In [1]:

```
import numpy as np
import pandas as pd
import scipy.stats as sps
import warnings

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifie
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.calibration import CalibrationDisplay
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_blobs, make_classification

import seaborn as sns
import matplotlib.pyplot as plt

warnings.filterwarnings('ignore')
sns.set(style='darkgrid', font_scale=1.7, palette='Set2')
```

executed in 794ms, finished 16:27:06 2022-11-29

1. Калибровочная кривая

При решении задачи классификаци нам часто требуется помимо метки класса предсказать еще и вероятность принадлежности соответствующему классу. Использование оценок вероятностей позволяет дать более полный ответ и оставляет некоторую гибкость при решении задачи.

- Например, для оценки бизнес-рисков требуется несмещенно оценить вероятности.
- Или же для оценки риска осложнений после планируемой медицинской операции.
- Кроме того, даже если нам не нужны сами вероятности, мы можем легко поменять порог для бинарной классификации и получить некоторый прирост по целевым метриками.

При использовании моделей, которые предсказывают вероятности, хотелось бы, чтобы оценки вероятностей хорошо оценивали сами вероятности. Другими словами, если мы рассмотрим множество объектов, для которых $\widehat{p} \approx 0.8$, то разумно ожидать, что около 80% из этих рассмотренных объектов действительно имеют положительную метку. Отклонение может оказаться критичным, например, в задаче кредитного скоринга, где недооценка рисков может привести к потере средств.

Существуют некоторые методы для оценки того, насколько хорошо скалибрована модель, а также средства для калибровки оценок вероятностей.

Построить калибровочную кривую можно одним их следующих вариантов

1. `sklearn.calibration.CalibrationDisplay(prob_true, prob_pred, y_prob, *, estimator_name=None, pos_label=None)`

    - `prob_true` , размер `(n_bins,)` — доля объектов класса 1 в бине;
    - `prob_pred` , размер `(n_bins,)` — среднее предсказание в бине;
    - `y_prob` , размер `(n_samples,)` — оценки вероятностей класса 1 для элементов выборки.

2. `sklearn.calibration.CalibrationDisplay.from_estimator(estimator, X, y, *[, n_bins, ...])`

   - `estimator` — обученный классификатор;
   - `X`, `y` — тестовая выборка.

3. `sklearn.calibration.CalibrationDisplay.from_predictions(y_true, y_prob, *[, ...])`

   - `y_true`, размер `(n_samples,)` — метки класса;
   - `y_prob`, размер `(n_samples,)` — оценки вероятностей класса 1 для элементов выборки;

In [2]:

```
def calibration_curves(clf_list, figsize=(12, 7)):
    '''
    Отрисовка калибровочной кривой.

    clf_list -- список кортежей (обученный классификатор, его название)
    figsize -- размер фигуры
    '''

    f, ax = plt.subplots(figsize=figsize)

    for i, (clf, name) in enumerate(clf_list):
        probs = clf.predict_proba(X_test)[:, 1]
        CalibrationDisplay.from_predictions(
            y_test, probs, n_bins=20, lw=3, name=name, ax=ax, strategy='quanti
        )
    ax.set_xlabel('Предсказанная вероятность (среднее внутри бина)')
    ax.set_ylabel('Доля класса 1 в бине')
    ax.set_title('Сравнение калибровочных кривых')
    plt.show();
```

executed in 4ms, finished 16:27:06 2022-11-29

In [3]:

```python
def draw_probs_hist(clf_list, X_test, ncols=3):
    '''
    Отрисовка гистограмм предсказаний вероятностей.

    clf_list -- список кортежей (обученный классификатор, его название)
    X_test -- тестовая выборка
    ncols -- количество колонок для отрисовки графика
    '''

    with sns.axes_style('whitegrid'):
        nrows = np.ceil(len(clf_list) / ncols)
        plt.figure(figsize=(18, 4*nrows))

        for i, (clf, name) in enumerate(clf_list):
            plt.subplot(nrows, ncols, i+1)
            y_prob = clf.predict_proba(X_test)[:, 0]
            plt.hist(y_prob, range=(0, 1), bins=10, label=name)
            plt.title(name)
            plt.xlabel('Предсказ. вероятность')

        plt.tight_layout()
        plt.show()
```

executed in 10ms, finished 16:27:06 2022-11-29

In [4]:

```
def draw_prob_predictions(clf, X_test, name, step=0.1):
    '''
    Отрисовка предсказаний вероятностей.

    clf -- обученный классификатор
    X_test -- тестовая выборка
    name -- имя классификатора
    step -- шаг сетки
    '''

    x_min, x_max = X_test[:, 0].min(), X_test[:, 0].max()
    y_min, y_max = X_test[:, 1].min(), X_test[:, 1].max()

    X_grid = np.mgrid[x_min:x_max:step, y_min:y_max:step]
    size_x, size_y = X_grid.shape[1:]
    X_grid_list = X_grid.reshape((2, size_x*size_y)).T

    probs_pred = clf.predict_proba(X_grid_list)[:, 0]

    plt.figure(figsize=(10, 7))
    plt.title(f'Предсказания {name}')
    plt.pcolormesh(
        X_grid[0], X_grid[1], probs_pred.reshape((size_x, size_y)), cmap='Gree
        vmin=0, vmax=1
    )
    plt.scatter(
        X_test[:, 0], X_test[:, 1],
        c=y_test, alpha=0.5, cmap='cool'
    )
    plt.xlim((x_min, x_max)), plt.ylim((y_min, y_max))
    plt.xlabel('Признак 1')
    plt.ylabel('Признак 2')
    plt.show()
```

executed in 8ms, finished 16:27:06 2022-11-29

1.1 Линейное изменение вероятности классов

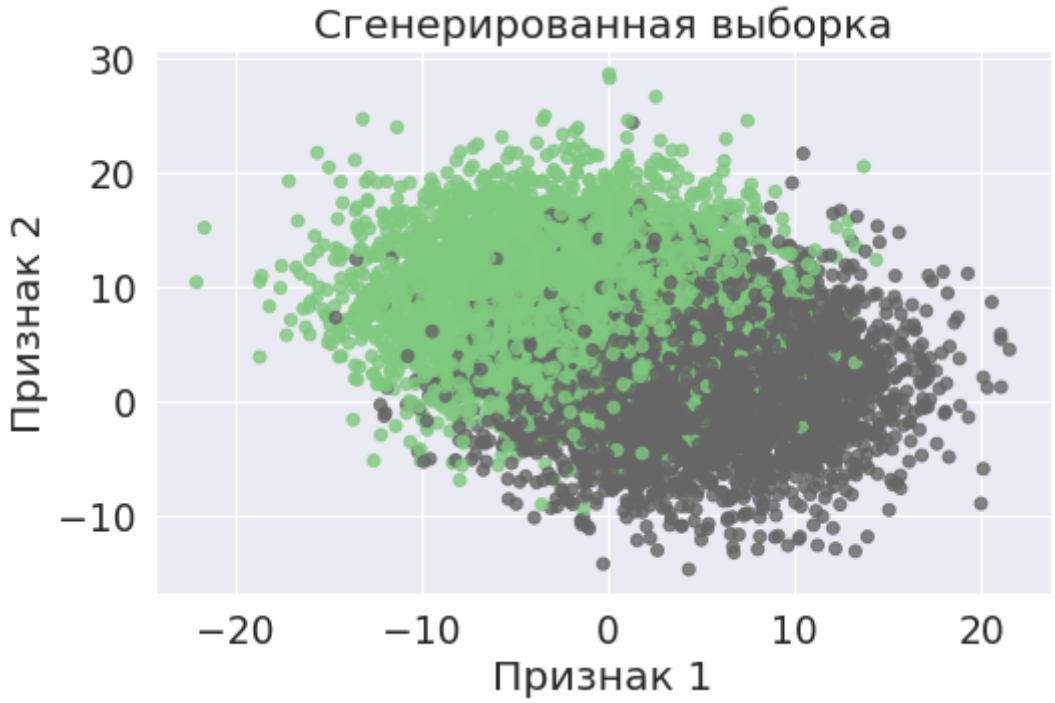
Сгенерируем датасет из двух пересекающихся шарообразных классов

In [5]:

```
X, y = make_blobs(
    n_samples=10000, n_features=2, centers=2,
    cluster_std=5.0, random_state=42
)

plt.figure(figsize=(8, 5))
plt.title('Сгенерированная выборка')
plt.scatter(
    X[:, 0], X[:, 1],
    c=y, alpha=0.8, cmap='Accent'
)
plt.xlabel('Признак 1')
plt.ylabel('Признак 2')
plt.show()
```

executed in 309ms, finished 16:27:06 2022-11-29



Проверим линейность логита по каждому признаку

In [6]:

```
h = 1  # ширина ядра
size = 5000 # проверим линейность на небольшой подвыборке

plt.figure(figsize=(16, 5))
# цикл по признакам
for feature_idx in range(X.shape[1]):
    # отсортированная сетка по признаку feature_idx
    # для построения ядерной оценки
    x_grid = np.linspace(
        np.percentile(X[:, feature_idx], 5),
        np.percentile(X[:, feature_idx], 95),
        100
    )
    # гауссовское ядро с шириной h
    kernel = sps.norm(scale=h)
    # значения ядра в точках выборки
    kernel_values = kernel.pdf(X[:size, feature_idx][:size, np.newaxis] - x_gr
    # оценка по ядерной регрессии для признака feature_idx
    y_est = (y[:size, np.newaxis] * kernel_values).sum(axis=0) / kernel_values
    # приближение логита по оценкам y_est
    l_sm = np.log(y_est / (1 - y_est))

    # отрисуем графики
    plt.subplot(1, 2, feature_idx + 1)
    plt.plot(x_grid, l_sm, lw=5)
    plt.xlabel(f'Признак {feature_idx}')
    plt.ylabel('Приближение логита')

plt.suptitle('Проверка линейности логита')
plt.tight_layout()
plt.show()
```

executed in 349ms, finished 16:27:06 2022-11-29

Разделите выборку на тестовую и обучающую. Кроме того, для калибровки необходимо выделить отдельную выборку.

Использование обучающей выборки для калибровки скорее всего приведет к смещению нашего классификатора, т.к. на обучающей выборки классификатор ведет себе гораздо лучше, а значит будет сильнее прижимать вероятности к 0 и 1.

Примечание. Тестовая часть выборки не используется в этом ноутбуке. Она выделена для того, чтобы показать общий принцип и избежать возможных ошибок в реальных задачах.

In [7]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
X_train, X_calib, y_train, y_calib = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42
)
```

executed in 5ms, finished 16:27:06 2022-11-29

Обучим несколько классификаторов

In [8]:

```
lr = LogisticRegression().fit(X_train, y_train)
rf = RandomForestClassifier(random_state=42).fit(X_train, y_train)
nb = GaussianNB().fit(X_train, y_train)
```

executed in 672ms, finished 16:27:07 2022-11-29

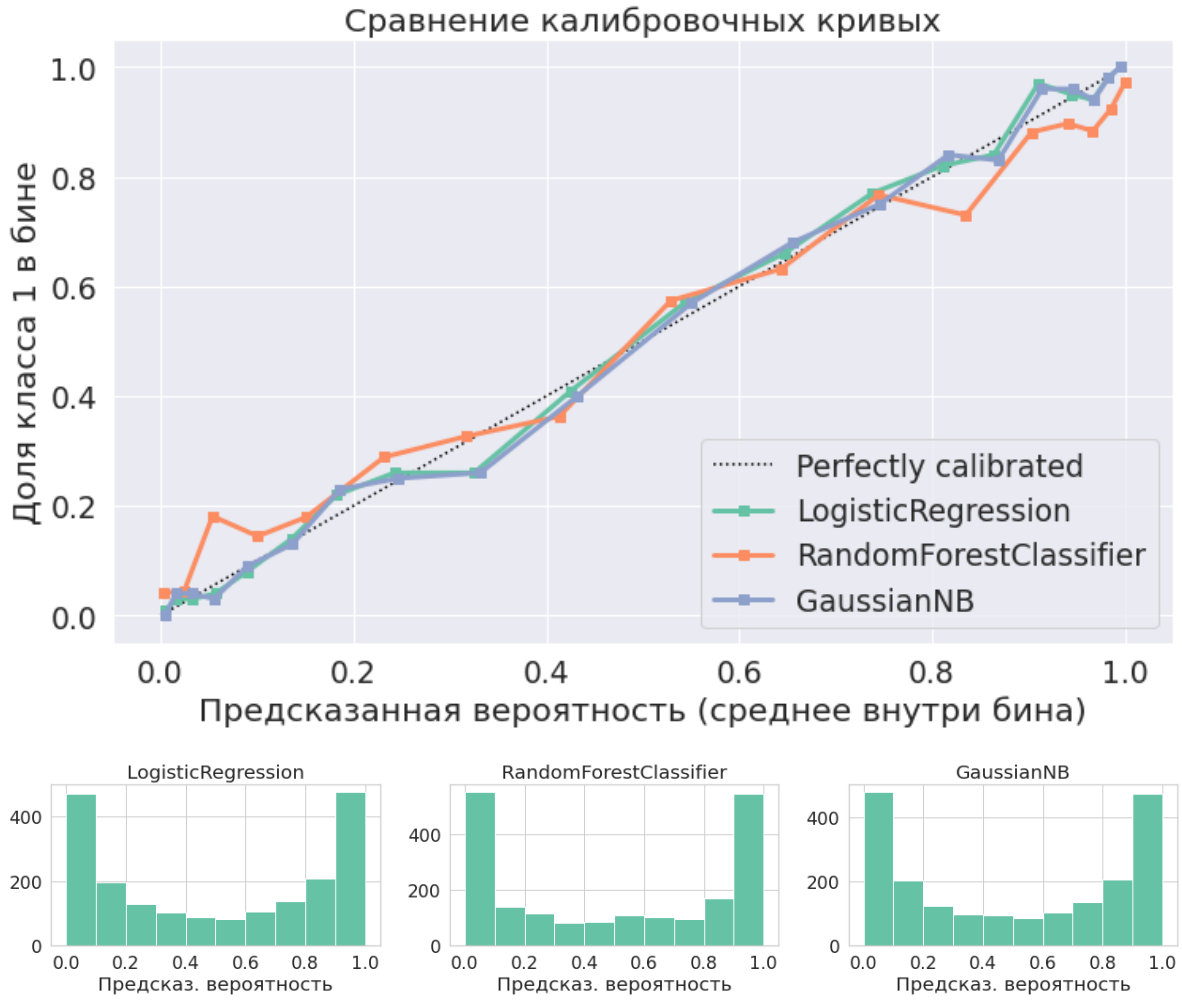
Посмотрим на их калибровочные кривые, а также гистограммы значений прогнозов

In [9]:

```
clf_list = [
    (lr, 'LogisticRegression'),
    (rf, 'RandomForestClassifier'),
    (nb, 'GaussianNB'),
]

calibration_curves(clf_list)
draw_probs_hist(clf_list, X_test)
```

executed in 737ms, finished 16:27:08 2022-11-29



Можно заметить, что прогнозы для всех классификаторов оказываются достаточно хорошо-скалиброванными. Некоторое отклонение наблюдается у случайного леса.

1.2 Более сложная выборка

Сгенерируем выборку, в которой классы не являются линейно-разделимыми, а также добавим к ней фоновый шум из точек разных классов.

In [10]:

```
X, y = make_blobs(
    n_samples=20000, n_features=2, centers=4,
    cluster_std=2.1, random_state=21
)
y = (y>=2).astype(int)

# Шум
n_noise = 2000
X = np.vstack([X, sps.uniform(loc=-17, scale=30).rvs((n_noise, 2))])
y = np.hstack([y, sps.bernoulli(p=0.5).rvs(n_noise)])

# Посмотрим на выборку
plt.figure(figsize=(12, 6))
plt.title('Сгенерированная выборка')
plt.scatter(
    X[:, 0], X[:, 1],
    c=y, alpha=0.15, cmap='Accent'
)
plt.xlabel('Признак 1')
plt.ylabel('Признак 2')
plt.show()
```

executed in 559ms, finished 16:27:08 2022-11-29

Логистическая регрессия

Для начала проверим, выполнена ли линейность логита по каждому из признаков

In [11]:

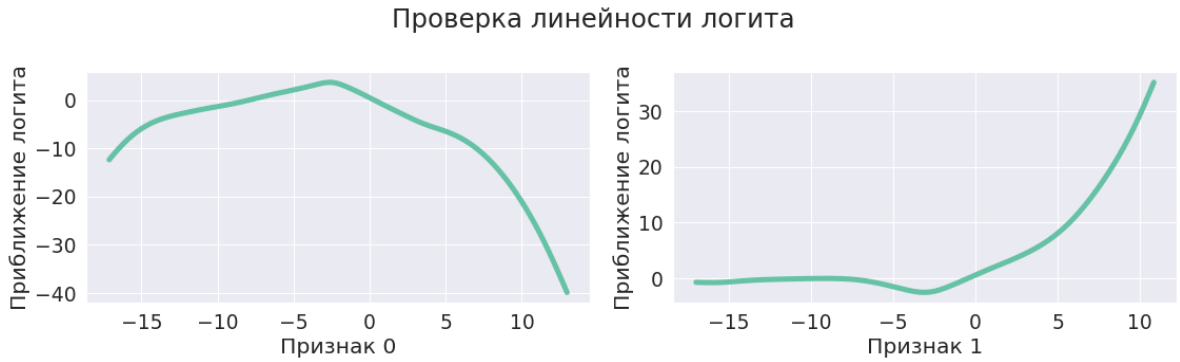
```
h = 1  # ширина ядра
size = 5000 # проверим линейность на небольшой подвыборке

plt.figure(figsize=(16, 5))
# цикл по признакам
for feature_idx in range(X.shape[1]):
    # отсортированная сетка по признаку feature_idx
    # для построения ядерной оценки
    x_grid = np.linspace(X[:, feature_idx].min(), X[:, feature_idx].max(), 10(
    # гауссовское ядро с шириной h
    kernel = sps.norm(scale=h)
    # значения ядра в точках выборки
    kernel_values = kernel.pdf(X[:size, feature_idx][:size, np.newaxis] - x_gr
    # оценка по ядерной регрессии для признака feature_idx
    y_est = (y[:size, np.newaxis] * kernel_values).sum(axis=0) / kernel_values
    # приближение логита по оценкам y_est
    l_sm = np.log(y_est / (1 - y_est))

    # отрисуем графики
    plt.subplot(1, 2, feature_idx + 1)
    plt.plot(x_grid, l_sm, lw=5)
    plt.xlabel(f'Признак {feature_idx}')
    plt.ylabel('Приближение логита')

plt.suptitle('Проверка линейности логита')
plt.tight_layout()
plt.show()
```

executed in 385ms, finished 16:27:09 2022-11-29



Вполне логично, что в этот раз логит нелинеен.

Аналогично разделим выборку на три части.

Примечание. Тестовая часть выборки не используется в этом ноутбуке. Она выделена для того, чтобы показать общий принцип и избежать возможных ошибок в реальных задачах.

In [12]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
X_train, X_calib, y_train, y_calib = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42
)
```

executed in 5ms, finished 16:27:09 2022-11-29

Обучим логистическую регрессию

In [13]:

```
lr = LogisticRegression().fit(X_train, y_train)
```

executed in 37ms, finished 16:27:09 2022-11-29

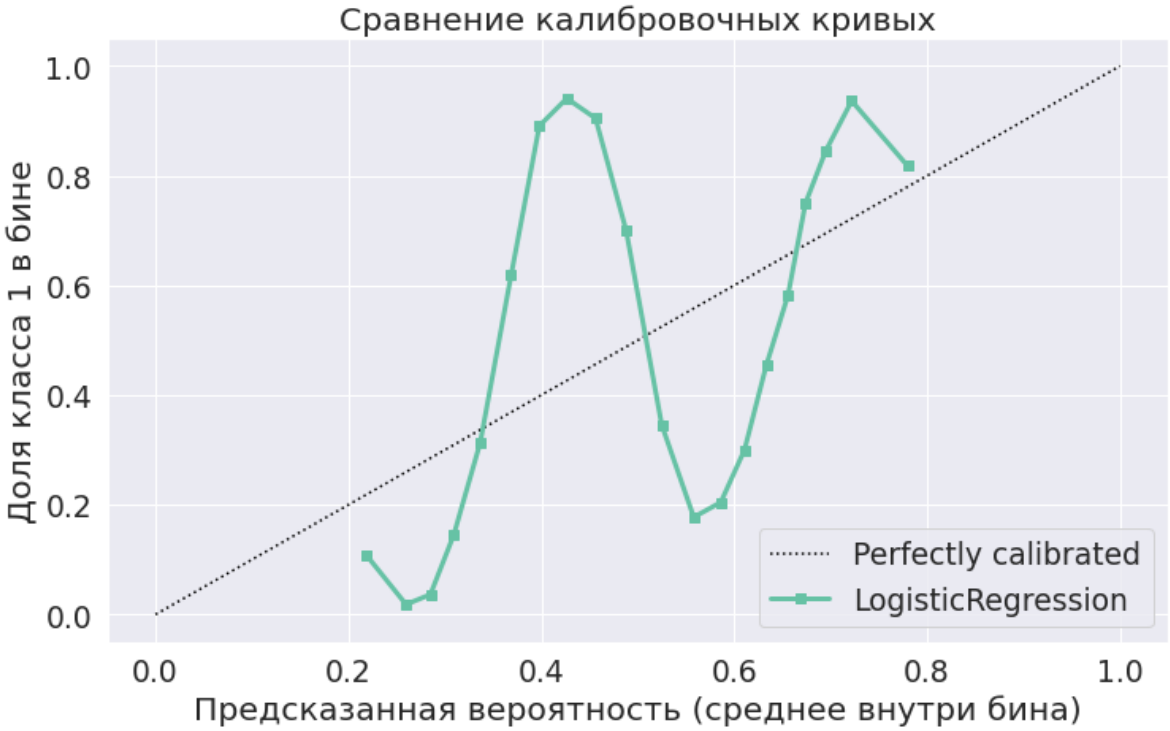
Посмотрим на ее результаты

In [14]:

```
clf_list = [
    (lr, 'LogisticRegression'),
]

calibration_curves(clf_list)
draw_probs_hist(clf_list, X_test)
```

executed in 425ms, finished 16:27:09 2022-11-29



Как и ожидалось, предсказанные вероятности неоткалиброваны. Это происходит из-за нелинейности зависимостей в данных. Как мы видим из гистограммы предсказанных вероятностей, модель в основном неуверена в классификации.

Посмотрим ниже на график предсказаний вероятностей, где хорошо видно эффект неуверенного предсказания. Цвета на графике отмастабированы от темно-зеленого до белого, крайних цветов на этом графике нет.

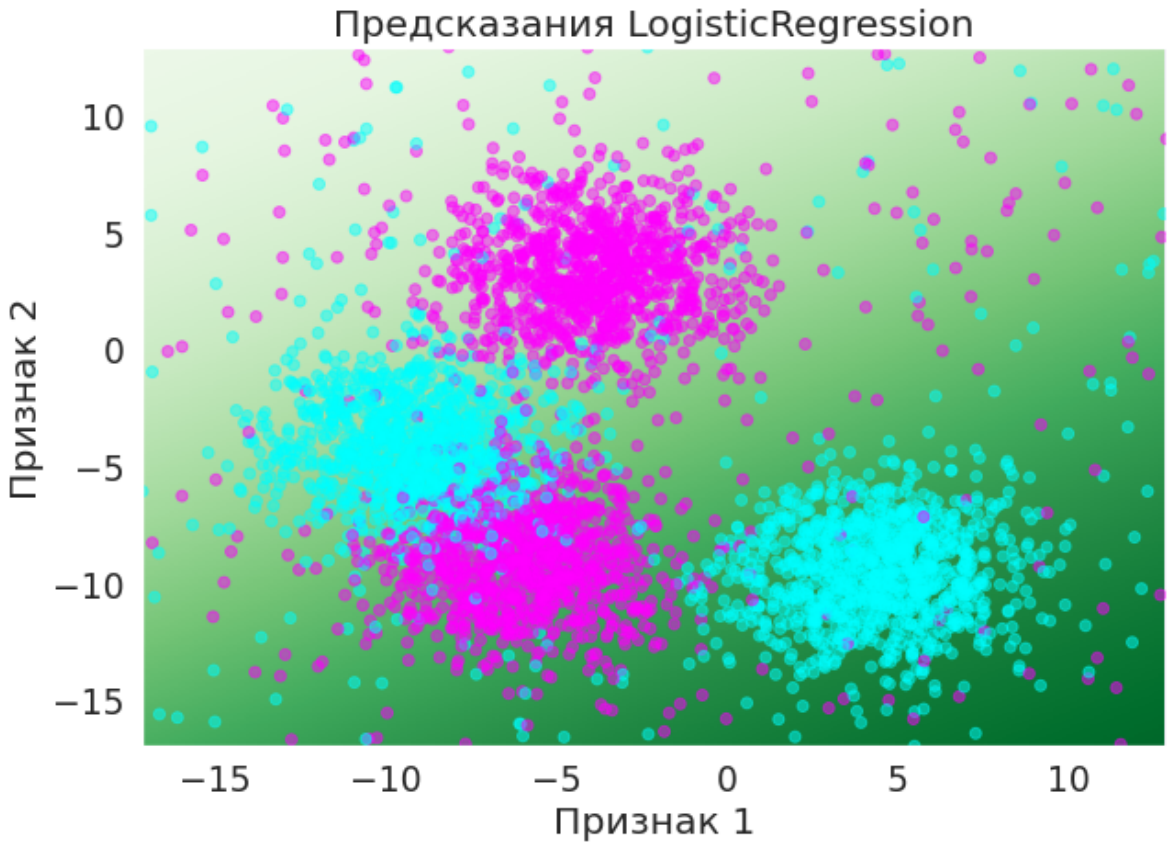
Модель в целом уловила тренд изменения вероятностей классов по дальним кластерам. Но для двух "слипвшихся" кластеров модель предсказывает вероятности неправильно. Как видим из калибровочной кривой, она предсказывает

- вероятности 40%-50% быть классом 1 для кластера, где почти все объекты имеют класс 1,
- вероятности 50%-60% быть классом 1 для кластера, где почти все объекты имеют класс 0.

In [15]:

```
1 draw_prob_predictions(lr, X_test, name='LogisticRegression')
```

executed in 327ms, finished 16:27:09 2022-11-29



Попробуем добавить полиномы всех степей до 5-й включительно в качестве признаков и обучить на них логистическую регрессию

In [16]:

```
1 poly_lr = make_pipeline(
2     PolynomialFeatures(5),
3     LogisticRegression()
4 ).fit(
5     X_train, y_train
6 )
```

executed in 259ms, finished 16:27:10 2022-11-29

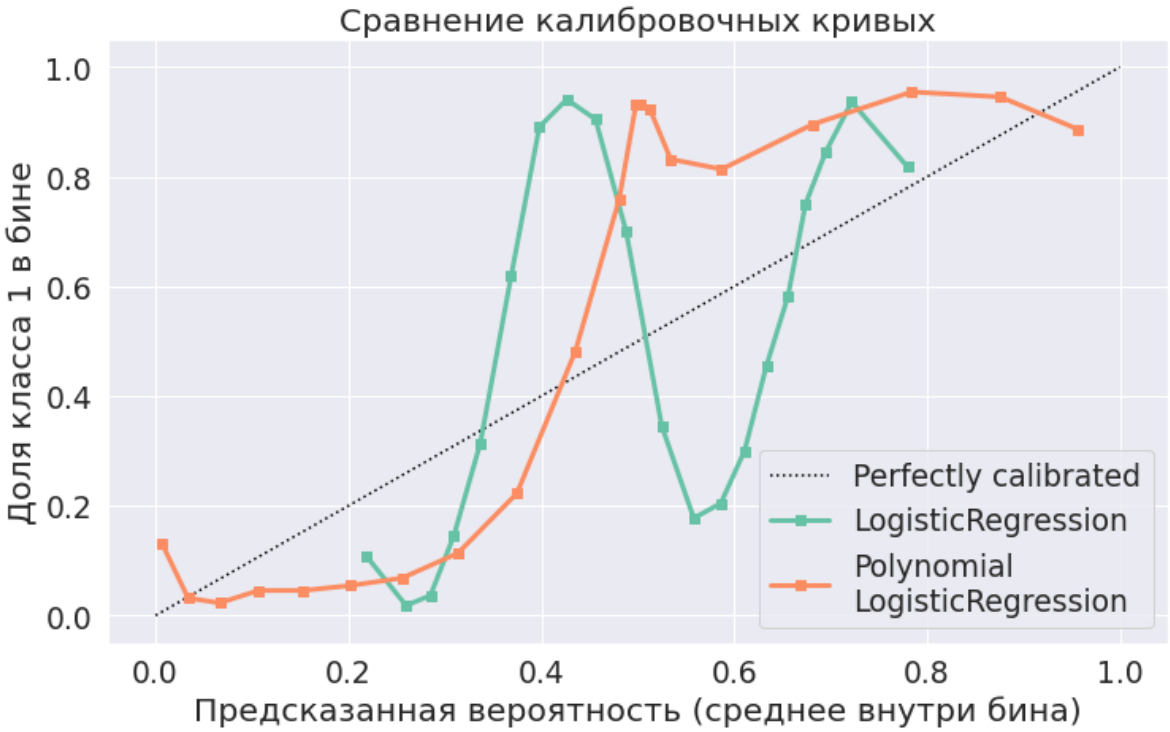
Сравним результаты

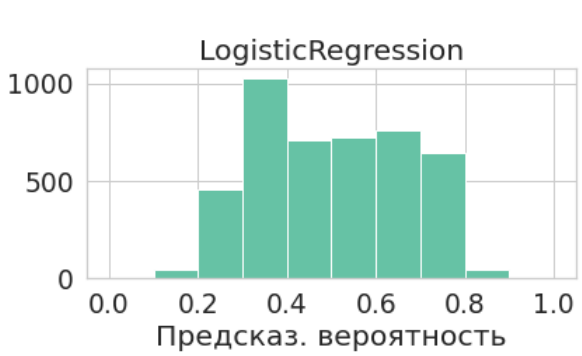
In [17]:

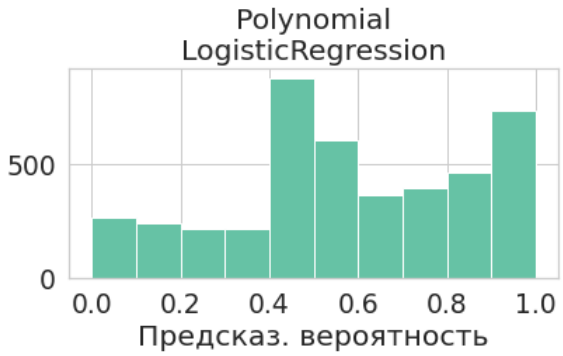
```
clf_list = [
    (lr, 'LogisticRegression'),
    (poly_lr, 'Polynomial\nLogisticRegression'),
]

calibration_curves(clf_list)
draw_probs_hist(clf_list, X_test)
```

executed in 530ms, finished 16:27:10 2022-11-29





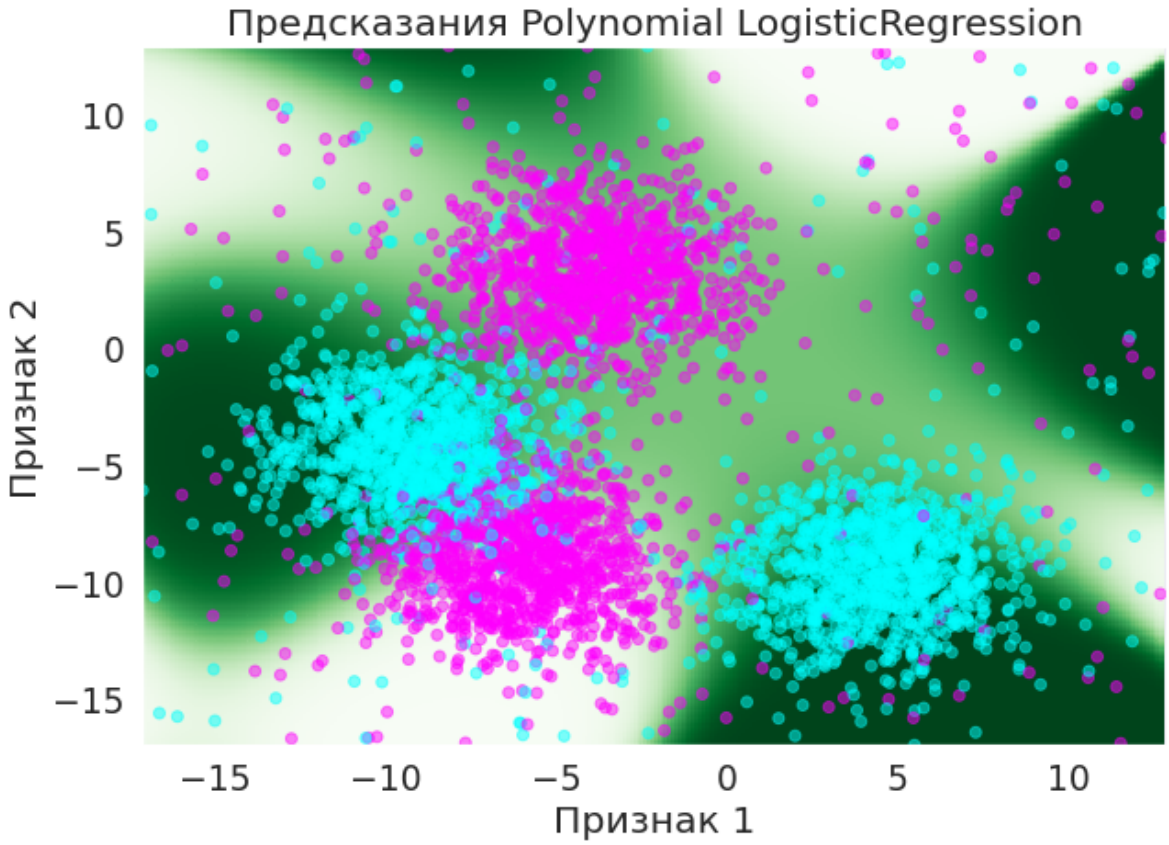


Такая логистическая регрессия скорее лучше справляется с классификацией, но предсказания вероятностей все равно не откалиброваны

In [18]:

```
1 draw_prob_predictions(poly_lr, X_test, name='Polynomial LogisticRegression')
```

executed in 378ms, finished 16:27:11 2022-11-29



Случайный лес

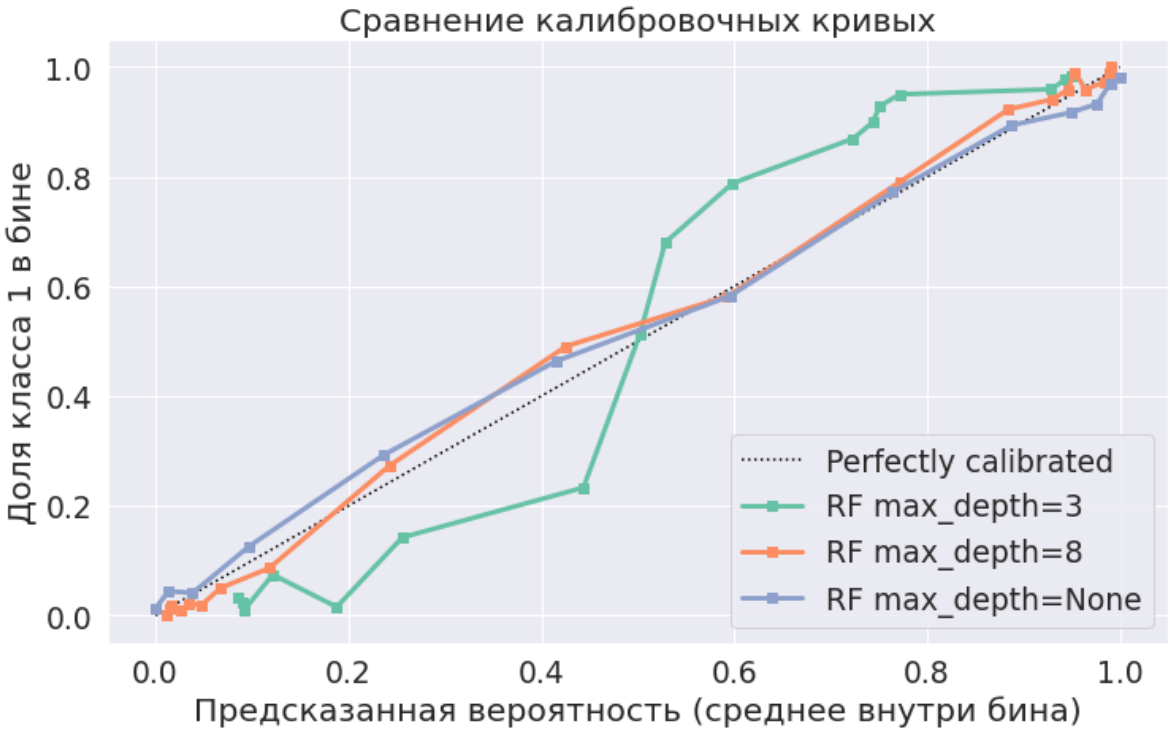
Обучим случайный лес с ограничением глубины деревьев и без ограничения. Посмотрим на калибровочные кривые

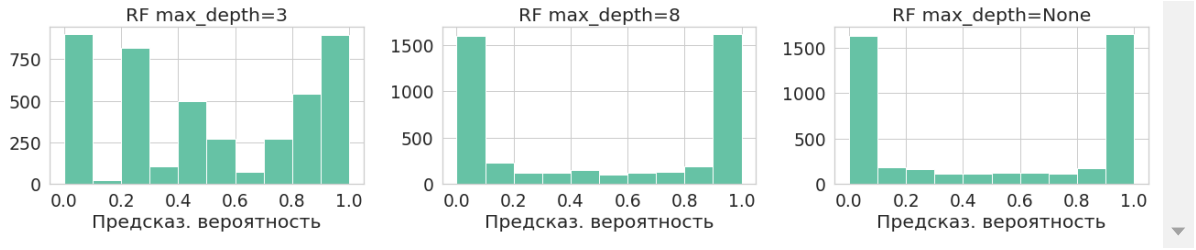
In [19]:

```
clf_list = [
    (
        RandomForestClassifier(
            max_depth=d, random_state=42
        ).fit(
            X_train, y_train
        ),
        f'RF max_depth={d}'
    )
    for d in [3, 8, None]
]

calibration_curves(clf_list)
draw_probs_hist(clf_list, X_test)
```

executed in 4.22s, finished 16:27:15 2022-11-29





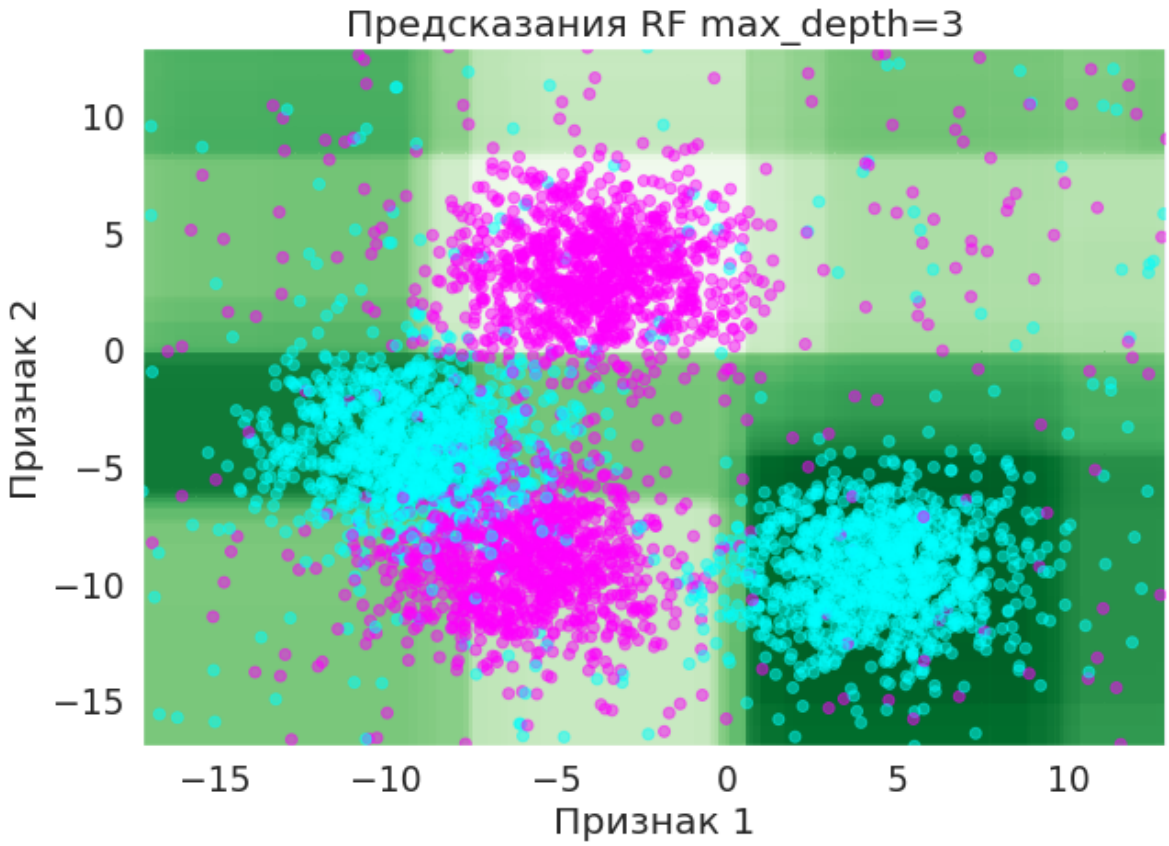
Как видим, если ограничивать глубину деревьев, случайный лес не может построить уверенное предсказание вероятностей. У него все предсказания вероятностей смещены ближе к 50%. Например, из калибровочной кривой можем сделать, что из всех объектов, для которых лес предсказывает вероятность класса 1 около 60% реально имеется более 70% объектов класса 1.

По графику предсказаний видим, что модель недообучилась

In [20]:

```
1 draw_prob_predictions(clf_list[0][0], X_test, name='RF max_depth=3')
```

executed in 624ms, finished 16:27:15 2022-11-29

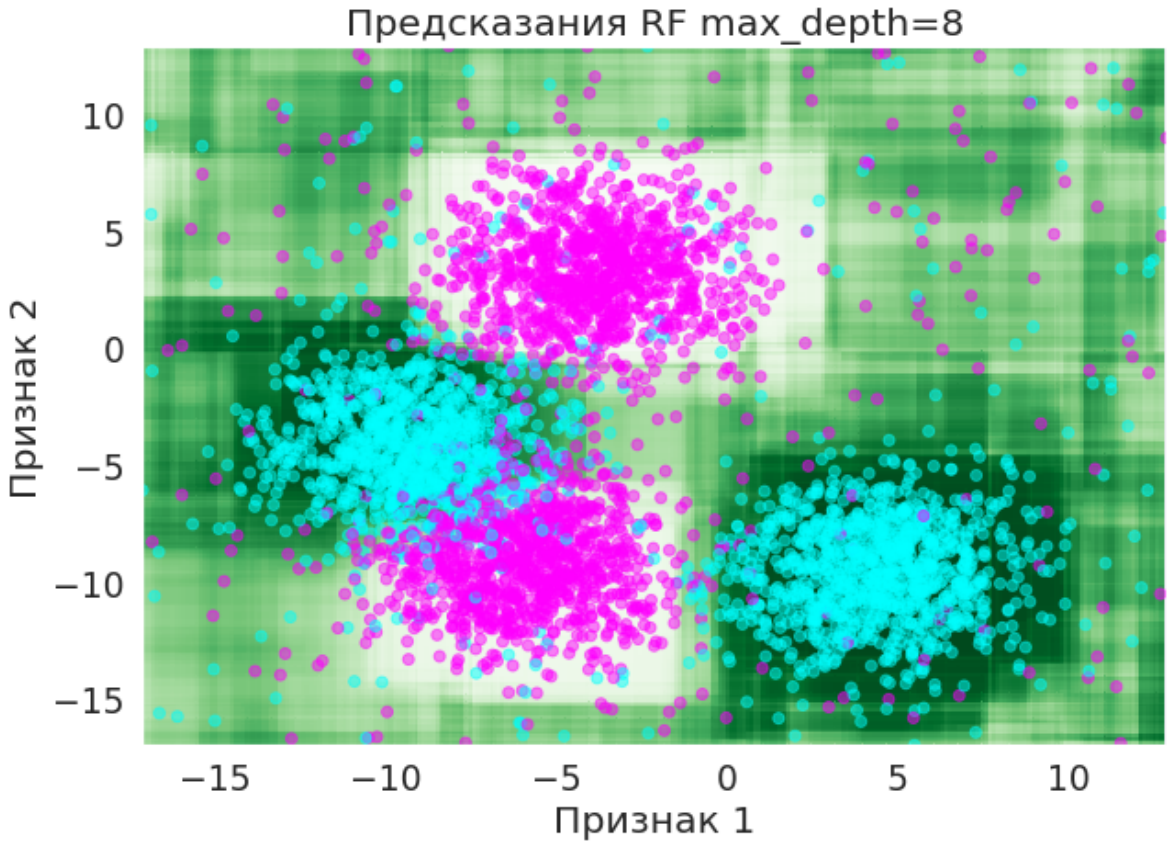


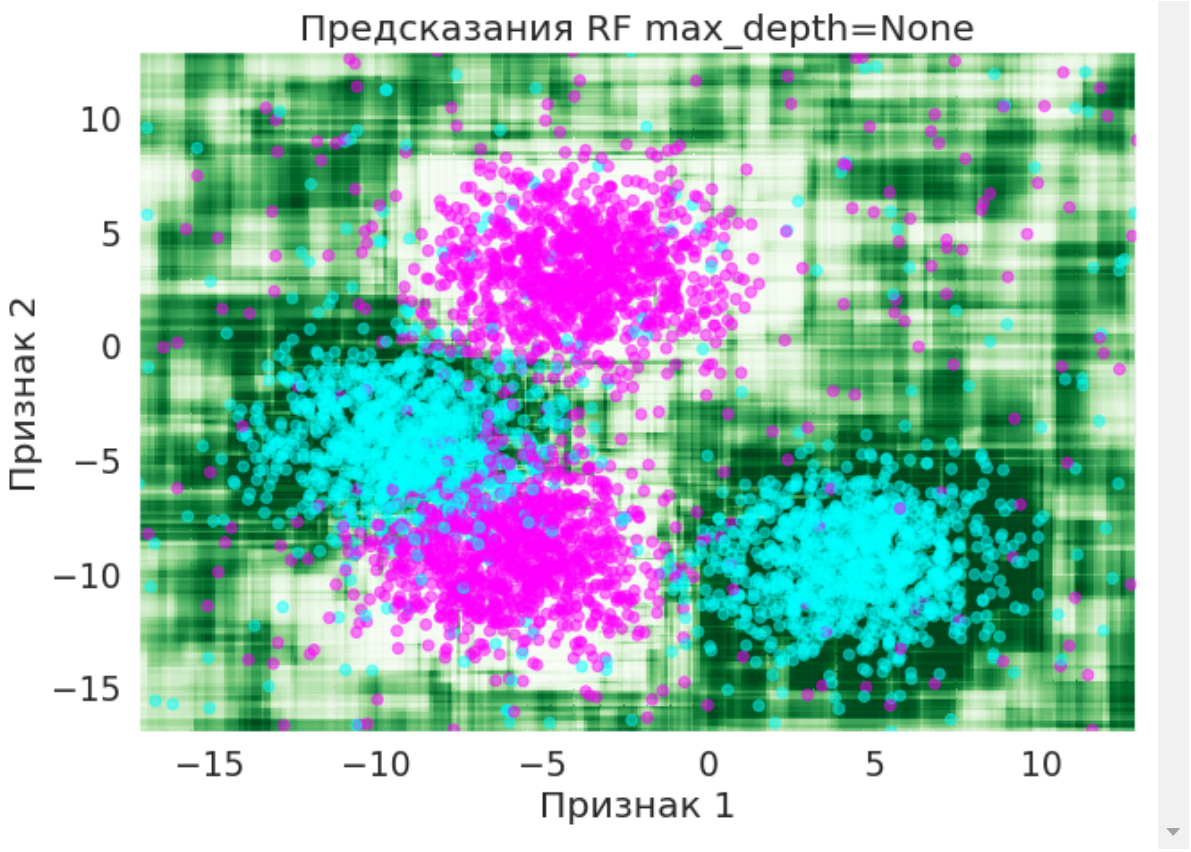
Можно сравнить с предсказанием для большей глубины и без ограничения глубины

In [21]:

```
1 draw_prob_predictions(clf_list[1][0], X_test, name='RF max_depth=8')
2 draw_prob_predictions(clf_list[2][0], X_test, name='RF max_depth=None')
```

executed in 1.61s, finished 16:27:17 2022-11-29





Градиентный бустинг

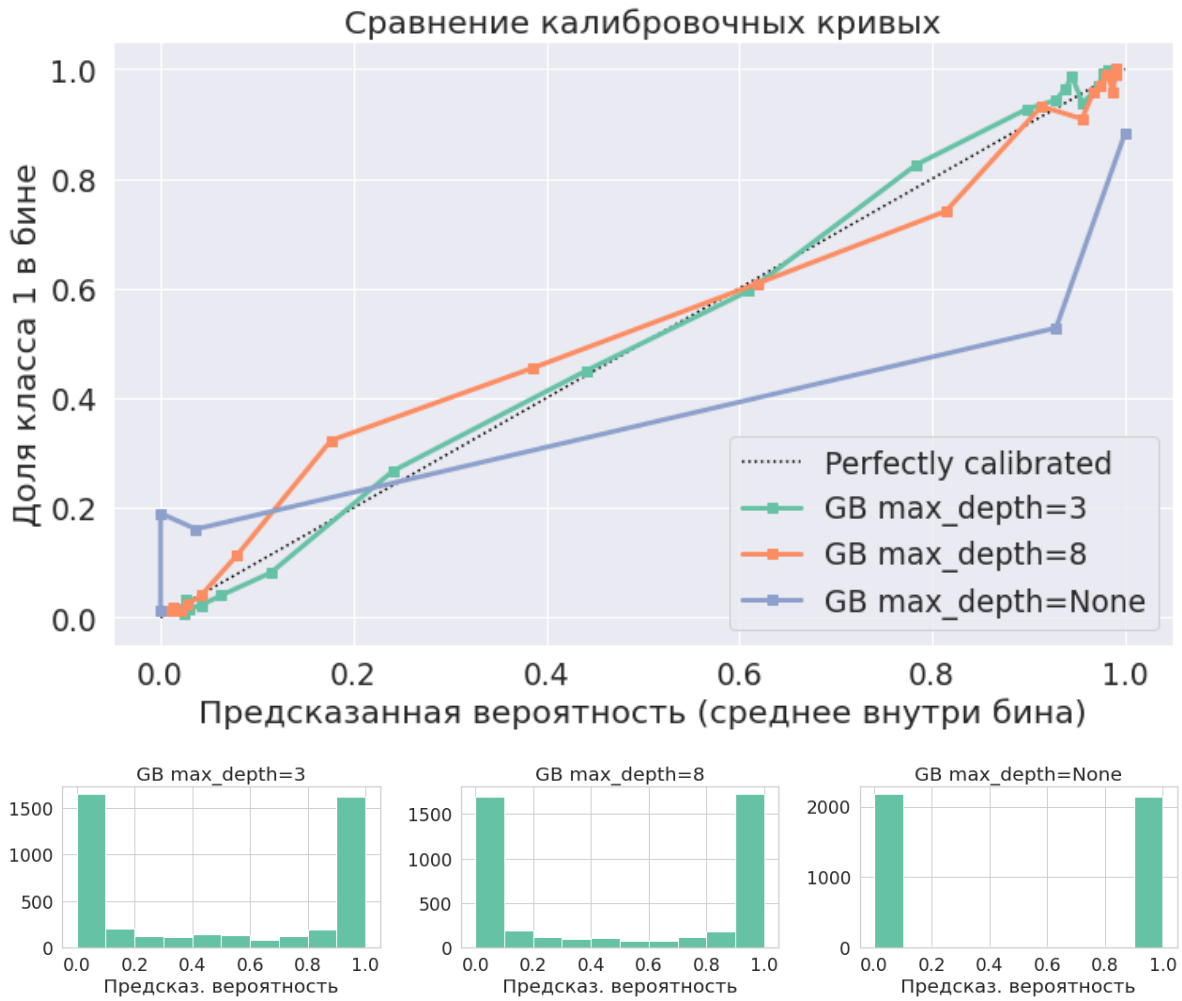
Посмотрим на аналогичные результаты по градиентному бустингу

In [22]:

```
clf_list = [
    (
        GradientBoostingClassifier(
            max_depth=d, random_state=42
        ).fit(
            X_train, y_train
        ),
        f'GB max_depth={d}'
    )
    for d in [3, 8, None]
]

calibration_curves(clf_list)
draw_probs_hist(clf_list, X_test)
```

executed in 13.6s, finished 16:27:31 2022-11-29

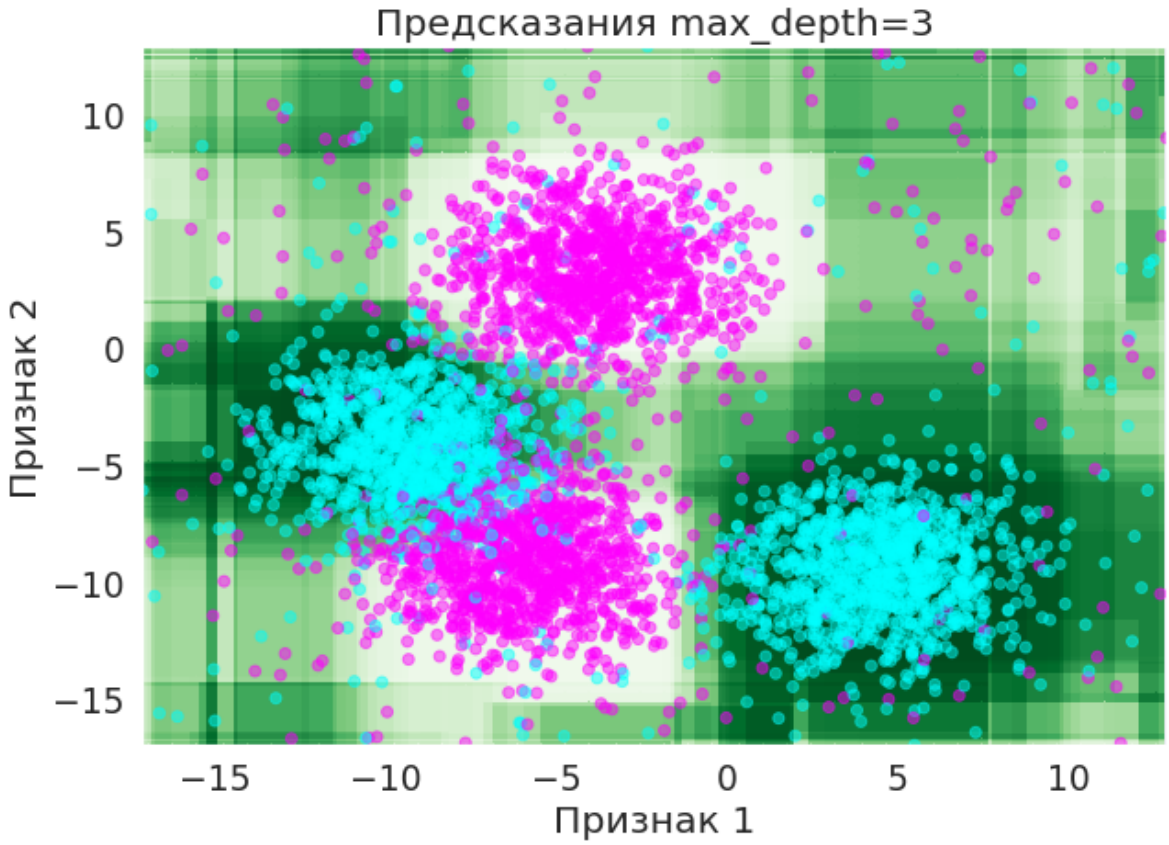


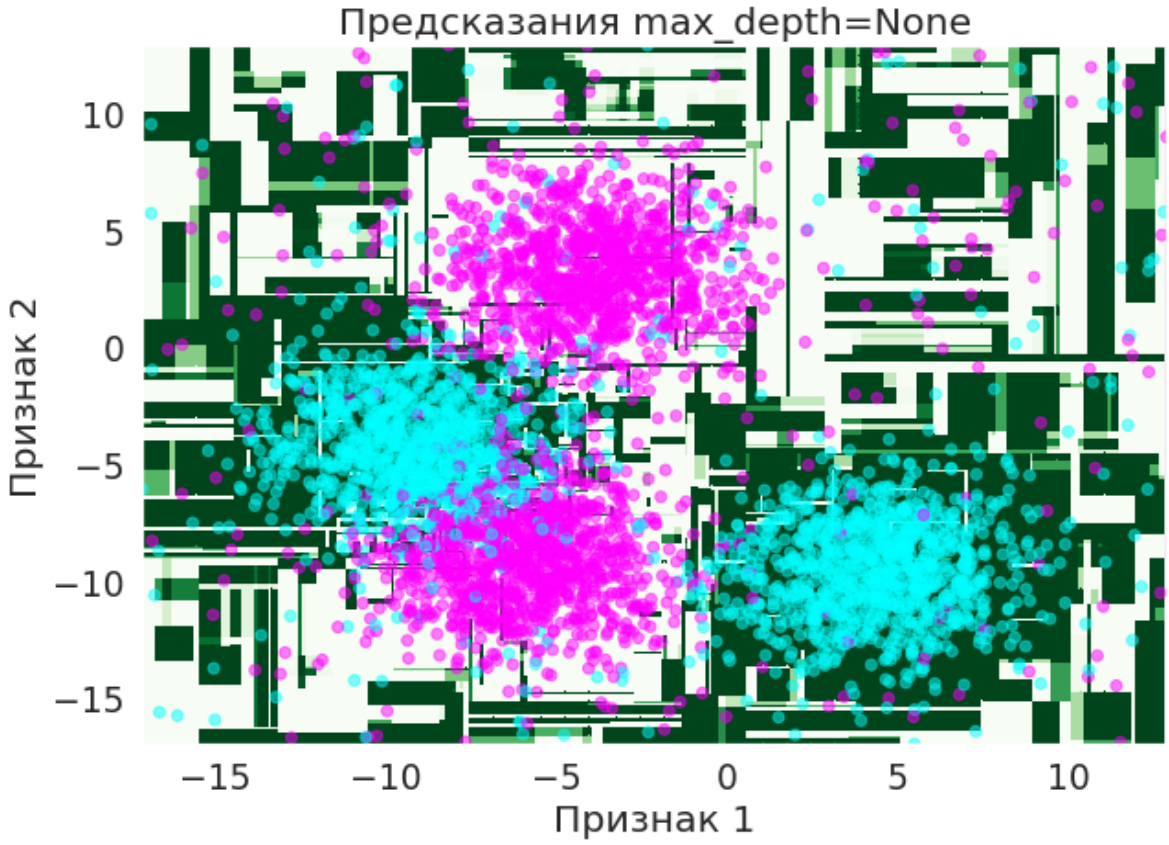
Как мы знаем, градиентный бустинг пытается уменьшить смещение при относительно небольшом разбросе, и поэтому для него нужны неглубокие деревья. Поэтому здесь мы видим обратную картину: лучше всего с предсказаниями справляется классификатор с меньшей глубиной дерева. Наоборот, при отсутствии ограничений на глубину деревьев бустинг начинает классифицировать слишком уверенно. Например, как видно из калибровочной кривой, для объектов, с предсказанием вероятностей класса 1 около 85% (хотя таких мало) реальная доля объектов класса 1 составляет 60%.

In [23]:

```
1 draw_prob_predictions(clf_list[0][0], X_test, name='max_depth=3')
2 draw_prob_predictions(clf_list[2][0], X_test, name='max_depth=None')
```

executed in 958ms, finished 16:27:32 2022-11-29





2. Методы калибровки

Калибровку вероятностей методами изотонической регрессии и Платта можно выполнить с помощью класса

`sklearn.calibration.CalibratedClassifierCV(base_estimator=None, *, method='sigmoid', cv=None, n_jobs=None, ensemble=True)`

Основные аргументы:

- `base_estimator` — классификатор, предсказания которого следует откалибровать;
- `method` — методы калибровки
  - `‘sigmoid’` — метод Платта,
  - `‘isotonic’` — изотоническая регрессия;
- `cv` — стратегия кросс-валидации для обучения и калибровки; если равно `“prefit”`, то считается, что модель уже обучена, и вся новая выборка идет на калибровку.

Методы:

- `fit(X, y[, sample_weight])` — обучение модели;
- `predict(X)` — предсказания;
- `predict_proba(X)` — откалиброванные предсказания вероятностей.

2.1 Случайный лес

Проведем калибровку случайного леса двумя методами

In [24]:

```
rf = RandomForestClassifier(
    max_depth=3, random_state=42
).fit(
    X_train, y_train
)

rf_sigmoid = CalibratedClassifierCV(
    base_estimator=rf,
    cv="prefit",
    method='sigmoid'
).fit(
    X_calib, y_calib
)

rf_isotonic = CalibratedClassifierCV(
    base_estimator=rf,
    cv="prefit",
    method='isotonic'
).fit(
    X_calib, y_calib
)
```

executed in 659ms, finished 16:27:32 2022-11-29

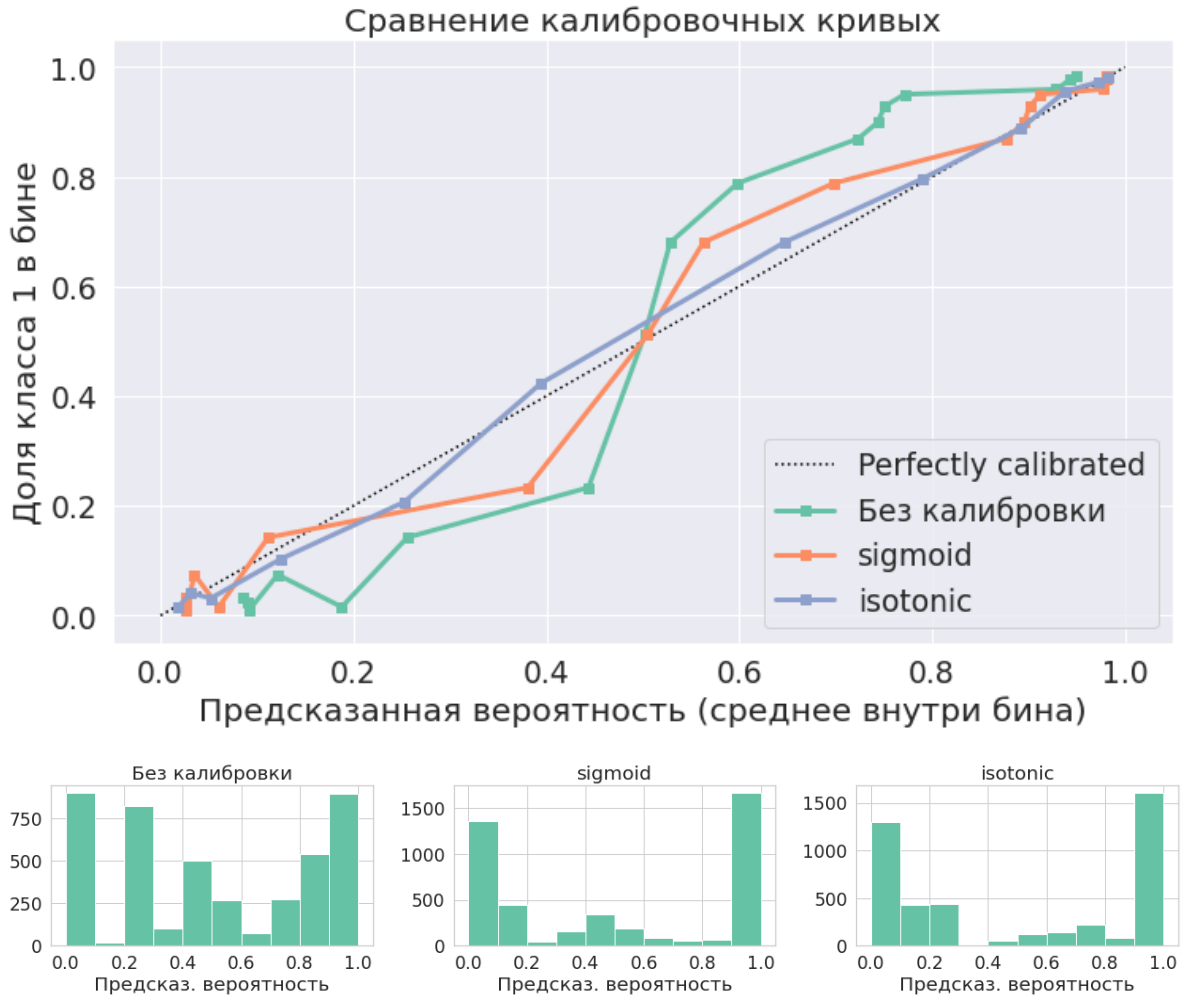
Сравним их со случаем отсутствия калибровки

In [25]:

```
clf_list = [
    (rf, 'Без калибровки'),
    (rf_sigmoid, 'sigmoid'),
    (rf_isotonic, 'isotonic')
]

calibration_curves(clf_list)
draw_probs_hist(clf_list, X_test)
```

executed in 874ms, finished 16:27:33 2022-11-29



Видим, что калибровка методом изотонической регрессии лучше всего справилась. Из графика можем сказать, например, следующее

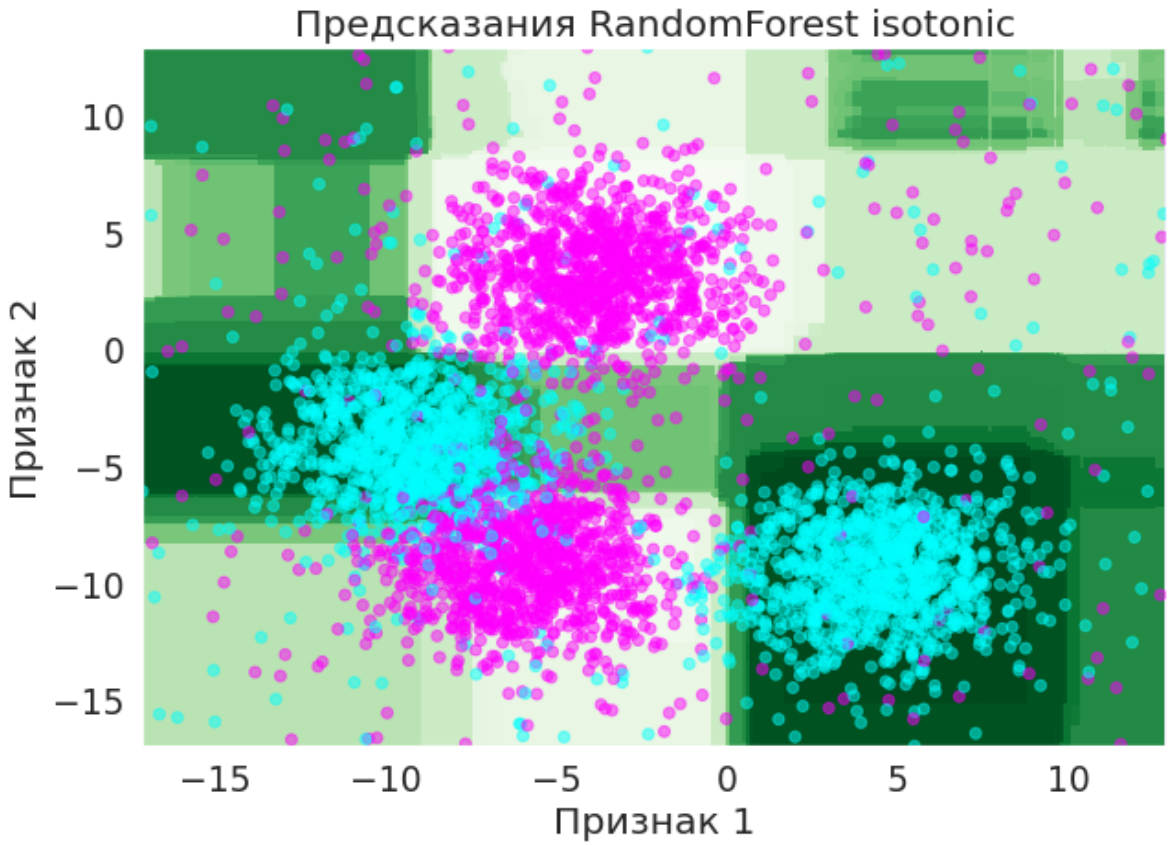
- среди тех объектов, для которых **случайный лес без калибровки** предсказывает вероятность около 40% быть классом 1, реальная доля объектов класса 1 оставляет **около 20%**;
- среди тех объектов, для которых **случайный лес с калибровкой Платта** предсказывает вероятность около 40% быть классом 1, реальная доля объектов класса 1 оставляет **около 25%**;
- среди тех объектов, для которых **случайный лес с калибровкой методом изотонической регрессии** предсказывает вероятность около 40% быть классом 1, реальная доля объектов класса 1 оставляет **около 40%**.

Посмотрим на график откалиброванных предсказаний вероятностей и сравним с исходными

In [26]:

```
1 draw_prob_predictions(rf_isotonic, X_test, name='RandomForest isotonic')
```

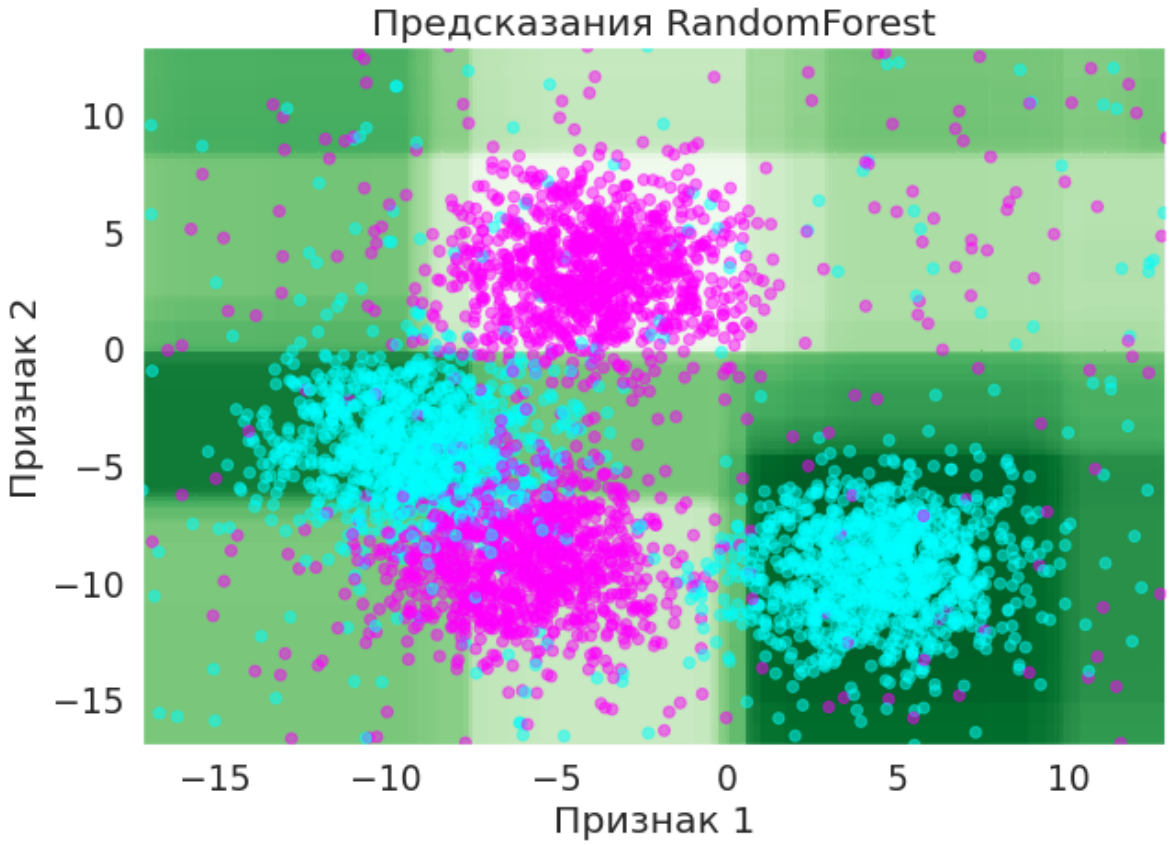
executed in 641ms, finished 16:27:34 2022-11-29



In [27]:

```
1  draw_prob_predictions(rf, X_test, name='RandomForest')
```

executed in 639ms, finished 16:27:34 2022-11-29



2.2 Логистическая регрессия

Проведем калибровку логистической регрессии двумя методами

In [28]:

```
lr_sigmoid = CalibratedClassifierCV(
    base_estimator=lr,
    cv="prefit",
    method='sigmoid'
).fit(
    X_calib, y_calib
)

lr_isotonic = CalibratedClassifierCV(
    base_estimator=lr,
    cv="prefit",
    method='isotonic'
).fit(
    X_calib, y_calib
)
```

executed in 8ms, finished 16:27:34 2022-11-29

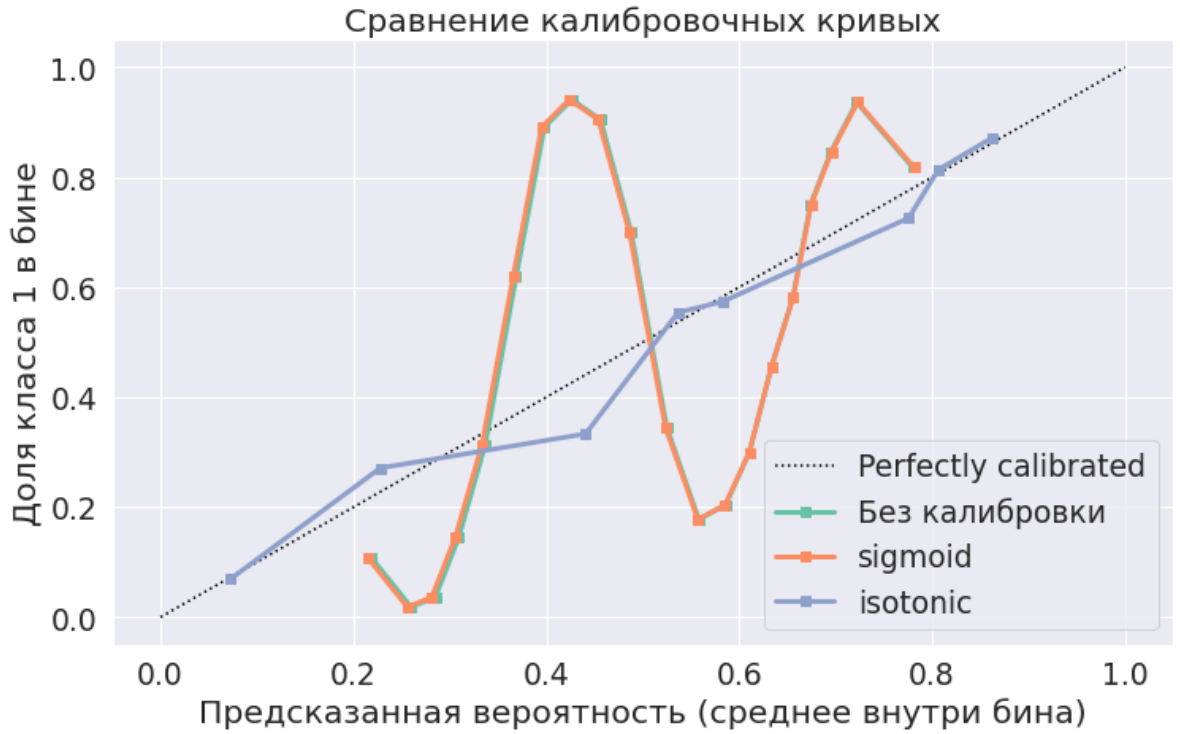
Сравним их со случаем отсутствия калибровки

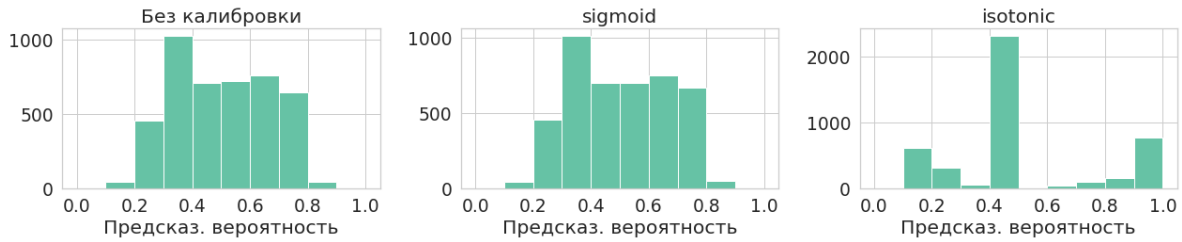
In [29]:

```
clf_list = [
    (lr, 'Без калибровки'),
    (lr_sigmoid, 'sigmoid'),
    (lr_isotonic, 'isotonic')
]

calibration_curves(clf_list)
draw_probs_hist(clf_list, X_test)
```

executed in 756ms, finished 16:27:35 2022-11-29





Видим, что даже в этом случае калибровка методом изотонической регрессии очень хорошо справилась. Калибровка методом Платта почти не меняет результат, что логично, ведь предсказания вероятностей сами по себе являются сигмоидой.

Из графика можем сказать, например, следующее

- среди тех объектов, для которых **логистическая регрессия без калибровки** предсказывает вероятность около 60% быть классом 1, реальная доля объектов класса 1 оставляет **около 20%**;
- среди тех объектов, для которых **логистическая регрессия с калибровкой Платта** предсказывает вероятность около 60% быть классом 1, реальная доля объектов класса 1 оставляет **около 20%**;

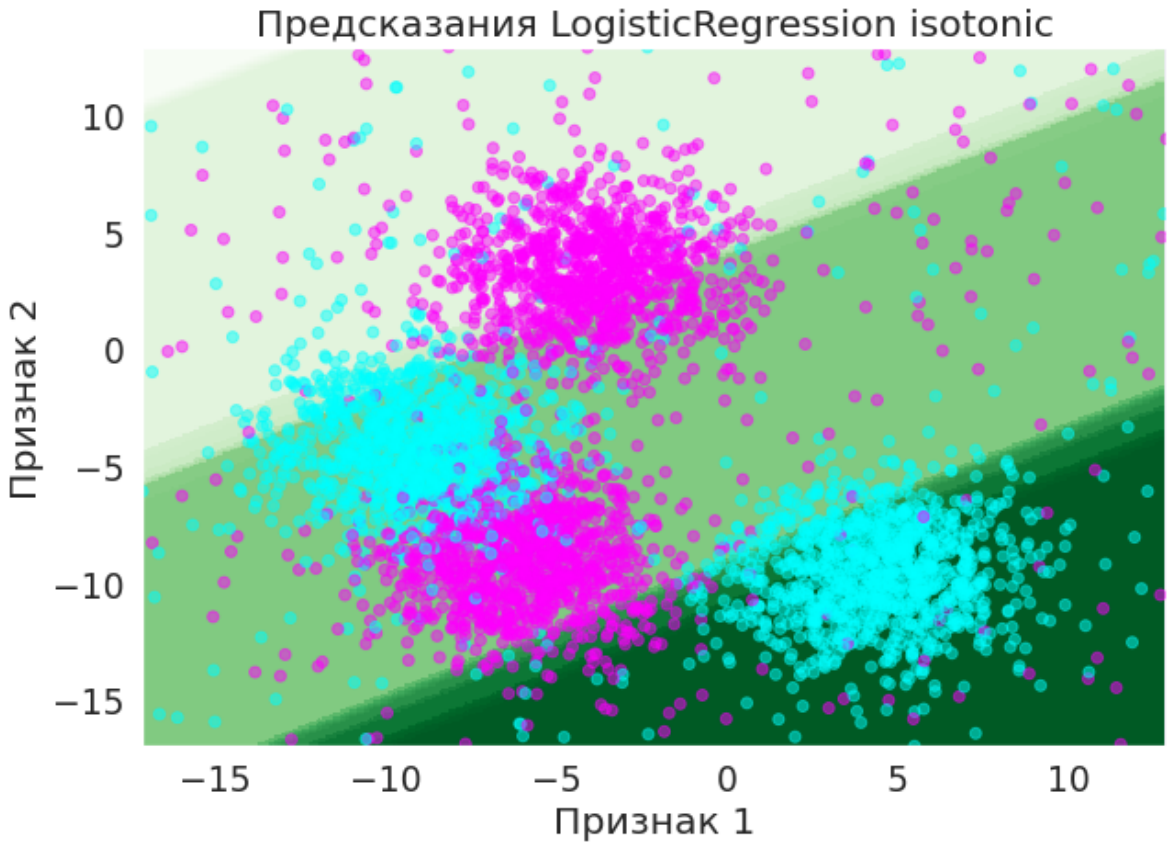
- среди тех объектов, для которых **логистическая регрессия с калибровкой методом изотонической регрессии** предсказывает вероятность около 60% быть классом 1, реальная доля объектов класса 1 оставляет **около 60%**.

Посмотрим на график откалиброванных предсказаний вероятностей и сравним с исходными

In [30]:

```
draw_prob_predictions(lr_isotonic, X_test, name='LogisticRegression isotonic')
```

executed in 350ms, finished 16:27:36 2022-11-29



Посмотрим также, как калибровка справится с логистической регрессией на полиномиальных признаках

In [31]:

```
poly_lr_sigmoid = CalibratedClassifierCV(
    base_estimator=poly_lr,
    cv="prefit",
    method='sigmoid'
).fit(
    X_calib, y_calib
)

poly_lr_isotonic = CalibratedClassifierCV(
    base_estimator=poly_lr,
    cv="prefit",
    method='isotonic'
).fit(
    X_calib, y_calib
)
```

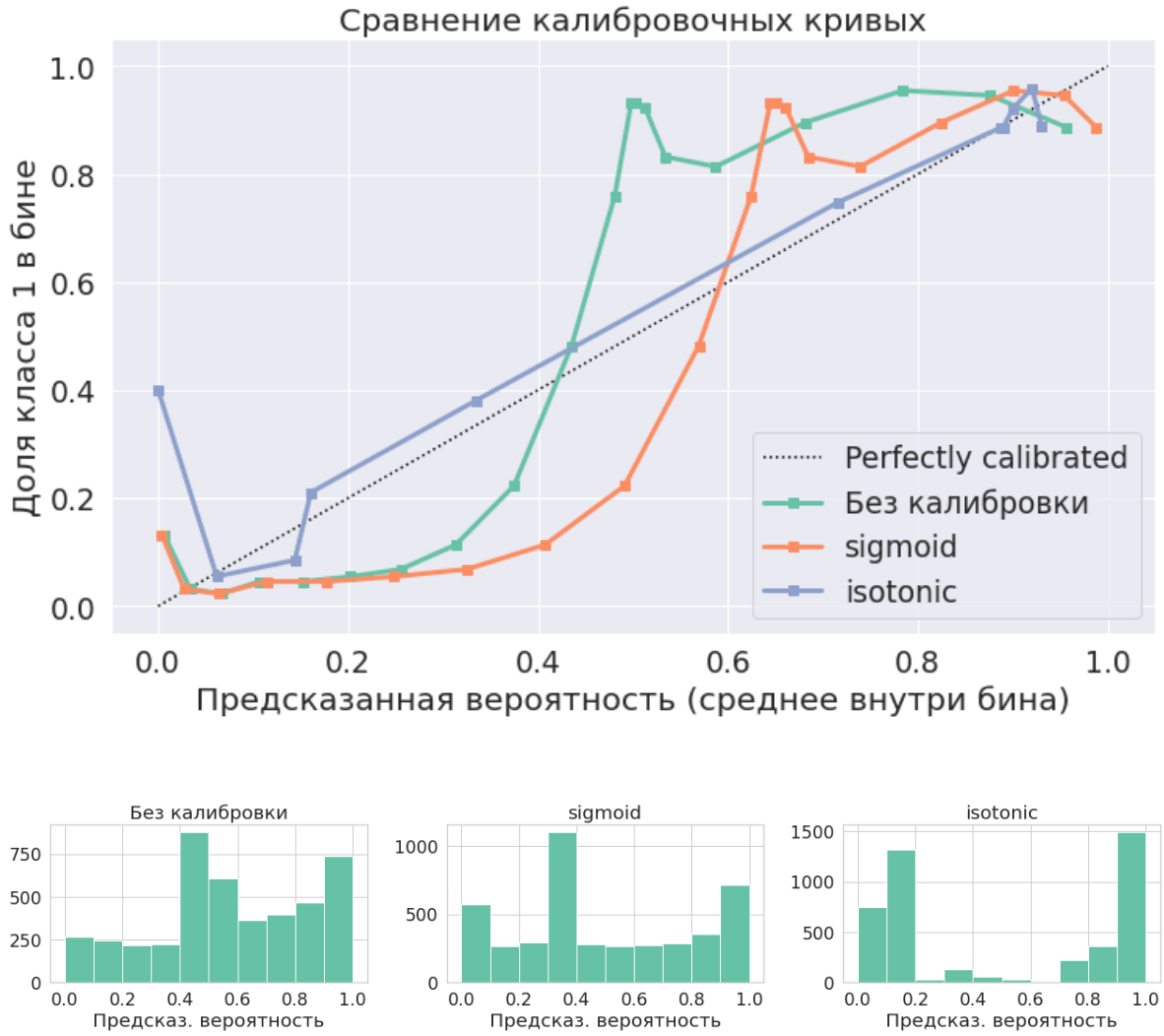
executed in 20ms, finished 16:27:36 2022-11-29

In [32]:

```
clf_list = [
    (poly_lr, 'Без калибровки'),
    (poly_lr_sigmoid, 'sigmoid'),
    (poly_lr_isotonic, 'isotonic')
]

calibration_curves(clf_list)
draw_probs_hist(clf_list, X_test)
```

executed in 744ms, finished 16:27:36 2022-11-29



Как и раньше, изотоническая регрессия справилась достаточно неплохо. Калибровка методом Платта позволяет добавить некоторое смещение.

Посмотрим на откалиброванные предсказания вероятностей методом изотонической регрессии

In [33]:

```
1 draw_prob_predictions(poly_lr_isotonic, X_test, name='Polynomial LogisticRegre
```

executed in 349ms, finished 16:27:37 2022-11-29