Случайный лес

Цель этого ноутбука — знакомство со случайными лесами, с их параметрами и свойствами. В ноутбуке будут рассмотрены примеры применения случайного леса для решения задач классификации и регрессии.

In [1]:

```
import copy
import random
import warnings

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

from matplotlib import pyplot as plt
import seaborn as sns
from tqdm.notebook import tqdm

from sklearn import datasets
from sklearn.metrics import accuracy_score, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

warnings.simplefilter("ignore", DeprecationWarning)
sns.set(context='poster')
%matplotlib inline

np.random.seed(42)
```

Основные параметры

Реализации: `RandomForestClassifier`, `RandomForestRegressor`

Набор гиперпараметров случайного леса очень похож на набор гиперпараметров решающего дерева. Основным отличием является наличие у случайного леса параметра `n_estimators`, задающего количество решающих деревьев, используемых для получения предсказаний. Это **основной гиперпараметр** для случайного леса.

Напомним главные гиперпараметры решающего дерева, которые также имеются у случайного леса.

- `criterion` — критерий информативности, по которому происходит разбиение вершины дерева.
- `max_depth` — ограничение на глубину каждого дерева в лесе.
- `min_samples_split` — минимальное количество элементов обучающей выборки в вершине дерева, чтобы её можно было разбивать.
- `min_samples_leaf` — минимальное количество элементов обучающей выборке в листовой вершине.

- `splitter` — способ разбиения вершины каждого решающего дерева. Есть 2 возможных варианта: `best` и `random`. В первом случае рассматриваются все возможные способы разбить вершину дерева на две и берётся тот из них, значение критерия для которого оптимально. При `splitter=random` берётся несколько случайных возможных разбиений и среди них выбирается то, значение критерия для которого оптимально.
- `max_features` — максимальное количество признаков, которые могут быть перебраны при разбиении вершины дерева. Перед каждым разбиением дерева генерируется выборка из `min(k, max_features)` случайных признаков (`k` — количество признаков в датасете) и только эти признаки рассматриваются как разделяющие в данной вершине.
  Этот признак может принимать
  - целочисленное значение — число признаков,
  - вещественное значение — доля признаков,
  - `None` — все признаки,
  - `"auto"` — квадратный корень от числа всех признаков (по умолчанию),
  - `"sqrt"` — квадратный корень от числа всех признаков,
  - `"log2"` — двоичный логарифм от числа всех признаков.
- `min_impurity_decrease` — минимальное значение уменьшения взвешенного критерия неопределенности (`impurity`), чтобы можно было разбить выборку в данной вершине.

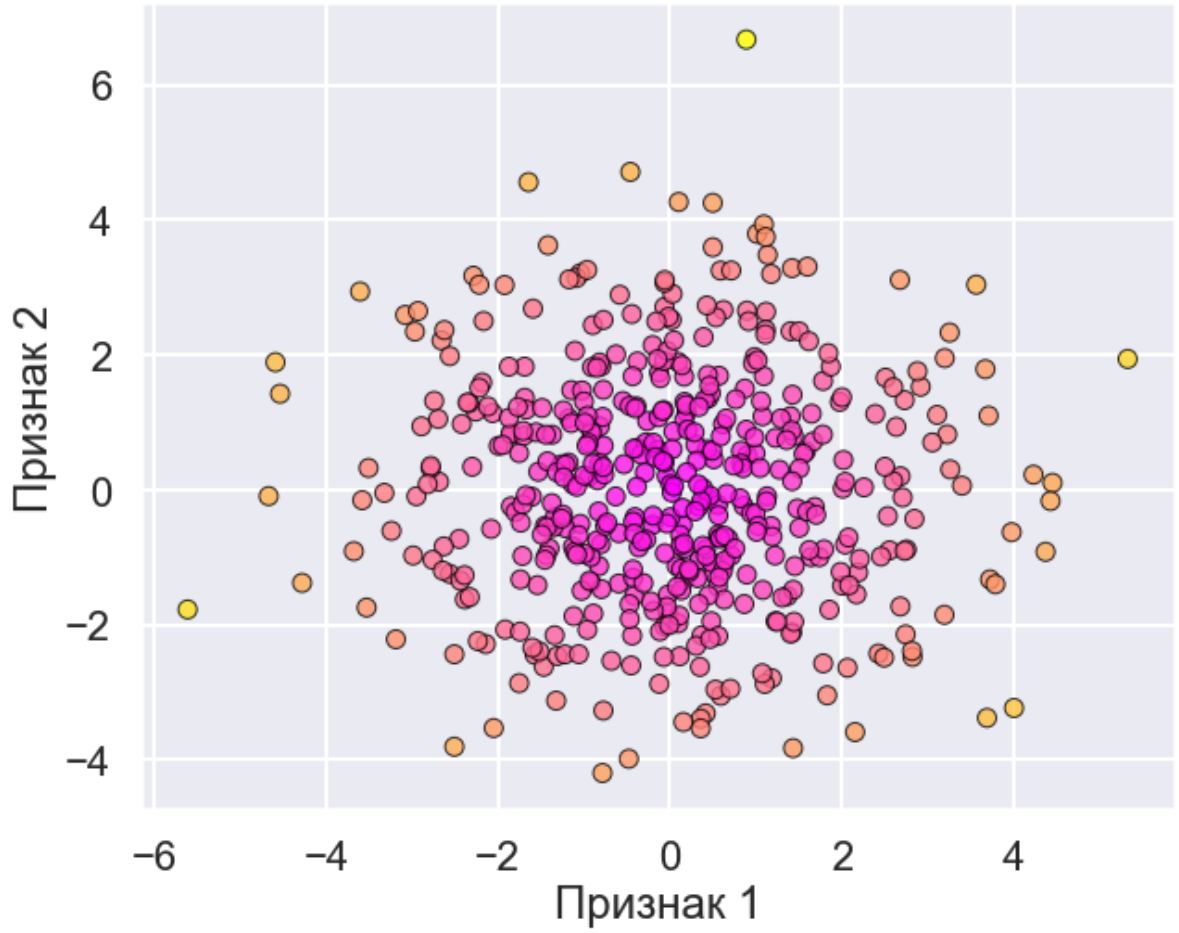
О других гиперпараметрах случайного леса можно почитать в документации (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html).

1. Решение задачи регрессии с помощью Random Forest

Сгенерируем выборку из многомерного нормального распределения и в качестве целевой функции возьмем *расстояние от точки до центра координат*.

In [2]:

```
X_train = sps.multivariate_normal(mean=[0, 0], cov=[[3, 0], [0, 3]]).rvs(size=500)
y_train = (X_train[:, 0] ** 2 + X_train[:, 1] ** 2) ** 0.5

plt.figure(figsize=(10, 8))
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='spring',
            s=100, alpha=0.8, linewidths=1, edgecolors='black')
plt.xlabel('Признак 1'), plt.ylabel('Признак 2');
plt.show()
```



In [3]:

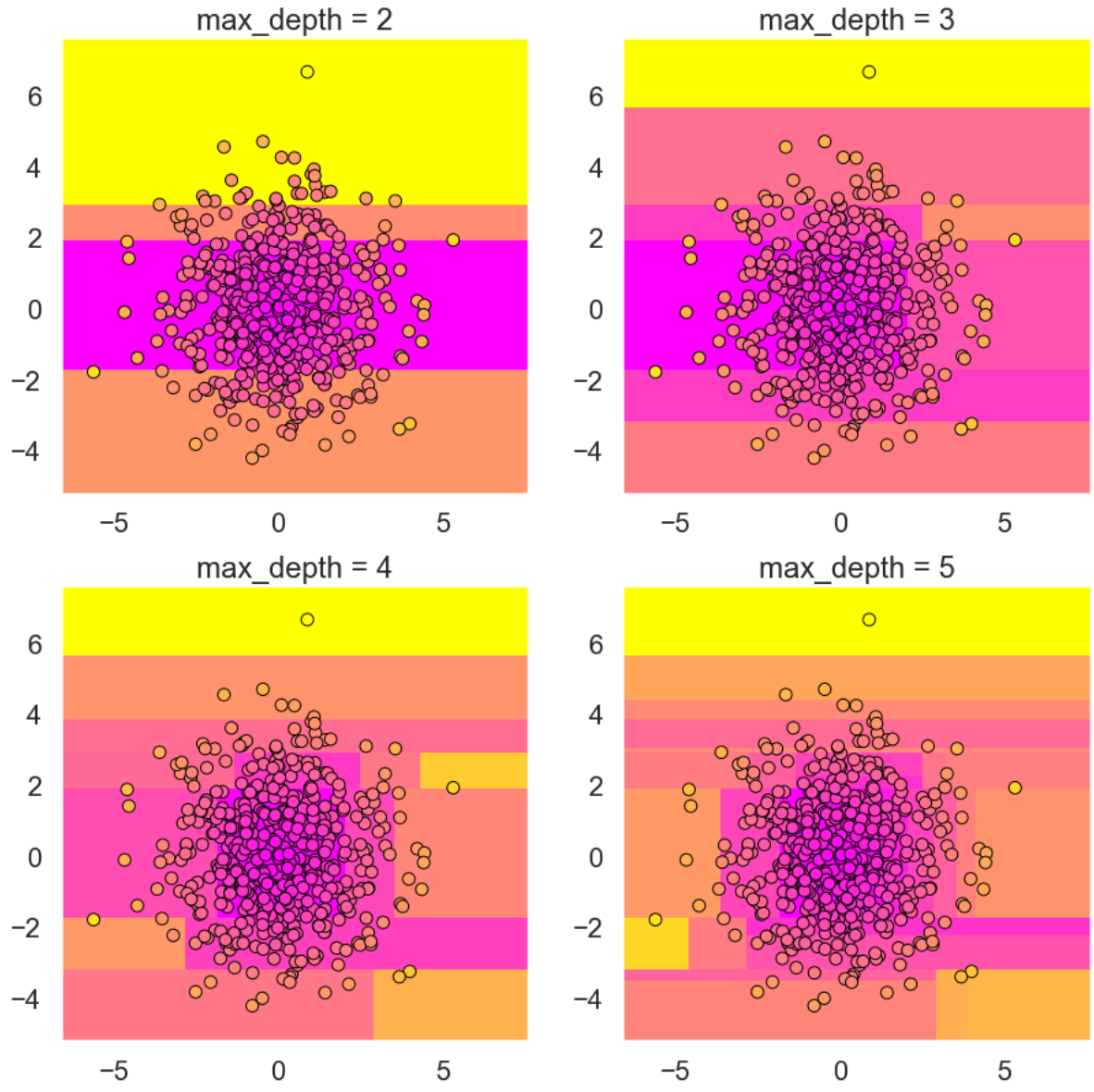
```
def generate_grid(sample, border=1, step=0.05):
    ''' Создает сетку на основе выборки для раскраски пространства '''

    return np.meshgrid(np.arange(min(sample[:, 0]) - border,
                                 max(sample[:, 1]) + border,
                                 step),
                       np.arange(min(sample[:, 1]) - border,
                                 max(sample[:, 1]) + border,
                                 step))


def create_picture(X_train, y_train, model, border=1, step=0.05,
                   cmap='spring', alpha=1, create_new_figure=True,
                   figsize=(7, 7), s=100, linewidths=1, points=True):
    '''
    Раскрашивает пространство в соответствии с предсказаниями
    случайного леса/решающего дерева

    Параметры.
    1) X_train — данные обучающей выборки,
    2) y_train — метки обучающей выборки,
    3) model — визуализируемая модель,
    4) border — размер границы между областями пространства,
       полученными моделью,
    5) step — точность сетки пространства,
    6) cmap — цветовая схема,
    7) aplha — прозрачность точек обучающей выборки,
    8) create_new_figure — флаг, определяющий создавать ли
       новую фигуру,
    9) figsize — размер создаваемой фигуры,
    10) s — размер точек обучающей выборки,
    11) linewidths — размер границы каждой точки,
    12) point — флаг, определяющий, отображать ли точки
        обучающей выборки на графике.
    '''

    # Создание сетки
    grid = generate_grid(X_train, border, step)
    # Выворачивание сетки для применения модели
    grid_ravel = np.c_[grid[0].ravel(), grid[1].ravel()]

    # Предсказание значений для сетки
    grid_predicted_ravel = model.predict(grid_ravel)
    grid_predicted = grid_predicted_ravel.reshape(grid[0].shape)  # Подгоняем разме


    # Построение фигуры
    if create_new_figure:
        plt.figure(figsize=figsize)

    plt.pcolormesh(grid[0], grid[1], grid_predicted, cmap=cmap, shading='auto')
    if points:
        plt.scatter(
            X_train[:, 0], X_train[:, 1], c=y_train, s=s,
            alpha=alpha, cmap=cmap, linewidths=linewidths, edgecolors='black'
        )

    plt.xlim((min(grid_ravel[:, 0]), max(grid_ravel[:, 0])))
    plt.ylim((min(grid_ravel[:, 1]), max(grid_ravel[:, 1])))
```

```
    plt.title('max_depth = ' + str(model.get_params()['max_depth']))

    if create_new_figure:
        plt.show()
```

1.1. Визуализация предсказания в зависимости от максимальной глубины

In [4]:

```
plt.figure(figsize=(15, 15))
for i, max_depth in enumerate(range(2, 6)):
    plt.subplot(2, 2, i+1)
    create_picture(
        X_train, y_train,
        DecisionTreeRegressor(max_depth=max_depth).fit(X_train, y_train),
        create_new_figure=False
    )
```



1.2. Визуализация предсказаний случайного леса

Обучим случайный лес на 35 деревьев для удобства визуализации.

In [5]:

```
n_estimators = 35
model = RandomForestRegressor(n_estimators=n_estimators)
model.fit(X_train, y_train)
```

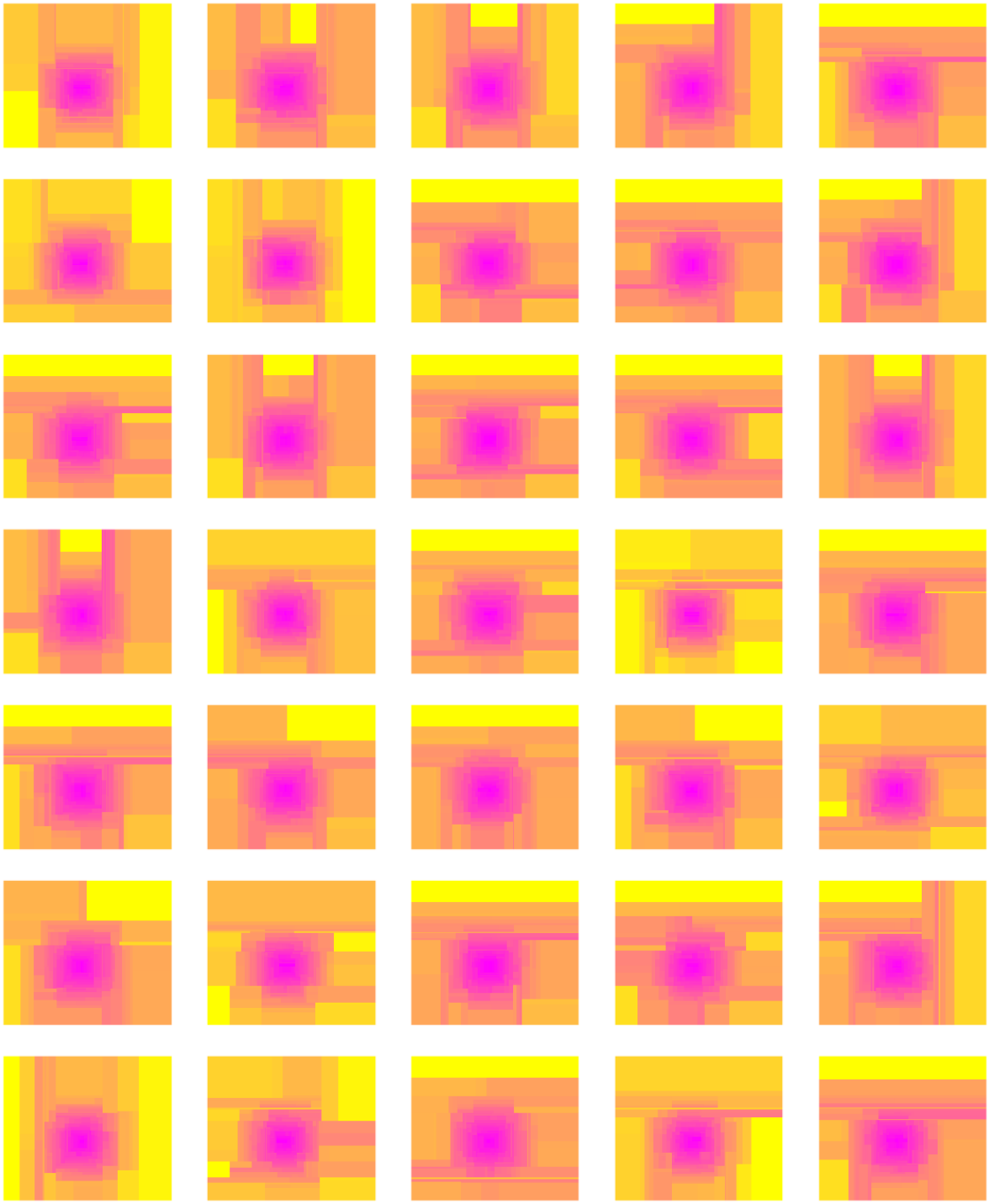
Out[5]:

```
RandomForestRegressor(n_estimators=35)
```

Визуализация предсказания по каждому из деревьев по отдельности. Они все такие разные и такие переобученные...

In [6]:

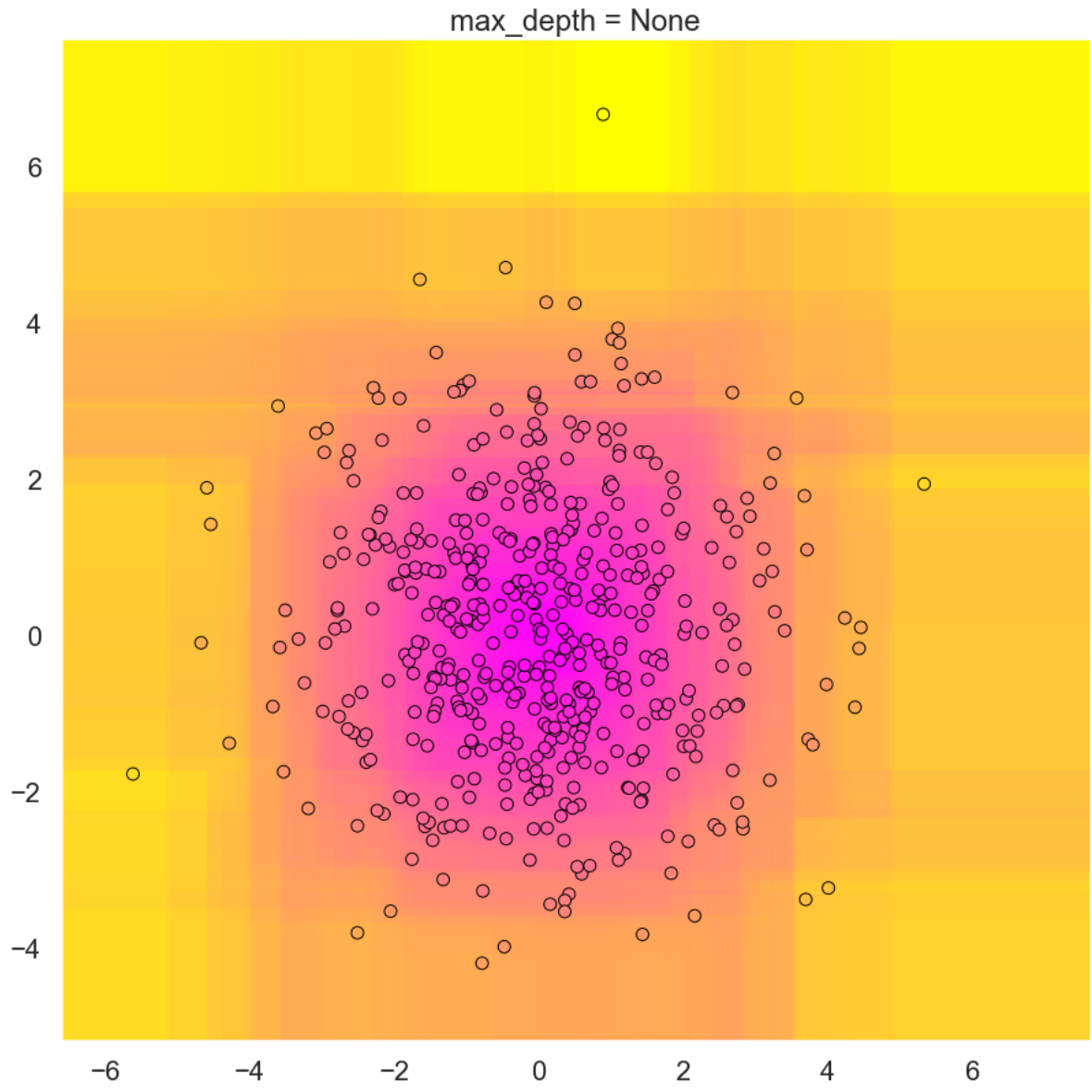
```
plt.figure(figsize=(20, 25))
for i in range(n_estimators):
    plt.subplot(int(np.floor(n_estimators / 5)), 5, i+1)
    create_picture(X_train, y_train, model.estimators_[i],
                   create_new_figure=False, s=20, linewidths=0, points=False)
    plt.title('')
    plt.xticks([]), plt.yticks([])
```



Усредненное предсказание.

In [7]:

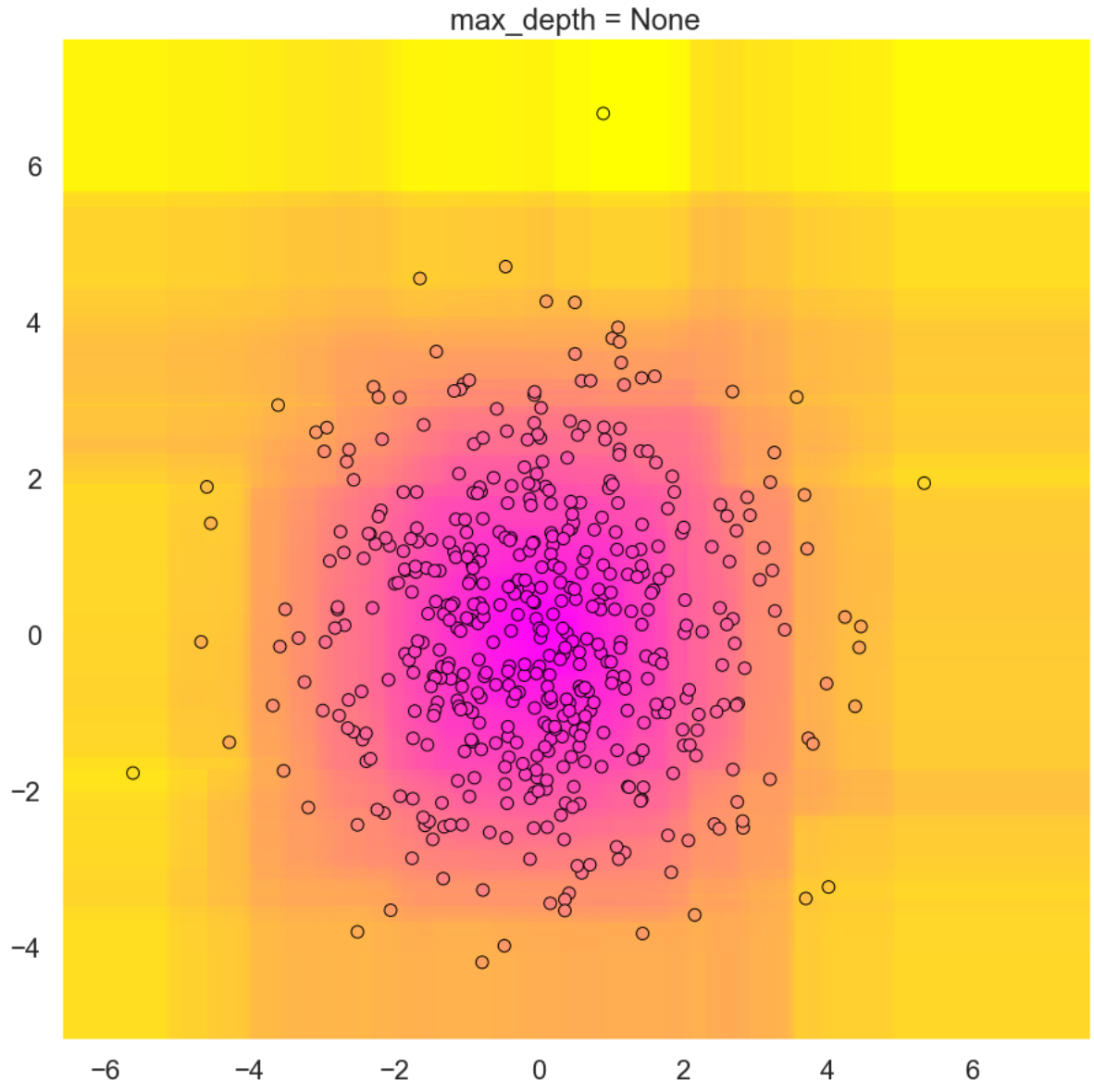
```
create_picture(X_train, y_train, model, figsize=(15, 15))
```



То же самое с 100 деревьями.

In [8]:

```
n_estimators = 100
model = RandomForestRegressor(n_estimators=n_estimators)
model.fit(X_train, y_train)

create_picture(X_train, y_train, model, figsize=(15, 15))
```



1.3. Зависимость качества предсказаний леса от значений гиперпараметров

Теперь исследуем зависимость качества предсказаний случайного леса от значений гиперпараметров. Рассмотрим датасет `diabetes` из `sklearn`. В нём исследуется численная оценка прогрессирования диабета у пациентов на основе таких признаков, как возраст, пол, масса тела, среднее кровяное давление и некоторых других. Для того, чтобы лучше понять, что из себя представляют признаки в этом датасете, можно обратиться к этой странице: https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html (https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html).

In [9]:

```
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

print('data shape:', X.shape)
print('target shape:', y.shape)
```

```
data shape: (442, 10)
target shape: (442,)
```

Как и в предыдущих экспериментах, разобьём данные на обучение и тест.

In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=42
)
```

Подберём оптимальные параметры для `RandomForestRegressor` по сетке.

In [11]:

```
tree_gridsearch = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid={
        'max_depth': [3, 5, None],
        'n_estimators': [5, 10, 25, 50],
        'min_samples_leaf': [1, 2, 5],
        'min_samples_split': [2, 5]
    }
)

tree_gridsearch.fit(X_train, y_train)
print(tree_gridsearch.best_params_)
```

```
{'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_est
imators': 50}
```

Посчитаем значение метрики `r2-score`.

In [12]:

```
print('train r2_score {:.4f}'.format(
    r2_score(y_train, tree_gridsearch.best_estimator_.predict(X_train))
))
print('test r2_score {:.4f}'.format(
    r2_score(y_test, tree_gridsearch.best_estimator_.predict(X_test))
))
```

```
train r2_score 0.6969
test r2_score 0.4751
```

Теперь попробуем резко увеличить значение `min_samples_leaf` .

In [13]:

```
regressor = RandomForestRegressor(
    random_state=42,
    min_samples_leaf=50,
    n_estimators=50,
    max_depth=5,
    min_samples_split=2
)
regressor.fit(X_train, y_train)

print('train r2_score {:.4f}'.format(
    r2_score(y_train, regressor.predict(X_train))
))
print('test r2_score {:.4f}'.format(
    r2_score(y_test, regressor.predict(X_test))
))
```

```
train r2_score 0.4273
test r2_score 0.4610
```

Вывод. Заметим, что значение `r2_score` снизилось как на обучающей, так и на тестовой выборке. Это значит, что модель с таким ограничением на минимальное количество элементов в листовой вершине недообучена и плохо улавливает закономерности в данных.

Теперь попробуем, наоборот, сделать значение `min_samples_leaf` меньше оптимального.

In [14]:

```
regressor = RandomForestRegressor(
    random_state=42,
    min_samples_leaf=3,
    min_samples_split=2,
    n_estimators=25,
    max_depth=5
)
regressor.fit(X_train, y_train)

print('train r2_score {:.4f}'.format(
    r2_score(y_train, regressor.predict(X_train))
))
print('test r2_score {:.4f}'.format(
    r2_score(y_test, regressor.predict(X_test))
))
```

```
train r2_score 0.7159
test r2_score 0.4645
```

Вывод.

Видно, что значение `r2_score` на обучающей выборке выросло, а на валидационной — упало. Значит, лес немного переобучился.

2. Решение задачи классификации с помощью Random Forest

Теперь проделаем похожие эксперименты для задачи классификации. Возьмём датасет (https://archive.ics.uci.edu/ml/datasets/Letter+Recognition) для распознавания латинских букв на изображениях.

Некотрые из признаков, содержащихся в датасете:

- `lettr` — заглавная буква (принимает значения от A до Z);
- `x-box` — горизонтальная позиция прямоугольника с буквой;
- `y-box` — вертикальная позиция прямоугольника с буквой;
- `width` — ширина прямоугольника;
- `high` — высота прямоугольника;
- `onpix` — количество пикселей, относящихся к цифре;
- `x-bar` — среднее значение x всех пикселей в прямоугольнике;
- `y-bar` — среднее значение y всех пикселей в прямоугольнике;
- `x2-bar` — выборочная дисперсия x;
- `y2-bar` — выборочная дисперсия y;
- `xybar` — корреляция x и y.

In [15]:

```
letters_df = pd.read_csv('letter-recognition.data', header=None)
print('dataset shape:', letters_df.shape)

letters_df.head()
```

```
dataset shape: (20000, 17)
```

Out[15]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	T	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	I	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	D	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	N	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

In [16]:

```
X = letters_df.values[:, 1:]
y = letters_df.values[:, 0]
```

2.1. Зависимость точности классификации от значений гиперпараметров

Разобьём данные на обучающую и тестовую выборки.

In [17]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

Для начала попробуем оценить оптимальное количество решающих деревьев в лесе, взяв значения всех остальных параметров по умолчанию. Построим график зависимости `accuracy` от `n_estimators` на обучающей и на тестовой выборках. В большинстве случаев, значение `n_estimators` берут в диапазоне от 10 до 100. Но здесь мы рассмотрим более широкий набор значений — от 1 до 200.

In [18]:

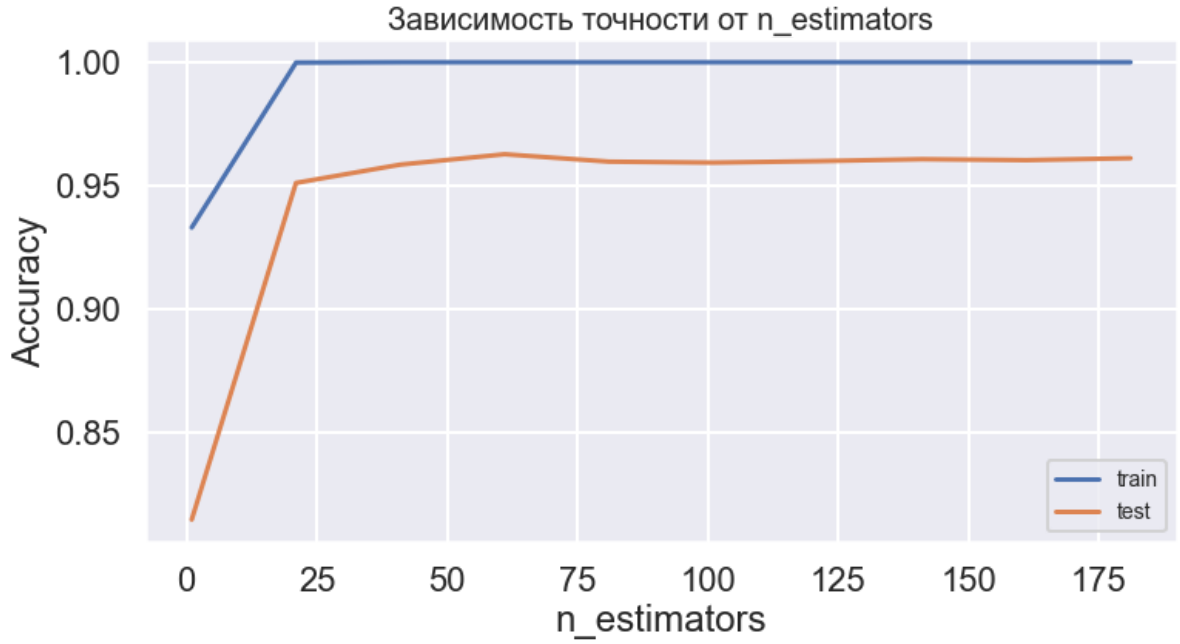
```
def get_train_and_test_accuracy(param_name, grid, other_params_dict={}):
    '''
    Функция для оценки точности классификации
    для заданных значений параметра param_name

    Параметры:
    1) param_name – название параметра, который собираемся варьировать,
    2) grid – сетка значений параметра,
    3) other_params_dict – словарь со значениями остальных параметров.
    '''

    train_acc, test_acc = [], []
    params_dict = copy.copy(other_params_dict)

    for param_value in grid:
        params_dict.update({param_name: param_value})
        estimator = RandomForestClassifier(**params_dict, random_state=42)
        estimator.fit(X_train, y_train)

        train_acc.append(accuracy_score(y_train, estimator.predict(X_train)))
        test_acc.append(accuracy_score(y_test, estimator.predict(X_test)))

    return train_acc, test_acc
```

In [19]:

```
def plot_dependence(
    param_name, grid=range(2, 20),
    other_params_dict={}, title=''
):
    '''
    Функция для отображения графика зависимости accuracy
    от значения параметра с названием param_name

    Параметры:
    1) param_name – название параметра, который собираемся варьировать,
    2) grid – сетка значений параметра,
    3) other_params_dict – словарь со значениями остальных параметров,
    4) title – заголовок графика.
    '''

    plt.figure(figsize=(12, 6))

    train_acc, test_acc = get_train_and_test_accuracy(
        param_name, grid, other_params_dict
    )

    plt.plot(grid, train_acc, label='train')
    plt.plot(grid, test_acc, label='test')
    plt.legend(fontsize=14)
    plt.xlabel(param_name)
    plt.ylabel('Accuracy')
    plt.title(title, fontsize=20)
    plt.show()
```

In [20]:

```
plot_dependence(
    'n_estimators', range(1, 200, 20),
    title='Зависимость точности от n_estimators'
)
```

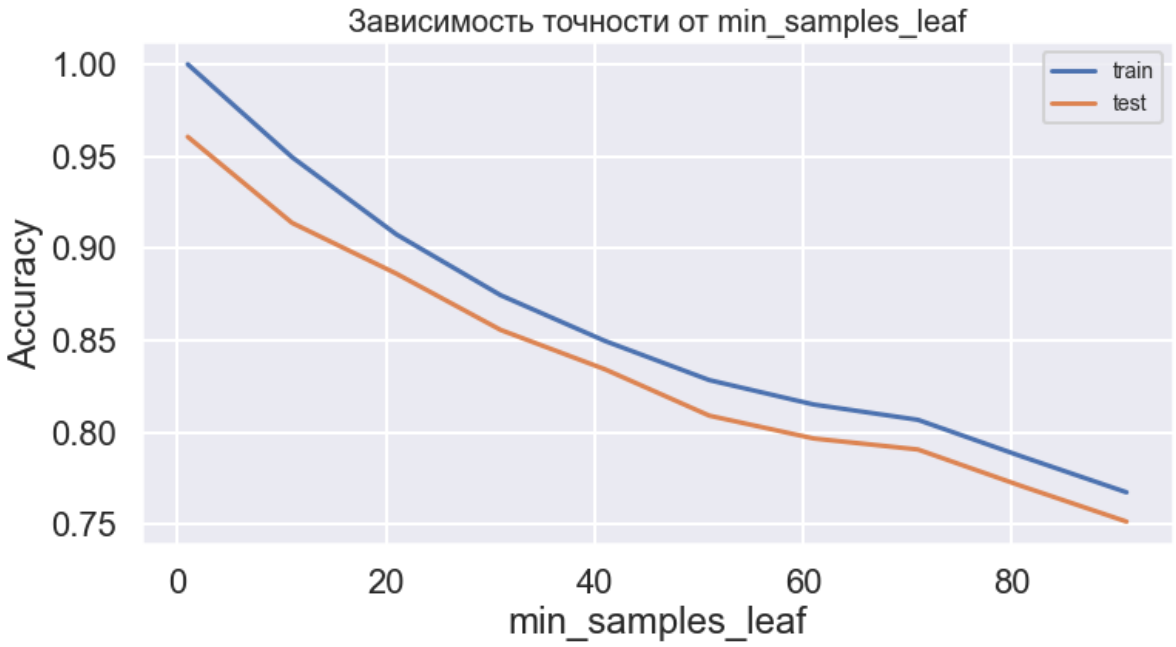


Как можно заметить, при `n_estimators` > 75 заметных изменений в `accuracy` как на обучающей, так и на тестовой выборке не происходит. В теории, при предположении, что все решающие деревья в лесе независимы между собой, должно получаться, что при увеличении числа случайных решающих деревьев в лесе дисперсия предсказания монотонно снижается, а точность монотонно повышается. Однако из-за того, что на практике решающие деревья попарно скоррелированны, такой эффект наблюдается лишь до некоторого значения `n_estimators` , а затем значительных изменений не происходит.

Теперь зафиксируем `n_estimators` равным 75 и будем использовать это значение во всех последующих экспериментах с данным датасетом. Построим график зависимости `accuracy` от `min_samples_leaf` на обучающей и на тестовой выборках.

In [21]:

```
plot_dependence(
    'min_samples_leaf', range(1, 100, 10), {'n_estimators': 75},
    title='Зависимость точности от min_samples_leaf'
)
```

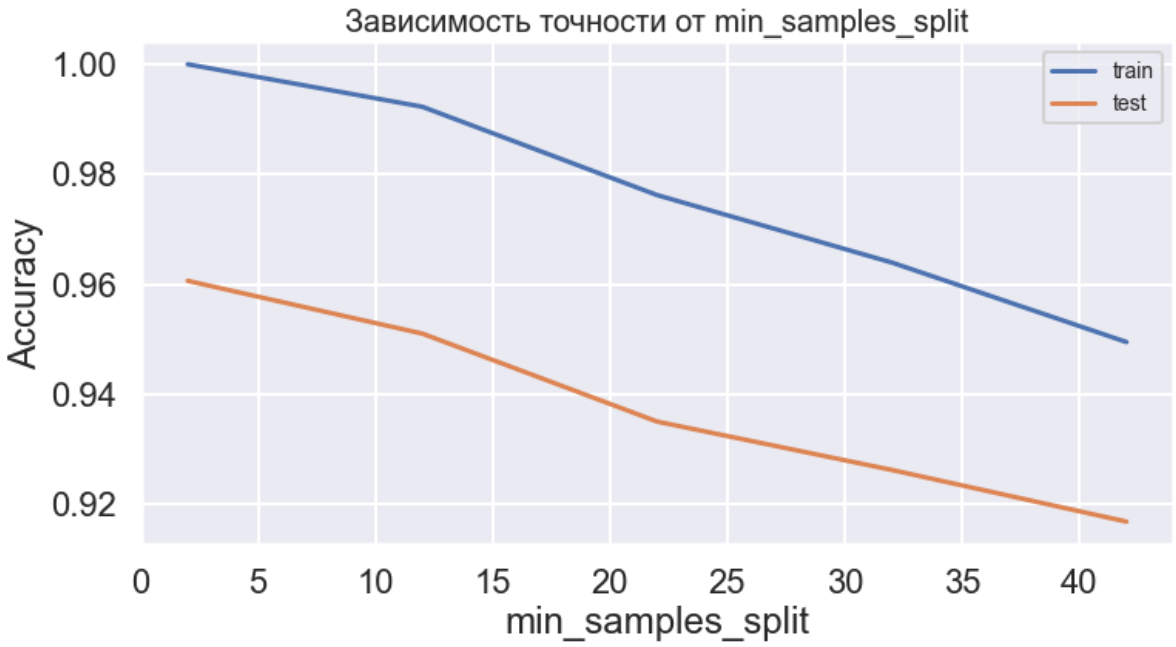


Вывод.

В целом наблюдается следующая закономерность: с увеличением значения `min_samples_leaf` падает качество и на обучающей и на тестовой выборке. Напомним, что при использовании одного решающего дерева закономерность была иной: до некоторого значения `min_samples_leaf` качество на тестовой выборке повышалось. Такая разница в поведении связана с тем, что при увеличении `min_samples_leaf` понижается дисперсия предсказаний, но повышается их смещенность (а как мы помним усреднение решающих деревьев уменьшает дисперсию и не изменяет смещение). Если в одиночном решающем дереве такой способ понижения дисперсии мог приносить положительные результаты, то при использовании случайного леса это теряет смысл.

In [22]:

```
plot_dependence(
    'min_samples_split', range(2, 50, 10), {'n_estimators': 75},
    title='Зависимость точности от min_samples_split'
)
```



Вывод.

При повышении значения `min_samples_split` происходит то же, что и при повышении `min_samples_leaf` .

А теперь найдём оптимальный набор гиперпараметров и использованием кросс-валидации. Зададим сетку для подбора параметров и сделаем кросс-валидацию с 5 фолдами (значение по-умолчанию). Напомним, что значение `None` параметра `max_depth` соответствует отсутствию ограничения на глубину дерева.

In [23]:

```
tree_gridsearch = GridSearchCV(
    estimator=RandomForestClassifier(),
    param_grid={
        'n_estimators': [10, 50, 75, 100],
        'max_depth': [2, 5, None],
        'min_samples_leaf': [1, 2, 5, 10]
    }
)

tree_gridsearch.fit(X_train, y_train)
print(tree_gridsearch.best_params_)
```

```
{'max_depth': None, 'min_samples_leaf': 1, 'n_estimators': 100}
```

In [24]:

```
train_accuracy = accuracy_score(
    y_train, tree_gridsearch.best_estimator_.predict(X_train)
)
test_accuracy = accuracy_score(
    y_test, tree_gridsearch.best_estimator_.predict(X_test)
)
print(f'train accuracy: {train_accuracy:.3f}')
print(f'test accuracy: {test_accuracy:.3f}')
```

```
train accuracy: 1.000
test accuracy: 0.961
```

Получилось довольно неплохое качество предсказания. Заметим ещё одну особенность подбора оптимальных гиперпараметров для случайного леса. Как вы помните, при использовании решающего дерева было довольно полезно ограничить его максимальную глубину. И, когда мы находили по кросс-валидации оптимальные гиперпараметры для одного решающего дерева, оптимальное значение `max_depth` не превосходило 5. В случае со случайным лесом оптимально вообще не ограничивать глубину решающего дерева, так как в таком случае деревья получаются менее смещёнными, а попарная корреляция между ними становится меньше.

3. Переобучение случайного леса

Переобучение — явление, при котором построенная модель хорошо объясняет примеры из обучающей выборки, но относительно плохо работает на объектах, не участвовавших в обучении, например, на объектах из тестовой/валидационной выборок.

Это связано с тем, что при обучении модели в обучающей выборке обнаруживаются некоторые случайные закономерности, которые отсутствуют в глобальном смысле.

In [25]:

```
def cum_metric(model, metric, x_test, y_test, shuffle=False):
    '''
    Считает значения метрики metric в зависимости от количества деревьев
    в обученной модели model метом композиции деревьев.

    :param model: модель RandomForest
    :param metric: метрика
    :param x_test: тестовая выборка
    :param y_test: ответы на тестовой выборке
    :param shuffle: надо ли перемешивать деревья

    :return: кумулятивные значения метрики
    '''

    # считаем предсказания по каждому дереву отдельно
    predictions_by_estimators = [
        est.predict(x_test) for est in model.estimators_
    ]

    if shuffle:
        random.shuffle(predictions_by_estimators)

    # кумулятивное среднее предсказаний
    numerator = np.array(predictions_by_estimators).cumsum(axis=0)
    n = len(predictions_by_estimators)
    denominator = (np.arange(n) + 1)[:, np.newaxis]
    cumpred = numerator / denominator

    cumacc = [metric(y_test, pred) for pred in cumpred]
    return np.array(cumacc)
```

3.1. Сильно шумные данные

Сгенерируем сильно зашумленную зависимость:

1. Выбираем x случайно на отрезке $[-5, 5]$
2. Генерируем $y = x$ + большой шум $\mathcal{N}(0, 10^2)$
3. Берем признаки:
   - x;
   - x^2.

Обучим на этих данных случайный лес на 200 деревьев и обычную линейную регрессию.

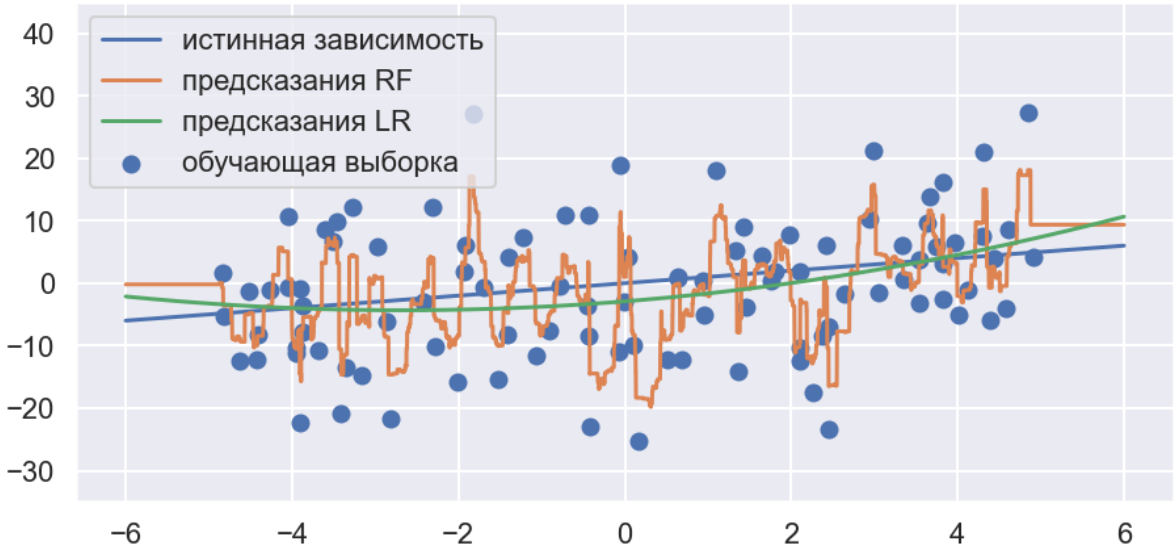
In [26]:

```
np.random.seed(44)  # специально подобрано, чтобы показать "что бывает" :)

train_size = 100  # размер обучающей выборки
poly_degree = 2  # максимальная степень полинома
noise_scale = 10  # станд. отклонение шума
n_estimators = 200  # количество деревьев

# Генерируем выборку
x_train = sps.uniform(loc=-5, scale=10).rvs(size=(train_size, 1))
y_train = x_train.ravel() + sps.norm(scale=noise_scale).rvs(size=train_size)
features_train = x_train ** np.arange(1, poly_degree+1)

# Обучаем модели
rf = RandomForestRegressor(n_estimators=n_estimators).fit(
    features_train, y_train
)
lr = LinearRegression().fit(features_train, y_train)
```

Для обоих методов построим предсказания по сетке.

In [27]:

```
grid = np.arange(-6, 6, 0.001)
features_test = grid[:, np.newaxis] ** np.arange(1, poly_degree+1)
preds_rf = rf.predict(features_test)
preds_lr = lr.predict(features_test)

plt.figure(figsize=(15, 7))
plt.plot(grid, grid, label='истинная зависимость')
plt.plot(grid, preds_rf, label='предсказания RF')
plt.plot(grid, preds_lr, label='предсказания LR')
plt.scatter(x_train, y_train, label='обучающая выборка')
plt.legend(loc=2)
plt.ylim((-35, 45));
```



Как видим, предсказания случайного леса очень шумные, явно видно сильное переобучение.

Посчитаем MSE на обучающей и тестовой выборках для обоих моделей.

In [28]:

```
# зашумляем метки на тесте аналогичным образом
y_grid = grid + sps.norm(scale=noise_scale).rvs(size=len(grid))

print('RF: mse_train = {:.2f}'.format(mean_squared_error(y_train, rf.predict(featur
print('RF: mse_test = {:.2f}'.format(mean_squared_error(y_grid, rf.predict(features_
print('LR: mse_train = {:.2f}'.format(mean_squared_error(y_train, lr.predict(featur
print('LR: mse_test = {:.2f}'.format(mean_squared_error(y_grid, lr.predict(features_
```

```
RF: mse_train = 21.51
RF: mse_test = 148.32
LR: mse_train = 108.45
LR: mse_test = 104.80
```

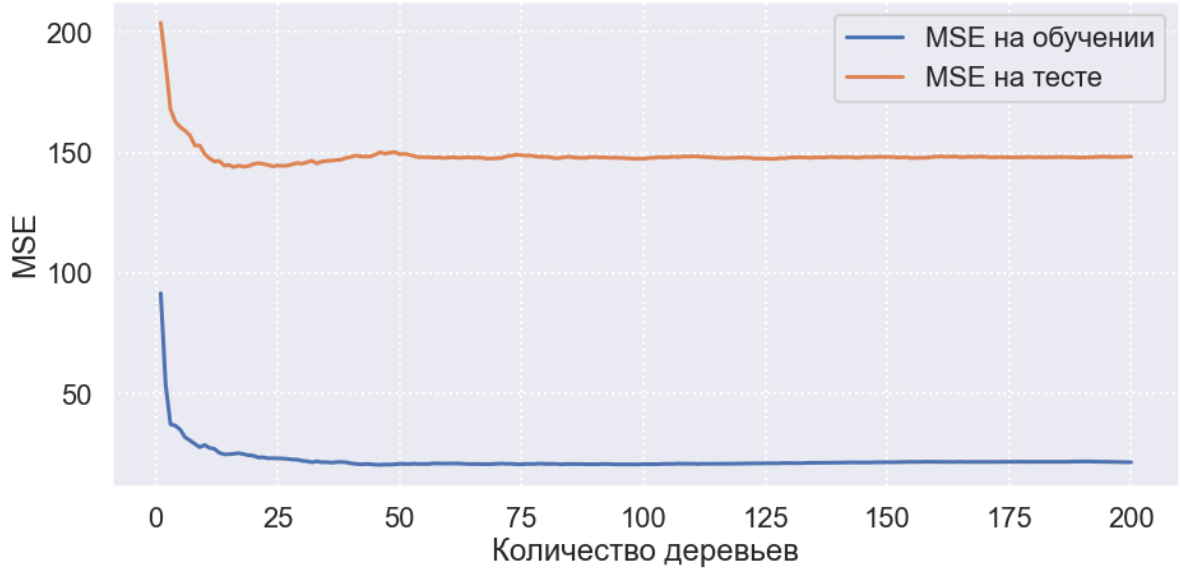
Как видим, ошибка на тесте для случайного леса существенно выше ошибки на трейне, это явно говорит о сильном переобучении. В то же время ошибки для линейной регрессии слабо отличаются, т.е. возможно, что линейная регрессия не переобучается. Более того, ошибка на тесте для линейной регрессии существенно ниже ошибки для случайного леса.

3.1.1 Зависимость от количества деревьев для одной модели

Построим график зависимости MSE от количества деревьев в случайном лесе для обучающей и тестовой выборок.

In [29]:

```
cum_mse_train = cum_metric(rf, mean_squared_error, features_train, y_train)
cum_mse_test = cum_metric(rf, mean_squared_error, features_test, y_grid)
estrimator_range = np.arange(n_estimators) + 1

plt.figure(figsize=(15, 7))
plt.plot(estrimator_range, cum_mse_train, label='MSE на обучении', lw=3)
plt.plot(estrimator_range, cum_mse_test, label='MSE на тесте', lw=3)
plt.grid(ls=':')
plt.xlabel('Количество деревьев'), plt.ylabel('MSE')
plt.legend();
```



Как видим, начиная с некоторого момента ошибка на тесте начинает расти, в то время как ошибка на

трейне падает. Это так же является признаком переобучения.

НО! Это свойство конкретной **реализации** модели. Вспоминаем, что случайный лес это усреднение независимых одинаково распределенных случайных деревьев. Т.е. мы получили **свойство, верное для выпавшего элементарного исхода** из вероятностного пространства, что не означает его выполнение для всех других элементарных исходов.

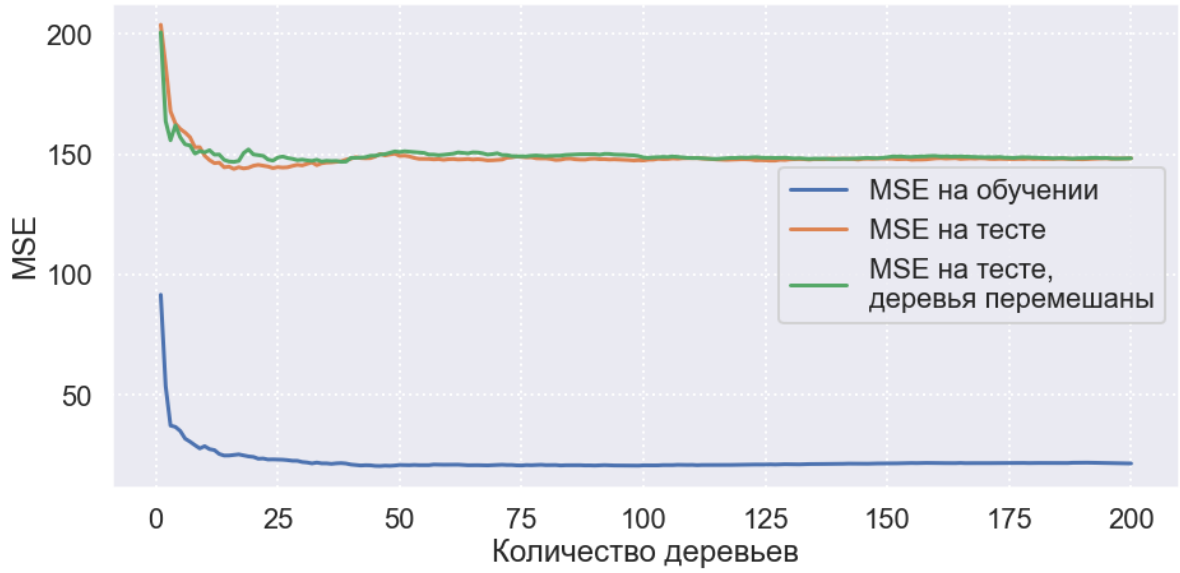
Вообще, нет никакой причины, почему случайный лес в среднем должен переобучаться с увеличением количества деревьев, ведь мы просто усредняем независимые при фиксированной выборке, одинаково распределенные деревья.

Если бы лес переобучался с ростом количества деревьев, то мы могли бы перемешать деревья в лесу так, чтобы ошибка на графике уменьшалась. Тогда переобучения не будет.

Переставим деревья в нашей обученной модели случайный образом.

In [30]:

```
cum_mse_train = cum_metric(rf, mean_squared_error, features_train, y_train)
cum_mse_test = cum_metric(rf, mean_squared_error, features_test, y_grid)
cum_mse_test_shuffle = cum_metric(
    rf, mean_squared_error, features_test, y_grid, shuffle=True
)

plt.figure(figsize=(15, 7))
plt.plot(estrimator_range, cum_mse_train, label='MSE на обучении', lw=3)
plt.plot(estrimator_range, cum_mse_test, label='MSE на тесте', lw=3)
plt.plot(estrimator_range, cum_mse_test_shuffle,
         label='MSE на тесте,\nдеревья перемешаны', lw=3)
plt.grid(ls=':')
plt.xlabel('Количество деревьев'), plt.ylabel('MSE')
plt.legend();
```



Как видим, теперь ошибка на тесте в целом падает с ростом количества деревьев.

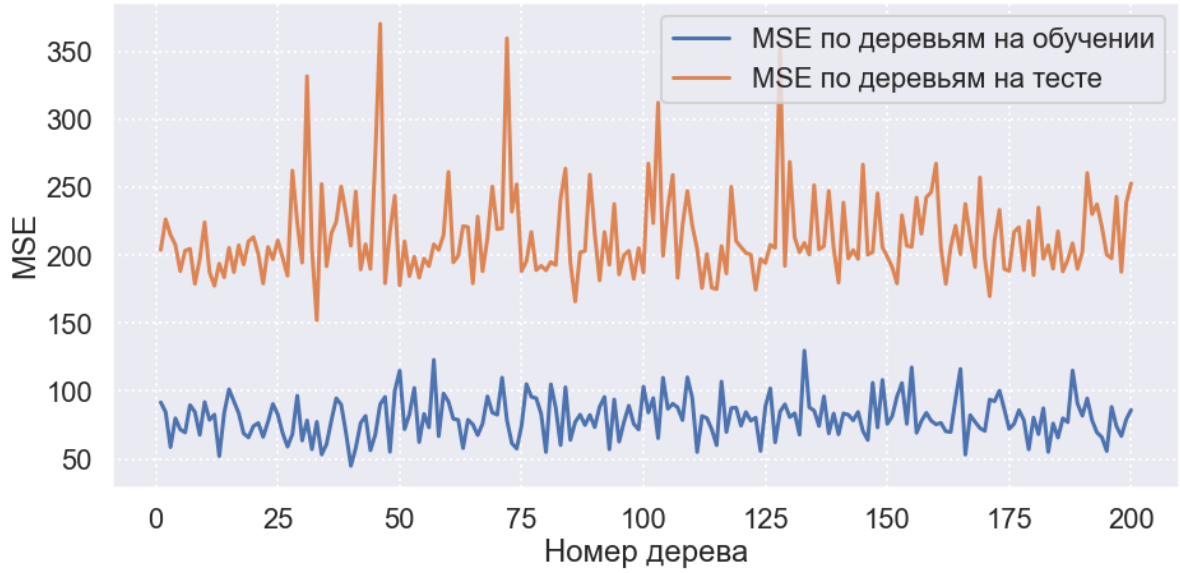
Чтобы лучше понять, посчитаем и визуализируем ошибку по каждом конкретному дереву.

In [31]:

```
def metric_trees(model, metric, x_test, y_test):
    '''Подсчет метрики для каждого дерева.

    :param model: модель RandomForest
    :param metric: метрика
    :param x_test: тестовая выборка
    :param y_test: ответы на тестовой выборки

    :return: массив значений метрики для каждого дерева
    '''
    predictions_by_estimators = [
        est.predict(x_test) for est in model.estimators_
    ]
    predictions_by_estimators = np.array(predictions_by_estimators)
    values = [metric(y_test, pred) for pred in predictions_by_estimators]
    return np.array(values)
```

In [32]:

```
trees_mse_train = metric_trees(
    rf, mean_squared_error, features_train, y_train
)
trees_mse_test = metric_trees(
    rf, mean_squared_error, features_test, y_grid
)

plt.figure(figsize=(15, 7))
plt.plot(estrimator_range, trees_mse_train,
         label='MSE по деревьям на обучении', lw=3)
plt.plot(estrimator_range, trees_mse_test,
         label='MSE по деревьям на тесте', lw=3)
plt.grid(ls=':')
plt.xlabel('Номер дерева'), plt.ylabel('MSE')
plt.legend()
plt.show()
```

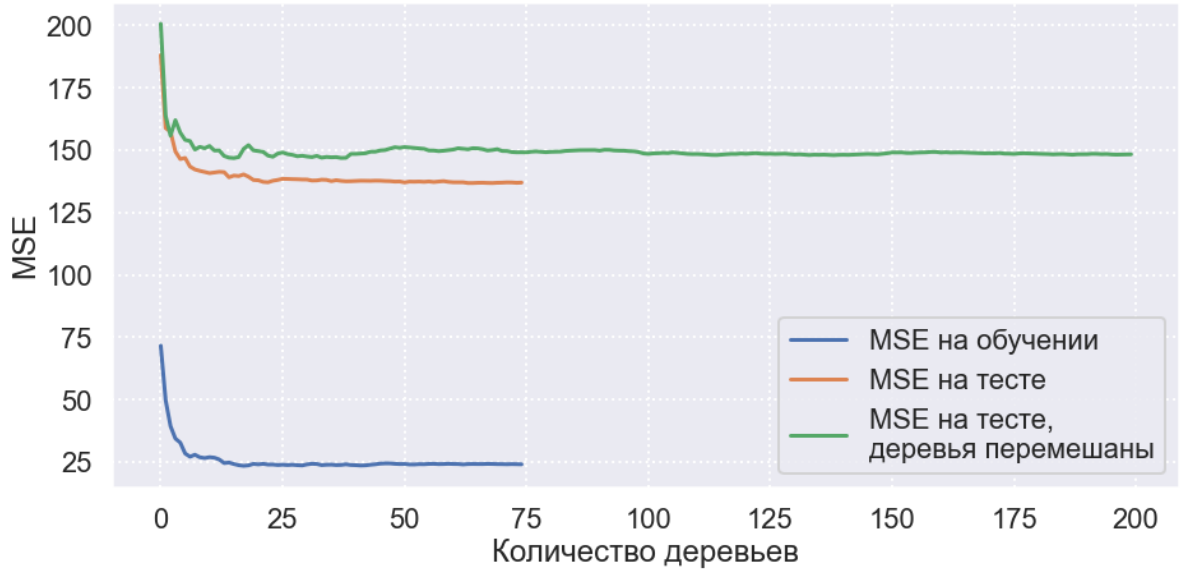


Как видим, звезды так совпали, что в начале идут деревья только с относительно низкой ошибкой, в следствии чего для всего леса ошибка с ростом количества деревьев возрастает.

Интересно, что будет если оставить **только "хорошие" деревья**, т.е. деревья, для которых ошибка не сильно велика. В таком случае можно получить лучшее качество композиции, но лишь немного.

In [33]:

```
rf.estimators_ = np.array(rf.estimators_)[trees_mse_test < 200]

cum_mse_train = cum_metric(rf, mean_squared_error, features_train, y_train)
cum_mse_test = cum_metric(rf, mean_squared_error, features_test, y_grid)

plt.figure(figsize=(15, 7))
plt.plot(cum_mse_train, label='MSE на обучении', lw=3)
plt.plot(cum_mse_test, label='MSE на тесте', lw=3)
plt.plot(cum_mse_test_shuffle,
         label='MSE на тесте,\nдеревья перемешаны', lw=3)
plt.grid(ls=':')
plt.xlabel('Количество деревьев'), plt.ylabel('MSE')
plt.legend();
```



3.1.2 Зависимость от количества деревьев в среднем

Если же мы будем рассматривать среднюю ошибку по всем моделям, то она будет монотонно убывать. Получим это с помощью симуляции. Для этого **1000 раз обучим модель случайного леса**, по каждой посчитаем кривые и усредним.

In [34]:

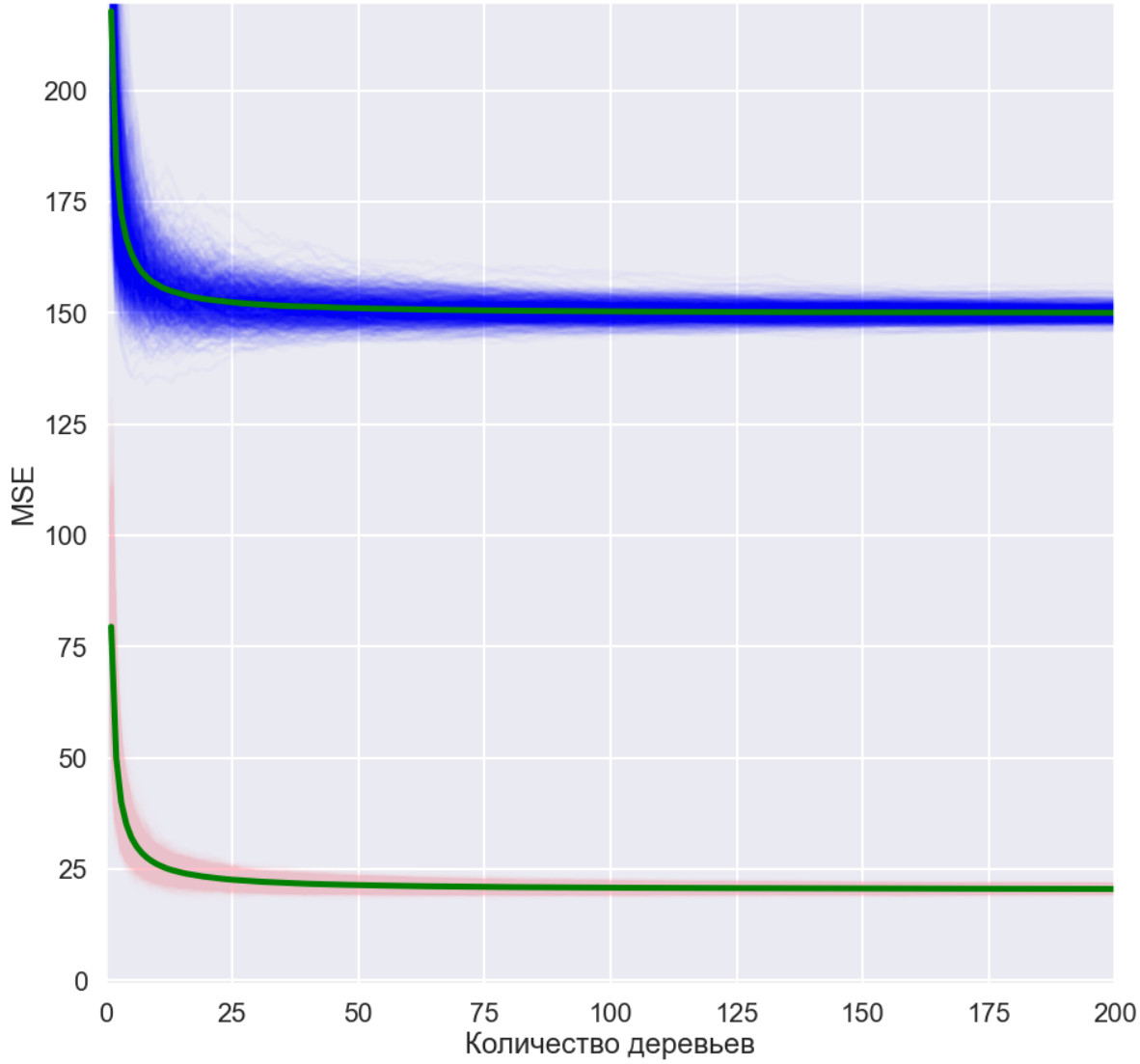
```
n_estimators = 200  # количество деревьев
n_iterations = 1000  # сколько раз обучим случайный лес
estrimator_range = np.arange(n_estimators) + 1

# В (i, j) записана точность на j деревьях для запуска i
scores_train = np.zeros((n_iterations, n_estimators))
scores_test = np.zeros((n_iterations, n_estimators))

# На каждой итерации обучаем свой случайный лес
for i in tqdm(range(n_iterations)):
    rf = RandomForestRegressor(n_estimators=n_estimators).fit(
        features_train, y_train
    )
    # Считаем точность классификации для всех "кумулятивных" наборах поддеревьев
    scores_train[i] = cum_metric(
        rf, mean_squared_error, features_train, y_train
    )
    scores_test[i] = cum_metric(
        rf, mean_squared_error, features_test, y_grid
    )
```

100% 1000/1000 [04:20<00:00, 4.60it/s]

На графике ниже изображено 1000 синих кривых, которые есть ошибка на трейне в засимости от количества деревьев для каждой конкретной модели. Зеленая кривая — их усреднение. Аналогично, розовые кривые — ошибка на тесте.

In [35]:

```
plt.figure(figsize=(15, 15))
for i in range(n_iterations):
    plt.plot(estrimator_range, scores_train[i], color='pink', alpha=0.02)
    plt.plot(estrimator_range, scores_test[i], color='blue', alpha=0.02)

plt.plot(estrimator_range, scores_train.mean(axis=0), lw=5, color='green')
plt.plot(estrimator_range, scores_test.mean(axis=0), lw=5, color='green')
plt.xlabel('Количество деревьев'), plt.ylabel('MSE')
plt.xlim((0, 200)), plt.ylim((None, 220))
plt.show()
```



Как видно из графика, зеленые кривые монотонно убывают. Так же на графике заметен провал для некоторых синих кривых при небольшом количестве деревьев. Этот провал соответствует случаям, в которых в начале попадались только хорошие деревья. На зеленой кривой такого провала нет, поскольку есть случаи и с очень плохими деревьями.

Отсюда можно сделать вывод про подбор оптимального количества деревьев. Подбор стоит делать только в случае рассмотрения одной конкретной реализации модели. Если же рассматривать среднюю ошибку, то она тем меньше, чем больше деревьев. Например, **бесполезно обучать модель заново на подобранное количество деревьев по одной реализации.**

Почему же в **среднем ошибка монотонно падает**? Вспомним формулы с лекции для случая аддитивного несмещенного шума:

$$\text{bias}^2 = \left(\mathsf{E}\,\widehat{y}_1(x) - f(x)\right)^2,$$
$$\text{variance} = \mathsf{D}\,\widehat{y}_1(x)\left[\frac{1}{T} + \frac{T-1}{T}\text{corr}\left(\widehat{y}_1(x), \widehat{y}_2(x)\right)\right],$$

где

- $y = f(x)$ — средняя ожидаемая зависимость,
- T — количество деревьев в композиции.

Как видим, bias-компонентна не меняется с увеличением количества деревьев, а variance-компонентна монотонно убывает к значению ковариации двух моделей. Они будут зависимы, т.к. обучаются на одной выборке.

Если T достаточно большое, получаем

$$\text{variance} = \mathsf{D}\,\widehat{y}_1(x) \cdot \text{corr}\left(\widehat{y}_1(x), \widehat{y}_2(x)\right).$$

3.1.3 Зависимость от максимальной глубины дерева

А как качество зависит от максимальной глубины деревьев? Для этого для каждого значения глубины обучим случайный лес на 200 деревьев 500 раз. Посчитаем качество и построим предсказательные интервалы для значения ошибки конкретного леса, т.е. учтем как ошибку определения среднего, так и разброс конкретных реализаций.

In [36]:

```
np.random.seed(42)

depths = np.arange(1, 15)
n_estimators = 200  # количество деревьев
n_iterations = 500  # сколько раз обучим случайный лес
scores_train = np.zeros((n_iterations, len(depths)))
scores_test = np.zeros((n_iterations, len(depths)))

for i in tqdm(range(n_iterations)):
    for max_depth in depths:
        rf = RandomForestRegressor(
            n_estimators=n_estimators, max_depth=max_depth
        ).fit(features_train, y_train)

        scores_train[i, max_depth-1] = mean_squared_error(
            y_train, rf.predict(features_train)
        )
        scores_test[i, max_depth-1] = mean_squared_error(
            y_grid, rf.predict(features_test)
        )
```

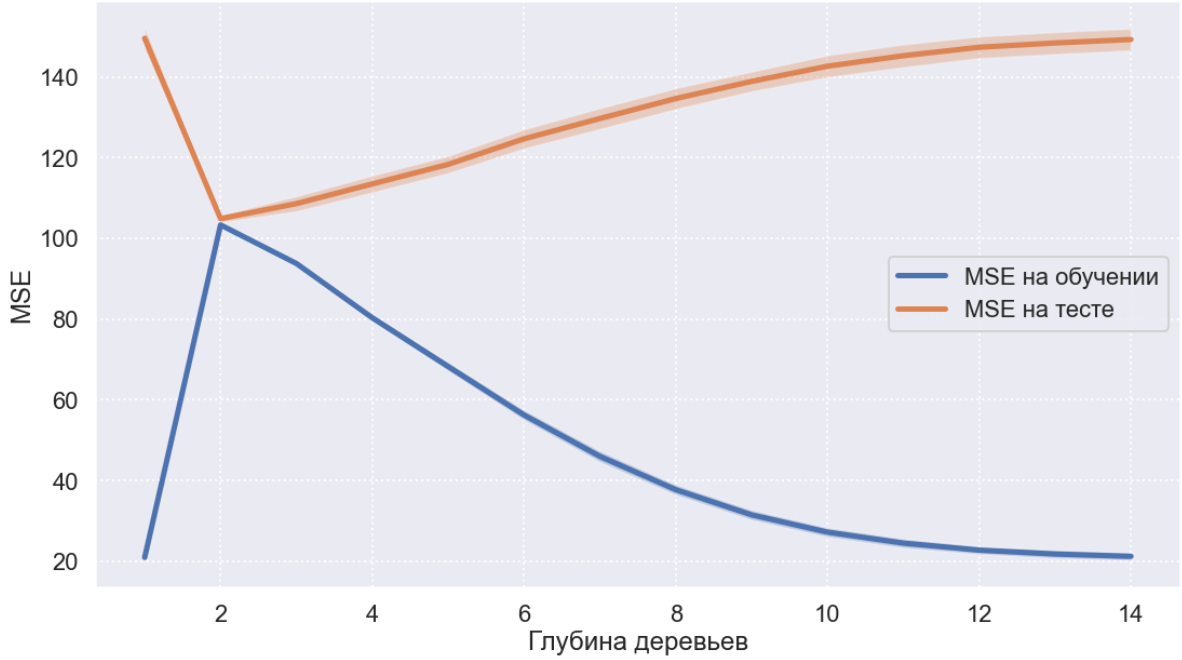
```
100%                                                500/500 [17:59<00:00, 2.14s/it]
```

In [37]:

```
scores_train_new = scores_train.copy()
for i in range(len(depths)):
    scores_train_new[:, (i+1)%len(depths)] = scores_train[:, i]

scores_test_new = scores_test.copy()
for i in range(len(depths)):
    scores_test_new[:, (i+1)%len(depths)] = scores_test[:, i]
```

In [38]:

```
plt.figure(figsize=(18, 10))
for scores, label in zip(
    [scores_train_new, scores_test_new], ['MSE на обучении', 'MSE на тесте']
):
    means = scores.mean(axis=0)
    std = scores.std(axis=0)
    deviations_value = std * 2 * np.sqrt(1 + 1 / n_iterations)
    plt.plot(depths, means, lw=5, label=label)
    plt.fill_between(
        depths, means - deviations_value, means + deviations_value, alpha=0.3
    )

plt.xlabel('Глубина деревьев')
plt.ylabel('MSE')
plt.grid(ls=':')
plt.legend()
plt.show()
```

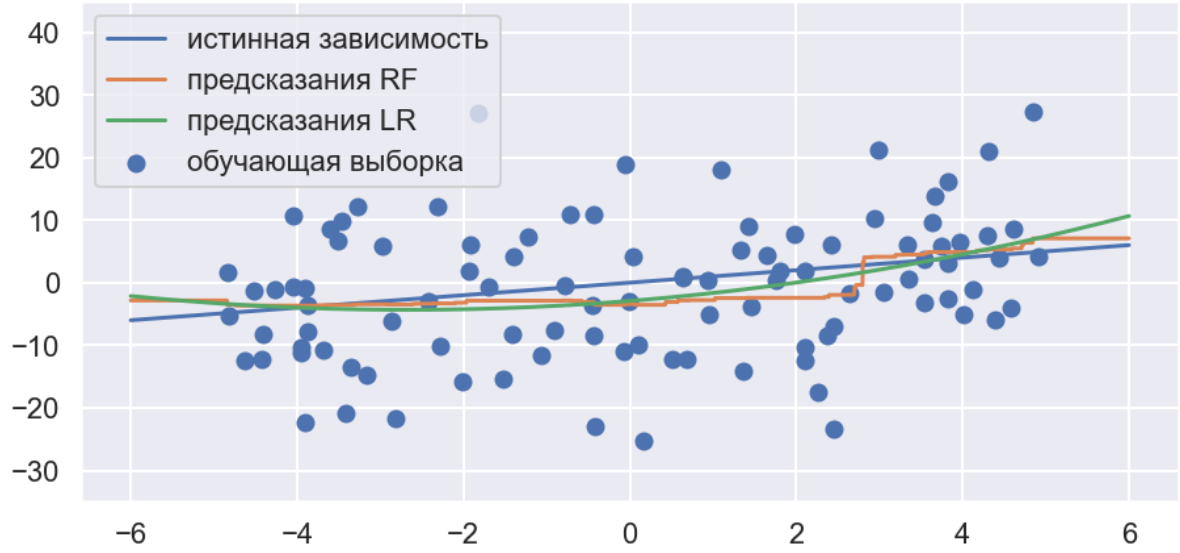


Как видим, ошибка на тесте растет с увеличением разрешенной глубины, в то время как на трейне она падает.

Опимальная глубина для теста равна 1. Посмотрим на сами предсказания для этой глубины.

In [39]:

```
rf = RandomForestRegressor(
    n_estimators=n_estimators, max_depth=1
).fit(features_train, y_train)

grid = np.arange(-6, 6, 0.001)
features_test = grid[:, np.newaxis] ** np.arange(1, poly_degree+1)
preds_rf = rf.predict(features_test)
preds_lr = lr.predict(features_test)

plt.figure(figsize=(15, 7))
plt.plot(grid, grid, label='истинная зависимость')
plt.plot(grid, preds_rf, label='предсказания RF', lw=3)
plt.plot(grid, preds_lr, label='предсказания LR')
plt.scatter(x_train, y_train, label='обучающая выборка')
plt.legend(loc=2)
plt.ylim((-35, 45));
```



Предсказания выглядят лучше, чем было в самом начале, видно, что модель не переобучена.

3.2. Не сильно шумные данные

Наконец, посмотрим, как зависит ошибка от глубины деревьев в случае если данные не сильно шумные. Для этого поставим стандартное отклонение шума равным 2, что гораздо меньше 10 из предыдущего случая.

In [40]:

```
np.random.seed(42)

train_size = 100  # размер обучающей выборки
poly_degree = 2  # максимальная степень полинома
noise_scale = 2  # станд. отклонение шума
n_estimators = 200  # количество деревьев

x_train = sps.uniform(loc=-5, scale=10).rvs(size=(train_size, 1))
y_train = x_train.ravel() + sps.norm(scale=noise_scale).rvs(size=train_size)
features_train = x_train ** np.arange(1, poly_degree+1)

rf = RandomForestRegressor(
    n_estimators=n_estimators
).fit(
    features_train, y_train
)
lr = LinearRegression().fit(features_train, y_train)
```

In [41]:

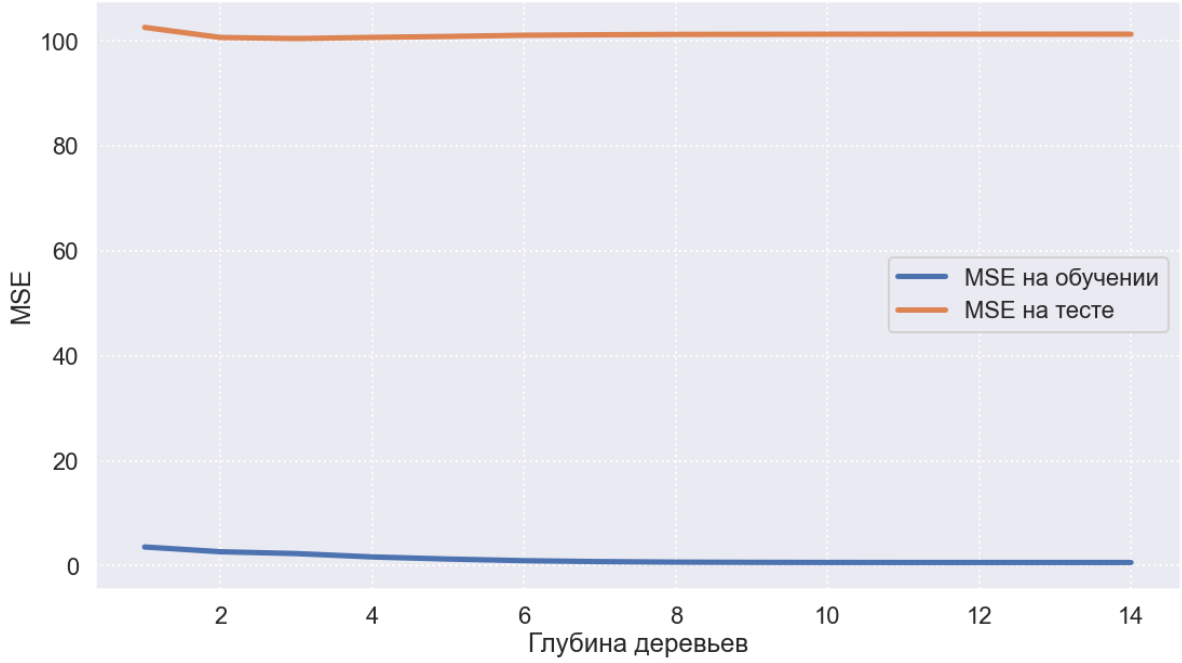
```
depths = np.arange(1, 15)
n_estimators = 200  # количество деревьев
n_iterations = 500  # сколько раз обучим случайный лес
scores_train = np.zeros((n_iterations, len(depths)))
scores_test = np.zeros((n_iterations, len(depths)))

for i in tqdm(range(n_iterations)):
    for max_depth in depths:
        rf = RandomForestRegressor(
            n_estimators=n_estimators, max_depth=max_depth
        ).fit(features_train, y_train)

        scores_train[i, max_depth-1] = mean_squared_error(
            y_train, rf.predict(features_train)
        )
        scores_test[i, max_depth-1] = mean_squared_error(
            y_grid, rf.predict(features_test)
        )
```

100% 500/500 [20:37<00:00, 2.97s/it]

Как видим из графика, от глубины почти ничего не зависит.

In [42]:

```
plt.figure(figsize=(18, 10))
for scores, label in zip(
    [scores_train, scores_test], ['MSE на обучении', 'MSE на тесте']
):
    means = scores.mean(axis=0)
    std = scores.std(axis=0)
    deviations_value = std * 2 * np.sqrt(1 + 1 / n_iterations)
    plt.plot(depths, means, lw=5, label=label)
    plt.fill_between(
        depths, means - deviations_value, means + deviations_value, alpha=0.3
    )

plt.xlabel('Глубина деревьев')
plt.ylabel('MSE')
plt.grid(ls=':')
plt.legend()
plt.show()
```



Таким образом, **глубину деревьев в случайном лесе имеет смысл подбирать только для очень шумных данных.**