color=green, style=bold];
 H -> C [color=green, style=bold];
 }
"""
st.graphviz_chart(dot_string)
```

Здесь в код добавлен текст описания логической схемы процессов в формате DOT, используемый в библиотеке graphviz. Если этот граф не отображается, установите библиотеку

pip install graphviz

В папке **pages** создайте файлы `Main_model.py` и `Experiment_1.py`, вызываемые из стартовой страницы. В них для отладки временно запишите пустой оператор **pass.**

Для правильной настройки работы веб-приложения *streamlit* нужно изменить его конфигурацию через файл **config.toml** , который должен находиться в папке **.**streamlit

Содержание файла `config.toml`:

```
[server]
folderWatchBlacklist = []
fileWatcherType = "auto"
headless = false
runOnSave = false
enableCORS = false
enableXsrfProtection = false
port = 8501
[client]
showErrorDetails = "none"
showSidebarNavigation = false
toolbarMode = "viewer"
[runner]
magicEnabled = true
fastReruns = true
enforceSerializableSessionState = false
enumCoercion = "nameOnly"
[browser]
serverAddress = "localhost"
serverPort = 8501
gatherUsageStats = false
[theme]
# This can be one of the following: "light" or "dark"
base = "light"
primaryColor = "#263543"
backgroundColor = "honeydew"
```

Для проверки работы модели запускайте Streamlit-приложение:

streamlit.exe run start.py

После проверки стартовой страницы внесём код вызова основной модели в файл `pages/Main_model.py` Этот код в основном был протестирован в первом варианте эксперимента `app.py`.

```
# -*- coding: utf-8 -*-
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt

# Импорт модели
import exper_tickets as mdl
```

```
df1, df2 = 0, 0

# Функция для запуска имитации и получения данных в реальном времени
def run_simulation (params):
    """Запускает симуляцию с использованием параметров"""
    results = []
    # Запуск моделирования
    results = mdl.model_env(config=params)
    return results

# Настройка интерфейса Streamlit -первый оператор!
st.set_page_config(
   page_title="модель_1",
   page_icon=":shark:",
   layout="centered",
   initial_sidebar_state="expanded"
)

st.title(":hearts: Моделирование прохода на мероприятие")
ts1 = st.container()

with st.sidebar:
  st.page_link("Start.py", label="[ Стартовая страница ]")
  with st.form(key="input_form"):
    st.header("Параметры прогона модели")
    # Параметры модели
    num_cashiers = st.slider("Количество кассиров", 3, 10, 1)
    cashier_processing_time = st.slider(
       "Среднее время обслуживания в кассе (сек)", 30, 100, 10)
    sel_scanners = st.selectbox("Количество контролёров", (3,4,5,6,7,8))
    queue_timeout = st.slider("Среднее время обслуживания у контролёра (сек)",
15, 50, 5)
    seed_select = st.radio('seed', ['Magadan', 'Vorkuta', 'Ulanudae'])
    simulation_time = st.slider("Время моделирования (мин)", 10, 100, 5)
    st.session_state.pr_click = st.form_submit_button(label=":spades: Запуск
модели")

# Подготовка параметров запуска модели
params = {
    "num_cashiers": num_cashiers,
    "cashier_processing_time": cashier_processing_time,
    "sel_scanners": sel_scanners,
    "queue_timeout": queue_timeout,
    "seed_select": seed_select,
    "simulation_time": simulation_time,
}

# Контейнеры для графиков и метрик
tc1 = st.container()
tc1.write (":spades: Модель")
tc1.graph1_placeholder = st.empty()
tc1.graph2_placeholder = st.empty()
tc2 = st.container()
tc2.data1_placeholder = st.empty()
```

```
tc2.data2_placeholder = st.empty()

# Кнопка для запуска модели
if st.session_state.pr_click:
    with st.spinner("♺ Запуск модели..." ):
       events, sell_queues = run_simulation(params)
       ts1.success("Моделирование завершено!", icon=" ")
    df1 = pd.DataFrame(events)
    df2 = pd.DataFrame(sell_queues)

if isinstance(df2, pd.DataFrame):
   fig, ax = plt.subplots(figsize=(7, 4),layout='constrained',facecolor='0.7')
   for sc in df2.columns:
       ax.step(df2.index, df2[sc], label='Касса_'+sc)
   ax.legend(loc='upper left')
   # Обновление графика
   tc1.graph2_placeholder = st.pyplot(fig)

# Обновление графика
if isinstance(df1, pd.DataFrame):
   tc1.graph1_placeholder = st.scatter_chart(df1[["person"]])

# Итоговые метрики
if isinstance(df1, pd.DataFrame):
   st.write("Итоговый журнал моделирования")
   tc2.data1_placeholder = st.dataframe(df1)

if isinstance(df2, pd.DataFrame):
   st.write("Журнал очередей в кассы")
   tc2.data2_placeholder = st.dataframe(df2)

st.divider()
st.text("_")
```

Теперь внесём код запуска эксперимента в файл `pages/Experiment_1.py` Этот код был протестирован в первом варианте эксперимента `app.py`.

```
# -*- coding: utf-8 -*-
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt

# Импорт модели
import exper_tickets as mdl

res =[]

# Настройка интерфейса Streamlit -первый оператор!
st.set_page_config(
   page_title="эксперимент_А",
   page_icon=":shark:",
   layout="wide",
```

```
   initial_sidebar_state="expanded"
)
st.title(":hearts: Моделирование прохода на мероприятие")
levels=4; sellers=3; scanners=2; replics=10

with st.sidebar:
    st.page_link("Start.py", label="[ Стартовая страница ]")
    st.markdown('*Эксперимент и статистический анализ*')

    with st.form(key="input_form"):
      st.header("Параметры эксперимента")
      # Параметры модели
      sellers = st.slider("Нач.количество кассиров", 1, 9, 1)
      levels = st.selectbox("Количество уровней фактора", (3,4,5,6,7,8))
      st.divider()
      scanners = st.selectbox("Количество контролёров", (2,3,4,5,6,7,8,9))
      replics = int(st.number_input("Количество реплик", min_value=2, value=9))
      st.session_state.ex_click = st.form_submit_button(label=":spades:
Выполнить эксперимент")

# Подготовка параметров запуска эксперимента
params = {
    "Levels": levels,
    "Sellers": sellers,
    "Scanners": scanners,
    "Replics": replics
}

st.write (":computer: Отсеивающий эксперимент по параметру количества касс")
# Запуск моделирования
if st.session_state.ex_click :
   with st.spinner('Выполняется...'):
      res = mdl.stat_experimentA (config=params)
      st.markdown("__Однофакторный дисперсионный анализ ANOVA (alpha=0.05)__")
      t_df = pd.DataFrame(res)
      st.write(t_df)
      ff = t_df[['F'][0]]
      fp = t_df[['pval_F'][0]]

   st.metric("критерий F-тест", ff[0], fp[0])
   if fp[0]>0.05:
      st.markdown("**Гипотеза H0 не отклоняется!**  \
      _Эксперимент не подтверждает влияние фактора_ \
      ")
   else:
      st.markdown("**Гипотеза H0 отклоняется!**  \
      _Эксперимент подтверждает влияние фактора_ \
      ")

   st.success(" Статистический анализ завершен!", icon=" ")
```

Код в файле Experiment_1 предполагает использование созданного расчётного эксперимента с передачей настроечных параметров для проведения расчёта.

Для этого исправьте в файле **exper_tickets.py** оформление кода выполнения `stat_experimentA` в таком виде (найдите этот фрагмент и замените эту часть кода):

```
def stat_experimentA(config=None):
    global SELLER_LINES, SCANNER_LINES, OKHO, seller_waits, scan_waits
    OKHO = False
    #
    Level = 4  # кол-во уровней фактора
    numReplica = 10  # кол-во реплик на каждом уровне
    Sellers = 4 # нач.кол-во касс
    Scanners = 6 # фикс.кол-во контролеров
    #
    if config:
      Level = config["Levels"]
      Sellers = config["Sellers"]
      Scanners = config["Scanners"]
      numReplica = config["Replics"]
    #
    # эксперимент 1
    print("_Запуск эксперимента_")
    #_1_
    experiment_ds ={o:[] for o in range(Level)} # словарь списков результатов
    for f in range(Level):
        SELLER_LINES = Sellers + f
        SCANNER_LINES = Scanners
        print(f'_уровень ++ {f+1} ++ SELLER_LINES= {SELLER_LINES}')
        for i in range(numReplica):
        # восстановление нач.значений
            # сброс списков статистики
            arrivals = defaultdict(lambda: 0)
            seller_waits = defaultdict(lambda: [])
            scan_waits = defaultdict(lambda: [])
            #
            rd.seed(731 + i*numReplica + f) # установка нового зерна random
            model_env()  # запуск модели
            if OKHO: print(f'+ {f+1} + реплика_{i+1} +')
            experiment_ds[f].append(avg_wait(seller_waits)+avg_wait(scan_waits))
    if OKHO: print (experiment_ds)
```

Для проверки работы модели запускайте Streamlit-приложение:

streamlit.exe run start.py

Для удобства запуска приложения модели нужно сделать файл-ярлык model-app.lnk
Проведите несколько экспериментов с разными значениями параметров.

Задание для самостоятельной работы - добавьте новую страницу для своего эксперимента, сделанного ранее в **фазе 6**.

Пришлите полученный код всех файлов модели (py, toml, lnk) в виде архива (zip) и отчёт о проведённых экспериментах (в файле docx) на почту для проверки преподавателем.

Примечание 1

Односторонний дисперсионный анализ ANOVA использует такие нулевые и альтернативные гипотезы:

H0 (нулевая гипотеза): μ1 = μ2 = μ3 = ... = *μk* (все средние значения совокупностей равны).

H1 (альтернативная гипотеза): по крайней мере, одно среднее значение выборок отличается от остальных.

Если рассчитанное значение *p-value* менее 0,05, то можно отвергнуть нулевую гипотезу. Это означает, что у нас достаточно оснований, чтобы утверждать, что *существует* статистически значимая разница между выборками.

Если отвергается нулевая гипотеза, это значит, что, по крайней мере, одно из средних значений совокупности отличается от других, но в ANOVA не указано, какие средние значения совокупности отличаются. Если же нужно определить такую выборку, то необходимо выполнить специальные (post hoc) тесты, также известные как тесты множественных сравнений, например, тест Тьюки, метод Хольма, тест Даннета.

Поскольку p-value действительно кажется значимым, и поскольку у нас есть несколько различных факторов, то обычно запускают post-hoc тест, чтобы проверить, что разница между средними все еще значима даже после проверки ошибок первого типа.
Можно провести post-hoc тесты с помощью модуля multicomp библиотеки statsmodels, используя Tukey Honestly Significant Difference (Tukey HSD) тест.

Сейчас по плану эксперимента нам этого делать не нужно.

---

[i] 1