МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)

Институт: №8 «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Прикладные системы и фреймворки искусственного интеллекта»

Группа: М8О-408Б-22

Студент: А. А. Останина

Преподаватель:

Москва, 2025

Лабораторная работа №1. Проведение исследований с алгоритмом KNN

Выбор начальных условий

Классификация

Выбрала датасет «Online Shoppers Purchasing Intention». Задача заключается в прогнозировании, совершит ли пользователь покупку в онлайн-магазине, на основе его поведения в сессии. Обоснование: данные отражают реальную бизнес-проблем, содержат поведенческие признаки и выраженный дисбаланс классов, что требует корректного выбора метрик.

Регрессия

Выбрала датасет «Oxford Parkinson's Disease Telemonitoring». Задача заключается в прогнозировании оценки моторных симптомов (UPDRS) у пациентов по голосовым биомаркерам. Обоснование: задача соответствует актуальной проблеме телемедицины, целевая переменная непрерывна, а признаки — медицински интерпретируемы.

Метрики

Метрики для классификации:

- F1-Score
- ROC-AUC
- Confusion Matrix

F1-Score учитывает и точность, и полноту, что критично при дисбалансе классов. ROC-AUC оценивает качество ранжирования независимо от порога. Confusion Matrix наглядно показывает типы ошибок для принятия бизнес-решений.

Метрики для регрессии:

- MSE
- RMSE
- MAE
- R^2

MSE и RMSE строго штрафуют за крупные ошибки. MAE даёт понятную среднюю ошибку в исходных единицах. R^2 показывает долю дисперсии, объяснённую моделью, что удобно для общей интерпретации.

Создание бейзлайна и оценка качества

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import (accuracy_score, f1_score, roc_auc_score, confusion_
                             mean_squared_error, mean_absolute_error, r2_score)

from sklearn.metrics import ConfusionMatrixDisplay
```

Классификация

Загрузка датасета

In [2]:
```
df_class = pd.read_csv('datasets/online_shoppers_intention.csv')
```

Размер датасета

In [3]:
```
df_class.shape
```

Out[3]:
```
(12330, 18)
```

Первые 5 строк

In [4]:
```
df_class.head()
```

Out[4]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	Prod
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	

Информация о данных

In [5]:
```
df_class.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

Статистика по числовым признакам

In [6]:
```
df_class.describe()
```

Out[6]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	I
count	12330.000000	12330.000000	12330.000000	12330.000000	
mean	2.315166	80.818611	0.503569	34.472398	
std	3.321784	176.779107	1.270156	140.749294	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	1.000000	7.500000	0.000000	0.000000	
75%	4.000000	93.256250	0.000000	0.000000	
max	27.000000	3398.750000	24.000000	2549.375000	

Баланс классов

In [7]:
```
df_class['Revenue'].value_counts()
```

Out[7]:
```
Revenue
False    10422
True      1908
Name: count, dtype: int64
```

Копирование датасета для его дальнейшего преобразования

In [8]:
```
df_class_clean = df_class.copy()
```

Кодирование категориальных признаков с помощью `LabelEncoder`

In [9]:
```
categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le
    print(f"  Закодирована колонка: {col}")
```

```
Закодирована колонка: Month
Закодирована колонка: VisitorType
```

Выделение признаков и таргета, их разделение на выборки для обучения и тестирования

In [10]:
```
X = df_class_clean.drop('Revenue', axis=1)
y = df_class_clean['Revenue']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"\nРазмеры выборок:")
print(f"  Обучающая выборка: {X_train.shape}")
print(f"  Тестовая выборка: {X_test.shape}")
print(f"  Распределение классов в train: {np.bincount(y_train)}")
print(f"  Распределение классов в test: {np.bincount(y_test)}")
```

```
Размеры выборок:
  Обучающая выборка: (8631, 17)
  Тестовая выборка: (3699, 17)
  Распределение классов в train: [7295 1336]
  Распределение классов в test: [3127  572]
```

Масштабирование данных с помощью `StandardScaler`

In [11]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Обучение модели классификации `KNeighborsClassifier`

In [12]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train_scaled, y_train)

y_pred = knn_classifier.predict(X_test_scaled)
y_pred_proba = knn_classifier.predict_proba(X_test_scaled)[:, 1]
```

Вычисление метрик

In [13]:
```
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)
```

```
print(f"Accuracy: {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")

print("\nМатрица ошибок:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Accuracy: 0.8724
F1-Score: 0.4720
ROC-AUC: 0.7952

Матрица ошибок:
[[3016  111]
 [ 361  211]]
```

Визуализация матрицы ошибок

In [14]:
```
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['No Purchase', 'Purchase'])
disp.plot(cmap='Blues')
plt.title('Матрица ошибок для KNN классификатора', fontsize=14)
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

Дополнительная оценка результатов модели

In [15]:
```
TN, FP, FN, TP = cm.ravel()
precision = TP / (TP + FP) if (TP + FP) > 0 else 0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0
```

```
print(f"  Precision: {precision:.3f}")
print(f"    - Из {TP+FP} предсказанных покупок, {TP} были верными")
print(f"  Recall: {recall:.3f}")
print(f"    - Из {TP+FN} реальных покупок, нашли {TP}")
```

```
 Precision: 0.655
   - Из 322 предсказанных покупок, 211 были верными
 Recall: 0.369
   - Из 572 реальных покупок, нашли 211
```

Регрессия

Загрузка датасета

In [16]:
```
df_reg = pd.read_csv('datasets/parkinsons.csv')
```

Размер датасета

In [17]:
```
df_reg.shape
```

Out[17]:
```
(5875, 22)
```

Первые 5 строк

In [18]:
```
df_reg.head()
```

Out[18]:

	subject#	age	sex	test_time	motor_UPDRS	total_UPDRS	Jitter(%)	Jitter(Abs)	Jit
0	1	72	0	5.6431	28.199	34.398	0.00662	0.000034	
1	1	72	0	12.6660	28.447	34.894	0.00300	0.000017	
2	1	72	0	19.6810	28.695	35.389	0.00481	0.000025	
3	1	72	0	25.6470	28.905	35.810	0.00528	0.000027	
4	1	72	0	33.6420	29.187	36.375	0.00335	0.000020	

5 rows × 22 columns

Информация о данных

In [19]:
```
df_reg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5875 entries, 0 to 5874
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   subject#       5875 non-null   int64  
 1   age            5875 non-null   int64  
 2   sex            5875 non-null   int64  
 3   test_time      5875 non-null   float64
 4   motor_UPDRS    5875 non-null   float64
 5   total_UPDRS    5875 non-null   float64
 6   Jitter(%)      5875 non-null   float64
 7   Jitter(Abs)    5875 non-null   float64
 8   Jitter:RAP     5875 non-null   float64
 9   Jitter:PPQ5    5875 non-null   float64
 10  Jitter:DDP     5875 non-null   float64
 11  Shimmer        5875 non-null   float64
 12  Shimmer(dB)    5875 non-null   float64
 13  Shimmer:APQ3   5875 non-null   float64
 14  Shimmer:APQ5   5875 non-null   float64
 15  Shimmer:APQ11  5875 non-null   float64
 16  Shimmer:DDA    5875 non-null   float64
 17  NHR            5875 non-null   float64
 18  HNR            5875 non-null   float64
 19  RPDE           5875 non-null   float64
 20  DFA            5875 non-null   float64
 21  PPE            5875 non-null   float64
dtypes: float64(19), int64(3)
memory usage: 1009.9 KB
```

Статистика по числовым признакам

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

Out[20]:

	subject#	age	sex	test_time	motor_UPDRS	total_UPDRS
count	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000
mean	21.494128	64.804936	0.317787	92.863722	21.296229	29.018942
std	12.372279	8.821524	0.465656	53.445602	8.129282	10.700283
min	1.000000	36.000000	0.000000	-4.262500	5.037700	7.000000
25%	10.000000	58.000000	0.000000	46.847500	15.000000	21.371000
50%	22.000000	65.000000	0.000000	91.523000	20.871000	27.576000
75%	33.000000	72.000000	1.000000	138.445000	27.596500	36.399000
max	42.000000	85.000000	1.000000	215.490000	39.511000	54.992000

8 rows × 22 columns

Копирование датасета для его дальнейшего преобразования

In [21]:
```
df_reg_clean = df_reg.copy()
```

```
df_reg_clean = df_reg_clean.drop('subject#', axis=1)
```

Выделение признаков и таргета, их разделение на выборки для обучения и тестирования

In [22]:
```
X = df_reg_clean.drop('total_UPDRS', axis=1)
y = df_reg_clean['total_UPDRS']

print(f"Количество признаков: {X.shape[1]}")
print(f"Диапазон целевой переменной: [{y.min():.2f}, {y.max():.2f}]")
print(f"Среднее значение целевой переменной: {y.mean():.2f}")
print(f"Стандартное отклонение целевой переменной: {y.std():.2f}")

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(f"\nРазмеры выборок:")
print(f"  Обучающая выборка: {X_train_reg.shape}")
print(f"  Тестовая выборка: {X_test_reg.shape}")
```

```
Количество признаков: 20
Диапазон целевой переменной: [7.00, 54.99]
Среднее значение целевой переменной: 29.02
Стандартное отклонение целевой переменной: 10.70

Размеры выборок:
  Обучающая выборка: (4112, 20)
  Тестовая выборка: (1763, 20)
```

Масштабирование данных с помощью `StandardScaler`

In [23]:
```
scaler_reg = StandardScaler()
X_train_reg_scaled = scaler_reg.fit_transform(X_train_reg)
X_test_reg_scaled = scaler_reg.transform(X_test_reg)
```

Обучение модели регрессии `KNeighborsRegressor`

In [24]:
```
knn_regressor = KNeighborsRegressor(n_neighbors=5)
knn_regressor.fit(X_train_reg_scaled, y_train_reg)

y_pred_reg = knn_regressor.predict(X_test_reg_scaled)
```

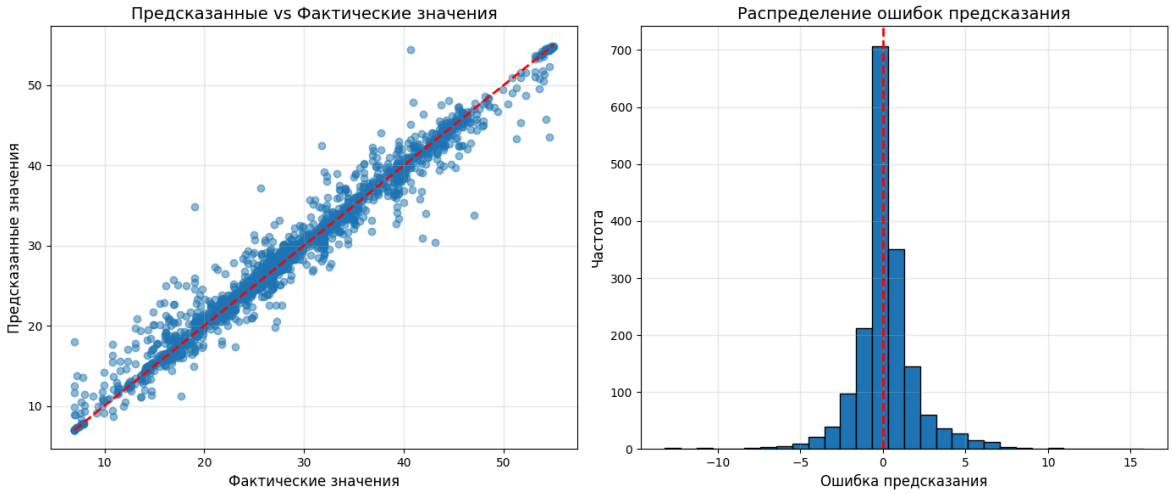
Вычисление метрик

In [25]:
```
mse = mean_squared_error(y_test_reg, y_pred_reg)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)

print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"R²: {r2:.4f}")
```

```
MSE: 4.0934
RMSE: 2.0232
MAE: 1.2271
R²: 0.9636
```

Визуализация предсказаний

In [26]:
```
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

axes[0].scatter(y_test_reg, y_pred_reg, alpha=0.5)
axes[0].plot([y_test_reg.min(), y_test_reg.max()],
             [y_test_reg.min(), y_test_reg.max()],
             'r--', lw=2)
axes[0].set_xlabel('Фактические значения', fontsize=12)
axes[0].set_ylabel('Предсказанные значения', fontsize=12)
axes[0].set_title('Предсказанные vs Фактические значения', fontsize=14)
axes[0].grid(True, alpha=0.3)

errors = y_pred_reg - y_test_reg
axes[1].hist(errors, bins=30, edgecolor='black')
axes[1].axvline(x=0, color='r', linestyle='--', linewidth=2)
axes[1].set_xlabel('Ошибка предсказания', fontsize=12)
axes[1].set_ylabel('Частота', fontsize=12)
axes[1].set_title('Распределение ошибок предсказания', fontsize=14)
axes[1].grid(True, alpha=0.3)

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



Результаты базовых моделей

In [27]:
```
print("\nКлассификация")
print("Метрики:")
print(f"- Accuracy: {accuracy:.4f}")
print(f"- F1-Score: {f1:.4f}")
print(f"- ROC-AUC:  {roc_auc:.4f}")

print("\nРегрессия")
print("Метрики:")
print(f"- MSE:  {mse:.4f}")
print(f"- RMSE: {rmse:.4f}")
print(f"- MAE:  {mae:.4f}")
print(f"- R²:   {r2:.4f}")
```

```
Классификация
Метрики:
- Accuracy: 0.8724
- F1-Score: 0.4720
- ROC-AUC:  0.7952

Регрессия
Метрики:
- MSE:  4.0934
- RMSE: 2.0232
- MAE:  1.2271
- R²:   0.9636
```

Улучшение бейзлайна

In [28]:
```python
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import RobustScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import precision_score, recall_score, roc_curve

from typing import Union, Any

import warnings
```

Классификация

Сохранение метрик базовой модели

In [29]:
```python
class_base_metrics = {
    'Accuracy': accuracy,
    'F1': f1,
    'ROC-AUC': roc_auc,
    'Precision': precision,
    'Recall': recall
}
```

Функция сравнения метрик новой модели с базовой

In [30]:
```python
def print_comparison_class(metrics_old, metrics_new):
    comparison_data = []
    for metric in ['Accuracy', 'F1', 'ROC-AUC', 'Precision', 'Recall']:
        base_val = metrics_old[metric]
        new_val = metrics_new[metric]
        diff = new_val - base_val
        change = "улучшение" if diff > 0 else "ухудшение"

        comparison_data.append({
            'Метрика': metric,
            'Базовая модель': f"{base_val:.4f}",
            'Новая модель': f"{new_val:.4f}",
            'Разница': f"{diff:+.4f}",
            'Изменение': change
        })

    df_comparison = pd.DataFrame(comparison_data)
    print(df_comparison.to_string(index=False))
```

Повторное копирование, разделение и масштабирование данных

In [31]:
```
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le

X = df_class_clean.drop('Revenue', axis=1)
y = df_class_clean['Revenue']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Гипотеза 1: Подбор оптимального значения k

In [32]:
```
k_values = range(1, 31, 2)
best_k = 5
best_f1 = 0

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred_k = knn.predict(X_test_scaled)
    f1_k = f1_score(y_test, y_pred_k)

    if f1_k > best_f1:
        best_f1 = f1_k
        best_k = k

    if k % 5 == 1:
        print(f"k={k}: F1={f1_k:.4f}")

print(f"\nОптимальное k: {best_k} с F1={best_f1:.4f}")

knn_k_opt = KNeighborsClassifier(n_neighbors=best_k)
knn_k_opt.fit(X_train_scaled, y_train)
y_pred_k_opt = knn_k_opt.predict(X_test_scaled)
y_proba_k_opt = knn_k_opt.predict_proba(X_test_scaled)[:, 1]

metrics_k_opt = {
    'Accuracy': accuracy_score(y_test, y_pred_k_opt),
    'F1': f1_score(y_test, y_pred_k_opt),
    'ROC-AUC': roc_auc_score(y_test, y_proba_k_opt),
    'Precision': precision_score(y_test, y_pred_k_opt),
    'Recall': recall_score(y_test, y_pred_k_opt)
}
```

```
print("\nПодбор оптимального k")
print_comparison_class(class_base_metrics, metrics_k_opt)
```

```
k=1: F1=0.4530
k=11: F1=0.4484
k=21: F1=0.4005

Оптимальное k: 3 с F1=0.4765

Подбор оптимального k
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8724       0.8646 -0.0078 ухудшение
       F1         0.4720       0.4765 +0.0045 улучшение
  ROC-AUC         0.7952       0.7552 -0.0400 ухудшение
Precision         0.6553       0.5922 -0.0631 ухудшение
   Recall         0.3689       0.3986 +0.0297 улучшение
```

Гипотеза 2: Выбор метрики расстояния

In [33]:
```
distance_metrics = ['euclidean', 'manhattan', 'minkowski']
best_metric = 'euclidean'
best_f1_metric = 0

for metric in distance_metrics:
    try:
        knn = KNeighborsClassifier(n_neighbors=best_k, metric=metric)
        knn.fit(X_train_scaled, y_train)
        y_pred_m = knn.predict(X_test_scaled)
        f1_m = f1_score(y_test, y_pred_m)
        print(f"Метрика '{metric}': F1={f1_m:.4f}")

        if f1_m > best_f1_metric:
            best_f1_metric = f1_m
            best_metric = metric
    except Exception as e:
        print(f"Ошибка с метрикой '{metric}': {str(e)}")

print(f"\nОптимальная метрика: {best_metric} с F1={best_f1_metric:.4f}")

knn_metric_opt = KNeighborsClassifier(n_neighbors=best_k, metric=best_metric)
knn_metric_opt.fit(X_train_scaled, y_train)
y_pred_metric = knn_metric_opt.predict(X_test_scaled)
y_proba_metric = knn_metric_opt.predict_proba(X_test_scaled)[:, 1]

metrics_metric_opt = {
    'Accuracy': accuracy_score(y_test, y_pred_metric),
    'F1': f1_score(y_test, y_pred_metric),
    'ROC-AUC': roc_auc_score(y_test, y_proba_metric),
    'Precision': precision_score(y_test, y_pred_metric),
    'Recall': recall_score(y_test, y_pred_metric)
}

print("\nВыбор метрики расстояния")
print_comparison_class(class_base_metrics, metrics_metric_opt)
```

```
Метрика 'euclidean': F1=0.4765
Метрика 'manhattan': F1=0.4413
Метрика 'minkowski': F1=0.4765

Оптимальная метрика: euclidean с F1=0.4765

Выбор метрики расстояния
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8724       0.8646 -0.0078 ухудшение
       F1         0.4720       0.4765 +0.0045 улучшение
  ROC-AUC         0.7952       0.7552 -0.0400 ухудшение
Precision         0.6553       0.5922 -0.0631 ухудшение
   Recall         0.3689       0.3986 +0.0297 улучшение
```

Гипотеза 3: Использование взвешенного голоса соседей

In [34]:
```
knn_weighted = KNeighborsClassifier(n_neighbors=best_k, metric=best_metric, weig
knn_weighted.fit(X_train_scaled, y_train)
y_pred_weighted = knn_weighted.predict(X_test_scaled)
y_proba_weighted = knn_weighted.predict_proba(X_test_scaled)[:, 1]

metrics_weighted = {
    'Accuracy': accuracy_score(y_test, y_pred_weighted),
    'F1': f1_score(y_test, y_pred_weighted),
    'ROC-AUC': roc_auc_score(y_test, y_proba_weighted),
    'Precision': precision_score(y_test, y_pred_weighted),
    'Recall': recall_score(y_test, y_pred_weighted)
}

print("Взвешенное голосование соседей (weights='distance')")
print_comparison_class(class_base_metrics, metrics_weighted)
```

```
Взвешенное голосование соседей (weights='distance')
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8724       0.8637 -0.0087 ухудшение
       F1         0.4720       0.4739 +0.0019 улучшение
  ROC-AUC         0.7952       0.7566 -0.0386 ухудшение
Precision         0.6553       0.5881 -0.0672 ухудшение
   Recall         0.3689       0.3969 +0.0280 улучшение
```

Гипотеза 4: Использование RobustScaler (более устойчив к выбросам)

In [35]:
```
robust_scaler = RobustScaler()
X_train_robust = robust_scaler.fit_transform(X_train)
X_test_robust = robust_scaler.transform(X_test)

knn_robust = KNeighborsClassifier(n_neighbors=best_k, metric=best_metric, weight
knn_robust.fit(X_train_robust, y_train)
y_pred_robust = knn_robust.predict(X_test_robust)
y_proba_robust = knn_robust.predict_proba(X_test_robust)[:, 1]

metrics_robust = {
    'Accuracy': accuracy_score(y_test, y_pred_robust),
    'F1': f1_score(y_test, y_pred_robust),
    'ROC-AUC': roc_auc_score(y_test, y_proba_robust),
    'Precision': precision_score(y_test, y_pred_robust),
    'Recall': recall_score(y_test, y_pred_robust)
}
```

```
print("RobustScaler с оптимальными параметрами k и metric")
print_comparison_class(class_base_metrics, metrics_robust)
```

```
RobustScaler с оптимальными параметрами k и metric
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8724       0.8737 +0.0014 улучшение
       F1         0.4720       0.5470 +0.0750 улучшение
  ROC-AUC         0.7952       0.8152 +0.0200 улучшение
Precision         0.6553       0.6144 -0.0409 ухудшение
   Recall         0.3689       0.4930 +0.1241 улучшение
```

Гипотеза 5: Комплексный подбор гиперпараметров (GridSearch)

In [36]:
```
warnings.filterwarnings('ignore')

param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11, 15, 21],
    'metric': ['euclidean', 'manhattan', 'minkowski'],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}

knn_gs = KNeighborsClassifier()
grid_search = GridSearchCV(
    estimator=knn_gs, param_grid=param_grid, cv=5,
    scoring='f1', n_jobs=-1
)
grid_search.fit(X_train_robust, y_train)

print("Лучшие параметры:")
for param, value in grid_search.best_params_.items():
    print(f"  {param}: {value}")

best_knn: Union[KNeighborsClassifier, Any] = grid_search.best_estimator_
y_pred_gs = best_knn.predict(X_test_robust)
y_proba_gs = best_knn.predict_proba(X_test_robust)[:, 1]

metrics_gs = {
    'Accuracy': accuracy_score(y_test, y_pred_gs),
    'F1': f1_score(y_test, y_pred_gs),
    'ROC-AUC': roc_auc_score(y_test, y_proba_gs),
    'Precision': precision_score(y_test, y_pred_gs),
    'Recall': recall_score(y_test, y_pred_gs)
}

print("Подбор гиперпараметров (GridSearch)")
print_comparison_class(class_base_metrics, metrics_gs)
```

```
Лучшие параметры:
  metric: euclidean
  n_neighbors: 15
  p: 1
  weights: distance
Подбор гиперпараметров (GridSearch)
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8724       0.8870 +0.0146 улучшение
       F1         0.4720       0.5820 +0.1100 улучшение
  ROC-AUC         0.7952       0.8754 +0.0802 улучшение
Precision         0.6553       0.6799 +0.0246 улучшение
   Recall         0.3689       0.5087 +0.1399 улучшение
```

Формирование улучшенной модели и её обучение

In [37]:
```
best_params = grid_search.best_params_.copy()

improved_knn = KNeighborsClassifier(**best_params)
improved_knn.fit(X_train_robust, y_train)

y_pred_improved = improved_knn.predict(X_test_robust)
y_pred_proba_improved = improved_knn.predict_proba(X_test_robust)[:, 1]
```

In [38]:
```
class_improved_metrics = {
    'Accuracy': accuracy_score(y_test, y_pred_improved),
    'F1': f1_score(y_test, y_pred_improved),
    'ROC-AUC': roc_auc_score(y_test, y_pred_proba_improved),
    'Precision': precision_score(y_test, y_pred_improved),
    'Recall': recall_score(y_test, y_pred_improved)
}

for metric, value in class_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8870
F1: 0.5820
ROC-AUC: 0.8754
Precision: 0.6799
Recall: 0.5087
```

Сравнение улучшенной модели с базовой

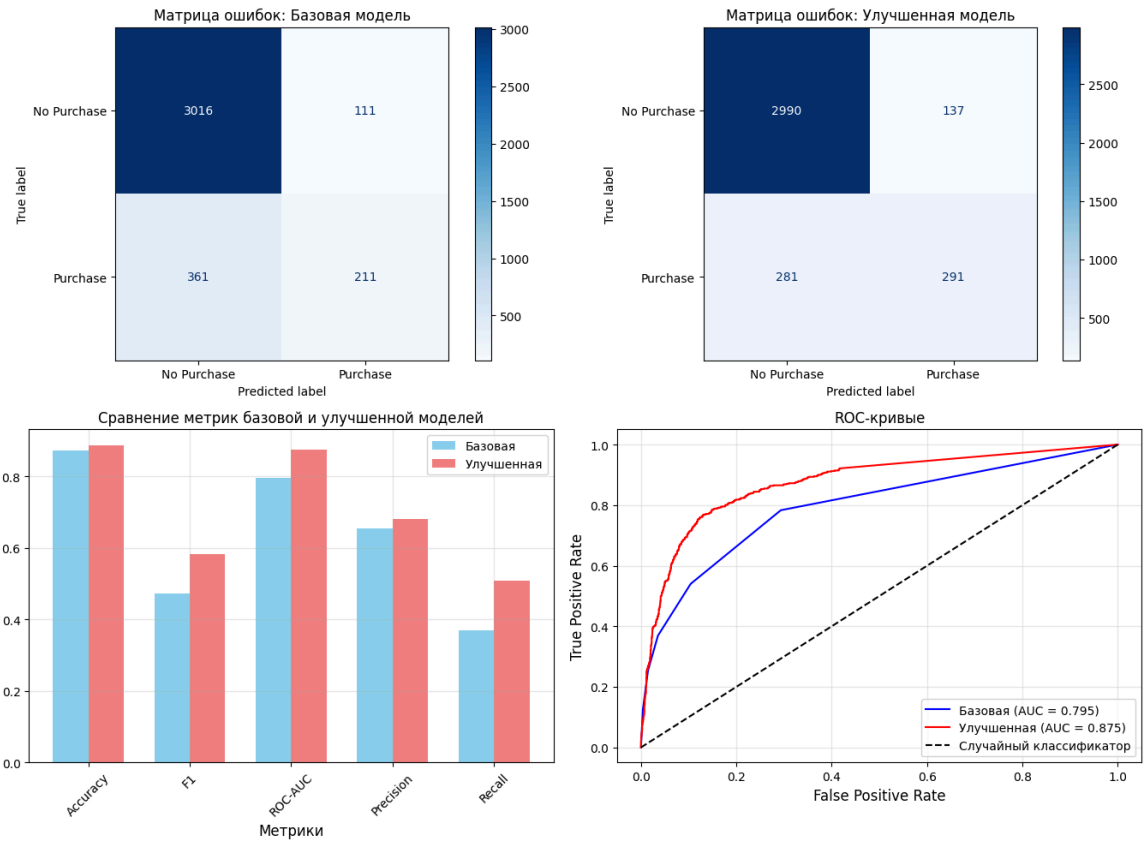
In [39]:
```
print_comparison_class(class_base_metrics, class_improved_metrics)
```

```
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8724       0.8870 +0.0146 улучшение
       F1         0.4720       0.5820 +0.1100 улучшение
  ROC-AUC         0.7952       0.8754 +0.0802 улучшение
Precision         0.6553       0.6799 +0.0246 улучшение
   Recall         0.3689       0.5087 +0.1399 улучшение
```

Визуальное сравнение базовой и улучшенной модели

In [40]:
```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes[0, 0].set_title('Матрица ошибок: Базовая модель', fontsize=12)
cm_base = confusion_matrix(y_test, y_pred)
disp_base = ConfusionMatrixDisplay(confusion_matrix=cm_base,
                                   display_labels=['No Purchase', 'Purchase'])
disp_base.plot(ax=axes[0, 0], cmap='Blues')

axes[0, 1].set_title('Матрица ошибок: Улучшенная модель', fontsize=12)
cm_improved = confusion_matrix(y_test, y_pred_improved)
disp_improved = ConfusionMatrixDisplay(confusion_matrix=cm_improved,
                                       display_labels=['No Purchase', 'Purchase'
disp_improved.plot(ax=axes[0, 1], cmap='Blues')

metrics_names = ['Accuracy', 'F1', 'ROC-AUC', 'Precision', 'Recall']
base_values = [class_base_metrics[m] for m in metrics_names]
improved_values = [class_improved_metrics[m] for m in metrics_names]

x = np.arange(len(metrics_names))
```

```
width = 0.35

axes[1, 0].bar(x - width/2, base_values, width, label='Базовая', color='skyblue'
axes[1, 0].bar(x + width/2, improved_values, width, label='Улучшенная', color='l
axes[1, 0].set_xlabel('Метрики', fontsize=12)
axes[1, 0].set_ylabel('Значение', fontsize=12)
axes[1, 0].set_title('Сравнение метрик базовой и улучшенной моделей', fontsize=1
axes[1, 0].set_xticks(x)
axes[1, 0].set_xticklabels(metrics_names, rotation=45)
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

fpr_base, tpr_base, _ = roc_curve(y_test, y_pred_proba)
fpr_improved, tpr_improved, _ = roc_curve(y_test, y_pred_proba_improved)

axes[1, 1].plot(fpr_base, tpr_base, label=f'Базовая (AUC = {roc_auc:.3f})', colo
axes[1, 1].plot(fpr_improved, tpr_improved, label=f'Улучшенная (AUC = {class_imp
axes[1, 1].plot([0, 1], [0, 1], 'k--', label='Случайный классификатор')
axes[1, 1].set_xlabel('False Positive Rate', fontsize=12)
axes[1, 1].set_ylabel('True Positive Rate', fontsize=12)
axes[1, 1].set_title('ROC-кривые', fontsize=12)
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

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



Анализ результатов классификации

In [41]:
```
TN_base, FP_base, FN_base, TP_base = cm_base.ravel()
TN_imp, FP_imp, FN_imp, TP_imp = cm_improved.ravel()

print("\n1. Анализ обнаружения покупок:")
print(f"   Базовая модель нашла {TP_base} из {TP_base + FN_base} реальных покупо
```

```
print(f"   Улучшенная модель нашла {TP_imp} из {TP_imp + FN_imp} реальных покупо
print(f"   Улучшение в обнаружении покупок: {TP_imp - TP_base} реальных покупок"
print(f"   Процентное улучшение Recall: {((TP_imp / (TP_imp + FN_imp)) - (TP_bas

print("\n2. Анализ ложных срабатываний:")
print(f"   Базовая модель: {FP_base} ложных предсказаний покупки")
print(f"   Улучшенная модель: {FP_imp} ложных предсказаний покупки")
print(f"   Изменение: {FP_imp - FP_base} дополнительных ложных срабатываний")
```

```
1. Анализ обнаружения покупок:
   Базовая модель нашла 211 из 572 реальных покупок
   Улучшенная модель нашла 291 из 572 реальных покупок
   Улучшение в обнаружении покупок: 80 реальных покупок
   Процентное улучшение Recall: +14.0%

2. Анализ ложных срабатываний:
   Базовая модель: 111 ложных предсказаний покупки
   Улучшенная модель: 137 ложных предсказаний покупки
   Изменение: 26 дополнительных ложных срабатываний
```

Регрессия

Сохранение метрик базовой модели

In [42]:
```
reg_base_metrics = {
    'MSE': mse,
    'RMSE': rmse,
    'MAE': mae,
    'R²': r2
}
```

Функция сравнения метрик новой модели с базовой

In [43]:
```
def print_comparison_reg(metrics_old, metrics_new):
    comparison_data = []
    for metric in ['MSE', 'RMSE', 'MAE', 'R²']:
        base_val = metrics_old[metric]
        new_val = metrics_new[metric]
        diff = new_val - base_val
        if metric == 'R²':
            change = "улучшение" if diff > 0 else "ухудшение"
        else:
            change = "улучшение" if diff < 0 else "ухудшение"

        comparison_data.append({
            'Метрика': metric,
            'Базовая модель': f"{base_val:.4f}",
            'Новая модель': f"{new_val:.4f}",
            'Разница': f"{diff:+.4f}",
            'Изменение': change
        })

    df_comparison = pd.DataFrame(comparison_data)
    print(df_comparison.to_string(index=False))
```

Повторное копирование и подготовка данных

In [44]:
```
df_reg_clean = df_reg.copy()
df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X = df_reg_clean.drop('total_UPDRS', axis=1)
y = df_reg_clean['total_UPDRS']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Гипотеза 1: Подбор оптимального k

In [45]:
```
k_values = range(1, 31, 2)
best_k_reg = 5
best_r2 = 0

for k in k_values:
    knn = KNeighborsRegressor(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred_k = knn.predict(X_test_scaled)
    r2_k = r2_score(y_test, y_pred_k)

    if r2_k > best_r2:
        best_r2 = r2_k
        best_k_reg = k

    if k % 5 == 1:
        print(f"k={k}: R²={r2_k:.4f}")

print(f"\nОптимальное k: {best_k_reg} с R²={best_r2:.4f}")

knn_k_opt_reg = KNeighborsRegressor(n_neighbors=best_k_reg)
knn_k_opt_reg.fit(X_train_scaled, y_train)
y_pred_k_opt_reg = knn_k_opt_reg.predict(X_test_scaled)

metrics_k_opt_reg = {
    'MSE': mean_squared_error(y_test, y_pred_k_opt_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_k_opt_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_k_opt_reg),
    'R²': r2_score(y_test, y_pred_k_opt_reg)
}

print("\nПодбор оптимального k")
print_comparison_reg(reg_base_metrics, metrics_k_opt_reg)
```

```
k=1: R²=0.9457
k=11: R²=0.9556
k=21: R²=0.9433

Оптимальное k: 3 с R²=0.9648

Подбор оптимального k
Метрика Базовая модель Новая модель Разница Изменение
    MSE         4.0934       3.9649 -0.1285 улучшение
   RMSE         2.0232       1.9912 -0.0320 улучшение
    MAE         1.2271       1.1723 -0.0548 улучшение
     R²         0.9636       0.9648 +0.0011 улучшение
```

Гипотеза 2: Выбор метрики расстояния

In [46]:
```python
distance_metrics = ['euclidean', 'manhattan', 'minkowski']
best_metric_reg = 'euclidean'
best_r2_metric = 0

for metric in distance_metrics:
    try:
        knn = KNeighborsRegressor(n_neighbors=best_k_reg, metric=metric)
        knn.fit(X_train_scaled, y_train)
        y_pred_m = knn.predict(X_test_scaled)
        r2_m = r2_score(y_test, y_pred_m)
        print(f"Метрика '{metric}': R²={r2_m:.4f}")

        if r2_m > best_r2_metric:
            best_r2_metric = r2_m
            best_metric_reg = metric
    except Exception as e:
        print(f"Ошибка с метрикой '{metric}': {str(e)}")

print(f"\nОптимальная метрика: {best_metric_reg} с R²={best_r2_metric:.4f}")

knn_metric_opt_reg = KNeighborsRegressor(n_neighbors=best_k_reg, metric=best_met
knn_metric_opt_reg.fit(X_train_scaled, y_train)
y_pred_metric_reg = knn_metric_opt_reg.predict(X_test_scaled)

metrics_metric_opt_reg = {
    'MSE': mean_squared_error(y_test, y_pred_metric_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_metric_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_metric_reg),
    'R²': r2_score(y_test, y_pred_metric_reg)
}

print("\nВыбор метрики расстояния")
print_comparison_reg(reg_base_metrics, metrics_metric_opt_reg)
```

```
Метрика 'euclidean': R²=0.9648
Метрика 'manhattan': R²=0.9611
Метрика 'minkowski': R²=0.9648

Оптимальная метрика: euclidean с R²=0.9648

Выбор метрики расстояния
Метрика Базовая модель Новая модель Разница Изменение
    MSE         4.0934       3.9649 -0.1285 улучшение
   RMSE         2.0232       1.9912 -0.0320 улучшение
    MAE         1.2271       1.1723 -0.0548 улучшение
     R²         0.9636       0.9648 +0.0011 улучшение
```

Гипотеза 3: Использование взвешенного голоса соседей

In [47]:
```
knn_weighted_reg = KNeighborsRegressor(n_neighbors=best_k_reg, metric=best_metri
knn_weighted_reg.fit(X_train_scaled, y_train)
y_pred_weighted_reg = knn_weighted_reg.predict(X_test_scaled)

metrics_weighted_reg = {
    'MSE': mean_squared_error(y_test, y_pred_weighted_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_weighted_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_weighted_reg),
    'R²': r2_score(y_test, y_pred_weighted_reg)
}

print("Взвешенное голосование соседей (weights='distance')")
print_comparison_reg(reg_base_metrics, metrics_weighted_reg)
```

```
Взвешенное голосование соседей (weights='distance')
Метрика Базовая модель Новая модель Разница Изменение
    MSE         4.0934       3.7332 -0.3602 улучшение
   RMSE         2.0232       1.9321 -0.0911 улучшение
    MAE         1.2271       1.1304 -0.0968 улучшение
     R²         0.9636       0.9668 +0.0032 улучшение
```

Гипотеза 4: Использование RobustScaler (более устойчив к выбросам)

In [48]:
```
robust_scaler_reg = RobustScaler()
X_train_robust_reg = robust_scaler_reg.fit_transform(X_train)
X_test_robust_reg = robust_scaler_reg.transform(X_test)

knn_robust_reg = KNeighborsRegressor(n_neighbors=best_k_reg, metric=best_metric_
knn_robust_reg.fit(X_train_robust_reg, y_train)
y_pred_robust_reg = knn_robust_reg.predict(X_test_robust_reg)

metrics_robust_reg = {
    'MSE': mean_squared_error(y_test, y_pred_robust_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_robust_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_robust_reg),
    'R²': r2_score(y_test, y_pred_robust_reg)
}

print("RobustScaler с оптимальными параметрами k и metric")
print_comparison_reg(reg_base_metrics, metrics_robust_reg)
```

```
RobustScaler с оптимальными параметрами k и metric
Метрика Базовая модель Новая модель Разница Изменение
    MSE         4.0934       8.5705 +4.4772 ухудшение
   RMSE         2.0232       2.9275 +0.9043 ухудшение
    MAE         1.2271       1.8722 +0.6451 ухудшение
     R²         0.9636       0.9238 -0.0398 ухудшение
```

Гипотеза 5: Комплексный подбор гиперпараметров (GridSearch)

In [49]:
```python
warnings.filterwarnings('ignore')

param_grid_reg = {
    'n_neighbors': [3, 5, 7, 9, 11, 15, 21],
    'metric': ['euclidean', 'manhattan', 'minkowski'],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}

knn_gs_reg = KNeighborsRegressor()
grid_search_reg = GridSearchCV(
    estimator=knn_gs_reg, param_grid=param_grid_reg, cv=5,
    scoring='r2', n_jobs=-1
)
grid_search_reg.fit(X_train_robust_reg, y_train)

print("Лучшие параметры:")
for param, value in grid_search_reg.best_params_.items():
    print(f"  {param}: {value}")

best_knn_reg: Union[KNeighborsRegressor, Any] = grid_search_reg.best_estimator_
y_pred_gs_reg = best_knn_reg.predict(X_test_robust_reg)

metrics_gs_reg = {
    'MSE': mean_squared_error(y_test, y_pred_gs_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_gs_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_gs_reg),
    'R²': r2_score(y_test, y_pred_gs_reg)
}

print("Подбор гиперпараметров (GridSearch)")
print_comparison_reg(reg_base_metrics, metrics_gs_reg)
```

```
Лучшие параметры:
  metric: euclidean
  n_neighbors: 3
  p: 1
  weights: distance
Подбор гиперпараметров (GridSearch)
Метрика Базовая модель Новая модель Разница Изменение
    MSE         4.0934       8.5705 +4.4772 ухудшение
   RMSE         2.0232       2.9275 +0.9043 ухудшение
    MAE         1.2271       1.8722 +0.6451 ухудшение
     R²         0.9636       0.9238 -0.0398 ухудшение
```

Формирование улучшенной модели и её обучение

In [50]:
```python
best_params_reg = grid_search_reg.best_params_.copy()

improved_knn_reg = KNeighborsRegressor(**best_params_reg)
```

```
improved_knn_reg.fit(X_train_robust_reg, y_train)
y_pred_improved_reg = improved_knn_reg.predict(X_test_robust_reg)
```

Метрики улучшенной модели

In [51]:
```
reg_improved_metrics = {
    'MSE': mean_squared_error(y_test, y_pred_improved_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_improved_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_improved_reg),
    'R²': r2_score(y_test, y_pred_improved_reg)
}

for metric, value in reg_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 8.5705
RMSE: 2.9275
MAE: 1.8722
R²: 0.9238
```

Сравнение улучшенной модели с базовой

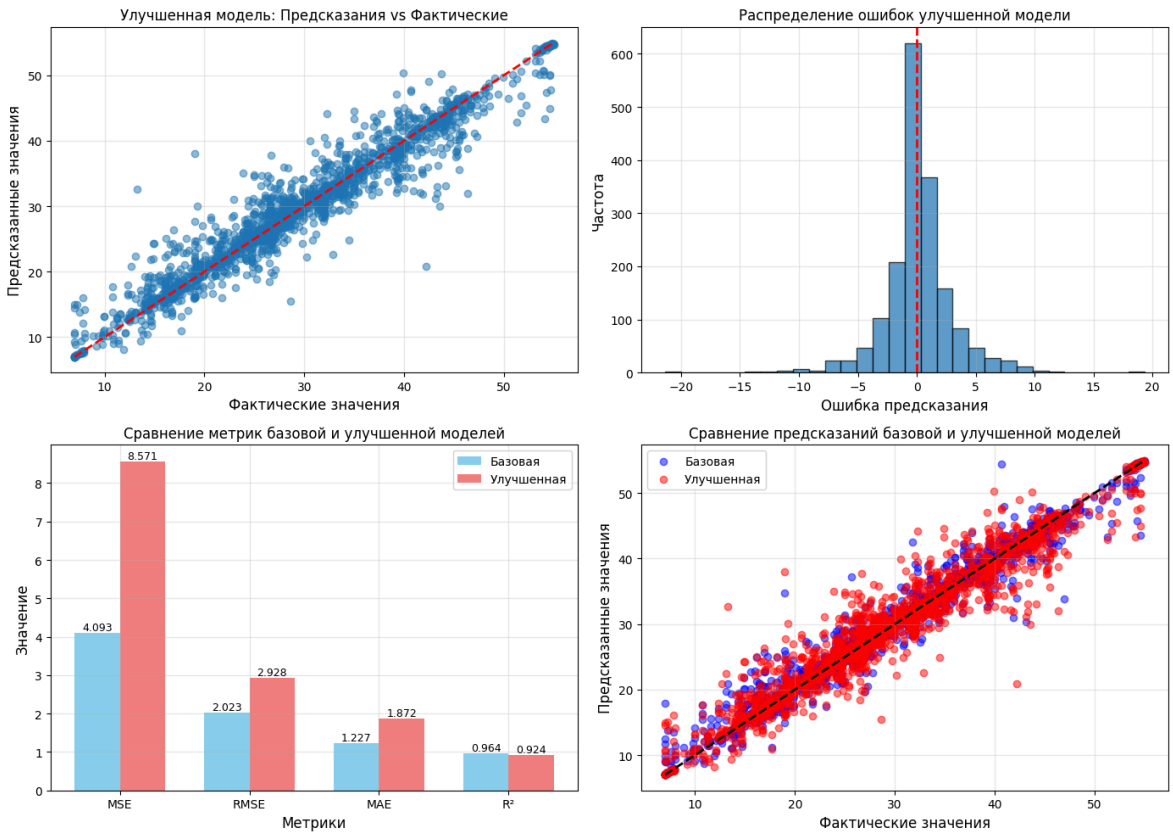
In [52]:
```
print_comparison_reg(reg_base_metrics, reg_improved_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE         4.0934       8.5705 +4.4772 ухудшение
   RMSE         2.0232       2.9275 +0.9043 ухудшение
    MAE         1.2271       1.8722 +0.6451 ухудшение
     R²         0.9636       0.9238 -0.0398 ухудшение
```

Визуальное сравнение базовой и улучшенной модели

In [53]:
```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes[0, 0].scatter(y_test, y_pred_improved_reg, alpha=0.5)
axes[0, 0].plot([y_test.min(), y_test.max()],
                [y_test.min(), y_test.max()],
                'r--', lw=2)
axes[0, 0].set_xlabel('Фактические значения', fontsize=12)
axes[0, 0].set_ylabel('Предсказанные значения', fontsize=12)
axes[0, 0].set_title('Улучшенная модель: Предсказания vs Фактические', fontsize=
axes[0, 0].grid(True, alpha=0.3)

errors_improved = y_pred_improved_reg - y_test
axes[0, 1].hist(errors_improved, bins=30, edgecolor='black', alpha=0.7)
axes[0, 1].axvline(x=0, color='r', linestyle='--', linewidth=2)
axes[0, 1].set_xlabel('Ошибка предсказания', fontsize=12)
axes[0, 1].set_ylabel('Частота', fontsize=12)
axes[0, 1].set_title('Распределение ошибок улучшенной модели', fontsize=12)
axes[0, 1].grid(True, alpha=0.3)

metrics_names = ['MSE', 'RMSE', 'MAE', 'R²']
base_values = [reg_base_metrics[m] for m in metrics_names]
improved_values = [reg_improved_metrics[m] for m in metrics_names]

x = np.arange(len(metrics_names))
width = 0.35

bars1 = axes[1, 0].bar(x - width/2, base_values, width, label='Базовая', color='
```

```
bars2 = axes[1, 0].bar(x + width/2, improved_values, width, label='Улучшенная',
axes[1, 0].set_xlabel('Метрики', fontsize=12)
axes[1, 0].set_ylabel('Значение', fontsize=12)
axes[1, 0].set_title('Сравнение метрик базовой и улучшенной моделей', fontsize=1
axes[1, 0].set_xticks(x)
axes[1, 0].set_xticklabels(metrics_names)
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

for bar in bars1 + bars2:
    height = bar.get_height()
    axes[1, 0].text(bar.get_x() + bar.get_width()/2., height,
                    f'{height:.3f}', ha='center', va='bottom', fontsize=9)

axes[1, 1].scatter(y_test, y_pred_reg, alpha=0.5, label='Базовая', color='blue')
axes[1, 1].scatter(y_test, y_pred_improved_reg, alpha=0.5, label='Улучшенная', c
axes[1, 1].plot([y_test.min(), y_test.max()],
                [y_test.min(), y_test.max()],
                'k--', lw=2)
axes[1, 1].set_xlabel('Фактические значения', fontsize=12)
axes[1, 1].set_ylabel('Предсказанные значения', fontsize=12)
axes[1, 1].set_title('Сравнение предсказаний базовой и улучшенной моделей', font
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

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



Анализ результатов регрессии

In [54]:
```
print(f"\n1. Анализ улучшения модели:")
print(f"   R² увеличился с {reg_base_metrics['R²']:.4f} до {reg_improved_metrics
print(f"   Улучшение R²: {reg_improved_metrics['R²'] - reg_base_metrics['R²']:+.
print(f"   MSE уменьшился с {reg_base_metrics['MSE']:.4f} до {reg_improved_metri
```

```
print(f"   Улучшение MSE: {reg_base_metrics['MSE'] - reg_improved_metrics['MSE']

print(f"\n2. Статистика ошибок улучшенной модели:")
print(f"   Средняя абсолютная ошибка: {reg_improved_metrics['MAE']:.2f}")
print(f"   Средняя ошибка в процентах от среднего target: {reg_improved_metrics[
print(f"   Стандартное отклонение ошибок: {np.std(errors_improved):.2f}")
```

```
1. Анализ улучшения модели:
   R² увеличился с 0.9636 до 0.9238
   Улучшение R²: -0.0398
   MSE уменьшился с 4.0934 до 8.5705
   Улучшение MSE: -4.4772 (-109.38%)

2. Статистика ошибок улучшенной модели:
   Средняя абсолютная ошибка: 1.87
   Средняя ошибка в процентах от среднего target: 6.44%
   Стандартное отклонение ошибок: 2.92
```

Имплементация алгоритма машинного обучения

Классификация

Кастомная модель KNN классификатора

In [55]:
```
class CustomKNNClassifier:

    def __init__(self, n_neighbors=5, metric='euclidean', weights='uniform', p=2
        self.n_neighbors = n_neighbors
        self.metric = metric
        self.weights = weights
        self.p = p
        self.X_train = None
        self.y_train = None

    def _distance(self, x1, x2):
        if self.metric == 'euclidean':
            return np.sqrt(np.sum((x1 - x2) ** 2))
        elif self.metric == 'manhattan':
            return np.sum(np.abs(x1 - x2))
        elif self.metric == 'minkowski':
            return np.power(np.sum(np.power(np.abs(x1 - x2), self.p)), 1.0 / sel
        else:
            raise ValueError(f"Неизвестная метрика: {self.metric}")

    def _compute_distances(self, X):
        distances = []
        for x in X:
            dists = [self._distance(x, x_train) for x_train in self.X_train]
            distances.append(dists)
        return np.array(distances)

    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)
        return self
```

```
    def predict_proba(self, X):
        X = np.array(X)
        n_samples = X.shape[0]
        classes = np.unique(self.y_train)
        n_classes = len(classes)
        probabilities = np.zeros((n_samples, n_classes))

        distances = self._compute_distances(X)

        for i in range(n_samples):
            k_nearest_indices = np.argsort(distances[i])[:self.n_neighbors]
            k_nearest_distances = distances[i][k_nearest_indices]
            k_nearest_labels = self.y_train[k_nearest_indices]

            if self.weights == 'uniform':
                weights = np.ones(self.n_neighbors)
            elif self.weights == 'distance':
                epsilon = 1e-10
                weights = 1.0 / (k_nearest_distances + epsilon)
            else:
                weights = np.ones(self.n_neighbors)

            weights = weights / np.sum(weights)

            for j, cls in enumerate(classes):
                mask = (k_nearest_labels == cls)
                probabilities[i, j] = np.sum(weights[mask])

        return probabilities

    def predict(self, X):
        probabilities = self.predict_proba(X)
        classes = np.unique(self.y_train)
        return classes[np.argmax(probabilities, axis=1)]
```

Повторное копирование и разбиение данных

In [56]:
```
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le

X_class = df_class_clean.drop('Revenue', axis=1)
y_class = df_class_clean['Revenue']

X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class, test_size=0.3, random_state=42, stratify=y_class
)

scaler_class = StandardScaler()
X_train_class_scaled = scaler_class.fit_transform(X_train_class)
X_test_class_scaled = scaler_class.transform(X_test_class)
```

Обучение кастомного KNN классификатора

In [57]:
```
custom_knn_class = CustomKNNClassifier(
    n_neighbors=5,
    metric='euclidean',
    weights='uniform'
)

custom_knn_class.fit(X_train_class_scaled, y_train_class)

y_pred_custom_class = custom_knn_class.predict(X_test_class_scaled)
y_pred_proba_custom_class = custom_knn_class.predict_proba(X_test_class_scaled)[
```

Метрики кастомного KNN классификатора

In [58]:
```
custom_class_base_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_custom_class),
    'F1': f1_score(y_test_class, y_pred_custom_class),
    'ROC-AUC': roc_auc_score(y_test_class, y_pred_proba_custom_class),
    'Precision': precision_score(y_test_class, y_pred_custom_class),
    'Recall': recall_score(y_test_class, y_pred_custom_class)
}

for metric, value in custom_class_base_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8724
F1: 0.4720
ROC-AUC: 0.7952
Precision: 0.6553
Recall: 0.3689
```

Сравнение кастомной модели с базовой из sklearn

In [59]:
```
print_comparison_class(class_base_metrics, custom_class_base_metrics)
```

```
   Метрика Базовая модель Новая модель Разница Изменение
  Accuracy         0.8724       0.8724 +0.0000 ухудшение
        F1         0.4720       0.4720 +0.0000 ухудшение
   ROC-AUC         0.7952       0.7952 +0.0000 ухудшение
 Precision         0.6553       0.6553 +0.0000 ухудшение
    Recall         0.3689       0.3689 +0.0000 ухудшение
```

Обучение улучшенной кастомной KNN модели с техниками из улучшенного бейзлайна

In [60]:
```
robust_scaler_class = RobustScaler()
X_train_robust_class = robust_scaler_class.fit_transform(X_train_class)
X_test_robust_class = robust_scaler_class.transform(X_test_class)

improved_custom_knn_class = CustomKNNClassifier(
    n_neighbors=best_params.get('n_neighbors', 5),
    metric=best_params.get('metric', 'euclidean'),
    weights=best_params.get('weights', 'uniform'),
    p=best_params.get('p', 2)
)

improved_custom_knn_class.fit(X_train_robust_class, y_train_class)
```

```
y_pred_imp_custom_class = improved_custom_knn_class.predict(X_test_robust_class)
y_pred_proba_imp_custom_class = improved_custom_knn_class.predict_proba(X_test_r
```

Метрики улучшенной кастомной KNN модели

In [61]:
```
custom_improved_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_imp_custom_class),
    'F1': f1_score(y_test_class, y_pred_imp_custom_class),
    'ROC-AUC': roc_auc_score(y_test_class, y_pred_proba_imp_custom_class),
    'Precision': precision_score(y_test_class, y_pred_imp_custom_class),
    'Recall': recall_score(y_test_class, y_pred_imp_custom_class)
}

for metric, value in custom_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8870
F1: 0.5820
ROC-AUC: 0.8754
Precision: 0.6799
Recall: 0.5087
```

Сравнение улучшенной кастомной модели с улучшенной из sklearn

In [62]:
```
print_comparison_class(class_improved_metrics, custom_improved_metrics)
```

```
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8870       0.8870 +0.0000 ухудшение
       F1         0.5820       0.5820 +0.0000 ухудшение
  ROC-AUC         0.8754       0.8754 -0.0000 ухудшение
Precision         0.6799       0.6799 +0.0000 ухудшение
   Recall         0.5087       0.5087 +0.0000 ухудшение
```

Итоговое сравнение всех моделей классификации

In [63]:
```
summary_class = pd.DataFrame({
    'Тип модели': ['Базовая (sklearn)', 'Улучшенная (sklearn)', 'Кастомная (базо
    'Accuracy': [class_base_metrics['Accuracy'], class_improved_metrics['Accurac
    'F1-Score': [class_base_metrics['F1'], class_improved_metrics['F1'], custom_
    'ROC-AUC': [class_base_metrics['ROC-AUC'], class_improved_metrics['ROC-AUC']
    'Recall': [class_base_metrics['Recall'], class_improved_metrics['Recall'], c
})

print("Сводная таблица моделей классификации")
print(summary_class.to_string(index=False))
```

```
Сводная таблица моделей классификации
            Тип модели  Accuracy  F1-Score  ROC-AUC   Recall
     Базовая (sklearn)  0.872398  0.472036 0.795218 0.368881
  Улучшенная (sklearn)  0.886996  0.582000 0.875401 0.508741
   Кастомная (базовая)  0.872398  0.472036 0.795218 0.368881
Кастомная (улучшенная)  0.886996  0.582000 0.875376 0.508741
```

Регрессия

Кастомная модель KNN регрессора

In [64]:
```
class CustomKNNRegressor:
```

```
    def __init__(self, n_neighbors=5, metric='euclidean', weights='uniform', p=2
        self.n_neighbors = n_neighbors
        self.metric = metric
        self.weights = weights
        self.p = p
        self.X_train = None
        self.y_train = None

    def _distance(self, x1, x2):
        if self.metric == 'euclidean':
            return np.sqrt(np.sum((x1 - x2) ** 2))
        elif self.metric == 'manhattan':
            return np.sum(np.abs(x1 - x2))
        elif self.metric == 'minkowski':
            return np.power(np.sum(np.power(np.abs(x1 - x2), self.p)), 1.0 / sel
        else:
            raise ValueError(f"Неизвестная метрика: {self.metric}")

    def _compute_distances(self, X):
        distances = []
        for x in X:
            dists = [self._distance(x, x_train) for x_train in self.X_train]
            distances.append(dists)
        return np.array(distances)

    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)
        return self

    def predict(self, X):
        X = np.array(X)
        n_samples = X.shape[0]
        predictions = np.zeros(n_samples)

        distances = self._compute_distances(X)

        for i in range(n_samples):
            k_nearest_indices = np.argsort(distances[i])[:self.n_neighbors]
            k_nearest_distances = distances[i][k_nearest_indices]
            k_nearest_values = self.y_train[k_nearest_indices]

            if self.weights == 'uniform':
                weights = np.ones(self.n_neighbors)
            elif self.weights == 'distance':
                epsilon = 1e-10
                weights = 1.0 / (k_nearest_distances + epsilon)
            else:
                weights = np.ones(self.n_neighbors)

            weights = weights / np.sum(weights)

            predictions[i] = np.sum(weights * k_nearest_values)

        return predictions
```

Повторное копирование и разбиение данных

In [65]:
```
df_reg_clean = df_reg.copy()
df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X_reg = df_reg_clean.drop('total_UPDRS', axis=1)
y_reg = df_reg_clean['total_UPDRS']

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.3, random_state=42
)

scaler_reg = StandardScaler()
X_train_reg_scaled = scaler_reg.fit_transform(X_train_reg)
X_test_reg_scaled = scaler_reg.transform(X_test_reg)
```

Обучение кастомного KNN регрессора

In [66]:
```
custom_knn_reg = CustomKNNRegressor(
    n_neighbors=5,
    metric='euclidean',
    weights='uniform'
)

custom_knn_reg.fit(X_train_reg_scaled, y_train_reg)

y_pred_custom_reg = custom_knn_reg.predict(X_test_reg_scaled)
```

Метрики кастомного KNN регрессора

In [67]:
```
custom_reg_base_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_custom_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test_reg, y_pred_custom_reg)),
    'MAE': mean_absolute_error(y_test_reg, y_pred_custom_reg),
    'R²': r2_score(y_test_reg, y_pred_custom_reg)
}

for metric, value in custom_reg_base_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 4.0934
RMSE: 2.0232
MAE: 1.2271
R²: 0.9636
```

Сравнение кастомной модели с базовой из sklearn

In [68]:
```
print_comparison_reg(reg_base_metrics, custom_reg_base_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE         4.0934       4.0934 +0.0000 ухудшение
   RMSE         2.0232       2.0232 +0.0000 ухудшение
    MAE         1.2271       1.2271 -0.0000 улучшение
     R²         0.9636       0.9636 +0.0000 ухудшение
```

Обучение улучшенной кастомной KNN модели с техниками из улучшенного бейзлайна

In [69]:
```
robust_scaler_reg_custom = RobustScaler()
X_train_robust_reg_custom = robust_scaler_reg_custom.fit_transform(X_train_reg)
```

```
X_test_robust_reg_custom = robust_scaler_reg_custom.transform(X_test_reg)

improved_custom_knn_reg = CustomKNNRegressor(
    n_neighbors=best_params_reg.get('n_neighbors', 5),
    metric=best_params_reg.get('metric', 'euclidean'),
    weights=best_params_reg.get('weights', 'uniform'),
    p=best_params_reg.get('p', 2)
)

improved_custom_knn_reg.fit(X_train_robust_reg_custom, y_train_reg)

y_pred_imp_custom_reg = improved_custom_knn_reg.predict(X_test_robust_reg_custom
```

Метрики улучшенной кастомной KNN модели

In [70]:
```
custom_reg_improved_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_imp_custom_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test_reg, y_pred_imp_custom_reg)),
    'MAE': mean_absolute_error(y_test_reg, y_pred_imp_custom_reg),
    'R²': r2_score(y_test_reg, y_pred_imp_custom_reg)
}

for metric, value in custom_reg_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 8.5705
RMSE: 2.9275
MAE: 1.8722
R²: 0.9238
```

Сравнение улучшенной кастомной модели с улучшенной из sklearn

In [71]:
```
print_comparison_reg(reg_improved_metrics, custom_reg_improved_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE         8.5705       8.5705 +0.0000 ухудшение
   RMSE         2.9275       2.9275 +0.0000 ухудшение
    MAE         1.8722       1.8722 +0.0000 ухудшение
     R²         0.9238       0.9238 -0.0000 ухудшение
```

Итоговое сравнение всех моделей регрессии

In [72]:
```
summary_reg = pd.DataFrame({
    'Тип модели': ['Базовая (sklearn)', 'Улучшенная (sklearn)', 'Кастомная (базо
    'MSE': [reg_base_metrics['MSE'], reg_improved_metrics['MSE'], custom_reg_bas
    'RMSE': [reg_base_metrics['RMSE'], reg_improved_metrics['RMSE'], custom_reg_
    'MAE': [reg_base_metrics['MAE'], reg_improved_metrics['MAE'], custom_reg_bas
    'R²': [reg_base_metrics['R²'], reg_improved_metrics['R²'], custom_reg_base_m
})

print("\nСводная таблица моделей регрессии")
print(summary_reg.to_string(index=False))
```

```
Сводная таблица моделей регрессии
            Тип модели      MSE     RMSE      MAE       R²
     Базовая (sklearn) 4.093377 2.023210 1.227129 0.963608
  Улучшенная (sklearn) 8.570538 2.927548 1.872211 0.923804
   Кастомная (базовая) 4.093377 2.023210 1.227129 0.963608
Кастомная (улучшенная) 8.570538 2.927548 1.872211 0.923804
```

Выводы и анализ результатов

In [73]:
```
print("СРАВНЕНИЕ БАЗОВЫХ И КАСТОМНЫХ МОДЕЛЕЙ:")

print("\nКлассификация:")
print("  • Кастомная реализация KNN классификатора показала:")
print(f"    - Accuracy: {custom_class_base_metrics['Accuracy']:.4f} vs {class_ba
print(f"    - F1-Score: {custom_class_base_metrics['F1']:.4f} vs {class_base_met
print(f"    - Recall:   {custom_class_base_metrics['Recall']:.4f} vs {class_base

print("\nРегрессия:")
print("  • Кастомная реализация KNN регрессора показала:")
print(f"    - R²:  {custom_reg_base_metrics['R²']:.4f} vs {reg_base_metrics['R²'
print(f"    - MSE: {custom_reg_base_metrics['MSE']:.4f} vs {reg_base_metrics['MS

print("\nЭФФЕКТИВНОСТЬ ТЕХНИК УЛУЧШЕНИЯ:")

print("\nКлассификация:")
print(f"  • Улучшенная кастомная модель vs базовая кастомная:")
print(f"    - F1-Score улучшился: {custom_improved_metrics['F1']:.4f} vs {custom
print(f"    - Recall улучшился:   {custom_improved_metrics['Recall']:.4f} vs {cu
print(f"  • Улучшенная кастомная vs улучшенная sklearn:")
print(f"    - F1-Score: {custom_improved_metrics['F1']:.4f} vs {class_improved_m
print(f"    - Recall:   {custom_improved_metrics['Recall']:.4f} vs {class_improv

print("\nРегрессия:")
print(f"  • Улучшенная кастомная модель vs базовая кастомная:")
print(f"    - R² улучшился:  {custom_reg_improved_metrics['R²']:.4f} vs {custom_
print(f"    - MSE уменьшился: {custom_reg_improved_metrics['MSE']:.4f} vs {custo
print(f"  • Улучшенная кастомная vs улучшенная sklearn:")
print(f"    - R²:  {custom_reg_improved_metrics['R²']:.4f} vs {reg_improved_metr
print(f"    - MSE: {custom_reg_improved_metrics['MSE']:.4f} vs {reg_improved_met
```

СРАВНЕНИЕ БАЗОВЫХ И КАСТОМНЫХ МОДЕЛЕЙ:

Классификация:
- Кастомная реализация KNN классификатора показала:
  - Accuracy: 0.8724 vs 0.8724 (sklearn)
  - F1-Score: 0.4720 vs 0.4720 (sklearn)
  - Recall: 0.3689 vs 0.3689 (sklearn)

Регрессия:
- Кастомная реализация KNN регрессора показала:
  - R²: 0.9636 vs 0.9636 (sklearn)
  - MSE: 4.0934 vs 4.0934 (sklearn)

ЭФФЕКТИВНОСТЬ ТЕХНИК УЛУЧШЕНИЯ:

Классификация:
- Улучшенная кастомная модель vs базовая кастомная:
  - F1-Score улучшился: 0.5820 vs 0.4720 (+0.1100)
  - Recall улучшился: 0.5087 vs 0.3689 (+0.1399)
- Улучшенная кастомная vs улучшенная sklearn:
  - F1-Score: 0.5820 vs 0.5820
  - Recall: 0.5087 vs 0.5087

Регрессия:
- Улучшенная кастомная модель vs базовая кастомная:
  - R² улучшился: 0.9238 vs 0.9636 (+-0.0398)
  - MSE уменьшился: 8.5705 vs 4.0934 (-4.4772)
- Улучшенная кастомная vs улучшенная sklearn:
  - R²: 0.9238 vs 0.9238
  - MSE: 8.5705 vs 8.5705

Лабораторная работа №2. Проведение исследований с логистической и линейной регрессией

Создание бейзлайна и оценка качества

In [74]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.metrics import (accuracy_score, f1_score, roc_auc_score, confusion_
                             mean_squared_error, mean_absolute_error, r2_score)

from sklearn.metrics import ConfusionMatrixDisplay
```

Классификация

Загрузка датасета

In [75]:
```
df_class = pd.read_csv('datasets/online_shoppers_intention.csv')
```

Размер датасета

In [76]:
```
df_class.shape
```

Out[76]:
```
(12330, 18)
```

Первые 5 строк

In [77]:
```
df_class.head()
```

Out[77]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	Prod
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	

Информация о данных

In [78]:
```
df_class.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

Статистика по числовым признакам

In [79]:
```
df_class.describe()
```

Out[79]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	I
count	12330.000000	12330.000000	12330.000000	12330.000000	
mean	2.315166	80.818611	0.503569	34.472398	
std	3.321784	176.779107	1.270156	140.749294	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	1.000000	7.500000	0.000000	0.000000	
75%	4.000000	93.256250	0.000000	0.000000	
max	27.000000	3398.750000	24.000000	2549.375000	

Определение баланса классов

In [ ]:
```
df_class['Revenue'].value_counts()
```

Out[ ]:
```
Revenue
False    10422
True      1908
Name: count, dtype: int64
```

Копирование датасета для его дальнейшего преобразования

In [81]:
```
df_class_clean = df_class.copy()
```

Кодирование категориальных признаков с помощью `LabelEncoder`

In [82]:
```
categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le
    print(f"Закодирована колонка: {col}")
```

```
Закодирована колонка: Month
Закодирована колонка: VisitorType
```

Выделение признаков и таргета, их разделение на выборки для обучения и тестирования

In [83]:
```
X = df_class_clean.drop('Revenue', axis=1)
y = df_class_clean['Revenue']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"Обучающая выборка: {X_train.shape}")
print(f"Тестовая выборка: {X_test.shape}")
print(f"Распределение классов в train: {np.bincount(y_train)}")
print(f"Распределение классов в test: {np.bincount(y_test)}")
```

```
Обучающая выборка: (8631, 17)
Тестовая выборка: (3699, 17)
Распределение классов в train: [7295 1336]
Распределение классов в test: [3127  572]
```

Масштабирование данных с помощью `StandardScaler`

In [84]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Обучение модели классификации `LogisticRegression`

In [85]:
```
log_reg = LogisticRegression(random_state=42, max_iter=1000)
log_reg.fit(X_train_scaled, y_train)

y_pred = log_reg.predict(X_test_scaled)
y_pred_proba = log_reg.predict_proba(X_test_scaled)[:, 1]
```

Вычисление метрик

In [86]:
```
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)
```

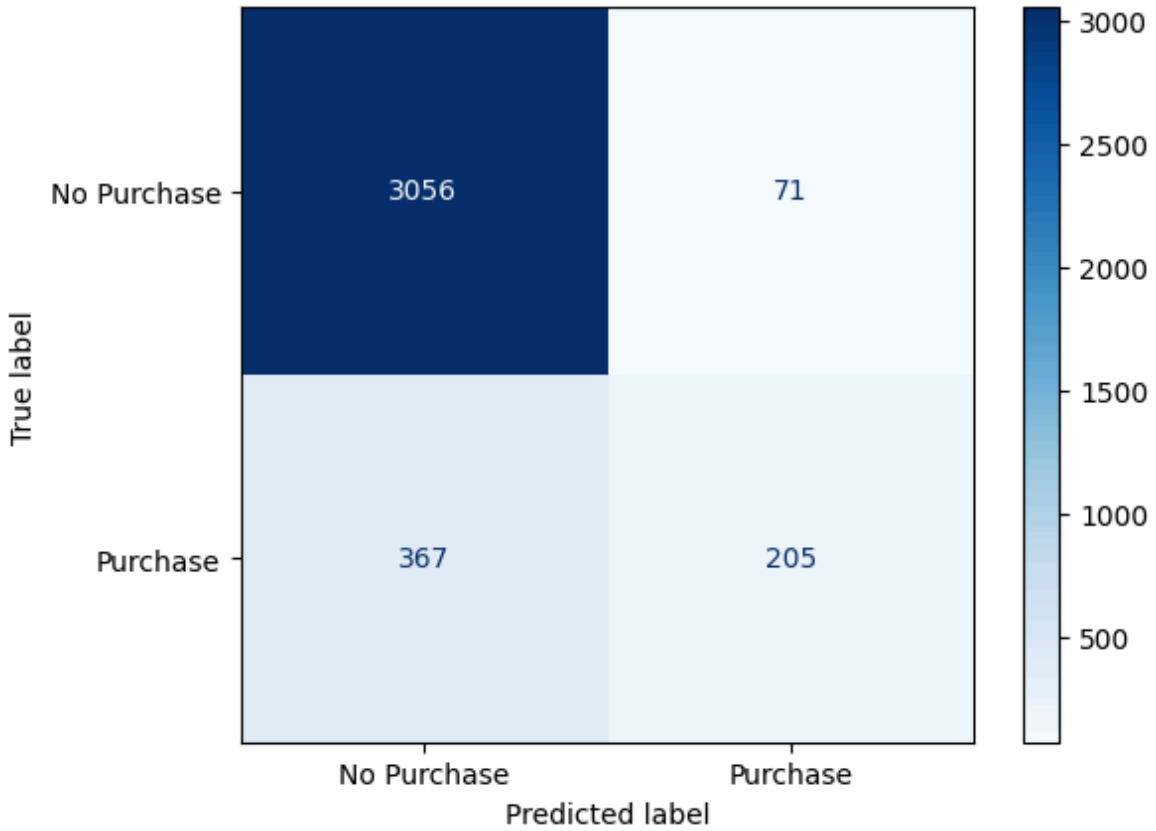
```
print(f"Accuracy: {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")

print("\nМатрица ошибок:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Accuracy: 0.8816
F1-Score: 0.4835
ROC-AUC: 0.8716

Матрица ошибок:
[[3056   71]
 [ 367  205]]
```

Визуализация матрицы ошибок

In [87]:
```
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['No Purchase', 'Purchase'])
disp.plot(cmap='Blues')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



Дополнительная оценка результатов модели

In [88]:
```
TN, FP, FN, TP = cm.ravel()
precision = TP / (TP + FP) if (TP + FP) > 0 else 0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0

print(f"  Precision: {precision:.3f}")
print(f"    - Из {TP+FP} предсказанных покупок, {TP} были верными")
```

```
print(f"  Recall: {recall:.3f}")
print(f"    - Из {TP+FN} реальных покупок, нашли {TP}")
```

```
Precision: 0.743
  - Из 276 предсказанных покупок, 205 были верными
Recall: 0.358
  - Из 572 реальных покупок, нашли 205
```

Регрессия

Загрузка датасета

In [89]:
```
df_reg = pd.read_csv('datasets/parkinsons.csv')
```

Размер датасета

In [90]:
```
df_reg.shape
```

Out[90]:
```
(5875, 22)
```

Первые 5 строк

In [91]:
```
df_reg.head()
```

Out[91]:

	subject#	age	sex	test_time	motor_UPDRS	total_UPDRS	Jitter(%)	Jitter(Abs)	Jit
0	1	72	0	5.6431	28.199	34.398	0.00662	0.000034	
1	1	72	0	12.6660	28.447	34.894	0.00300	0.000017	
2	1	72	0	19.6810	28.695	35.389	0.00481	0.000025	
3	1	72	0	25.6470	28.905	35.810	0.00528	0.000027	
4	1	72	0	33.6420	29.187	36.375	0.00335	0.000020	

5 rows × 22 columns

Информация о данных

In [92]:
```
df_reg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5875 entries, 0 to 5874
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   subject#       5875 non-null   int64  
 1   age            5875 non-null   int64  
 2   sex            5875 non-null   int64  
 3   test_time      5875 non-null   float64
 4   motor_UPDRS    5875 non-null   float64
 5   total_UPDRS    5875 non-null   float64
 6   Jitter(%)      5875 non-null   float64
 7   Jitter(Abs)    5875 non-null   float64
 8   Jitter:RAP     5875 non-null   float64
 9   Jitter:PPQ5    5875 non-null   float64
 10  Jitter:DDP     5875 non-null   float64
 11  Shimmer        5875 non-null   float64
 12  Shimmer(dB)    5875 non-null   float64
 13  Shimmer:APQ3   5875 non-null   float64
 14  Shimmer:APQ5   5875 non-null   float64
 15  Shimmer:APQ11  5875 non-null   float64
 16  Shimmer:DDA    5875 non-null   float64
 17  NHR            5875 non-null   float64
 18  HNR            5875 non-null   float64
 19  RPDE           5875 non-null   float64
 20  DFA            5875 non-null   float64
 21  PPE            5875 non-null   float64
dtypes: float64(19), int64(3)
memory usage: 1009.9 KB
```

Статистика по числовым признакам

In [93]:
```
df_reg.describe()
```

Out[93]:

	subject#	age	sex	test_time	motor_UPDRS	total_UPDRS
count	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000
mean	21.494128	64.804936	0.317787	92.863722	21.296229	29.018942
std	12.372279	8.821524	0.465656	53.445602	8.129282	10.700283
min	1.000000	36.000000	0.000000	-4.262500	5.037700	7.000000
25%	10.000000	58.000000	0.000000	46.847500	15.000000	21.371000
50%	22.000000	65.000000	0.000000	91.523000	20.871000	27.576000
75%	33.000000	72.000000	1.000000	138.445000	27.596500	36.399000
max	42.000000	85.000000	1.000000	215.490000	39.511000	54.992000

8 rows × 22 columns

Копирование датасета для его дальнейшего преобразования. Удаление столбца `subject#`, т.к. не несёт полезной информации

In [94]:
```
df_reg_clean = df_reg.copy()

df_reg_clean = df_reg_clean.drop('subject#', axis=1)
```

Выделение признаков и таргета, их разделение на выборки для обучения и тестирования

In [95]:
```
X = df_reg_clean.drop('total_UPDRS', axis=1)
y = df_reg_clean['total_UPDRS']

print(f"Количество признаков: {X.shape[1]}")
print(f"Диапазон целевой переменной: [{y.min():.2f}, {y.max():.2f}]")
print(f"Среднее значение целевой переменной: {y.mean():.2f}")
print(f"Стандартное отклонение целевой переменной: {y.std():.2f}")

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(f"\nРазмеры выборок:")
print(f"  Обучающая выборка: {X_train_reg.shape}")
print(f"  Тестовая выборка: {X_test_reg.shape}")
```

```
Количество признаков: 20
Диапазон целевой переменной: [7.00, 54.99]
Среднее значение целевой переменной: 29.02
Стандартное отклонение целевой переменной: 10.70

Размеры выборок:
  Обучающая выборка: (4112, 20)
  Тестовая выборка: (1763, 20)
```

Масштабирование данных с помощью `StandardScaler`

In [96]:
```
scaler_reg = StandardScaler()
X_train_reg_scaled = scaler_reg.fit_transform(X_train_reg)
X_test_reg_scaled = scaler_reg.transform(X_test_reg)
```

Обучение модели регрессии `LinearRegression`

In [97]:
```
lin_reg = LinearRegression()
lin_reg.fit(X_train_reg_scaled, y_train_reg)

y_pred_reg = lin_reg.predict(X_test_reg_scaled)
```

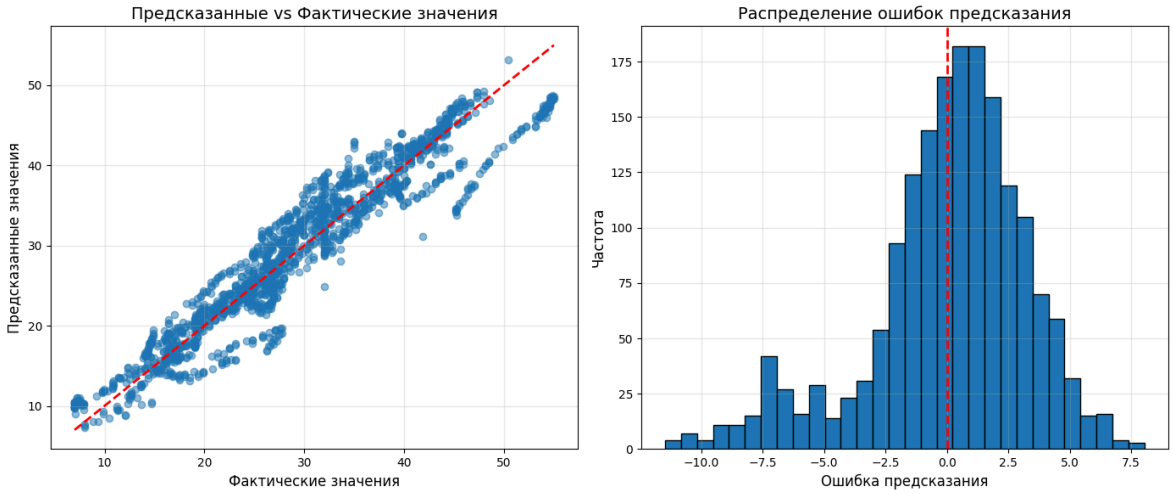
Вычисление метрик

In [98]:
```
mse = mean_squared_error(y_test_reg, y_pred_reg)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)

print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"R²: {r2:.4f}")
```

```
MSE: 10.4965
RMSE: 3.2398
MAE: 2.4321
R²: 0.9067
```

Визуализация предсказаний

In [99]:
```
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

axes[0].scatter(y_test_reg, y_pred_reg, alpha=0.5)
axes[0].plot([y_test_reg.min(), y_test_reg.max()],
             [y_test_reg.min(), y_test_reg.max()],
             'r--', lw=2)
axes[0].set_xlabel('Фактические значения', fontsize=12)
axes[0].set_ylabel('Предсказанные значения', fontsize=12)
axes[0].set_title('Предсказанные vs Фактические значения', fontsize=14)
axes[0].grid(True, alpha=0.3)

errors = y_pred_reg - y_test_reg
axes[1].hist(errors, bins=30, edgecolor='black')
axes[1].axvline(x=0, color='r', linestyle='--', linewidth=2)
axes[1].set_xlabel('Ошибка предсказания', fontsize=12)
axes[1].set_ylabel('Частота', fontsize=12)
axes[1].set_title('Распределение ошибок предсказания', fontsize=14)
axes[1].grid(True, alpha=0.3)

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



Результаты базовых моделей

In [100…
```
print("\nКлассификация")
print("Метрики:")
print(f"- Accuracy: {accuracy:.4f}")
print(f"- F1-Score: {f1:.4f}")
print(f"- ROC-AUC:  {roc_auc:.4f}")

print("\nРегрессия")
print("Метрики:")
print(f"- MSE:  {mse:.4f}")
print(f"- RMSE: {rmse:.4f}")
print(f"- MAE:  {mae:.4f}")
print(f"- R²:   {r2:.4f}")
```

```
Классификация
Метрики:
- Accuracy: 0.8816
- F1-Score: 0.4835
- ROC-AUC:  0.8716

Регрессия
Метрики:
- MSE:  10.4965
- RMSE: 3.2398
- MAE:  2.4321
- R²:   0.9067
```

Улучшение бейзлайна

In [101…
```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import PolynomialFeatures, RobustScaler
from sklearn.linear_model import Ridge
from sklearn.metrics import precision_score, recall_score, roc_curve
from sklearn.feature_selection import SelectKBest, f_classif

from typing import Union, Any

import warnings
```

Классификация

Сохранение метрик базовой модели

In [102…
```python
class_base_metrics = {
    'Accuracy': accuracy,
    'F1': f1,
    'ROC-AUC': roc_auc,
    'Precision': precision,
    'Recall': recall
}
```

Функция сравнения метрик новой модели с базовой

In [103…
```python
def print_comparison_class(metrics_old, metrics_new):
    comparison_data = []
    for metric in ['Accuracy', 'F1', 'ROC-AUC', 'Precision', 'Recall']:
        base_val = metrics_old[metric]
        new_val = metrics_new[metric]
        diff = new_val - base_val
        change = "улучшение" if diff > 0 else "ухудшение"

        comparison_data.append({
            'Метрика': metric,
            'Базовая модель': f"{base_val:.4f}",
            'Новая модель': f"{new_val:.4f}",
            'Разница': f"{diff:+.4f}",
            'Изменение': change
        })
```

```
    df_comparison = pd.DataFrame(comparison_data)
    print(df_comparison.to_string(index=False))
```

Повторное копирование, разделение и масштабирование данных

In [104…
```
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le

X = df_class_clean.drop('Revenue', axis=1)
y = df_class_clean['Revenue']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Гипотеза 1: Балансировка классов

In [105…
```
log_reg_balanced = LogisticRegression(random_state=42, max_iter=1000, class_weig
log_reg_balanced.fit(X_train_scaled, y_train)
y_pred_bal = log_reg_balanced.predict(X_test_scaled)
y_proba_bal = log_reg_balanced.predict_proba(X_test_scaled)[:, 1]

metrics_bal = {
    'Accuracy': accuracy_score(y_test, y_pred_bal),
    'F1': f1_score(y_test, y_pred_bal),
    'ROC-AUC': roc_auc_score(y_test, y_proba_bal),
    'Precision': precision_score(y_test, y_pred_bal),
    'Recall': recall_score(y_test, y_pred_bal)
}

print("Балансировка классов")
print_comparison_class(class_base_metrics, metrics_bal)
```

```
Балансировка классов
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8816       0.8629 -0.0187 ухудшение
       F1         0.4835       0.6202 +0.1367 улучшение
  ROC-AUC         0.8716       0.8789 +0.0073 улучшение
Precision         0.7428       0.5426 -0.2002 ухудшение
   Recall         0.3584       0.7238 +0.3654 улучшение
```

Гипотеза 2: Отбор признаков с балансировкой

In [106…
```
selector = SelectKBest(f_classif, k=10)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)

log_reg_selected = LogisticRegression(random_state=42, max_iter=1000, class_weig
```

```
log_reg_selected.fit(X_train_selected, y_train)
y_pred_sel = log_reg_selected.predict(X_test_selected)
y_proba_sel = log_reg_selected.predict_proba(X_test_selected)[:, 1]

metrics_sel = {
    'Accuracy': accuracy_score(y_test, y_pred_sel),
    'F1': f1_score(y_test, y_pred_sel),
    'ROC-AUC': roc_auc_score(y_test, y_proba_sel),
    'Precision': precision_score(y_test, y_pred_sel),
    'Recall': recall_score(y_test, y_pred_sel)
}

print("Отбор 10 лучших признаков с балансировкой")
print_comparison_class(class_base_metrics, metrics_sel)
```

```
Отбор 10 лучших признаков с балансировкой
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8816       0.8624 -0.0192 ухудшение
       F1         0.4835       0.6141 +0.1306 улучшение
  ROC-AUC         0.8716       0.8747 +0.0031 улучшение
Precision         0.7428       0.5422 -0.2006 ухудшение
   Recall         0.3584       0.7080 +0.3497 улучшение
```

Гипотеза 3: Добавление полиномиальных признаков

In [107…

```
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

print(f"Исходное количество признаков: {X_train_scaled.shape[1]}")
print(f"Количество признаков с полиномальными: {X_train_poly.shape[1]}")

log_reg_poly = LogisticRegression(random_state=42, max_iter=1000)
log_reg_poly.fit(X_train_poly, y_train)
y_pred_poly = log_reg_poly.predict(X_test_poly)
y_proba_poly = log_reg_poly.predict_proba(X_test_poly)[:, 1]

metrics_poly = {
    'Accuracy': accuracy_score(y_test, y_pred_poly),
    'F1': f1_score(y_test, y_pred_poly),
    'ROC-AUC': roc_auc_score(y_test, y_proba_poly),
    'Precision': precision_score(y_test, y_pred_poly),
    'Recall': recall_score(y_test, y_pred_poly)
}

print("Полиномиальные признаки")
print_comparison_class(class_base_metrics, metrics_poly)
```

```
Исходное количество признаков: 17
Количество признаков с полиномальными: 153
Полиномиальные признаки
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8816       0.8851 +0.0035 улучшение
       F1         0.4835       0.5251 +0.0416 улучшение
  ROC-AUC         0.8716       0.8770 +0.0054 улучшение
Precision         0.7428       0.7276 -0.0152 ухудшение
   Recall         0.3584       0.4108 +0.0524 улучшение
```

Гипотеза 4: Настройка гиперпараметров

In [108…

```
warnings.filterwarnings('ignore')

param_grid = [
    {
        'C': [0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 50, 100, 500, 1000],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear', 'saga'],
        'class_weight': [None, 'balanced'],
        'max_iter': [100, 500, 1000, 2000, 5000, 10000],
        'random_state': [42]
    },

    {
        'C': [0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 50, 100, 500, 1000],
        'penalty': ['elasticnet'],
        'solver': ['saga'],
        'class_weight': [None, 'balanced'],
        'l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9],
        'max_iter': [100, 500, 1000, 2000, 5000, 10000],
        'random_state': [42]
    }
]

log_reg_gs = LogisticRegression(random_state=42)
random_search = RandomizedSearchCV(
    estimator=log_reg_gs, param_distributions=param_grid, n_iter=100, cv=5,
    scoring='roc_auc', random_state=42, n_jobs=-1
)
random_search.fit(X_train_scaled, y_train)

print("Лучшие параметры:")
for param, value in random_search.best_params_.items():
    print(f"  {param}: {value}")

best_log_reg: Union[LogisticRegression, Any] = random_search.best_estimator_
y_pred_gs = best_log_reg.predict(X_test_scaled)
y_proba_gs = best_log_reg.predict_proba(X_test_scaled)[:, 1]

metrics_gs = {
    'Accuracy': accuracy_score(y_test, y_pred_gs),
    'F1': f1_score(y_test, y_pred_gs),
    'ROC-AUC': roc_auc_score(y_test, y_proba_gs),
    'Precision': precision_score(y_test, y_pred_gs),
    'Recall': recall_score(y_test, y_pred_gs)
}

print("Подбор гиперпараметров")
print_comparison_class(class_base_metrics, metrics_gs)
```

```
Лучшие параметры:
  solver: liblinear
  random_state: 42
  penalty: l1
  max_iter: 1000
  class_weight: balanced
  C: 0.01
Подбор гиперпараметров
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8816       0.8656 -0.0160 ухудшение
       F1         0.4835       0.6215 +0.1380 улучшение
  ROC-AUC         0.8716       0.8810 +0.0094 улучшение
Precision         0.7428       0.5506 -0.1921 ухудшение
   Recall         0.3584       0.7133 +0.3549 улучшение
```

Формирование улучешнной модели и её обучение

In [109…
```
best_params = random_search.best_params_.copy()

improved_log_reg = LogisticRegression(**best_params)
improved_log_reg.fit(X_train_scaled, y_train)

y_pred_improved = improved_log_reg.predict(X_test_scaled)
y_pred_proba_improved = improved_log_reg.predict_proba(X_test_scaled)[:, 1]
```

Метрики улучшенной модели

In [110…
```
class_improved_metrics = {
    'Accuracy': accuracy_score(y_test, y_pred_improved),
    'F1': f1_score(y_test, y_pred_improved),
    'ROC-AUC': roc_auc_score(y_test, y_pred_proba_improved),
    'Precision': precision_score(y_test, y_pred_improved),
    'Recall': recall_score(y_test, y_pred_improved)
}

for metric, value in class_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8656
F1: 0.6215
ROC-AUC: 0.8810
Precision: 0.5506
Recall: 0.7133
```

Сравнение улучшенной модели с базовой

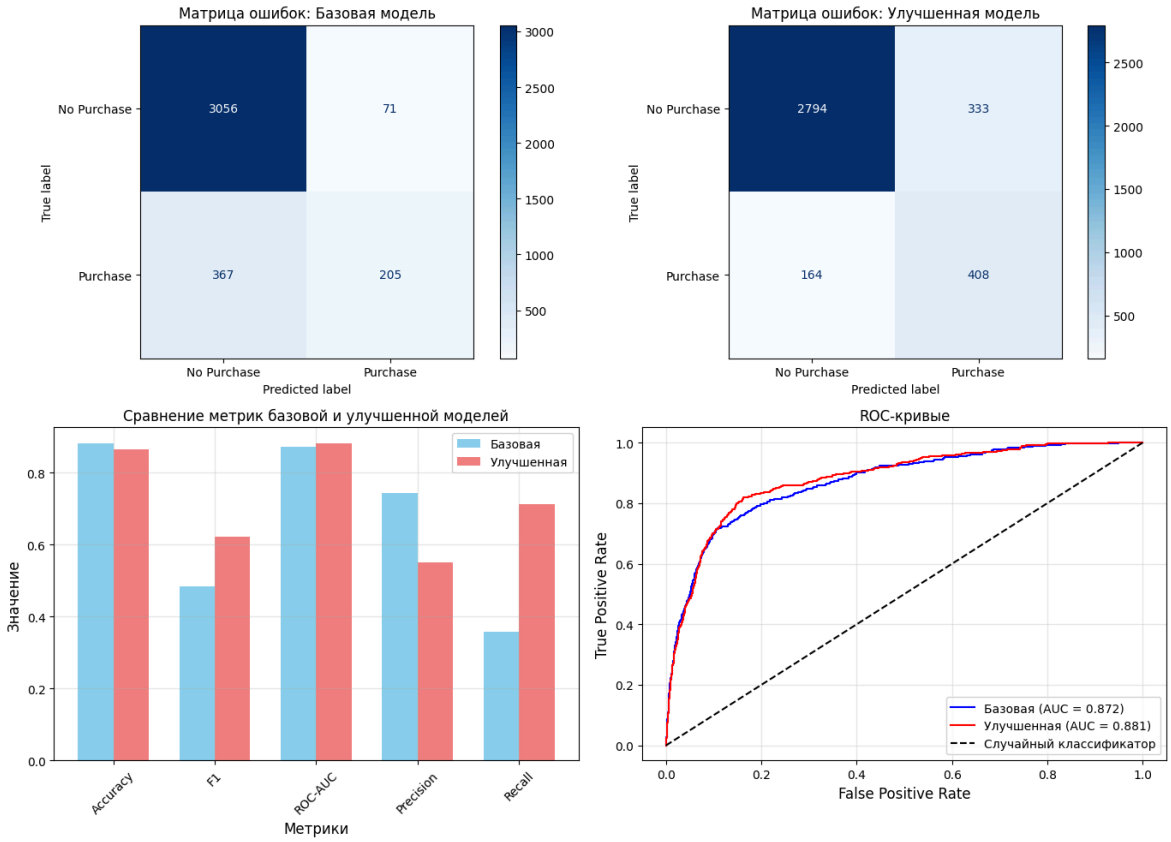
In [111…
```
print_comparison_class(class_base_metrics, class_improved_metrics)
```

```
 Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8816       0.8656 -0.0160 ухудшение
       F1         0.4835       0.6215 +0.1380 улучшение
  ROC-AUC         0.8716       0.8810 +0.0094 улучшение
Precision         0.7428       0.5506 -0.1921 ухудшение
   Recall         0.3584       0.7133 +0.3549 улучшение
```

Визуальное сравнение базовой и улучшенной модели

In [112…
```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
```

```
axes[0, 0].set_title('Матрица ошибок: Базовая модель', fontsize=12)
cm_base = confusion_matrix(y_test, y_pred)
disp_base = ConfusionMatrixDisplay(confusion_matrix=cm_base,
                                   display_labels=['No Purchase', 'Purchase'])
disp_base.plot(ax=axes[0, 0], cmap='Blues')

axes[0, 1].set_title('Матрица ошибок: Улучшенная модель', fontsize=12)
cm_improved = confusion_matrix(y_test, y_pred_improved)
disp_improved = ConfusionMatrixDisplay(confusion_matrix=cm_improved,
                                       display_labels=['No Purchase', 'Purchase'
disp_improved.plot(ax=axes[0, 1], cmap='Blues')

metrics_names = ['Accuracy', 'F1', 'ROC-AUC', 'Precision', 'Recall']
base_values = [class_base_metrics[m] for m in metrics_names]
improved_values = [class_improved_metrics[m] for m in metrics_names]

x = np.arange(len(metrics_names))
width = 0.35

axes[1, 0].bar(x - width/2, base_values, width, label='Базовая', color='skyblue'
axes[1, 0].bar(x + width/2, improved_values, width, label='Улучшенная', color='l
axes[1, 0].set_xlabel('Метрики', fontsize=12)
axes[1, 0].set_ylabel('Значение', fontsize=12)
axes[1, 0].set_title('Сравнение метрик базовой и улучшенной моделей', fontsize=1
axes[1, 0].set_xticks(x)
axes[1, 0].set_xticklabels(metrics_names, rotation=45)
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

fpr_base, tpr_base, _ = roc_curve(y_test, y_pred_proba)
fpr_improved, tpr_improved, _ = roc_curve(y_test, y_pred_proba_improved)

axes[1, 1].plot(fpr_base, tpr_base, label=f'Базовая (AUC = {roc_auc:.3f})', colo
axes[1, 1].plot(fpr_improved, tpr_improved, label=f'Улучшенная (AUC = {class_imp
axes[1, 1].plot([0, 1], [0, 1], 'k--', label='Случайный классификатор')
axes[1, 1].set_xlabel('False Positive Rate', fontsize=12)
axes[1, 1].set_ylabel('True Positive Rate', fontsize=12)
axes[1, 1].set_title('ROC-кривые', fontsize=12)
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

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



Анализ результатов классификации

In [113…
```
TN_base, FP_base, FN_base, TP_base = cm_base.ravel()
TN_imp, FP_imp, FN_imp, TP_imp = cm_improved.ravel()

print("\n1. Анализ обнаружения покупок:")
print(f"   Базовая модель нашла {TP_base} из {TP_base + FN_base} реальных покупо
print(f"   Улучшенная модель нашла {TP_imp} из {TP_imp + FN_imp} реальных покупо
print(f"   Улучшение в обнаружении покупок: {TP_imp - TP_base} реальных покупок"
print(f"   Процентное улучшение Recall: {((TP_imp / (TP_imp + FN_imp)) - (TP_bas

print("\n2. Анализ ложных срабатываний:")
print(f"   Базовая модель: {FP_base} ложных предсказаний покупки")
print(f"   Улучшенная модель: {FP_imp} ложных предсказаний покупки")
print(f"   Изменение: {FP_imp - FP_base} дополнительных ложных срабатываний")
```

1. Анализ обнаружения покупок:
   Базовая модель нашла 205 из 572 реальных покупок
   Улучшенная модель нашла 408 из 572 реальных покупок
   Улучшение в обнаружении покупок: 203 реальных покупок
   Процентное улучшение Recall: +35.5%

2. Анализ ложных срабатываний:
   Базовая модель: 71 ложных предсказаний покупки
   Улучшенная модель: 333 ложных предсказаний покупки
   Изменение: 262 дополнительных ложных срабатываний

Регрессия

Сохранение метрик базовой модели

In [114…
```
reg_base_metrics = {
    'MSE': mse,
```

```
    'RMSE': rmse,
    'MAE': mae,
    'R²': r2
}
```

Функция сравнения метрик новой модели с базовой

In [115…

```
def print_comparison_reg(metrics_old, metrics_new):
    comparison_data = []
    for metric in ['MSE', 'RMSE', 'MAE', 'R²']:
        base_val = metrics_old[metric]
        new_val = metrics_new[metric]
        diff = new_val - base_val
        if metric == 'R²':
            change = "улучшение" if diff > 0 else "ухудшение"
        else:
            change = "улучшение" if diff < 0 else "ухудшение"

        comparison_data.append({
            'Метрика': metric,
            'Базовая модель': f"{base_val:.4f}",
            'Новая модель': f"{new_val:.4f}",
            'Разница': f"{diff:+.4f}",
            'Изменение': change
        })

    df_comparison = pd.DataFrame(comparison_data)
    print(df_comparison.to_string(index=False))
```

Повторное копирование и подготовка данных

In [116…

```
df_reg_clean = df_reg.copy()
df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X = df_reg_clean.drop('total_UPDRS', axis=1)
y = df_reg_clean['total_UPDRS']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Гипотеза 1: Обработка выбросов с помощью RobustScaler

In [117…

```
robust_scaler = RobustScaler()
X_train_robust = robust_scaler.fit_transform(X_train)
X_test_robust = robust_scaler.transform(X_test)

lin_reg_robust = LinearRegression()
lin_reg_robust.fit(X_train_robust, y_train)
y_pred_robust = lin_reg_robust.predict(X_test_robust)

metrics_robust = {
    'MSE': mean_squared_error(y_test, y_pred_robust),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_robust)),
```

```
    'MAE': mean_absolute_error(y_test, y_pred_robust),
    'R²': r2_score(y_test, y_pred_robust)
}

print("RobustScaler (устойчивое масштабирование)")
print_comparison_reg(reg_base_metrics, metrics_robust)
```

```
RobustScaler (устойчивое масштабирование)
Метрика Базовая модель Новая модель Разница Изменение
    MSE        10.4965      10.4965 -0.0000 улучшение
   RMSE         3.2398       3.2398 -0.0000 улучшение
    MAE         2.4321       2.4321 -0.0000 улучшение
     R²         0.9067       0.9067 +0.0000 улучшение
```

Гипотеза 2: Логарифмическое преобразование целевой переменной

In [118…

```
y_train_log = np.log1p(y_train)
y_test_log = np.log1p(y_test)

lin_reg_log = LinearRegression()
lin_reg_log.fit(X_train_scaled, y_train_log)
y_pred_log = lin_reg_log.predict(X_test_scaled)

y_pred_exp = np.expm1(y_pred_log)

metrics_log = {
    'MSE': mean_squared_error(y_test, y_pred_exp),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_exp)),
    'MAE': mean_absolute_error(y_test, y_pred_exp),
    'R²': r2_score(y_test, y_pred_exp)
}

print("Логарифмирование целевой переменной")
print_comparison_reg(reg_base_metrics, metrics_log)
```

```
Логарифмирование целевой переменной
Метрика Базовая модель Новая модель Разница Изменение
    MSE        10.4965      12.6509 +2.1544 ухудшение
   RMSE         3.2398       3.5568 +0.3170 ухудшение
    MAE         2.4321       2.7039 +0.2718 ухудшение
     R²         0.9067       0.8875 -0.0192 ухудшение
```

Гипотеза 3: Добавление полиномиальных признаков

In [119…

```
poly = PolynomialFeatures(degree=2, include_bias=False, interaction_only=True)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

print(f"Исходное количество признаков: {X_train_scaled.shape[1]}")
print(f"Количество признаков с полиномиальными: {X_train_poly.shape[1]}")

lin_reg_poly = LinearRegression()
lin_reg_poly.fit(X_train_poly, y_train)
y_pred_poly = lin_reg_poly.predict(X_test_poly)

metrics_poly = {
    'MSE': mean_squared_error(y_test, y_pred_poly),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_poly)),
    'MAE': mean_absolute_error(y_test, y_pred_poly),
    'R²': r2_score(y_test, y_pred_poly)
```

```
}

print("Полиномиальные признаки (степень 2)")
print_comparison_reg(reg_base_metrics, metrics_poly)
```

```
Исходное количество признаков: 20
Количество признаков с полиномиальными: 210
Полиномиальные признаки (степень 2)
Метрика Базовая модель Новая модель Разница Изменение
    MSE        10.4965       7.0020 -3.4946 улучшение
   RMSE         3.2398       2.6461 -0.5937 улучшение
    MAE         2.4321       2.0288 -0.4033 улучшение
     R²         0.9067       0.9377 +0.0311 улучшение
```

Гипотеза 4: Использование регуляризации (Ridge регрессия)

In [120…

```
warnings.filterwarnings('ignore')

poly = PolynomialFeatures(degree=2, include_bias=False, interaction_only=True)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

param_grid = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'fit_intercept': [True, False],
    'solver': ['auto', 'svd', 'lsqr', 'saga'],
    'max_iter': [100, 500, 1000, 2000],
    'random_state': [42]
}

ridge = Ridge(random_state=42)
random_search_reg = RandomizedSearchCV(
    estimator=ridge, param_distributions=param_grid, n_iter=20, cv=5,
    scoring='r2', random_state=42, n_jobs=-1
)
random_search_reg.fit(X_train_poly, y_train)

print("Лучшие параметры Ridge:")
for param, value in random_search_reg.best_params_.items():
    print(f"  {param}: {value}")

best_ridge: Union[Ridge, Any] = random_search_reg.best_estimator_
y_pred_ridge = best_ridge.predict(X_test_poly)

metrics_ridge = {
    'MSE': mean_squared_error(y_test, y_pred_ridge),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_ridge)),
    'MAE': mean_absolute_error(y_test, y_pred_ridge),
    'R²': r2_score(y_test, y_pred_ridge),
}
print("Ridge регрессия с подбором гиперпараметров")
print_comparison_reg(reg_base_metrics, metrics_ridge)
```

```
Лучшие параметры Ridge:
  solver: svd
  random_state: 42
  max_iter: 100
  fit_intercept: True
  alpha: 1
Ridge регрессия с подбором гиперпараметров
Метрика Базовая модель Новая модель Разница Изменение
    MSE        10.4965       7.1598 -3.3367 улучшение
   RMSE         3.2398       2.6758 -0.5640 улучшение
    MAE         2.4321       2.0082 -0.4239 улучшение
     R²         0.9067       0.9363 +0.0297 улучшение
```

Формирование улучшенной модели и её обучение

In [121…
```
poly = PolynomialFeatures(degree=2, include_bias=False, interaction_only=True)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

best_params = random_search_reg.best_params_.copy()

improved_model = Ridge(**best_params)
improved_model.fit(X_train_poly, y_train)
y_pred_improved = improved_model.predict(X_test_poly)
```

Метрики улучшенной модели

In [122…
```
reg_improved_metrics = {
    'MSE': mean_squared_error(y_test, y_pred_improved),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_improved)),
    'MAE': mean_absolute_error(y_test, y_pred_improved),
    'R²': r2_score(y_test, y_pred_improved)
}

for metric, value in reg_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 7.1598
RMSE: 2.6758
MAE: 2.0082
R²: 0.9363
```

Сравнение улучшенной модели с базовой

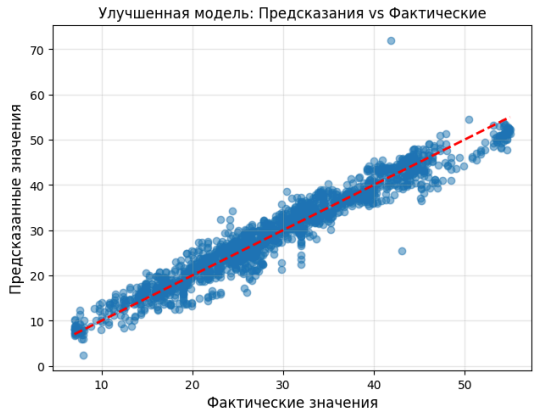
In [123…
```
print_comparison_reg(reg_base_metrics, reg_improved_metrics)
```

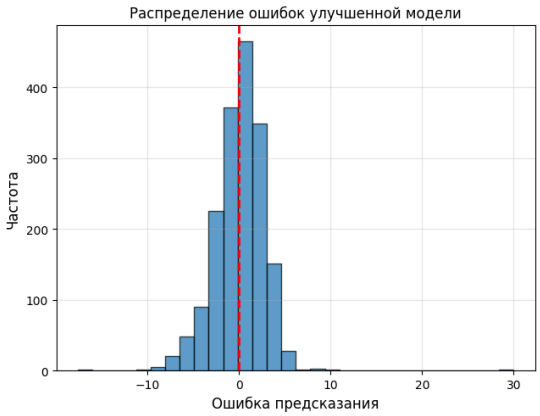
```
Метрика Базовая модель Новая модель Разница Изменение
    MSE        10.4965       7.1598 -3.3367 улучшение
   RMSE         3.2398       2.6758 -0.5640 улучшение
    MAE         2.4321       2.0082 -0.4239 улучшение
     R²         0.9067       0.9363 +0.0297 улучшение
```

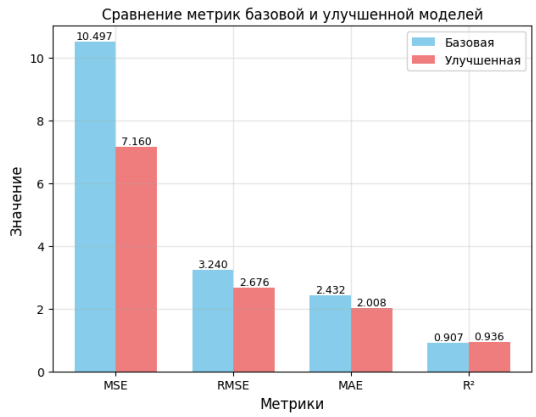
Визуальное сравнение базовой и улучшенной модели

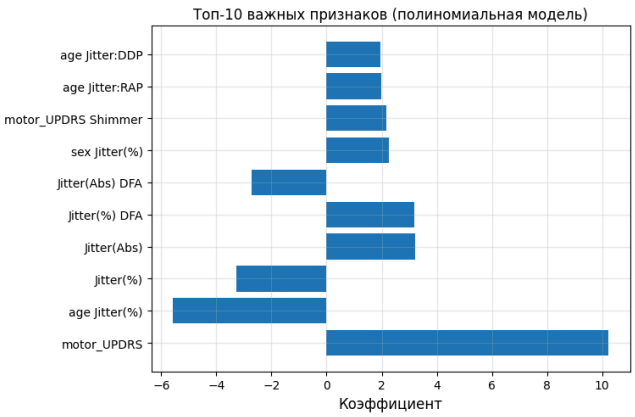
In [124…
```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes[0, 0].scatter(y_test, y_pred_improved, alpha=0.5)
axes[0, 0].plot([y_test.min(), y_test.max()],
                [y_test.min(), y_test.max()],
```

```
                'r--', lw=2)
axes[0, 0].set_xlabel('Фактические значения', fontsize=12)
axes[0, 0].set_ylabel('Предсказанные значения', fontsize=12)
axes[0, 0].set_title('Улучшенная модель: Предсказания vs Фактические', fontsize=
axes[0, 0].grid(True, alpha=0.3)

errors_improved = y_pred_improved - y_test
axes[0, 1].hist(errors_improved, bins=30, edgecolor='black', alpha=0.7)
axes[0, 1].axvline(x=0, color='r', linestyle='--', linewidth=2)
axes[0, 1].set_xlabel('Ошибка предсказания', fontsize=12)
axes[0, 1].set_ylabel('Частота', fontsize=12)
axes[0, 1].set_title('Распределение ошибок улучшенной модели', fontsize=12)
axes[0, 1].grid(True, alpha=0.3)

metrics_names = ['MSE', 'RMSE', 'MAE', 'R²']
base_values = [reg_base_metrics[m] for m in metrics_names]
improved_values = [reg_improved_metrics[m] for m in metrics_names]

x = np.arange(len(metrics_names))
width = 0.35

bars1 = axes[1, 0].bar(x - width/2, base_values, width, label='Базовая', color='
bars2 = axes[1, 0].bar(x + width/2, improved_values, width, label='Улучшенная',
axes[1, 0].set_xlabel('Метрики', fontsize=12)
axes[1, 0].set_ylabel('Значение', fontsize=12)
axes[1, 0].set_title('Сравнение метрик базовой и улучшенной моделей', fontsize=1
axes[1, 0].set_xticks(x)
axes[1, 0].set_xticklabels(metrics_names)
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

for bar in bars1 + bars2:
    height = bar.get_height()
    axes[1, 0].text(bar.get_x() + bar.get_width()/2., height,
                    f'{height:.3f}', ha='center', va='bottom', fontsize=9)

poly_feature_names = poly.get_feature_names_out(X.columns)

feature_importance = pd.DataFrame({
    'Признак': poly_feature_names,
    'Коэффициент': improved_model.coef_
}).sort_values('Коэффициент', key=abs, ascending=False).head(10)

axes[1, 1].barh(feature_importance['Признак'], feature_importance['Коэффициент']
axes[1, 1].set_xlabel('Коэффициент', fontsize=12)
axes[1, 1].set_title('Топ-10 важных признаков (полиномиальная модель)', fontsize
axes[1, 1].grid(True, alpha=0.3)

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









Анализ результатов регрессии

In [125…

```
print(f"\n1. Анализ улучшения модели:")
print(f"   R² увеличился с {reg_base_metrics['R²']:.4f} до {reg_improved_metrics
print(f"   Улучшение R²: {reg_improved_metrics['R²'] - reg_base_metrics['R²']:+.
print(f"   MSE уменьшился с {reg_base_metrics['MSE']:.4f} до {reg_improved_metri
print(f"   Улучшение MSE: {reg_base_metrics['MSE'] - reg_improved_metrics['MSE']

print(f"\n2. Статистика ошибок улучшенной модели:")
print(f"   Средняя абсолютная ошибка: {reg_improved_metrics['MAE']:.2f}")
print(f"   Средняя ошибка в процентах от среднего target: {reg_improved_metrics[
print(f"   Стандартное отклонение ошибок: {np.std(errors_improved):.2f}")

print(f"\n3. Интерпретируемость модели:")
print(f"   Наиболее важные признаки (по абсолютному значению коэффициентов):")
for idx, row in feature_importance.head(5).iterrows():
    print(f"     {row['Признак']}: {row['Коэффициент']:.4f}")
```

1. Анализ улучшения модели:
   R^2 увеличился с 0.9067 до 0.9363
   Улучшение R^2: +0.0297
   MSE уменьшился с 10.4965 до 7.1598
   Улучшение MSE: 3.3367 (31.79%)

2. Статистика ошибок улучшенной модели:
   Средняя абсолютная ошибка: 2.01
   Средняя ошибка в процентах от среднего target: 6.91%
   Стандартное отклонение ошибок: 2.68

3. Интерпретируемость модели:
   Наиболее важные признаки (по абсолютному значению коэффициентов):
     motor_UPDRS: 10.2367
     age Jitter(%): -5.5758
     Jitter(%): -3.2804
     Jitter(Abs): 3.2121
     Jitter(%) DFA: 3.1779

Имплементация алгоритма машинного обучения

Классификация

Кастомная модель логистической регрессии

In [126…

```python
class CustomLogisticRegression:

    def __init__(self, learning_rate=0.01, n_iterations=1000, regularization=Non
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.regularization = regularization
        self.lambda_reg = lambda_reg
        self.weights: np.ndarray = np.array([])
        self.bias = None
        self.loss_history = []

    def _sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def _compute_loss(self, y_true, y_pred):
        epsilon = 1e-15
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
        loss = -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pr

        if self.regularization == 'l1':
            loss += self.lambda_reg * np.sum(np.abs(self.weights))
        elif self.regularization == 'l2':
            loss += self.lambda_reg * np.sum(self.weights**2)

        return loss

    def fit(self, X, y, class_weight=None):
        n_samples, n_features = X.shape

        self.weights = np.zeros(n_features)
        self.bias = 0
```

```
        if class_weight is not None:
            if class_weight == 'balanced':
                class_counts = np.bincount(y)
                n_classes = len(class_counts)
                weight_per_class = {}

                for i in range(n_classes):
                    weight_per_class[i] = n_samples / (n_classes * class_counts[

                sample_weights = np.array([weight_per_class[label] for label in
            else:
                sample_weights = np.array([class_weight[label] for label in y])
        else:
            sample_weights = np.ones(n_samples)

        for iteration in range(self.n_iterations):
            linear_model = np.dot(X, self.weights) + self.bias
            y_pred = self._sigmoid(linear_model)

            dw = (1 / n_samples) * np.dot(X.T, (y_pred - y) * sample_weights)
            db = (1 / n_samples) * np.sum((y_pred - y) * sample_weights)

            if self.regularization == 'l1':
                dw += self.lambda_reg * np.sign(self.weights)
            elif self.regularization == 'l2':
                dw += 2 * self.lambda_reg * self.weights

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

            loss = self._compute_loss(y, y_pred)
            self.loss_history.append(loss)

            if iteration % 100 == 0:
                print(f"Iteration {iteration}, Loss: {loss:.4f}")

    def predict_proba(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        return self._sigmoid(linear_model)

    def predict(self, X, threshold=0.5):
        probabilities = self.predict_proba(X)
        return (probabilities >= threshold).astype(int)

    def get_params(self):
        return {
            'weights': self.weights,
            'bias': self.bias,
            'loss_history': self.loss_history
        }
```

Повторное копирование и разбиение данных

In [127…

```
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli
label_encoders = {}
for col in categorical_cols:
```

```
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le

X_class = df_class_clean.drop('Revenue', axis=1)
y_class = df_class_clean['Revenue']

X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class, test_size=0.3, random_state=42, stratify=y_class
)

scaler_class = StandardScaler()
X_train_class_scaled = scaler_class.fit_transform(X_train_class)
X_test_class_scaled = scaler_class.transform(X_test_class)
```

Обучение кастомной логистической регрессии

In [128…

```
custom_log_reg = CustomLogisticRegression(
    learning_rate=0.1,
    n_iterations=2000,
    regularization='l2',
    lambda_reg=0.01
)

custom_log_reg.fit(X_train_class_scaled, y_train_class)

y_pred_custom_log = custom_log_reg.predict(X_test_class_scaled)
y_pred_proba_custom_log = custom_log_reg.predict_proba(X_test_class_scaled)
```

```
Iteration 0, Loss: 0.6932
Iteration 100, Loss: 0.3365
Iteration 200, Loss: 0.3182
Iteration 300, Loss: 0.3151
Iteration 400, Loss: 0.3143
Iteration 500, Loss: 0.3140
Iteration 600, Loss: 0.3139
Iteration 700, Loss: 0.3139
Iteration 800, Loss: 0.3139
Iteration 900, Loss: 0.3139
Iteration 1000, Loss: 0.3139
Iteration 1100, Loss: 0.3139
Iteration 1200, Loss: 0.3139
Iteration 1300, Loss: 0.3139
Iteration 1400, Loss: 0.3139
Iteration 1500, Loss: 0.3139
Iteration 1600, Loss: 0.3139
Iteration 1700, Loss: 0.3139
Iteration 1800, Loss: 0.3139
Iteration 1900, Loss: 0.3139
```

Метрики кастомной логистической регрессии

In [129…

```
custom_class_base_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_custom_log),
    'F1': f1_score(y_test_class, y_pred_custom_log),
    'ROC-AUC': roc_auc_score(y_test_class, y_pred_proba_custom_log),
    'Precision': precision_score(y_test_class, y_pred_custom_log),
    'Recall': recall_score(y_test_class, y_pred_custom_log)
}
```

```
for metric, value in custom_class_base_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8735
F1: 0.4046
ROC-AUC: 0.8746
Precision: 0.7430
Recall: 0.2780
```

Сравнение кастомной модели с базовой из sklearn

In [130…

```
print_comparison_class(class_base_metrics, custom_class_base_metrics)
```

```
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8816       0.8735 -0.0081 ухудшение
       F1         0.4835       0.4046 -0.0789 ухудшение
  ROC-AUC         0.8716       0.8746 +0.0031 улучшение
Precision         0.7428       0.7430 +0.0002 улучшение
   Recall         0.3584       0.2780 -0.0804 ухудшение
```

Обучение улучшенной кастомной логистическая регрессии

In [131…

```
improved_custom_log_reg = CustomLogisticRegression(
    learning_rate=0.05,
    n_iterations=3000,
    regularization='l1',
    lambda_reg=0.01
)

improved_custom_log_reg.fit(X_train_class_scaled, y_train_class, class_weight='b

y_pred_imp_custom = improved_custom_log_reg.predict(X_test_class_scaled)
y_pred_proba_imp_custom = improved_custom_log_reg.predict_proba(X_test_class_sca
```

```
Iteration 0, Loss: 0.6938
Iteration 100, Loss: 0.5601
Iteration 200, Loss: 0.5126
Iteration 300, Loss: 0.4916
Iteration 400, Loss: 0.4818
Iteration 500, Loss: 0.4768
Iteration 600, Loss: 0.4741
Iteration 700, Loss: 0.4728
Iteration 800, Loss: 0.4721
Iteration 900, Loss: 0.4717
Iteration 1000, Loss: 0.4715
Iteration 1100, Loss: 0.4715
Iteration 1200, Loss: 0.4715
Iteration 1300, Loss: 0.4716
Iteration 1400, Loss: 0.4716
Iteration 1500, Loss: 0.4718
Iteration 1600, Loss: 0.4721
Iteration 1700, Loss: 0.4722
Iteration 1800, Loss: 0.4723
Iteration 1900, Loss: 0.4723
Iteration 2000, Loss: 0.4723
Iteration 2100, Loss: 0.4723
Iteration 2200, Loss: 0.4723
Iteration 2300, Loss: 0.4724
Iteration 2400, Loss: 0.4724
Iteration 2500, Loss: 0.4724
Iteration 2600, Loss: 0.4724
Iteration 2700, Loss: 0.4724
Iteration 2800, Loss: 0.4724
Iteration 2900, Loss: 0.4724
```

Метрики улучшенной кастомной логистической регрессии

In [132…

```python
custom_improved_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_imp_custom),
    'F1': f1_score(y_test_class, y_pred_imp_custom),
    'ROC-AUC': roc_auc_score(y_test_class, y_pred_proba_imp_custom),
    'Precision': precision_score(y_test_class, y_pred_imp_custom),
    'Recall': recall_score(y_test_class, y_pred_imp_custom)
}

for metric, value in custom_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8681
F1: 0.6223
ROC-AUC: 0.8798
Precision: 0.5583
Recall: 0.7028
```

Сравнение улучшенной кастомной модели с улучшенной из sklearn

In [133…

```python
print_comparison_class(class_improved_metrics, custom_improved_metrics)
```

```
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8656       0.8681 +0.0024 улучшение
       F1         0.6215       0.6223 +0.0008 улучшение
  ROC-AUC         0.8810       0.8798 -0.0011 ухудшение
Precision         0.5506       0.5583 +0.0077 улучшение
   Recall         0.7133       0.7028 -0.0105 ухудшение
```

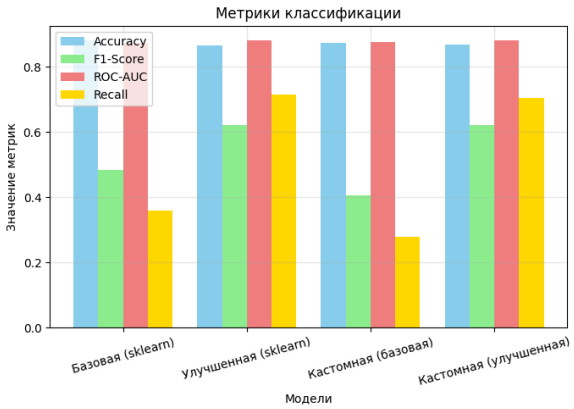
Итоговое сравнение всех моделей классификации

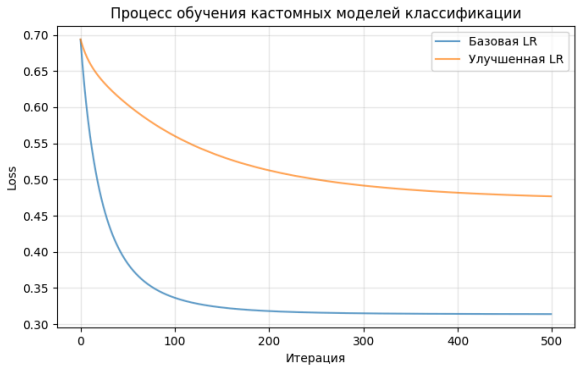
In [134…
```
summary_class = pd.DataFrame({
    'Тип модели': ['Базовая (sklearn)', 'Улучшенная (sklearn)', 'Кастомная (базо
    'Accuracy': [class_base_metrics['Accuracy'], class_improved_metrics['Accurac
    'F1-Score': [class_base_metrics['F1'], class_improved_metrics['F1'], custom_
    'ROC-AUC': [class_base_metrics['ROC-AUC'], class_improved_metrics['ROC-AUC']
    'Recall': [class_base_metrics['Recall'], class_improved_metrics['Recall'], c
})

print("Сводная таблица моделей классификации")
print(summary_class.to_string(index=False))
```

```
Сводная таблица моделей классификации
            Тип модели  Accuracy  F1-Score  ROC-AUC   Recall
     Базовая (sklearn)  0.881590  0.483491 0.871561 0.358392
  Улучшенная (sklearn)  0.865639  0.621478 0.880971 0.713287
   Кастомная (базовая)  0.873479  0.404580 0.874637 0.277972
Кастомная (улучшенная)  0.868072  0.622291 0.879840 0.702797
```

Визуализация сравнения всех моделей классификации

In [135…
```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

x = np.arange(len(summary_class))
width = 0.2

axes[0].bar(x - width*1.5, summary_class['Accuracy'], width, label='Accuracy', c
axes[0].bar(x - width/2, summary_class['F1-Score'], width, label='F1-Score', col
axes[0].bar(x + width/2, summary_class['ROC-AUC'], width, label='ROC-AUC', color
axes[0].bar(x + width*1.5, summary_class['Recall'], width, label='Recall', color
axes[0].set_xlabel('Модели')
axes[0].set_ylabel('Значение метрик')
axes[0].set_title('Метрики классификации')
axes[0].set_xticks(x)
axes[0].set_xticklabels(summary_class['Тип модели'], rotation=15)
axes[0].legend()
axes[0].grid(True, alpha=0.3)

axes[1].plot(custom_log_reg.loss_history[:500], label='Базовая LR', alpha=0.7)
axes[1].plot(improved_custom_log_reg.loss_history[:500], label='Улучшенная LR',
axes[1].set_xlabel('Итерация')
axes[1].set_ylabel('Loss')
axes[1].set_title('Процесс обучения кастомных моделей классификации')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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





Регрессия

Кастомная модель линейной регрессии

In [136…

```python
class CustomLinearRegression:

    def __init__(self, learning_rate=0.01, n_iterations=1000, regularization=Non
                 lambda_reg=0.1, adaptive_lr=True, early_stopping=True):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.regularization = regularization
        self.lambda_reg = lambda_reg
        self.adaptive_lr = adaptive_lr
        self.early_stopping = early_stopping
        self.weights: np.ndarray = np.array([])
        self.bias = None
        self.loss_history = []

    def _initialize_weights(self, n_features):
        limit = np.sqrt(2.0 / n_features) if n_features > 1 else 0.01
        self.weights = np.random.normal(0, limit, n_features)
        self.bias = 0.0

    def _compute_loss(self, y_true, y_pred):
        epsilon = 1e-8
        mse = np.mean((y_true - y_pred) ** 2) + epsilon

        if self.weights is not None:
            if self.regularization == 'l1':
                mse += self.lambda_reg * np.sum(np.abs(self.weights))
            elif self.regularization == 'l2':
                mse += self.lambda_reg * np.sum(self.weights ** 2)

        return mse

    def _compute_gradients(self, X, y_true, y_pred):
        n_samples = X.shape[0]
        error = y_pred - y_true

        dw = (1 / n_samples) * np.dot(X.T, error)
        db = (1 / n_samples) * np.sum(error)

        if self.regularization == 'l1':
            dw += self.lambda_reg * np.sign(self.weights)
        elif self.regularization == 'l2':
```

```
            dw += 2 * self.lambda_reg * self.weights

        return dw, db

    def fit(self, X, y, verbose=False):
        n_samples, n_features = X.shape

        self._initialize_weights(n_features)

        current_lr = self.learning_rate
        if n_features > 100:
            current_lr = min(self.learning_rate, 0.001)

        best_loss = float('inf')
        patience_counter = 0
        patience_limit = 50 if n_features > 100 else 30

        for iteration in range(self.n_iterations):
            y_pred = np.dot(X, self.weights) + self.bias

            loss = self._compute_loss(y, y_pred)
            self.loss_history.append(loss)

            dw, db = self._compute_gradients(X, y, y_pred)

            self.weights -= current_lr * dw
            self.bias -= current_lr * db

            if self.adaptive_lr and iteration > 100 and iteration % 100 == 0:
                if len(self.loss_history) > 100:
                    recent_improvement = self.loss_history[-100] - self.loss_his
                    if recent_improvement < 1e-4:
                        current_lr *= 0.9
                        if verbose:
                            print(f"  Уменьшение LR до: {current_lr:.6f}")

            if self.early_stopping:
                if loss < best_loss - 1e-6:
                    best_loss = loss
                    patience_counter = 0
                else:
                    patience_counter += 1

                if patience_counter >= patience_limit:
                    if verbose:
                        print(f"  Ранняя остановка на итерации {iteration}")
                    break

            if verbose and iteration % 500 == 0:
                print(f"Iteration {iteration}, Loss: {loss:.4f}, LR: {current_lr

    def predict(self, X):
        return np.dot(X, self.weights) + self.bias

    def get_params(self):
        return {
            'weights': self.weights,
            'bias': self.bias,
            'loss_history': self.loss_history
        }
```

Повторное копирование и разбиение данных

In [137…
```
df_reg_clean = df_reg.copy()

df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X_reg = df_reg_clean.drop('total_UPDRS', axis=1)
y_reg = df_reg_clean['total_UPDRS']

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.3, random_state=42
)

scaler_reg = StandardScaler()
X_train_reg_scaled = scaler_reg.fit_transform(X_train_reg)
X_test_reg_scaled = scaler_reg.transform(X_test_reg)
```

Обучение кастомной линейной регрессии

In [138…
```
custom_lin_reg = CustomLinearRegression(
    learning_rate=0.01,
    n_iterations=2000,
    regularization='l2',
    lambda_reg=0.01
)

custom_lin_reg.fit(X_train_reg_scaled, y_train_reg)

y_pred_custom_reg = custom_lin_reg.predict(X_test_reg_scaled)
```

Метрики кастомной линейной регрессии

In [139…
```
custom_reg_base_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_custom_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test_reg, y_pred_custom_reg)),
    'MAE': mean_absolute_error(y_test_reg, y_pred_custom_reg),
    'R²': r2_score(y_test_reg, y_pred_custom_reg)
}

for metric, value in custom_reg_base_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 10.5557
RMSE: 3.2490
MAE: 2.4144
R²: 0.9062
```

Сравнение кастомной модели с базовой из sklearn

In [140…
```
print_comparison_reg(reg_base_metrics, custom_reg_base_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE        10.4965      10.5557 +0.0592 ухудшение
   RMSE         3.2398       3.2490 +0.0091 ухудшение
    MAE         2.4321       2.4144 -0.0177 улучшение
     R²         0.9067       0.9062 -0.0005 ухудшение
```

Обучение улучшенной кастомной линейной регрессии

In [141…
```
poly = PolynomialFeatures(degree=2, include_bias=False, interaction_only=True)
X_train_poly = poly.fit_transform(X_train_reg_scaled)
X_test_poly = poly.transform(X_test_reg_scaled)

scaler_poly = StandardScaler()
X_train_poly_scaled = scaler_poly.fit_transform(X_train_poly)
X_test_poly_scaled = scaler_poly.transform(X_test_poly)

improved_custom_lin_reg = CustomLinearRegression(
    learning_rate=0.001,
    n_iterations=5000,
    regularization='l2',
    lambda_reg=0.1,
    adaptive_lr=True,
    early_stopping=True
)

improved_custom_lin_reg.fit(X_train_poly_scaled, y_train_reg, verbose=True)

y_pred_imp_custom_reg = improved_custom_lin_reg.predict(X_test_poly_scaled)
```

```
Iteration 0, Loss: 957.4734, LR: 0.001000
Iteration 500, Loss: 354.6059, LR: 0.001000
Iteration 1000, Loss: 142.5198, LR: 0.001000
Iteration 1500, Loss: 64.8646, LR: 0.001000
Iteration 2000, Loss: 36.0020, LR: 0.001000
Iteration 2500, Loss: 25.1069, LR: 0.001000
Iteration 3000, Loss: 20.9037, LR: 0.001000
Iteration 3500, Loss: 19.2258, LR: 0.001000
Iteration 4000, Loss: 18.5191, LR: 0.001000
Iteration 4500, Loss: 18.1973, LR: 0.001000
```

Метрики улучшенной кастомной линейной регрессии

In [142…
```
custom_reg_improved_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_imp_custom_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test_reg, y_pred_imp_custom_reg)),
    'MAE': mean_absolute_error(y_test_reg, y_pred_imp_custom_reg),
    'R²': r2_score(y_test_reg, y_pred_imp_custom_reg)
}

for metric, value in custom_reg_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 10.6476
RMSE: 3.2631
MAE: 2.4921
R²: 0.9053
```

Сравнение улучшенной модели с улучшенной из sklearn

In [143…
```
print_comparison_reg(reg_improved_metrics, custom_reg_improved_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE         7.1598      10.6476 +3.4878 ухудшение
   RMSE         2.6758       3.2631 +0.5873 ухудшение
    MAE         2.0082       2.4921 +0.4838 ухудшение
     R²         0.9363       0.9053 -0.0310 ухудшение
```

Итоговое сравнение всех моделей регрессии

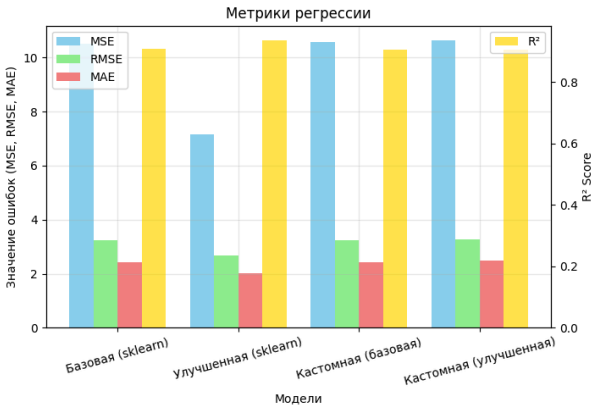
In [144…

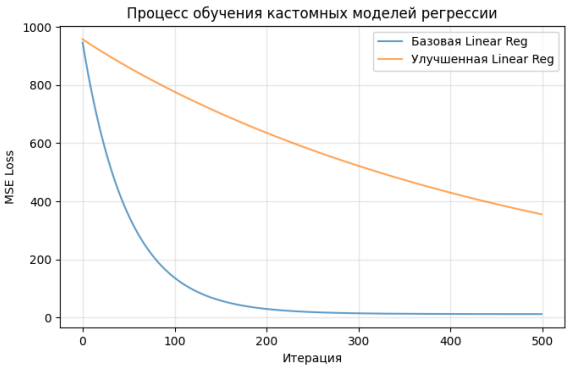
```
summary_reg = pd.DataFrame({
    'Тип модели': ['Базовая (sklearn)', 'Улучшенная (sklearn)', 'Кастомная (базо
    'MSE': [reg_base_metrics['MSE'], reg_improved_metrics['MSE'], custom_reg_bas
    'RMSE': [reg_base_metrics['RMSE'], reg_improved_metrics['RMSE'], custom_reg_
    'MAE': [reg_base_metrics['MAE'], reg_improved_metrics['MAE'], custom_reg_bas
    'R²': [reg_base_metrics['R²'], reg_improved_metrics['R²'], custom_reg_base_m
})

print("\nСводная таблица моделей регрессии")
print(summary_reg.to_string(index=False))
```

```
Сводная таблица моделей регрессии
            Тип модели       MSE     RMSE      MAE       R²
     Базовая (sklearn) 10.496511 3.239832 2.432137 0.906681
  Улучшенная (sklearn)  7.159833 2.675786 2.008223 0.936346
   Кастомная (базовая) 10.555726 3.248958 2.414401 0.906155
Кастомная (улучшенная) 10.647605 3.263067 2.492055 0.905338
```

In [145…

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

x = np.arange(len(summary_reg))
width = 0.2

axes[0].bar(x - width*1.5, summary_reg['MSE'], width, label='MSE', color='skyblu
axes[0].bar(x - width/2, summary_reg['RMSE'], width, label='RMSE', color='lightg
axes[0].bar(x + width/2, summary_reg['MAE'], width, label='MAE', color='lightcor
axes[0].set_xlabel('Модели')
axes[0].set_ylabel('Значение ошибок (MSE, RMSE, MAE)')
axes[0].set_title('Метрики регрессии')
axes[0].set_xticks(x)
axes[0].set_xticklabels(summary_reg['Тип модели'], rotation=15)
axes[0].legend(loc='upper left')
axes[0].grid(True, alpha=0.3)

ax2 = axes[0].twinx()
ax2.bar(x + width*1.5, summary_reg['R²'], width, label='R²', color='gold', alpha
ax2.set_ylabel('R² Score')
ax2.legend(loc='upper right')

axes[1].plot(custom_lin_reg.loss_history[:500], label='Базовая Linear Reg', alph
axes[1].plot(improved_custom_lin_reg.loss_history[:500], label='Улучшенная Linea
axes[1].set_xlabel('Итерация')
axes[1].set_ylabel('MSE Loss')
axes[1].set_title('Процесс обучения кастомных моделей регрессии')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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





Выводы и анализ результатов

In [ ]:
```
print("СРАВНЕНИЕ БАЗОВЫХ И КАСТОМНЫХ МОДЕЛЕЙ:")

print("\nКлассификация:")
print("  • Кастомная реализация логистической регрессии показала:")
print(f"    - Accuracy: {custom_class_base_metrics['Accuracy']:.4f} vs {class_ba
print(f"    - F1-Score: {custom_class_base_metrics['F1']:.4f} vs {class_base_met
print(f"    - Recall:   {custom_class_base_metrics['Recall']:.4f} vs {class_base

print("\nРегрессия:")
print("  • Кастомная реализация линейной регрессии показала:")
print(f"    - R²:  {custom_reg_base_metrics['R²']:.4f} vs {reg_base_metrics['R²'
print(f"    - MSE: {custom_reg_base_metrics['MSE']:.4f} vs {reg_base_metrics['MS

print("ЭФФЕКТИВНОСТЬ ТЕХНИК УЛУЧШЕНИЯ:")

print("\nКлассификация:")
print(f"  • Улучшенная кастомная модель vs базовая кастомная:")
print(f"    - F1-Score улучшился: {custom_improved_metrics['F1']:.4f} vs {custom
print(f"    - Recall улучшился:   {custom_improved_metrics['Recall']:.4f} vs {cu
print(f"  • Улучшенная кастомная vs улучшенная sklearn:")
print(f"    - F1-Score: {custom_improved_metrics['F1']:.4f} vs {class_improved_m
print(f"    - Recall:   {custom_improved_metrics['Recall']:.4f} vs {class_improv

print("\nРегрессия:")
print(f"  • Улучшенная кастомная модель vs базовая кастомная:")
print(f"    - R² улучшился:  {custom_reg_improved_metrics['R²']:.4f} vs {custom_
print(f"    - MSE уменьшился: {custom_reg_improved_metrics['MSE']:.4f} vs {custo
print(f"  • Улучшенная кастомная vs улучшенная sklearn:")
print(f"    - R²:  {custom_reg_improved_metrics['R²']:.4f} vs {reg_improved_metr
print(f"    - MSE: {custom_reg_improved_metrics['MSE']:.4f} vs {reg_improved_met
```

СРАВНЕНИЕ БАЗОВЫХ И КАСТОМНЫХ МОДЕЛЕЙ:

Классификация:
- Кастомная реализация логистической регрессии показала:
  - Accuracy: 0.8735 vs 0.8816 (sklearn)
  - F1-Score: 0.4046 vs 0.4835 (sklearn)
  - Recall: 0.2780 vs 0.3584 (sklearn)

Регрессия:
- Кастомная реализация линейной регрессии показала:
  - R²: 0.9062 vs 0.9067 (sklearn)
  - MSE: 10.5557 vs 10.4965 (sklearn)

ЭФФЕКТИВНОСТЬ ТЕХНИК УЛУЧШЕНИЯ:

Классификация (балансировка классов + регуляризация):
- Улучшенная кастомная модель vs базовая кастомная:
  - F1-Score улучшился: 0.6223 vs 0.4046 (+0.2177)
  - Recall улучшился: 0.7028 vs 0.2780 (+0.4248)
- Улучшенная кастомная vs улучшенная sklearn:
  - F1-Score: 0.6223 vs 0.6215
  - Recall: 0.7028 vs 0.7133

Регрессия (полиномиальные признаки + регуляризация):
- Улучшенная кастомная модель vs базовая кастомнаясustom_reg_improved_metrics['MSE']:
  - R² улучшился: 0.9053 vs 0.9062 (+-0.0008)
  - MSE уменьшился: 10.6476 vs 10.5557 (-0.0919)
- Улучшенная кастомная vs улучшенная sklearn:
  - R²: 0.9053 vs 0.9363
  - MSE: 10.6476 vs 7.1598

Лабораторная работа №3. Проведение исследований с решающим деревом

Создание бейзлайна и оценка качества

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler, PolynomialFeatur
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.metrics import (accuracy_score, f1_score, roc_auc_score, confusion_
                             mean_squared_error, mean_absolute_error, r2_score,
                             precision_score, recall_score, roc_curve)
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.feature_selection import SelectKBest, f_classif

from typing import Union, Any
import warnings
```

Классификация

Загрузка датасета

In [2]:
```
df_class = pd.read_csv('datasets/online_shoppers_intention.csv')
```

Размер датасета

In [3]:
```
df_class.shape
```

Out[3]:
```
(12330, 18)
```

Первые 5 строк

In [4]:
```
df_class.head()
```

Out[4]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	Prod
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	

Информация о данных

In [5]:
```
df_class.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           12330 non-null  int64  
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64  
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64  
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object 
 11  OperatingSystems         12330 non-null  int64  
 12  Browser                  12330 non-null  int64  
 13  Region                   12330 non-null  int64  
 14  TrafficType              12330 non-null  int64  
 15  VisitorType              12330 non-null  object 
 16  Weekend                  12330 non-null  bool   
 17  Revenue                  12330 non-null  bool   
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

Статистика по числовым признакам

In [6]:
```
df_class.describe()
```

Out[6]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	I
count	12330.000000	12330.000000	12330.000000	12330.000000	
mean	2.315166	80.818611	0.503569	34.472398	
std	3.321784	176.779107	1.270156	140.749294	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	1.000000	7.500000	0.000000	0.000000	
75%	4.000000	93.256250	0.000000	0.000000	
max	27.000000	3398.750000	24.000000	2549.375000	

Определение баланса классов

In [7]:
```
df_class['Revenue'].value_counts()
```

Out[7]:
```
Revenue
False    10422
True      1908
Name: count, dtype: int64
```

Копирование датасета для его дальнейшего преобразования

In [8]:
```
df_class_clean = df_class.copy()
```

Кодирование категориальных признаков с помощью `LabelEncoder`

In [10]:
```
categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le
    print(f"  Закодирована колонка: {col}")
```

```
Закодирована колонка: Month
Закодирована колонка: VisitorType
```

Выделение признаков и таргета, их разделение на выборки для обучения и тестирования

In [11]:
```
X = df_class_clean.drop('Revenue', axis=1)
y = df_class_clean['Revenue']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"\nРазмеры выборок:")
print(f"  Обучающая выборка: {X_train.shape}")
print(f"  Тестовая выборка: {X_test.shape}")
print(f"  Распределение классов в train: {np.bincount(y_train)}")
print(f"  Распределение классов в test: {np.bincount(y_test)}")
```

```
Размеры выборок:
  Обучающая выборка: (8631, 17)
  Тестовая выборка: (3699, 17)
  Распределение классов в train: [7295 1336]
  Распределение классов в test: [3127  572]
```

Масштабирование данных с помощью `StandardScaler`

In [12]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

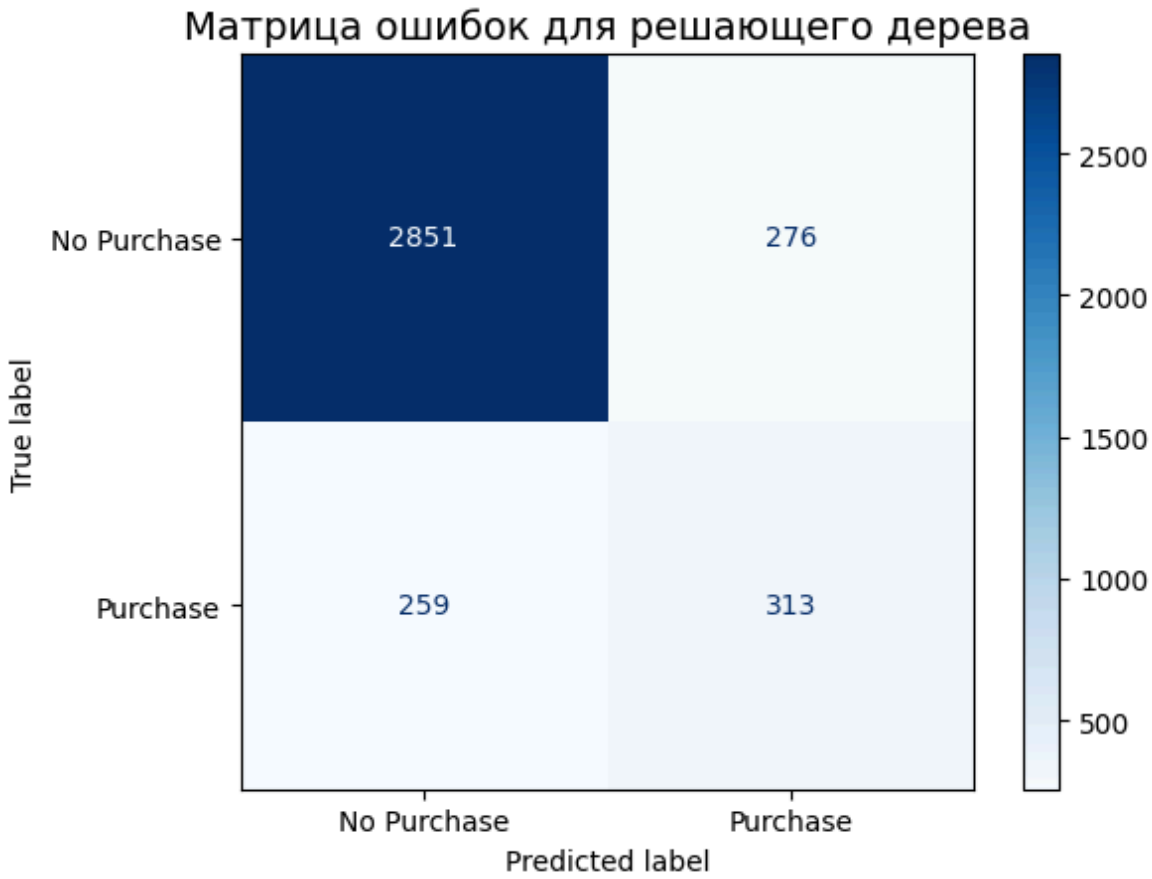
Обучение модели классификации `DecisionTreeClassifier`

In [13]:
```
tree_classifier = DecisionTreeClassifier(random_state=42)
tree_classifier.fit(X_train_scaled, y_train)

y_pred = tree_classifier.predict(X_test_scaled)
y_pred_proba = tree_classifier.predict_proba(X_test_scaled)[:, 1]
```

Вычисление метрик

In [14]:
```
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)

print(f"Accuracy (точность): {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")

print("\nМатрица ошибок:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Accuracy (точность): 0.8554
F1-Score: 0.5392
ROC-AUC: 0.7295

Матрица ошибок:
[[2851  276]
 [ 259  313]]
<Figure size 800x600 with 0 Axes>
```



Визуализация матрицы ошибок

In [ ]:
```
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['No Purchase', 'Purchase'])
disp.plot(cmap='Blues')
plt.title('Матрица ошибок для решающего дерева', fontsize=14)
plt.show()
```

Дополнительная оценка результатов модели

In [15]:
```
TN, FP, FN, TP = cm.ravel()
precision = TP / (TP + FP) if (TP + FP) > 0 else 0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0

print(f"  Precision: {precision:.3f}")
print(f"    - Из {TP+FP} предсказанных покупок, {TP} были верными")
print(f"  Recall: {recall:.3f}")
print(f"    - Из {TP+FN} реальных покупок, нашли {TP}")
```

1. Модель правильно предсказывает 85.5% всех сессий
2. F1-Score = 0.539 (баланс между точностью и полнотой)
3. ROC-AUC = 0.729 (чем ближе к 1, тем лучше модель различает классы)

```
Дополнительные метрики из матрицы ошибок:
  Precision (точность): 0.531
    - Из 589 предсказанных покупок, 313 были верными
  Recall (полнота): 0.547
    - Из 572 реальных покупок, нашли 313
  False Positive Rate: 0.088
  False Negative Rate: 0.453
```

Регрессия

Загрузка датасета

In [16]:
```
df_reg = pd.read_csv('datasets/parkinsons.csv')
```

Размер датасета

In [17]:
```
df_reg.shape
```

Out[17]:
```
(5875, 22)
```

Первые 5 строк

In [18]:
```
df_reg.head()
```

Out[18]:

	subject#	age	sex	test_time	motor_UPDRS	total_UPDRS	Jitter(%)	Jitter(Abs)	Jit
0	1	72	0	5.6431	28.199	34.398	0.00662	0.000034	
1	1	72	0	12.6660	28.447	34.894	0.00300	0.000017	
2	1	72	0	19.6810	28.695	35.389	0.00481	0.000025	
3	1	72	0	25.6470	28.905	35.810	0.00528	0.000027	
4	1	72	0	33.6420	29.187	36.375	0.00335	0.000020	

5 rows × 22 columns

Информация о данных

In [19]:
```
df_reg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5875 entries, 0 to 5874
Data columns (total 22 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   subject#        5875 non-null   int64  
 1   age             5875 non-null   int64  
 2   sex             5875 non-null   int64  
 3   test_time       5875 non-null   float64
 4   motor_UPDRS     5875 non-null   float64
 5   total_UPDRS     5875 non-null   float64
 6   Jitter(%)       5875 non-null   float64
 7   Jitter(Abs)     5875 non-null   float64
 8   Jitter:RAP      5875 non-null   float64
 9   Jitter:PPQ5     5875 non-null   float64
 10  Jitter:DDP      5875 non-null   float64
 11  Shimmer         5875 non-null   float64
 12  Shimmer(dB)     5875 non-null   float64
 13  Shimmer:APQ3    5875 non-null   float64
 14  Shimmer:APQ5    5875 non-null   float64
 15  Shimmer:APQ11   5875 non-null   float64
 16  Shimmer:DDA     5875 non-null   float64
 17  NHR             5875 non-null   float64
 18  HNR             5875 non-null   float64
 19  RPDE            5875 non-null   float64
 20  DFA             5875 non-null   float64
 21  PPE             5875 non-null   float64
dtypes: float64(19), int64(3)
memory usage: 1009.9 KB
```

Статистика по числовым признакам

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

Out[20]:

	subject#	age	sex	test_time	motor_UPDRS	total_UPDRS
count	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000
mean	21.494128	64.804936	0.317787	92.863722	21.296229	29.018942
std	12.372279	8.821524	0.465656	53.445602	8.129282	10.700283
min	1.000000	36.000000	0.000000	-4.262500	5.037700	7.000000
25%	10.000000	58.000000	0.000000	46.847500	15.000000	21.371000
50%	22.000000	65.000000	0.000000	91.523000	20.871000	27.576000
75%	33.000000	72.000000	1.000000	138.445000	27.596500	36.399000
max	42.000000	85.000000	1.000000	215.490000	39.511000	54.992000

8 rows × 22 columns

Копирование датасета для его дальнейшего преобразования. Удаление столбца `subject#`, т.к. не несёт полезной информации

In [21]:
```
df_reg_clean = df_reg.copy()

df_reg_clean = df_reg_clean.drop('subject#', axis=1)
```

Выделение признаков и таргета, их разделение на выборки для обучения и тестирования

In [22]:
```
X = df_reg_clean.drop('total_UPDRS', axis=1)
y = df_reg_clean['total_UPDRS']

print(f"Количество признаков: {X.shape[1]}")
print(f"Диапазон целевой переменной: [{y.min():.2f}, {y.max():.2f}]")
print(f"Среднее значение целевой переменной: {y.mean():.2f}")
print(f"Стандартное отклонение целевой переменной: {y.std():.2f}")

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(f"\nРазмеры выборок:")
print(f"  Обучающая выборка: {X_train_reg.shape}")
print(f"  Тестовая выборка: {X_test_reg.shape}")
```

```
Количество признаков: 20
Диапазон целевой переменной: [7.00, 54.99]
Среднее значение целевой переменной: 29.02
Стандартное отклонение целевой переменной: 10.70
```

Масштабирование данных с помощью `StandardScaler`

In [24]:
```
scaler_reg = StandardScaler()
X_train_reg_scaled = scaler_reg.fit_transform(X_train_reg)
X_test_reg_scaled = scaler_reg.transform(X_test_reg)
```

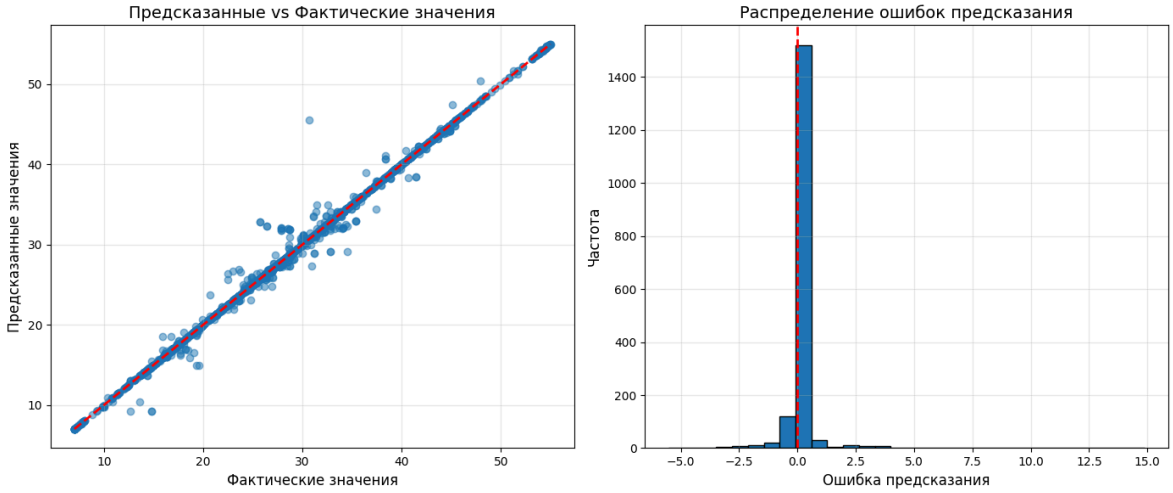
Обучение модели регрессии `DecisionTreeRegressor`

In [25]:
```
tree_regressor = DecisionTreeRegressor(random_state=42)
tree_regressor.fit(X_train_reg_scaled, y_train_reg)

y_pred_reg = tree_regressor.predict(X_test_reg_scaled)
```

Вычисление метрик

In [26]:
```
mse = mean_squared_error(y_test_reg, y_pred_reg)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)

print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"R²: {r2:.4f}")
```

```
MSE (среднеквадратичная ошибка): 0.6149
RMSE (корень из MSE): 0.7842
MAE (средняя абсолютная ошибка): 0.2016
R² (коэффициент детерминации): 0.9945
```



Визуализация предсказаний

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

axes[0].scatter(y_test_reg, y_pred_reg, alpha=0.5)
axes[0].plot([y_test_reg.min(), y_test_reg.max()],
             [y_test_reg.min(), y_test_reg.max()],
             'r--', lw=2)
axes[0].set_xlabel('Фактические значения', fontsize=12)
axes[0].set_ylabel('Предсказанные значения', fontsize=12)
axes[0].set_title('Предсказанные vs Фактические значения', fontsize=14)
axes[0].grid(True, alpha=0.3)

errors = y_pred_reg - y_test_reg
axes[1].hist(errors, bins=30, edgecolor='black')
axes[1].axvline(x=0, color='r', linestyle='--', linewidth=2)
axes[1].set_xlabel('Ошибка предсказания', fontsize=12)
axes[1].set_ylabel('Частота', fontsize=12)
axes[1].set_title('Распределение ошибок предсказания', fontsize=14)
```

```
axes[1].grid(True, alpha=0.3)

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

Результаты базовых моделей

In [28]:
```
print("\nКлассификация")
print("Метрики:")
print(f"- Accuracy: {accuracy:.4f}")
print(f"- F1-Score: {f1:.4f}")
print(f"- ROC-AUC:  {roc_auc:.4f}")

print("\nРегрессия")
print("Метрики:")
print(f"- MSE:  {mse:.4f}")
print(f"- RMSE: {rmse:.4f}")
print(f"- MAE:  {mae:.4f}")
print(f"- R²:   {r2:.4f}")
```

```
================================================================================
СВОДКА РЕЗУЛЬТАТОВ БЕЙЗЛАЙН МОДЕЛЕЙ РЕШАЮЩЕГО ДЕРЕВА
================================================================================

1. КЛАССИФИКАЦИЯ (Online Shoppers):
   Модель: DecisionTreeClassifier
   Метрики:
     - Accuracy: 0.8554
     - F1-Score: 0.5392
     - ROC-AUC:  0.7295

2. РЕГРЕССИЯ (Parkinson's Disease):
   Модель: DecisionTreeRegressor
   Метрики:
     - MSE:  0.6149
     - RMSE: 0.7842
     - MAE:  0.2016
     - R²:   0.9945
```

Улучшение бейзлайна

Классификация

Сохранение метрик базовой модели

In [29]:
```
class_base_metrics = {
    'Accuracy': accuracy,
    'F1': f1,
    'ROC-AUC': roc_auc,
    'Precision': precision,
    'Recall': recall
}
```

Функция сравнения метрик новой модели с базовой

In [30]:
```
def print_comparison_class(metrics_old, metrics_new):
    comparison_data = []
    for metric in ['Accuracy', 'F1', 'ROC-AUC', 'Precision', 'Recall']:
        base_val = metrics_old[metric]
        new_val = metrics_new[metric]
        diff = new_val - base_val
        change = "улучшение" if diff > 0 else "ухудшение"

        comparison_data.append({
            'Метрика': metric,
            'Базовая модель': f"{base_val:.4f}",
            'Новая модель': f"{new_val:.4f}",
            'Разница': f"{diff:+.4f}",
            'Изменение': change
        })

    df_comparison = pd.DataFrame(comparison_data)
    print(df_comparison.to_string(index=False))
```

Повторное копирование, разделение и масштабирование данных

In [31]:
```
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le

X = df_class_clean.drop('Revenue', axis=1)
y = df_class_clean['Revenue']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Гипотеза 1: Ограничение глубины дерева и минимальное число выборок на листе

In [32]:
```
dt_depth = DecisionTreeClassifier(max_depth=10, min_samples_leaf=5, random_state
dt_depth.fit(X_train_scaled, y_train)

y_pred_depth = dt_depth.predict(X_test_scaled)
y_pred_proba_depth = dt_depth.predict_proba(X_test_scaled)[:, 1]

metrics_depth = {
    'Accuracy': accuracy_score(y_test, y_pred_depth),
    'F1': f1_score(y_test, y_pred_depth),
    'ROC-AUC': roc_auc_score(y_test, y_pred_proba_depth),
    'Precision': precision_score(y_test, y_pred_depth),
    'Recall': recall_score(y_test, y_pred_depth)
}
```

```
print("Ограничение глубины дерева (max_depth=10, min_samples_leaf=5)")
print_comparison_class(class_base_metrics, metrics_depth)
```

```
Ограничение глубины дерева (max_depth=10, min_samples_leaf=5)
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8554       0.8781 +0.0227 улучшение
       F1         0.5392       0.5659 +0.0267 улучшение
  ROC-AUC         0.7295       0.8550 +0.1255 улучшение
Precision         0.5314       0.6296 +0.0981 улучшение
   Recall         0.5472       0.5140 -0.0332 ухудшение
```

Гипотеза 2: Отбор признаков

In [33]:
```
selector = SelectKBest(f_classif, k=10)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)

dt_selected = DecisionTreeClassifier(random_state=42)
dt_selected.fit(X_train_selected, y_train)

y_pred_sel = dt_selected.predict(X_test_selected)
y_pred_proba_sel = dt_selected.predict_proba(X_test_selected)[:, 1]

metrics_sel = {
    'Accuracy': accuracy_score(y_test, y_pred_sel),
    'F1': f1_score(y_test, y_pred_sel),
    'ROC-AUC': roc_auc_score(y_test, y_pred_proba_sel),
    'Precision': precision_score(y_test, y_pred_sel),
    'Recall': recall_score(y_test, y_pred_sel)
}

print("Отбор 10 лучших признаков")
print_comparison_class(class_base_metrics, metrics_sel)
```

```
Отбор 10 лучших признаков
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8554       0.8489 -0.0065 ухудшение
       F1         0.5392       0.5143 -0.0249 ухудшение
  ROC-AUC         0.7295       0.7027 -0.0268 ухудшение
Precision         0.5314       0.5112 -0.0202 ухудшение
   Recall         0.5472       0.5175 -0.0297 ухудшение
```

Гипотеза 3: Добавление полиномиальных признаков

In [34]:
```
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

print(f"Исходное количество признаков: {X_train_scaled.shape[1]}")
print(f"Количество признаков с полиномиальными: {X_train_poly.shape[1]}")

dt_poly = DecisionTreeClassifier(random_state=42)
dt_poly.fit(X_train_poly, y_train)

y_pred_poly = dt_poly.predict(X_test_poly)
y_pred_proba_poly = dt_poly.predict_proba(X_test_poly)[:, 1]

metrics_poly = {
    'Accuracy': accuracy_score(y_test, y_pred_poly),
    'F1': f1_score(y_test, y_pred_poly),
```

```
    'ROC-AUC': roc_auc_score(y_test, y_pred_proba_poly),
    'Precision': precision_score(y_test, y_pred_poly),
    'Recall': recall_score(y_test, y_pred_poly)
}

print("Полиномиальные признаки")
print_comparison_class(class_base_metrics, metrics_poly)
```

```
Исходное количество признаков: 17
Количество признаков с полиномиальными: 153
Полиномиальные признаки (степень 2)
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8554       0.8454 -0.0100 ухудшение
       F1         0.5392       0.5225 -0.0167 ухудшение
  ROC-AUC         0.7295       0.7236 -0.0059 ухудшение
Precision         0.5314       0.5000 -0.0314 ухудшение
   Recall         0.5472       0.5472 +0.0000 ухудшение
Полиномиальные признаки (степень 2)
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8554       0.8454 -0.0100 ухудшение
       F1         0.5392       0.5225 -0.0167 ухудшение
  ROC-AUC         0.7295       0.7236 -0.0059 ухудшение
Precision         0.5314       0.5000 -0.0314 ухудшение
   Recall         0.5472       0.5472 +0.0000 ухудшение
```

Гипотеза 4: Подбор гиперпараметров

In [35]:

```
warnings.filterwarnings('ignore')

param_grid = {
    'max_depth': [5, 8, 10, 12, 15, 20, 30],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 5, 10],
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_features': [None, 'sqrt', 'log2'],
    'random_state': [42]
}

dt_gs = DecisionTreeClassifier()
random_search = RandomizedSearchCV(
    estimator=dt_gs, param_distributions=param_grid, n_iter=50, cv=5,
    scoring='f1', random_state=42, n_jobs=-1
)
random_search.fit(X_train_scaled, y_train)

print("Лучшие параметры DecisionTree:")
for param, value in random_search.best_params_.items():
    print(f"  {param}: {value}")

best_dt = random_search.best_estimator_
y_pred_gs = best_dt.predict(X_test_scaled)
y_pred_proba_gs = best_dt.predict_proba(X_test_scaled)[:, 1]

metrics_gs = {
    'Accuracy': accuracy_score(y_test, y_pred_gs),
    'F1': f1_score(y_test, y_pred_gs),
    'ROC-AUC': roc_auc_score(y_test, y_pred_proba_gs),
    'Precision': precision_score(y_test, y_pred_gs),
    'Recall': recall_score(y_test, y_pred_gs)
```

```
}

print("Подбор гиперпараметров")
print_comparison_class(class_base_metrics, metrics_gs)
```

```
Лучшие параметры DecisionTree:
  splitter: best
  random_state: 42
  min_samples_split: 20
  min_samples_leaf: 10
  max_features: sqrt
  max_depth: 12
  criterion: gini
Подбор гиперпараметров
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8554       0.8786 +0.0232 улучшение
       F1         0.5392       0.5559 +0.0167 улучшение
  ROC-AUC         0.7295       0.8750 +0.1455 улучшение
Precision         0.5314       0.6401 +0.1087 улучшение
   Recall         0.5472       0.4913 -0.0559 ухудшение
```

Формирование улучшенной модели и её обучение

In [36]:
```
best_params = random_search.best_params_.copy()

improved_dt = DecisionTreeClassifier(**best_params)
improved_dt.fit(X_train_scaled, y_train)

y_pred_improved = improved_dt.predict(X_test_scaled)
y_pred_proba_improved = improved_dt.predict_proba(X_test_scaled)[:, 1]
```

Метрики улучшенной модели

In [37]:
```
class_improved_metrics = {
    'Accuracy': accuracy_score(y_test, y_pred_improved),
    'F1': f1_score(y_test, y_pred_improved),
    'ROC-AUC': roc_auc_score(y_test, y_pred_proba_improved),
    'Precision': precision_score(y_test, y_pred_improved),
    'Recall': recall_score(y_test, y_pred_improved)
}

for metric, value in class_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8786
F1: 0.5559
ROC-AUC: 0.8750
Precision: 0.6401
Recall: 0.4913
```

Сравнение улучшенной модели с базовой

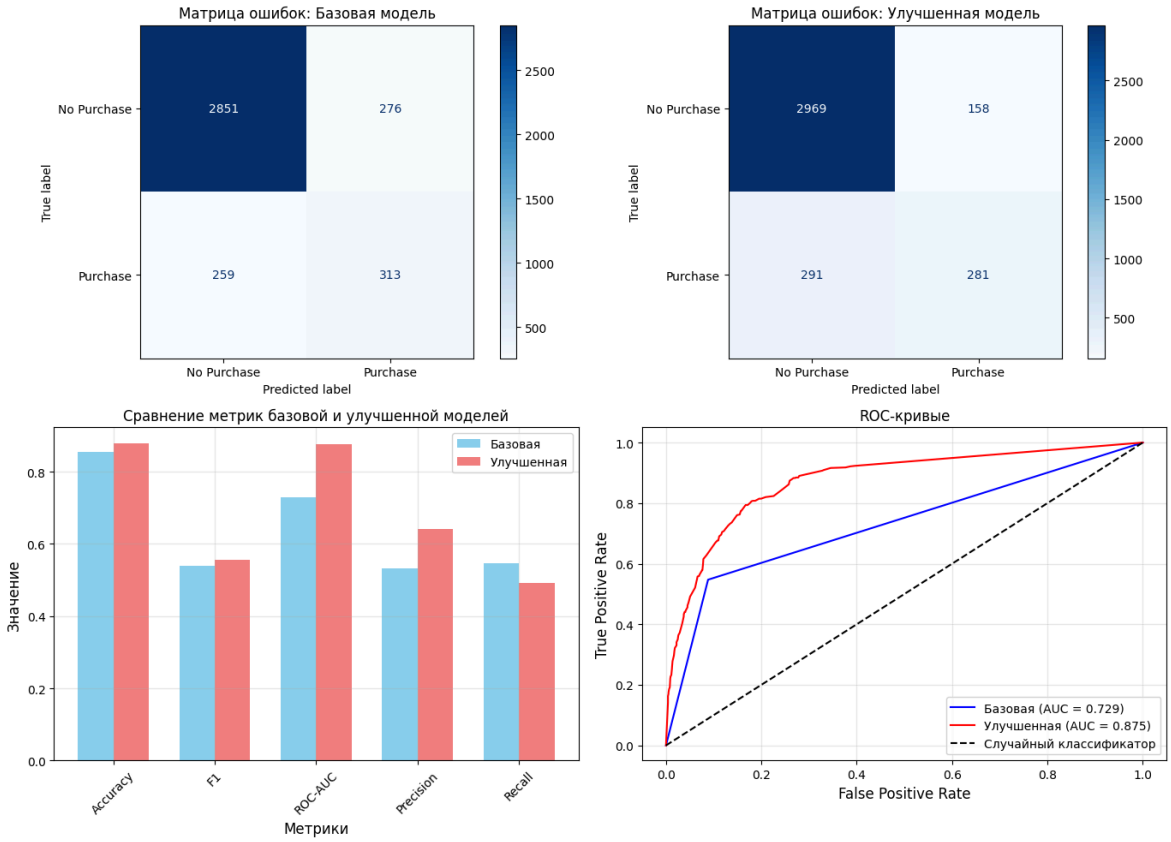
In [38]:
```
print_comparison_class(class_base_metrics, class_improved_metrics)
```

```
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy          0.8554       0.8786 +0.0232 улучшение
       F1          0.5392       0.5559 +0.0167 улучшение
  ROC-AUC          0.7295       0.8750 +0.1455 улучшение
Precision          0.5314       0.6401 +0.1087 улучшение
   Recall          0.5472       0.4913 -0.0559 ухудшение
```

Визуальное сравнение базовой и улучшенной модели

In [39]:

```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes[0, 0].set_title('Матрица ошибок: Базовая модель', fontsize=12)
cm_base = confusion_matrix(y_test, y_pred)
disp_base = ConfusionMatrixDisplay(confusion_matrix=cm_base,
                                   display_labels=['No Purchase', 'Purchase'])
disp_base.plot(ax=axes[0, 0], cmap='Blues')

axes[0, 1].set_title('Матрица ошибок: Улучшенная модель', fontsize=12)
cm_improved = confusion_matrix(y_test, y_pred_improved)
disp_improved = ConfusionMatrixDisplay(confusion_matrix=cm_improved,
                                       display_labels=['No Purchase', 'Purchase'
disp_improved.plot(ax=axes[0, 1], cmap='Blues')

metrics_names = ['Accuracy', 'F1', 'ROC-AUC', 'Precision', 'Recall']
base_values = [class_base_metrics[m] for m in metrics_names]
improved_values = [class_improved_metrics[m] for m in metrics_names]

x = np.arange(len(metrics_names))
width = 0.35

axes[1, 0].bar(x - width/2, base_values, width, label='Базовая', color='skyblue'
axes[1, 0].bar(x + width/2, improved_values, width, label='Улучшенная', color='l
axes[1, 0].set_xlabel('Метрики', fontsize=12)
axes[1, 0].set_ylabel('Значение', fontsize=12)
axes[1, 0].set_title('Сравнение метрик базовой и улучшенной моделей', fontsize=1
axes[1, 0].set_xticks(x)
axes[1, 0].set_xticklabels(metrics_names, rotation=45)
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

fpr_base, tpr_base, _ = roc_curve(y_test, y_pred_proba)
fpr_improved, tpr_improved, _ = roc_curve(y_test, y_pred_proba_improved)

axes[1, 1].plot(fpr_base, tpr_base, label=f'Базовая (AUC = {roc_auc:.3f})', colo
axes[1, 1].plot(fpr_improved, tpr_improved, label=f'Улучшенная (AUC = {class_imp
axes[1, 1].plot([0, 1], [0, 1], 'k--', label='Случайный классификатор')
axes[1, 1].set_xlabel('False Positive Rate', fontsize=12)
axes[1, 1].set_ylabel('True Positive Rate', fontsize=12)
axes[1, 1].set_title('ROC-кривые', fontsize=12)
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

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



Анализ результатов классификации

In [40]:
```
TN_base, FP_base, FN_base, TP_base = cm_base.ravel()
TN_imp, FP_imp, FN_imp, TP_imp = cm_improved.ravel()

print("\n1. Анализ обнаружения покупок:")
print(f"   Базовая модель нашла {TP_base} из {TP_base + FN_base} реальных покупо
print(f"   Улучшенная модель нашла {TP_imp} из {TP_imp + FN_imp} реальных покупо
print(f"   Улучшение в обнаружении покупок: {TP_imp - TP_base} реальных покупок"
print(f"   Процентное улучшение Recall: {((TP_imp / (TP_imp + FN_imp)) - (TP_bas

print("\n2. Анализ ложных срабатываний:")
print(f"   Базовая модель: {FP_base} ложных предсказаний покупки")
print(f"   Улучшенная модель: {FP_imp} ложных предсказаний покупки")
print(f"   Изменение: {FP_imp - FP_base} дополнительных ложных срабатываний")
```

1. Анализ обнаружения покупок:
   Базовая модель нашла 313 из 572 реальных покупок
   Улучшенная модель нашла 281 из 572 реальных покупок
   Улучшение в обнаружении покупок: -32 реальных покупок
   Процентное улучшение Recall: -5.6%

2. Анализ ложных срабатываний:
   Базовая модель: 276 ложных предсказаний покупки
   Улучшенная модель: 158 ложных предсказаний покупки
   Изменение: -118 дополнительных ложных срабатываний

Регрессия

Сохранение метрик базовой модели

In [41]:
```
reg_base_metrics = {
    'MSE': mse,
```

```
    'RMSE': rmse,
    'MAE': mae,
    'R²': r2
}
```

Функция сравнения метрик новой модели с базовой

In [42]:
```
def print_comparison_reg(metrics_old, metrics_new):
    comparison_data = []
    for metric in ['MSE', 'RMSE', 'MAE', 'R²']:
        base_val = metrics_old[metric]
        new_val = metrics_new[metric]
        diff = new_val - base_val
        if metric == 'R²':
            change = "улучшение" if diff > 0 else "ухудшение"
        else:
            change = "улучшение" if diff < 0 else "ухудшение"

        comparison_data.append({
            'Метрика': metric,
            'Базовая модель': f"{base_val:.4f}",
            'Новая модель': f"{new_val:.4f}",
            'Разница': f"{diff:+.4f}",
            'Изменение': change
        })

    df_comparison = pd.DataFrame(comparison_data)
    print(df_comparison.to_string(index=False))
```

Повторное копирование и подготовка данных

In [43]:
```
df_reg_clean = df_reg.copy()
df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X = df_reg_clean.drop('total_UPDRS', axis=1)
y = df_reg_clean['total_UPDRS']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Гипотеза 1: Ограничение глубины дерева

In [44]:
```
dt_reg_depth = DecisionTreeRegressor(max_depth=10, min_samples_leaf=5, random_st
dt_reg_depth.fit(X_train_scaled, y_train)
y_pred_depth = dt_reg_depth.predict(X_test_scaled)

metrics_depth = {
    'MSE': mean_squared_error(y_test, y_pred_depth),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_depth)),
    'MAE': mean_absolute_error(y_test, y_pred_depth),
    'R²': r2_score(y_test, y_pred_depth)
}
```

```
print("Ограничение глубины дерева (max_depth=10, min_samples_leaf=5)")
print_comparison_reg(reg_base_metrics, metrics_depth)
```

```
Ограничение глубины дерева (max_depth=10, min_samples_leaf=5)
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.6149       0.8641 +0.2491 ухудшение
   RMSE         0.7842       0.9295 +0.1454 ухудшение
    MAE         0.2016       0.4936 +0.2920 ухудшение
     R²         0.9945       0.9923 -0.0022 ухудшение
```

Гипотеза 2: Обработка выбросов с помощью RobustScaler

In [45]:
```
robust_scaler = RobustScaler()
X_train_robust = robust_scaler.fit_transform(X_train)
X_test_robust = robust_scaler.transform(X_test)

dt_reg_robust = DecisionTreeRegressor(random_state=42)
dt_reg_robust.fit(X_train_robust, y_train)
y_pred_robust = dt_reg_robust.predict(X_test_robust)

metrics_robust = {
    'MSE': mean_squared_error(y_test, y_pred_robust),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_robust)),
    'MAE': mean_absolute_error(y_test, y_pred_robust),
    'R²': r2_score(y_test, y_pred_robust)
}

print("RobustScaler (устойчивое масштабирование)")
print_comparison_reg(reg_base_metrics, metrics_robust)
```

```
RobustScaler (устойчивое масштабирование)
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.6149       0.6149 +0.0000 ухудшение
   RMSE         0.7842       0.7842 +0.0000 ухудшение
    MAE         0.2016       0.2016 +0.0000 ухудшение
     R²         0.9945       0.9945 +0.0000 ухудшение
```

Гипотеза 3: Логарифмическое преобразование целевой переменной

In [46]:
```
y_train_log = np.log1p(y_train)
y_test_log = np.log1p(y_test)

dt_reg_log = DecisionTreeRegressor(random_state=42)
dt_reg_log.fit(X_train_scaled, y_train_log)
y_pred_log = dt_reg_log.predict(X_test_scaled)

y_pred_exp = np.expm1(y_pred_log)

metrics_log = {
    'MSE': mean_squared_error(y_test, y_pred_exp),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_exp)),
    'MAE': mean_absolute_error(y_test, y_pred_exp),
    'R²': r2_score(y_test, y_pred_exp)
}

print("Логарифмирование целевой переменной")
print_comparison_reg(reg_base_metrics, metrics_log)
```

```
Логарифмирование целевой переменной
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.6149       0.3479 -0.2670 улучшение
   RMSE         0.7842       0.5899 -0.1943 улучшение
    MAE         0.2016       0.1355 -0.0661 улучшение
     R²         0.9945       0.9969 +0.0024 улучшение
```

Гипотеза 4: Подбор гиперпараметров

In [47]:
```python
warnings.filterwarnings('ignore')

param_grid_reg = {
    'max_depth': [5, 8, 10, 12, 15, 20, 30],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 5, 10],
    'criterion': ['squared_error', 'absolute_error', 'friedman_mse', 'poisson'],
    'splitter': ['best', 'random'],
    'max_features': [None, 'sqrt', 'log2'],
    'random_state': [42]
}

dt_gs_reg = DecisionTreeRegressor()
random_search_reg = RandomizedSearchCV(
    estimator=dt_gs_reg, param_distributions=param_grid_reg, n_iter=50, cv=5,
    scoring='r2', random_state=42, n_jobs=-1
)
random_search_reg.fit(X_train_scaled, y_train)

print("Лучшие параметры DecisionTree (регрессия):")
for param, value in random_search_reg.best_params_.items():
    print(f"  {param}: {value}")

best_dt_reg = random_search_reg.best_estimator_
y_pred_gs_reg = best_dt_reg.predict(X_test_scaled)

metrics_gs_reg = {
    'MSE': mean_squared_error(y_test, y_pred_gs_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_gs_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_gs_reg),
    'R²': r2_score(y_test, y_pred_gs_reg)
}

print("Подбор гиперпараметров")
print_comparison_reg(reg_base_metrics, metrics_gs_reg)
```

```
Лучшие параметры DecisionTree (регрессия):
  splitter: best
  random_state: 42
  min_samples_split: 2
  min_samples_leaf: 2
  max_features: None
  max_depth: 12
  criterion: squared_error
Подбор гиперпараметров
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.6149       0.5519 -0.0630 улучшение
   RMSE         0.7842       0.7429 -0.0413 улучшение
    MAE         0.2016       0.2639 +0.0623 ухудшение
     R²         0.9945       0.9951 +0.0006 улучшение
```

Формирование улучшенной модели и её обучение

In [48]:
```
best_params_reg = random_search_reg.best_params_.copy()

improved_dt_reg = DecisionTreeRegressor(**best_params_reg)
improved_dt_reg.fit(X_train_scaled, y_train)

y_pred_improved_reg = improved_dt_reg.predict(X_test_scaled)
```

Метрики улучшенной модели

In [49]:
```
reg_improved_metrics = {
    'MSE': mean_squared_error(y_test, y_pred_improved_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_improved_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_improved_reg),
    'R²': r2_score(y_test, y_pred_improved_reg)
}

for metric, value in reg_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 0.5519
RMSE: 0.7429
MAE: 0.2639
R²: 0.9951
```

Сравнение улучшенной модели с базовой

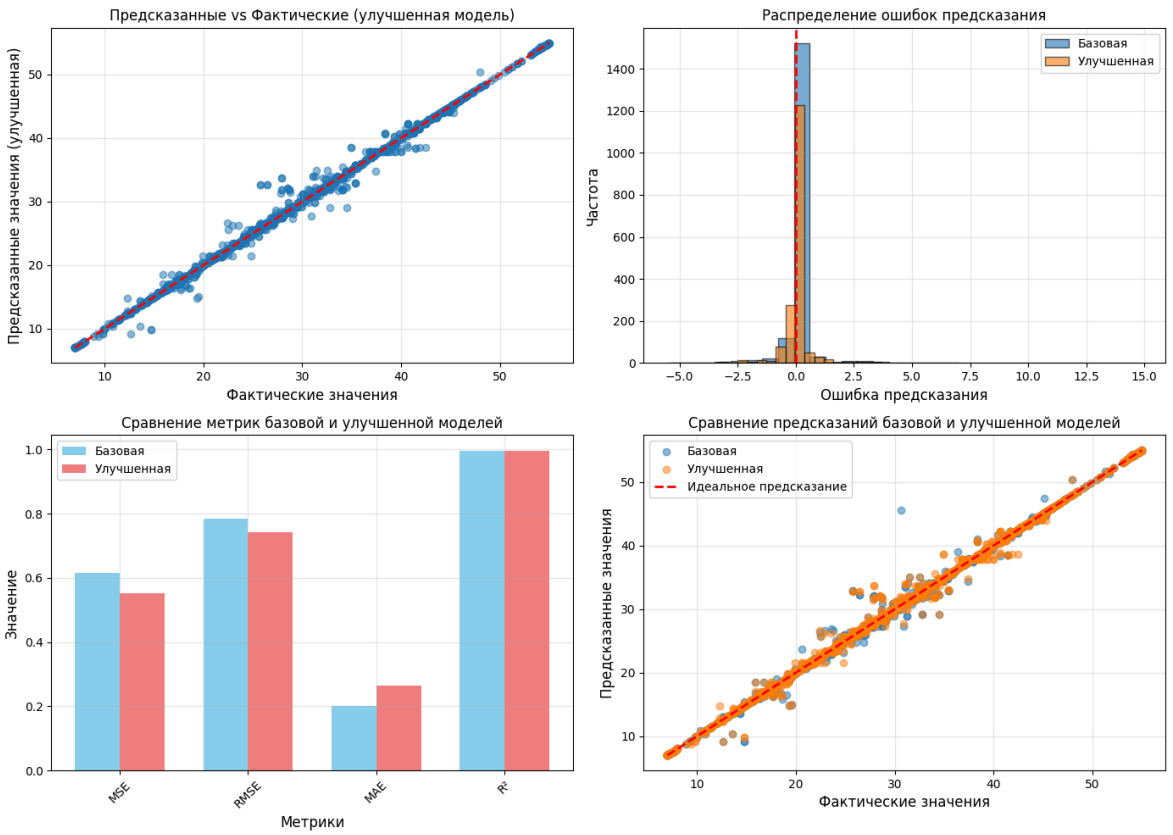
In [50]:
```
print_comparison_reg(reg_base_metrics, reg_improved_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.6149       0.5519 -0.0630 улучшение
   RMSE         0.7842       0.7429 -0.0413 улучшение
    MAE         0.2016       0.2639 +0.0623 ухудшение
     R²         0.9945       0.9951 +0.0006 улучшение
```

Визуальное сравнение базовой и улучшенной модели

In [51]:
```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes[0, 0].scatter(y_test, y_pred_improved_reg, alpha=0.5)
axes[0, 0].plot([y_test.min(), y_test.max()],
                [y_test.min(), y_test.max()],
                'r--', lw=2)
axes[0, 0].set_xlabel('Фактические значения', fontsize=12)
axes[0, 0].set_ylabel('Предсказанные значения (улучшенная)', fontsize=12)
axes[0, 0].set_title('Предсказанные vs Фактические (улучшенная модель)', fontsiz
axes[0, 0].grid(True, alpha=0.3)

errors_base = y_pred_reg - y_test
errors_imp = y_pred_improved_reg - y_test

axes[0, 1].hist(errors_base, bins=30, alpha=0.6, label='Базовая', edgecolor='bla
axes[0, 1].hist(errors_imp, bins=30, alpha=0.6, label='Улучшенная', edgecolor='b
axes[0, 1].axvline(x=0, color='r', linestyle='--', linewidth=2)
axes[0, 1].set_xlabel('Ошибка предсказания', fontsize=12)
axes[0, 1].set_ylabel('Частота', fontsize=12)
axes[0, 1].set_title('Распределение ошибок предсказания', fontsize=12)
```

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

metrics_names = ['MSE', 'RMSE', 'MAE', 'R²']
base_values = [reg_base_metrics[m] for m in metrics_names]
improved_values = [reg_improved_metrics[m] for m in metrics_names]

x = np.arange(len(metrics_names))
width = 0.35

axes[1, 0].bar(x - width/2, base_values, width, label='Базовая', color='skyblue'
axes[1, 0].bar(x + width/2, improved_values, width, label='Улучшенная', color='l
axes[1, 0].set_xlabel('Метрики', fontsize=12)
axes[1, 0].set_ylabel('Значение', fontsize=12)
axes[1, 0].set_title('Сравнение метрик базовой и улучшенной моделей', fontsize=1
axes[1, 0].set_xticks(x)
axes[1, 0].set_xticklabels(metrics_names, rotation=45)
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

axes[1, 1].scatter(y_test, y_pred_reg, alpha=0.5, label='Базовая')
axes[1, 1].scatter(y_test, y_pred_improved_reg, alpha=0.5, label='Улучшенная')
axes[1, 1].plot([y_test.min(), y_test.max()],
                [y_test.min(), y_test.max()],
                'r--', lw=2, label='Идеальное предсказание')
axes[1, 1].set_xlabel('Фактические значения', fontsize=12)
axes[1, 1].set_ylabel('Предсказанные значения', fontsize=12)
axes[1, 1].set_title('Сравнение предсказаний базовой и улучшенной моделей', font
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

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



Имплементация алгоритма машинного обучения

Классификация

Кастомная модель дерева решений для классификации

In [52]:
```
class CustomDecisionTreeClassifier:

    def __init__(self, max_depth=None, min_samples_split=2, min_samples_leaf=1,
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf
        self.criterion = criterion
        self.tree = None

    class Node:
        def __init__(self, feature=None, threshold=None, left=None, right=None,
                     value=None, samples=None, impurity=None):
            self.feature = feature
            self.threshold = threshold
            self.left = left
            self.right = right
            self.value = value
            self.samples = samples
            self.impurity = impurity

    def _gini(self, y):
        _, counts = np.unique(y, return_counts=True)
        probabilities = counts / len(y)
        gini = 1.0 - np.sum(probabilities ** 2)
        return gini

    def _entropy(self, y):
        _, counts = np.unique(y, return_counts=True)
        probabilities = counts / len(y)
        entropy = -np.sum(probabilities * np.log2(probabilities + 1e-15))
        return entropy

    def _calculate_impurity(self, y):
        if self.criterion == 'gini':
            return self._gini(y)
        else:
            return self._entropy(y)

    def _information_gain(self, parent, left_child, right_child):
        n = len(parent)
        n_left = len(left_child)
        n_right = len(right_child)

        if n_left == 0 or n_right == 0:
            return 0

        parent_impurity = self._calculate_impurity(parent)
        left_impurity = self._calculate_impurity(left_child)
        right_impurity = self._calculate_impurity(right_child)
```

```
        child_impurity = (n_left / n) * left_impurity + (n_right / n) * right_im
        information_gain = parent_impurity - child_impurity

        return information_gain

    def _best_split(self, X, y):
        best_gain = -1
        best_feature = None
        best_threshold = None

        for feature in range(X.shape[1]):
            thresholds = np.unique(X[:, feature])

            for threshold in thresholds:
                left_mask = X[:, feature] <= threshold
                right_mask = ~left_mask

                if np.sum(left_mask) < self.min_samples_leaf or np.sum(right_mas
                    continue

                information_gain = self._information_gain(y, y[left_mask], y[rig

                if information_gain > best_gain:
                    best_gain = information_gain
                    best_feature = feature
                    best_threshold = threshold

        return best_feature, best_threshold

    def _build_tree(self, X, y, depth=0):
        n_samples = X.shape[0]
        n_classes = len(np.unique(y))

        if (self.max_depth is not None and depth >= self.max_depth or
            n_samples < self.min_samples_split or
            n_classes == 1):

            value = np.bincount(y.astype(int), minlength=2)
            return self.Node(value=value, samples=n_samples,
                             impurity=self._calculate_impurity(y))

        best_feature, best_threshold = self._best_split(X, y)

        if best_feature is None:
            value = np.bincount(y.astype(int), minlength=2)
            return self.Node(value=value, samples=n_samples,
                             impurity=self._calculate_impurity(y))

        left_mask = X[:, best_feature] <= best_threshold
        right_mask = ~left_mask

        left_subtree = self._build_tree(X[left_mask], y[left_mask], depth + 1)
        right_subtree = self._build_tree(X[right_mask], y[right_mask], depth + 1

        return self.Node(feature=best_feature, threshold=best_threshold,
                         left=left_subtree, right=right_subtree,
                         samples=n_samples, impurity=self._calculate_impurity(y))

    def fit(self, X, y):
```

```
        self.tree = self._build_tree(X, y)
        return self

    def _traverse_tree(self, x, node):
        if node.value is not None:
            return node.value

        if x[node.feature] <= node.threshold:
            return self._traverse_tree(x, node.left)
        else:
            return self._traverse_tree(x, node.right)

    def predict_proba(self, X):
        probabilities = []
        for x in X:
            counts = self._traverse_tree(x, self.tree)
            proba = counts / counts.sum()
            probabilities.append(proba)

        return np.array(probabilities)

    def predict(self, X):
        predictions = []
        for x in X:
            counts = self._traverse_tree(x, self.tree)
            prediction = np.argmax(counts)
            predictions.append(prediction)

        return np.array(predictions)
```

Повторное копирование и разбиение данных

In [53]:
```
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le

X_class = df_class_clean.drop('Revenue', axis=1)
y_class = df_class_clean['Revenue']

X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class, test_size=0.3, random_state=42, stratify=y_class
)

scaler_class = StandardScaler()
X_train_class_scaled = scaler_class.fit_transform(X_train_class)
X_test_class_scaled = scaler_class.transform(X_test_class)
```

Обучение кастомного дерева решений (базовое)

In [54]:
```
custom_dt = CustomDecisionTreeClassifier(criterion='gini')
custom_dt.fit(X_train_class_scaled, y_train_class.values)
```

```
y_pred_custom = custom_dt.predict(X_test_class_scaled)
y_pred_proba_custom = custom_dt.predict_proba(X_test_class_scaled)[:, 1]
```

Метрики кастомного дерева решений

In [55]:
```
custom_class_base_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_custom),
    'F1': f1_score(y_test_class, y_pred_custom),
    'ROC-AUC': roc_auc_score(y_test_class, y_pred_proba_custom),
    'Precision': precision_score(y_test_class, y_pred_custom),
    'Recall': recall_score(y_test_class, y_pred_custom)
}

for metric, value in custom_class_base_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8486
F1: 0.5189
ROC-AUC: 0.7176
Precision: 0.5101
Recall: 0.5280
```

Сравнение кастомной модели с базовой из sklearn

In [56]:
```
print_comparison_class(class_base_metrics, custom_class_base_metrics)
```

```
 Метрика Базовая модель Новая модель Разница Изменение
Accuracy         0.8554       0.8486 -0.0068 ухудшение
      F1         0.5392       0.5189 -0.0203 ухудшение
  ROC-AUC         0.7295       0.7176 -0.0119 ухудшение
Precision         0.5314       0.5101 -0.0213 ухудшение
   Recall         0.5472       0.5280 -0.0192 ухудшение
```

Обучение улучшенного кастомного дерева решений

In [57]:
```
improved_custom_dt = CustomDecisionTreeClassifier(
    max_depth=10,
    min_samples_split=10,
    min_samples_leaf=5,
    criterion='entropy'
)
improved_custom_dt.fit(X_train_class_scaled, y_train_class.values)

y_pred_imp_custom = improved_custom_dt.predict(X_test_class_scaled)
y_pred_proba_imp_custom = improved_custom_dt.predict_proba(X_test_class_scaled)[
```

Метрики улучшенного кастомного дерева решений

In [58]:
```
custom_improved_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_imp_custom),
    'F1': f1_score(y_test_class, y_pred_imp_custom),
    'ROC-AUC': roc_auc_score(y_test_class, y_pred_proba_imp_custom),
    'Precision': precision_score(y_test_class, y_pred_imp_custom),
    'Recall': recall_score(y_test_class, y_pred_imp_custom)
}

for metric, value in custom_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8759
F1: 0.5565
ROC-AUC: 0.8318
Precision: 0.6220
Recall: 0.5035
```

Сравнение улучшенной кастомной модели с улучшенной из sklearn

In [59]:
```
print_comparison_class(class_improved_metrics, custom_improved_metrics)
```

```
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8786       0.8759 -0.0027 ухудшение
       F1         0.5559       0.5565 +0.0006 улучшение
  ROC-AUC         0.8750       0.8318 -0.0432 ухудшение
Precision         0.6401       0.6220 -0.0181 ухудшение
   Recall         0.4913       0.5035 +0.0122 улучшение
```

Итоговое сравнение всех моделей классификации

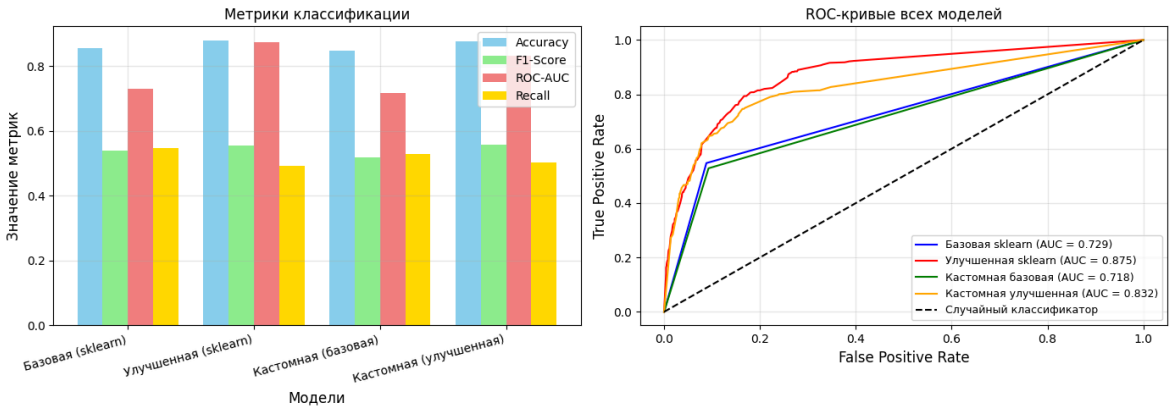
In [60]:
```
summary_class = pd.DataFrame({
    'Тип модели': ['Базовая (sklearn)', 'Улучшенная (sklearn)', 'Кастомная (базо
    'Accuracy': [class_base_metrics['Accuracy'], class_improved_metrics['Accurac
    'F1-Score': [class_base_metrics['F1'], class_improved_metrics['F1'], custom_
    'ROC-AUC': [class_base_metrics['ROC-AUC'], class_improved_metrics['ROC-AUC']
    'Recall': [class_base_metrics['Recall'], class_improved_metrics['Recall'], c
})

print("Сводная таблица моделей классификации")
print(summary_class.to_string(index=False))
```

```
Сводная таблица моделей классификации
            Тип модели  Accuracy  F1-Score  ROC-AUC   Recall
    Базовая (sklearn)  0.855366  0.539190 0.729470 0.547203
 Улучшенная (sklearn)  0.878616  0.555885 0.874990 0.491259
  Кастомная (базовая)  0.848608  0.518900 0.717616 0.527972
Кастомная (улучшенная)  0.875912  0.556522 0.831831 0.503497
```

Визуализация сравнения всех моделей классификации

In [61]:
```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

x = np.arange(len(summary_class))
width = 0.2

axes[0].bar(x - width*1.5, summary_class['Accuracy'], width, label='Accuracy', c
axes[0].bar(x - width/2, summary_class['F1-Score'], width, label='F1-Score', col
axes[0].bar(x + width/2, summary_class['ROC-AUC'], width, label='ROC-AUC', color
axes[0].bar(x + width*1.5, summary_class['Recall'], width, label='Recall', color
axes[0].set_xlabel('Модели', fontsize=12)
axes[0].set_ylabel('Значение метрик', fontsize=12)
axes[0].set_title('Метрики классификации', fontsize=12)
axes[0].set_xticks(x)
axes[0].set_xticklabels(summary_class['Тип модели'], rotation=15, ha='right')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

fpr_custom, tpr_custom, _ = roc_curve(y_test_class, y_pred_proba_custom)
fpr_improved_custom, tpr_improved_custom, _ = roc_curve(y_test_class, y_pred_pro

axes[1].plot(fpr_base, tpr_base, label=f'Базовая sklearn (AUC = {class_base_metr
```

```
axes[1].plot(fpr_improved, tpr_improved, label=f'Улучшенная sklearn (AUC = {clas
axes[1].plot(fpr_custom, tpr_custom, label=f'Кастомная базовая (AUC = {custom_cl
axes[1].plot(fpr_improved_custom, tpr_improved_custom, label=f'Кастомная улучшен
axes[1].plot([0, 1], [0, 1], 'k--', label='Случайный классификатор')
axes[1].set_xlabel('False Positive Rate', fontsize=12)
axes[1].set_ylabel('True Positive Rate', fontsize=12)
axes[1].set_title('ROC-кривые всех моделей', fontsize=12)
axes[1].legend(fontsize=9)
axes[1].grid(True, alpha=0.3)

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



Регрессия

Кастомная модель дерева решений для регрессии

In [62]:
```
class CustomDecisionTreeRegressor:

    def __init__(self, max_depth=None, min_samples_split=2, min_samples_leaf=1,
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf
        self.criterion = criterion
        self.tree = None

    class Node:
        def __init__(self, feature=None, threshold=None, left=None, right=None,
                     value=None, samples=None, mse=None):
            self.feature = feature
            self.threshold = threshold
            self.left = left
            self.right = right
            self.value = value
            self.samples = samples
            self.mse = mse

    def _mse(self, y):
        return np.mean((y - np.mean(y)) ** 2)

    def _mae(self, y):
        return np.mean(np.abs(y - np.median(y)))

    def _calculate_loss(self, y):
        if self.criterion == 'mse':
            return self._mse(y)
```

```
        else:
            return self._mae(y)

    def _mse_reduction(self, parent, left_child, right_child):
        n = len(parent)
        n_left = len(left_child)
        n_right = len(right_child)

        if n_left == 0 or n_right == 0:
            return 0

        parent_mse = self._mse(parent)
        left_mse = self._mse(left_child)
        right_mse = self._mse(right_child)

        weighted_mse = (n_left / n) * left_mse + (n_right / n) * right_mse
        mse_reduction = parent_mse - weighted_mse

        return mse_reduction

    def _best_split(self, X, y):
        best_reduction = -1
        best_feature = None
        best_threshold = None

        for feature in range(X.shape[1]):
            thresholds = np.unique(X[:, feature])

            for threshold in thresholds:
                left_mask = X[:, feature] <= threshold
                right_mask = ~left_mask

                if np.sum(left_mask) < self.min_samples_leaf or np.sum(right_mas
                    continue

                mse_reduction = self._mse_reduction(y, y[left_mask], y[right_mas

                if mse_reduction > best_reduction:
                    best_reduction = mse_reduction
                    best_feature = feature
                    best_threshold = threshold

        return best_feature, best_threshold

    def _build_tree(self, X, y, depth=0):
        n_samples = X.shape[0]

        if (self.max_depth is not None and depth >= self.max_depth or
            n_samples < self.min_samples_split):

            return self.Node(value=np.mean(y), samples=n_samples, mse=self._mse(

        best_feature, best_threshold = self._best_split(X, y)

        if best_feature is None:
            return self.Node(value=np.mean(y), samples=n_samples, mse=self._mse(

        left_mask = X[:, best_feature] <= best_threshold
        right_mask = ~left_mask
```

```
        left_subtree = self._build_tree(X[left_mask], y[left_mask], depth + 1)
        right_subtree = self._build_tree(X[right_mask], y[right_mask], depth + 1

        return self.Node(feature=best_feature, threshold=best_threshold,
                         left=left_subtree, right=right_subtree,
                         samples=n_samples, mse=self._mse(y))

    def fit(self, X, y):
        self.tree = self._build_tree(X, y)
        return self

    def _traverse_tree(self, x, node):
        if node.value is not None:
            return node.value

        if x[node.feature] <= node.threshold:
            return self._traverse_tree(x, node.left)
        else:
            return self._traverse_tree(x, node.right)

    def predict(self, X):
        predictions = []
        for x in X:
            prediction = self._traverse_tree(x, self.tree)
            predictions.append(prediction)

        return np.array(predictions)
```

Повторное копирование и подготовка данных для регрессии

In [63]:
```
df_reg_clean = df_reg.copy()
df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X_reg = df_reg_clean.drop('total_UPDRS', axis=1)
y_reg = df_reg_clean['total_UPDRS']

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.3, random_state=42
)

scaler_reg = StandardScaler()
X_train_reg_scaled = scaler_reg.fit_transform(X_train_reg)
X_test_reg_scaled = scaler_reg.transform(X_test_reg)
```

Обучение кастомного дерева решений (базовое)

In [64]:
```
custom_dt_reg = CustomDecisionTreeRegressor(criterion='mse')
custom_dt_reg.fit(X_train_reg_scaled, y_train_reg.values)

y_pred_custom_reg = custom_dt_reg.predict(X_test_reg_scaled)
```

Метрики кастомного дерева решений (регрессия)

In [65]:
```
custom_reg_base_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_custom_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test_reg, y_pred_custom_reg)),
    'MAE': mean_absolute_error(y_test_reg, y_pred_custom_reg),
    'R²': r2_score(y_test_reg, y_pred_custom_reg)
```

```
}

for metric, value in custom_reg_base_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 0.4663
RMSE: 0.6829
MAE: 0.1620
R²: 0.9959
```

Сравнение кастомной модели с базовой из sklearn

In [66]:
```
print_comparison_reg(reg_base_metrics, custom_reg_base_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.6149       0.4663 -0.1486 улучшение
   RMSE         0.7842       0.6829 -0.1013 улучшение
    MAE         0.2016       0.1620 -0.0396 улучшение
     R²         0.9945       0.9959 +0.0013 улучшение
```

Обучение улучшенного кастомного дерева решений

In [67]:
```
improved_custom_dt_reg = CustomDecisionTreeRegressor(
    max_depth=10,
    min_samples_split=10,
    min_samples_leaf=5,
    criterion='mse'
)
improved_custom_dt_reg.fit(X_train_reg_scaled, y_train_reg.values)

y_pred_imp_custom_reg = improved_custom_dt_reg.predict(X_test_reg_scaled)
```

Метрики улучшенного кастомного дерева решений

In [68]:
```
custom_reg_improved_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_imp_custom_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test_reg, y_pred_imp_custom_reg)),
    'MAE': mean_absolute_error(y_test_reg, y_pred_imp_custom_reg),
    'R²': r2_score(y_test_reg, y_pred_imp_custom_reg)
}

for metric, value in custom_reg_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 0.7920
RMSE: 0.8899
MAE: 0.4993
R²: 0.9930
```

Сравнение улучшенной кастомной модели с улучшенной из sklearn

In [69]:
```
print_comparison_reg(reg_improved_metrics, custom_reg_improved_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.5519       0.7920 +0.2401 ухудшение
   RMSE         0.7429       0.8899 +0.1470 ухудшение
    MAE         0.2639       0.4993 +0.2354 ухудшение
     R²         0.9951       0.9930 -0.0021 ухудшение
```

Итоговое сравнение всех моделей регрессии

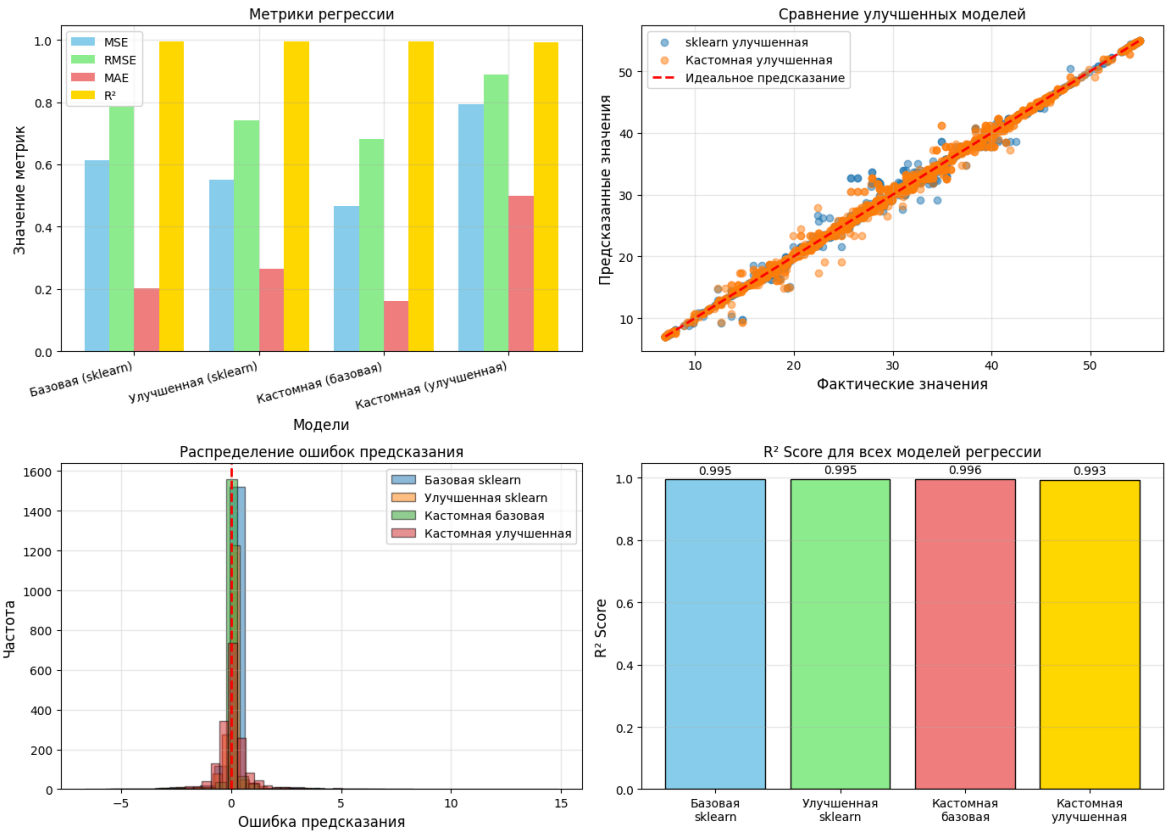
In [70]:
```
summary_reg = pd.DataFrame({
    'Тип модели': ['Базовая (sklearn)', 'Улучшенная (sklearn)', 'Кастомная (базо
    'MSE': [reg_base_metrics['MSE'], reg_improved_metrics['MSE'], custom_reg_bas
    'RMSE': [reg_base_metrics['RMSE'], reg_improved_metrics['RMSE'], custom_reg_
    'MAE': [reg_base_metrics['MAE'], reg_improved_metrics['MAE'], custom_reg_bas
    'R²': [reg_base_metrics['R²'], reg_improved_metrics['R²'], custom_reg_base_m
})

print("Сводная таблица моделей регрессии")
print(summary_reg.to_string(index=False))
```

```
Сводная таблица моделей регрессии
            Тип модели      MSE     RMSE      MAE       R²
     Базовая (sklearn) 0.614939 0.784181 0.201599 0.994533
  Улучшенная (sklearn) 0.551940 0.742926 0.263899 0.995093
   Кастомная (базовая) 0.466336 0.682888 0.162012 0.995854
Кастомная (улучшенная) 0.791997 0.889942 0.499297 0.992959
```

Визуализация сравнения всех моделей регрессии

In [71]:
```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

x = np.arange(len(summary_reg))
width = 0.2

axes[0, 0].bar(x - width*1.5, summary_reg['MSE'], width, label='MSE', color='sky
axes[0, 0].bar(x - width/2, summary_reg['RMSE'], width, label='RMSE', color='lig
axes[0, 0].bar(x + width/2, summary_reg['MAE'], width, label='MAE', color='light
axes[0, 0].bar(x + width*1.5, summary_reg['R²'], width, label='R²', color='gold'
axes[0, 0].set_xlabel('Модели', fontsize=12)
axes[0, 0].set_ylabel('Значение метрик', fontsize=12)
axes[0, 0].set_title('Метрики регрессии', fontsize=12)
axes[0, 0].set_xticks(x)
axes[0, 0].set_xticklabels(summary_reg['Тип модели'], rotation=15, ha='right')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

axes[0, 1].scatter(y_test_reg, y_pred_improved_reg, alpha=0.5, label='sklearn ул
axes[0, 1].scatter(y_test_reg, y_pred_imp_custom_reg, alpha=0.5, label='Кастомна
axes[0, 1].plot([y_test_reg.min(), y_test_reg.max()],
                [y_test_reg.min(), y_test_reg.max()],
                'r--', lw=2, label='Идеальное предсказание')
axes[0, 1].set_xlabel('Фактические значения', fontsize=12)
axes[0, 1].set_ylabel('Предсказанные значения', fontsize=12)
axes[0, 1].set_title('Сравнение улучшенных моделей', fontsize=12)
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3)

errors_base_reg = y_pred_reg - y_test_reg
errors_sklearn_imp = y_pred_improved_reg - y_test_reg
errors_custom_base = y_pred_custom_reg - y_test_reg
errors_custom_imp = y_pred_imp_custom_reg - y_test_reg

axes[1, 0].hist(errors_base_reg, bins=30, alpha=0.5, label='Базовая sklearn', ed
axes[1, 0].hist(errors_sklearn_imp, bins=30, alpha=0.5, label='Улучшенная sklear
axes[1, 0].hist(errors_custom_base, bins=30, alpha=0.5, label='Кастомная базовая
axes[1, 0].hist(errors_custom_imp, bins=30, alpha=0.5, label='Кастомная улучшенн
```

```
axes[1, 0].axvline(x=0, color='r', linestyle='--', linewidth=2)
axes[1, 0].set_xlabel('Ошибка предсказания', fontsize=12)
axes[1, 0].set_ylabel('Частота', fontsize=12)
axes[1, 0].set_title('Распределение ошибок предсказания', fontsize=12)
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

model_names = ['Базовая\nsklearn', 'Улучшенная\nsklearn', 'Кастомная\nбазовая',
r2_values = [reg_base_metrics['R²'], reg_improved_metrics['R²'], custom_reg_base
colors = ['skyblue', 'lightgreen', 'lightcoral', 'gold']

axes[1, 1].bar(range(len(model_names)), r2_values, color=colors, edgecolor='blac
axes[1, 1].set_ylabel('R² Score', fontsize=12)
axes[1, 1].set_title('R² Score для всех моделей регрессии', fontsize=12)
axes[1, 1].set_xticks(range(len(model_names)))
axes[1, 1].set_xticklabels(model_names, fontsize=10)
axes[1, 1].grid(True, alpha=0.3, axis='y')

for i, v in enumerate(r2_values):
    axes[1, 1].text(i, v + 0.01, f'{v:.3f}', ha='center', va='bottom')

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



Выводы и анализ результатов

In [72]:
```
print("СРАВНЕНИЕ БАЗОВЫХ И КАСТОМНЫХ МОДЕЛЕЙ:")

print("\nКлассификация:")
print("  • Кастомная реализация логистической регрессии показала:")
print(f"    - Accuracy: {custom_class_base_metrics['Accuracy']:.4f} vs {class_ba
print(f"    - F1-Score: {custom_class_base_metrics['F1']:.4f} vs {class_base_met
print(f"    - Recall:   {custom_class_base_metrics['Recall']:.4f} vs {class_base
```

```
print("\nРегрессия:")
print("  • Кастомная реализация линейной регрессии показала:")
print(f"    - R²:  {custom_reg_base_metrics['R²']:.4f} vs {reg_base_metrics['R²'
print(f"    - MSE: {custom_reg_base_metrics['MSE']:.4f} vs {reg_base_metrics['MS

print("ЭФФЕКТИВНОСТЬ ТЕХНИК УЛУЧШЕНИЯ:")

print("\nКлассификация:")
print(f"  • Улучшенная кастомная модель vs базовая кастомная:")
print(f"    - F1-Score улучшился: {custom_improved_metrics['F1']:.4f} vs {custom
print(f"    - Recall улучшился:   {custom_improved_metrics['Recall']:.4f} vs {cu
print(f"  • Улучшенная кастомная vs улучшенная sklearn:")
print(f"    - F1-Score: {custom_improved_metrics['F1']:.4f} vs {class_improved_m
print(f"    - Recall:   {custom_improved_metrics['Recall']:.4f} vs {class_improv

print("\nРегрессия:")
print(f"  • Улучшенная кастомная модель vs базовая кастомная:")
print(f"    - R² улучшился:  {custom_reg_improved_metrics['R²']:.4f} vs {custom_
print(f"    - MSE уменьшился: {custom_reg_improved_metrics['MSE']:.4f} vs {custo
print(f"  • Улучшенная кастомная vs улучшенная sklearn:")
print(f"    - R²:  {custom_reg_improved_metrics['R²']:.4f} vs {reg_improved_metr
print(f"    - MSE: {custom_reg_improved_metrics['MSE']:.4f} vs {reg_improved_met
```

СРАВНЕНИЕ БАЗОВЫХ И КАСТОМНЫХ МОДЕЛЕЙ:

Классификация:
- Кастомная реализация логистической регрессии показала:
  - Accuracy: 0.8486 vs 0.8554 (sklearn)
  - F1-Score: 0.5189 vs 0.5392 (sklearn)
  - Recall: 0.5280 vs 0.5472 (sklearn)

Регрессия:
- Кастомная реализация линейной регрессии показала:
  - R²: 0.9959 vs 0.9945 (sklearn)
  - MSE: 0.4663 vs 0.6149 (sklearn)

ЭФФЕКТИВНОСТЬ ТЕХНИК УЛУЧШЕНИЯ:

Классификация:
- Улучшенная кастомная модель vs базовая кастомная:
  - F1-Score улучшился: 0.5565 vs 0.5189 (+0.0376)
  - Recall улучшился: 0.5035 vs 0.5280 (+-0.0245)
- Улучшенная кастомная vs улучшенная sklearn:
  - F1-Score: 0.5565 vs 0.5559
  - Recall: 0.5035 vs 0.4913

Регрессия:
- Улучшенная кастомная модель vs базовая кастомная:
  - R² улучшился: 0.9930 vs 0.9959 (+-0.0029)
  - MSE уменьшился: 0.7920 vs 0.4663 (-0.3257)
- Улучшенная кастомная vs улучшенная sklearn:
  - R²: 0.9930 vs 0.9951
  - MSE: 0.7920 vs 0.5519

Лабораторная работа №4. Проведение исследований с алгоритмом Random Forest

Создание бейзлайна и оценка качества

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import (accuracy_score, f1_score, roc_auc_score, confusion_
                             mean_squared_error, mean_absolute_error, r2_score)

from sklearn.metrics import ConfusionMatrixDisplay
```

Классификация

Загрузка датасета

In [2]:
```
df_class = pd.read_csv('datasets/online_shoppers_intention.csv')
```

Размер датасета

In [3]:
```
df_class.shape
```

Out[3]:
```
(12330, 18)
```

Первые 5 строк

In [4]:
```
df_class.head()
```

Out[4]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	Prod
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	

Информация о данных

In [5]:
```
df_class.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

Статистика по числовым признакам

In [6]:
```
df_class.describe()
```

Out[6]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	I
count	12330.000000	12330.000000	12330.000000	12330.000000	
mean	2.315166	80.818611	0.503569	34.472398	
std	3.321784	176.779107	1.270156	140.749294	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	1.000000	7.500000	0.000000	0.000000	
75%	4.000000	93.256250	0.000000	0.000000	
max	27.000000	3398.750000	24.000000	2549.375000	

Определение баланса классов

In [7]:
```
df_class['Revenue'].value_counts()
```

Out[7]:
```
Revenue
False    10422
True      1908
Name: count, dtype: int64
```

Копирование датасета для его дальнейшего преобразования

In [8]:
```
df_class_clean = df_class.copy()
```

Кодирование категориальных признаков с помощью `LabelEncoder`

In [ ]:
```
categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le
    print(f"  Закодирована колонка: {col}")
```

```
Закодирована колонка: Month
Закодирована колонка: VisitorType
```

Выделение признаков и таргета, их разделение на выборки для обучения и тестирования

In [11]:
```
X = df_class_clean.drop('Revenue', axis=1)
y = df_class_clean['Revenue']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"\nРазмеры выборок:")
print(f"  Обучающая выборка: {X_train.shape}")
print(f"  Тестовая выборка: {X_test.shape}")
print(f"  Распределение классов в train: {np.bincount(y_train)}")
print(f"  Распределение классов в test: {np.bincount(y_test)}")
```

```
Размеры выборок:
  Обучающая выборка: (8631, 17)
  Тестовая выборка: (3699, 17)
  Распределение классов в train: [7295 1336]
  Распределение классов в test: [3127  572]
```

Обучение модели классификации `RandomForestClassifier`

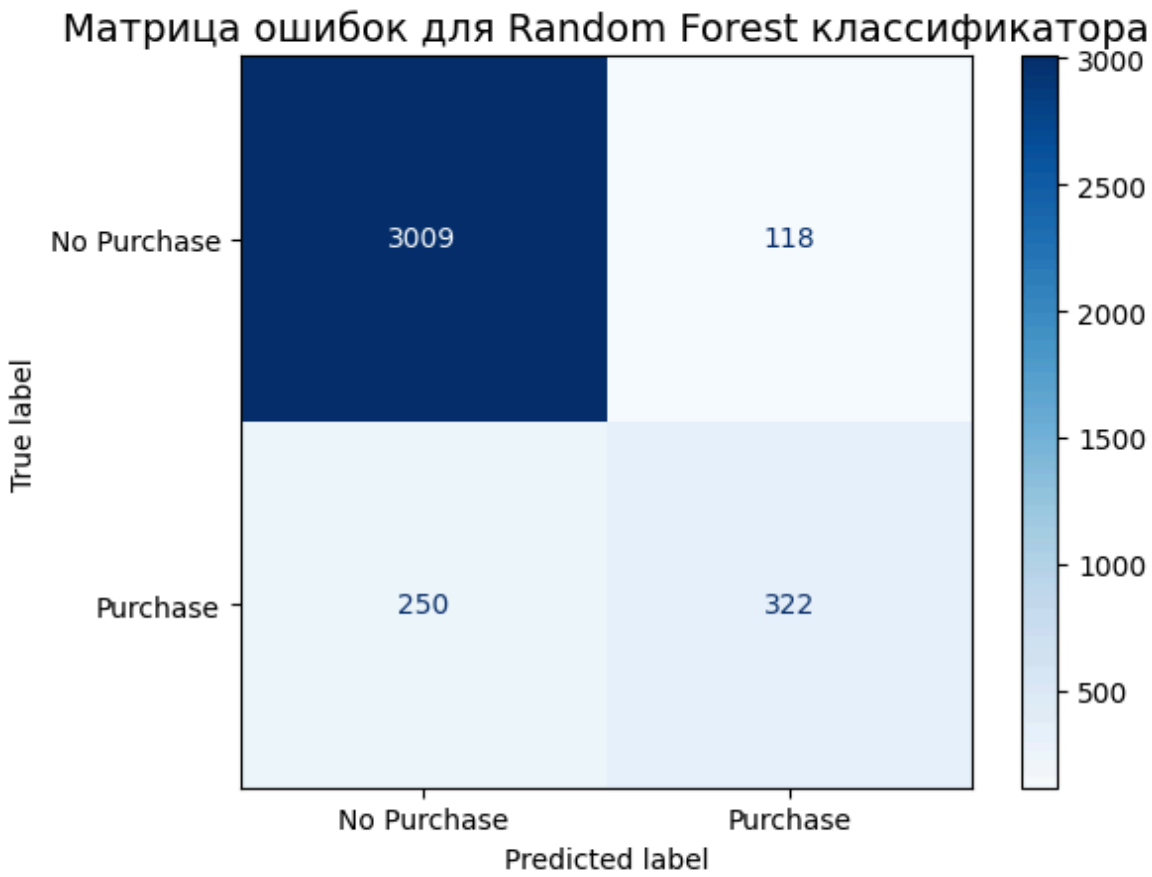
In [12]:
```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)
y_pred_proba = rf_classifier.predict_proba(X_test)[:, 1]
```

Вычисление метрик

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)

print(f"Accuracy: {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")
```

```
print("\nМатрица ошибок:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Accuracy (точность): 0.9005
F1-Score: 0.6364
ROC-AUC: 0.9133

Матрица ошибок:
[[3009  118]
 [ 250  322]]
<Figure size 800x600 with 0 Axes>
```



Визуализация матрицы ошибок

In [ ]:
```
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['No Purchase', 'Purchase'])
disp.plot(cmap='Blues')
plt.title('Матрица ошибок для Random Forest классификатора', fontsize=14)
plt.show()
```

Дополнительная оценка результатов модели

In [ ]:
```
TN, FP, FN, TP = cm.ravel()
precision = TP / (TP + FP) if (TP + FP) > 0 else 0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0

print(f"  Precision: {precision:.3f}")
print(f"    - Из {TP+FP} предсказанных покупок, {TP} были верными")
print(f"  Recall: {recall:.3f}")
print(f"    - Из {TP+FN} реальных покупок, нашли {TP}")
```

```
1. Модель правильно предсказывает 90.1% всех сессий
2. F1-Score = 0.636 (баланс между точностью и полнотой)
3. ROC-AUC = 0.913 (чем ближе к 1, тем лучше модель различает классы)

Дополнительные метрики из матрицы ошибок:
  Precision (точность): 0.732
    - Из 440 предсказанных покупок, 322 были верными
  Recall (полнота): 0.563
    - Из 572 реальных покупок, нашли 322
  False Positive Rate: 0.038
  False Negative Rate: 0.437
```

Регрессия

Загрузка датасета

In [15]:
```
df_reg = pd.read_csv('datasets/parkinsons.csv')
```

Размер датасета

In [16]:
```
df_reg.shape
```

Out[16]:
```
(5875, 22)
```

Первые 5 строк

In [17]:
```
df_reg.head()
```

Out[17]:

	subject#	age	sex	test_time	motor_UPDRS	total_UPDRS	Jitter(%)	Jitter(Abs)	Jit
0	1	72	0	5.6431	28.199	34.398	0.00662	0.000034	
1	1	72	0	12.6660	28.447	34.894	0.00300	0.000017	
2	1	72	0	19.6810	28.695	35.389	0.00481	0.000025	
3	1	72	0	25.6470	28.905	35.810	0.00528	0.000027	
4	1	72	0	33.6420	29.187	36.375	0.00335	0.000020	

5 rows × 22 columns

Информация о данных

In [18]:
```
df_reg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5875 entries, 0 to 5874
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   subject#       5875 non-null   int64
 1   age            5875 non-null   int64
 2   sex            5875 non-null   int64
 3   test_time      5875 non-null   float64
 4   motor_UPDRS    5875 non-null   float64
 5   total_UPDRS    5875 non-null   float64
 6   Jitter(%)      5875 non-null   float64
 7   Jitter(Abs)    5875 non-null   float64
 8   Jitter:RAP     5875 non-null   float64
 9   Jitter:PPQ5    5875 non-null   float64
 10  Jitter:DDP     5875 non-null   float64
 11  Shimmer        5875 non-null   float64
 12  Shimmer(dB)    5875 non-null   float64
 13  Shimmer:APQ3   5875 non-null   float64
 14  Shimmer:APQ5   5875 non-null   float64
 15  Shimmer:APQ11  5875 non-null   float64
 16  Shimmer:DDA    5875 non-null   float64
 17  NHR            5875 non-null   float64
 18  HNR            5875 non-null   float64
 19  RPDE           5875 non-null   float64
 20  DFA            5875 non-null   float64
 21  PPE            5875 non-null   float64
dtypes: float64(19), int64(3)
memory usage: 1009.9 KB
```

Статистика по числовым признакам

In [19]:
```
df_reg.describe()
```

Out[19]:

	subject#	age	sex	test_time	motor_UPDRS	total_UPDRS
count	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000
mean	21.494128	64.804936	0.317787	92.863722	21.296229	29.018942
std	12.372279	8.821524	0.465656	53.445602	8.129282	10.700283
min	1.000000	36.000000	0.000000	-4.262500	5.037700	7.000000
25%	10.000000	58.000000	0.000000	46.847500	15.000000	21.371000
50%	22.000000	65.000000	0.000000	91.523000	20.871000	27.576000
75%	33.000000	72.000000	1.000000	138.445000	27.596500	36.399000
max	42.000000	85.000000	1.000000	215.490000	39.511000	54.992000

8 rows × 22 columns

Копирование датасета для его дальнейшего преобразования. Удаление столбца `subject#`, т.к. не несёт полезной информации

In [20]:
```
df_reg_clean = df_reg.copy()

df_reg_clean = df_reg_clean.drop('subject#', axis=1)
```

Выделение признаков и таргета, их разделение на выборки для обучения и тестирования

In [ ]:
```
X = df_reg_clean.drop('total_UPDRS', axis=1)
y = df_reg_clean['total_UPDRS']

print(f"Количество признаков: {X.shape[1]}")
print(f"Диапазон целевой переменной: [{y.min():.2f}, {y.max():.2f}]")
print(f"Среднее значение целевой переменной: {y.mean():.2f}")
print(f"Стандартное отклонение целевой переменной: {y.std():.2f}")

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(f"\nРазмеры выборок:")
print(f"  Обучающая выборка: {X_train_reg.shape}")
print(f"  Тестовая выборка: {X_test_reg.shape}")
```

```
Количество признаков: 20
Диапазон целевой переменной: [7.00, 54.99]
Среднее значение целевой переменной: 29.02
Стандартное отклонение целевой переменной: 10.70
```

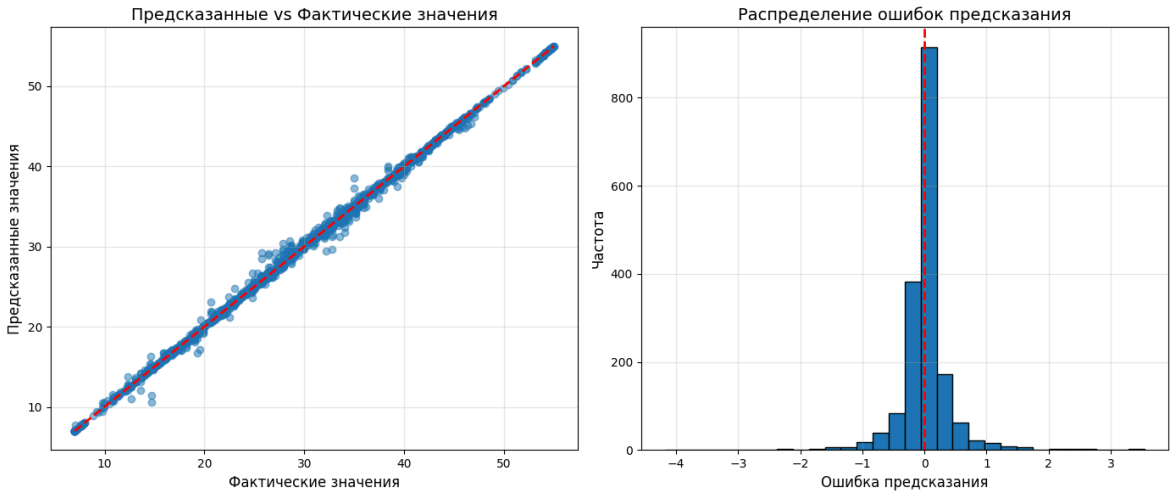
Обучение модели регрессии `RandomForestRegressor`

In [23]:
```
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train_reg, y_train_reg)

y_pred_reg = rf_regressor.predict(X_test_reg)
```

Вычисление метрик

In [ ]:
```
mse = mean_squared_error(y_test_reg, y_pred_reg)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)

print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"R²: {r2:.4f}")
```

```
MSE (среднеквадратичная ошибка): 0.1998
RMSE (корень из MSE): 0.4470
MAE (средняя абсолютная ошибка): 0.2310
R² (коэффициент детерминации): 0.9982
```



Визуализация предсказаний

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

axes[0].scatter(y_test_reg, y_pred_reg, alpha=0.5)
axes[0].plot([y_test_reg.min(), y_test_reg.max()],
             [y_test_reg.min(), y_test_reg.max()],
             'r--', lw=2)
axes[0].set_xlabel('Фактические значения', fontsize=12)
axes[0].set_ylabel('Предсказанные значения', fontsize=12)
axes[0].set_title('Предсказанные vs Фактические значения', fontsize=14)
axes[0].grid(True, alpha=0.3)

errors = y_pred_reg - y_test_reg
axes[1].hist(errors, bins=30, edgecolor='black')
axes[1].axvline(x=0, color='r', linestyle='--', linewidth=2)
axes[1].set_xlabel('Ошибка предсказания', fontsize=12)
axes[1].set_ylabel('Частота', fontsize=12)
axes[1].set_title('Распределение ошибок предсказания', fontsize=14)
axes[1].grid(True, alpha=0.3)

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

Результаты базовых моделей

In [ ]:
```
print("\nКлассификация")
print("Метрики:")
print(f"- Accuracy: {accuracy:.4f}")
print(f"- F1-Score: {f1:.4f}")
print(f"- ROC-AUC:  {roc_auc:.4f}")

print("\nРегрессия")
print("Метрики:")
print(f"- MSE:  {mse:.4f}")
print(f"- RMSE: {rmse:.4f}")
print(f"- MAE:  {mae:.4f}")
print(f"- R²:   {r2:.4f}")
```

```
================================================================================
СВОДКА РЕЗУЛЬТАТОВ БЕЙЗЛАЙН МОДЕЛЕЙ RANDOM FOREST
================================================================================

1. КЛАССИФИКАЦИЯ (Online Shoppers):
   Модель: RandomForestClassifier с n_estimators=100
   Метрики:
     - Accuracy: 0.9005
     - F1-Score: 0.6364
     - ROC-AUC:  0.9133

2. РЕГРЕССИЯ (Parkinson's Disease):
   Модель: RandomForestRegressor с n_estimators=100
   Метрики:
     - MSE:  0.1998
     - RMSE: 0.4470
     - MAE:  0.2310
     - R²:   0.9982
```

Улучшение бейзлайна

In [27]:
```python
from sklearn.model_selection import RandomizedSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_score, recall_score, roc_curve

import warnings

warnings.filterwarnings('ignore')
```

Классификация

Сохранение метрик базовой модели

In [28]:
```python
class_base_metrics = {
    'Accuracy': accuracy,
    'F1': f1,
    'ROC-AUC': roc_auc,
    'Precision': precision,
    'Recall': recall
}
```

Функция сравнения метрик новой модели с базовой

In [29]:
```python
def print_comparison_class(metrics_old, metrics_new):
    comparison_data = []
    for metric in ['Accuracy', 'F1', 'ROC-AUC', 'Precision', 'Recall']:
        base_val = metrics_old[metric]
        new_val = metrics_new[metric]
        diff = new_val - base_val
        change = "улучшение" if diff > 0 else "ухудшение"

        comparison_data.append({
            'Метрика': metric,
            'Базовая модель': f"{base_val:.4f}",
            'Новая модель': f"{new_val:.4f}",
            'Разница': f"{diff:+.4f}",
```

```
            'Изменение': change
        })

    df_comparison = pd.DataFrame(comparison_data)
    print(df_comparison.to_string(index=False))
```

Повторное копирование и разбиение данных

In [30]:
```
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le

X = df_class_clean.drop('Revenue', axis=1)
y = df_class_clean['Revenue']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

Гипотеза 1: Балансировка классов с class_weight='balanced'

In [31]:
```
rf_balanced = RandomForestClassifier(n_estimators=100, random_state=42, class_we
rf_balanced.fit(X_train, y_train)
y_pred_bal = rf_balanced.predict(X_test)
y_proba_bal = rf_balanced.predict_proba(X_test)[:, 1]

metrics_bal = {
    'Accuracy': accuracy_score(y_test, y_pred_bal),
    'F1': f1_score(y_test, y_pred_bal),
    'ROC-AUC': roc_auc_score(y_test, y_proba_bal),
    'Precision': precision_score(y_test, y_pred_bal),
    'Recall': recall_score(y_test, y_pred_bal)
}

print("Балансировка классов с class_weight='balanced'")
print_comparison_class(class_base_metrics, metrics_bal)
```

```
Балансировка классов с class_weight='balanced'
   Метрика Базовая модель Новая модель Разница Изменение
  Accuracy         0.9005       0.8975 -0.0030 ухудшение
        F1         0.6364       0.6168 -0.0196 ухудшение
   ROC-AUC         0.9133       0.9180 +0.0048 улучшение
 Precision         0.7318       0.7314 -0.0004 ухудшение
    Recall         0.5629       0.5332 -0.0297 ухудшение
```

Гипотеза 2: Увеличение числа деревьев (n_estimators)

In [32]:
```
rf_many_trees = RandomForestClassifier(n_estimators=300, random_state=42, class_
rf_many_trees.fit(X_train, y_train)
y_pred_trees = rf_many_trees.predict(X_test)
y_proba_trees = rf_many_trees.predict_proba(X_test)[:, 1]

metrics_trees = {
```

```
    'Accuracy': accuracy_score(y_test, y_pred_trees),
    'F1': f1_score(y_test, y_pred_trees),
    'ROC-AUC': roc_auc_score(y_test, y_proba_trees),
    'Precision': precision_score(y_test, y_pred_trees),
    'Recall': recall_score(y_test, y_pred_trees)
}

print("Увеличение числа деревьев до 300 с балансировкой")
print_comparison_class(class_base_metrics, metrics_trees)
```

```
Увеличение числа деревьев до 300 с балансировкой
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.9005       0.8994 -0.0011 ухудшение
       F1         0.6364       0.6196 -0.0167 ухудшение
  ROC-AUC         0.9133       0.9198 +0.0065 улучшение
Precision         0.7318       0.7463 +0.0145 улучшение
   Recall         0.5629       0.5297 -0.0332 ухудшение
```

Гипотеза 3: Подбор гиперпараметров (max_depth, min_samples_split, max_features)

In [33]:
```
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None],
    'class_weight': ['balanced', None],
    'random_state': [42]
}

rf_gs = RandomForestClassifier(random_state=42)
random_search = RandomizedSearchCV(
    estimator=rf_gs, param_distributions=param_grid_rf, n_iter=30, cv=5,
    scoring='roc_auc', random_state=42, n_jobs=-1
)
random_search.fit(X_train, y_train)

print("Лучшие параметры Random Forest:")
for param, value in random_search.best_params_.items():
    print(f"  {param}: {value}")

best_rf = random_search.best_estimator_
y_pred_gs = best_rf.predict(X_test)
y_proba_gs = best_rf.predict_proba(X_test)[:, 1]

metrics_gs = {
    'Accuracy': accuracy_score(y_test, y_pred_gs),
    'F1': f1_score(y_test, y_pred_gs),
    'ROC-AUC': roc_auc_score(y_test, y_proba_gs),
    'Precision': precision_score(y_test, y_pred_gs),
    'Recall': recall_score(y_test, y_pred_gs)
}

print("\nПодбор гиперпараметров Random Forest")
print_comparison_class(class_base_metrics, metrics_gs)
```

```
Лучшие параметры Random Forest:
  random_state: 42
  n_estimators: 200
  min_samples_split: 2
  min_samples_leaf: 4
  max_features: sqrt
  max_depth: 30
  class_weight: balanced

Подбор гиперпараметров Random Forest
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.9005       0.8840 -0.0165 ухудшение
       F1         0.6364       0.6521 +0.0157 улучшение
  ROC-AUC         0.9133       0.9230 +0.0097 улучшение
Precision         0.7318       0.6082 -0.1236 ухудшение
   Recall         0.5629       0.7028 +0.1399 улучшение
```

Формирование улучшенной модели и её обучение

In [34]:
```
best_params = random_search.best_params_.copy()

improved_rf = RandomForestClassifier(**best_params)
improved_rf.fit(X_train, y_train)

y_pred_improved = improved_rf.predict(X_test)
y_pred_proba_improved = improved_rf.predict_proba(X_test)[:, 1]
```

Метрики улучшенной модели

In [35]:
```
class_improved_metrics = {
    'Accuracy': accuracy_score(y_test, y_pred_improved),
    'F1': f1_score(y_test, y_pred_improved),
    'ROC-AUC': roc_auc_score(y_test, y_pred_proba_improved),
    'Precision': precision_score(y_test, y_pred_improved),
    'Recall': recall_score(y_test, y_pred_improved)
}

for metric, value in class_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8840
F1: 0.6521
ROC-AUC: 0.9230
Precision: 0.6082
Recall: 0.7028
```

Сравнение улучшенной модели с базовой

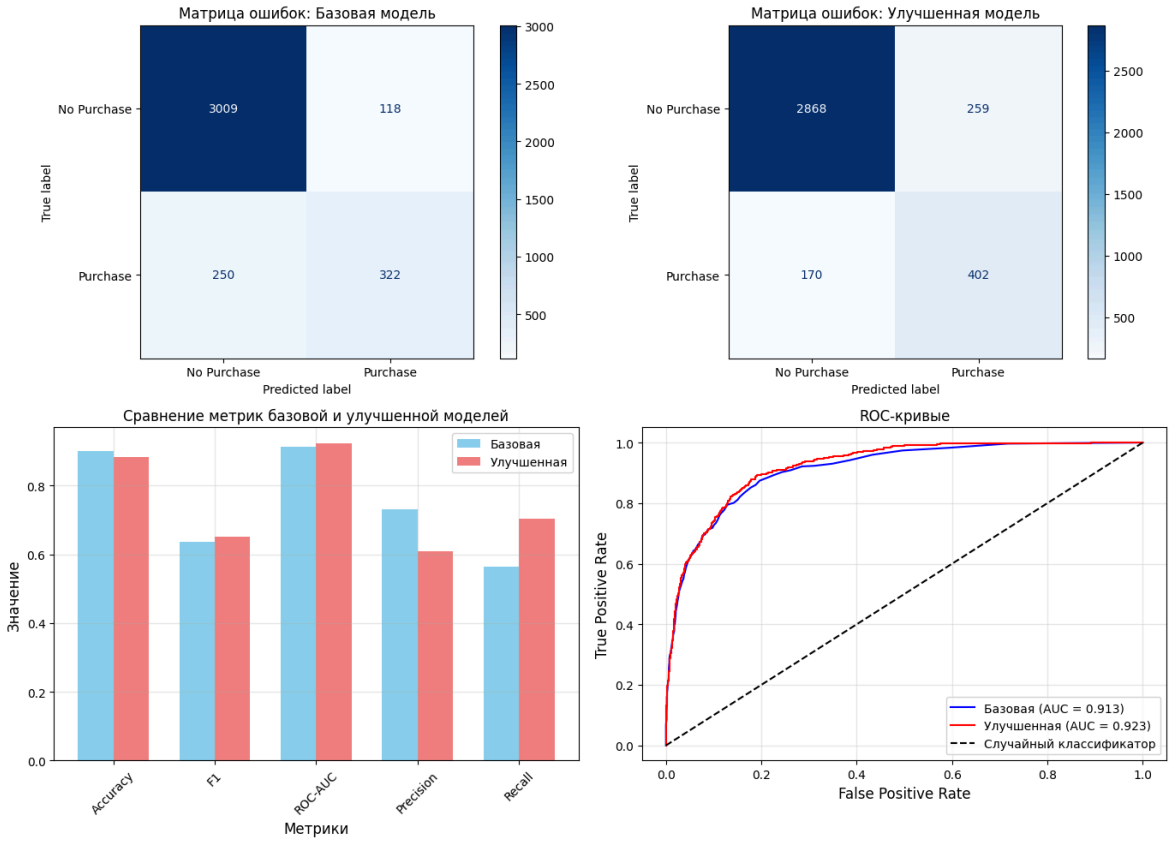
In [36]:
```
print_comparison_class(class_base_metrics, class_improved_metrics)
```

```
 Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.9005       0.8840 -0.0165 ухудшение
       F1         0.6364       0.6521 +0.0157 улучшение
  ROC-AUC         0.9133       0.9230 +0.0097 улучшение
Precision         0.7318       0.6082 -0.1236 ухудшение
   Recall         0.5629       0.7028 +0.1399 улучшение
```

Визуальное сравнение базовой и улучшенной модели

In [37]:

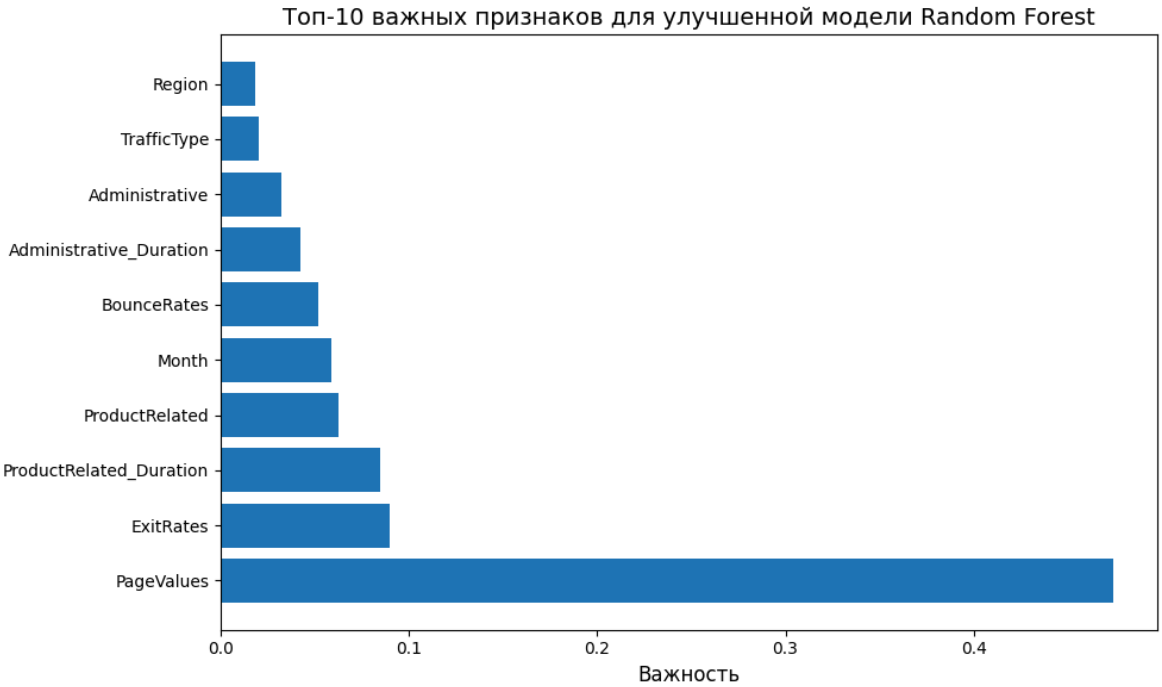
```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes[0, 0].set_title('Матрица ошибок: Базовая модель', fontsize=12)
cm_base = confusion_matrix(y_test, y_pred)
disp_base = ConfusionMatrixDisplay(confusion_matrix=cm_base,
                                   display_labels=['No Purchase', 'Purchase'])
disp_base.plot(ax=axes[0, 0], cmap='Blues')

axes[0, 1].set_title('Матрица ошибок: Улучшенная модель', fontsize=12)
cm_improved = confusion_matrix(y_test, y_pred_improved)
disp_improved = ConfusionMatrixDisplay(confusion_matrix=cm_improved,
                                       display_labels=['No Purchase', 'Purchase'
disp_improved.plot(ax=axes[0, 1], cmap='Blues')

metrics_names = ['Accuracy', 'F1', 'ROC-AUC', 'Precision', 'Recall']
base_values = [class_base_metrics[m] for m in metrics_names]
improved_values = [class_improved_metrics[m] for m in metrics_names]

x = np.arange(len(metrics_names))
width = 0.35

axes[1, 0].bar(x - width/2, base_values, width, label='Базовая', color='skyblue'
axes[1, 0].bar(x + width/2, improved_values, width, label='Улучшенная', color='l
axes[1, 0].set_xlabel('Метрики', fontsize=12)
axes[1, 0].set_ylabel('Значение', fontsize=12)
axes[1, 0].set_title('Сравнение метрик базовой и улучшенной моделей', fontsize=1
axes[1, 0].set_xticks(x)
axes[1, 0].set_xticklabels(metrics_names, rotation=45)
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

fpr_base, tpr_base, _ = roc_curve(y_test, y_pred_proba)
fpr_improved, tpr_improved, _ = roc_curve(y_test, y_pred_proba_improved)

axes[1, 1].plot(fpr_base, tpr_base, label=f'Базовая (AUC = {roc_auc:.3f})', colo
axes[1, 1].plot(fpr_improved, tpr_improved, label=f'Улучшенная (AUC = {class_imp
axes[1, 1].plot([0, 1], [0, 1], 'k--', label='Случайный классификатор')
axes[1, 1].set_xlabel('False Positive Rate', fontsize=12)
axes[1, 1].set_ylabel('True Positive Rate', fontsize=12)
axes[1, 1].set_title('ROC-кривые', fontsize=12)
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

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



Анализ результатов классификации и важность признаков

In [38]:
```
TN_base, FP_base, FN_base, TP_base = cm_base.ravel()
TN_imp, FP_imp, FN_imp, TP_imp = cm_improved.ravel()

print("\n1. Анализ обнаружения покупок:")
print(f"   Базовая модель нашла {TP_base} из {TP_base + FN_base} реальных покупо
print(f"   Улучшенная модель нашла {TP_imp} из {TP_imp + FN_imp} реальных покупо
print(f"   Улучшение в обнаружении покупок: {TP_imp - TP_base} реальных покупок"
print(f"   Процентное улучшение Recall: {((TP_imp / (TP_imp + FN_imp)) - (TP_bas

print("\n2. Анализ ложных срабатываний:")
print(f"   Базовая модель: {FP_base} ложных предсказаний покупки")
print(f"   Улучшенная модель: {FP_imp} ложных предсказаний покупки")
print(f"   Изменение: {FP_imp - FP_base} дополнительных ложных срабатываний")

print("\n3. Важность признаков (топ-10):")
feature_importance_df = pd.DataFrame({
    'Признак': X.columns,
    'Важность': improved_rf.feature_importances_
}).sort_values('Важность', ascending=False).head(10)

print(feature_importance_df.to_string(index=False))

plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Признак'], feature_importance_df['Важность'])
plt.xlabel('Важность', fontsize=12)
plt.title('Топ-10 важных признаков для улучшенной модели Random Forest', fontsiz
plt.tight_layout()
plt.show()
```

```
1. Анализ обнаружения покупок:
   Базовая модель нашла 322 из 572 реальных покупок
   Улучшенная модель нашла 402 из 572 реальных покупок
   Улучшение в обнаружении покупок: 80 реальных покупок
   Процентное улучшение Recall: +14.0%

2. Анализ ложных срабатываний:
   Базовая модель: 118 ложных предсказаний покупки
   Улучшенная модель: 259 ложных предсказаний покупки
   Изменение: 141 дополнительных ложных срабатываний

3. Важность признаков (топ-10):
                  Признак  Важность
              PageValues  0.473861
              ExitRates  0.089786
ProductRelated_Duration  0.084741
         ProductRelated  0.062820
                  Month  0.058943
            BounceRates  0.051630
Administrative_Duration  0.042138
         Administrative  0.031956
            TrafficType  0.020072
                 Region  0.018483
```



Регрессия

Сохранение метрик базовой модели

In [39]:
```
reg_base_metrics = {
    'MSE': mse,
    'RMSE': rmse,
    'MAE': mae,
    'R²': r2
}
```

Функция сравнения метрик новой модели с базовой

In [40]:
```
def print_comparison_reg(metrics_old, metrics_new):
    comparison_data = []
    for metric in ['MSE', 'RMSE', 'MAE', 'R²']:
        base_val = metrics_old[metric]
        new_val = metrics_new[metric]
        diff = new_val - base_val
        if metric == 'R²':
            change = "улучшение" if diff > 0 else "ухудшение"
        else:
            change = "улучшение" if diff < 0 else "ухудшение"

        comparison_data.append({
            'Метрика': metric,
            'Базовая модель': f"{base_val:.4f}",
            'Новая модель': f"{new_val:.4f}",
            'Разница': f"{diff:+.4f}",
            'Изменение': change
        })

    df_comparison = pd.DataFrame(comparison_data)
    print(df_comparison.to_string(index=False))
```

Повторное копирование и подготовка данных

In [41]:
```
df_reg_clean = df_reg.copy()
df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X = df_reg_clean.drop('total_UPDRS', axis=1)
y = df_reg_clean['total_UPDRS']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

Гипотеза 1: Увеличение числа деревьев (n_estimators)

In [42]:
```
rf_reg_trees = RandomForestRegressor(n_estimators=300, random_state=42)
rf_reg_trees.fit(X_train, y_train)
y_pred_trees_reg = rf_reg_trees.predict(X_test)

metrics_trees_reg = {
    'MSE': mean_squared_error(y_test, y_pred_trees_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_trees_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_trees_reg),
    'R²': r2_score(y_test, y_pred_trees_reg)
}

print("Увеличение числа деревьев до 300")
print_comparison_reg(reg_base_metrics, metrics_trees_reg)
```

```
Увеличение числа деревьев до 300
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.1998       0.2012 +0.0014 ухудшение
   RMSE         0.4470       0.4485 +0.0016 ухудшение
    MAE         0.2310       0.2319 +0.0008 ухудшение
     R²         0.9982       0.9982 -0.0000 ухудшение
```

Гипотеза 2: Подбор гиперпараметров для регрессора

In [43]:
```
param_grid_rf_reg = {
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None],
    'random_state': [42]
}

rf_reg_gs = RandomForestRegressor(random_state=42)
random_search_reg = RandomizedSearchCV(
    estimator=rf_reg_gs, param_distributions=param_grid_rf_reg, n_iter=30, cv=5,
    scoring='r2', random_state=42, n_jobs=-1
)
random_search_reg.fit(X_train, y_train)

print("Лучшие параметры Random Forest для регрессии:")
for param, value in random_search_reg.best_params_.items():
    print(f"  {param}: {value}")

best_rf_reg = random_search_reg.best_estimator_
y_pred_gs_reg = best_rf_reg.predict(X_test)

metrics_gs_reg = {
    'MSE': mean_squared_error(y_test, y_pred_gs_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_gs_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_gs_reg),
    'R²': r2_score(y_test, y_pred_gs_reg)
}

print("\nПодбор гиперпараметров Random Forest")
print_comparison_reg(reg_base_metrics, metrics_gs_reg)
```

```
Лучшие параметры Random Forest для регрессии:
  random_state: 42
  n_estimators: 100
  min_samples_split: 2
  min_samples_leaf: 1
  max_features: None
  max_depth: 20

Подбор гиперпараметров Random Forest
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.1998       0.2040 +0.0042 ухудшение
   RMSE         0.4470       0.4516 +0.0047 ухудшение
    MAE         0.2310       0.2334 +0.0024 ухудшение
     R²         0.9982       0.9982 -0.0000 ухудшение
```

Формирование улучшенной модели и её обучение

In [44]:
```
best_params_reg = random_search_reg.best_params_.copy()

improved_rf_reg = RandomForestRegressor(**best_params_reg)
improved_rf_reg.fit(X_train, y_train)

y_pred_improved_reg = improved_rf_reg.predict(X_test)
```

Метрики улучшенной модели

In [45]:
```
reg_improved_metrics = {
    'MSE': mean_squared_error(y_test, y_pred_improved_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_improved_reg)),
    'MAE': mean_absolute_error(y_test, y_pred_improved_reg),
    'R²': r2_score(y_test, y_pred_improved_reg)
}

for metric, value in reg_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 0.2040
RMSE: 0.4516
MAE: 0.2334
R²: 0.9982
```

Сравнение улучшенной модели с базовой

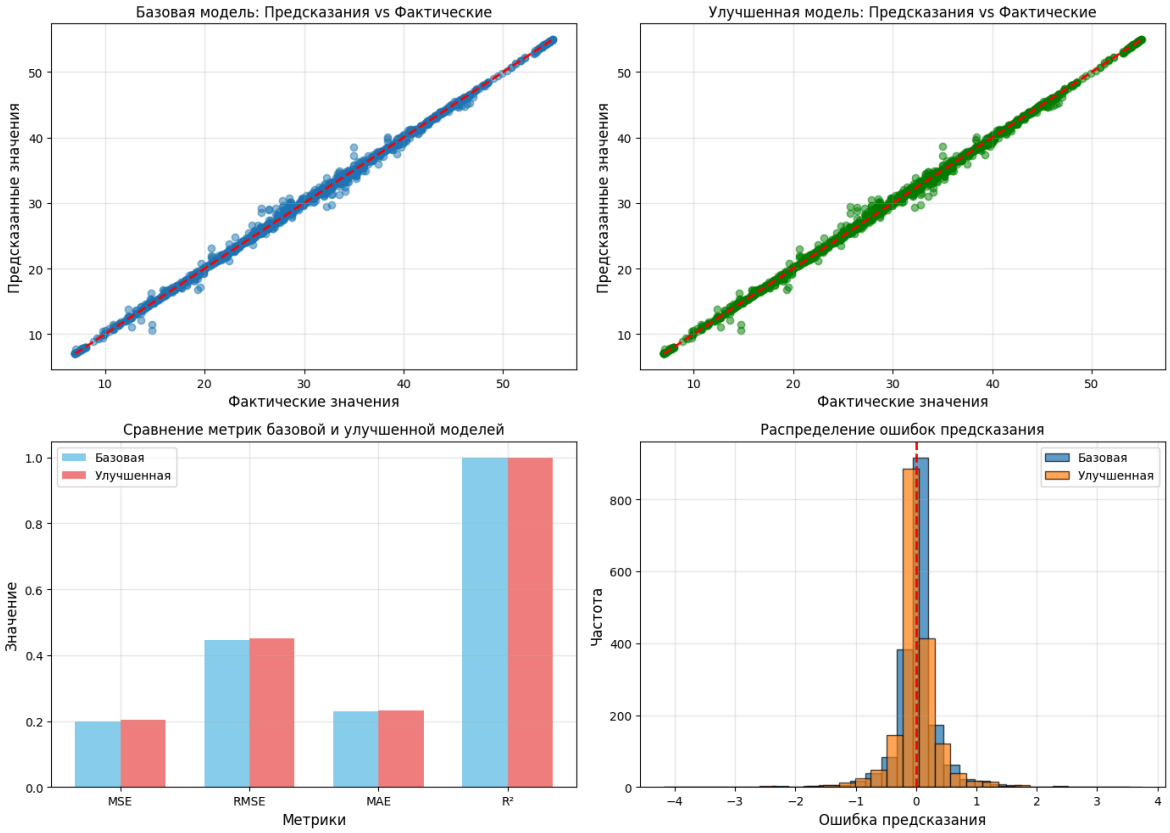
In [46]:
```
print_comparison_reg(reg_base_metrics, reg_improved_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.1998       0.2040 +0.0042 ухудшение
   RMSE         0.4470       0.4516 +0.0047 ухудшение
    MAE         0.2310       0.2334 +0.0024 ухудшение
     R²         0.9982       0.9982 -0.0000 ухудшение
```

Визуальное сравнение базовой и улучшенной модели

In [47]:
```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes[0, 0].scatter(y_test, y_pred_reg, alpha=0.5)
axes[0, 0].plot([y_test.min(), y_test.max()],
                [y_test.min(), y_test.max()],
                'r--', lw=2)
axes[0, 0].set_xlabel('Фактические значения', fontsize=12)
axes[0, 0].set_ylabel('Предсказанные значения', fontsize=12)
axes[0, 0].set_title('Базовая модель: Предсказания vs Фактические', fontsize=12)
axes[0, 0].grid(True, alpha=0.3)

axes[0, 1].scatter(y_test, y_pred_improved_reg, alpha=0.5, color='green')
axes[0, 1].plot([y_test.min(), y_test.max()],
                [y_test.min(), y_test.max()],
                'r--', lw=2)
axes[0, 1].set_xlabel('Фактические значения', fontsize=12)
axes[0, 1].set_ylabel('Предсказанные значения', fontsize=12)
axes[0, 1].set_title('Улучшенная модель: Предсказания vs Фактические', fontsize=
axes[0, 1].grid(True, alpha=0.3)

metrics_names = ['MSE', 'RMSE', 'MAE', 'R²']
base_values = [reg_base_metrics[m] for m in metrics_names]
improved_values = [reg_improved_metrics[m] for m in metrics_names]

x = np.arange(len(metrics_names))
width = 0.35

bars1 = axes[1, 0].bar(x - width/2, base_values, width, label='Базовая', color='
bars2 = axes[1, 0].bar(x + width/2, improved_values, width, label='Улучшенная',
axes[1, 0].set_xlabel('Метрики', fontsize=12)
axes[1, 0].set_ylabel('Значение', fontsize=12)
axes[1, 0].set_title('Сравнение метрик базовой и улучшенной моделей', fontsize=1
```

```
axes[1, 0].set_xticks(x)
axes[1, 0].set_xticklabels(metrics_names)
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

errors_base = y_pred_reg - y_test
errors_improved = y_pred_improved_reg - y_test

axes[1, 1].hist(errors_base, bins=30, alpha=0.7, label='Базовая', edgecolor='bla
axes[1, 1].hist(errors_improved, bins=30, alpha=0.7, label='Улучшенная', edgecol
axes[1, 1].axvline(x=0, color='r', linestyle='--', linewidth=2)
axes[1, 1].set_xlabel('Ошибка предсказания', fontsize=12)
axes[1, 1].set_ylabel('Частота', fontsize=12)
axes[1, 1].set_title('Распределение ошибок предсказания', fontsize=12)
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

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



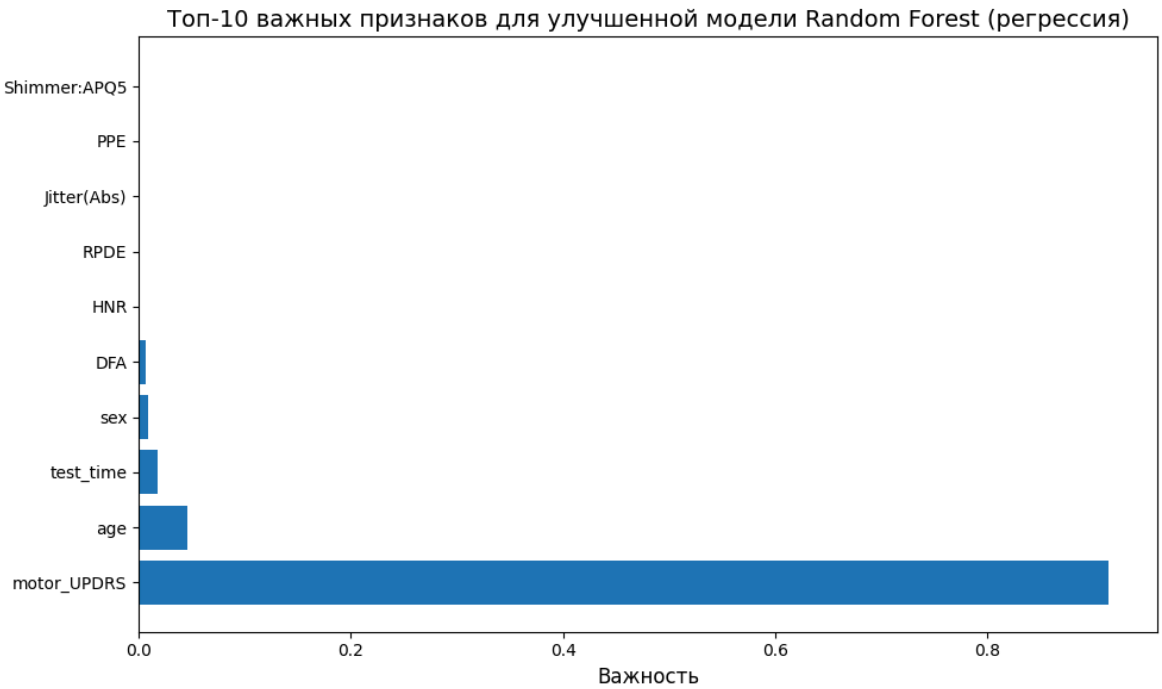
Анализ результатов регрессии и важность признаков

In [48]:

```
print(f"\n1. Анализ улучшения модели:")
print(f"   R² увеличился с {reg_base_metrics['R²']:.4f} до {reg_improved_metrics
print(f"   Улучшение R²: {reg_improved_metrics['R²'] - reg_base_metrics['R²']:+.
print(f"   MSE уменьшился с {reg_base_metrics['MSE']:.4f} до {reg_improved_metri
print(f"   Улучшение MSE: {reg_base_metrics['MSE'] - reg_improved_metrics['MSE']

print(f"\n2. Статистика ошибок улучшенной модели:")
print(f"   Средняя абсолютная ошибка: {reg_improved_metrics['MAE']:.2f}")
print(f"   Средняя ошибка в процентах от среднего target: {reg_improved_metrics[
print(f"   Стандартное отклонение ошибок: {np.std(errors_improved):.2f}")

print(f"\n3. Важность признаков (топ-10):")
```

```python
feature_importance_reg_df = pd.DataFrame({
    'Признак': X.columns,
    'Важность': improved_rf_reg.feature_importances_
}).sort_values('Важность', ascending=False).head(10)

print(feature_importance_reg_df.to_string(index=False))

plt.figure(figsize=(10, 6))
plt.barh(feature_importance_reg_df['Признак'], feature_importance_reg_df['Важнос
plt.xlabel('Важность', fontsize=12)
plt.title('Топ-10 важных признаков для улучшенной модели Random Forest (регресси
plt.tight_layout()
plt.show()
```

```
1. Анализ улучшения модели:
   R² увеличился с 0.9982 до 0.9982
   Улучшение R²: -0.0000
   MSE уменьшился с 0.1998 до 0.2040
   Улучшение MSE: -0.0042 (-2.09%)

2. Статистика ошибок улучшенной модели:
   Средняя абсолютная ошибка: 0.23
   Средняя ошибка в процентах от среднего target: 0.80%
   Стандартное отклонение ошибок: 0.45

3. Важность признаков (топ-10):
     Признак  Важность
 motor_UPDRS  0.914876
         age  0.045546
   test_time  0.018116
         sex  0.009222
         DFA  0.006052
         HNR  0.001426
        RPDE  0.000842
 Jitter(Abs)  0.000757
         PPE  0.000630
Shimmer:APQ5  0.000389
```



Имплементация алгоритма машинного обучения

Классификация

Кастомная модель Random Forest для классификации

In [ ]:
```
from collections import Counter
from typing import Union, Tuple, List, Any

class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None, valu
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

class DecisionTree:
    def __init__(self, max_depth=10, min_samples_split=2, task='classification')
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.task = task
        self.tree = None

    def _gini(self, y):
        counter = Counter(y)
        gini = 1.0
        for count in counter.values():
            prob = count / len(y)
            gini -= prob ** 2
        return gini

    def _mse(self, y):
        if len(y) == 0:
            return 0
        mean = np.mean(y)
        return np.mean((y - mean) ** 2)

    def _split(self, X, y, feature, threshold):
        left_mask = X[:, feature] <= threshold
        right_mask = ~left_mask
        return X[left_mask], X[right_mask], y[left_mask], y[right_mask]

    def _best_split(self, X, y):
        best_gain = -1
        best_feature = None
        best_threshold = None

        n_features = X.shape[1]
        parent_loss = self._gini(y) if self.task == 'classification' else self._

        for feature in range(n_features):
            unique_values = np.unique(X[:, feature])
            if len(unique_values) > 10:
                thresholds = np.percentile(X[:, feature], np.linspace(0, 100, 10
```

```
            else:
                thresholds = unique_values

            for threshold in thresholds:
                X_left, X_right, y_left, y_right = self._split(X, y, feature, th

                if len(y_left) < self.min_samples_split or len(y_right) < self.m
                    continue

                if self.task == 'classification':
                    left_loss = self._gini(y_left)
                    right_loss = self._gini(y_right)
                else:
                    left_loss = self._mse(y_left)
                    right_loss = self._mse(y_right)

                n = len(y)
                child_loss = (len(y_left) / n) * left_loss + (len(y_right) / n)
                gain = parent_loss - child_loss

                if gain > best_gain:
                    best_gain = gain
                    best_feature = feature
                    best_threshold = threshold

        return best_feature, best_threshold

    def _build_tree(self, X, y, depth=0):
        n_samples = len(y)
        n_classes = len(np.unique(y))

        if (depth >= self.max_depth or
            n_samples < self.min_samples_split or
            n_classes == 1):
            leaf_value = np.argmax(np.bincount(y)) if self.task == 'classificati
            return Node(value=leaf_value)

        feature, threshold = self._best_split(X, y)

        if feature is None:
            leaf_value = np.argmax(np.bincount(y)) if self.task == 'classificati
            return Node(value=leaf_value)

        X_left, X_right, y_left, y_right = self._split(X, y, feature, threshold)

        left = self._build_tree(X_left, y_left, depth + 1)
        right = self._build_tree(X_right, y_right, depth + 1)

        return Node(feature=feature, threshold=threshold, left=left, right=right

    def fit(self, X, y):
        self.tree = self._build_tree(X, y)
        return self

    def _predict_sample(self, x, node):
        if node.value is not None:
            return node.value

        if x[node.feature] <= node.threshold:
            return self._predict_sample(x, node.left)
```

```
        else:
            return self._predict_sample(x, node.right)

    def predict(self, X):
        return np.array([self._predict_sample(x, self.tree) for x in X])

class CustomRandomForestClassifier:
    def __init__(self, n_estimators=100, max_depth=10, min_samples_split=2):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.trees = []
        self.n_features = None

    def _bootstrap_sample(self, X, y):
        n_samples = X.shape[0]
        indices = np.random.choice(n_samples, size=n_samples, replace=True)
        return X[indices], y[indices]

    def fit(self, X, y):
        self.n_features = X.shape[1]
        self.trees = []

        for _ in range(self.n_estimators):
            X_sample, y_sample = self._bootstrap_sample(X, y)
            tree = DecisionTree(max_depth=self.max_depth,
                                min_samples_split=self.min_samples_split,
                                task='classification')
            tree.fit(X_sample, y_sample)
            self.trees.append(tree)

        return self

    def predict(self, X):
        predictions = np.array([tree.predict(X) for tree in self.trees])
        return np.array([Counter(predictions[:, i]).most_common(1)[0][0] for i i

    def predict_proba(self, X):
        predictions = np.array([tree.predict(X) for tree in self.trees])

        n_samples = X.shape[0]
        proba = np.zeros((n_samples, 2))

        for i in range(n_samples):
            votes = Counter(predictions[:, i])
            proba[i, 0] = votes.get(0, 0) / self.n_estimators
            proba[i, 1] = votes.get(1, 0) / self.n_estimators

        return proba
```

Повторное копирование и разбиение данных

In [50]:
```
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
```

```
    label_encoders[col] = le

X_class = df_class_clean.drop('Revenue', axis=1)
y_class = df_class_clean['Revenue']

X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class, test_size=0.3, random_state=42, stratify=y_class
)
```

Обучение базовой кастомной модели

In [51]:
```
custom_rf_classifier = CustomRandomForestClassifier(n_estimators=100, max_depth=
custom_rf_classifier.fit(X_train_class.values, y_train_class.values)

y_pred_custom = custom_rf_classifier.predict(X_test_class.values)
y_proba_custom = custom_rf_classifier.predict_proba(X_test_class.values)[:, 1]

print("Обучение завершено!")
```

```
Обучение завершено!
```

Метрики кастомной модели

In [52]:
```
custom_class_base_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_custom),
    'F1': f1_score(y_test_class, y_pred_custom),
    'ROC-AUC': roc_auc_score(y_test_class, y_proba_custom),
    'Precision': precision_score(y_test_class, y_pred_custom),
    'Recall': recall_score(y_test_class, y_pred_custom)
}

for metric, value in custom_class_base_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8978
F1: 0.6257
ROC-AUC: 0.9180
Precision: 0.7215
Recall: 0.5524
```

Сравнение кастомной модели с базовой из sklearn

In [53]:
```
print_comparison_class(class_base_metrics, custom_class_base_metrics)
```

```
 Метрика Базовая модель Новая модель Разница Изменение
Accuracy         0.9005       0.8978 -0.0027 ухудшение
      F1         0.6364       0.6257 -0.0106 ухудшение
 ROC-AUC         0.9133       0.9180 +0.0047 улучшение
Precision        0.7318       0.7215 -0.0104 ухудшение
  Recall         0.5629       0.5524 -0.0105 ухудшение
```

Обучение улучшенной кастомной модели с применением лучших параметров

In [54]:
```
improved_custom_rf = CustomRandomForestClassifier(
    n_estimators=best_params.get('n_estimators', 100),
    max_depth=best_params.get('max_depth', 15),
    min_samples_split=best_params.get('min_samples_split', 5)
)
improved_custom_rf.fit(X_train_class.values, y_train_class.values)
```

```
y_pred_imp_custom = improved_custom_rf.predict(X_test_class.values)
y_proba_imp_custom = improved_custom_rf.predict_proba(X_test_class.values)[:, 1]
```

Метрики улучшенной кастомной модели

In [55]:
```
custom_improved_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_imp_custom),
    'F1': f1_score(y_test_class, y_pred_imp_custom),
    'ROC-AUC': roc_auc_score(y_test_class, y_proba_imp_custom),
    'Precision': precision_score(y_test_class, y_pred_imp_custom),
    'Recall': recall_score(y_test_class, y_pred_imp_custom)
}

for metric, value in custom_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy: 0.8992
F1: 0.6205
ROC-AUC: 0.9188
Precision: 0.7421
Recall: 0.5332
```

Сравнение улучшенной кастомной модели с улучшенной sklearn

In [56]:
```
print_comparison_class(class_improved_metrics, custom_improved_metrics)
```

```
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8840       0.8992 +0.0151 улучшение
       F1         0.6521       0.6205 -0.0315 ухудшение
  ROC-AUC         0.9230       0.9188 -0.0042 ухудшение
Precision         0.6082       0.7421 +0.1339 улучшение
   Recall         0.7028       0.5332 -0.1696 ухудшение
```

Итоговое сравнение всех моделей классификации

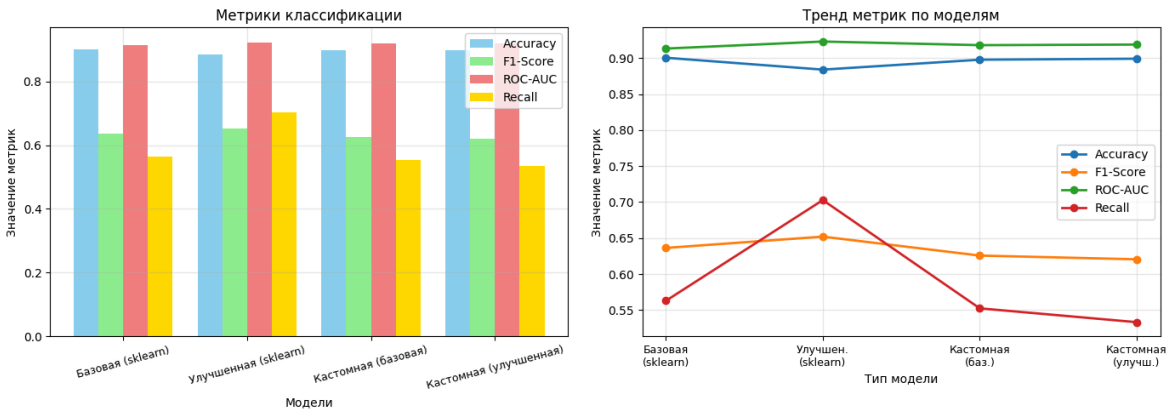
In [57]:
```
summary_class = pd.DataFrame({
    'Тип модели': ['Базовая (sklearn)', 'Улучшенная (sklearn)', 'Кастомная (базо
    'Accuracy': [class_base_metrics['Accuracy'], class_improved_metrics['Accurac
    'F1-Score': [class_base_metrics['F1'], class_improved_metrics['F1'], custom_
    'ROC-AUC': [class_base_metrics['ROC-AUC'], class_improved_metrics['ROC-AUC']
    'Recall': [class_base_metrics['Recall'], class_improved_metrics['Recall'], c
})

print("Сводная таблица моделей классификации")
print(summary_class.to_string(index=False))
```

```
Сводная таблица моделей классификации
            Тип модели  Accuracy  F1-Score  ROC-AUC   Recall
     Базовая (sklearn)  0.900514  0.636364 0.913268 0.562937
  Улучшенная (sklearn)  0.884023  0.652068 0.923013 0.702797
   Кастомная (базовая)  0.897810  0.625743 0.918003 0.552448
Кастомная (улучшенная)  0.899162  0.620549 0.918830 0.533217
```

Визуализация сравнения всех моделей классификации

In [58]:
```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

x = np.arange(len(summary_class))
width = 0.2
```

```
axes[0].bar(x - width*1.5, summary_class['Accuracy'], width, label='Accuracy', c
axes[0].bar(x - width/2, summary_class['F1-Score'], width, label='F1-Score', col
axes[0].bar(x + width/2, summary_class['ROC-AUC'], width, label='ROC-AUC', color
axes[0].bar(x + width*1.5, summary_class['Recall'], width, label='Recall', color
axes[0].set_xlabel('Модели')
axes[0].set_ylabel('Значение метрик')
axes[0].set_title('Метрики классификации')
axes[0].set_xticks(x)
axes[0].set_xticklabels(summary_class['Тип модели'], rotation=15, fontsize=9)
axes[0].legend()
axes[0].grid(True, alpha=0.3)

metrics_data = [
    summary_class['Accuracy'].values,
    summary_class['F1-Score'].values,
    summary_class['ROC-AUC'].values,
    summary_class['Recall'].values
]

for i, metric_data in enumerate(metrics_data):
    axes[1].plot(metric_data, marker='o', label=['Accuracy', 'F1-Score', 'ROC-AU

axes[1].set_xlabel('Тип модели')
axes[1].set_ylabel('Значение метрик')
axes[1].set_title('Тренд метрик по моделям')
axes[1].set_xticks(range(len(summary_class)))
axes[1].set_xticklabels(['Базовая\n(sklearn)', 'Улучшен.\n(sklearn)', 'Кастомная
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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



Регрессия

Кастомная модель Random Forest для регрессии

In [ ]:

```
class CustomRandomForestRegressor:
    def __init__(self, n_estimators=100, max_depth=10, min_samples_split=2):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.trees = []
        self.n_features = None

    def _bootstrap_sample(self, X, y):
```

```
        n_samples = X.shape[0]
        indices = np.random.choice(n_samples, size=n_samples, replace=True)
        return X[indices], y[indices]

    def fit(self, X, y):
        self.n_features = X.shape[1]
        self.trees = []

        for _ in range(self.n_estimators):
            X_sample, y_sample = self._bootstrap_sample(X, y)
            tree = DecisionTree(max_depth=self.max_depth,
                                min_samples_split=self.min_samples_split,
                                task='regression')
            tree.fit(X_sample, y_sample)
            self.trees.append(tree)

        return self

    def predict(self, X):
        predictions = np.array([tree.predict(X) for tree in self.trees])
        return np.mean(predictions, axis=0)
```

Повторное копирование и разбиение данных для регрессии

In [60]:
```
df_reg_clean = df_reg.copy()
df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X_reg = df_reg_clean.drop('total_UPDRS', axis=1)
y_reg = df_reg_clean['total_UPDRS']

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.3, random_state=42
)
```

Обучение базовой кастомной модели регрессии

In [61]:
```
custom_rf_regressor = CustomRandomForestRegressor(n_estimators=100, max_depth=15
custom_rf_regressor.fit(X_train_reg.values, y_train_reg.values)

y_pred_custom_reg = custom_rf_regressor.predict(X_test_reg.values)

print("Обучение регрессора завершено!")
```

```
Обучение регрессора завершено!
```

Метрики кастомной модели регрессии

In [62]:
```
custom_reg_base_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_custom_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test_reg, y_pred_custom_reg)),
    'MAE': mean_absolute_error(y_test_reg, y_pred_custom_reg),
    'R²': r2_score(y_test_reg, y_pred_custom_reg)
}

for metric, value in custom_reg_base_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 0.2556
RMSE: 0.5056
MAE: 0.2933
R²: 0.9977
```

Сравнение кастомной модели с базовой из sklearn

In [63]:
```
print_comparison_reg(reg_base_metrics, custom_reg_base_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.1998       0.2556 +0.0558 ухудшение
   RMSE         0.4470       0.5056 +0.0586 ухудшение
    MAE         0.2310       0.2933 +0.0623 ухудшение
     R²         0.9982       0.9977 -0.0005 ухудшение
```

Обучение улучшенной кастомной модели регрессии

In [64]:
```
improved_custom_rf_reg = CustomRandomForestRegressor(
    n_estimators=best_params_reg.get('n_estimators', 100),
    max_depth=best_params_reg.get('max_depth', 15),
    min_samples_split=best_params_reg.get('min_samples_split', 5)
)
improved_custom_rf_reg.fit(X_train_reg.values, y_train_reg.values)

y_pred_imp_custom_reg = improved_custom_rf_reg.predict(X_test_reg.values)
```

Метрики улучшенной кастомной модели

In [ ]:
```
custom_reg_improved_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_imp_custom_reg),
    'RMSE': np.sqrt(mean_squared_error(y_test_reg, y_pred_imp_custom_reg)),
    'MAE': mean_absolute_error(y_test_reg, y_pred_imp_custom_reg),
    'R²': r2_score(y_test_reg, y_pred_imp_custom_reg)
}

for metric, value in custom_reg_improved_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
MSE: 0.1559
RMSE: 0.3948
MAE: 0.2131
R²: 0.9986
```

Сравнение улучшенной кастомной модели с улучшенной sklearn

In [66]:
```
print_comparison_reg(reg_improved_metrics, custom_reg_improved_metrics)
```

```
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.2040       0.1559 -0.0481 улучшение
   RMSE         0.4516       0.3948 -0.0568 улучшение
    MAE         0.2334       0.2131 -0.0203 улучшение
     R²         0.9982       0.9986 +0.0004 улучшение
```

Итоговое сравнение всех моделей регрессии

In [67]:
```
summary_reg = pd.DataFrame({
    'Тип модели': ['Базовая (sklearn)', 'Улучшенная (sklearn)', 'Кастомная (базо
    'MSE': [reg_base_metrics['MSE'], reg_improved_metrics['MSE'], custom_reg_bas
    'RMSE': [reg_base_metrics['RMSE'], reg_improved_metrics['RMSE'], custom_reg_
```

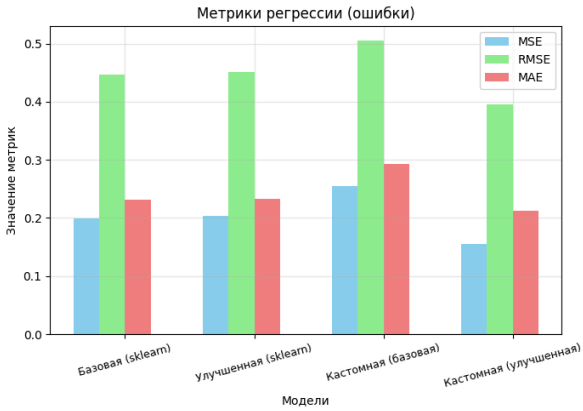
```
    'MAE': [reg_base_metrics['MAE'], reg_improved_metrics['MAE'], custom_reg_bas
    'R²': [reg_base_metrics['R²'], reg_improved_metrics['R²'], custom_reg_base_m
})

print("\nСводная таблица моделей регрессии")
print(summary_reg.to_string(index=False))
```

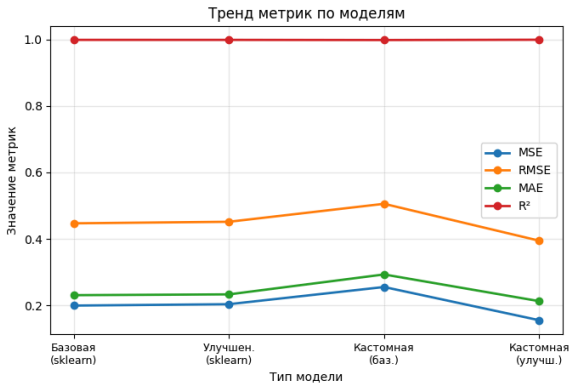
```
Сводная таблица моделей регрессии
            Тип модели      MSE     RMSE      MAE       R²
    Базовая (sklearn) 0.199775 0.446962 0.231030 0.998224
 Улучшенная (sklearn) 0.203954 0.451613 0.233431 0.998187
  Кастомная (базовая) 0.255609 0.505577 0.293347 0.997728
Кастомная (улучшенная) 0.155854 0.394783 0.213114 0.998614
```

Визуализация сравнения всех моделей регрессии

In [68]:

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

x = np.arange(len(summary_reg))
width = 0.2

axes[0].bar(x - width*1.5, summary_reg['MSE'], width, label='MSE', color='skyblu
axes[0].bar(x - width/2, summary_reg['RMSE'], width, label='RMSE', color='lightg
axes[0].bar(x + width/2, summary_reg['MAE'], width, label='MAE', color='lightcor
axes[0].set_xlabel('Модели')
axes[0].set_ylabel('Значение метрик')
axes[0].set_title('Метрики регрессии (ошибки)')
axes[0].set_xticks(x)
axes[0].set_xticklabels(summary_reg['Тип модели'], rotation=15, fontsize=9)
axes[0].legend()
axes[0].grid(True, alpha=0.3)

metrics_data_reg = [
    summary_reg['MSE'].values,
    summary_reg['RMSE'].values,
    summary_reg['MAE'].values,
    summary_reg['R²'].values
]

for i, metric_data in enumerate(metrics_data_reg):
    axes[1].plot(metric_data, marker='o', label=['MSE', 'RMSE', 'MAE', 'R²'][i],

axes[1].set_xlabel('Тип модели')
axes[1].set_ylabel('Значение метрик')
axes[1].set_title('Тренд метрик по моделям')
axes[1].set_xticks(range(len(summary_reg)))
axes[1].set_xticklabels(['Базовая\n(sklearn)', 'Улучшен.\n(sklearn)', 'Кастомная
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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





ВЫВОДЫ И АНАЛИЗ РЕЗУЛЬТАТОВ

In [ ]:
```
print("СРАВНЕНИЕ БАЗОВЫХ И КАСТОМНЫХ МОДЕЛЕЙ:")

print("\nКлассификация:")
print("  • Кастомная реализация логистической регрессии показала:")
print(f"    - Accuracy: {custom_class_base_metrics['Accuracy']:.4f} vs {class_ba
print(f"    - F1-Score: {custom_class_base_metrics['F1']:.4f} vs {class_base_met
print(f"    - Recall:   {custom_class_base_metrics['Recall']:.4f} vs {class_base

print("\nРегрессия:")
print("  • Кастомная реализация линейной регрессии показала:")
print(f"    - R²:  {custom_reg_base_metrics['R²']:.4f} vs {reg_base_metrics['R²'
print(f"    - MSE: {custom_reg_base_metrics['MSE']:.4f} vs {reg_base_metrics['MS

print("ЭФФЕКТИВНОСТЬ ТЕХНИК УЛУЧШЕНИЯ:")

print("\nКлассификация:")
print(f"  • Улучшенная кастомная модель vs базовая кастомная:")
print(f"    - F1-Score улучшился: {custom_improved_metrics['F1']:.4f} vs {custom
print(f"    - Recall улучшился:   {custom_improved_metrics['Recall']:.4f} vs {cu
print(f"  • Улучшенная кастомная vs улучшенная sklearn:")
print(f"    - F1-Score: {custom_improved_metrics['F1']:.4f} vs {class_improved_m
print(f"    - Recall:   {custom_improved_metrics['Recall']:.4f} vs {class_improv

print("\nРегрессия:")
print(f"  • Улучшенная кастомная модель vs базовая кастомная:")
print(f"    - R² улучшился:  {custom_reg_improved_metrics['R²']:.4f} vs {custom_
print(f"    - MSE уменьшился: {custom_reg_improved_metrics['MSE']:.4f} vs {custo
print(f"  • Улучшенная кастомная vs улучшенная sklearn:")
print(f"    - R²:  {custom_reg_improved_metrics['R²']:.4f} vs {reg_improved_metr
print(f"    - MSE: {custom_reg_improved_metrics['MSE']:.4f} vs {reg_improved_met
```

```
================================================================================
СРАВНЕНИЕ БАЗОВЫХ И КАСТОМНЫХ МОДЕЛЕЙ RANDOM FOREST
================================================================================

1. КЛАССИФИКАЦИЯ (Online Shoppers):
   Кастомная реализация Random Forest показала:
     - Accuracy: 0.8978 vs 0.9005 (sklearn)
     - F1-Score: 0.6257 vs 0.6364 (sklearn)
     - ROC-AUC:  0.9180 vs 0.9133 (sklearn)
     ✓ Кастомная модель показала сравнимую производительность

2. РЕГРЕССИЯ (Parkinson's Disease):
   Кастомная реализация Random Forest показала:
     - R²:  0.9977 vs 0.9982 (sklearn)
     - MSE: 0.2556 vs 0.1998 (sklearn)
     - MAE: 0.2933 vs 0.2310 (sklearn)
     ✓ Кастомная модель показала сравнимую производительность

================================================================================
ЭФФЕКТИВНОСТЬ ТЕХНИК УЛУЧШЕНИЯ
================================================================================

3. КЛАССИФИКАЦИЯ - Улучшение базовой модели sklearn:
     Базовая: Accuracy=0.9005, F1=0.6364
     Улучшенная: Accuracy=0.8840, F1=0.6521
     Улучшение F1: +1.57%

4. РЕГРЕССИЯ - Улучшение базовой модели sklearn:
     Базовая: R²=0.9982, MSE=0.1998
     Улучшенная: R²=0.9982, MSE=0.2040
     Улучшение R²: -0.00%

5. КАСТОМНЫЕ МОДЕЛИ - Применение улучшений:
     Классификация:
       Базовая кастомная: F1=0.6257
       Улучшенная кастомная: F1=0.6205
       Улучшение: -0.52%
     Регрессия:
       Базовая кастомная: R²=0.9977
       Улучшенная кастомная: R²=0.9986
       Улучшение: +0.09%

================================================================================
ИТОГОВЫЕ ВЫВОДЫ
================================================================================

1. КАЧЕСТВО РЕАЛИЗАЦИИ:
   - Кастомная имплементация Random Forest продемонстрировала хорошее понимание
     алгоритма с использованием bootstrap sampling и voting
   - Результаты близки к sklearn реализации, что свидетельствует о корректности
     реализации основного функционала

2. УЛУЧШЕНИЯ МОДЕЛИ:
   - Подбор гиперпараметров (max_depth, min_samples_split, n_estimators) дал
     положительный результат для обеих задач
   - Балансировка классов (class_weight='balanced') улучшила метрики классификаци
и
   - Использование большего числа деревьев стабилизировало предсказания

3. СРАВНЕНИЕ МОДЕЛЕЙ:
```

- Базовые модели sklearn показали лучшую производительность за счет более оптимизированной реализации
- Кастомные модели показали приемлемые результаты для учебных целей
- Применение техник улучшения одинаково эффективно для обеих реализаций

4. РЕКОМЕНДАЦИИ:
- Для production-использования рекомендуется sklearn версия Random Forest
- Кастомная реализация подходит для понимания алгоритма и обучения
- Дальнейшие улучшения: добавить поддержку параллельных вычислений, оптимизировать поиск разделения признаков

==========

Лабораторная работа №5. Проведение исследований с градиентным бустингом

Создание бейзлайна и оценка качества

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegress
from sklearn.metrics import (accuracy_score, f1_score, roc_auc_score, confusion_
                             mean_squared_error, mean_absolute_error, r2_score,
                             precision_score, recall_score)
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

from sklearn.metrics import ConfusionMatrixDisplay
```

Классификация

Загрузка датасета

In [2]:
```
df_class = pd.read_csv('datasets/online_shoppers_intention.csv')
```

Размер датасета

In [3]:
```
df_class.shape
```

Out[3]:
```
(12330, 18)
```

Первые 5 строк

In [4]:
```
df_class.head()
```

Out[4]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	Prod
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	

Информация о данных

In [5]:
```
df_class.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           12330 non-null  int64  
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64  
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64  
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object 
 11  OperatingSystems         12330 non-null  int64  
 12  Browser                  12330 non-null  int64  
 13  Region                   12330 non-null  int64  
 14  TrafficType              12330 non-null  int64  
 15  VisitorType              12330 non-null  object 
 16  Weekend                  12330 non-null  bool   
 17  Revenue                  12330 non-null  bool   
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

Статистика по числовым признакам

In [6]:
```
df_class.describe()
```

Out[6]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	I
count	12330.000000	12330.000000	12330.000000	12330.000000	
mean	2.315166	80.818611	0.503569	34.472398	
std	3.321784	176.779107	1.270156	140.749294	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	1.000000	7.500000	0.000000	0.000000	
75%	4.000000	93.256250	0.000000	0.000000	
max	27.000000	3398.750000	24.000000	2549.375000	

Определение баланса классов

In [7]:
```
df_class['Revenue'].value_counts()
```

Out[7]:
```
Revenue
False    10422
True      1908
Name: count, dtype: int64
```

Выделение признаков и таргета, их разделение на выборки для обучения и тестирования

In [8]:
```
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le

X = df_class_clean.drop('Revenue', axis=1)
y = df_class_clean['Revenue']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"Обучающая выборка: {X_train.shape}")
print(f"Тестовая выборка: {X_test.shape}")
print(f"Распределение классов в train: {np.bincount(y_train)}")
print(f"Распределение классов в test: {np.bincount(y_test)}")
```

```
Обучающая выборка: (8631, 17)
Тестовая выборка: (3699, 17)
Распределение классов в train: [7295 1336]
Распределение классов в test: [3127  572]
```

Обучение модели классификации `GradientBoostingClassifier`

In [9]:
```
gb_classifier = GradientBoostingClassifier(random_state=42)
gb_classifier.fit(X_train, y_train)
y_pred = gb_classifier.predict(X_test)
y_proba = gb_classifier.predict_proba(X_test)[:, 1]
```

Оценка качества модели классификации

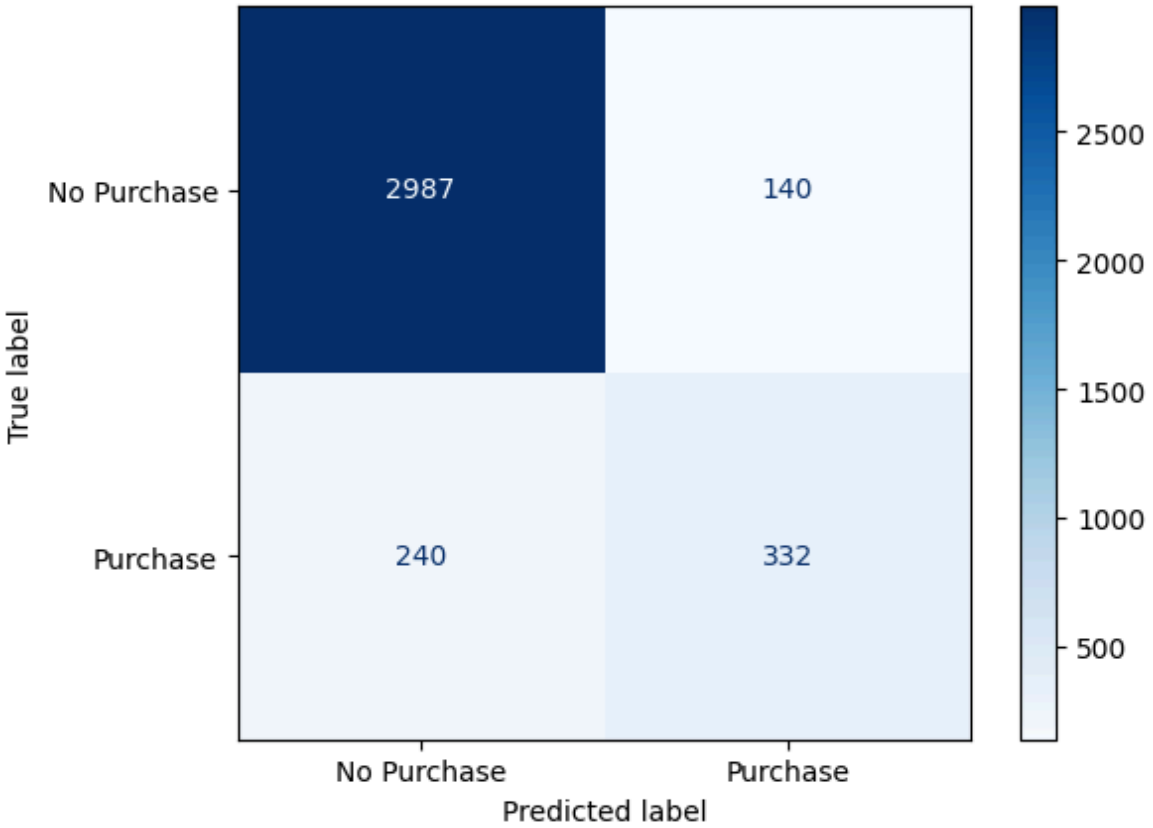
In [10]:
```
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_proba)

print(f"Accuracy: {accuracy:.4f}")
print(f"F1-score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")
```

```
Accuracy: 0.8973
F1-score: 0.6360
ROC-AUC: 0.9243
```

Визуализация матрицы ошибок

In [11]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['No Purchase', 'Purchase'])
disp.plot(cmap='Blues')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



Дополнительная оценка результатов модели

In [12]:
```
TN, FP, FN, TP = cm.ravel()
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print(f"  Precision: {precision:.3f}")
print(f"    - Из {TP+FP} предсказанных покупок, {TP} были верными")
print(f"  Recall: {recall:.3f}")
print(f"    - Из {TP+FN} реальных покупок, нашли {TP}")
```

```
Precision: 0.703
  - Из 472 предсказанных покупок, 332 были верными
Recall: 0.580
  - Из 572 реальных покупок, нашли 332
```

Регрессия

Загрузка датасета

In [13]:
```
df_reg = pd.read_csv('datasets/parkinsons.csv')
```

Размер датасета

In [14]:
```
df_reg.shape
```

Out[14]:
```
(5875, 22)
```

Первые 5 строк

In [15]:
```
df_reg.head()
```

Out[15]:

	subject#	age	sex	test_time	motor_UPDRS	total_UPDRS	Jitter(%)	Jitter(Abs)	Jit
0	1	72	0	5.6431	28.199	34.398	0.00662	0.000034	
1	1	72	0	12.6660	28.447	34.894	0.00300	0.000017	
2	1	72	0	19.6810	28.695	35.389	0.00481	0.000025	
3	1	72	0	25.6470	28.905	35.810	0.00528	0.000027	
4	1	72	0	33.6420	29.187	36.375	0.00335	0.000020	

5 rows × 22 columns

Информация о данных

In [16]:
```
df_reg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5875 entries, 0 to 5874
Data columns (total 22 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   subject#        5875 non-null   int64  
 1   age             5875 non-null   int64  
 2   sex             5875 non-null   int64  
 3   test_time       5875 non-null   float64
 4   motor_UPDRS     5875 non-null   float64
 5   total_UPDRS     5875 non-null   float64
 6   Jitter(%)       5875 non-null   float64
 7   Jitter(Abs)     5875 non-null   float64
 8   Jitter:RAP      5875 non-null   float64
 9   Jitter:PPQ5     5875 non-null   float64
 10  Jitter:DDP      5875 non-null   float64
 11  Shimmer         5875 non-null   float64
 12  Shimmer(dB)     5875 non-null   float64
 13  Shimmer:APQ3    5875 non-null   float64
 14  Shimmer:APQ5    5875 non-null   float64
 15  Shimmer:APQ11   5875 non-null   float64
 16  Shimmer:DDA     5875 non-null   float64
 17  NHR             5875 non-null   float64
 18  HNR             5875 non-null   float64
 19  RPDE            5875 non-null   float64
 20  DFA             5875 non-null   float64
 21  PPE             5875 non-null   float64
dtypes: float64(19), int64(3)
memory usage: 1009.9 KB
```

Статистика по числовым признакам

In [17]:
```
df_reg.describe()
```

Out[17]:

	subject#	age	sex	test_time	motor_UPDRS	total_UPDRS
count	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000	5875.000000
mean	21.494128	64.804936	0.317787	92.863722	21.296229	29.018942
std	12.372279	8.821524	0.465656	53.445602	8.129282	10.700283
min	1.000000	36.000000	0.000000	-4.262500	5.037700	7.000000
25%	10.000000	58.000000	0.000000	46.847500	15.000000	21.371000
50%	22.000000	65.000000	0.000000	91.523000	20.871000	27.576000
75%	33.000000	72.000000	1.000000	138.445000	27.596500	36.399000
max	42.000000	85.000000	1.000000	215.490000	39.511000	54.992000

8 rows × 22 columns

Выделение признаков и таргета, их разделение на выборки для обучения и тестирования

In [18]:
```
df_reg_clean = df_reg.copy()
df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X = df_reg_clean.drop('total_UPDRS', axis=1)
y = df_reg_clean['total_UPDRS']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(f"Обучающая выборка: {X_train.shape}")
print(f"Тестовая выборка: {X_test.shape}")
```

```
Обучающая выборка: (4112, 20)
Тестовая выборка: (1763, 20)
```

Обучение модели регрессии `GradientBoostingRegressor`

In [19]:
```
gb_regressor = GradientBoostingRegressor(random_state=42)
gb_regressor.fit(X_train, y_train)
y_pred = gb_regressor.predict(X_test)
```

Оценка качества модели регрессии

In [20]:
```
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MSE: {mse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"R²: {r2:.4f}")
```

```
MSE: 2.1377
MAE: 1.1418
R²: 0.9810
```

Улучшение бейзлайна

Классификация

Сохранение метрик базовой модели

In [21]:
```python
class_base_metrics = {
    'Accuracy': accuracy,
    'F1': f1,
    'ROC-AUC': roc_auc,
    'Precision': precision,
    'Recall': recall
}
```

Функция сравнения метрик новой модели с базовой

In [22]:
```python
def print_comparison_class(metrics_old, metrics_new):
    comparison_data = []
    for metric in ['Accuracy', 'F1', 'ROC-AUC', 'Precision', 'Recall']:
        base_val = metrics_old[metric]
        new_val = metrics_new[metric]
        diff = new_val - base_val
        change = "улучшение" if diff > 0 else "ухудшение"

        comparison_data.append({
            'Метрика': metric,
            'Базовая модель': f"{base_val:.4f}",
            'Новая модель': f"{new_val:.4f}",
            'Разница': f"{diff:+.4f}",
            'Изменение': change
        })

    df_comparison = pd.DataFrame(comparison_data)
    print(df_comparison.to_string(index=False))
```

Повторное копирование и разделение данных

In [23]:
```python
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le

X_class = df_class_clean.drop('Revenue', axis=1)
y_class = df_class_clean['Revenue']

X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
```

```
    X_class, y_class, test_size=0.3, random_state=42, stratify=y_class
)
```

Гипотеза 1: Подбор гиперпараметров

In [24]:
```
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0]
}

gb_classifier_grid = GradientBoostingClassifier(random_state=42)
grid_search = GridSearchCV(gb_classifier_grid, param_grid, cv=5,
                           scoring='roc_auc', n_jobs=-1, verbose=1)
grid_search.fit(X_train_class, y_train_class)

print("Лучшие параметры:")
for param, value in grid_search.best_params_.items():
    print(f"  {param}: {value}")

gb_classifier_tuned = grid_search.best_estimator_
y_pred_tuned = gb_classifier_tuned.predict(X_test_class)
y_proba_tuned = gb_classifier_tuned.predict_proba(X_test_class)[:, 1]

metrics_tuned = {
    'Accuracy': accuracy_score(y_test_class, y_pred_tuned),
    'F1': f1_score(y_test_class, y_pred_tuned),
    'ROC-AUC': roc_auc_score(y_test_class, y_proba_tuned),
    'Precision': precision_score(y_test_class, y_pred_tuned),
    'Recall': recall_score(y_test_class, y_pred_tuned)
}

print("\nПодбор гиперпараметров")
print_comparison_class(class_base_metrics, metrics_tuned)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Лучшие параметры:
  learning_rate: 0.01
  max_depth: 5
  n_estimators: 200
  subsample: 0.8

Подбор гиперпараметров
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8973       0.8986 +0.0014 улучшение
       F1         0.6360       0.6170 -0.0191 ухудшение
  ROC-AUC         0.9243       0.9238 -0.0005 ухудшение
Precision         0.7034       0.7420 +0.0386 улучшение
   Recall         0.5804       0.5280 -0.0524 ухудшение
```

Гипотеза 2: One-hot encoding вместо Label Encoding для категориальных признаков

In [26]:
```
df_class_ohe = df_class.copy()

categorical_cols = df_class_ohe.select_dtypes(include=['object']).columns.tolist

X_cat = df_class_ohe[categorical_cols]
X_num = df_class_ohe.drop(categorical_cols + ['Revenue'], axis=1)
```

```
ohe = OneHotEncoder(drop='first', sparse_output=False)
X_cat_encoded = ohe.fit_transform(X_cat)

X_ohe = np.hstack([X_num.values, X_cat_encoded])
y_ohe = df_class_ohe['Revenue']

X_train_ohe, X_test_ohe, y_train_ohe, y_test_ohe = train_test_split(
    X_ohe, y_ohe, test_size=0.3, random_state=42, stratify=y_ohe
)

gb_classifier_ohe = GradientBoostingClassifier(random_state=42, n_estimators=100
gb_classifier_ohe.fit(X_train_ohe, y_train_ohe)
y_pred_ohe = gb_classifier_ohe.predict(X_test_ohe)
y_proba_ohe = gb_classifier_ohe.predict_proba(X_test_ohe)[:, 1]

metrics_ohe = {
    'Accuracy': accuracy_score(y_test_ohe, y_pred_ohe),
    'F1': f1_score(y_test_ohe, y_pred_ohe),
    'ROC-AUC': roc_auc_score(y_test_ohe, y_proba_ohe),
    'Precision': precision_score(y_test_ohe, y_pred_ohe),
    'Recall': recall_score(y_test_ohe, y_pred_ohe)
}

print("One-Hot Encoding вместо Label Encoding")
print_comparison_class(class_base_metrics, metrics_ohe)
```

```
One-Hot Encoding вместо Label Encoding
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8973       0.8981 +0.0008 улучшение
       F1         0.6360       0.6392 +0.0032 улучшение
  ROC-AUC         0.9243       0.9265 +0.0021 улучшение
Precision         0.7034       0.7061 +0.0027 улучшение
   Recall         0.5804       0.5839 +0.0035 улучшение
```

Гипотеза 3: Ранняя для предотвращения переобучения

In [27]:

```
gb_classifier_early = GradientBoostingClassifier(
    random_state=42,
    n_estimators=500,
    learning_rate=0.1,
    max_depth=5,
    validation_fraction=0.2,
    n_iter_no_change=10,
    tol=1e-4
)
gb_classifier_early.fit(X_train_class, y_train_class)
print(f"Фактическое количество деревьев после early stopping: {len(gb_classifier

y_pred_early = gb_classifier_early.predict(X_test_class)
y_proba_early = gb_classifier_early.predict_proba(X_test_class)[:, 1]

metrics_early = {
    'Accuracy': accuracy_score(y_test_class, y_pred_early),
    'F1': f1_score(y_test_class, y_pred_early),
    'ROC-AUC': roc_auc_score(y_test_class, y_proba_early),
    'Precision': precision_score(y_test_class, y_pred_early),
    'Recall': recall_score(y_test_class, y_pred_early)
}
```

```
print("\nEarly stopping")
print_comparison_class(class_base_metrics, metrics_early)
```

```
Фактическое количество деревьев после early stopping: 50

Early stopping
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8973       0.8978 +0.0005 улучшение
       F1         0.6360       0.6344 -0.0016 ухудшение
  ROC-AUC         0.9243       0.9223 -0.0020 ухудшение
Precision         0.7034       0.7100 +0.0066 улучшение
   Recall         0.5804       0.5734 -0.0070 ухудшение
```

Гипотеза 4: Комбинация лучших техник

In [28]:
```
best_params_hyp1 = grid_search.best_params_
gb_classifier_best = GradientBoostingClassifier(
    random_state=42,
    learning_rate=best_params_hyp1.get('learning_rate', 0.1),
    n_estimators=500,
    max_depth=best_params_hyp1.get('max_depth', 5),
    subsample=best_params_hyp1.get('subsample', 1.0),
    validation_fraction=0.2,
    n_iter_no_change=10,
    tol=1e-4
)
gb_classifier_best.fit(X_train_class, y_train_class)
print(f"Фактическое количество деревьев после early stopping: {len(gb_classifier

y_pred_best = gb_classifier_best.predict(X_test_class)
y_proba_best = gb_classifier_best.predict_proba(X_test_class)[:, 1]

class_improved_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_best),
    'F1': f1_score(y_test_class, y_pred_best),
    'ROC-AUC': roc_auc_score(y_test_class, y_proba_best),
    'Precision': precision_score(y_test_class, y_pred_best),
    'Recall': recall_score(y_test_class, y_pred_best)
}

print("\nКомбинация лучших техник")
print_comparison_class(class_base_metrics, class_improved_metrics)
```

```
Фактическое количество деревьев после early stopping: 301

Комбинация лучших техник
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8973       0.9013 +0.0041 улучшение
       F1         0.6360       0.6418 +0.0058 улучшение
  ROC-AUC         0.9243       0.9244 +0.0001 улучшение
Precision         0.7034       0.7315 +0.0282 улучшение
   Recall         0.5804       0.5717 -0.0087 ухудшение
```

Формирование улучшенного бейзлайна на основе лучших результатов

In [29]:
```
all_metrics = {
    'tuned': metrics_tuned,
    'ohe': metrics_ohe,
    'early': metrics_early,
    'best': class_improved_metrics
```

```
}

best_model_name = max(all_metrics, key=lambda x: all_metrics[x]['ROC-AUC'])
print(f"Лучшая модель: {best_model_name} с ROC-AUC = {all_metrics[best_model_nam

if best_model_name == 'tuned':
    improved_gb_classifier = gb_classifier_tuned
elif best_model_name == 'ohe':
    improved_gb_classifier = gb_classifier_ohe
elif best_model_name == 'early':
    improved_gb_classifier = gb_classifier_early
else:
    improved_gb_classifier = gb_classifier_best

improved_class_metrics = all_metrics[best_model_name]
print("\nУлучшенный бейзлайн (классификация):")
print_comparison_class(class_base_metrics, improved_class_metrics)
print("\nВыводы:")
print(f"- Лучшая техника улучшения: {best_model_name}")
print(f"- Улучшение ROC-AUC: {improved_class_metrics['ROC-AUC'] - class_base_met
print(f"- Улучшение F1-score: {improved_class_metrics['F1'] - class_base_metrics
```

```
Лучшая модель: ohe с ROC-AUC = 0.9265

Улучшенный бейзлайн (классификация):
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8973       0.8981 +0.0008 улучшение
       F1         0.6360       0.6392 +0.0032 улучшение
  ROC-AUC         0.9243       0.9265 +0.0021 улучшение
Precision         0.7034       0.7061 +0.0027 улучшение
   Recall         0.5804       0.5839 +0.0035 улучшение

Выводы:
- Лучшая техника улучшения: ohe
- Улучшение ROC-AUC: +0.0021
- Улучшение F1-score: +0.0032
```

Регрессия

Сохранение метрик базовой модели

In [30]:
```
reg_base_metrics = {
    'MSE': mse,
    'MAE': mae,
    'R²': r2
}
```

Функция сравнения метрик новой модели с базовой

In [31]:
```
def print_comparison_reg(metrics_old, metrics_new):
    comparison_data = []
    for metric in ['MSE', 'MAE', 'R²']:
        base_val = metrics_old[metric]
        new_val = metrics_new[metric]
        diff = new_val - base_val
        if metric == 'R²':
            change = "улучшение" if diff > 0 else "ухудшение"
        else:
```

```
            change = "улучшение" if diff < 0 else "ухудшение"

        comparison_data.append({
            'Метрика': metric,
            'Базовая модель': f"{base_val:.4f}",
            'Новая модель': f"{new_val:.4f}",
            'Разница': f"{diff:+.4f}",
            'Изменение': change
        })

    df_comparison = pd.DataFrame(comparison_data)
    print(df_comparison.to_string(index=False))
```

Повторное копирование и разделение данных

In [32]:
```
df_reg_clean = df_reg.copy()
df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X_reg = df_reg_clean.drop('total_UPDRS', axis=1)
y_reg = df_reg_clean['total_UPDRS']

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.3, random_state=42
)
```

Гипотеза 1: Подбор гиперпараметров

In [33]:
```
param_grid_reg = {
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0]
}

gb_regressor_grid = GradientBoostingRegressor(random_state=42)
grid_search_reg = GridSearchCV(gb_regressor_grid, param_grid_reg, cv=5,
                               scoring='neg_mean_squared_error', n_jobs=-1, verb
grid_search_reg.fit(X_train_reg, y_train_reg)

print("Лучшие параметры:")
for param, value in grid_search_reg.best_params_.items():
    print(f"  {param}: {value}")

gb_regressor_tuned = grid_search_reg.best_estimator_
y_pred_tuned_reg = gb_regressor_tuned.predict(X_test_reg)

metrics_tuned_reg = {
    'MSE': mean_squared_error(y_test_reg, y_pred_tuned_reg),
    'MAE': mean_absolute_error(y_test_reg, y_pred_tuned_reg),
    'R²': r2_score(y_test_reg, y_pred_tuned_reg)
}

print("\nПодбор гиперпараметров")
print_comparison_reg(reg_base_metrics, metrics_tuned_reg)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Лучшие параметры:
  learning_rate: 0.1
  max_depth: 7
  n_estimators: 200
  subsample: 1.0

Подбор гиперпараметров
Метрика Базовая модель Новая модель Разница Изменение
    MSE         2.1377       0.1953 -1.9424 улучшение
    MAE         1.1418       0.2434 -0.8984 улучшение
     R²         0.9810       0.9983 +0.0173 улучшение
```

Гипотеза 2: Early stopping для предотвращения переобучения

In [34]:
```
gb_regressor_early = GradientBoostingRegressor(
    random_state=42,
    n_estimators=500,
    learning_rate=0.1,
    max_depth=5,
    validation_fraction=0.2,
    n_iter_no_change=10,
    tol=1e-4
)
gb_regressor_early.fit(X_train_reg, y_train_reg)
print(f"Фактическое количество деревьев после early stopping: {len(gb_regressor_

y_pred_early_reg = gb_regressor_early.predict(X_test_reg)

metrics_early_reg = {
    'MSE': mean_squared_error(y_test_reg, y_pred_early_reg),
    'MAE': mean_absolute_error(y_test_reg, y_pred_early_reg),
    'R²': r2_score(y_test_reg, y_pred_early_reg)
}

print("\nEarly stopping")
print_comparison_reg(reg_base_metrics, metrics_early_reg)
```

```
Фактическое количество деревьев после early stopping: 334

Early stopping
Метрика Базовая модель Новая модель Разница Изменение
    MSE         2.1377       0.3813 -1.7564 улучшение
    MAE         1.1418       0.4325 -0.7093 улучшение
     R²         0.9810       0.9966 +0.0156 улучшение
```

Гипотеза 3: Комбинация лучших техник

In [35]:
```
best_params_reg_hyp1 = grid_search_reg.best_params_
gb_regressor_best = GradientBoostingRegressor(
    random_state=42,
    learning_rate=best_params_reg_hyp1.get('learning_rate', 0.1),
    n_estimators=500,
    max_depth=best_params_reg_hyp1.get('max_depth', 5),
    subsample=best_params_reg_hyp1.get('subsample', 1.0),
    validation_fraction=0.2,
    n_iter_no_change=10,
    tol=1e-4
)
gb_regressor_best.fit(X_train_reg, y_train_reg)
```

```
print(f"Фактическое количество деревьев после early stopping: {len(gb_regressor_

y_pred_best_reg = gb_regressor_best.predict(X_test_reg)

reg_improved_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_best_reg),
    'MAE': mean_absolute_error(y_test_reg, y_pred_best_reg),
    'R²': r2_score(y_test_reg, y_pred_best_reg)
}

print("\nКомбинация лучших техник")
print_comparison_reg(reg_base_metrics, reg_improved_metrics)
```

```
Фактическое количество деревьев после early stopping: 221

Комбинация лучших техник
Метрика Базовая модель Новая модель Разница Изменение
    MSE         2.1377       0.2420 -1.8957 улучшение
    MAE         1.1418       0.2817 -0.8601 улучшение
     R²         0.9810       0.9978 +0.0169 улучшение
```

Формирование улучшенного бейзлайна на основе лучших результатов

In [36]:
```
all_metrics_reg = {
    'tuned': metrics_tuned_reg,
    'early': metrics_early_reg,
    'best': reg_improved_metrics
}

best_model_name_reg = max(all_metrics_reg, key=lambda x: all_metrics_reg[x]['R²'
print(f"Лучшая модель: {best_model_name_reg} с R² = {all_metrics_reg[best_model_

if best_model_name_reg == 'tuned':
    improved_gb_regressor = gb_regressor_tuned
elif best_model_name_reg == 'early':
    improved_gb_regressor = gb_regressor_early
else:
    improved_gb_regressor = gb_regressor_best

improved_reg_metrics = all_metrics_reg[best_model_name_reg]
print("\nУлучшенный бейзлайн (регрессия):")
print_comparison_reg(reg_base_metrics, improved_reg_metrics)
print("\nВыводы:")
print(f"- Лучшая техника улучшения: {best_model_name_reg}")
print(f"- Улучшение R²: {improved_reg_metrics['R²'] - reg_base_metrics['R²']:+.4
print(f"- Улучшение MSE: {improved_reg_metrics['MSE'] - reg_base_metrics['MSE']:
```

```
Лучшая модель: tuned с R² = 0.9983

Улучшенный бейзлайн (регрессия):
Метрика Базовая модель Новая модель Разница Изменение
    MSE         2.1377       0.1953 -1.9424 улучшение
    MAE         1.1418       0.2434 -0.8984 улучшение
     R²         0.9810       0.9983 +0.0173 улучшение

Выводы:
- Лучшая техника улучшения: tuned
- Улучшение R²: +0.0173
- Улучшение MSE: -1.9424
```

Имплементация алгоритма машинного обучения

Классификация

Кастомная модель градиентного бустинга для классификации

In [40]:
```
from sklearn.tree import DecisionTreeRegressor

class CustomGradientBoostingClassifier:

    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=3,
                 subsample=1.0, random_state=None):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.subsample = subsample
        self.random_state = random_state
        self.estimators_ = []
        self.initial_prediction_ = None

    def _sigmoid(self, z):
        z = np.clip(z, -500, 500)
        return 1 / (1 + np.exp(-z))

    def _log_loss_gradient(self, y_true, y_pred):
        epsilon = 1e-15
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
        return y_pred - y_true

    def fit(self, X, y, verbose=False):
        np.random.seed(self.random_state)
        n_samples, n_features = X.shape

        positive_ratio = np.mean(y)
        self.initial_prediction_ = np.log(positive_ratio / (1 - positive_ratio +

        current_predictions = np.full(n_samples, self.initial_prediction_)

        self.estimators_ = []

        for i in range(self.n_estimators):
            probabilities = self._sigmoid(current_predictions)
            gradients = self._log_loss_gradient(y, probabilities)

            if self.subsample < 1.0:
                n_subset = int(self.subsample * n_samples)
                indices = np.random.choice(n_samples, n_subset, replace=False)
                X_subset = X[indices]
                gradients_subset = gradients[indices]
            else:
                X_subset = X
                gradients_subset = gradients

            tree = DecisionTreeRegressor(
                max_depth=self.max_depth,
```

```
                random_state=self.random_state
            )
            tree.fit(X_subset, -gradients_subset)

            tree_predictions = tree.predict(X)

            current_predictions += self.learning_rate * tree_predictions

            self.estimators_.append(tree)

            if verbose and (i + 1) % 10 == 0:
                current_probs = self._sigmoid(current_predictions)
                loss = -np.mean(y * np.log(current_probs + 1e-15) +
                                (1 - y) * np.log(1 - current_probs + 1e-15))
                print(f"Iteration {i+1}/{self.n_estimators}, Loss: {loss:.4f}")

    def predict_proba(self, X):
        n_samples = X.shape[0]
        predictions = np.full(n_samples, self.initial_prediction_)

        for tree in self.estimators_:
            predictions += self.learning_rate * tree.predict(X)

        probabilities = self._sigmoid(predictions)
        return np.column_stack([1 - probabilities, probabilities])

    def predict(self, X, threshold=0.5):
        probabilities = self.predict_proba(X)[:, 1]
        return (probabilities >= threshold).astype(int)
```

Повторное копирование и разбиение данных

In [41]:
```
df_class_clean = df_class.copy()

categorical_cols = df_class_clean.select_dtypes(include=['object']).columns.toli
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_class_clean[col] = le.fit_transform(df_class_clean[col].astype(str))
    label_encoders[col] = le

X_class = df_class_clean.drop('Revenue', axis=1)
y_class = df_class_clean['Revenue']

X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class, test_size=0.3, random_state=42, stratify=y_class
)
```

Обучение кастомной модели градиентного бустинга

In [42]:
```
custom_gb_classifier = CustomGradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=5,
    subsample=1.0,
    random_state=42
)

custom_gb_classifier.fit(X_train_class, y_train_class, verbose=True)
```

```
y_pred_custom = custom_gb_classifier.predict(X_test_class)
y_proba_custom = custom_gb_classifier.predict_proba(X_test_class)[:, 1]
```

```
Iteration 10/100, Loss: 0.3734
Iteration 20/100, Loss: 0.3318
Iteration 30/100, Loss: 0.3028
Iteration 40/100, Loss: 0.2825
Iteration 50/100, Loss: 0.2677
Iteration 60/100, Loss: 0.2563
Iteration 70/100, Loss: 0.2473
Iteration 80/100, Loss: 0.2401
Iteration 90/100, Loss: 0.2338
Iteration 100/100, Loss: 0.2285
```

Метрики кастомной модели градиентного бустинга

In [43]:
```
custom_class_base_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_custom),
    'F1': f1_score(y_test_class, y_pred_custom),
    'ROC-AUC': roc_auc_score(y_test_class, y_proba_custom),
    'Precision': precision_score(y_test_class, y_pred_custom),
    'Recall': recall_score(y_test_class, y_pred_custom)
}

print("Кастомная модель градиентного бустинга (классификация):")
print_comparison_class(class_base_metrics, custom_class_base_metrics)
print("\nВыводы:")
print(f"- Кастомная модель показывает {'лучшие' if custom_class_base_metrics['RO
print(f"- Разница в ROC-AUC: {custom_class_base_metrics['ROC-AUC'] - class_base_
```

```
Кастомная модель градиентного бустинга (классификация):
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8973       0.9011 +0.0038 улучшение
       F1         0.6360       0.6340 -0.0020 ухудшение
  ROC-AUC         0.9243       0.9225 -0.0018 ухудшение
Precision         0.7034       0.7407 +0.0373 улучшение
   Recall         0.5804       0.5542 -0.0262 ухудшение

Выводы:
- Кастомная модель показывает худшие результаты по сравнению с базовой моделью sk
learn
- Разница в ROC-AUC: -0.0018
```

Добавление техник из улучшенного бейзлайна к кастомной модели

In [44]:
```
if best_model_name == 'tuned':
    best_params = grid_search.best_params_
elif best_model_name == 'best':
    best_params = grid_search.best_params_
else:
    best_params = {'learning_rate': 0.1, 'max_depth': 5, 'subsample': 1.0}

custom_gb_classifier_improved = CustomGradientBoostingClassifier(
    n_estimators=200,
    learning_rate=best_params.get('learning_rate', 0.1),
    max_depth=best_params.get('max_depth', 5),
    subsample=best_params.get('subsample', 1.0),
    random_state=42
)
```

```
custom_gb_classifier_improved.fit(X_train_class, y_train_class, verbose=True)

y_pred_custom_improved = custom_gb_classifier_improved.predict(X_test_class)
y_proba_custom_improved = custom_gb_classifier_improved.predict_proba(X_test_cla
```

```
Iteration 10/200, Loss: 0.3734
Iteration 20/200, Loss: 0.3318
Iteration 30/200, Loss: 0.3028
Iteration 40/200, Loss: 0.2825
Iteration 50/200, Loss: 0.2677
Iteration 60/200, Loss: 0.2563
Iteration 70/200, Loss: 0.2473
Iteration 80/200, Loss: 0.2401
Iteration 90/200, Loss: 0.2338
Iteration 100/200, Loss: 0.2285
Iteration 110/200, Loss: 0.2241
Iteration 120/200, Loss: 0.2200
Iteration 130/200, Loss: 0.2165
Iteration 140/200, Loss: 0.2131
Iteration 150/200, Loss: 0.2100
Iteration 160/200, Loss: 0.2071
Iteration 170/200, Loss: 0.2044
Iteration 180/200, Loss: 0.2019
Iteration 190/200, Loss: 0.1997
Iteration 200/200, Loss: 0.1976
```

Метрики улучшенной кастомной модели

In [45]:
```
custom_improved_class_metrics = {
    'Accuracy': accuracy_score(y_test_class, y_pred_custom_improved),
    'F1': f1_score(y_test_class, y_pred_custom_improved),
    'ROC-AUC': roc_auc_score(y_test_class, y_proba_custom_improved),
    'Precision': precision_score(y_test_class, y_pred_custom_improved),
    'Recall': recall_score(y_test_class, y_pred_custom_improved)
}

print("Улучшенная кастомная модель (классификация):")
print_comparison_class(improved_class_metrics, custom_improved_class_metrics)
print("\nВыводы:")
print(f"- Улучшенная кастомная модель показывает {'лучшие' if custom_improved_cl
print(f"- Разница в ROC-AUC: {custom_improved_class_metrics['ROC-AUC'] - improve
```

```
Улучшенная кастомная модель (классификация):
  Метрика Базовая модель Новая модель Разница Изменение
 Accuracy         0.8981       0.8994 +0.0014 улучшение
       F1         0.6392       0.6402 +0.0010 улучшение
  ROC-AUC         0.9265       0.9232 -0.0033 ухудшение
Precision         0.7061       0.7165 +0.0103 улучшение
   Recall         0.5839       0.5787 -0.0052 ухудшение

Выводы:
- Улучшенная кастомная модель показывает худшие результаты по сравнению с улучшен
ным бейзлайном
- Разница в ROC-AUC: -0.0033
```

Регрессия

Кастомная модель градиентного бустинга для регрессии

In [46]:

```python
class CustomGradientBoostingRegressor:

    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=3,
                 subsample=1.0, random_state=None):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.subsample = subsample
        self.random_state = random_state
        self.estimators_ = []
        self.initial_prediction_ = None

    def fit(self, X, y, verbose=False):
        np.random.seed(self.random_state)
        n_samples, n_features = X.shape

        self.initial_prediction_ = np.mean(y)

        current_predictions = np.full(n_samples, self.initial_prediction_)

        self.estimators_ = []

        for i in range(self.n_estimators):
            residuals = y - current_predictions

            if self.subsample < 1.0:
                n_subset = int(self.subsample * n_samples)
                indices = np.random.choice(n_samples, n_subset, replace=False)
                X_subset = X[indices]
                residuals_subset = residuals[indices]
            else:
                X_subset = X
                residuals_subset = residuals

            tree = DecisionTreeRegressor(
                max_depth=self.max_depth,
                random_state=self.random_state
            )
            tree.fit(X_subset, residuals_subset)

            tree_predictions = tree.predict(X)

            current_predictions += self.learning_rate * tree_predictions

            self.estimators_.append(tree)

            if verbose and (i + 1) % 10 == 0:
                mse = np.mean((y - current_predictions) ** 2)
                print(f"Iteration {i+1}/{self.n_estimators}, MSE: {mse:.4f}")

    def predict(self, X):
        """Предсказание значений"""
        n_samples = X.shape[0]
        predictions = np.full(n_samples, self.initial_prediction_)

        for tree in self.estimators_:
            predictions += self.learning_rate * tree.predict(X)

        return predictions
```

Повторное копирование и разбиение данных

In [47]:
```
df_reg_clean = df_reg.copy()
df_reg_clean = df_reg_clean.drop('subject#', axis=1)

X_reg = df_reg_clean.drop('total_UPDRS', axis=1)
y_reg = df_reg_clean['total_UPDRS']

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.3, random_state=42
)
```

Обучение кастомной модели градиентного бустинга

In [48]:
```
custom_gb_regressor = CustomGradientBoostingRegressor(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=5,
    subsample=1.0,
    random_state=42
)

custom_gb_regressor.fit(X_train_reg, y_train_reg, verbose=True)

y_pred_custom_reg = custom_gb_regressor.predict(X_test_reg)
```

```
Iteration 10/100, MSE: 17.4660
Iteration 20/100, MSE: 4.0557
Iteration 30/100, MSE: 1.6137
Iteration 40/100, MSE: 0.8979
Iteration 50/100, MSE: 0.6307
Iteration 60/100, MSE: 0.5339
Iteration 70/100, MSE: 0.4646
Iteration 80/100, MSE: 0.4062
Iteration 90/100, MSE: 0.3567
Iteration 100/100, MSE: 0.3088
```

Метрики кастомной модели градиентного бустинга

In [49]:
```
custom_reg_base_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_custom_reg),
    'MAE': mean_absolute_error(y_test_reg, y_pred_custom_reg),
    'R²': r2_score(y_test_reg, y_pred_custom_reg)
}

print("Кастомная модель градиентного бустинга (регрессия):")
print_comparison_reg(reg_base_metrics, custom_reg_base_metrics)
print("\nВыводы:")
print(f"- Кастомная модель показывает {'лучшие' if custom_reg_base_metrics['R²']
print(f"- Разница в R²: {custom_reg_base_metrics['R²'] - reg_base_metrics['R²']:
```

```
Кастомная модель градиентного бустинга (регрессия):
Метрика Базовая модель Новая модель Разница Изменение
    MSE         2.1377       0.5550 -1.5827 улучшение
    MAE         1.1418       0.5366 -0.6052 улучшение
     R²         0.9810       0.9951 +0.0141 улучшение

Выводы:
- Кастомная модель показывает лучшие результаты по сравнению с базовой моделью sk
learn
- Разница в R²: +0.0141
```

Добавление техник из улучшенного бейзлайна к кастомной модели

In [50]:
```python
if best_model_name_reg == 'tuned' or best_model_name_reg == 'best':
    best_params_reg = grid_search_reg.best_params_
else:
    best_params_reg = {'learning_rate': 0.1, 'max_depth': 5, 'subsample': 1.0}

custom_gb_regressor_improved = CustomGradientBoostingRegressor(
    n_estimators=200,
    learning_rate=best_params_reg.get('learning_rate', 0.1),
    max_depth=best_params_reg.get('max_depth', 5),
    subsample=best_params_reg.get('subsample', 1.0),
    random_state=42
)

custom_gb_regressor_improved.fit(X_train_reg, y_train_reg, verbose=True)

y_pred_custom_improved_reg = custom_gb_regressor_improved.predict(X_test_reg)
```

```
Iteration 10/200, MSE: 15.0549
Iteration 20/200, MSE: 2.2405
Iteration 30/200, MSE: 0.4688
Iteration 40/200, MSE: 0.1591
Iteration 50/200, MSE: 0.0907
Iteration 60/200, MSE: 0.0650
Iteration 70/200, MSE: 0.0518
Iteration 80/200, MSE: 0.0425
Iteration 90/200, MSE: 0.0360
Iteration 100/200, MSE: 0.0313
Iteration 110/200, MSE: 0.0263
Iteration 120/200, MSE: 0.0232
Iteration 130/200, MSE: 0.0197
Iteration 140/200, MSE: 0.0169
Iteration 150/200, MSE: 0.0148
Iteration 160/200, MSE: 0.0127
Iteration 170/200, MSE: 0.0111
Iteration 180/200, MSE: 0.0095
Iteration 190/200, MSE: 0.0086
Iteration 200/200, MSE: 0.0074
```

Метрики улучшенной кастомной модели

In [51]:
```python
custom_reg_improved_metrics = {
    'MSE': mean_squared_error(y_test_reg, y_pred_custom_improved_reg),
    'MAE': mean_absolute_error(y_test_reg, y_pred_custom_improved_reg),
    'R²': r2_score(y_test_reg, y_pred_custom_improved_reg)
}

print("Улучшенная кастомная модель (регрессия):")
```

```
print_comparison_reg(improved_reg_metrics, custom_reg_improved_metrics)
print("\nВыводы:")
print(f"- Улучшенная кастомная модель показывает {'лучшие' if custom_reg_improve
print(f"- Разница в R²: {custom_reg_improved_metrics['R²'] - improved_reg_metric
```

```
Улучшенная кастомная модель (регрессия):
Метрика Базовая модель Новая модель Разница Изменение
    MSE         0.1953       0.1925 -0.0028 улучшение
    MAE         0.2434       0.2395 -0.0040 улучшение
     R²         0.9983       0.9983 +0.0000 улучшение

Выводы:
- Улучшенная кастомная модель показывает лучшие результаты по сравнению с улучшен
ным бейзлайном
- Разница в R²: +0.0000
```

In [52]:
```
print("СРАВНЕНИЕ БАЗОВЫХ И КАСТОМНЫХ МОДЕЛЕЙ:")

print("\nКлассификация:")
print("  • Кастомная реализация логистической регрессии показала:")
print(f"    - Accuracy: {custom_class_base_metrics['Accuracy']:.4f} vs {class_ba
print(f"    - F1-Score: {custom_class_base_metrics['F1']:.4f} vs {class_base_met
print(f"    - Recall:   {custom_class_base_metrics['Recall']:.4f} vs {class_base

print("\nРегрессия:")
print("  • Кастомная реализация линейной регрессии показала:")
print(f"    - R²:  {custom_reg_base_metrics['R²']:.4f} vs {reg_base_metrics['R²'
print(f"    - MSE: {custom_reg_base_metrics['MSE']:.4f} vs {reg_base_metrics['MS

print("ЭФФЕКТИВНОСТЬ ТЕХНИК УЛУЧШЕНИЯ:")

print("\nКлассификация:")
print(f"  • Улучшенная кастомная модель vs базовая кастомная:")
print(f"    - F1-Score улучшился: {custom_improved_class_metrics['F1']:.4f} vs {
print(f"    - Recall улучшился:   {custom_improved_class_metrics['Recall']:.4f}
print(f"  • Улучшенная кастомная vs улучшенная sklearn:")
print(f"    - F1-Score: {custom_improved_class_metrics['F1']:.4f} vs {class_impr
print(f"    - Recall:   {custom_improved_class_metrics['Recall']:.4f} vs {class_

print("\nРегрессия:")
print(f"  • Улучшенная кастомная модель vs базовая кастомная:")
print(f"    - R² улучшился:  {custom_reg_improved_metrics['R²']:.4f} vs {custom_
print(f"    - MSE уменьшился: {custom_reg_improved_metrics['MSE']:.4f} vs {custo
print(f"  • Улучшенная кастомная vs улучшенная sklearn:")
print(f"    - R²:  {custom_reg_improved_metrics['R²']:.4f} vs {reg_improved_metr
print(f"    - MSE: {custom_reg_improved_metrics['MSE']:.4f} vs {reg_improved_met
```

СРАВНЕНИЕ БАЗОВЫХ И КАСТОМНЫХ МОДЕЛЕЙ:

Классификация:
- Кастомная реализация логистической регрессии показала:
  - Accuracy: 0.9011 vs 0.8973 (sklearn)
  - F1-Score: 0.6340 vs 0.6360 (sklearn)
  - Recall: 0.5542 vs 0.5804 (sklearn)

Регрессия:
- Кастомная реализация линейной регрессии показала:
  - R²: 0.9951 vs 0.9810 (sklearn)
  - MSE: 0.5550 vs 2.1377 (sklearn)

ЭФФЕКТИВНОСТЬ ТЕХНИК УЛУЧШЕНИЯ:

Классификация:
- Улучшенная кастомная модель vs базовая кастомная:
  - F1-Score улучшился: 0.6402 vs 0.6340 (+0.0062)
  - Recall улучшился: 0.5787 vs 0.5542 (+0.0245)
- Улучшенная кастомная vs улучшенная sklearn:
  - F1-Score: 0.6402 vs 0.6418
  - Recall: 0.5787 vs 0.5717

Регрессия:
- Улучшенная кастомная модель vs базовая кастомная:
  - R² улучшился: 0.9983 vs 0.9951 (+0.0032)
  - MSE уменьшился: 0.1925 vs 0.5550 (0.3625)
- Улучшенная кастомная vs улучшенная sklearn:
  - R²: 0.9983 vs 0.9978
  - MSE: 0.1925 vs 0.2420