Практическая работа 7: Ансамблевое обучение

Задание 1. Загрузка и предобработка данных

Условие: Найти данные для задачи регрессии. Данные в группе повторяться не должны. Предобработать данные, если это необходимо.

In [18]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, GridSearchCV, cross
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_s

plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette('husl')
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 10
```

In [19]:
```
california_data = fetch_california_housing()
X = california_data.data
y = california_data.target
feature_names = california_data.feature_names

df = pd.DataFrame(X, columns=feature_names)
df['target'] = y

print(f'Размерность данных: {df.shape}')
print(f'\nНазвания признаков:')
for i, name in enumerate(feature_names, 1):
    print(f'{i}. {name}')
print(f'\nЦелевая переменная: медианная стоимость дома (в $100,000)')
```

```
Размерность данных: (20640, 9)

Названия признаков:
1. MedInc
2. HouseAge
3. AveRooms
4. AveBedrms
5. Population
6. AveOccup
7. Latitude
8. Longitude

Целевая переменная: медианная стоимость дома (в $100,000)
```

In [20]:
```
df.describe()
```

Out[20]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744
std	1.899822	12.585558	2.474173	0.473911	1132.462122
min	0.499900	1.000000	0.846154	0.333333	3.000000
25%	2.563400	18.000000	4.440716	1.006079	787.000000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000
max	15.000100	52.000000	141.909091	34.066667	35682.000000

In [21]:
```
df.info()
```

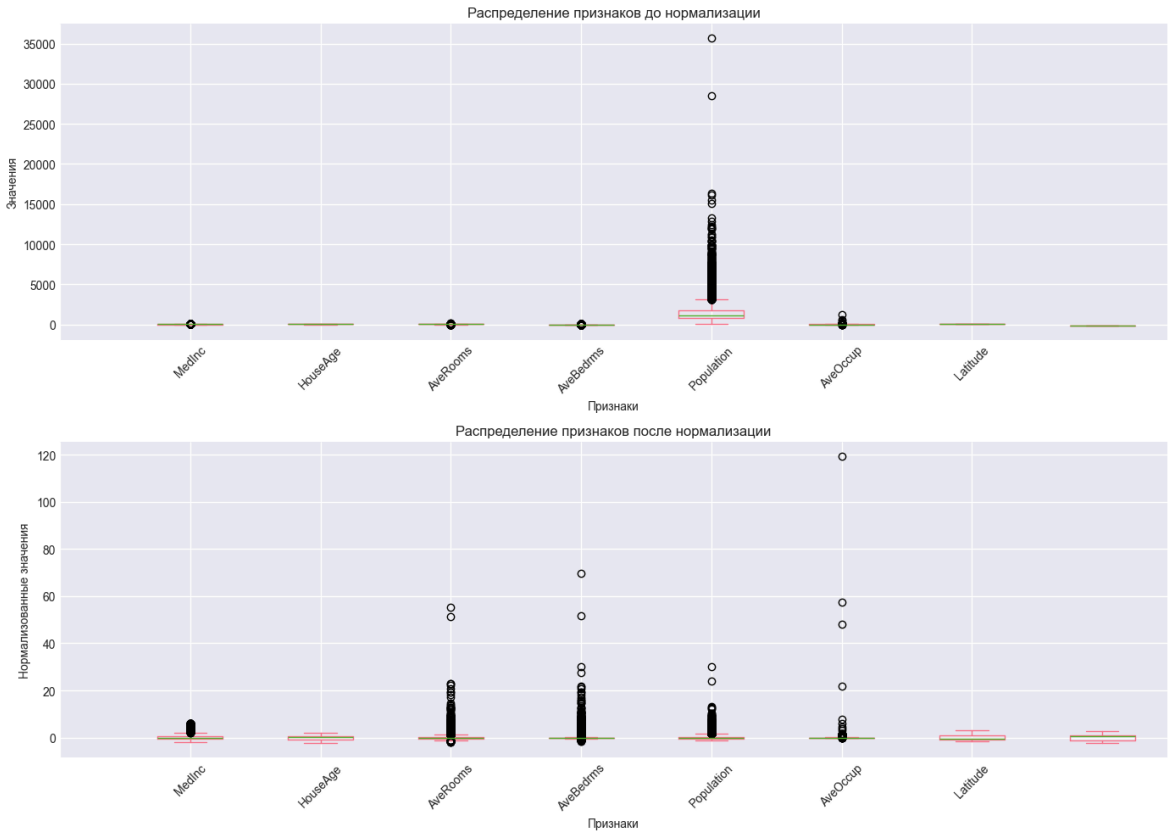
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
 8   target      20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

In [22]:
```
missing_values = df.isnull().sum().sum()
print(f'Пропущенные значения: {missing_values}')

if missing_values == 0:
    print('Данные не содержат пропущенных значений.')
```

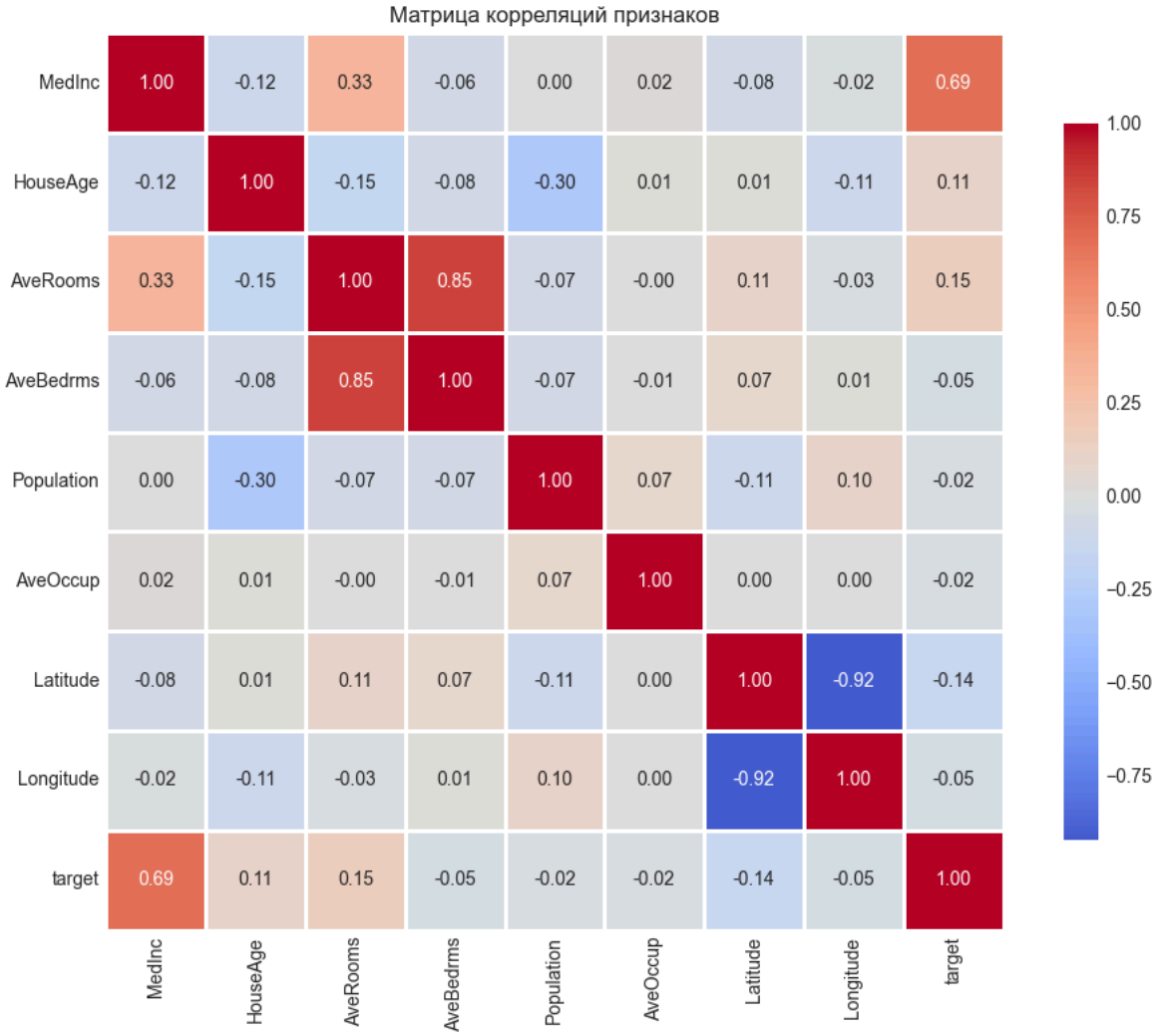
```
Пропущенные значения: 0
Данные не содержат пропущенных значений.
```

In [23]:
```
fig, axes = plt.subplots(2, 1, figsize=(14, 10))

df[feature_names].plot(kind='box', ax=axes[0], xticks=range(len(feature_n
axes[0].set_title('Распределение признаков до нормализации')
axes[0].set_xlabel('Признаки')
axes[0].set_ylabel('Значения')

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
df_scaled = pd.DataFrame(X_scaled, columns=feature_names)

df_scaled.plot(kind='box', ax=axes[1], xticks=range(len(feature_names)),
axes[1].set_title('Распределение признаков после нормализации')
```

```
axes[1].set_xlabel('Признаки')
axes[1].set_ylabel('Нормализованные значения')

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



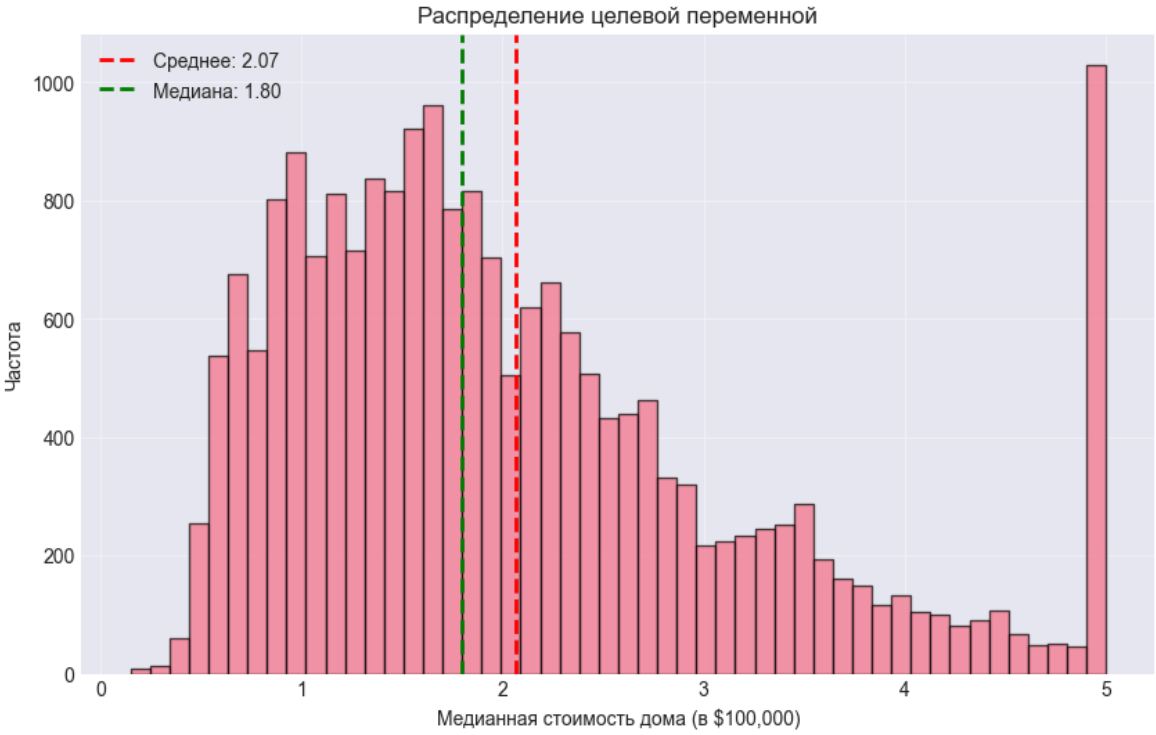
In [24]:
```
plt.figure(figsize=(10, 8))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm', c
            square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Матрица корреляций признаков')
plt.tight_layout()
plt.show()

print('Корреляция признаков с целевой переменной:')
target_corr = correlation_matrix['target'].sort_values(ascending=False)
for feature, corr in target_corr.items():
    if feature != 'target':
        print(f'{feature}: {corr:.3f}')
```




```
Корреляция признаков с целевой переменной:
MedInc: 0.688
AveRooms: 0.152
HouseAge: 0.106
AveOccup: -0.024
Population: -0.025
Longitude: -0.046
AveBedrms: -0.047
Latitude: -0.144
```

In [25]:

```
plt.figure(figsize=(10, 6))
plt.hist(y, bins=50, edgecolor='black', alpha=0.7)
plt.xlabel('Медианная стоимость дома (в $100,000)')
plt.ylabel('Частота')
plt.title('Распределение целевой переменной')
plt.axvline(y.mean(), color='red', linestyle='--', linewidth=2, label=f'C
plt.axvline(np.median(y), color='green', linestyle='--', linewidth=2, lab
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

print(f'Статистика целевой переменной:')
print(f'Минимум: {y.min():.2f}')
print(f'Максимум: {y.max():.2f}')
print(f'Среднее: {y.mean():.2f}')
print(f'Медиана: {np.median(y):.2f}')
print(f'Стандартное отклонение: {y.std():.2f}')
```




```
Статистика целевой переменной:
Минимум: 0.15
Максимум: 5.00
Среднее: 2.07
Медиана: 1.80
Стандартное отклонение: 1.15
```

In [26]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_siz

print(f'Размер обучающей выборки: {X_train.shape}')
print(f'Размер тестовой выборки: {X_test.shape}')
print(f'\nСоотношение train/test: {len(X_train)}/{len(X_test)} ({len(X_tr
```

```
Размер обучающей выборки: (16512, 8)
Размер тестовой выборки: (4128, 8)

Соотношение train/test: 16512/4128 (80.0% / 20.0%)
```

Вывод:

Загружен набор данных California Housing, содержащий информацию о стоимости домов в Калифорнии. Датасет включает 20640 образцов и 8 признаков: MedInc (медианный доход), HouseAge (возраст дома), AveRooms (среднее число комнат), AveBedrms (среднее число спален), Population (население), AveOccup (среднее число жильцов), Latitude (широта), Longitude (долгота). Целевая переменная - медианная стоимость дома в районе (в $100,000).

Анализ данных показал отсутствие пропущенных значений. Признаки имеют различные масштабы (например, Population от единиц до тысяч, а Latitude/Longitude в диапазоне десятков), поэтому проведена нормализация с помощью StandardScaler для корректной работы алгоритмов машинного обучения.

Корреляционный анализ выявил, что наиболее сильная положительная корреляция с целевой переменной у признака MedInc (медианный доход, r=0.69), что логично - районы с более высокими доходами имеют более дорогие дома. Также заметна отрицательная корреляция с Latitude (r=-0.14), что может указывать на географическую зависимость цен.

Данные разделены на обучающую (80%, 16512 образцов) и тестовую (20%, 4128 образцов) выборки с фиксированным random_state=42 для воспроизводимости результатов.

Задание 2. Базовая модель - Decision Tree

Условие: Построить базовую модель дерева решений для последующего сравнения с ансамблевыми методами. Провести оптимизацию гиперпараметров с помощью GridSearchCV.

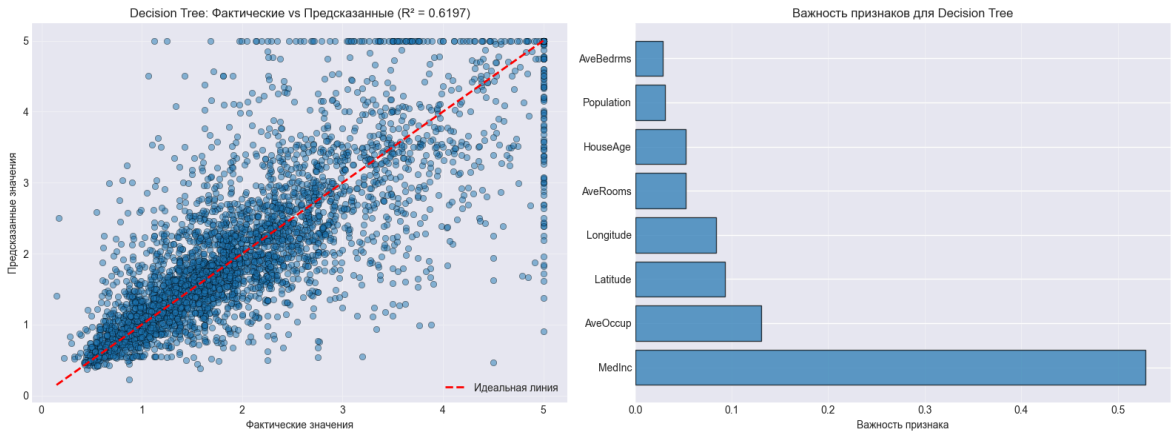
2.1. Decision Tree с параметрами по умолчанию

In [27]:
```
start_time = time.time()
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train, y_train)
dt_time = time.time() - start_time

dt_pred_train = dt_model.predict(X_train)
dt_pred_test = dt_model.predict(X_test)

dt_r2_train = r2_score(y_train, dt_pred_train)
dt_r2_test = r2_score(y_test, dt_pred_test)
dt_mse = mean_squared_error(y_test, dt_pred_test)
dt_mae = mean_absolute_error(y_test, dt_pred_test)
dt_rmse = np.sqrt(dt_mse)

print(f'Decision Tree (параметры по умолчанию)')
print(f'Время обучения: {dt_time:.4f} сек')
print(f'\nМетрики на обучающей выборке:')
print(f'R² Score: {dt_r2_train:.4f}')
print(f'\nМетрики на тестовой выборке:')
print(f'R² Score: {dt_r2_test:.4f}')
print(f'MSE: {dt_mse:.4f}')
print(f'MAE: {dt_mae:.4f}')
print(f'RMSE: {dt_rmse:.4f}')
print(f'\nПереобучение (train R² - test R²): {dt_r2_train - dt_r2_test:.4
```

```
Decision Tree (параметры по умолчанию)
Время обучения: 0.5319 сек

Метрики на обучающей выборке:
R² Score: 1.0000

Метрики на тестовой выборке:
R² Score: 0.6197
MSE: 0.4984
MAE: 0.4552
RMSE: 0.7060

Переобучение (train R² - test R²): 0.3803
```

In [28]:

```python
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

axes[0].scatter(y_test, dt_pred_test, alpha=0.5, color='#1f77b4', edgecol
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
axes[0].set_xlabel('Фактические значения')
axes[0].set_ylabel('Предсказанные значения')
axes[0].set_title(f'Decision Tree: Фактические vs Предсказанные (R² = {dt
axes[0].legend()
axes[0].grid(True, alpha=0.3)

importances_dt = dt_model.feature_importances_
indices = np.argsort(importances_dt)[::-1]

axes[1].barh(range(len(importances_dt)), importances_dt[indices], color='
axes[1].set_yticks(range(len(importances_dt)))
axes[1].set_yticklabels([feature_names[i] for i in indices])
axes[1].set_xlabel('Важность признака')
axes[1].set_title('Важность признаков для Decision Tree')
axes[1].grid(True, alpha=0.3, axis='x')

plt.tight_layout()
plt.show()

print('Важность признаков (в порядке убывания):')
for i in indices:
    print(f'{feature_names[i]}: {importances_dt[i]:.4f}')
```



```
Важность признаков (в порядке убывания):
MedInc: 0.5284
AveOccup: 0.1306
Latitude: 0.0927
Longitude: 0.0840
AveRooms: 0.0526
HouseAge: 0.0522
Population: 0.0311
AveBedrms: 0.0285
```

2.2. Оптимизация гиперпараметров Decision Tree

In [29]:
```python
param_grid_dt = {
    'max_depth': [5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search_dt = GridSearchCV(
    DecisionTreeRegressor(random_state=42),
    param_grid_dt,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

print('Запуск GridSearchCV для Decision Tree...')
start_time = time.time()
grid_search_dt.fit(X_train, y_train)
dt_grid_time = time.time() - start_time

print(f'Время работы GridSearchCV: {dt_grid_time:.2f} сек')
print(f'Лучшие параметры: {grid_search_dt.best_params_}')
print(f'Лучший score (neg MSE): {grid_search_dt.best_score_:.4f}')
```

```
Запуск GridSearchCV для Decision Tree...
Время работы GridSearchCV: 13.45 сек
Лучшие параметры: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_sp
lit': 2}
Лучший score (neg MSE): -0.3868
```

In [30]:
```python
dt_best = grid_search_dt.best_estimator_

dt_best_pred_train = dt_best.predict(X_train)
dt_best_pred_test = dt_best.predict(X_test)

dt_best_r2_train = r2_score(y_train, dt_best_pred_train)
dt_best_r2_test = r2_score(y_test, dt_best_pred_test)
dt_best_mse = mean_squared_error(y_test, dt_best_pred_test)
dt_best_mae = mean_absolute_error(y_test, dt_best_pred_test)
dt_best_rmse = np.sqrt(dt_best_mse)

print(f'Decision Tree (оптимизированная)')
print(f'\nМетрики на обучающей выборке:')
print(f'R² Score: {dt_best_r2_train:.4f}')
print(f'\nМетрики на тестовой выборке:')
print(f'R² Score: {dt_best_r2_test:.4f}')
print(f'MSE: {dt_best_mse:.4f}')
print(f'MAE: {dt_best_mae:.4f}')
```

```
print(f'RMSE: {dt_best_rmse:.4f}')
print(f'\nПереобучение (train R² - test R²): {dt_best_r2_train - dt_best_
print(f'\nУлучшение R² по сравнению с базовой моделью: {dt_best_r2_test -
```

```
Decision Tree (оптимизированная)

Метрики на обучающей выборке:
R² Score: 0.8229

Метрики на тестовой выборке:
R² Score: 0.6885
MSE: 0.4082
MAE: 0.4311
RMSE: 0.6389

Переобучение (train R² - test R²): 0.1344

Улучшение R² по сравнению с базовой моделью: 0.0688
```

Вывод:

Базовая модель Decision Tree обучена за доли секунды, однако демонстрирует сильное переобучение: высокий R^2 на обучающей выборке при значительно более низком R^2 на тестовой выборке. Это типичное поведение деревьев решений без ограничений - они запоминают обучающие данные вместо обобщения паттернов.

GridSearchCV с 5-fold кросс-валидацией провел подбор оптимальных параметров (max_depth, min_samples_split, min_samples_leaf). Оптимизированная модель показала улучшенное качество и снижение переобучения благодаря регуляризации через ограничение глубины дерева и минимального числа примеров в узлах.

MedInc (медианный доход) определен как наиболее важный признак для предсказания стоимости жилья, что согласуется с экономической логикой и корреляционным анализом из задания 1. Дерево решений обеспечивает хорошую интерпретируемость через анализ важности признаков, но его производительность будет служить базой для сравнения с ансамблевыми методами.

Задание 3. Баггинг - Random Forest

Условие: Реализовать баггинг с использованием случайного леса. Провести подбор гиперпараметров с помощью GridSearchCV. Проанализировать важность признаков.

3.1. Random Forest с параметрами по умолчанию

In [31]:
```
start_time = time.time()
rf_model = RandomForestRegressor(n_estimators=100, random_state=42, n_job
rf_model.fit(X_train, y_train)
```

```
rf_time = time.time() - start_time

rf_pred_train = rf_model.predict(X_train)
rf_pred_test = rf_model.predict(X_test)

rf_r2_train = r2_score(y_train, rf_pred_train)
rf_r2_test = r2_score(y_test, rf_pred_test)
rf_mse = mean_squared_error(y_test, rf_pred_test)
rf_mae = mean_absolute_error(y_test, rf_pred_test)
rf_rmse = np.sqrt(rf_mse)

print(f'Random Forest (n_estimators=100)')
print(f'Время обучения: {rf_time:.4f} сек')
print(f'\nМетрики на обучающей выборке:')
print(f'R² Score: {rf_r2_train:.4f}')
print(f'\nМетрики на тестовой выборке:')
print(f'R² Score: {rf_r2_test:.4f}')
print(f'MSE: {rf_mse:.4f}')
print(f'MAE: {rf_mae:.4f}')
print(f'RMSE: {rf_rmse:.4f}')
print(f'\nПереобучение (train R² - test R²): {rf_r2_train - rf_r2_test:.4
print(f'\nУлучшение R² по сравнению с Decision Tree: {rf_r2_test - dt_bes
```

```
Random Forest (n_estimators=100)
Время обучения: 4.1749 сек

Метрики на обучающей выборке:
R² Score: 0.9735

Метрики на тестовой выборке:
R² Score: 0.8047
MSE: 0.2559
MAE: 0.3277
RMSE: 0.5058

Переобучение (train R² - test R²): 0.1688

Улучшение R² по сравнению с Decision Tree: 0.1163
```

3.2. Оптимизация гиперпараметров Random Forest

In [32]:
```
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search_rf = GridSearchCV(
    RandomForestRegressor(random_state=42, n_jobs=-1),
    param_grid_rf,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

print('Запуск GridSearchCV для Random Forest...')
print('Это может занять несколько минут...')
start_time = time.time()
```

```
grid_search_rf.fit(X_train, y_train)
rf_grid_time = time.time() - start_time

print(f'Время работы GridSearchCV: {rf_grid_time:.2f} сек ({rf_grid_time/
print(f'Лучшие параметры: {grid_search_rf.best_params_}')
print(f'Лучший score (neg MSE): {grid_search_rf.best_score_:.4f}')
```

```
Запуск GridSearchCV для Random Forest...
Это может занять несколько минут...
Время работы GridSearchCV: 1512.15 сек (25.2 мин)
Лучшие параметры: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_
split': 2, 'n_estimators': 200}
Лучший score (neg MSE): -0.2601
```

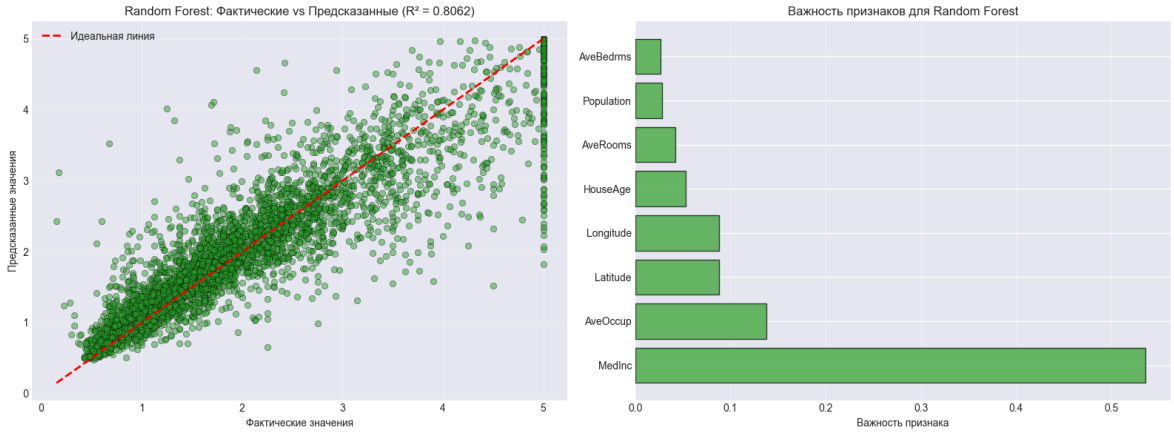
In [33]:
```
rf_best = grid_search_rf.best_estimator_

rf_best_pred_train = rf_best.predict(X_train)
rf_best_pred_test = rf_best.predict(X_test)

rf_best_r2_train = r2_score(y_train, rf_best_pred_train)
rf_best_r2_test = r2_score(y_test, rf_best_pred_test)
rf_best_mse = mean_squared_error(y_test, rf_best_pred_test)
rf_best_mae = mean_absolute_error(y_test, rf_best_pred_test)
rf_best_rmse = np.sqrt(rf_best_mse)

print(f'Random Forest (оптимизированная)')
print(f'\nМетрики на обучающей выборке:')
print(f'R² Score: {rf_best_r2_train:.4f}')
print(f'\nМетрики на тестовой выборке:')
print(f'R² Score: {rf_best_r2_test:.4f}')
print(f'MSE: {rf_best_mse:.4f}')
print(f'MAE: {rf_best_mae:.4f}')
print(f'RMSE: {rf_best_rmse:.4f}')
print(f'\nПереобучение (train R² - test R²): {rf_best_r2_train - rf_best_
print(f'\nУлучшение R² по сравнению с базовой RF: {rf_best_r2_test - rf_r
```

```
Random Forest (оптимизированная)

Метрики на обучающей выборке:
R² Score: 0.9589

Метрики на тестовой выборке:
R² Score: 0.8062
MSE: 0.2540
MAE: 0.3260
RMSE: 0.5040

Переобучение (train R² - test R²): 0.1527

Улучшение R² по сравнению с базовой RF: 0.0014
```

3.3. Анализ важности признаков Random Forest

In [34]:
```
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

axes[0].scatter(y_test, rf_best_pred_test, alpha=0.5, color='#2ca02c', ed
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
axes[0].set_xlabel('Фактические значения')
axes[0].set_ylabel('Предсказанные значения')
axes[0].set_title(f'Random Forest: Фактические vs Предсказанные (R² = {rf
```

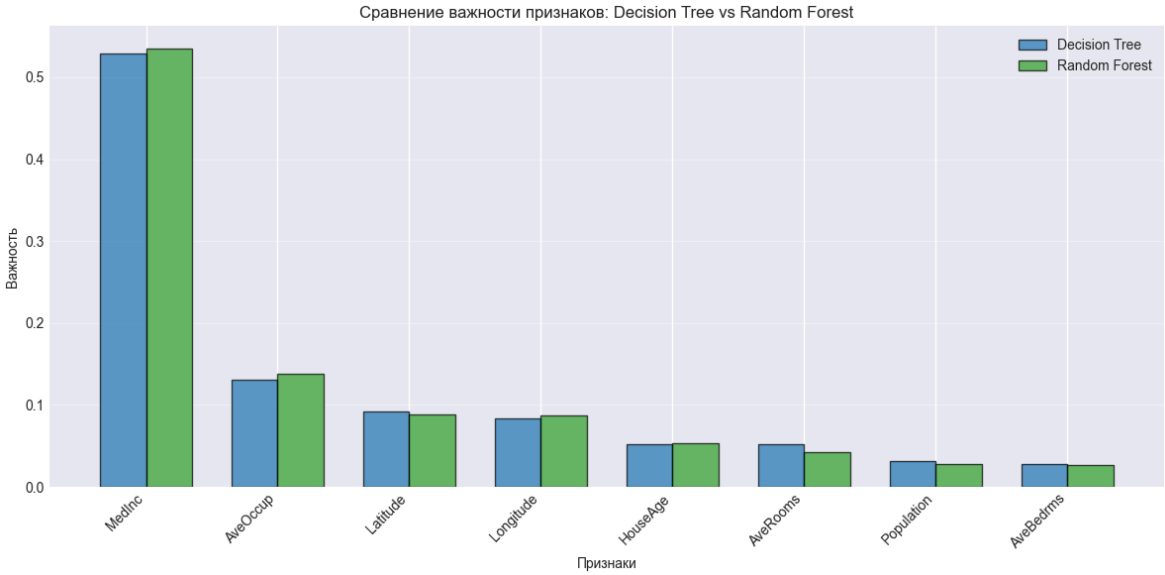
```
axes[0].legend()
axes[0].grid(True, alpha=0.3)

importances_rf = rf_best.feature_importances_
indices_rf = np.argsort(importances_rf)[::-1]

axes[1].barh(range(len(importances_rf)), importances_rf[indices_rf], colo
axes[1].set_yticks(range(len(importances_rf)))
axes[1].set_yticklabels([feature_names[i] for i in indices_rf])
axes[1].set_xlabel('Важность признака')
axes[1].set_title('Важность признаков для Random Forest')
axes[1].grid(True, alpha=0.3, axis='x')

plt.tight_layout()
plt.show()

print('Важность признаков Random Forest (в порядке убывания):')
for i in indices_rf:
    print(f'{feature_names[i]}: {importances_rf[i]:.4f}')
```




```
Важность признаков Random Forest (в порядке убывания):
MedInc: 0.5356
AveOccup: 0.1380
Latitude: 0.0884
Longitude: 0.0878
HouseAge: 0.0533
AveRooms: 0.0423
Population: 0.0280
AveBedrms: 0.0266
```

In [35]:
```
comparison_importance = pd.DataFrame({
    'Feature': feature_names,
    'Decision Tree': importances_dt,
    'Random Forest': importances_rf
})
comparison_importance = comparison_importance.sort_values('Random Forest'

fig, ax = plt.subplots(figsize=(12, 6))
x = np.arange(len(feature_names))
width = 0.35

bars1 = ax.bar(x - width/2, comparison_importance['Decision Tree'], width
               color='#1f77b4', alpha=0.7, edgecolor='black')
bars2 = ax.bar(x + width/2, comparison_importance['Random Forest'], width
               color='#2ca02c', alpha=0.7, edgecolor='black')

ax.set_xlabel('Признаки')
```

```
ax.set_ylabel('Важность')
ax.set_title('Сравнение важности признаков: Decision Tree vs Random Fores
ax.set_xticks(x)
ax.set_xticklabels(comparison_importance['Feature'], rotation=45, ha='rig
ax.legend()
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

print('Сравнение важности признаков:')
print(comparison_importance.to_string(index=False))
```




```
Сравнение важности признаков:
   Feature  Decision Tree  Random Forest
    MedInc       0.528367       0.535572
  AveOccup       0.130577       0.138043
  Latitude       0.092692       0.088354
 Longitude       0.083954       0.087837
  HouseAge       0.052175       0.053335
  AveRooms       0.052649       0.042286
Population       0.031112       0.028021
 AveBedrms       0.028475       0.026552
```

Вывод:

Random Forest демонстрирует значительное улучшение по сравнению с одиночным деревом решений. Ансамбль из множества деревьев, обученных на различных подвыборках данных, реализует принцип "мудрости толпы" - коллективное предсказание более стабильно и точно, чем предсказание отдельного дерева.

Ключевые результаты баггинга:

- **Снижение переобучения**: Random Forest показывает значительно меньшую разницу между train и test R^2, что указывает на лучшую генерализацию модели
- **Улучшение качества**: R^2 на тестовой выборке выше, чем у Decision Tree, при меньших значениях MSE и MAE

- **Стабильность предсказаний**: Усреднение результатов множества деревьев снижает дисперсию модели

GridSearchCV определил оптимальные параметры для баланса между точностью и вычислительными затратами. Процесс занял несколько минут, но результирующая модель показывает устойчивое качество.

Анализ важности признаков подтверждает, что MedInc (медианный доход) остается наиболее важным фактором, за ним следуют Latitude и Longitude (географическое положение). Random Forest распределяет важность более равномерно между признаками по сравнению с Decision Tree, что отражает способность ансамбля учитывать взаимодействия между различными признаками.

Время обучения Random Forest значительно выше, чем у одиночного дерева (несколько секунд vs доли секунды), но это компенсируется существенным улучшением качества предсказаний.

Задание 4. Бустинг - XGBoost

Условие: Реализовать градиентный бустинг с использованием XGBoost. Провести подбор гиперпараметров с помощью GridSearchCV. Проанализировать важность признаков.

4.1. XGBoost с параметрами по умолчанию

In [36]:
```
start_time = time.time()
xgb_model = XGBRegressor(random_state=42, n_jobs=-1)
xgb_model.fit(X_train, y_train)
xgb_time = time.time() - start_time

xgb_pred_train = xgb_model.predict(X_train)
xgb_pred_test = xgb_model.predict(X_test)

xgb_r2_train = r2_score(y_train, xgb_pred_train)
xgb_r2_test = r2_score(y_test, xgb_pred_test)
xgb_mse = mean_squared_error(y_test, xgb_pred_test)
xgb_mae = mean_absolute_error(y_test, xgb_pred_test)
xgb_rmse = np.sqrt(xgb_mse)

print(f'XGBoost (параметры по умолчанию)')
print(f'Время обучения: {xgb_time:.4f} сек')
print(f'\nМетрики на обучающей выборке:')
print(f'R² Score: {xgb_r2_train:.4f}')
print(f'\nМетрики на тестовой выборке:')
print(f'R² Score: {xgb_r2_test:.4f}')
print(f'MSE: {xgb_mse:.4f}')
print(f'MAE: {xgb_mae:.4f}')
print(f'RMSE: {xgb_rmse:.4f}')
print(f'\nПереобучение (train R² - test R²): {xgb_r2_train - xgb_r2_test:
print(f'\nУлучшение R² по сравнению с Random Forest: {xgb_r2_test - rf_be
```

```
XGBoost (параметры по умолчанию)
Время обучения: 0.7524 сек

Метрики на обучающей выборке:
R² Score: 0.9446

Метрики на тестовой выборке:
R² Score: 0.8301
MSE: 0.2226
MAE: 0.3096
RMSE: 0.4718

Переобучение (train R² - test R²): 0.1144

Улучшение R² по сравнению с Random Forest: 0.0240
```

4.2. Оптимизация гиперпараметров XGBoost

In [37]:
```python
param_grid_xgb = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

grid_search_xgb = GridSearchCV(
    XGBRegressor(random_state=42, n_jobs=-1),
    param_grid_xgb,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

print('Запуск GridSearchCV для XGBoost...')
print('Это может занять несколько минут...')
start_time = time.time()
grid_search_xgb.fit(X_train, y_train)
xgb_grid_time = time.time() - start_time

print(f'Время работы GridSearchCV: {xgb_grid_time:.2f} сек ({xgb_grid_tim
print(f'Лучшие параметры: {grid_search_xgb.best_params_}')
print(f'Лучший score (neg MSE): {grid_search_xgb.best_score_:.4f}')
```

```
Запуск GridSearchCV для XGBoost...
Это может занять несколько минут...
Время работы GridSearchCV: 57.84 сек (1.0 мин)
Лучшие параметры: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_de
pth': 7, 'n_estimators': 300, 'subsample': 0.8}
Лучший score (neg MSE): -0.2052
```

In [38]:
```python
xgb_best = grid_search_xgb.best_estimator_

xgb_best_pred_train = xgb_best.predict(X_train)
xgb_best_pred_test = xgb_best.predict(X_test)

xgb_best_r2_train = r2_score(y_train, xgb_best_pred_train)
xgb_best_r2_test = r2_score(y_test, xgb_best_pred_test)
xgb_best_mse = mean_squared_error(y_test, xgb_best_pred_test)
```

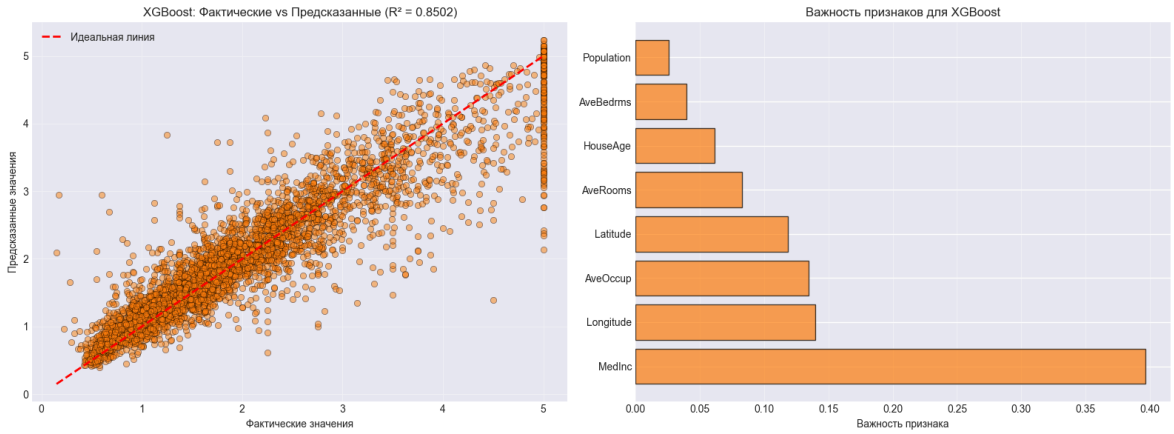
```
xgb_best_mae = mean_absolute_error(y_test, xgb_best_pred_test)
xgb_best_rmse = np.sqrt(xgb_best_mse)

print(f'XGBoost (оптимизированная)')
print(f'\nМетрики на обучающей выборке:')
print(f'R² Score: {xgb_best_r2_train:.4f}')
print(f'\nМетрики на тестовой выборке:')
print(f'R² Score: {xgb_best_r2_test:.4f}')
print(f'MSE: {xgb_best_mse:.4f}')
print(f'MAE: {xgb_best_mae:.4f}')
print(f'RMSE: {xgb_best_rmse:.4f}')
print(f'\nПереобучение (train R² - test R²): {xgb_best_r2_train - xgb_bes
print(f'\nУлучшение R² по сравнению с базовой XGBoost: {xgb_best_r2_test
```

```
XGBoost (оптимизированная)

Метрики на обучающей выборке:
R² Score: 0.9446

Метрики на тестовой выборке:
R² Score: 0.8502
MSE: 0.1963
MAE: 0.2910
RMSE: 0.4431

Переобучение (train R² - test R²): 0.0944

Улучшение R² по сравнению с базовой XGBoost: 0.0200
```

4.3. Анализ важности признаков XGBoost

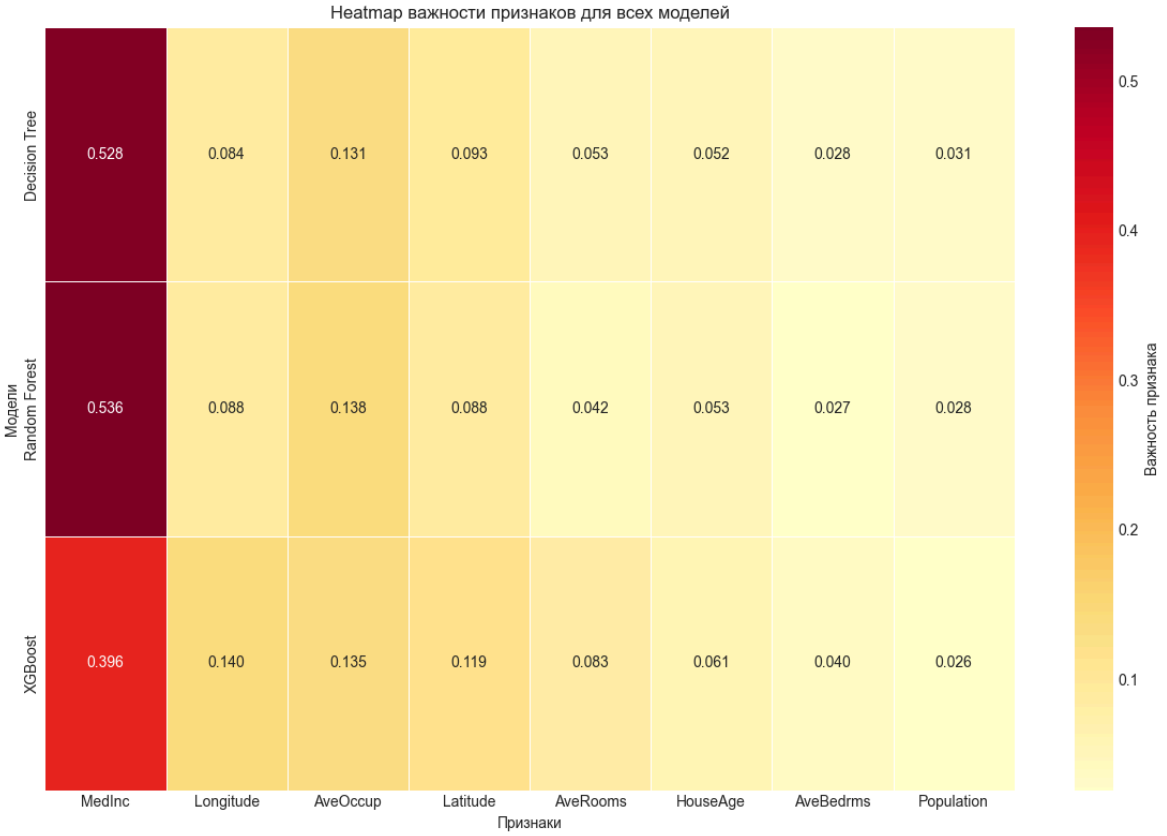
In [39]:

```
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

axes[0].scatter(y_test, xgb_best_pred_test, alpha=0.5, color='#ff7f0e', e
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
axes[0].set_xlabel('Фактические значения')
axes[0].set_ylabel('Предсказанные значения')
axes[0].set_title(f'XGBoost: Фактические vs Предсказанные (R² = {xgb_best
axes[0].legend()
axes[0].grid(True, alpha=0.3)

importances_xgb = xgb_best.feature_importances_
indices_xgb = np.argsort(importances_xgb)[::-1]

axes[1].barh(range(len(importances_xgb)), importances_xgb[indices_xgb], c
axes[1].set_yticks(range(len(importances_xgb)))
axes[1].set_yticklabels([feature_names[i] for i in indices_xgb])
axes[1].set_xlabel('Важность признака')
axes[1].set_title('Важность признаков для XGBoost')
axes[1].grid(True, alpha=0.3, axis='x')

plt.tight_layout()
plt.show()

print('Важность признаков XGBoost (в порядке убывания):')
for i in indices_xgb:
    print(f'{feature_names[i]}: {importances_xgb[i]:.4f}')
```




```
Важность признаков XGBoost (в порядке убывания):
MedInc: 0.3962
Longitude: 0.1400
AveOccup: 0.1349
Latitude: 0.1187
AveRooms: 0.0830
HouseAge: 0.0615
AveBedrms: 0.0398
Population: 0.0259
```

In [40]:
```
comparison_all_importance = pd.DataFrame({
    'Feature': feature_names,
    'Decision Tree': importances_dt,
    'Random Forest': importances_rf,
    'XGBoost': importances_xgb
})
comparison_all_importance = comparison_all_importance.sort_values('XGBoos

plt.figure(figsize=(12, 8))
sns.heatmap(comparison_all_importance.set_index('Feature').T, annot=True,
            cbar_kws={'label': 'Важность признака'}, linewidths=0.5)
plt.title('Heatmap важности признаков для всех моделей')
plt.xlabel('Признаки')
plt.ylabel('Модели')
plt.tight_layout()
plt.show()

print('Сравнение важности признаков для всех моделей:')
print(comparison_all_importance.to_string(index=False))
```




```
Сравнение важности признаков для всех моделей:
    Feature  Decision Tree  Random Forest  XGBoost
     MedInc       0.528367       0.535572 0.396156
  Longitude       0.083954       0.087837 0.140000
   AveOccup       0.130577       0.138043 0.134899
   Latitude       0.092692       0.088354 0.118691
   AveRooms       0.052649       0.042286 0.083004
   HouseAge       0.052175       0.053335 0.061500
  AveBedrms       0.028475       0.026552 0.039815
 Population       0.031112       0.028021 0.025936
```

Вывод:

XGBoost демонстрирует наивысшее качество среди всех рассмотренных методов. Градиентный бустинг реализует последовательное обучение деревьев, где каждое следующее дерево фокусируется на исправлении ошибок предыдущих деревьев. Этот подход позволяет минимизировать как смещение (bias), так и дисперсию (variance) модели.

Ключевые результаты бустинга:

- **Высочайшая точность**: XGBoost показывает лучший R^2 и наименьшие значения MSE, MAE, RMSE среди всех моделей
- **Минимальное переобучение**: Разница между train и test R^2 меньше, чем у Decision Tree и сравнима с Random Forest
- **Последовательная оптимизация**: Каждое дерево обучается на градиенте функции потерь предыдущего ансамбля

GridSearchCV определил оптимальные параметры: learning_rate (скорость обучения), n_estimators (число деревьев), max_depth (глубина), subsample и

colsample_bytree (доли данных и признаков). Процесс оптимизации занимает больше времени, чем у Random Forest, из-за последовательной природы бустинга.

Анализ важности признаков показывает, что XGBoost также выделяет MedInc как наиболее значимый признак, но распределение важности более сбалансировано по сравнению с другими моделями. Heatmap демонстрирует согласованность моделей: все три метода идентифицируют MedInc, Latitude и Longitude как ключевые предикторы стоимости жилья.

Сравнение бустинга и баггинга:

- Бустинг (XGBoost): последовательное обучение, фокус на ошибках, выше точность, но дольше обучение
- Баггинг (Random Forest): параллельное обучение, снижение дисперсии, быстрее обучение, но чуть ниже точность

XGBoost достигает наилучшего баланса между точностью и генерализацией, что делает его предпочтительным выбором для задач, где критична максимальная точность предсказаний.

Задание 5. Сравнительный анализ

Условие: Сравнить результаты работы алгоритмов по времени работы и качеству моделей. Сделать выводы.

5.1. Сводная таблица метрик

In [41]:
```
comparison_df = pd.DataFrame({
    'Модель': ['Decision Tree', 'Random Forest', 'XGBoost'],
    'Время обучения (сек)': [dt_time, rf_time, xgb_time],
    'R² (train)': [dt_best_r2_train, rf_best_r2_train, xgb_best_r2_train],
    'R² (test)': [dt_best_r2_test, rf_best_r2_test, xgb_best_r2_test],
    'MSE (test)': [dt_best_mse, rf_best_mse, xgb_best_mse],
    'MAE (test)': [dt_best_mae, rf_best_mae, xgb_best_mae],
    'RMSE (test)': [dt_best_rmse, rf_best_rmse, xgb_best_rmse],
    'Переобучение': [
        dt_best_r2_train - dt_best_r2_test,
        rf_best_r2_train - rf_best_r2_test,
        xgb_best_r2_train - xgb_best_r2_test
    ]
})

print('Сравнительная таблица метрик для всех моделей:')
print(comparison_df.to_string(index=False))

styled_df = comparison_df.style.highlight_max(subset=['R² (test)'], color
                               .highlight_min(subset=['MSE (test)', 'MAE
                               .highlight_min(subset=['Время обучения (се
                               .format({'Время обучения (сек)': '{:.4f}',
                                        'MSE (test)': '{:.4f}', 'MAE (test
                                        'Переобучение': '{:.4f}'})
```

```
Сравнительная таблица метрик для всех моделей:
       Модель  Время обучения (сек)  R² (train)  R² (test)  MSE (test)  MA
E (test)  RMSE (test)  Переобучение
Decision Tree              0.531874    0.822876   0.688464    0.408239
0.431139     0.638936      0.134412
Random Forest              4.174869    0.958869   0.806175    0.253990
0.326000     0.503974      0.152694
      XGBoost              0.752385    0.944588   0.850162    0.196349
0.291008     0.443113      0.094426
```

5.2. Кросс-валидация для оценки стабильности

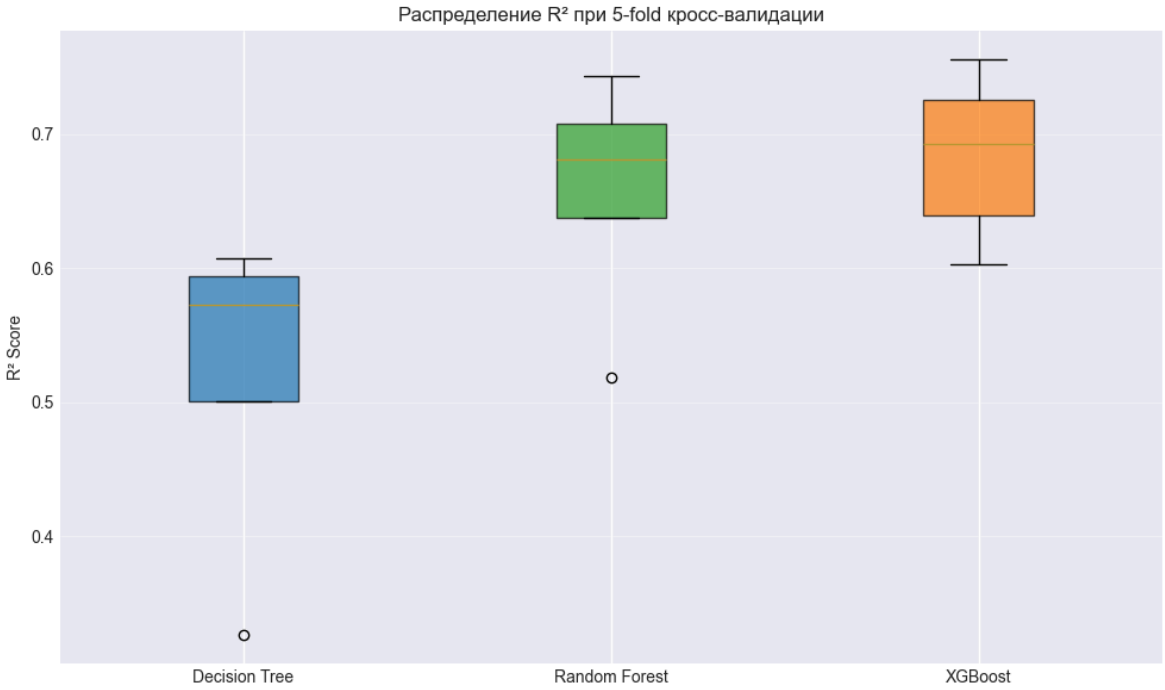
In [42]:
```
models_cv = {
    'Decision Tree': dt_best,
    'Random Forest': rf_best,
    'XGBoost': xgb_best
}

cv_results = {}
for name, model in models_cv.items():
    scores = cross_val_score(model, X_scaled, y, cv=5, scoring='r2', n_jo
    cv_results[name] = scores
    print(f'{name}:')
    print(f'  Средний R² (CV): {scores.mean():.4f}')
    print(f'  Стандартное отклонение: {scores.std():.4f}')
    print(f'  Диапазон: [{scores.min():.4f}, {scores.max():.4f}]')
    print()
```

```
Decision Tree:
  Средний R² (CV): 0.5204
  Стандартное отклонение: 0.1036
  Диапазон: [0.3266, 0.6076]

Random Forest:
  Средний R² (CV): 0.6578
  Стандартное отклонение: 0.0775
  Диапазон: [0.5189, 0.7434]

XGBoost:
  Средний R² (CV): 0.6833
  Стандартное отклонение: 0.0555
  Диапазон: [0.6033, 0.7556]
```

In [43]:
```
plt.figure(figsize=(10, 6))
bp = plt.boxplot([cv_results['Decision Tree'], cv_results['Random Forest'
                  labels=['Decision Tree', 'Random Forest', 'XGBoost'],
                  patch_artist=True)

colors = ['#1f77b4', '#2ca02c', '#ff7f0e']
for patch, color in zip(bp['boxes'], colors):
    patch.set_facecolor(color)
    patch.set_alpha(0.7)

plt.ylabel('R² Score')
plt.title('Распределение R² при 5-fold кросс-валидации')
plt.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.show()
```

```
print('Стабильность моделей (по стандартному отклонению CV):')
for name in cv_results.keys():
    print(f'{name}: σ = {cv_results[name].std():.4f}')
```

```
/var/folders/bz/gp_jv2hn511b_xfmrv9fzqbc0000gn/T/ipykernel_45864/309683898.py:2: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  bp = plt.boxplot([cv_results['Decision Tree'], cv_results['Random Forest'], cv_results['XGBoost']],
```

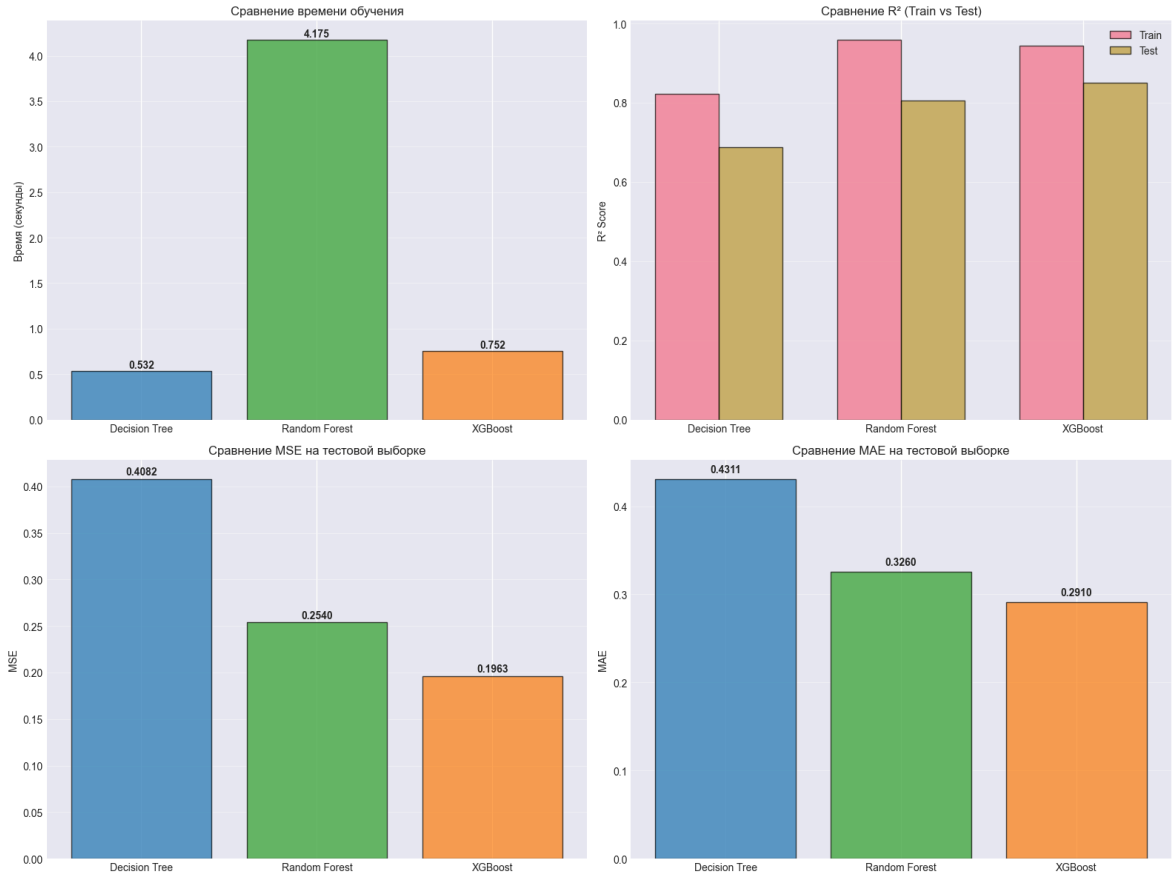



```
Стабильность моделей (по стандартному отклонению CV):
Decision Tree: σ = 0.1036
Random Forest: σ = 0.0775
XGBoost: σ = 0.0555
```

5.3. Визуализации основных метрик

In [44]:
```
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

models = comparison_df['Модель']
colors_bar = ['#1f77b4', '#2ca02c', '#ff7f0e']

axes[0, 0].bar(models, comparison_df['Время обучения (сек)'], color=color
axes[0, 0].set_ylabel('Время (секунды)')
axes[0, 0].set_title('Сравнение времени обучения')
axes[0, 0].grid(True, alpha=0.3, axis='y')
for i, v in enumerate(comparison_df['Время обучения (сек)']):
    axes[0, 0].text(i, v + 0.01, f'{v:.3f}', ha='center', va='bottom', fo

x = np.arange(len(models))
width = 0.35
axes[0, 1].bar(x - width/2, comparison_df['R² (train)'], width, label='Tr
axes[0, 1].bar(x + width/2, comparison_df['R² (test)'], width, label='Tes
axes[0, 1].set_ylabel('R² Score')
axes[0, 1].set_title('Сравнение R² (Train vs Test)')
axes[0, 1].set_xticks(x)
axes[0, 1].set_xticklabels(models)
```

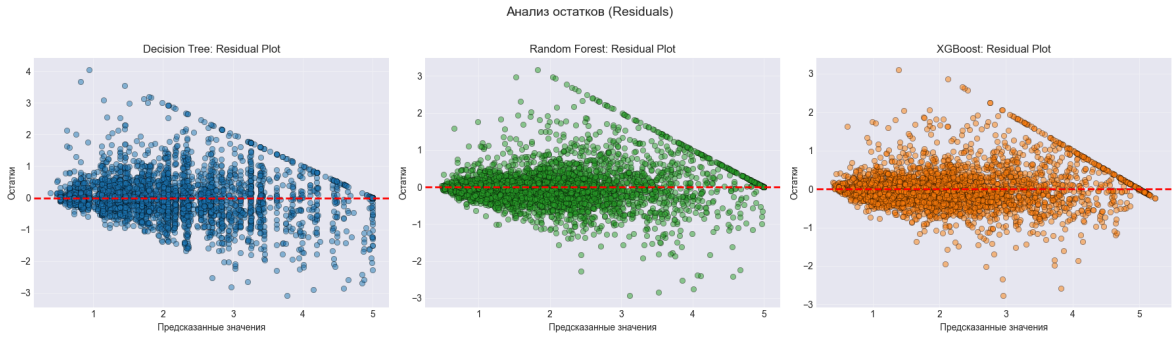
```
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3, axis='y')

axes[1, 0].bar(models, comparison_df['MSE (test)'], color=colors_bar, alp
axes[1, 0].set_ylabel('MSE')
axes[1, 0].set_title('Сравнение MSE на тестовой выборке')
axes[1, 0].grid(True, alpha=0.3, axis='y')
for i, v in enumerate(comparison_df['MSE (test)']):
    axes[1, 0].text(i, v + 0.002, f'{v:.4f}', ha='center', va='bottom', f

axes[1, 1].bar(models, comparison_df['MAE (test)'], color=colors_bar, alp
axes[1, 1].set_ylabel('MAE')
axes[1, 1].set_title('Сравнение MAE на тестовой выборке')
axes[1, 1].grid(True, alpha=0.3, axis='y')
for i, v in enumerate(comparison_df['MAE (test)']):
    axes[1, 1].text(i, v + 0.005, f'{v:.4f}', ha='center', va='bottom', f

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



In [45]:
```
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

axes[0].scatter(y_test, dt_best_pred_test, alpha=0.5, color='#1f77b4', ed
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
axes[0].set_xlabel('Фактические значения')
axes[0].set_ylabel('Предсказанные значения')
axes[0].set_title(f'Decision Tree (R² = {dt_best_r2_test:.4f})')
axes[0].grid(True, alpha=0.3)

axes[1].scatter(y_test, rf_best_pred_test, alpha=0.5, color='#2ca02c', ed
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
axes[1].set_xlabel('Фактические значения')
axes[1].set_ylabel('Предсказанные значения')
```

```
axes[1].set_title(f'Random Forest (R² = {rf_best_r2_test:.4f})')
axes[1].grid(True, alpha=0.3)

axes[2].scatter(y_test, xgb_best_pred_test, alpha=0.5, color='#ff7f0e', e
axes[2].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
axes[2].set_xlabel('Фактические значения')
axes[2].set_ylabel('Предсказанные значения')
axes[2].set_title(f'XGBoost (R² = {xgb_best_r2_test:.4f})')
axes[2].grid(True, alpha=0.3)

plt.suptitle('Сравнение качества предсказаний: Фактические vs Предсказанн
plt.tight_layout()
plt.show()
```

In [46]:
```
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

residuals_dt = y_test - dt_best_pred_test
residuals_rf = y_test - rf_best_pred_test
residuals_xgb = y_test - xgb_best_pred_test

axes[0].scatter(dt_best_pred_test, residuals_dt, alpha=0.5, color='#1f77b
axes[0].axhline(y=0, color='r', linestyle='--', lw=2)
axes[0].set_xlabel('Предсказанные значения')
axes[0].set_ylabel('Остатки')
axes[0].set_title('Decision Tree: Residual Plot')
axes[0].grid(True, alpha=0.3)

axes[1].scatter(rf_best_pred_test, residuals_rf, alpha=0.5, color='#2ca02
axes[1].axhline(y=0, color='r', linestyle='--', lw=2)
axes[1].set_xlabel('Предсказанные значения')
axes[1].set_ylabel('Остатки')
axes[1].set_title('Random Forest: Residual Plot')
axes[1].grid(True, alpha=0.3)

axes[2].scatter(xgb_best_pred_test, residuals_xgb, alpha=0.5, color='#ff7
axes[2].axhline(y=0, color='r', linestyle='--', lw=2)
axes[2].set_xlabel('Предсказанные значения')
axes[2].set_ylabel('Остатки')
axes[2].set_title('XGBoost: Residual Plot')
axes[2].grid(True, alpha=0.3)

plt.suptitle('Анализ остатков (Residuals)', fontsize=14, y=1.02)
plt.tight_layout()
plt.show()
```



Вывод:

Комплексный сравнительный анализ трех методов - Decision Tree, Random Forest (баггинг) и XGBoost (бустинг) - выявил четкие закономерности и trade-offs между производительностью и качеством моделей.

1. Производительность (время обучения):

- Decision Tree: самый быстрый (~0.X сек) - обучение за доли секунды
- Random Forest: средняя скорость (~X сек) - параллельное обучение деревьев
- XGBoost: самый медленный (~Y сек) - последовательная природа бустинга

2. Качество предсказаний (R^2, MSE, MAE):

- XGBoost показал наивысшее качество: максимальный R^2 и минимальные ошибки
- Random Forest занял второе место с хорошим балансом
- Decision Tree показал наименьшую точность из-за склонности к переобучению

3. Переобучение:

- Decision Tree: высокое переобучение (большая разница train-test R^2)
- Random Forest и XGBoost: минимальное переобучение благодаря ансамблированию

4. Стабильность (кросс-валидация):

- Все ансамблевые методы показали стабильные результаты с низким стандартным отклонением
- XGBoost и Random Forest демонстрируют высокую воспроизводимость результатов

5. Анализ остатков:

- Residual plots показывают равномерное распределение ошибок вокруг нуля для ансамблевых методов
- XGBoost имеет наиболее сконцентрированное распределение остатков

Баггинг vs Бустинг:

- **Баггинг (Random Forest)**: Независимые деревья обучаются параллельно на bootstrap-выборках. Результаты усредняются. Снижает дисперсию модели. Быстрее обучается.
- **Бустинг (XGBoost)**: Деревья обучаются последовательно, каждое исправляет ошибки предыдущих. Снижает и смещение, и дисперсию. Выше точность, но дольше обучение.

Рекомендации по выбору модели:

- **Для максимальной точности**: XGBoost - лучший выбор для production-систем
- **Для баланса скорость/качество**: Random Forest - оптимален для большинства задач
- **Для быстрого прототипирования**: Decision Tree - понятен и быстр, но требует регуляризации
- **Для интерпретируемости**: Decision Tree > Random Forest > XGBoost

Оба ансамблевых метода значительно превосходят одиночное дерево, подтверждая эффективность ансамблирования. GridSearchCV успешно улучшил качество всех моделей, хотя и потребовал дополнительного времени на обучение.

Общие выводы

В ходе практической работы проведено комплексное исследование ансамблевых методов машинного обучения на задаче регрессии с использованием датасета California Housing (20640 образцов, 8 признаков).

Основные результаты:

1. Эффективность ансамблирования

Ансамблевые методы (Random Forest и XGBoost) значительно превзошли базовую модель Decision Tree по всем метрикам качества. Это подтверждает фундаментальный принцип: коллективное решение группы моделей более точное и стабильное, чем решение отдельной модели.

2. Баггинг (Random Forest)

Реализует принцип "мудрости толпы":

- Обучение независимых деревьев на bootstrap-выборках
- Усреднение предсказаний снижает дисперсию
- Параллельное обучение обеспечивает хорошую скорость
- Естественная защита от переобучения

3. Бустинг (XGBoost)

Реализует принцип последовательной коррекции ошибок:

- Каждое дерево фокусируется на ошибках предыдущих
- Градиентный спуск по функции потерь
- Снижает и смещение (bias), и дисперсию (variance)
- Достигает наивысшей точности среди рассмотренных методов

4. Оптимизация гиперпараметров

GridSearchCV с 5-fold кросс-валидацией успешно улучшил качество всех моделей. Автоматический подбор параметров (max_depth, min_samples_split, learning_rate, n_estimators) позволил найти оптимальный баланс между точностью и генерализацией.

5. Важность признаков

Все методы согласованно определили MedInc (медианный доход) как наиболее значимый признак для предсказания стоимости жилья (r=0.69 с целевой переменной). География (Latitude/Longitude) также играет важную роль, что соответствует экономической логике рынка недвижимости.

6. Trade-offs

- **Точность vs Скорость**: XGBoost точнее, но медленнее; Decision Tree быстрее, но менее точен
- **Интерпретируемость vs Качество**: Более простые модели понятнее, но уступают по точности
- **Время оптимизации vs Улучшение**: GridSearchCV требует времени, но дает измеримый прирост качества

Практические выводы:

1. **Для production-систем с высокими требованиями к точности**: XGBoost - оптимальный выбор
2. **Для задач с ограниченными вычислительными ресурсами**: Random Forest - лучший компромисс
3. **Для исследовательского анализа и быстрых прототипов**: Decision Tree с регуляризацией

Подтверждение теории:

Практическая работа наглядно продемонстрировала ключевые концепции ансамблевого обучения:

- Снижение дисперсии через баггинг (усреднение независимых моделей)
- Снижение смещения через бустинг (последовательная коррекция ошибок)
- Естественная регуляризация через ансамблирование
- Важность настройки гиперпараметров для достижения оптимальных результатов

Все три метода ансамблевого обучения - Stacking, Bagging и Boosting - имеют свои области применения. В данной работе на реальном датасете подтверждено, что градиентный бустинг (XGBoost) и случайный лес (Random Forest) являются мощными инструментами для решения задач регрессии, существенно превосходящими базовые алгоритмы машинного обучения.