```
!pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.8-cp311-cp311-
manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in
/usr/local/lib/python3.11/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.11/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in
/usr/local/lib/python3.11/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in
/usr/local/lib/python3.11/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in
/usr/local/lib/python3.11/dist-packages (from catboost) (1.15.2)
Requirement already satisfied: plotly in
/usr/local/lib/python3.11/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-
packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost)
(2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost)
(2025.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost)
(1.3.2)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost)
(4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost)
(1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost)
(24.2)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost)
(11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost)
(3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.11/dist-packages (from plotly->catboost)
(9.1.2)
Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl (99.2
MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.2/99.2 MB 7.0 MB/s eta
0:00:00
```

```
import scipy.stats as sps
import numpy as np
import pandas as pd
from tqdm.notebook import tqdm
```

```
from graphviz import Digraph

from statsmodels.stats.proportion import proportion_confint
from statsmodels.formula.api import ols
from scipy.stats import ttest_rel

from joblib import Parallel, delayed

import warnings
from catboost import CatBoostRegressor
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error, r2_score

import torch
from torch.utils.data import DataLoader, TensorDataset
import torch.nn as nn
import torch.optim as optim

warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="darkgrid", font_scale=1, palette="Set2")
```

Рассмотрим данные о продажах 45 супермаркетов торговой сети Walmart с 2010 по 2012 годы. Датасет содержит 4 файла:

- `features.csv` — информация по неделям о средней температуре воздуха, цене на топливо, а также различная информация о рекламных акциях Walmart;
- `stores.csv` — информация о размере магазинов;
- `train.csv` — информация о недельных продажах для каждого отдела каждого магазина;
- `test.csv` — в данной задаче не требуется.

Цель: построить дизайн АБ-теста и ответить на вопрос, сколько магазинов потребуется для проведения АВ-теста.

Срок АБ-теста: от 4 до 8 недель.

Ожидаемый эффект: +5% к продажам магазина суммарно по всем отделам.

Хотим попробовать и сравнить разные подходы CUPED и стратификации, используя различные варианты дополнительных данных:

- без использования доп. данных, то есть простой t-test;
- данные о продажах предпериода, то есть стандартный CUPED;
- категориальные признаки, то есть простая стратификация;
- вещественные признаки, то есть CUPED при использовании различных ковариат;
- прогнозирование продаж с помощью различных моделей (CUPAC):
  - линейные модели;

- градиентный бустинг;
- нейронные сети.

```
features = pd.read_csv('/content/features.csv')
stores = pd.read_csv('/content/stores.csv')
train = pd.read_csv('/content/train.csv')
```

Предобработаем некоторые данные и соединим таблицы:

```
train['Date'] = pd.to_datetime(train['Date'])
features['Date'] = pd.to_datetime(features['Date'])

data = pd.merge(train, stores, on='Store', how='left')

data = pd.merge(data, features, on=['Store', 'Date'], how='left')

data.drop(columns='IsHoliday_y', inplace = True)

data['IsHoliday_x'] = data['IsHoliday_x'].astype(int)

data
```

```
{"type":"dataframe","variable_name":"data"}
```

Посмотрим из какого интервала количества пользователей можно примерно взять для выборки:

```
((data['Date'] > '2011-01-06') & (data['Date'] < '2011-02-06')).sum(),
((data['Date'] > '2011-01-06') & (data['Date'] < '2011-03-06')).sum()
```

```
(14621, 26394)
```

Оценка количества элементов в выборке:

```
def get_mde(alpha, beta, sample_size, var_x):
    '''Расчитывает MDE

    :param alpha: желаемая вероятность ошибки первого рода
    :param beta: желаемая мощность
    :param sample_size: размер выборок
    :param var_x, var_y: дисперсии выборок
    :returns: теоретический MDE
    '''
    var_y=var_x
    q_sum = sps.norm.ppf(1 - alpha) + sps.norm.ppf(beta)
    return q_sum / np.sqrt(sample_size) * np.sqrt(var_x + var_y)


def get_sample_size(alpha, beta, mde, var_x):
    '''Расчитывает размер выборки для детектирование MDE
    :param alpha: желаемая вероятность ошибки первого рода
    :param beta: желаемая мощность
    :param mde: необходимый эффект
    :param var_x, var_y: дисперсии выборок
    :returns: необходимый размер выборки
    '''
    var_y=var_x
    q_sum = sps.norm.ppf(1 - alpha) + sps.norm.ppf(beta)
    return np.ceil((q_sum ** 2) * (var_x + var_y) / (mde *
mde)).astype(int)

def plot(grid, mdes, sample_sizes, title='', third_dimension=None,
```

```
label=''):
    ''' Строит графики MDE и размера выборки по сетке значений. '''

    _, axes = plt.subplots(1, 2, figsize=(15, 6))

    plt.suptitle(title, fontsize=20)

    if third_dimension is not None:
        for mde, sample_size, third in zip(mdes, sample_sizes,
third_dimension):
            axes[0].plot(grid.reshape(-1), mde, lw=3, label=f'{label}
= {third[0]}')
            axes[0].legend()
            axes[1].plot(grid.reshape(-1), sample_size, lw=3)
    else:
        axes[0].plot(grid.reshape(-1), mdes, lw=3)
        axes[1].plot(grid.reshape(-1), sample_sizes, lw=3)


    axes[0].set_title('MDE')
    axes[0].set_xlabel(title)
    axes[1].set_title('Размер выборки')
    axes[1].set_xlabel(title)

    plt.show()

var_x = np.std(data['Weekly_Sales']) ** 2
mde = 0.05 * np.mean(data['Weekly_Sales'])
alpha = 0.05
beta = 0.8

get_sample_size(alpha, beta, mde, var_x)
```

```
9989
```

Получили 10к значений, для каждой выборки, что вписываетя в наш эксперимент, так как за месяц мы получаем примерно 15к, а за два порядка 30 к, а нам потребуется 10к сэмплов на контроль и тест

st

```
def generate_subarray(
    data,
    test_control_size,
    fraq_test = 0.5,
    target_column="Weekly_Sales",
):
    """
    Генерация выборок 2-х групп посредством случайного сэмплирования
строк данных

    Аргументы:
    data -- данные
    test_control_size -- общее количество строк в тесте и контроле
    fraq_test -- доля тестовой группы
    target_column -- целевая метрика

    Возвращает:
    test_sample, control_sample -- данные для тестовой и контрольной
групп
    """
    # общее количество строк для теста и контроля
```

```
    total_samples = min(test_control_size, len(data))
    test_size = int(total_samples * fraq_test)
    control_size = total_samples - test_size

    # случайное сэмплирование строк для теста и контроля
    test_sample = data.sample(n=test_size, replace=False)
    remaining_data = data.drop(test_sample.index)
    control_sample = remaining_data.sample(n=control_size,
replace=False)

    return test_sample, control_sample

def absolute_ttest(x, y):
    """
    Абсолютный t-test.

    Аргументы:
    x, y -- выборки одинакового размера
    alpha -- уровень значимости

    Возвращает:
    stat -- статистика критерия
    pvalue
    left_bound, right_bound -- границы дов. интервала
    """
    res = sps.ttest_ind(x, y)
    stat, pvalue = res.statistic, res.pvalue
    left_bound, right_bound = res.confidence_interval()

    return stat, pvalue, left_bound, right_bound
def relative_ttest(x, y, alpha=0.05):
    """
    Относительный t-test.

    Аргументы:
    x, y -- выборки одинакового размера
    alpha -- уровень значимости

    Возвращает:
    stat -- статистика критерия
    pvalue
    left_bound, right_bound -- границы дов. интервала
    """

    n = len(x)
    x_mean = x.mean()
    y_mean = y.mean()

    stat = x_mean / y_mean - 1
    var = x.var() / (y_mean**2) + y.var() * (x_mean**2) / (y_mean**4)
    std = np.sqrt(var)

    z_stat = np.sqrt(n) * stat / std
    pvalue = 2 * sps.norm.sf(np.abs(z_stat))

    q = sps.norm.ppf(1 - alpha / 2)
    left_bound = stat - q * std / np.sqrt(n)
    right_bound = stat + q * std / np.sqrt(n)

    return stat, pvalue, left_bound, right_bound
def estimate_reject_prob(n_rejects, n_iter):
    """
```

```
    Оценка вероятности отвержения критерия и ее дов. интервала.
    Используется для оценки вероятности ошибки первого рода и мощности

    Аргументы:
    n_rejects -- количество отвержений H_0 в эксперименте
    n_iter -- количество экспериментов

    Возвращает:
    prob_reject -- оценка вероятности отвержения критерия
    left_bound, right_bound -- границы соотв. дов. интервала
    """

    prob_reject = n_rejects / n_iter
    left_bound, right_bound = proportion_confint(n_rejects, n_iter,
method="wilson")

    return prob_reject, left_bound, right_bound


def visualization(
    prob_reject,
    left_bound,
    right_bound,
    show_pvals=False,
    pvals=None,
    alpha=0.05,
    figsize=(7, 2),
    title=None,
):
    """
    Отрисовка интервала для вероятности отвержения критерия
    и гистограммы p-value (опционально)
    prob_reject -- оценка вероятности отвержения
    left_bound, right_bound -- границы доверительного интервала
    alpha -- теор вероятность ошибки первого рода
    show_pvals -- показывать ли распределение p-value
    pvals -- массив из p-value
    figsize -- размер фигуры matplotlib
    """

    # построение гистограммы p-value (опционально)
    if show_pvals:
        with sns.axes_style("whitegrid"):
            plt.figure(figsize=figsize)
            plt.subplot(1, 2, 1)
            plt.hist(
                pvals,
                bins=np.linspace(0, 1, 21),
                alpha=0.7,
                weights=np.ones(len(pvals)) / len(pvals),
            )
            plt.title("Распределение p-value")

    # отрисовка интервала для вероятности отвержения критерия
    with sns.axes_style("whitegrid"):
        if show_pvals:
            plt.subplot(1, 2, 2)
        else:
            plt.figure(figsize=figsize)
        plt.hlines(0, 0, 1, color="black", lw=2, alpha=0.6)
        plt.vlines(alpha, -1, 1, color="red", lw=5, linestyle="--",
```

```
alpha=0.6)
        plt.fill_between(
            [left_bound, right_bound], [0.15] * 2, [-0.15] * 2,
color="green", alpha=0.6
        )
        plt.scatter(prob_reject, 0, s=300, marker="*", color="red")
        plt.xlim((min(alpha, left_bound) - 1e-3, max(alpha,
right_bound) + 1e-3))
        plt.title(
            f"Доля отвержений = {100*prob_reject:.2f}%, "
            f"интервал ({100*left_bound:.2f}%, {100*right_bound:.2f}
%)"
        )
        plt.suptitle(title)
        plt.ylim((-0.5, 0.5))
        plt.yticks([])
        plt.tight_layout()
        plt.show()


def draw_power(
    powers,
    left_powers,
    right_powers,
    effects_list=np.linspace(0, 0.1, 11),
    label=None,
    title="Графики мощности",
    new_figure=False,
):
    """
    Построение и отрисовка графика мощности критерия.

    Аргументы:
    effects -- сетка эффектов
    real_alpha -- оценка реальной мощности
    left_alpha, right_alpha -- границы соотв. дов. интервала
    title -- заголовок графика
    """

    if new_figure:
        plt.figure(figsize=(10, 4))

    plt.plot(effects_list, powers, label=label, lw=3)
    plt.fill_between(effects_list, left_powers, right_powers,
alpha=0.3)

    if new_figure:
        plt.hlines(
            0.8,
            effects_list[0],
            effects_list[-1],
            color="black",
            alpha=0.5,
            label="Мощность 0.8 (пересечение — MDE)",
        )
    plt.legend()
    plt.xlabel("Размер относительного эффекта")
    plt.ylabel("Мощность")
    plt.title(title)
def add_effect(x, effect, target_name="pilot", relative_effect=True):
    """
```

```
    Функция искусственного добавления эффекта в тестовую выборку

    Аргументы:
    x -- выборка
    effect -- добавляемый эффект

    Возвращает:
    x -- выборка с добавленным эффектом
    """

    x_copy = x.copy()
    if relative_effect:
        x_copy[target_name] *= 1 + effect
    else:
        x_copy[target_name] += effect

    return x_copy


def run_1_iteration(
    test,
    data,
    sample_size,
    generate_samples,
    target_name,
    add_effect,
    effects_list=[0],
    relative_effect=True,
):
    """
    Проведение серии AA-тестов на искусственных выборках.

    Аргументы:
    test -- статистический критерий
    generate_samples -- функция для семплирования выборок
    add_effect -- функция добавления эффекта
    effects_list --  массив размеров добавляемого эффекта (для оценки
мощности)
    relative_effect -- является ли эффект относительным или абсолютным

    Возвращает: pvalue
    """
    # Генерируем выборки
    x_data, y_data = generate_samples(data, sample_size)

    pvals = []
    for effect in effects_list:
        # Добавляем эффект, если хотим оценить мощность
        x_data_cp = add_effect(
            x_data, effect, target_name=target_name,
relative_effect=relative_effect
        )
        # Применяем критерий
        pvals.append(test(x_data_cp, y_data)[1])
    return pvals


def run_experiments(
    test,
    generate_samples,
    data=None,
    add_effect=add_effect,
```

```
    n_iter=10000,
    sample_size=1000,
    effect=0,
    relative_effect=True,
    n_jobs=4,
    alpha=0.05,
    target_name="pilot",
    draw=False,
    title=None,
    show_pvals=True,
):
    """
    Проведение серии AA-тестов на исторических данных.

    Аргументы:
    test -- статистический критерий
    generate_samples -- функция для семплирования выборок
    data -- исторические данные
    n_iter -- количество итераций
    sample_size -- размер выборок
    add_effect -- функция добавления эффекта
    effect -- размер добавляемого эффекта (для оценки мощности)
    relative_effect -- является ли эффект относительным или абсолютным
    alpha -- теор вероятность ошибки первого рода
    target_name -- имя колонки с таргет-метрикой
    """

    # Проведение серии экспериментов
    pvals = Parallel(n_jobs=n_jobs)(
        delayed(run_1_iteration)(
            test, data, sample_size, generate_samples, target_name,
add_effect, [effect], relative_effect
        )
        for _ in tqdm(range(n_iter), leave=False)
    )
    pvals = np.array(pvals).flatten()
    # Подсчет числа отвержений
    n_rejects = (pvals < alpha).sum()

    # Оценка вероятности отвержения
    prob_reject, left_bound, right_bound =
estimate_reject_prob(n_rejects, n_iter)

    # Визуализация
    if draw:
        figsize = (14, 3) if show_pvals else (5, 2)
        visualization(
            prob_reject,
            left_bound,
            right_bound,
            show_pvals,
            pvals,
            alpha,
            figsize,
            title,
        )

    return prob_reject, (left_bound, right_bound)


def estimate_power(
```

```
    test,
    generate_samples,
    data=None,
    n_jobs=4,
    n_iter=10000,
    sample_size=1000,
    effects_list=np.linspace(0, 0.1, 11),
    relative_effect=True,
    alpha=0.05,
    target_name="pilot",
):
    """
    Проведение серии AA-тестов на исторических данных с добавением
разных эффектов.

    Аргументы:
    test -- статистический критерий
    generate_samples -- функция для семплирования выборок
    data -- исторические данные
    n_iter -- количество итераций
    sample_size -- размер выборок
    add_effect -- функция добавления эффекта
    effect -- размер добавляемого эффекта (для оценки мощности)
    relative_effect -- является ли эффект относительным или абсолютным
    alpha -- теор вероятность ошибки первого рода
    target_name -- имя колонки с таргет-метрикой
    """
    # Проведение серии экспериментов
    pvals = Parallel(n_jobs=n_jobs)(
        delayed(run_1_iteration)(
            test,
            data,
            sample_size,
            generate_samples,
            target_name,
            add_effect,
            effects_list,
            relative_effect,
        )
        for _ in tqdm(range(n_iter), leave=False)
    )

    # Подсчет числа отвержений
    n_rejects = (np.array(pvals) < alpha).sum(axis=0)

    # Оценка мощности
    powers, left_bounds, right_bounds =
estimate_reject_prob(n_rejects, n_iter)

    return powers, left_bounds, right_bounds

def regression_ttest(
    x_data,
    y_data,
    cuped=False,
    stratified=False,
    sample_name="pilot",
    treatment_name="treatment",
    covariate_names=[],
    strat_names=[],
):
```

```
    """
    T-test с CUPED/без CUPED, со стратификацией/без реализованный
через линейную регрессию

    Аргументы:
    x, y -- выборки одинакового размера
    :param cuped: применять ли cuped
    :param stratified: применять ли стратификацию
    :param sample_name: имя столбца с целевым признаком
    :param covariate_names: массив имен столбцов-ковариат
    :param strat_names: массив имен стратификационных столбцов

    Возвращает:
    stat -- статистика критерия
    pvalue
    left_bound, right_bound -- границы дов. интервала
    """

    x_data_cp = x_data.copy()
    y_data_cp = y_data.copy()

    # добавляем столбец с индикатором тестовой группы
    x_data[treatment_name] = 1
    y_data[treatment_name] = 0
    # объединяем тест и контроль в один датасет
    data = pd.concat([x_data, y_data])
    # удаляем ненужные столбцы из датасета
    if not cuped:
        covariate_names = []
    if not stratified:
        strat_names = []
    data = data[
        [treatment_name] + list(covariate_names) + list(strat_names) +
[sample_name]
    ]

    # делаем стратификацию по нужным фичам
    data = pd.get_dummies(data, columns=strat_names)

    # определяем имена фичей, на которых будем обучать модель
    feature_names = list(set(data.columns) - set([sample_name]))

    # обучаем модель
    model = ols(f"{sample_name} ~ " + "+".join(feature_names),
data=data).fit()
    # забираем таблицу с результатами
    summary = model.summary2().tables[1]
    # берем из таблицы нужные поля
    stat = summary.loc[treatment_name, "t"]
    pvalue = summary.loc[treatment_name, "P>|t|"]
    left_bound = summary.loc[treatment_name, "[0.025"]
    right_bound = summary.loc[treatment_name, "0.975]"]
    return stat, pvalue, left_bound, right_bound
```

Разделим данные на трейн и тест, для предпосчета среднего значения целевой переменной на тесте

```
train_data, test_data = data.iloc[:int(len(data)/2)],
data.iloc[int(len(data)/2):]
```

```
mean_sales = train_data.groupby(['Store', 'Dept'])
['Weekly_Sales'].mean().reset_index()
mean_sales.rename(columns={'Weekly_Sales': 'Mean_Weekly_Sales'},
inplace=True)

test_data = pd.merge(test_data, mean_sales, on=['Store', 'Dept'],
how='left')

test_data['Mean_Weekly_Sales_is_missing'] =
test_data['Mean_Weekly_Sales'].isna().astype(int)

test_data['Mean_Weekly_Sales'] =
test_data['Mean_Weekly_Sales'].fillna(0)
```

Сохраним данные для общего графика:

```
remember = test_data[['Mean_Weekly_Sales',
'Mean_Weekly_Sales_is_missing']]
```

AB

```
test_control_size = 10000 * 2
fraq_test = 0.5

target_column = "Weekly_Sales"
mean = np.mean(data[target_column])
effects_list = np.linspace(0, 0.1, 16)

n_iter = 100
alpha = 0.05
n_jobs = 4
```

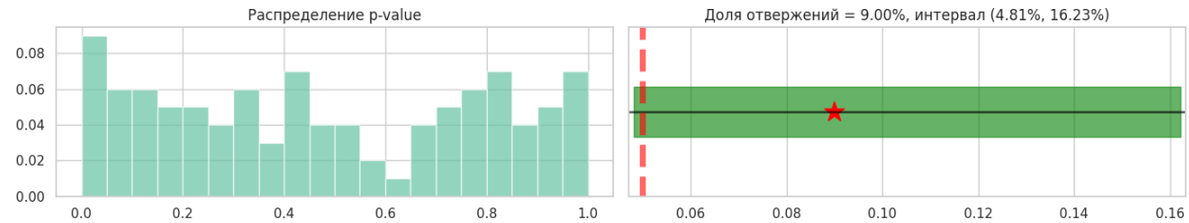
без использования доп. данных, то есть простой t-test;

Возьмем ttest на зависимых выборках, так как иногда могут попаладаться данные про один магазин

```
tests = [
    lambda x, y: ttest_rel(x[target_column], y[target_column]),
]
names = ['Обычный ttest']
for test, name in zip(tests, names):
    print(name)
    run_experiments(
        test=test,
        generate_samples=generate_subarray,
        sample_size = test_control_size,
        target_name=target_column,
        data=data,
        n_iter=n_iter,
        alpha=alpha,
        show_pvals=True,
        draw=True,
        n_jobs=n_jobs,
    );
```

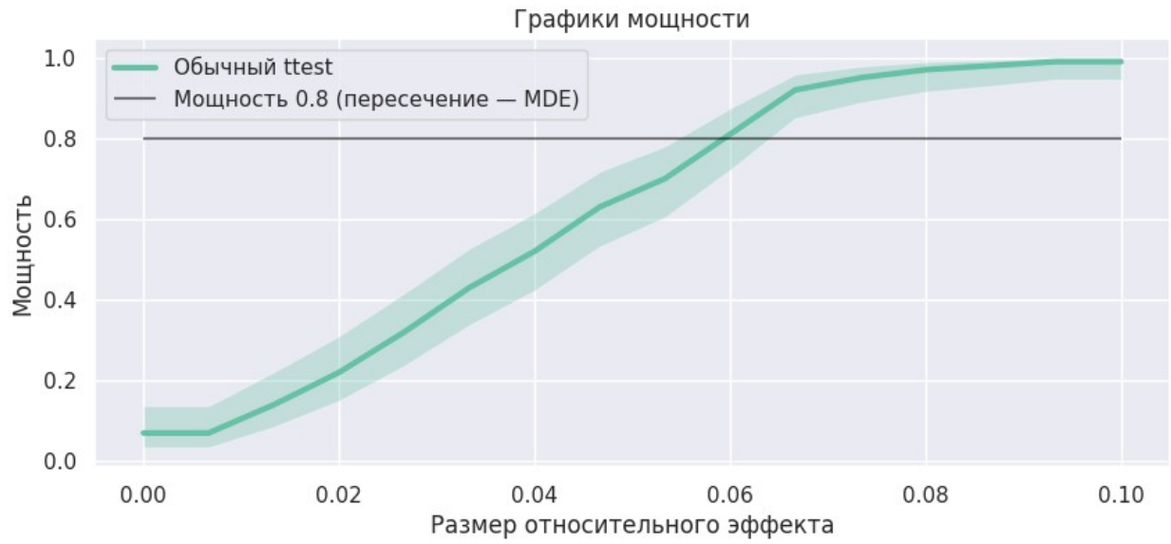
```
Обычный ttest
```


```
{"model_id":"26ac0a0f79d5419399c1b487e2274166","version_major":2,"vers
ion_minor":0}
```




```
tests = [
    lambda x, y: ttest_rel(x[target_column], y[target_column]),
]
names = ['Обычный ttest']

for i, (test, name) in enumerate(zip(tests, names)):
    powers, left_bounds, right_bounds = estimate_power(
        test=test,
        generate_samples=generate_subarray,
        target_name=target_column,
        sample_size = test_control_size,
        effects_list=effects_list,
        data=data,
        n_iter=n_iter,
        alpha=alpha,
        n_jobs=n_jobs,
    )
    draw_power(
        powers,
        left_bounds,
        right_bounds,
        effects_list=effects_list,
        label=name,
        new_figure=(i == 0),
    )
```

{"model_id":"770d73768ff941acaa74596a5ed33278","version_major":2,"version_minor":0}



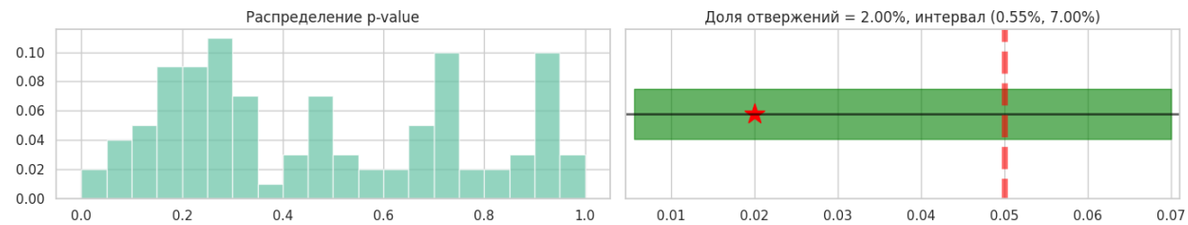
данные о продажах предпериода, то есть стандартный CUPED;

```
tests = [
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
```

```
cuped=True, covariate_names=['Mean_Weekly_Sales',
'Mean_Weekly_Sales_is_missing']),
]
names = ['Cuped по продажам предпериода']

for test, name in zip(tests, names):
    print(name)
    run_experiments(
        test=test,
        generate_samples=generate_subarray,
        sample_size = test_control_size,
        target_name=target_column,
        data=test_data,
        n_iter=n_iter,
        alpha=alpha,
        show_pvals=True,
        draw=True,
        n_jobs=n_jobs,
    );
```

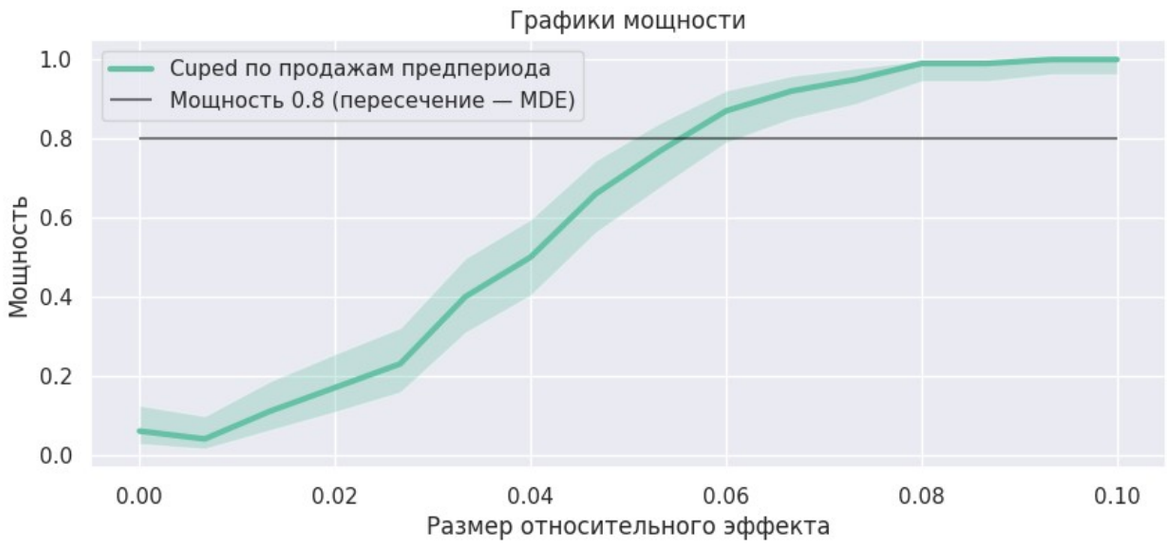
```
Cuped по продажам предпериода
```


```
{"model_id":"9115f96c77cf4ab4bd24f6b00bd7667f","version_major":2,"vers
ion_minor":0}
```


```
tests = [
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['Mean_Weekly_Sales',
'Mean_Weekly_Sales_is_missing']),
]
names = ['Cuped по продажам предпериода']

for i, (test, name) in enumerate(zip(tests, names)):
    powers, left_bounds, right_bounds = estimate_power(
        test=test,
        generate_samples=generate_subarray,
        target_name=target_column,
        sample_size = test_control_size,
        effects_list=effects_list,
        data=test_data,
        n_iter=n_iter,
        alpha=alpha,
        n_jobs=n_jobs,
    )
    draw_power(
        powers,
        left_bounds,
        right_bounds,
        effects_list=effects_list,
        label=name,
        new_figure=(i == 0),
    )
```

{"model_id":"d118c1280a6847c896c4995e98e14404","version_major":2,"version_minor":0}

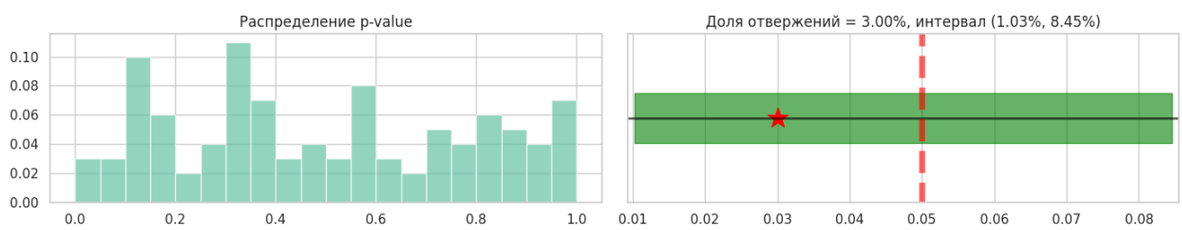


категориальные признаки, то есть простая стратификация;

```
tests = [
    lambda x, y: regression_ttest(x, y, sample_name = target_column, stratified=True, strat_names=['Store', 'Dept','IsHoliday_x', 'Type'],)
]
names = ['ttest со стратификацией']

for test, name in zip(tests, names):
    print(name)
    run_experiments(
        test=test,
        generate_samples=generate_subarray,
        sample_size = test_control_size,
        target_name=target_column,
        data=data,
        n_iter=n_iter,
        alpha=alpha,
        show_pvals=True,
        draw=True,
        n_jobs=n_jobs,
    );
```

```
ttest со стратификацией
```

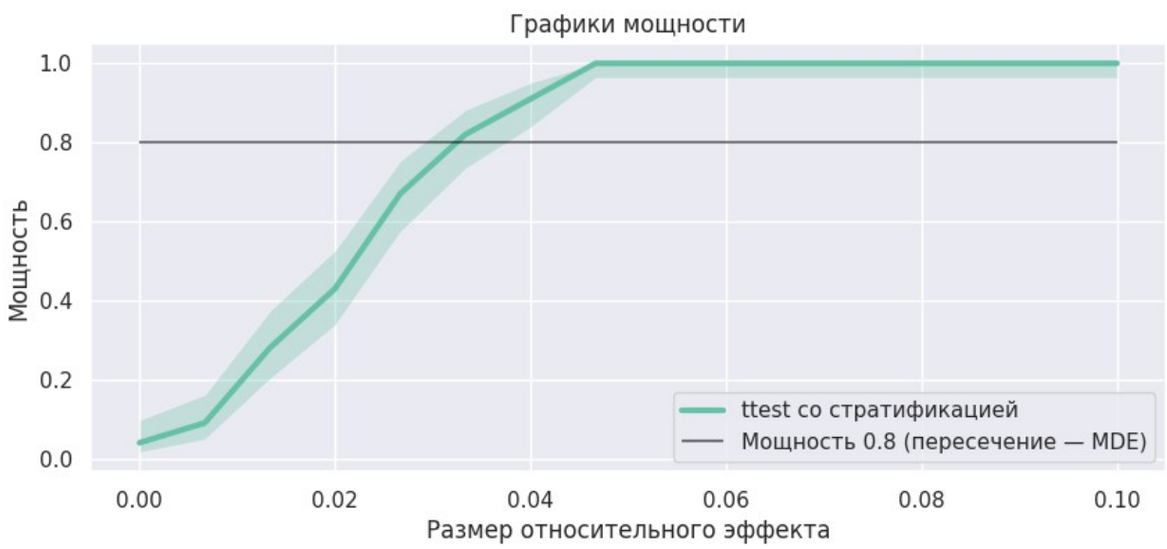
{"model_id":"dc48fe717ebe4ce2afb778fcc84540fc","version_major":2,"version_minor":0}




```
tests = [
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
```

```
stratified=True, strat_names=['Store', 'Dept','IsHoliday_x', 'Type'],)
]
names = ['ttest со стратификацией']

for i, (test, name) in enumerate(zip(tests, names)):
    powers, left_bounds, right_bounds = estimate_power(
        test=test,
        generate_samples=generate_subarray,
        target_name=target_column,
        sample_size = test_control_size,
        effects_list=effects_list,
        data=data,
        n_iter=n_iter,
        alpha=alpha,
        n_jobs=n_jobs,
    )
    draw_power(
        powers,
        left_bounds,
        right_bounds,
        effects_list=effects_list,
        label=name,
        new_figure=(i == 0),
    )
```

{"model_id":"3576c8b119e94e24b18e97a32b0b28a6","version_major":2,"version_minor":0}



вещественные признаки, то есть CUPED при использовании различных ковариат;

```
data
```

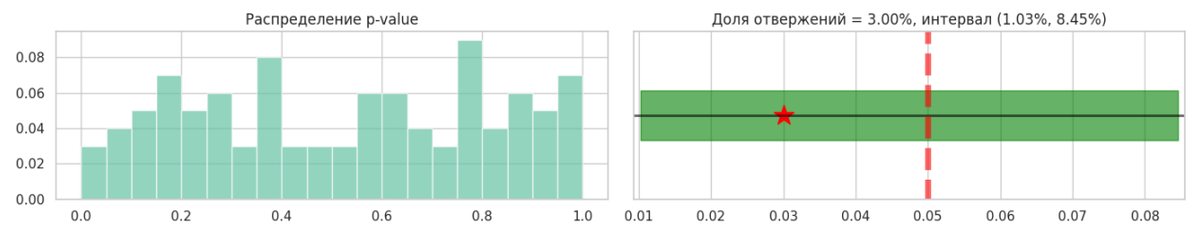
{"type":"dataframe","variable_name":"data"}

```
tests = [
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['Temperature', 'Unemployment',
'Fuel_Price', 'CPI', 'Size']),
]
names = ['Cuped по вещ. ковариатам']
```

```
for test, name in zip(tests, names):
    print(name)
    run_experiments(
        test=test,
        generate_samples=generate_subarray,
        sample_size = test_control_size,
        target_name=target_column,
        data=data,
        n_iter=n_iter,
        alpha=alpha,
        show_pvals=True,
        draw=True,
        n_jobs=n_jobs,
    );
```

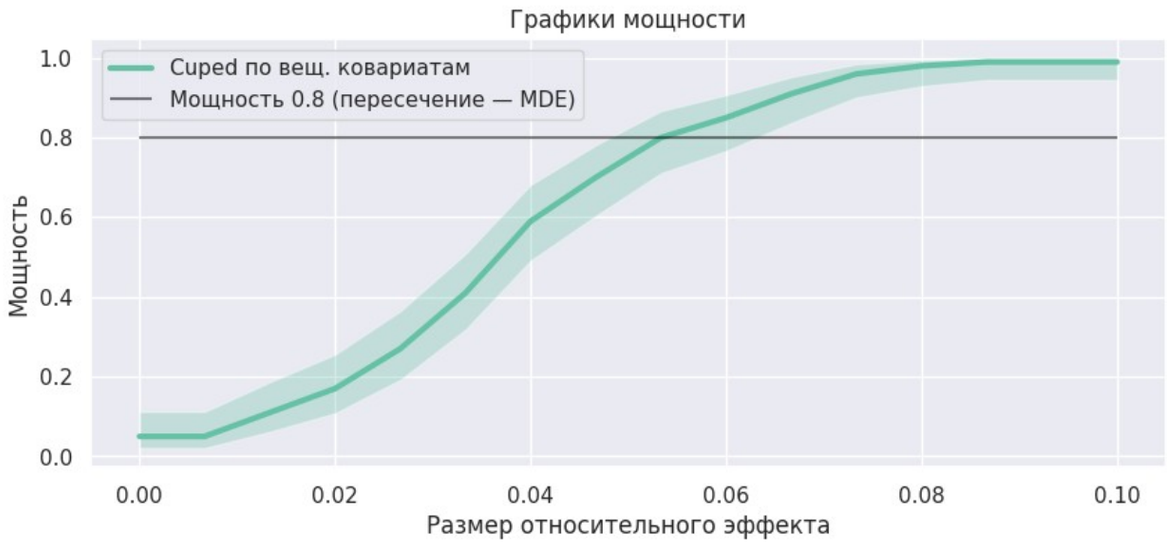
```
Cuped по вещ. ковариатам
```

{"model_id":"caeddbf7e9a246ecb8422b5b6e206ea4","version_major":2,"version_minor":0}




```
tests = [
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['Temperature', 'Unemployment',
'Fuel_Price', 'CPI', 'Size']),
]
names = ['Cuped по вещ. ковариатам']

for i, (test, name) in enumerate(zip(tests, names)):
    powers, left_bounds, right_bounds = estimate_power(
        test=test,
        generate_samples=generate_subarray,
        target_name=target_column,
        sample_size = test_control_size,
        effects_list=effects_list,
        data=data,
        n_iter=n_iter,
        alpha=alpha,
        n_jobs=n_jobs,
    )
    draw_power(
        powers,
        left_bounds,
        right_bounds,
        effects_list=effects_list,
        label=name,
        new_figure=(i == 0),
    )
```

{"model_id":"de95f6a6c28a4fa0b5ef57853e488cf8","version_major":2,"version_minor":0}



прогнозирование продаж с помощью различных моделей (CUPAC):

- линейные модели;
- градиентный бустинг;
- нейронные сети.

Предобработаем данные от пропусков и добавим индикаторы пропусков

```
for col in ['MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4',
'MarkDown5']:
    data[f'{col}_is_missing'] = data[col].isna().astype(int)

    data[col] = data[col].fillna(0)
```

Закодируем категориальные признаки:

```
encoder = OneHotEncoder(drop='first', sparse_output=False)

encoded_type = encoder.fit_transform(data[['Type', 'Store', 'Dept']])

encoded_type_df = pd.DataFrame(encoded_type,
columns=encoder.get_feature_names_out(['Type', 'Store', 'Dept']))

data = pd.concat([data, encoded_type_df], axis=1)

data.drop(columns=['Type', 'Store', 'Dept'], inplace=True)
```

Разобьем данные для обучения - важно не заглядывать в будещее:

```
train_data, test_data = data.iloc[:int(len(data)/2)],
data.iloc[int(len(data)/2):]

train_data['Date'] = (train_data['Date'] - pd.Timestamp('1970-01-
01')).dt.days
test_data['Date'] = (test_data['Date'] - pd.Timestamp('1970-01-
01')).dt.days
```

Отнормируем данные (важно для нейросетей):

```
scaler = StandardScaler()
numeric_features = ['Size', 'Temperature', 'Fuel_Price', 'CPI',
```

```
'Unemployment',
                    'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4',
'MarkDown5']

train_data[numeric_features] =
scaler.fit_transform(train_data[numeric_features])
test_data[numeric_features] =
scaler.fit_transform(test_data[numeric_features])

X_train = train_data.drop(columns=['Weekly_Sales'])
X_test = test_data.drop(columns=['Weekly_Sales'])

y_train = train_data['Weekly_Sales']
y_test = test_data['Weekly_Sales']

remember
```

```
{"type":"dataframe","variable_name":"remember"}
```

```
test_data['Mean_Weekly_Sales'] = remember['Mean_Weekly_Sales'].values
test_data['Mean_Weekly_Sales_is_missing'] =
remember['Mean_Weekly_Sales_is_missing'].values
```

Линейная модель с регуляризацией:

```
linear_model = Ridge(alpha=1.0)
linear_model.fit(X_train, y_train)

y_pred_linear = linear_model.predict(X_test)

r2_linear_test = r2_score(y_test, y_pred_linear)
r2_linear_train = r2_score(y_train, linear_model.predict(X_train))

print("Линейная регрессия:")
print(f"R² (на обучающей): {r2_linear_train}")
print(f"R² (на тестовой): {r2_linear_train}")
mean_squared_error(y_test, y_pred_linear)
```

```
Линейная регрессия:
R² (на обучающей): 0.6498957055983641
R² (на тестовой): 0.6498957055983641

261479246.679933
```

```
test_data['linear_pred'] = y_pred_linear
```

CatBoost:

```
catboost_model = CatBoostRegressor(iterations=1000, learning_rate=0.1,
depth=6, verbose=100)
catboost_model.fit(X_train, y_train)
y_pred_catboost = catboost_model.predict(X_test)

r2_catboost_test = r2_score(y_test, y_pred_catboost)
r2_catboost_train = r2_score(y_test, catboost_model.predict(X_train))

print("\nCatBoost:")
mean_squared_error(y_test, y_pred_catboost)
```

```
0:    learn: 23851.7402544  total: 61.5ms    remaining: 1m 1s
100:  learn: 11304.0422369  total: 2.7s      remaining: 24.1s
200:  learn: 9357.9069585   total: 5.34s     remaining: 21.2s
```

```
300:  learn: 8316.3519103   total: 8.04s    remaining: 18.7s
400:  learn: 7597.1372028   total: 11.6s    remaining: 17.4s
500:  learn: 7026.8110336   total: 15.8s    remaining: 15.7s
600:  learn: 6597.4516651   total: 21.1s    remaining: 14s
700:  learn: 6247.4572080   total: 28.4s    remaining: 12.1s
800:  learn: 5984.2699380   total: 31.2s    remaining: 7.76s
900:  learn: 5788.6571074   total: 34s remaining: 3.74s
999:  learn: 5598.2862390   total: 36.7s    remaining: 0us

CatBoost:

128306244.69662122

test_data['catboost_pred'] = y_pred_catboost
```

Нейронная сеть:

```
X_train_tensor = torch.tensor(X_train.values, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test.values, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32)

train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

train_loader = DataLoader(train_dataset, batch_size=2048,
shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=2048, shuffle=False)

class FC_model(nn.Module):
    def __init__(self, input_size):
        super(FC_model, self).__init__()
        self.fc1 = nn.Linear(input_size, 512)
        self.fc2 = nn.Linear(512, 2048)
        self.fc3 = nn.Linear(2048, 512)
        self.fc4 = nn.Linear(512, 64)
        self.fc5 = nn.Linear(64, 32)
        self.fc6 = nn.Linear(32, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.relu(self.fc4(x))
        x = self.relu(self.fc5(x))
        x = self.fc6(x)
        return x

input_size = X_train.shape[1]
model = FC_model(input_size)

train_losses = []
val_losses = []

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Обучение модели
num_epochs = 20
for epoch in range(num_epochs):
    model.train()
```

```
    epoch_train_loss = 0.0
    for batch_X, batch_y in train_loader:
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y.unsqueeze(1))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        epoch_train_loss += loss.item()

    # Средний лосс на эпоху для обучающей выборки
    epoch_train_loss /= len(train_loader)
    train_losses.append(epoch_train_loss)

    # Валидация
    model.eval()
    epoch_test_loss = 0.0
    with torch.no_grad():
        for batch_X, batch_y in test_loader:
            outputs = model(batch_X)
            loss = criterion(outputs, batch_y.unsqueeze(1))
            epoch_test_loss += loss.item()

    # Средний лосс на эпоху для валидационной выборки
    epoch_test_loss /= len(test_loader)
    val_losses.append(epoch_test_loss)

    print(f"Epoch [{epoch+1}/{num_epochs}], Train Loss:
{epoch_train_loss:.4f}, Val Loss: {epoch_test_loss:.4f}")
```

```
Epoch [1/20], Train Loss: 623890157.9806, Val Loss: 412879530.4854
Epoch [2/20], Train Loss: 623940512.3107, Val Loss: 413948305.0097
Epoch [3/20], Train Loss: 623996497.3981, Val Loss: 411019509.7476
Epoch [4/20], Train Loss: 623989825.2427, Val Loss: 411370753.6311
Epoch [5/20], Train Loss: 623973612.1165, Val Loss: 411912126.9903
Epoch [6/20], Train Loss: 623946591.3786, Val Loss: 417266597.1262
Epoch [7/20], Train Loss: 624062196.1942, Val Loss: 415367427.4951
Epoch [8/20], Train Loss: 623925669.9029, Val Loss: 411984919.7670
Epoch [9/20], Train Loss: 623885091.1068, Val Loss: 413155913.0097
Epoch [10/20], Train Loss: 623948272.1553, Val Loss: 412795645.3592
Epoch [11/20], Train Loss: 623910954.5631, Val Loss: 417790534.6796
Epoch [12/20], Train Loss: 623947283.2621, Val Loss: 415273156.9709
Epoch [13/20], Train Loss: 623845003.1845, Val Loss: 412736085.7476
Epoch [14/20], Train Loss: 623942563.7282, Val Loss: 415044469.1262
Epoch [15/20], Train Loss: 623968795.3398, Val Loss: 409847003.7282
Epoch [16/20], Train Loss: 623938204.2718, Val Loss: 412472381.0485
Epoch [17/20], Train Loss: 624067172.3495, Val Loss: 414743242.4854
Epoch [18/20], Train Loss: 623938354.0194, Val Loss: 412260138.5631
Epoch [19/20], Train Loss: 624022908.2718, Val Loss: 415447279.8447
Epoch [20/20], Train Loss: 623926348.4272, Val Loss: 412570293.3592
```

```
model.eval()
y_pred_nn = []
with torch.no_grad():
    for batch_X, _ in test_loader:
        outputs = model(batch_X)
        y_pred_nn.extend(outputs.squeeze().tolist())

# Оценка качества
rmse_nn = mean_squared_error(y_test, y_pred_nn)
r2_nn = r2_score(y_test, y_pred_nn)

print("Нейронная сеть (PyTorch):")
```

```
print(f"RMSE: {rmse_nn}")
print(f"R²: {r2_nn}")
```

```
Нейронная сеть (PyTorch):
RMSE: 412659283.2285213
R²: -0.026538055121262172
```

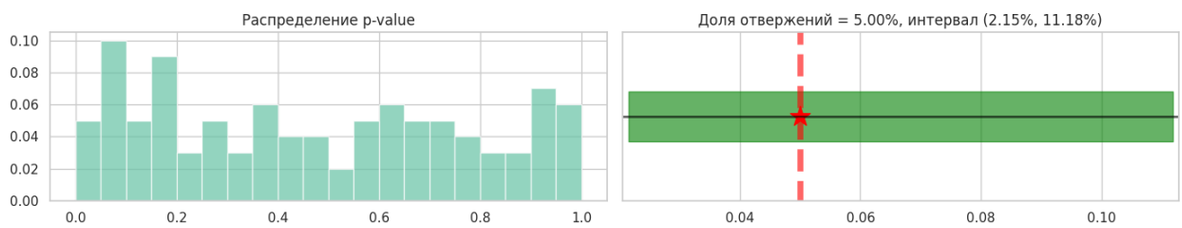
```
test_data['nn_model_pred'] = y_pred_nn
```

Cuped по предсказаниям моделей:

```
tests = [
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['linear_pred']),
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['catboost_pred']),
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['nn_model_pred']),
]
names = [
    'Cuped по линейной модели',
    'Cuped по гр. бустингу',
    'Cuped по нейросети',
        ]

for test, name in zip(tests, names):
    print(name)
    run_experiments(
        test=test,
        generate_samples=generate_subarray,
        sample_size = test_control_size,
        target_name=target_column,
        data=test_data,
        n_iter=n_iter,
        alpha=alpha,
        show_pvals=True,
        draw=True,
        n_jobs=n_jobs,
    );
```

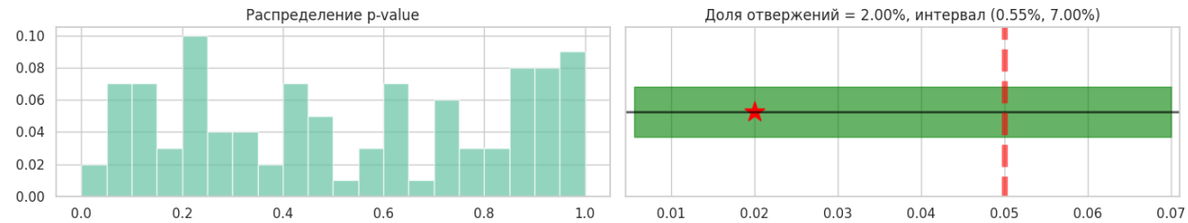
```
Cuped по линейной модели
```

{"model_id":"51ad17cada1c47b5b47b9e9a4700487b","version_major":2,"version_minor":0}



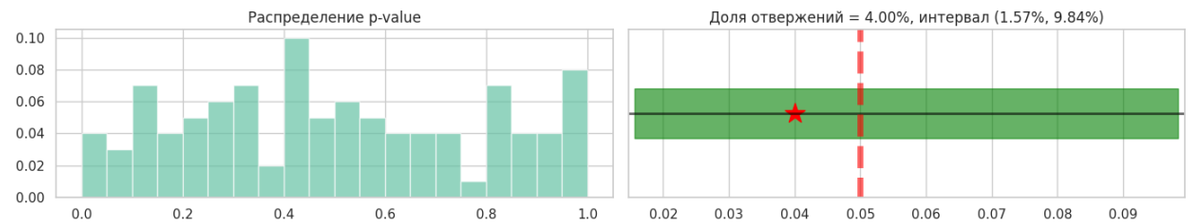

```
Cuped по гр. бустингу
```

{"model_id":"495dd6aeceed4ae7bb38fd27fb12b2bd","version_major":2,"version_minor":0}



Cuped по нейросети

{"model_id":"4f2fcbe69d454aadbfe5926b6d4d5a50","version_major":2,"version_minor":0}

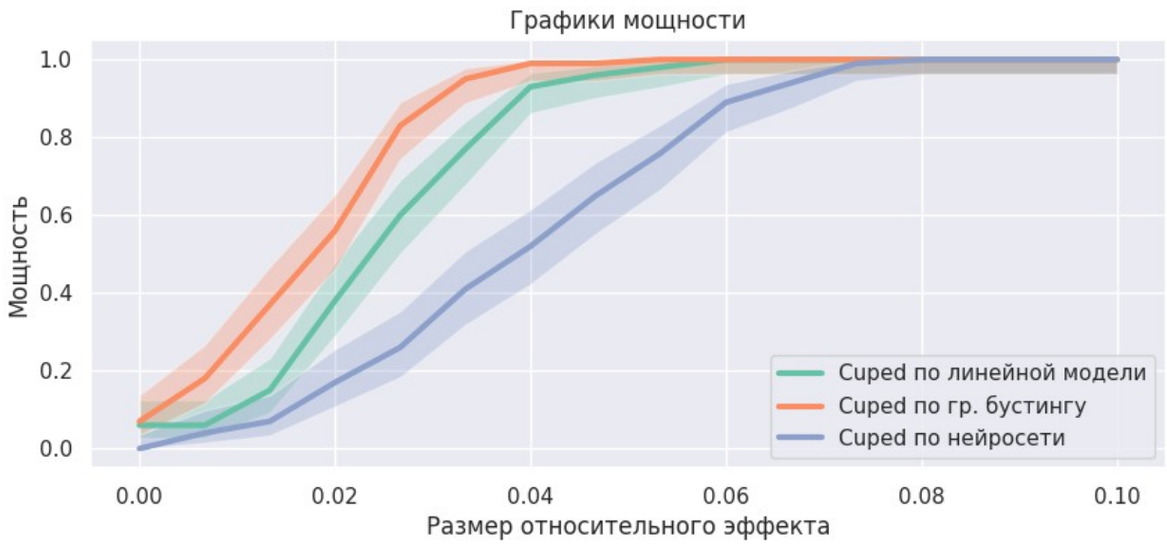



```
tests = [
    lambda x, y: regression_ttest(x, y, sample_name = target_column, cuped=True, covariate_names=['linear_pred']),
    lambda x, y: regression_ttest(x, y, sample_name = target_column, cuped=True, covariate_names=['catboost_pred']),
    lambda x, y: regression_ttest(x, y, sample_name = target_column, cuped=True, covariate_names=['nn_model_pred']),
]
names = [
    'Cuped по линейной модели',
    'Cuped по гр. бустингу',
    'Cuped по нейросети',
        ]

for i, (test, name) in enumerate(zip(tests, names)):
    powers, left_bounds, right_bounds = estimate_power(
        test=test,
        generate_samples=generate_subarray,
        target_name=target_column,
        sample_size = test_control_size,
        effects_list=effects_list,
        data=test_data,
        n_iter=n_iter,
        alpha=alpha,
        n_jobs=n_jobs,
    )
    draw_power(
        powers,
        left_bounds,
        right_bounds,
        effects_list=effects_list,
        label=name,
        new_figure=(i == 0),
    )
```

{"model_id":"84725f90ccd7455999cfdc733e1e9536","version_major":2,"version_minor":0}

{"model_id":"a0e8636cb5bb45758d7d9f739dbe87a6","version_major":2,"version_minor":0}

{"model_id":"23b0537cce3f4ebfa8df718ad409d8f7","version_major":2,"version_minor":0}



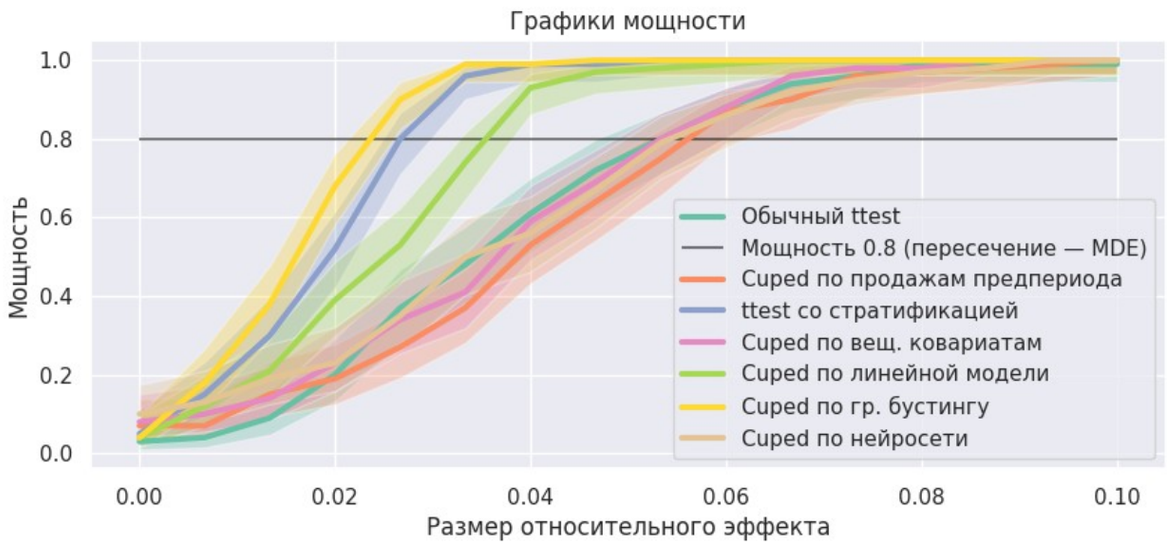
Нарисуем общий график мощности:

```
def draw_power(
    powers,
    left_powers,
    right_powers,
    effects_list=np.linspace(0, 0.1, 11),
    label=None,
    title="Графики мощности",
    new_figure=False,
):
    """
    Построение и отрисовка графика мощности критерия.

    Аргументы:
    effects -- сетка эффектов
    real_alpha -- оценка реальной мощности
    left_alpha, right_alpha -- границы соотв. дов. интервала
    title -- заголовок графика
    """

    if new_figure:
        plt.figure(figsize=(10, 4))

    plt.plot(effects_list, powers, label=label, lw=3)
    plt.fill_between(effects_list, left_powers, right_powers,
alpha=0.3)

    if new_figure:
        plt.hlines(
            0.8,
            effects_list[0],
            effects_list[-1],
            color="black",
            alpha=0.5,
            label="Мощность 0.8 (пересечение — MDE)",
        )
    plt.legend()
    plt.xlabel("Размер относительного эффекта")
```

```
    plt.ylabel("Мощность")
    plt.title(title)

test_control_size = 10000 * 2
fraq_test = 0.5

target_column = "Weekly_Sales"
mean = np.mean(data[target_column])
effects_list = np.linspace(0, 0.1, 16)

n_iter = 100
alpha = 0.05
n_jobs = 4

tests = [
    lambda x, y: ttest_rel(x[target_column], y[target_column]),
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['Mean_Weekly_Sales',
'Mean_Weekly_Sales_is_missing']),
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
stratified=True, strat_names=['Store', 'Dept','IsHoliday_x',
'Type'],),
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['Temperature', 'Unemployment',
'Fuel_Price', 'CPI', 'Size']),
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['linear_pred']),
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['catboost_pred']),
    lambda x, y: regression_ttest(x, y, sample_name = target_column,
cuped=True, covariate_names=['nn_model_pred']),
    ]
names = [
    'Обычный ttest',
    'Cuped по продажам предпериода',
    'ttest со стратификацией',
    'Cuped по вещ. ковариатам',
    'Cuped по линейной модели',
    'Cuped по гр. бустингу',
    'Cuped по нейросети',
        ]

for i, (test, name) in enumerate(zip(tests, names)):
    powers, left_bounds, right_bounds = estimate_power(
        test=test,
        generate_samples=generate_subarray,
        target_name=target_column,
        sample_size = test_control_size,
        effects_list=effects_list,
        data=test_data,
        n_iter=n_iter,
        alpha=alpha,
        n_jobs=n_jobs,
    )
    draw_power(
        powers,
        left_bounds,
        right_bounds,
        effects_list=effects_list,
        label=name,
```

```
        new_figure=(i == 0),
    )
```


```
{"model_id":"ed29188edc2b4e2ab4e5f01f1dd8feff","version_major":2,"vers
ion_minor":0}

{"model_id":"2bed9b0069fc45acb8fcdd99a2187ff8","version_major":2,"vers
ion_minor":0}

{"model_id":"828e0d7b010a4023a83bccff1dc0bf0a","version_major":2,"vers
ion_minor":0}

{"model_id":"a2c9cb23ff6d4dd6a5cd1497c3bc4adf","version_major":2,"vers
ion_minor":0}

{"model_id":"d547a6d63eb5451bb78383fa97b825a5","version_major":2,"vers
ion_minor":0}

{"model_id":"0a4b85ff805e4865bf1c600a3f2e037d","version_major":2,"vers
ion_minor":0}

{"model_id":"679113448f1c4bd78013051725478763","version_major":2,"vers
ion_minor":0}
```

Выводы:

Все реализованные критерии прошли валидацию, с последующим построением графика мощности:

- Лучше всех себя показал Cuped при использовании предсказаний градиентного бустинга как ковариат, в целом ожидаемо, так как бустинги всегда себя хорошо показывают на табличных данных
- Мощность критерия обычного ttesta почти совпадает с мощностями при использовании ковариат и стратификации на общем графике, однако в графиках выше при использование ковариат и стратификации показывает результат лучше
- При использовании нейросети получился не очень хороший результат - неожиданно он хуже линейной модели
- Cuped по продажам предпериода хоть и увеличил мощность, но не так круто как модели с Cuped-ом по предсказаниям градиентного бустинга, линейно модели и стратификации
- Стратификация существенно увеличила мощность критерия - что и ожидаемо, так как мы рассматриваем каждый товар и магазин как страты