Статистика, прикладной поток

Практическое задание 6

В данном задании вы исследуете некоторые свойства непараметрических методов, а также проверки статистических гипотез.

Правила:

- Дедлайн **2 декабря 23:59**. После дедлайна работы не принимаются кроме случаев наличия уважительной причины.
- Выполненную работу нужно отправить на почту `mipt.stats@yandex.ru`, указав тему письма `"[applied] Фамилия Имя - задание 6"`. Квадратные скобки обязательны. Если письмо дошло, придет ответ от автоответчика.
- Прислать нужно ноутбук и его pdf-версию (без архивов). Названия файлов должны быть такими: `6.N.ipynb` и `6.N.pdf`, где `N` - ваш номер из таблицы с оценками.
- Решения, размещенные на каких-либо интернет-ресурсах не принимаются. Кроме того, публикация решения в открытом доступе может быть приравнена к предоставлении возможности списать.
- Для выполнения задания используйте этот ноутбук в качестве основы, ничего не удаляя из него.
- Никакой код из данного задания при проверке запускаться не будет.

Баллы за задание:

- Задача 1 - 2 балла **O3**
- Задача 2 - 10 баллов **O2**
- Задача 3 - 5 баллов **O3**
- Задача 4 - 5 баллов **O3**
- Задача 5 - 5 баллов **O3**
- Задача 6 - 12 баллов **O3**
- Задача 7 - 20 баллов **O3**
- Задача 8 - 15 баллов **O3**

In [2]:
```
import numpy as np
import scipy.stats as sps
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import warnings
from statsmodels.nonparametric.kde import kernel_switch, KDEUnivariate
from statsmodels.distributions.empirical_distribution import ECDF

sns.set()
warnings.filterwarnings("ignore")

%matplotlib inline
```

Непараметрический подход

Задача 1.

Напишите определение ядра, используемого для построения ядерных оценок плотности.

<...>

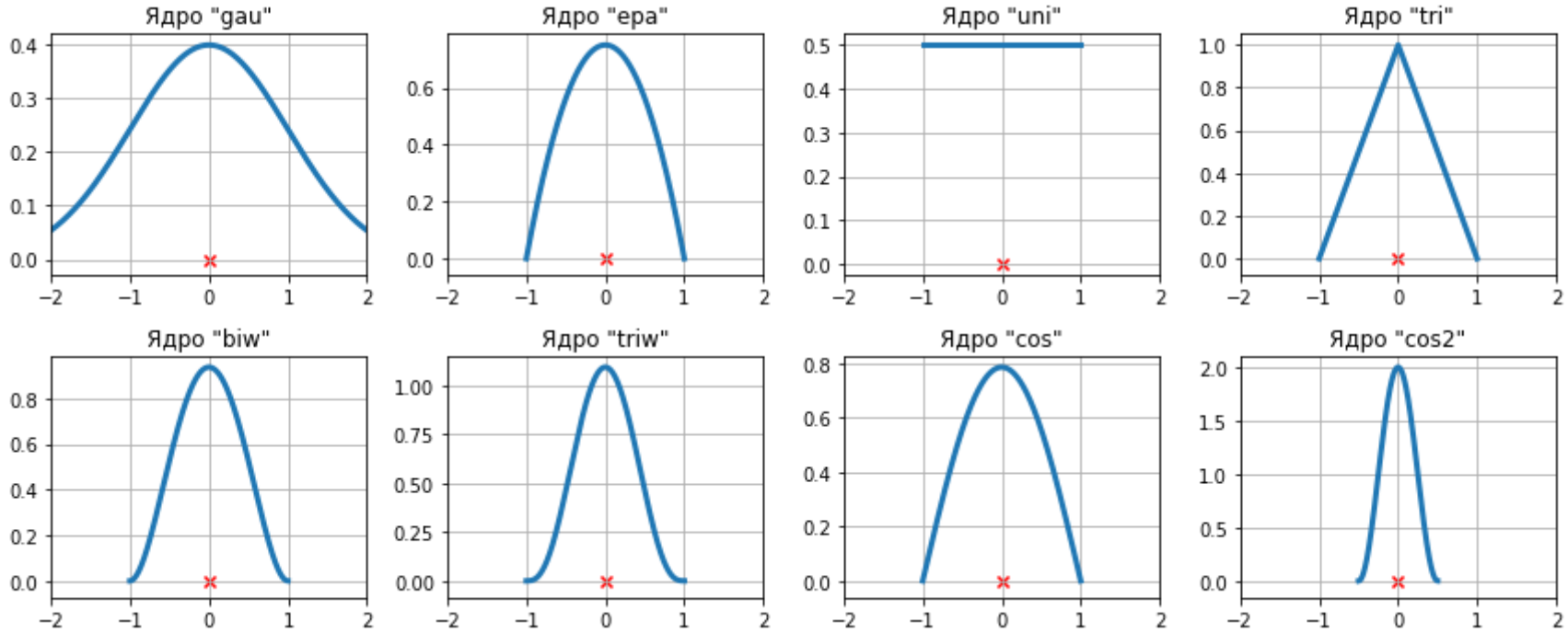
Прежде чем начать работу с ядерными оценками плотности, изучим виды ядер. В библиотеке `statsmodels` реализованы следующие ядра:

In [3]:
```
list(kernel_switch.keys())
```

Out[3]:
```
['gau', 'epa', 'uni', 'tri', 'biw', 'triw', 'cos', 'cos2']
```

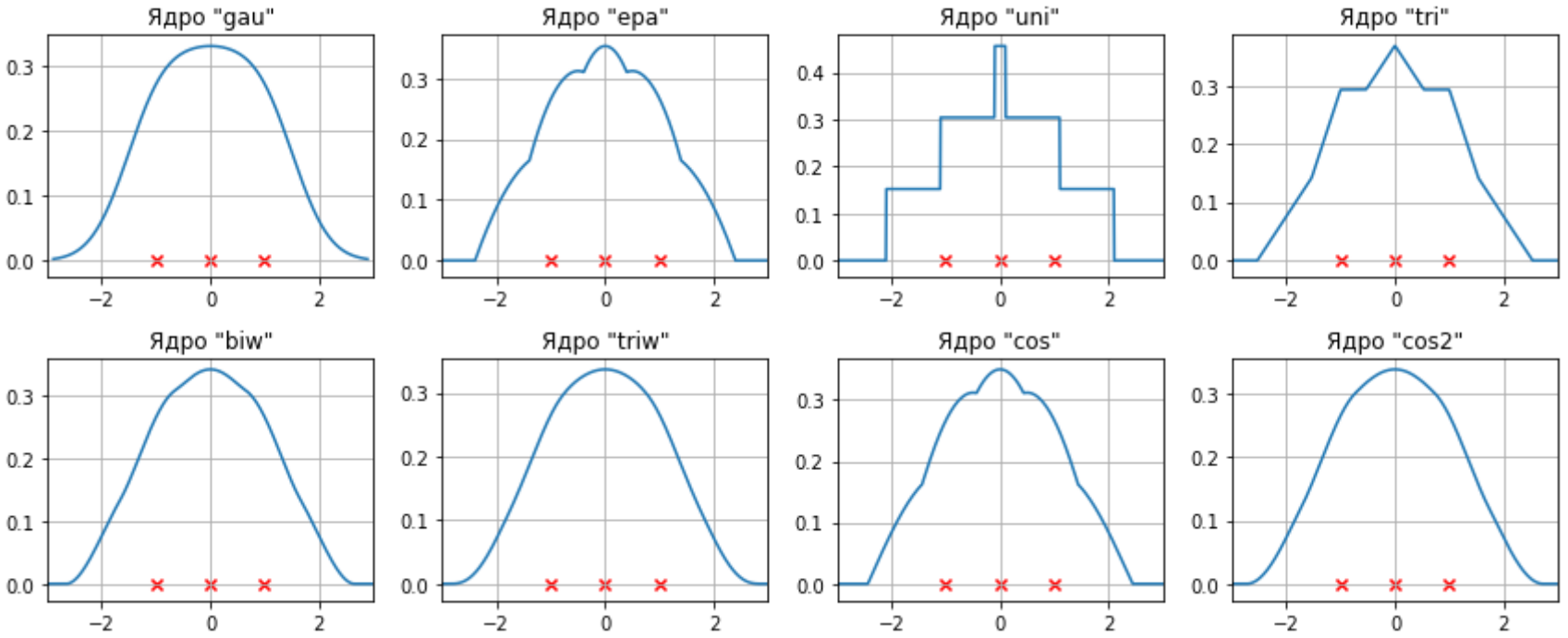
Нарисуем эти ядра. Запустите код в ячейке.

In [4]:
```
fig = plt.figure(figsize=(12, 5))

for i, (ker_name, ker_class) in enumerate(kernel_switch.items()):

    kernel = ker_class()  # ядро
    domain = kernel.domain or [-2, 2]  # носитель
    x_vals = np.linspace(*domain, 2**10)
    y_vals = kernel(x_vals)

    ax = fig.add_subplot(2, 4, i + 1)
    ax.set_title('Ядро "{}"'.format(ker_name))
    ax.plot(x_vals, y_vals, lw=3, label='{}'.format(ker_name))
    ax.scatter([0], [0], marker='x', color='red')
    plt.grid(True, zorder=-5)
    ax.set_xlim((-2, 2))

plt.tight_layout()
```



Посморим, как будет выглядеть ядерная оценка плотности в зависимости от типа ядра, взяв в качестве реализации выборки набор точек [-1, 0, 1].
Запустите код в ячейке.

In [159]:
```
data = np.linspace(-1, 1, 3)  # выборка
kde = KDEUnivariate(data)  # объект, выполняющий построение оценки

fig = plt.figure(figsize=(12, 5))

for i, kernel in enumerate(kernel_switch.keys()):

    ax = fig.add_subplot(2, 4, i + 1)
    ax.set_title('Ядро "{}"'.format(kernel))

    # построение ядерной оценки плотности с заданным ядром
    kde.fit(kernel=kernel, fft=False, gridsize=2**10)
    # отрисовка полученной оценки
    ax.plot(kde.support, kde.density)
    # отрисовка выборки
    ax.scatter(data, np.zeros_like(data), marker='x', color='red')
    plt.grid(True, zorder=-5)
    ax.set_xlim([-3, 3])

plt.tight_layout()
```



Какие вы можете назвать преимущества и недостатки для первых трех ядер как с точки зрения самой оценки плотности, так и с вычислительной точки зрения?

Думаю, что самым сложным для вычисления ядром будет гаусово, потому что одна точка влияет на плотность на всей прямой, а не только на плотность на ограниченном отрезке. Хотя гаусово можно считать через fft.

Недостаток равномерного ядра в том, что функция плотности получается разрывной. Также епанечниково и равномерное ядра дают негладкие распределения. Поэтому если что-то известно о непрерывности или гладкости исходного распределния, то не стоит использовать для приближения ядра, которые не обладают этими свойствами.

Другие примеры работы с ядерными оценками плотности можно посмотреть по ссылке
http://www.statsmodels.org/dev/examples/notebooks/generated/kernel_density.html
(http://www.statsmodels.org/dev/examples/notebooks/generated/kernel_density.html)

Задача 2.

1. Сгенерируйте выборку $X_1, \ldots, X_{10000}$ из стандартного нормального распределения. Для каждого $n \leqslant 10000$ постройте эмпирическую функцию распределения $\hat{F}_n$ и посчитайте **точное** значение статистики

$$D_n = \sup_{x \in \mathrm{R}} \left| \hat{F}_n(x) - F(x) \right|.$$

Постройте график зависимости статистики D_n от n. Верно ли, что $D_n \to 0$ и в каком смысле?

Для выполнения задания можно использовать следующую функцию:

In [ ]:
```
from statsmodels.distributions.empirical_distribution import ECDF
```

Заметим, что супремум $\left| \hat{F}_n(x) - F(x) \right|$ будет достигаться на точке из выборки

In [5]:
```
def find_D(sample):
    #print(sample)
    n = sample.size
    left_max = np.abs(ECDF(sample, side='left')(sample) - sample).max()
    right_max = np.abs(ECDF(sample, side='right')(sample) - sample).max()
    return max(left_max, right_max)

find_D = np.vectorize(find_D, signature="(n)->()")
```

In [94]:
```
sample = sps.uniform.rvs(size=10000)
D = find_D([sample[:i] for i in range(1, n)])

plt.plot(np.arange(2, n+1), D)
plt.grid(True)
plt.ylim(0,0.1)
plt.xlabel('$n$')
plt.ylabel('$D_n$')
plt.title('график зависимости статистики $D_n$ от $n$');
```

Вывод:

По теореме Гливенко-Кантелли $D_n \to 0$ почти наверно. По графику, можно увидеть, что действительно сходится, но медленно.

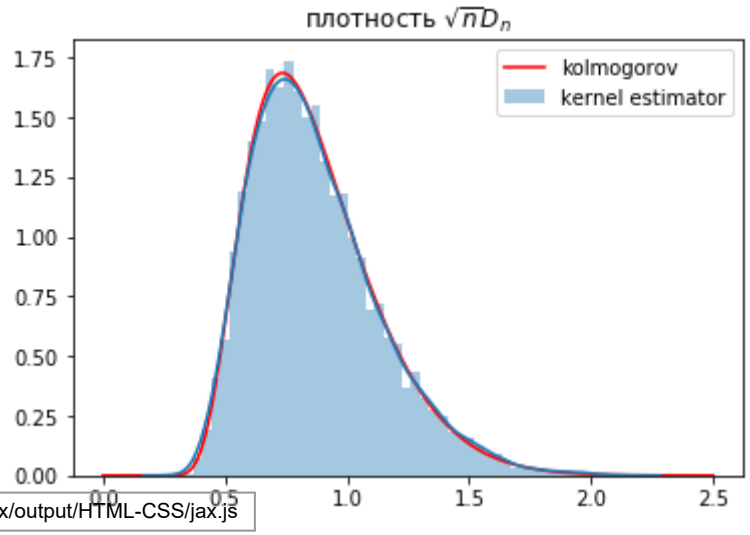
2. Для $n = 10000$ и $k = 10000$ (значение k можно взять больше) выполните следующее:

Вычислите $D_n^1, \ldots, D_n^k$ для независимых выборок $(X_1^1, \ldots, X_n^1)$, ... ,$(X_1^k, \ldots, X_n^k)$ из стандартного нормального распределения. Постройте график гистограммы значений $\sqrt{n}D_n^1, \ldots, \sqrt{n}D_n^k$ и ядерной оценки плотности распределения этих величин.

Для выполнения задания можно воспользоваться как функцией `seaborn.distplot` (желательно), так и реализацией в `statsmodels`.

In [21]:
```
n, k = 10000, 10000
sample = sps.uniform.rvs(size=(k, n))
D = n**(1/2) * find_D(sample)
grid = np.linspace(0,2.5,1000)
plt.plot(grid, sps.kstwobign.pdf(grid), label='kolmogorov', color='red')
sns.distplot(D, label='kernel estimator');
plt.legend()
plt.title('плотность $\\sqrt{n}D_n$');
```

```
C:\Users\1\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated,
and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



Вывод:

$\sqrt{n}D_n$ сходится к распределению Колмогорова, а значит проверен критерий Колмогорова-Смирнова .

Задача 3.

Для каждого распределения постройте график эмпирической функции распределения, гистограмму и график ядерной оценки плотности.

Вам выдется почти готовый код для выполнения задания с некоторыми пропусками. Код предполагает использование реализации ядерных оценок плотности из `statsmodels`. При желании вы можете написать аналогичный код, используя реализацию в `seaborn`.

In [123]:
```
def draw_ecdf(sample, grid, cdf=None):
    """По сетке grid строит графики эмпирической функции распределения
    и истинной (если она задана) для всей выборки и для 1/10 ее части.
    """
    plt.figure(figsize=(16, 3))
    for i, size in enumerate([len(sample) // 10, len(sample)]):
        plt.subplot(1, 2, i + 1)

        plt.scatter(sample[:size], np.zeros(size), #<первые size точек из sample с нулевой y-координатой>,
                    alpha=0.4, label='sample')

        if cdf is not None:
            plt.plot(grid,
                     cdf(grid),
                     color='green', alpha=0.3, lw=2, label='true cdf')

        plt.plot(grid,
                 ECDF(sample[:size])(grid),
                 color='red', label='ecdf')

        plt.legend()
        plt.grid(ls=':')
        plt.title('sample size = {}'.format(size))
    plt.show()
```

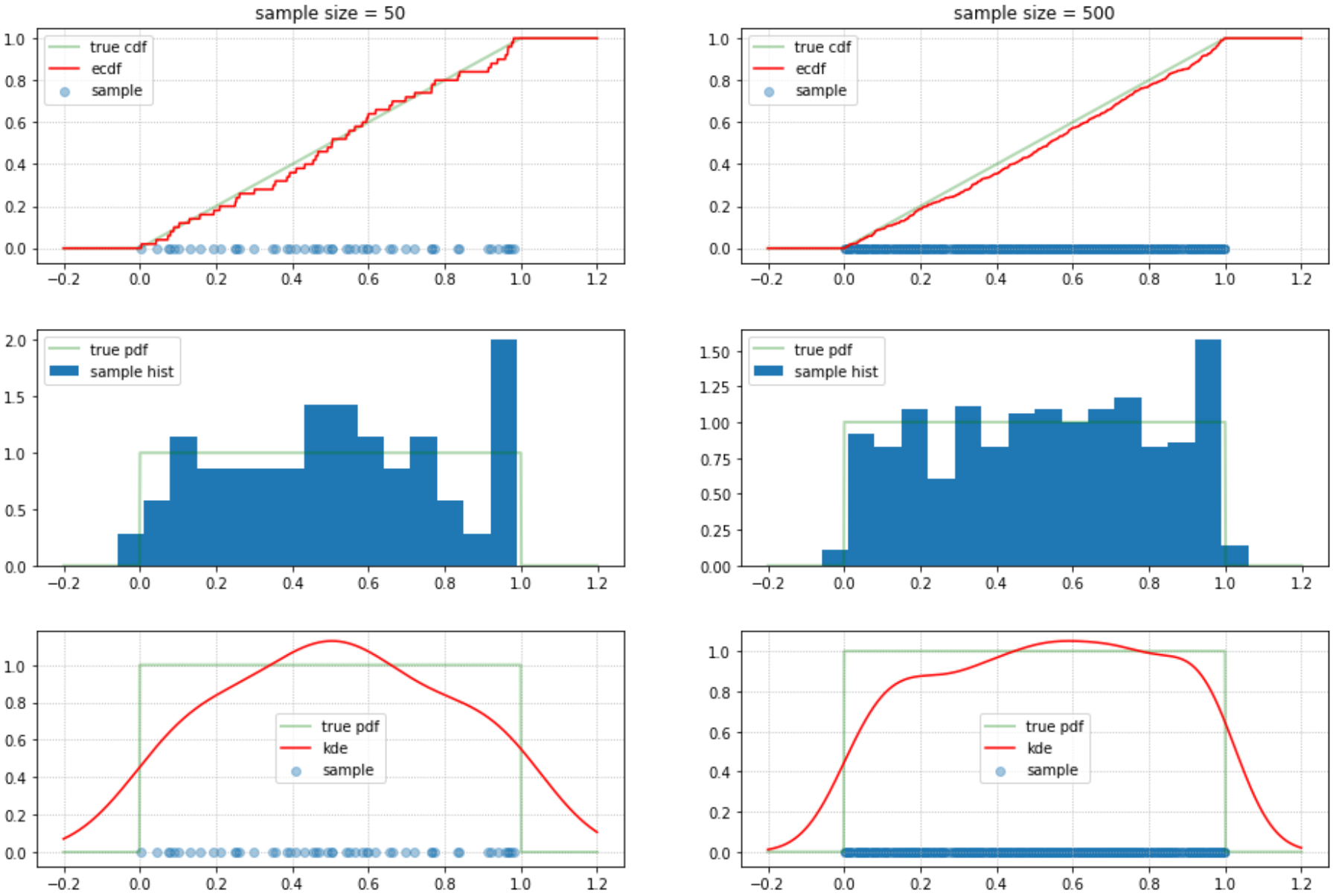
In [124]:
```
def draw_hist(sample, grid, pdf=None):
    """Строит гистограмму и, по сетке grid, график истинной плотности
    (если она задана) для всей выборки и для 1/10 ее части.
    """
    plt.figure(figsize=(16, 3))
    for i, size in enumerate([len(sample) // 10, len(sample)]):
        plt.subplot(1, 2, i + 1)

        plt.hist(sample[:size],
                 bins=20,
                 range=(grid.min(), grid.max()),
                 density=True, label='sample hist')

        if pdf is not None:
            plt.plot(grid,
                     pdf(grid),
                     color='green', alpha=0.3, lw=2, label='true pdf')

        plt.legend()
    plt.show()
```

In [151]:
```
def draw_pdf(sample, grid, pdf=None):
    """По сетке grid строит графики ядерной оценки плотности
    и истинной плотности (если она задана) для всей выборки
    и для 1/10 ее части.
    """
    plt.figure(figsize=(16, 3))
    for i, size in enumerate([len(sample) // 10, len(sample)]):
        plt.subplot(1, 2, i + 1)

        plt.scatter(sample[:size], np.zeros(size),
                    alpha=0.4, label='sample')

        if pdf is not None:
            plt.plot(grid,
                     pdf(grid), color='green',
                     alpha=0.3, lw=2, label='true pdf')

        density = KDEUnivariate(sample[:size])
        density.fit()
        plt.plot(grid, density.evaluate(grid),
                 #<Значение ядерной оценки плотности
                 # (по первым size точек из sample)
                 # в точках grid (используйте класс KDEUnivariate,
                 # в частности, его метод evaluate)>,
                 color='red', label='kde')

        plt.legend()
        plt.grid(ls=':')
    plt.show()
```

Теперь примените реализованные выше функции к выборкам размера 500 для следующих распределений:

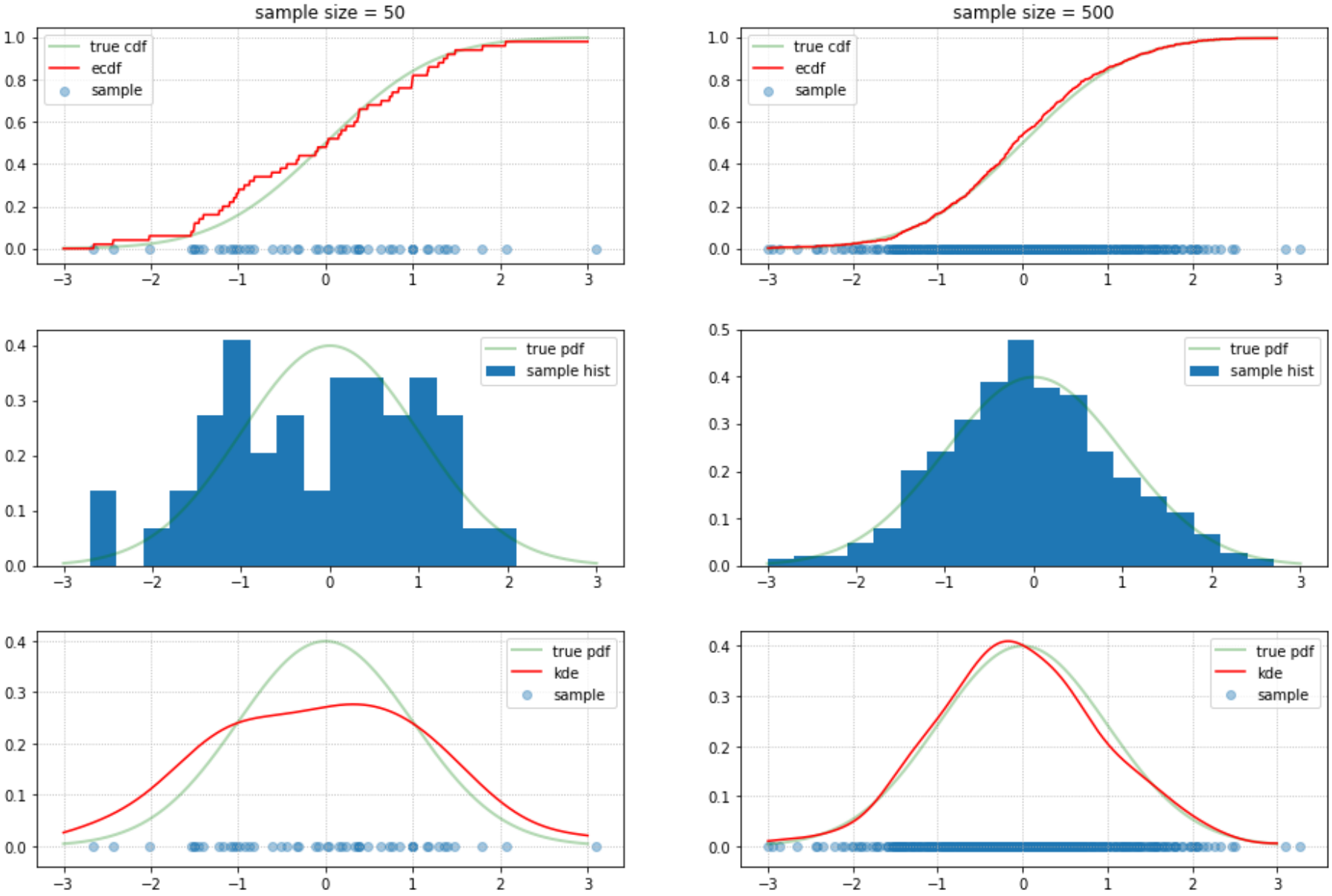
1. Равномерное распределение $U[0, 1]$. Графики (ф.р., плотностей) строить на интервале $(-0.2, 1.2)$.

In [152]:
```
def task(distribution, xlim):
    sample = distribution.rvs(size=500)
    grid = np.linspace(xlim[0], xlim[1], 1000)

    draw_ecdf(sample, grid, distribution.cdf)
    draw_hist(sample, grid, distribution.pdf)
    draw_pdf(sample, grid, distribution.pdf)

task(sps.uniform, (-0.2, 1.2))
```



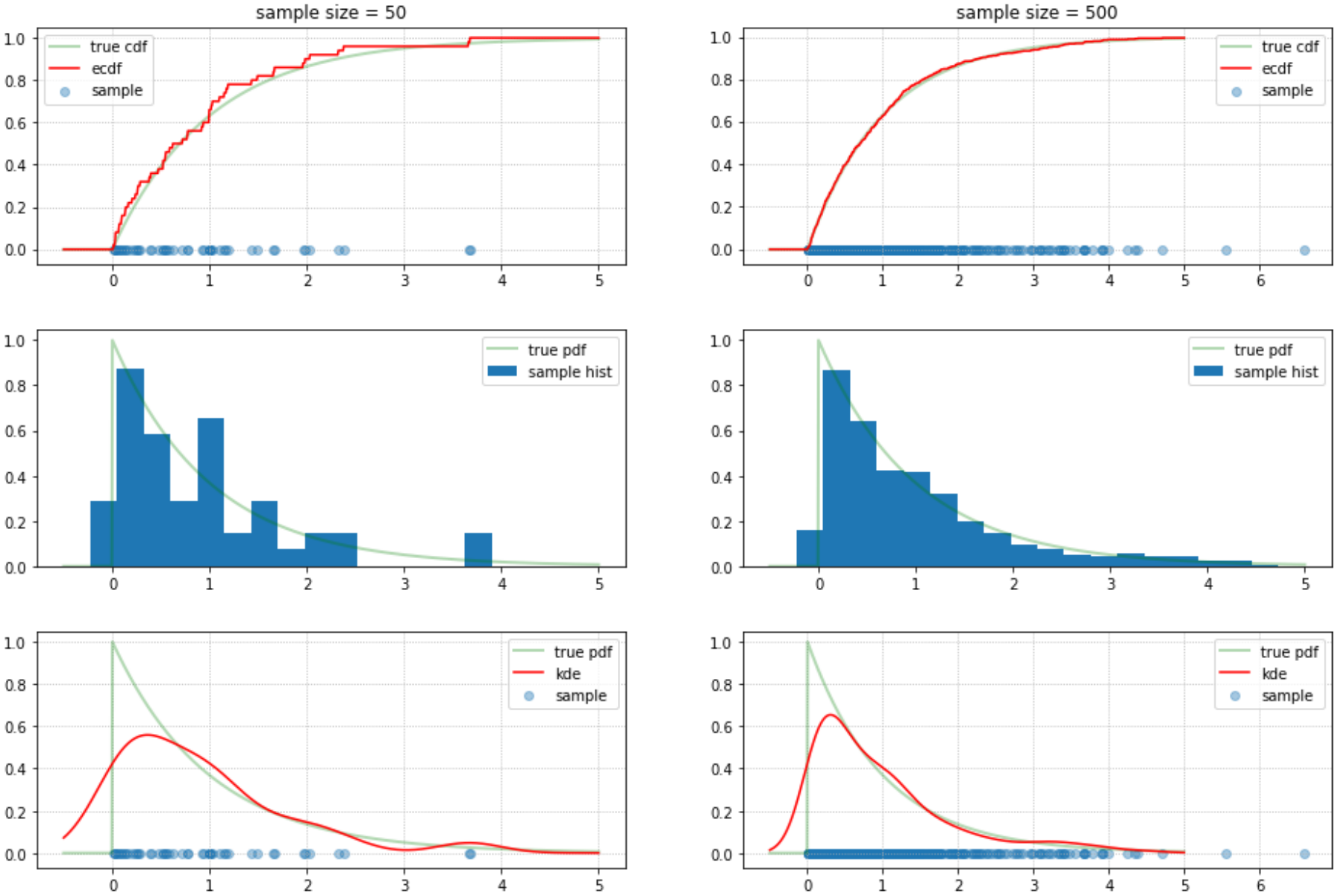
2. Нормальное распределение $N(0, 1)$. Графики строить на интервале $(-3, 3)$.

In [153]:
```
task(sps.norm, (-3,3))
```



3. Экспоненциальное распределение $Exp(1)$. Графики строить на интервале $(-0.5, 5)$.

In [154]:
```
task(sps.expon, (-0.5, 5))
```



Вывод:

Эмпирическая функция распределения хорошо оценивает функцию распределения.

Плотность нормального распределения хорошо приближается ядерной плотностью, чего нельзя сказать о плотности равномерного и экспоненциального, потому что у этих распределений плотность имеет разрыв

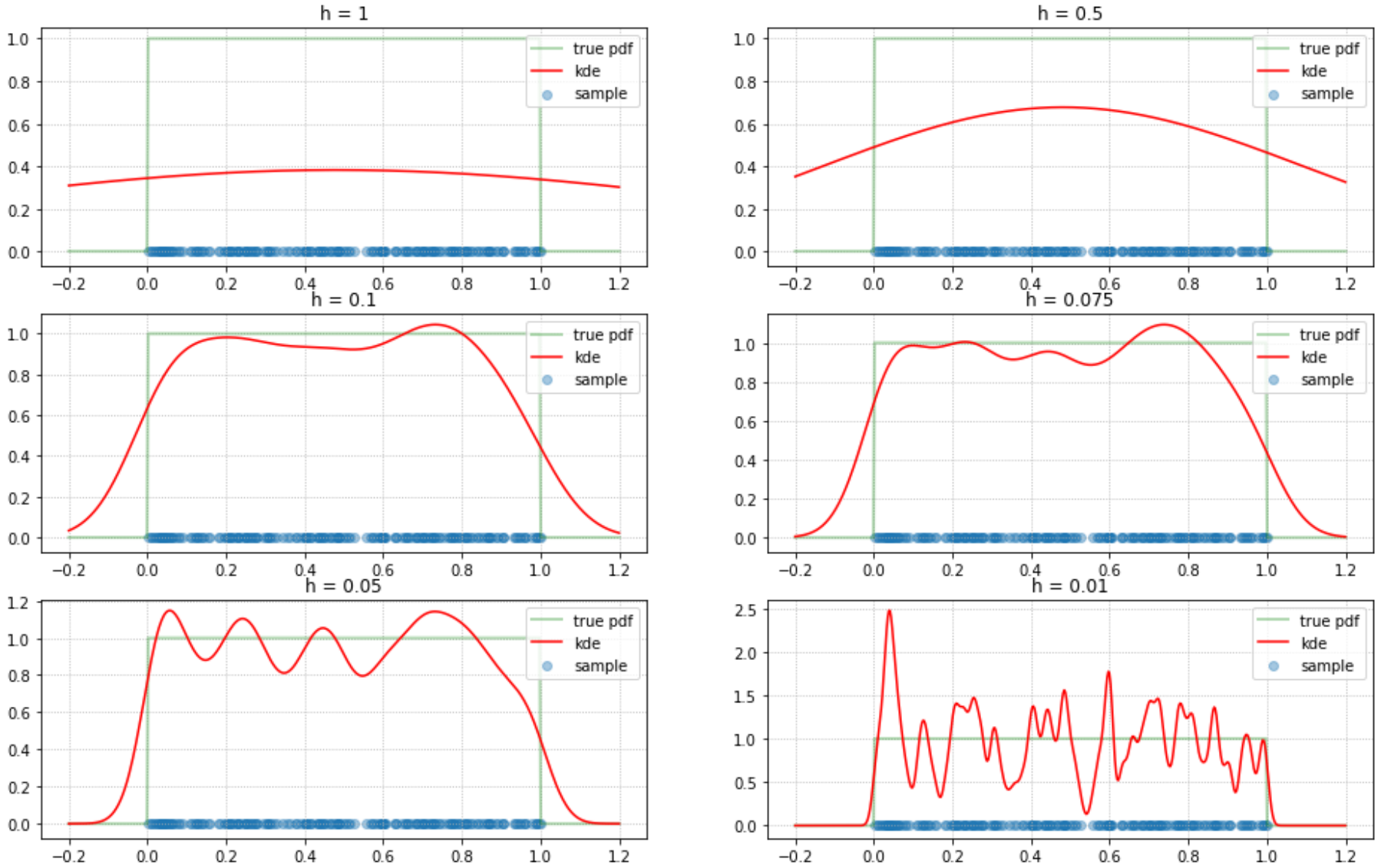
Задача 4.

Исследуйте вид ядерной оценки плотности в зависимости от вида ядра и его ширины.

Для этого сгенерируйте выборку $X_1, \ldots, X_{200}$ из распределения $U[0, 1]$ и постройте серию графиков для различной ширины гауссовского ядра, а затем другую серию графиков для различных типов ядер при фиксированной ширине. На каждом графике на отрезке $[-0.2, 1.2]$ должны быть изображены истинная плотность (полупрозрачным цветом) и ее ядерная оценка, а так же с нулевой y-координатой должны быть нанесены точки выборки. Для экономии места стройте графики в два столбца.

Вам выдется почти готовый код для выполнения задания с некоторыми пропусками. Код предполагает использование реализации ядерных оценок плотности из `statsmodels`. При желании вы можете написать аналогичный код, используя реализацию в `seaborn`.

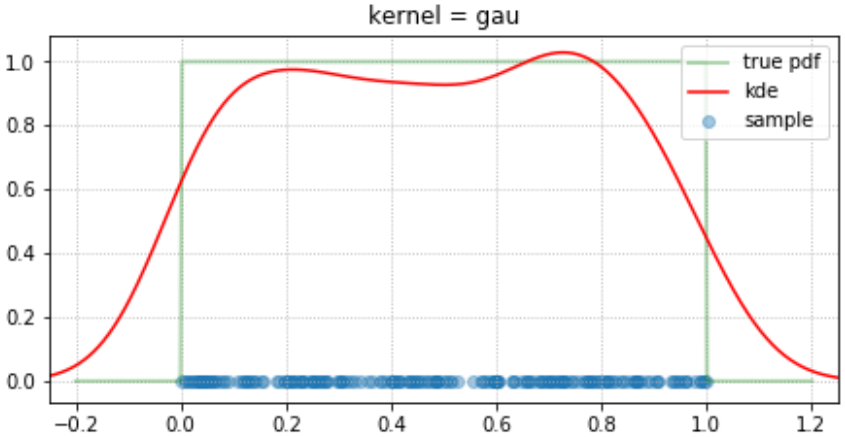
In [28]:
```
size = 200
sample = sps.uniform.rvs(size=size)
grid = np.linspace(-0.2, 1.2, 500)

plt.figure(figsize=(16, 10))
for i, bw in enumerate([1, 0.5, 0.1, 0.075, 0.05, 0.01]):
    plt.subplot(3, 2, i + 1)
    kernel_density = KDEUnivariate(sample)
    kernel_density.fit(bw=bw)
    plt.scatter(sample, np.zeros(size), alpha=0.4, label='sample')
    plt.plot(grid, sps.uniform.pdf(grid), color='green',
             alpha=0.3, lw=2, label='true pdf')
    plt.plot(grid, kernel_density.evaluate(grid),
             color='red', label='kde')
    plt.legend(loc=1)
    plt.grid(ls=':')
    plt.title('h = {}'.format(bw))
plt.show()
```

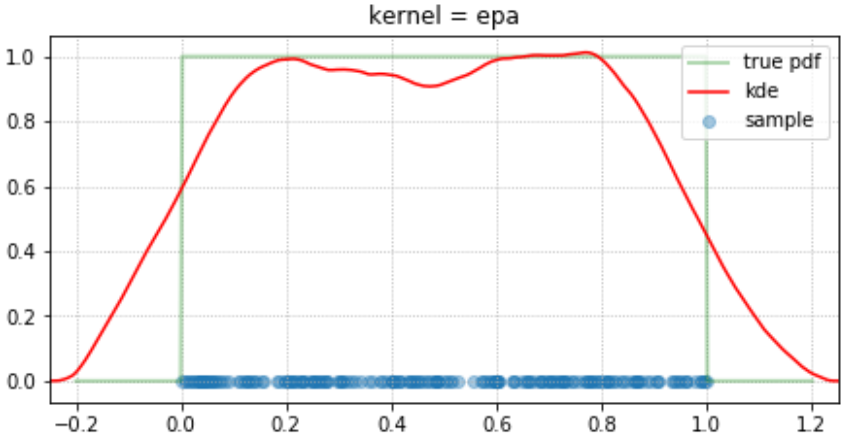


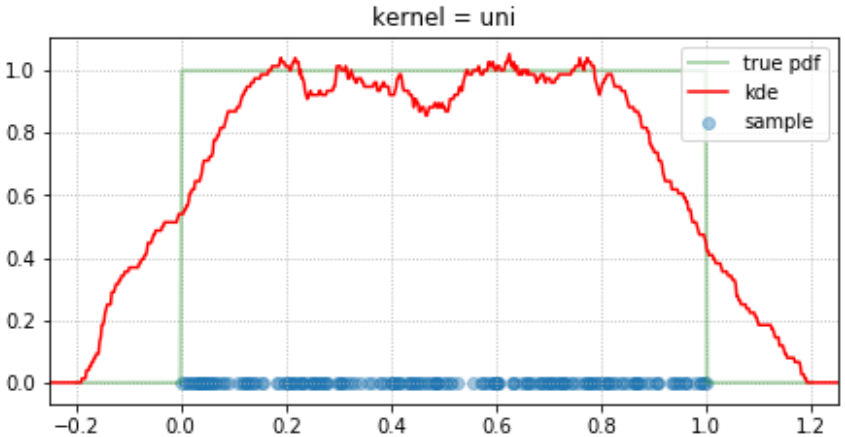
Вывод: При больших значениях ширины ядра $h \geq 0.5$ получается плохое приближение, при малых $h \leq 0.05$ происходит переобучение.

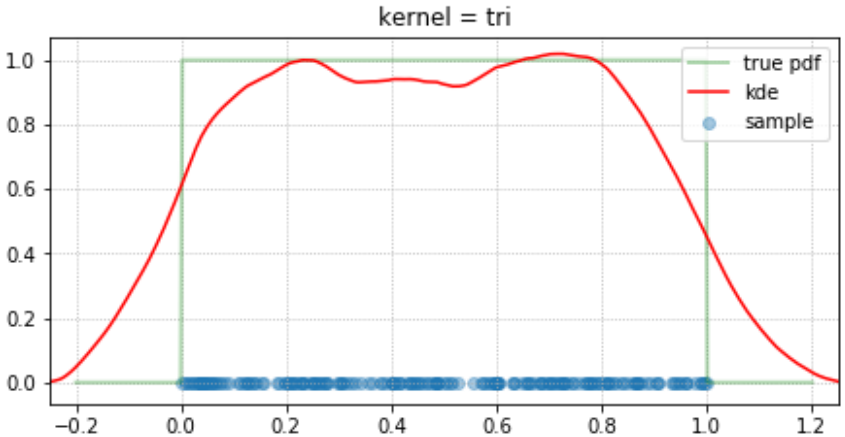
Во втором случае графики постройте аналогичным образом, проведя итерации по типу ядра.

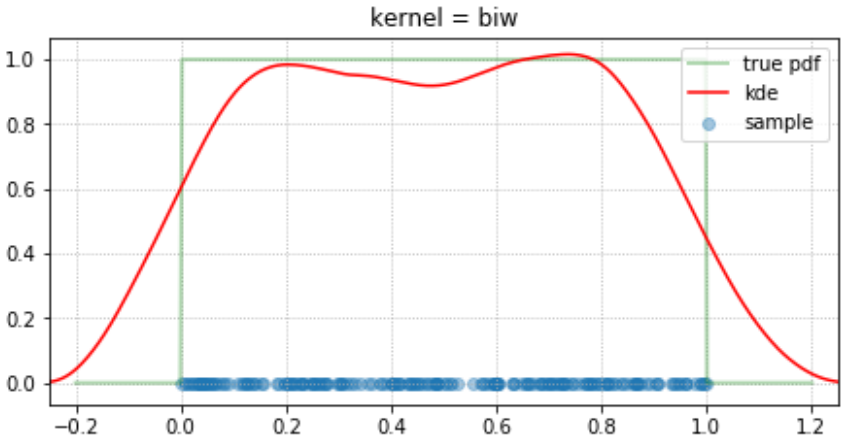
In [34]:
```
plt.figure(figsize=(16, 16))
for i, kernel in enumerate(kernel_switch.keys()):
    plt.subplot(4, 2, i + 1)
    kernel_density = KDEUnivariate(sample)
    kernel_density.fit(kernel=kernel, fft=False, gridsize=2**10)

    plt.scatter(sample, np.zeros(size), alpha=0.4, label='sample')
    plt.plot(grid, sps.uniform.pdf(grid), color='green',
             alpha=0.3, lw=2, label='true pdf')
    plt.plot(kernel_density.support, kernel_density.density, #grid, kernel_density.evaluate(grid),
             color='red', label='kde')
    plt.legend(loc=1)
    plt.grid(ls=':')
    plt.xlim([-0.25, 1.25])
    plt.title('kernel = {}'.format(kernel))
plt.show()
```

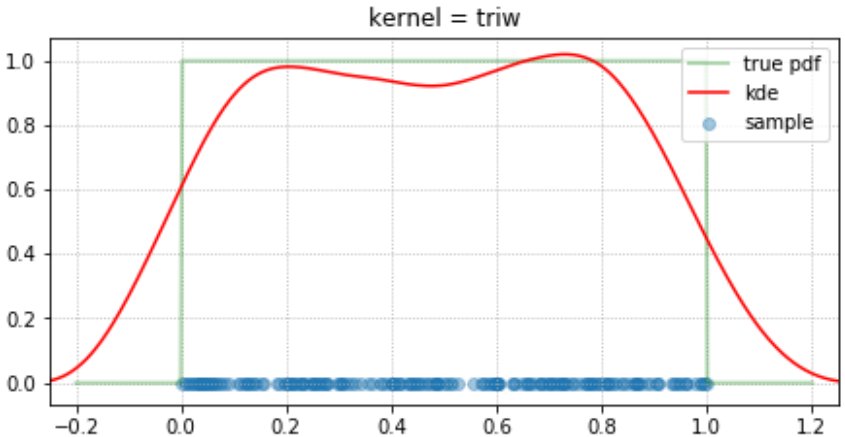


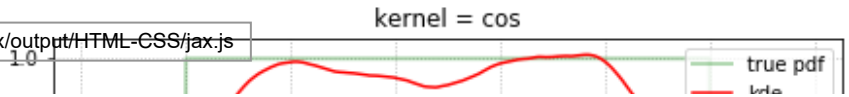


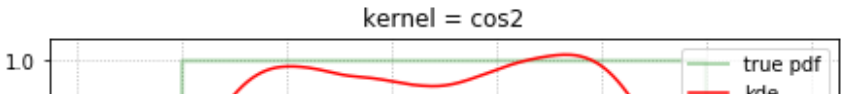


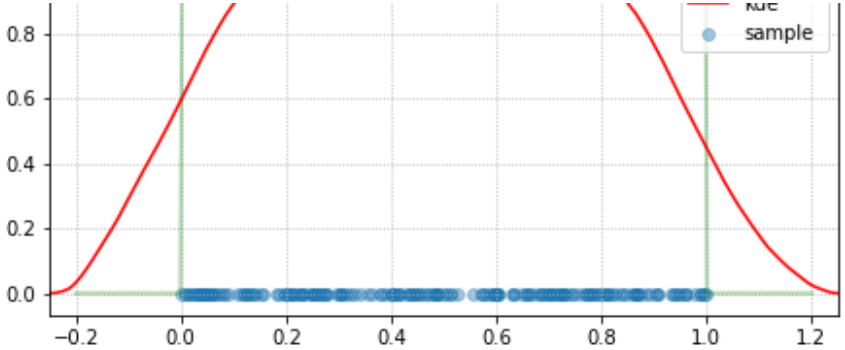












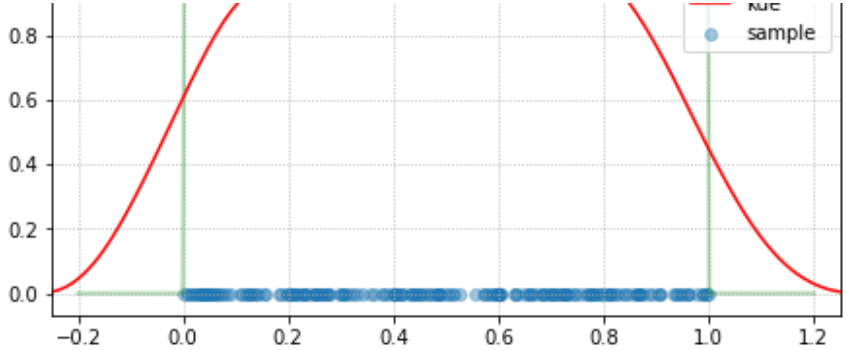


image.png

image.png

Вывод: Все ядра дают примерно одинаковые оценки плотности. Только равномерное ядро выделяется из толпы, потому что разрывно.

Задача 5.

Скачайте данные `wine dataset` (http://archive.ics.uci.edu/ml/datasets/wine) и выберите произвольные 7 столбцов с действительными числами. С помощью `seaborn.PairGrid` постройте таблицу графиков, состоящую из

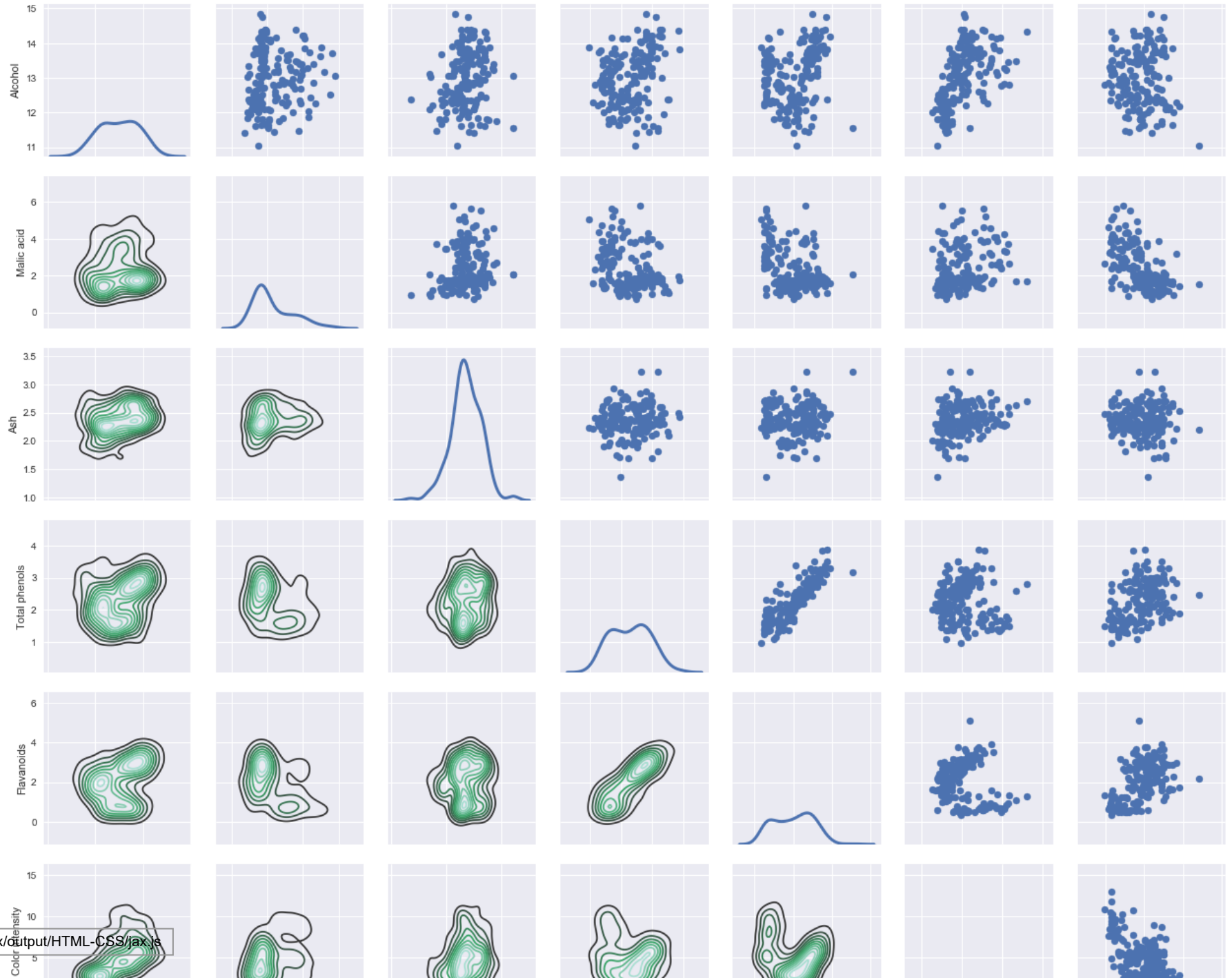
- одномерных ядерных оценок плотности по диагонали;
- двумерных ядерных оценок плотности ниже диагонали;
- scatter-plot выше диагонали (`plt.scatter`).

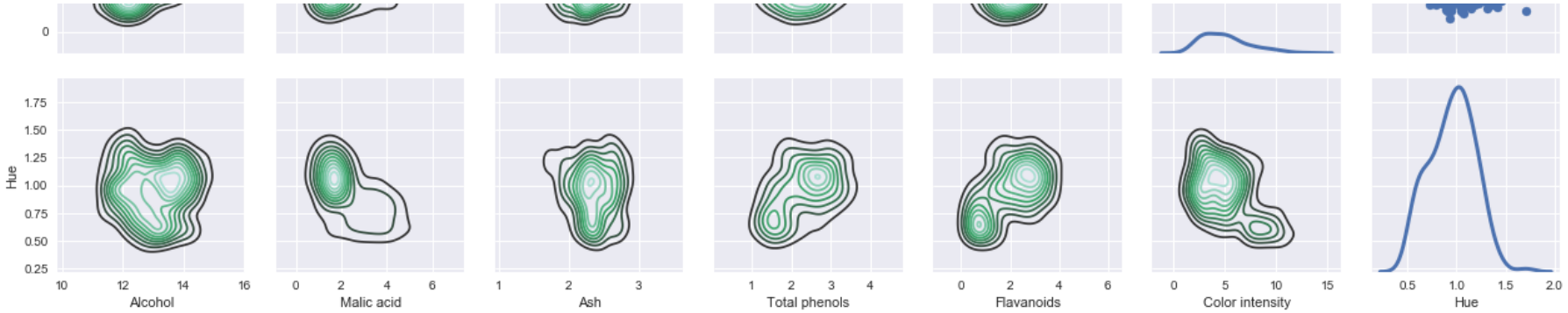
При возникновении затруднений посмотрите обучающий ноутбук по `seaborn`.

In [54]:
```
names= ['Cultivar', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols', 'Flavanoids',
        'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline'
]
sample = pd.read_csv('wine.data', names=names)
sample = sample[['Alcohol', 'Malic acid', 'Ash', 'Total phenols',
                 'Flavanoids', 'Color intensity', 'Hue']]
```

In [55]:
```
g = sns.PairGrid(sample)
g.map_lower(sns.kdeplot)
g.map_upper(plt.scatter)
g.map_diag(sns.kdeplot, lw=3)
g.add_legend();
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```





Вывод:

<...>

Задача 6.

Около месяца назад (24-25 октября) в Краснодарском крае сильные ливни привели наводнению, из-за которого сильно пострадали города Сочи и Туапсе. Вам выданы данные об уровне воды за 2014-2018 год по следующим рекам Краснодарского края:

- Мзымта (Сочи, Адлерский район, Роза Хутор)
- Сочи (Сочи, Центральный район)
- Туапсе (Туапсе)
- Херота (Сочи, Адлерский район)
- Хоста (Сочи, Хостинский район)

В файлах используйте столбец `Уровень воды (по БСВ)`. Это уровень воды по Балтийской системе высот (https://ru.wikipedia.org/wiki/Балтийская_система_высот) — принятой в СССР системе нормальных высот, отсчёт которых ведётся от нуля Кронштадтского футштока. Для каждой реки нарисуйте график уровня воды.

Данные собранны за каждые 10 минут, что достаточно тяжело обрабатывать. Преобразуйте данные, рассмотрев максимальное значение уровня воды за сутки. Вам нужно по данным до октября 2018 года не включительно построить верхнюю границу предсказательного интервала уровня воды и сравнить ее с максимальным значением, достигавшимся в октябре 2018 года.

Предсказательный интервал постройте в три этапа:

1. Бутстрепный доверительный интервал для среднего значения максимального уровня воды за сутки;
2. Бутстрепный доверительный интервал для стандартного отклонения максимального уровня воды за сутки;
3. Сложите границу доверительного интервала для среднего с границей доверительного интервала для стандартного отклонения, домноженной на 2.

Рассмотрите три способа построения бутстрепных доверительный интервалов, рассказанные на лекции.

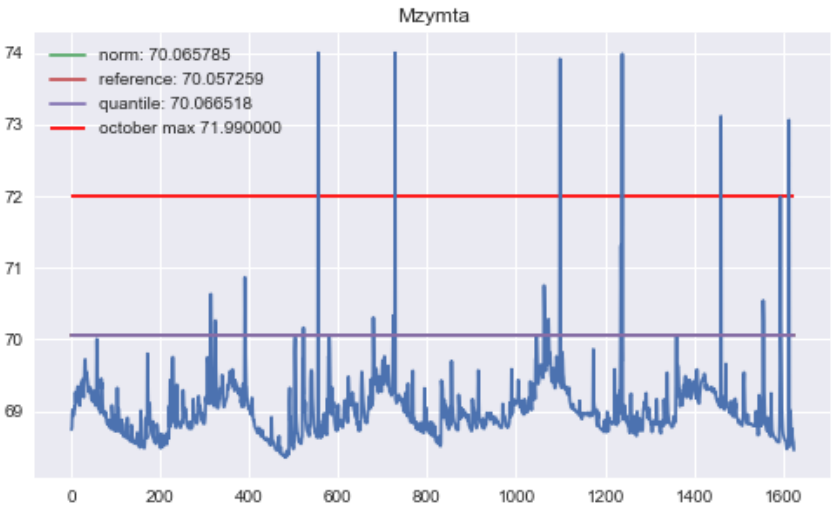
Сделайте выводы.

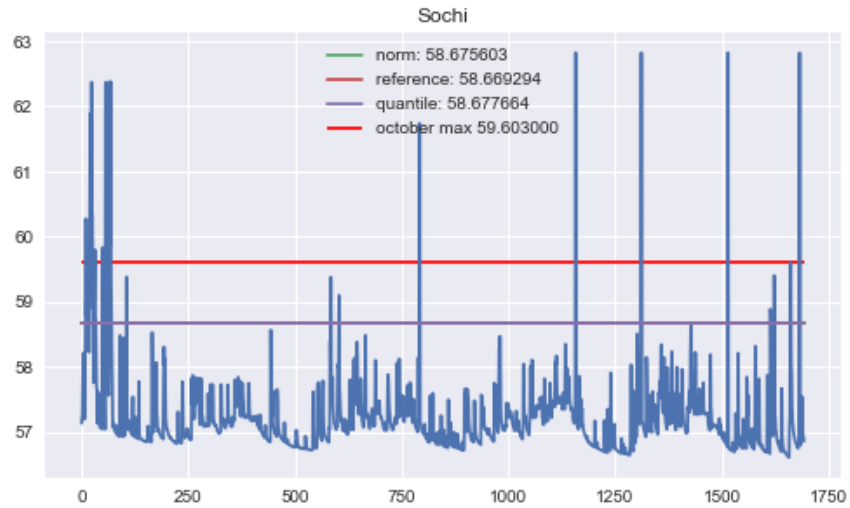
In [65]:
```
from math import floor
from math import ceil
```

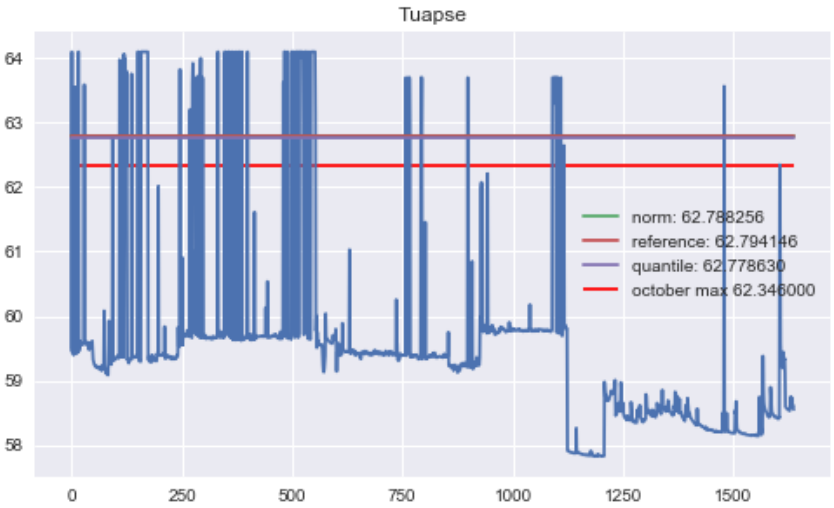
In [87]:
```
def find_confidence_interval(sample, statistics=np.mean, method='norm', alpha=0.95, B=1000):
    B = sample.size
    estimator = statistics(sample)
    bootstrap = statistics(np.random.choice(sample, size=(B, sample.size)), axis=1)

    if method == 'norm':
        return estimator + np.std(bootstrap)*sps.norm.ppf((1+alpha)/2)
    if method == 'reference':
        return 2*estimator - np.sort(bootstrap)[floor(B*(1-alpha)/2)]
    if method == 'quantile':
        return np.sort(bootstrap)[ceil(B*(1+alpha)/2)]
```

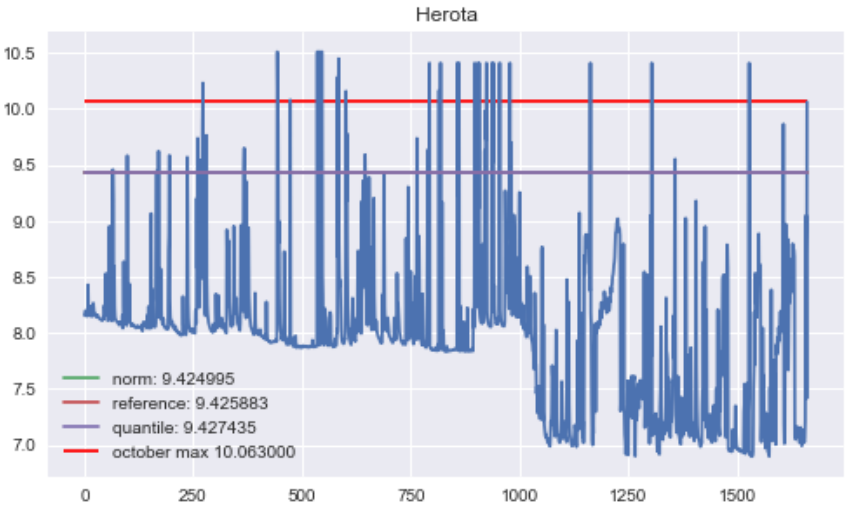
In [88]:
```
def find_prediction_interval(sample, method='norm'):
    return find_confidence_interval(sample, method=method) + 2*find_confidence_interval(sample, statistics=np.std, met
hod=method)
```

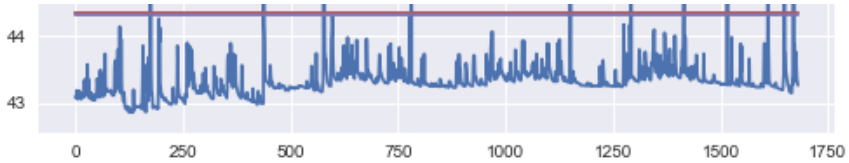
In [123]:
```
plt.figure(figsize=(18, 16))
for i, river in enumerate(['Mzymta', 'Sochi', 'Tuapse', 'Herota', 'Hosta']):
    plt.subplot(3, 2, i+1)
    plt.title(river)

    sample = pd.read_csv(river+".csv", sep='\t')[['Время', 'Уровень воды (по БСВ)']]
    sample['day'] = sample['Время'].str.split(' ').str.get(0)
    sample = sample.groupby(['day'])['Уровень воды (по БСВ)'].max()
    before_october = sample[:'2018-09-30']
    max_october = sample['2018-10-01':'2018-10-31'].max()
    plt.plot(sample.values)
    plt.hlines(max_october, 0, sample.size, color='r', label='october max %f'%max_october)

    for method in ['norm', 'reference', 'quantile']:
        upper_bound = find_prediction_interval(before_october, method=method)
        plt.plot([0, sample.size], [upper_bound, upper_bound], label='%s: %f'%(method, upper_bound))

    plt.legend()

plt.show()
```









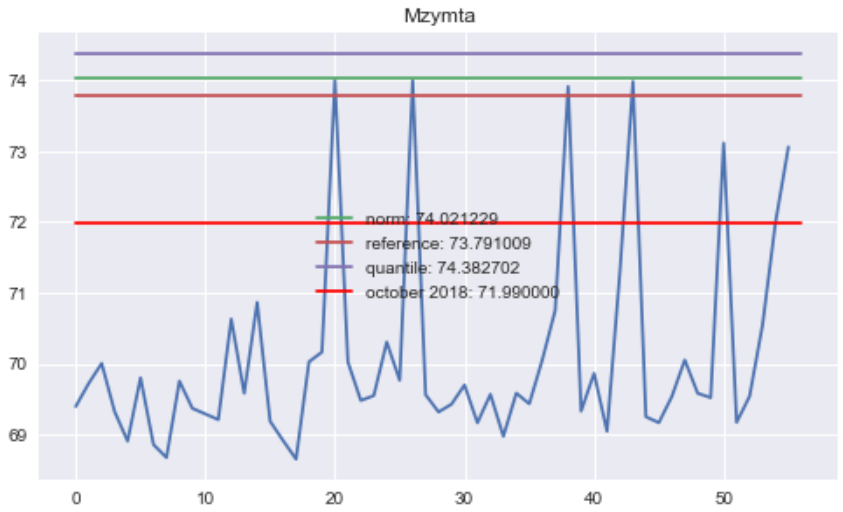


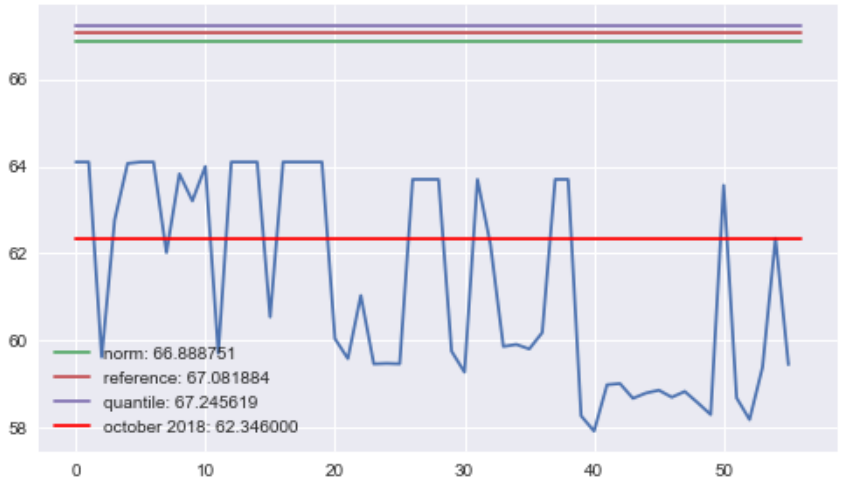
Вывод: Все методы строят примерно одинаковые доверительные интервалы. Только в Туапсе доверительный интервал содержит максимальный уровень воды в октябре. Для остальных городов и рек максимальный уровень воды в октябре довольно далек от интервала. С Туапсе скорее всего повезло, по графику видно, что в какой-то момент уровень воды резко снизился.

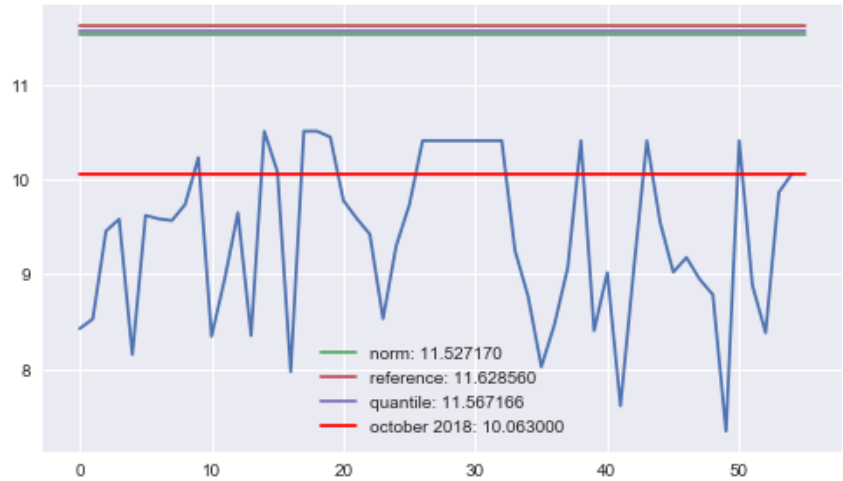
Наверно методы плохо работают потому что они рассчитаны на нормальное распределение, а уровень воды в реке имеет другое.

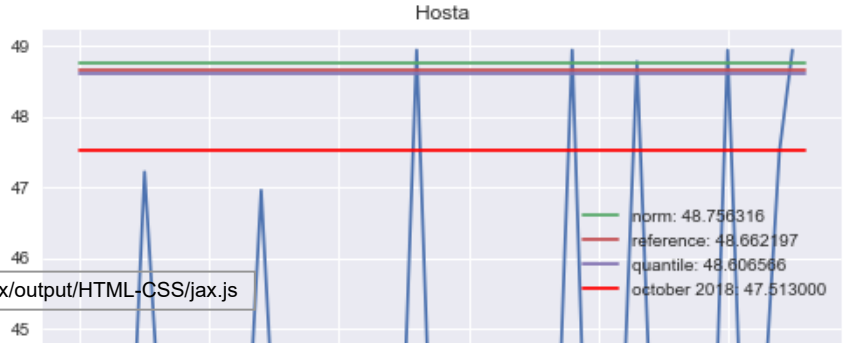
Еще одна возможная причина заключаетя в том, что эти методы вообще решают другую задачу. Мы сравниваем предсказание для максимального уровня воды за день с максимальным за месяц. Логично, что предсказание получится плохим. Ниже сделано предсказание, где вместо максимума за день рассматривается максимум за месяц.

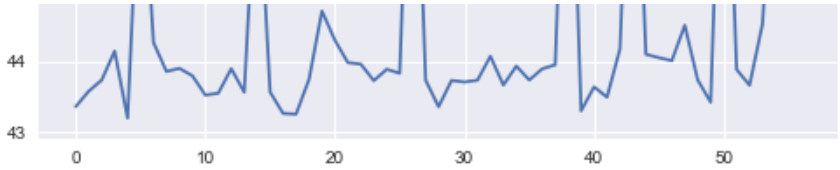
In [121]:
```
plt.figure(figsize=(18, 16))
for i, river in enumerate(['Mzymta', 'Sochi', 'Tuapse', 'Herota', 'Hosta']):
    plt.subplot(3, 2, i+1)
    plt.title(river)

    sample = pd.read_csv(river+".csv", sep='\t')[['Время', 'Уровень воды (по БСВ)']]
    sample['month'] = sample['Время'].str[:7]
    sample = sample.groupby(['month'])['Уровень воды (по БСВ)'].max()
    plt.plot(sample.values)
    before_october = sample[:'2018-09']
    for method in ['norm', 'reference', 'quantile']:
        upper_bound = find_prediction_interval(before_october, method=method)
        plt.plot([0, sample.size], [upper_bound, upper_bound], label='%s: %f'%(method, upper_bound))

    plt.plot([0, sample.size], [sample['2018-10'], sample['2018-10']],
             label='%s: %f'%('october 2018', sample['2018-10']), color='r')
    plt.legend()
plt.show()
```











Вывод: Таким методом получается лучшая оценка. Хотя она и менее устойчива к выборосам.

На самом деле, данные очень странные. Смотря на них нельзя сделать вывод о том, что в октябре 2018 в Сочи была катастрофа, потому что уровень воды за 2014-2018 годы довольно часто превышал максимальный уровень воды в октябре 2018. Еще одна странность данных в том, что на графике Мзымты можно ясно увидеть сезонные паводки, вызванные таянием снегов в горах. Хоста также как и Мзымта должна подвергаться влиянию таяния снегов, однако по графику это совсем не видно. В общем, я бы не особо доверяла этим данным.

Проверка статистических гипотез

Задача 7.

Существует примета, что если перед вам дорогу перебегает черный кот, то скоро случится неудача. Вы же уже достаточно хорошо знаете статистику и хотите проверить данную примету. Сформулируем задачу на математическом языке.

Пусть $X_1, \ldots, X_n \sim Bern(p)$ --- проведенные наблюдения, где $X_i = 1$, если в i-м испытании случилась неудача после того, как черный кот перебежал дорогу, а p --- неизвестная вероятность такого события. Вы хотите проверить гипотезу $\mathrm{H}_0 : p = 1/2$ (отсутствие связи между черным котом и неудачей) против альтернативы $\mathrm{H}_1 : p > 1/2$ (неудача происходит чаще если черный кот перебегает дорогу).

Известно, что $S = \left\{ T(X) \geqslant c_\alpha \right\}$, где $T(X) = \sum X_i$, является равномерно наиболее мощным критерием для проверки этих гипотез. Чему при этом равно c_α и как определяется p-value?

Заметим, что $T(X) \sim Bin(n, p)$

Значит, c_α это α-квантиль распределения $Bin(n, 1/2)$.

$p(t) = P_0(T(X) \geq t) = 1 - F_{T(X)}(t) = sps.binom.sf(t, n, p)$

Для начала проверьте, что критерий работает. Возьмите несколько значений n и реализаций статистики $T(X)$. В каждом случае найдите значение c_α и p-value. Оформите это в виде таблицы (можно через `pandas.DataFrame`).

Пользуйтесь функциями из `scipy.stats`. Внимательно проверьте правильность строгих и нестрогих знаков.

In [4]:
```
alpha = 0.95
n = [10, 30, 100, 300]
p = np.ones(4)*0.6

table = pd.DataFrame({'n': n, 'p': p})
table['T(X)'] = sps.binom.rvs(n, p)
table['c_alpha'] = sps.binom.ppf(alpha, n, np.ones(4)*1/2)
table['отвергается'] = table['T(X)'] > table['c_alpha']
table['p-value'] = sps.binom.sf(table['T(X)'], n, np.ones(4)*1/2)
table['p-value<0.05'] = table['p-value'] < 0.05
table
```

Out[4]:

	n	p	T(X)	c_alpha	отвергается	p-value	p-value<0.05
0	10	0.6	6	8.0	False	0.171875	False
1	30	0.6	16	19.0	False	0.292332	False
2	100	0.6	56	58.0	False	0.096674	False
3	300	0.6	184	164.0	True	0.000032	True

Для каких истинных значений p с точки зрения практики можно считать, что связь между черным котом и неудачей есть?

Для $p \geq 0.6$

Теперь сгенерируйте 10 выборок для двух случаев: 1). $n = 5, p = 0.75$; 2). $n = 10^5, p = 0.51$.

В каждом случае в виде таблицы выведите реализацию статистики $T(X)$, соответствующее p-value и 0/1 -- отвергается ли H_0 (выводите 1, если отвергается).

In [36]:
```
def make_table(n=150, p=1/2, number_of_samples = 10):
    table = pd.DataFrame({'T(X)': sps.binom.rvs(n, p, size=number_of_samples)})

    table['p-value'] = sps.binom.sf(table['T(X)'], n, 1/2)
    table['отвергается'] = table['T(X)'] > sps.binom.ppf(alpha, n, 1/2)
    return table
```

In [37]:
```
make_table(n=5, p=0.75)
```

Out[37]:

	T(X)	p-value	отвергается
0	3	0.18750	False
1	1	0.81250	False
2	4	0.03125	False
3	3	0.18750	False
4	4	0.03125	False
5	2	0.50000	False
6	4	0.03125	False
7	4	0.03125	False
8	2	0.50000	False
9	3	0.18750	False

In [38]:
```
make_table(n=10**5, p=0.51)
```

Out[38]:

	T(X)	p-value	отвергается
0	50814	1.292332e-07	True
1	50999	1.294358e-10	True
2	50996	1.463253e-10	True
3	50702	4.434225e-06	True
4	50945	1.115412e-09	True
5	51080	4.130613e-12	True
6	50689	6.478016e-06	True
7	50966	4.892816e-10	True
8	50948	9.925862e-10	True
9	50842	4.949738e-08	True

Вывод:

В первом случае размер выборок мал, а отличие вероятности от $1/2$ существенно. Однако нулевая гипотеза редко отвергалась.

Во втором случае размер выборок очень велик, но вероятность $p = 0.51$ несущественно отличается от $1/2$. Однако нулевая гипотеза всегда отвергается.

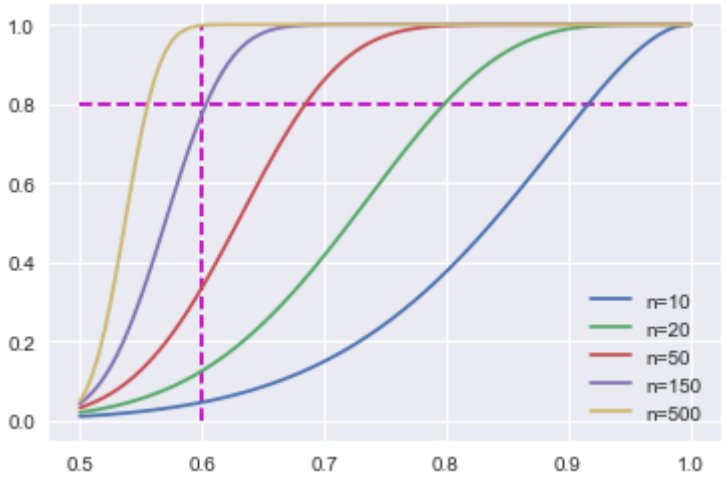
Значит, нужно контролировать размер выборок.

Возникает задача подбора оптимального размера выборки.

Для этого сначала зафиксируйте значение $p^* > 1/2$, которое будет обладать следующим свойством. Если истинное $p > p^*$, то такое отклонение от $1/2$ с практической точки зрения признается существенным, то есть действительно чаще случается неудача после того, как черный кот перебегает дорогу. В противном случае отклонение с практической точки зрения признается несущественным.

Теперь для некоторых n постройте графики функции мощности критерия при $1/2 < p < 1$ и уровне значимости 0.05. Выберите такое n^*, для которого функция мощности дает значение 0.8 при p^*.

In [63]:
```
grid = np.linspace(1/2, 1, 100)
p_star = 0.6
good_b = 0.8

plt.grid(True, which='major')
plt.plot([1/2, 1], [0.8, 0.8], 'm--')
plt.plot([p_star, p_star], [0, 1], 'm--')

for n in [10, 20, 50, 150, 500]:
    plt.plot(grid, sps.binom(n=n, p=grid).sf(sps.binom.ppf(alpha, n, 1/2)), label='n=%d'%n)

plt.legend();
```



Возьмем $n^* = 150$

Для выбранного n^* проведите эксперимент, аналогичный проведенным ранее экспериментам, сгенерировав выборки для следующих истинных значений p: 1). $1/2 < p < p^*$; 2). $p > p^*$.

In [45]:
```
make_table(n=150, p=0.51)
```

Out[45]:

	T(X)	p-value	отвергается
0	79	0.231275	False
1	80	0.184581	False
2	68	0.855772	False
3	72	0.658384	False
4	71	0.716112	False
5	78	0.283888	False
6	70	0.768725	False
7	74	0.532519	False
8	62	0.979566	False
9	69	0.815419	False

In [53]:
```
make_table(n=150, p=0.61)
```

Out[53]:

	T(X)	p-value	отвергается
0	93	0.001203	True
1	90	0.005561	True
2	103	0.000001	True
3	85	0.043036	False
4	98	0.000055	True
5	102	0.000003	True
6	92	0.002057	True
7	97	0.000108	True
8	93	0.001203	True
9	89	0.008799	True

Вывод:

Размер выборки $n^* = 150$ хорошо подобран, потому что при $p = 0.51 < 0.6 = p^*$ нулевая гипотеза не отвергается, а при $p = 0.61 > 0.6 = p^*$ нулевая гипотеза в основном отвергается.

Задача 8.

В Долгопрудном крупная торговая сеть Y10 имеет 100 магазинов и планирует открыть еще 5 магазинов. Сеть Y10 считает магазин успешным, если его дневная выручка в 27 днях из 30 превышает некоторый установленный порог. Благодаря модели машинного обучения, обученной на предыдущих 100 магазинах, было выбрано 5 потенциальных точек, в которых и решено было открыть новые магазины.

Вам предоставлена чистая выручка по каждому из 5 магазинов за день в течении трех месяцев ($31 + 30 + 31$ день) работы этих пяти магазинов. Считается, что магазин успешен в течении дня, если его выручка за этот день превышает 50000 рублей. По этому правилу сопоставьте каждому магазину набор бернуллиевских случайных величин, которе принимают значение 1, если магазин успешен в течении дня. Для простоты будем считать, что выручка за день не зависит от аналогичных показателей за предыдущие дни, и ее распределение не меняется во времени, то есть данные образуют выборку.

Модель машинного обучения выдает также параметры для априорного бета-распределения распределения по каждому из магазинов:

1. $Beta(1, 1)$;
2. $Beta(2900, 100)$;
3. $Beta(29, 1)$;
4. $Beta(29, 1)$;
5. $Beta(1, 1)$.

Проанализируйте данные и ответьте на следующие вопросы.

1. Для каких из магазинов можно утверждать, что с вероятностью 0.95 их можно считать успешными? Для ответа на этот вопрос нужно выполнить байесовскую проверку гипотез $\mathrm{H}_0\colon\ p = 27/30$ *vs.* $\mathrm{H}_1\colon\ p > 27/30$, посчитав апостериорную вероятность события $\{p > 27/30\}$. Если эта вероятность не меньше 0.95, магазин можно считать успешным.
2. Про какие из магазинов можно сказать, что их лучше закрыть? Закрывать магазин затратно, для этого он должен быть убыточным в 27 днях из 30. Для ответа на этот вопрос нужно выполнить байесовскую проверку гипотез $\mathrm{H}_0\colon\ p = 3/30$ *vs.* $\mathrm{H}_1\colon\ p < 3/30$, посчитав апостериорную вероятность события $\{p < 3/30\}$. Если эта вероятность не меньше 0.95, магазин можно закрыть.
3. Что можно сказать об остальных магазинах?
4. Выполнены ли предположения модели на практике? Имеет ли смысл собирать данные сразу после открытия магазина?

In [113]:
```
sample = pd.read_csv("Y10.csv", sep='\t')
n = sample.shape[0]
sample = (sample > 50000).astype(int).sum(axis=0)
sample
```

Out[113]:
```
Shop 1    91
Shop 2    31
Shop 3     2
Shop 4    36
Shop 5     4
dtype: int64
```

In [114]:
```
prior = pd.DataFrame({
    'Shop 1': [1, 1],
    'Shop 2': [2900, 100],
    'Shop 3': [29, 1],
    'Shop 4': [29, 1],
    'Shop 5': [1, 1]},
    index=['a', 'b'])

posterior = pd.DataFrame([sample, n-sample], index=['a', 'b']) + prior
posterior
```

Out[114]:

	Shop 1	Shop 2	Shop 3	Shop 4	Shop 5
a	92	2931	31	65	5
b	2	161	91	57	89

In [115]:
```
posterior_distribution = sps.beta(a=posterior.loc['a'], b=posterior.loc['b'])

success = posterior_distribution.sf(27/30) > 0.95
close = posterior_distribution.cdf(3/30) > 0.95
result = pd.DataFrame([success, close], columns=prior.columns,
                      index=['is successful', 'should be closed'])
result
```

Out[115]:

	Shop 1	Shop 2	Shop 3	Shop 4	Shop 5
is successful	True	True	False	False	False
should be closed	False	False	False	False	True

Вывод:

1) Можно считать успешными первый и второй магазины.

2) Лучше закрыть пятый магазин.

3) Про третий и четвертый магазин нельзя сказать, что они выгодны или их нужно закрывать.

4) У модели не было никаких предсказаний по поводу первого и пятого магазинов. Модель предполагала, что второй магазин будет очень успешен, однако он был успешен лишь в одной трети случаев. Также были предположения, что третий и четвертый магазины будут успешными, но судя по данным, этого не случилось. Кажется, что предположения модели были не очень хорошими.

Не стоит собирать данные сразу после открытия магазина. Нужно сначала рассчитать размер выборки, то есть сколько дней нужно ждать, и только по прошествии этого времени делать выводы. Как показала предыдущая задача, важно чтобы выборка была подходящего размера.

Возмножно не стоит судить по первым дням работы магазина, еще и потому что людям нужно какое-то время чтобы узнать про новый магазин, оценить его.