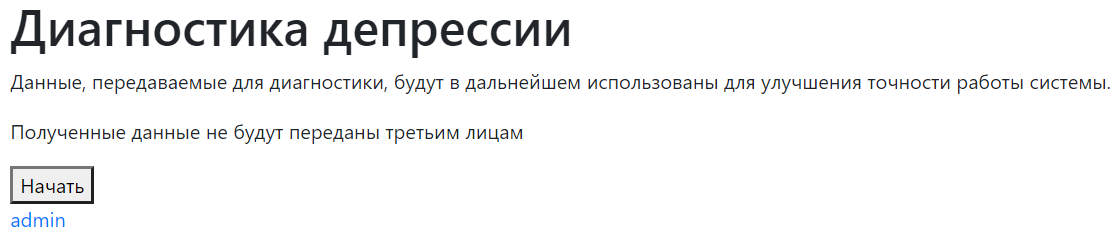
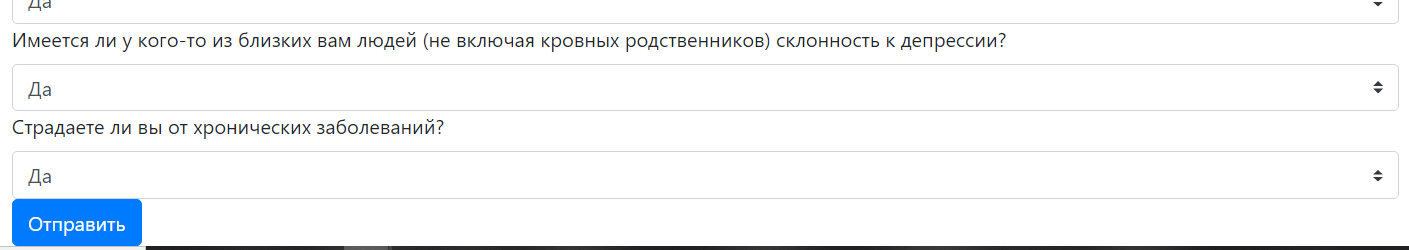
# Платформа для прохождения тестов

## Описание процесса прохождения диагностики



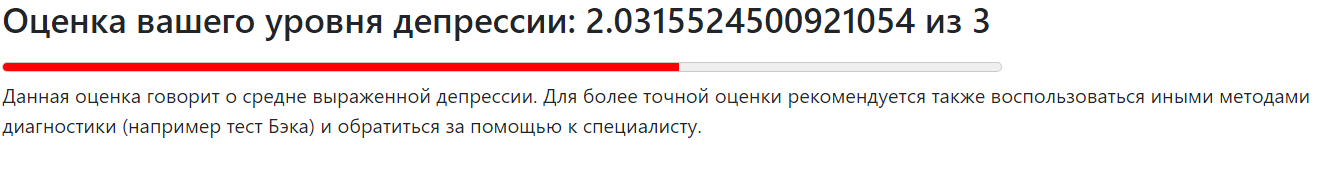
***Главная страница***

При нажатии кнопки “Начать” откроется анкета с вопросами.



***Страница тестирования***

После ввода ответов на все вопросы следует нажать “Отправить”. После нажатия откроется страница с результатами диагностики.



## Руководство администратора

### Запуск системы

Для локального запуска системы запустите файл !run.bat через Проводник или через консоль:

.\!run.bat

Для запуска системы через Docker: 1) Выполните сборку image системы, если она ещё не была совершена:

docker build --pull --rm -f "Dockerfile" -t finals:latest "."

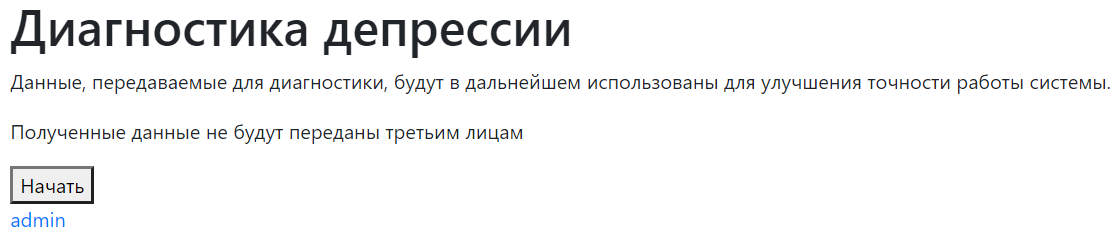
1. Запустите файл !rundocker.bat через Проводник или через консоль

.\!rundocker.bat

Для запуска тестов системы запустите файл !rundoctest.bat через Проводник или через консоль:

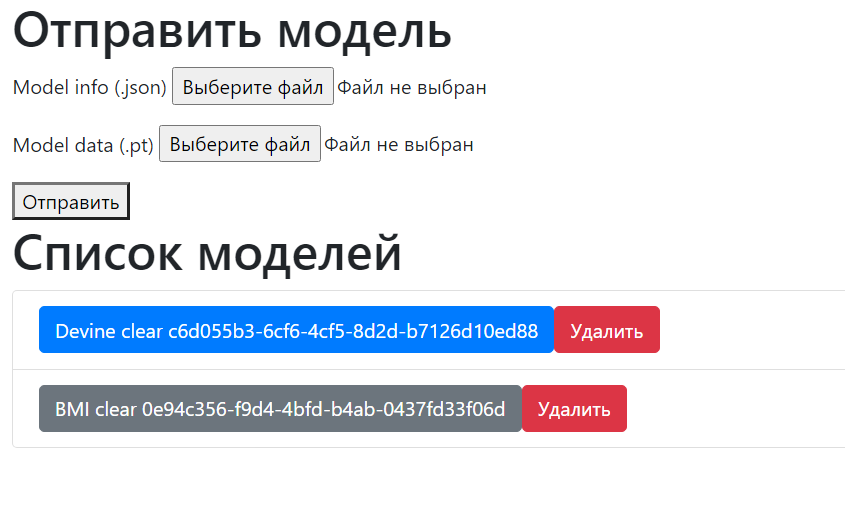
.\!rundoctest.bat

### Управление моделями



***Главная страница***

Для открытия меню администрирования нажмите ссылку Admin. После ввода логина и пароля откроется страница администрирования.



***Страница администрирования***

Сверху страницы расположен ввод новой модели в систему. Для ввода модели выполните выбор файла конфигурации модели и файла нейронной сети и нажмите Отправить.

Снизу страницы расположен список моделей. Для каждой модели указывается её название и уникальный идентификатор.

Кнопка основной модели страницы выделена синим цветом. Для изменения основной модели нажмите на блок модели.

Напротив кнопок моделей находятся кнопки для удаления моделей.

## Описание кода

### Модуль nn\_handler

#### nn\_handler.field

Модуль представления полей.

##### Класс FieldType

Перечисление типов полей. Используется для упрощения соответствия строк типов полей и классов полей.

Значения: - choice = 1 - numeric = 2 - calc = 3

Имеет преобразование к числу, используется для передачи в шаблон фронтенда

[Тесты](#test-fieldtype)

##### Класс Field

Абстрактный класс поля. Ожидается, что он не будет использоваться сам по себе, нобудет наследоваться.

Поля: - name — имя поля, str - field\_type — тип поля класса [FieldType](#класс-fieldtype) - ignore — игнорирование поля как входа в нейронную сеть

[Тесты](#test-field)

##### Класс QField

Класс поля с вопросом. Наследует Field. Используется в случае, если значение поля зависит от ввода пользователя.

Поля: - name — имя поля, str - field\_type — тип поля класса [FieldType](#класс-fieldtype) - ignore — игнорирование поля как входа в нейронную сеть - question — отображаемый вопрос, str

[Тесты](#test-qfield)

##### Класс QFieldChoice

Класс поля с множественным выбором ответа. Наследует [QField](#класс-qfield).

Поля: - name — имя поля, str - field\_type — тип поля класса FieldType- - ignore — игнорирование поля как входа в нейронну- сеть - question — отображаемый вопрос, str- - separate — отображать ли через “one hot encoding”- bool - answers — варианты ответа: list[str] если separate == True иначе dict[str, number]

[Тесты](#test-qfieldchoice)

##### Класс QFieldNumeric

Класс поля с числовым ответом. Наследует [QField](#класс-qfield).

Поля: - name — имя поля, str - ignore — игнорирование поля как входа в нейронную сеть - question — отображаемый вопрос, str - mean — среднее значение для Z-scoring - std — стандартное квадратичное отклонение для Z-scoring

[Тесты](#test-qfieldnumeric)

##### Класс FieldCalc

Класс вычислимого поля. Наследует [Field](#класс-field).

Поля: - name — имя поля, str - ignore — игнорирование поля как входа в нейронную сеть - code — исполняемый код для расчёта значения поля - mean — среднее значение для Z-scoring - std — стандартное квадратичное отклонение для Z-scoring, неотрицательное

Требования к исполняемому коду: 1. Должно иметь строковое представление. 2. Должно обращаться к объекту df. Он имеет тип dict. Ключами являются имена полей. 3. Должно быть упоминание df[<name>]. Желательно с присваиванием. 4. Нельзя обращаться к полям, объявления которых лежат ниже в описании конфигурации системы.

Пример - вычисление коэффициента массы тела по росту и весу с последующим игнорированием веса и роста:

>>> from nn\_handler.field import QFieldNumeric, FieldCalc  
>>> f1 = QFieldNumeric('вес', True, 'Ваш вес', 0.0, 1.0)  
>>> f2 = QFieldNumeric('рост', True, 'Ваш рост',0.0, 1.0)  
>>> f3 = FieldCalc('BMI',False,"df['BMI']=df['вес']/df['рост']\*\*2", 0.0 , 1.0)

[Тесты](#test-fieldcalc)

##### Класс FieldInputError

Базовый класс ошибки ввода в систему экземпляра [Field](#класс-field).

##### Класс FieldNameExists

Класс ошибки: поле с именем уже существует. Проверка на имя происходит при инициализации объекта на уровне [Field](#класс-field).

[Тесты](#test-fieldnameexists)

##### Класс CalcCodeParseError

Класс ошибки: код вычисляемого поля некорректен. Варианты содержания сообщения: 1. Syntax error - общая синтаксическая ошибка. 2. В коде отсутствует ссылка на df[“”]. 3. В коде сущесвует ссылка на df[“”].

#### Net

Данный модуль содержит реализацию нейронной сети.

##### Класс Net

Реализация нейронной сети.

###### **init**

Инициализация нейронной сети.

Аргументы:

* hidden\_cnt - количество нейронов на скрытом слое
* input\_cnt - количество входных нейронов
* output\_cnt - количество выходных нейронов
* act\_function=‘Sigmoid’ - функция активации нейронов

###### get\_torch\_act\_function

Функция преобразования строки названия функции активации к объекту функции активации из PyTorch.

###### forward

Прямое распространение сигнала по нейронной сети.

#### Calc

Модуль управления моделями

##### set\_main

Установка модели как главной.

Аргументы:

model\_uuid — str, id модели

##### load\_model

Загрузка модели из файловой системы.

Аргументы:

model\_uuid — str, id модели

##### save\_state

Сохранить текущее состояние моделей

##### delete\_model

Удалить модель из файловой системы.

Аргументы:

model\_uuid — str, id модели

##### load\_main\_model

Загрузить основную модель для вычислений

##### load\_module

Загрузить данный модуль

##### calc

Вычислить оценку.

Аргументы: input\_dict — dict[str]

Выход: - float - значение оценки - или при ошибке { 'missing\_input' - список названий отсутствующих полей. 'unknown\_input' - список неизвестных названий полей. }

##### save\_answer

Сохранить полученный результат. Создаёт или продолжает файл */logs/<main\_model\_uuid>.csv*.

Аргументы: - input\_dict - словарь полученных входных данных - score - полученная оценка

### Main

Описание доступно по пути /doc.

## Юнит-тесты

Запуск тестов - !rundoctest.bat.

Тесты реализованы при помощи doctest.

Содержание тестов - ниже.

# Очистка перед выполнением тестов  
>>> from nn\_handler.field import Field; Field.clear\_field\_list(); import os  
>>> try: os.rename(os.getcwd()+'/models', 'models-saved'); os.mkdir(os.getcwd()+'/models')  
... except: pass  
  
#

### Модуль nn\_handler.field

#### [FieldType](#класс-fieldtype)

##### import

>>> from nn\_handler.field import FieldType

##### int

>>> int(FieldType.choice)  
1  
>>> int(FieldType.numeric)  
2  
>>> int(FieldType.calc)  
3

#### [Field](#класс-field)

##### import

>>> from nn\_handler.field import Field

##### **init**

###### Количество аргументов - 3

>>> Field()  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 3 required positional arguments: 'name', 'field\_type', and 'ignore'  
  
>>> Field(1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 2 required positional arguments: 'field\_type' and 'ignore'  
  
>>> Field(1,2)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 1 required positional argument: 'ignore'

###### Типы аргументов

* name - str

>>> Field(1, 1, 1)  
Traceback (most recent call last):  
...  
ValueError: field name must be string

* field\_type - [FieldType](#classFieldType)

>>> Field('test',1,1)  
Traceback (most recent call last):  
...  
ValueError: field type must be from enum

* ignore - bool

>>> Field('test',FieldType.choice,1)  
Traceback (most recent call last):  
...  
ValueError: ignore must be bool

###### Доступность полей

>>> f = Field('test',FieldType.choice,True)  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.choice  
>>> assert f.ignore == True  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()  
   
>>> f = Field('test',FieldType.numeric,True)  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.numeric  
>>> assert f.ignore == True  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()  
   
>>> f = Field('test',FieldType.calc,False)  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.calc  
>>> assert f.ignore == False  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()

#### [FieldNameExists](#класс-fieldnameexists)

>>> Field('test', FieldType.calc, False); assert len(Field.df) == 1; Field('test', FieldType.calc, False)  
Traceback (most recent call last):  
...  
nn\_handler.field.FieldNameExists: (<class 'nn\_handler.field.FieldNameExists'>, FieldNameExists(...), "Поле с именем 'test' уже существует.")  
>>> assert len(Field.df) == 1; Field.clear\_field\_list()

#### [QField](#класс-qfield)

##### import

>>> from nn\_handler.field import QField

##### **init**

###### Количество аргументов - 4

>>> QField()  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 4 required positional arguments: 'name', 'field\_type', 'ignore', and 'question'  
  
>>> QField(1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 3 required positional arguments: 'field\_type', 'ignore', and 'question'  
  
>>> QField(1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 2 required positional arguments: 'ignore' and 'question'  
  
>>> QField(1,1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 1 required positional argument: 'question'

###### Типы аргументов:

* name - str

>>> QField(1,1,1,1)  
Traceback (most recent call last):  
...  
ValueError: field name must be string

* field\_type - FieldType

>>> QField('test',1,1,1)  
Traceback (most recent call last):  
...  
ValueError: field type must be from enum

* ignore - bool

>>> QField('test',FieldType.choice,1,1)  
Traceback (most recent call last):  
...  
ValueError: ignore must be bool

* question - str

>>> QField('test',FieldType.choice,False,1)  
Traceback (most recent call last):  
...  
ValueError: field question must be string

###### Доступность полей

>>> f = QField('test',FieldType.choice,True,'test\_question')  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.choice  
>>> assert f.ignore == True  
>>> assert f.question == 'test\_question'  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()  
   
>>> f = QField('test',FieldType.numeric,True,'test\_question')  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.numeric  
>>> assert f.ignore == True  
>>> assert f.question == 'test\_question'  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()  
   
>>> f = QField('test',FieldType.calc,False,'test\_question')  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.calc  
>>> assert f.ignore == False  
>>> assert f.question == 'test\_question'  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()

#### [QFieldChoice](#класс-qfieldchoice)

##### import

>>> from nn\_handler.field import QFieldChoice

##### **init**

###### Количество аргументов: 5

>>> QFieldChoice()  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 5 required positional arguments: 'name', 'ignore', 'question', 'separate', and 'answers'  
  
>>> QFieldChoice(1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 4 required positional arguments: 'ignore', 'question', 'separate', and 'answers'  
  
>>> QFieldChoice(1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 3 required positional arguments: 'question', 'separate', and 'answers'  
  
>>> QFieldChoice(1,1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 2 required positional arguments: 'separate' and 'answers'  
  
>>> QFieldChoice(1,1,1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 1 required positional argument: 'answers'

###### Типы аргументов:

* name - str

>>> QFieldChoice(1,1,1,1,1)  
Traceback (most recent call last):  
...  
ValueError: field name must be string

* ignore - bool

>>> QFieldChoice('test',1,1,1,1)  
Traceback (most recent call last):  
...  
ValueError: ignore must be bool

* question - str

>>> QFieldChoice('test',False,1,1,1)  
Traceback (most recent call last):  
...  
ValueError: field question must be string

* separate - bool

>>> QFieldChoice('test',False,'test\_qestion',1,1)  
Traceback (most recent call last):  
...  
ValueError: separate must be bool

* answers - list[str] если separate == True или dict[str, number] если separate == False

>>> QFieldChoice('test',False,'test\_question',False,1)  
Traceback (most recent call last):  
...  
ValueError: choice field answers on non-separate field must be dict  
  
>>> QFieldChoice('test',False,'test\_qauestion',False,['one','two'])  
Traceback (most recent call last):  
...  
ValueError: choice field answers on non-separate field must be dict  
  
>>> QFieldChoice('test',False,'test\_question',False,{1:2,'test':2})  
Traceback (most recent call last):   
...  
ValueError: all answer keys must be str  
  
>>> QFieldChoice('test',False,'test\_question',False,{'test':1,'test2':'qq'})  
Traceback (most recent call last):  
...  
ValueError: all answer values must be int of float  
  
>>> QFieldChoice('test',False,'test\_question',True,1)  
Traceback (most recent call last):  
...  
ValueError: choice filed answers on separate field must be list  
  
>>> QFieldChoice('test',False,'test\_question',True,{'test':1})  
Traceback (most recent call last):  
...  
ValueError: choice filed answers on separate field must be list  
  
>>> QFieldChoice('test',False,'test\_question',True,['hi',1])  
Traceback (most recent call last):  
...  
ValueError: all answer items must be str

###### Доступность полей

>>> f=QFieldChoice('test',False,'test\_question',True,['hi','all'])  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.choice  
>>> assert f.ignore == False  
>>> assert f.question == 'test\_question'  
>>> assert f.separate == True  
>>> assert len(f.answers) == 2  
>>> assert f.answers[0] == 'hi'  
>>> assert f.answers[1] == 'all'  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()  
  
>>> f=QFieldChoice('test',True,'test\_question',True,['hi'])  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.choice  
>>> assert f.ignore == True  
>>> assert f.question == 'test\_question'  
>>> assert f.separate == True  
>>> assert len(f.answers) == 1  
>>> assert f.answers[0] == 'hi'  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()  
  
>>> f=QFieldChoice('test',False,'test\_question',False,{'hi':1})  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.choice  
>>> assert f.ignore == False  
>>> assert f.question == 'test\_question'  
>>> assert f.separate == False  
>>> assert len(f.answers) == 1  
>>> assert f.answers['hi'] == 1  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()  
  
>>> f=QFieldChoice('test',True,'test\_question',False,{'hi':1, 'all':0.5})  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.choice  
>>> assert f.ignore == True  
>>> assert f.question == 'test\_question'  
>>> assert f.separate == False  
>>> assert len(f.answers) == 2  
>>> assert f.answers['hi'] == 1  
>>> assert f.answers['all'] == 0.5  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()

#### [QFieldNumeric](#класс-qfieldnumeric)

##### import

>>> from nn\_handler.field import QFieldNumeric

##### **init**

###### Количество аргументов: 5

>>> QFieldNumeric()  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 5 required positional arguments: 'name', 'ignore', 'question', 'mean', and 'std'  
  
>>> QFieldNumeric(1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 4 required positional arguments: 'ignore', 'question', 'mean', and 'std'  
  
>>> QFieldNumeric(1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 3 required positional arguments: 'question', 'mean', and 'std'  
  
>>> QFieldNumeric(1,1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 2 required positional arguments: 'mean' and 'std'  
  
>>> QFieldNumeric(1,1,1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 1 required positional argument: 'std'

###### Типы аргументов:

* name - str

>>> QFieldNumeric(1,1,1,1,1)  
Traceback (most recent call last):  
...  
ValueError: field name must be string

* ignore - bool

>>> QFieldNumeric('test',1,1,1,1)  
Traceback (most recent call last):  
...  
ValueError: ignore must be bool

* question - str

>>> QFieldNumeric('test',False,1,1,1)  
Traceback (most recent call last):  
...  
ValueError: field question must be string

* mean - float

>>> QFieldNumeric('test',False,'test\_question',1,1)  
Traceback (most recent call last):  
...  
ValueError: numeric field mean must be float

* std - float, non-negative

>>> QFieldNumeric('test',False,'test\_question',1.0,1)  
Traceback (most recent call last):  
...  
ValueError: numeric field std must be float  
  
>>> QFieldNumeric('test',False,'test\_question',1.0,-1.0)  
Traceback (most recent call last):  
...  
ValueError: deviation cannot be negative

###### Доступность полей

>>> f=QFieldNumeric('test',False,'test\_question',1.0,2.0)  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.numeric  
>>> assert f.ignore == False  
>>> assert f.question == 'test\_question'  
>>> assert f.mean == 1.0  
>>> assert f.std == 2.0  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()  
  
  
>>> f=QFieldNumeric('test',True,'test\_question',1.0,2.0)  
>>> assert f.name == 'test'  
>>> assert f.field\_type == FieldType.numeric  
>>> assert f.ignore == True  
>>> assert f.question == 'test\_question'  
>>> assert f.mean == 1.0  
>>> assert f.std == 2.0  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()

#### [FieldCalc](#класс-fieldcalc)

##### import

>>> from nn\_handler.field import FieldCalc

##### **init**

###### Количество аргументов:

>>> FieldCalc()  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 5 required positional arguments: 'name', 'ignore', 'code', 'mean', and 'std'  
  
>>> FieldCalc(1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 4 required positional arguments: 'ignore', 'code', 'mean', and 'std'  
  
>>> FieldCalc(1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 3 required positional arguments: 'code', 'mean', and 'std'  
  
>>> FieldCalc(1,1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 2 required positional arguments: 'mean' and 'std'  
  
>>> FieldCalc(1,1,1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 1 required positional argument: 'std'

###### Типы аргументов:

* name - str

>>> FieldCalc(1,1,1,1,1)  
Traceback (most recent call last):  
...  
ValueError: field name must be string

* ignore - bool

>>> FieldCalc('test',1,1,1,1)  
Traceback (most recent call last):  
...  
ValueError: ignore must be bool

* code - str

>>> FieldCalc('test',False,1,1,1)  
Traceback (most recent call last):  
...  
ValueError: code in calc field is expected to be str

* mean - float

>>> FieldCalc('test',False,'df["test"]',1,1)  
Traceback (most recent call last):  
...  
ValueError: numeric field mean must be float

* std - float, non-negative

>>> FieldCalc('test',False,'df["test"]',1.0,1)  
Traceback (most recent call last):  
...  
ValueError: numeric field std must be float  
  
>>> FieldCalc('test',False,'df["test"]',1.0,-1.0)  
Traceback (most recent call last):  
...  
ValueError: deviation cannot be negative

###### Доступность полей

>>> f = FieldCalc('test',False,'df["test"]',1.0,2.0)  
>>> assert f.name == 'test'  
>>> assert f.ignore == False  
>>> assert f.code == 'df["test"]'  
>>> assert f.mean == 1.0  
>>> assert f.std == 2.0  
>>> assert 'test' in Field.df  
>>> Field.clear\_field\_list()

#### [CalcCodeParseError](#класс-calccodeparseerror)

1. Syntax error

>>> FieldCalc('test',False,'def t',1.0,1.0)  
Traceback (most recent call last):  
...  
nn\_handler.field.CalcCodeParseError: invalid syntax

1. Отсутсвие в коде ссылки на поле

>>> FieldCalc('test',False,'2+2',1.0,1.0)  
Traceback (most recent call last):  
...  
nn\_handler.field.CalcCodeParseError: В коде отсутствует ссылка на df["test"]

1. Наличие ссылки на ещё не добавленное поле

>>> FieldCalc('test',False,'df["test"]=df["illegal\_field"]',1.0,1.0)  
Traceback (most recent call last):  
...  
nn\_handler.field.CalcCodeParseError: В коде используется ссылка на неизвестное поле с именем illegal\_field

### nn\_handler.net

#### import

>>> import os; os.environ['VERBOUSE'] = 'False'; from nn\_handler.net import Net

#### get\_torch\_act\_function

>>> Net.get\_torch\_act\_function()  
Traceback (most recent call last):  
...  
TypeError: get\_torch\_act\_function() missing 1 required positional argument: 'name'  
  
>>> Net.get\_torch\_act\_function(1)  
Traceback (most recent call last):  
...  
ValueError: Expected string for a name of activation function  
  
>>> Net.get\_torch\_act\_function('test')  
Traceback (most recent call last):  
...  
ValueError: Activation function "test" is not defined  
  
>>> Net.get\_torch\_act\_function('test', 1)  
Traceback (most recent call last):  
...  
TypeError: get\_torch\_act\_function() takes 1 positional argument but 2 were given  
  
>>> Net.get\_torch\_act\_function('tanh')  
Tanh()  
  
>>> Net.get\_torch\_act\_function('sigmoid')  
Sigmoid()

#### **init**

##### Количество аргументов: 3

>>> Net()  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 3 required positional arguments: 'hidden\_cnt', 'input\_count', and 'output\_count'  
>>> Net(1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 2 required positional arguments: 'input\_count' and 'output\_count'  
>>> Net(1,1)  
Traceback (most recent call last):  
...  
TypeError: \_\_init\_\_() missing 1 required positional argument: 'output\_count'

##### Аргументы:

3 натуральных числа + (опционально) название функции активации

>>> Net('hello, world',1,1)  
Traceback (most recent call last):  
...  
ValueError: hidden\_cnt must be a natrual number  
>>> Net(1,'hello, world',1)  
Traceback (most recent call last):  
...  
ValueError: input\_count must be a natrual number  
  
>>> Net(1,1,'hello, world' )  
Traceback (most recent call last):  
...  
ValueError: output\_count must be a natrual number  
>>> Net(-1,1,1)  
Traceback (most recent call last):  
...  
ValueError: hidden\_cnt must be a natrual number  
  
>>> Net(1,-1,1)  
Traceback (most recent call last):  
...  
ValueError: input\_count must be a natrual number  
  
>>> Net(1,1,-1)  
Traceback (most recent call last):  
...  
ValueError: output\_count must be a natrual number  
>>> Net(1,1,1,'test')  
Traceback (most recent call last):  
...  
ValueError: Activation function "test" is not defined

### nn\_handler.calc

#### import

>>> from nn\_handler import calc

#### load

Загрузка первой модели

>>> calc.load\_module()  
>>> assert len(calc.models\_state) == 0  
  
# Скопируем модель test из папки models-test в models с случайным uuid и попробуем загрузить  
>>> import shutil; test\_path = os.getcwd()+'/models-test/test/';model\_path = os.getcwd()+'/models/'; import uuid; test\_uuid = str(uuid.uuid4()); os.mkdir(model\_path+test\_uuid); r1 = shutil.copyfile(test\_path+'model\_data.pt',model\_path+test\_uuid+'/model\_data.pt'); r2 = shutil.copyfile(test\_path+'model\_info.json',model\_path+test\_uuid+'/model\_info.json'); calc.load\_model(test\_uuid);  
True  
>>> assert len(calc.models\_state['models']) == 1  
>>> assert calc.models\_state['models'][test\_uuid]['name'] == 'test'  
>>> assert calc.models\_state['models'][test\_uuid]['model\_data'] == '7c06541fa48e77b5e2a49fd25041619e'  
>>> assert calc.models\_state['models'][test\_uuid]['model\_info'] == 'eca818e482b6aa06bf6d19b5c411a87e'

Попытка повторной загрузки

# Загрузка той же модели  
>>> test\_uuid2 = str(uuid.uuid4()); assert test\_uuid2 != test\_uuid; os.mkdir(model\_path+test\_uuid2); r1 = shutil.copyfile(test\_path+'model\_data.pt',model\_path+test\_uuid2+'/model\_data.pt'); r2 = shutil.copyfile(test\_path+'model\_info.json',model\_path+test\_uuid2+'/model\_info.json'); calc.load\_model(test\_uuid2)  
False  
  
# Загрузка иной модели  
>>> test\_uuid2 = str(uuid.uuid4()); assert test\_uuid2 != test\_uuid; test\_path = os.getcwd()+'/models-test/test2/'; os.mkdir(model\_path+test\_uuid2); r1 = shutil.copyfile(test\_path+'model\_data.pt',model\_path+test\_uuid2+'/model\_data.pt'); r2 = shutil.copyfile(test\_path+'model\_info.json',model\_path+test\_uuid2+'/model\_info.json'); calc.load\_model(test\_uuid2)  
True

#### set\_main

>>> calc.set\_main(1)  
Traceback (most recent call last):  
...  
ValueError: На вход ожидалось str  
  
>>> calc.set\_main('Unknown')  
Traceback (most recent call last):  
...  
ValueError: Модель с таким идентификатором не была загружена  
  
>>> calc.set\_main(test\_uuid)  
  
>>> calc.set\_main(test\_uuid)  
  
>>> calc.set\_main(test\_uuid2)  
  
>>> calc.set\_main(test\_uuid)

#### calc

>>> calc.calc(1)  
Traceback (most recent call last):  
...  
ValueError: Ожидался ввод типа dict  
>>> calc.calc({})  
{'missing\_input': ['Пол', 'Возраст', 'Образование', 'Доход', 'Семейное положение', 'Рост (см)', 'Вес (кг)', 'Детство', 'Потеря родителей', 'Родственники', 'Творчество', 'Умственный труд', 'Близкие', 'Заболевание'], 'unknown\_input': []}  
>>> calc.calc({'Unknown':'input'})  
{'missing\_input': ['Пол', 'Возраст', 'Образование', 'Доход', 'Семейное положение', 'Рост (см)', 'Вес (кг)', 'Детство', 'Потеря родителей', 'Родственники', 'Творчество', 'Умственный труд', 'Близкие', 'Заболевание'], 'unknown\_input': ['Unknown']}  
>>> calc.calc({'Unknown':'input','Потеря родителей':'Нет', 'Семейное положение':'Не женат/замужем', 'Вес (кг)':105, 'Доход':'Менее 20 тыс.', 'Образование':'Среднее', 'Родственники':'Нет', 'Умственный труд':'Нет', 'Близкие':'Нет', 'Рост (см)':180, 'Возраст':21, 'Пол':'Мужской', 'Детство':'Нет', 'Заболевание':'Нет', 'Творчество':'Нет'})  
{'missing\_input': [], 'unknown\_input': ['Unknown']}  
>>> calc.calc({'Потеря родителей':'Нет', 'Семейное положение':'Не женат/замужем', 'Вес (кг)':105, 'Доход':'Менее 20 тыс.', 'Образование':'Среднее', 'Родственники':'Нет', 'Умственный труд':'Нет', 'Близкие':'Нет', 'Рост (см)':180, 'Возраст':21, 'Пол':'Мужской', 'Детство':'Нет', 'Заболевание':'Нет', 'Творчество':'Нет'})  
1.0

#### load(продолжение)

Загрузка системы по сохранённому состоянию

>>> import json  
>>> with open(model\_path+'state.json', 'w') as outfile:   
... json.dump({"models": {test\_uuid: {"name":"test3","model\_data": "7c06541fa48e77b5e2a49fd25041619e", "model\_info": "eca818e482b6aa06bf6d19b5c411a87e"}}, "main\_model":test\_uuid},outfile)  
>>> calc.load\_module()  
>>> assert len(calc.models\_state['models']) == 1  
>>> assert calc.models\_state['models'][test\_uuid]['name'] == 'test3'  
>>> assert calc.models\_state['models'][test\_uuid]['model\_data'] == '7c06541fa48e77b5e2a49fd25041619e'  
>>> assert calc.models\_state['models'][test\_uuid]['model\_info'] == 'eca818e482b6aa06bf6d19b5c411a87e'

TODO: загрузка моделей с некорретными описаниями полей

### main

>>> from fastapi.testclient import TestClient  
>>> import main  
>>> client = TestClient(main.app)

Главная страница

>>> client.get('/').status\_code  
200

Страница тестирования

>>> client.get('/test').status\_code  
200

Страница с результатом

>>> resp = client.get('/result/1.234'); assert resp.status\_code == 200; '1.234' in str(resp.content)  
True

Список моделей

>>> resp = client.get('/models'); resp.status\_code; resp.json()['detail']   
401  
'Not authenticated'  
  
>>> import base64; auth\_header = {"WWW-Authenticate": "Basic", "Authorization":"Basic "+ str(base64.b64encode(("admin"+":"+"12345").encode("utf-8")), "utf-8")}; auth\_fake\_header = {"WWW-Authenticate": "Basic", "Authorization":"Basic "+ str(base64.b64encode(("unknownuser"+":"+"123441245").encode("utf-8")))}  
>>> resp = client.get('/models',headers=auth\_fake\_header); resp.status\_code; resp.json()['detail']  
401  
'Invalid authentication credentials'  
  
>>> resp = client.get('/models',headers=auth\_header); resp.status\_code; resp.json() == [{'name': 'test3', 'uuid': test\_uuid, 'main': True}]  
200  
True

Прохождение диагностики

>>> resp = client.post('/diagnose/', json={}); resp.status\_code; resp.json()  
422  
{'detail': [{'loc': ['body', 'json\_q', 'items'], 'msg': 'field required', 'type': 'value\_error.missing'}]}  
  
>>> resp = client.post('/diagnose/', json={'json\_q':{}}); resp.status\_code; resp.json()   
422  
{'detail': [{'loc': ['body', 'json\_q', 'items'], 'msg': 'field required', 'type': 'value\_error.missing'}]}  
  
>>> resp = client.post('/diagnose/', json={'items':[]}); resp.status\_code; resp.json()  
400  
{'detail': {'missing\_input': ['Пол', 'Возраст', 'Образование', 'Доход', 'Семейное положение', 'Рост (см)', 'Вес (кг)', 'Детство', 'Потеря родителей', 'Родственники', 'Творчество', 'Умственный труд', 'Близкие', 'Заболевание'], 'unknown\_input': []}}  
  
>>> resp = client.post('/diagnose/', json={'items':[{'name':'Unknown','value':'input'}]}); resp.status\_code; resp.json()  
400  
{'detail': {'missing\_input': ['Пол', 'Возраст', 'Образование', 'Доход', 'Семейное положение', 'Рост (см)', 'Вес (кг)', 'Детство', 'Потеря родителей', 'Родственники', 'Творчество', 'Умственный труд', 'Близкие', 'Заболевание'], 'unknown\_input': ['Unknown']}}  
  
>>> resp = client.post('/diagnose/', json={'items':[{'name':'Потеря родителей','value':'Нет'},{ 'name':'Семейное положение','value':'Не женат/замужем'},{ 'name':'Вес (кг)','value':105},{ 'name':'Доход','value':'Менее 20 тыс.'},{ 'name':'Образование','value':'Среднее'},{ 'name':'Родственники','value':'Нет'},{ 'name':'Умственный труд','value':'Нет'},{ 'name':'Близкие','value':'Нет'},{ 'name':'Рост (см)','value':180},{ 'name':'Возраст','value':21},{ 'name':'Пол','value':'Мужской'},{ 'name':'Детство','value':'Нет'},{ 'name':'Заболевание','value':'Нет'},{ 'name':'Творчество','value':'Нет'}]}); resp.status\_code; resp.json()  
200  
1.0  
  
>>> resp = client.post('/diagnose/', json={'items':[{'name':'Потеря родителей','value':'Нет'},{ 'name':'Семейное положение','value':'Не женат/замужем'},{ 'name':'Вес (кг)','value':105},{ 'name':'Доход','value':'Менее 20 тыс.'},{ 'name':'Образование','value':'Среднее'},{ 'name':'Родственники','value':'Нет'},{ 'name':'Умственный труд','value':'Нет'},{ 'name':'Близкие','value':'Нет'},{ 'name':'Рост (см)','value':180},{ 'name':'Возраст','value':21},{ 'name':'Пол','value':'Мужской'},{ 'name':'Детство','value':'Нет'},{ 'name':'Заболевание','value':'Нет'},{ 'name':'Творчество','value':'Нет'}]}); resp.status\_code; resp.json()  
200  
1.0

Загрузка моделей

>>> resp = client.post('/upload\_model/',{}); resp.status\_code; resp.json()['detail']  
401  
'Not authenticated'  
  
>>> resp = client.post('/upload\_model/',{}, headers=auth\_fake\_header); resp.status\_code; resp.json()['detail']  
401  
'Invalid authentication credentials'  
  
>>> resp = client.post('/upload\_model/',{},headers=auth\_header); resp.status\_code; resp.json()['detail']  
422  
[{'loc': ['body', 'model\_info'], 'msg': 'field required', 'type': 'value\_error.missing'}, {'loc': ['body', 'model\_data'], 'msg': 'field required', 'type': 'value\_error.missing'}]  
  
>>> m\_info\_f = open(test\_path+"model\_info.json", 'r'); m\_data\_f = open(test\_path+"model\_data.pt", 'rb'); resp = client.post('/upload\_model/', files={"model\_info": ("model\_info.json", m\_info\_f, ".json"),"model\_data":("model\_data.pt",m\_data\_f, ".pt")}, headers=auth\_header); m\_info\_f.close(); m\_data\_f.close(); resp.status\_code; resp.json()['Success']; r\_uuid = uuid.UUID(resp.json()['UUID']); r\_uuid = resp.json()['UUID']; resp = client.get('/models',headers=auth\_header); resp.status\_code; resp = resp.json(); len(resp) == 2; resp[0]['name'] == 'test3'; resp[0]['uuid'] == test\_uuid; resp[0]['main'] == True; resp[1]['name'] == r\_uuid; resp[1]['uuid'] == r\_uuid; resp[1]['main'] == False  
200  
True  
200  
True  
True  
True  
True  
True  
True  
True  
  
>>> m\_info\_f = open(test\_path+"model\_info.json", 'r'); m\_data\_f = open(test\_path+"model\_data.pt", 'rb'); resp = client.post('/upload\_model/', files={"model\_info": ("model\_info.json", m\_info\_f, ".json"),"model\_data":("model\_data.pt",m\_data\_f, ".pt")}, headers=auth\_header); m\_info\_f.close(); m\_data\_f.close(); resp.status\_code; resp.json()['Success']  
200  
False  
>>>

Смена главной модели

>>> client.post('/set\_main').status\_code  
404  
  
>>> client.post('/set\_main/').status\_code  
404  
  
>>> client.post('/set\_main/123').status\_code  
401  
  
>>> client.post('/set\_main/123',headers=auth\_header).status\_code  
404  
  
>>> client.post('/set\_main/'+r\_uuid,headers=auth\_header).status\_code; resp = client.get('/models',headers=auth\_header); resp.status\_code; resp = resp.json(); len(resp) == 2; resp[0]['name'] == 'test3'; resp[0]['uuid'] == test\_uuid; resp[0]['main'] == False; resp[1]['name'] == r\_uuid; resp[1]['uuid'] == r\_uuid; resp[1]['main'] == True  
200  
200  
True  
True  
True  
True  
True  
True  
True  
  
>>> client.post('/set\_main/'+test\_uuid,headers=auth\_header).status\_code; resp = client.get('/models',headers=auth\_header); resp.status\_code; resp = resp.json(); len(resp) == 2; resp[0]['name'] == 'test3'; resp[0]['uuid'] == test\_uuid; resp[0]['main'] == True; resp[1]['name'] == r\_uuid; resp[1]['uuid'] == r\_uuid; resp[1]['main'] == False  
200  
200  
True  
True  
True  
True  
True  
True  
True

Удаление модели

>>> client.delete('/delete\_model').status\_code  
404  
  
>>> client.delete('/delete\_model/123').status\_code  
401  
  
>>> client.delete('/delete\_model/123', headers=auth\_header).status\_code  
404  
  
>>> client.delete('/delete\_model/'+r\_uuid, headers=auth\_header).status\_code; resp = client.get('/models',headers=auth\_header); resp.status\_code; resp = resp.json(); len(resp) == 1; resp[0]['name'] == 'test3'; resp[0]['uuid'] == test\_uuid; resp[0]['main'] == True  
200  
200  
True  
True  
True  
True

### Очистка после выполнения тестов

>>> import os; import shutil; shutil.rmtree(os.getcwd()+'/models', ignore\_errors=True)  
>>> try:os.rename(os.getcwd()+'/models-saved', 'models')   
... except: pass