

# HR Burnout Lab: прогноз и анализ риска выгорания сотрудников

## Описание проекта

Проект для портфолио, показывающий на практике навыки EDA, построение и оценку моделей бинарной классификации, интерпретацию признаков, проверку гипотез и подготовку рекомендаций для HR. Используется синтетический датасет с имитацией данных на уровне сотрудников о стрессе, удовлетворённости, удалённой работе и нагрузке с платформы Kaggle.

Профессиональное выгорание — серьезная проблема современных организаций, ведущая к снижению продуктивности на 20-30%, увеличению текучести кадров, ухудшению психологического климата и росту затрат на подбор персонала. Задача прогнозирования риска выгорания на основе объективных и субъективных факторов является ключевой для отделов HR и управления персоналом. Своевременное выявление факторов риска и прогнозирование выгорания позволяет компаниям принимать превентивные меры, улучшать условия труда и повышать удовлетворённость сотрудников.

**Цель проекта** — выявить ключевые факторы, влияющие на риск профессионального выгорания сотрудников, построить интерпретируемую модель бинарной классификации ( $\text{Burnout} = 0/1$ ) для его прогнозирования и предложить менеджерам действия по снижению риска. Это позволит сформировать рекомендации для HR-отделов по профилактике выгорания и улучшению благополучия сотрудников.

- Фокус: HR-аналитика, объяснимый ML, сегментация сотрудников, продуктовые метрики.
- Практическая ценность: политика нагрузки (рабочие часы, удалёнка), приоритизация инициатив (поддержка, гибкий график), выявление групп риска и формирование рекомендаций по «точкам вмешательства».

Результаты анализа помогут HR-специалистам и руководству компаний принимать обоснованные решения по улучшению условий труда и предотвращению выгорания сотрудников.

### Практическая ценность

- Результаты проекта могут быть использованы для:
  - Раннего выявления сотрудников группы риска
  - Оптимизации рабочих процессов
  - Разработки программ поддержки сотрудников
  - Улучшения условий труда
  - Снижения текучести кадров

Для проекта по прогнозу риска выгорания сотрудников нужно ориентироваться на метрики, которые лучше отражают ценность для HR-аналитики:

- Основная метрика Recall (чувствительность) для класса  $\text{Burnout}=1$ 
  - Важно не пропустить сотрудников группы риска.
  - Ложные тревоги (false positives) менее критичны, чем пропуск реального случая выгорания.
- Дополнительные метрики
  - PR-AUC (Precision-Recall AUC)
  - F1-score для класса  $\text{Burnout}=1$
  - ROC-AUC

Таким образом, модель будет оцениваться не только по «общей точности», но и по её способности рано выявлять сотрудников с высоким риском выгорания, что и является ключевой бизнес-ценностью.

## Описание данных

Синтетический датасет «Synthetic HR Burnout Dataset» взят с платформы Kaggle: <https://www.kaggle.com/datasets/ankam6010/synthetic-hr-burnout-dataset/>

Датасет содержит следующие признаки (столбцы):

- **Name** — синтетическое имя сотрудника (для реалистичности, не для использования в машинном обучении).
- **Age** — возраст сотрудника.
- **Gender** — пол: мужской (Male) или женский (Female).
- **JobRole** — должность (Инженер, HR, Менеджер и т. д.).
- **Experience** — количество лет рабочего стажа.
- **WorkHoursPerWeek** — среднее количество рабочих часов в неделю.
- **RemoteRatio** — % времени, проведённого на удалённой работе (0–100).
- **SatisfactionLevel** — уровень удовлетворённости по самооценке сотрудника (от 1 до 5).
- **StressLevel** — уровень стресса по самооценке сотрудника (от 1 до 10).
- **Burnout** — целевая переменная: 1 — если присутствуют признаки выгорания (высокий уровень стресса + низкая удовлетворённость + длинные рабочие часы), иначе 0 (выгорания нет).

Этот датасет содержит синтетические данные о сотрудниках различных должностей с информацией о возрасте, стаже, рабочих часах, уровне удалённой работы, самооценке удовлетворённости и стресса, а также целевой переменной — наличии признаков выгорания. Датасет включает как количественные характеристики (возраст, стаж, часы работы), так и качественные (должность, пол), что делает его идеальным для комплексного анализа.

# Цели исследования

- Выявить факторы риска выгорания.
  - Задача: проверить связи WorkHoursPerWeek, SatisfactionLevel, StressLevel, RemoteRatio, Experience, JobRole с Burnout.
  - Метрика результата: ранжирование признаков по важности и интерпретации.
  - Проверить корреляцию между возрастом, стажем и уровнем выгорания.
  - Изучить, как гендерные различия (Gender) связаны с исследуемым феноменом.
- Выявить профили сотрудников с высоким риском выгорания:
  - Определить "портрет" сотрудника, склонного к выгоранию
  - Выявить должности с наибольшим риском
  - Определить критические пороги рабочих часов и уровня стресса
- Построить интерпретируемую модель бинарной классификации.
  - Цель: точность и устойчивость на валидации; объяснение, почему предсказывается класс 1.
  - Метрики: ROC-AUC, PR-AUC, F1, Recall для класса 1 (группа риска).
  - Определить наиболее важные признаки для прогноза, используя feature importance
  - Сформулировать содержательные выводы и практические рекомендации для HR.
- Проверить HR-гипотезы.
  - Примеры:
    - удалёнка снижает риск при высоких часах
    - опыт смягчает стресс
    - роли отличаются по риску
    - критические пороги рабочего времени.
- Сформулировать управленческие рекомендации.
  - Цель: действия по снижению Burnout через политику нагрузки, баланс удалёнки, адресные программы поддержки.
- Предложить практические HR-рекомендации на основе инсайтов (минимальные пороговые значения stress/satisfaction/hours и др.).

# Ход исследования

## Шаг 1: Загрузка данных

- Подключение к Kaggle API, скачивание файлов
- Первичный анализ структуры и типов данных

## Шаг 2: Предобработка данных

- Приведение названий столбцов к единому стилю (snake\_case)
- Проверка и обработка пропусков
- Проверка дубликатов
- Уникальные значения категориальных признаков
- Создание новых признаков (Feature Engineering):
  - Описательная статистика
  - age\_bin: binned возраст (например, <25, 25–34, 35–44, 45+)
  - work\_hours\_bin: категории нагрузки (например: ≤40, 41–50, 51–60, >60)
  - remote\_bin: группы по RemoteRatio (0, 1–30, 31–70, 71–100)
  - remote\_cat: категории удалённой работы 0 (onsite), 1 (hybrid), 2 (remote) по RemoteRatio (0, 1–80, 81–100)
  - stress\_cat: категории стресса Низкий (1-3), Средний (4-6), Высокий (7-10)
  - satisfaction\_cat: категории удовлетворённости Низкая (1.0-2.5), Средняя (2.5-3.5), Высокая (3.5-5.0)
  - experience\_cat: категории стажа Junior (0-3 года), Middle (3-7 лет), Senior (7+ лет)
  - stress\_to\_satisfaction: отношение стресса к удовлетворённости (индикатор дисбаланса)
  - experience\_to\_age: отношение стажа к возрасту (Experience / Age), чем выше, тем позже началась карьера
  - career\_start\_age: Age - Experience (возраст начала карьеры)
  - индекс перегрузки: (WorkHoursPerWeek / 40) \* StressLevel

## Шаг 3: Исследовательский анализ данных (EDA)

- Распределение целевой переменной
  - проверка баланса классов
- Качественные признаки:
  - Гистограммы
  - Ящики с усами
  - Поиск артефактов (анализ выбросов)
- Категориальные признаки:
  - Столбчатые диаграммы
- Анализ распределений признаков по классам целевой переменной
- Матрица корреляций(heatmap) для выявления зависимостей
- Попарные графики (pairplot) для признаков
- Поиск мультиколлинеарности и выделение ключевых признаков
- Группировка и агрегация:
  - По должностям (JobRole):
    - Средний и медианный уровень выгорания

- Средние значения: WorkHoursPerWeek, StressLevel, SatisfactionLevel
- ТОП-5 должностей с наибольшей долей выгорания
- ТОП-5 должностей с наибольшим уровнем стресса
- По полу (Gender):
  - Доля выгорания среди мужчин и женщин
  - Средние характеристики по гендерным группам
- По категориям рабочего времени:
  - Доля выгорания в каждой категории
  - Средний уровень стресса и удовлетворённости
- По категориям удалённой работы:
  - Влияние RemoteRatio на выгорание
  - Сравнение уровня стресса при разных моделях работы
- По стажу:
  - Динамика выгорания в зависимости от Experience
  - Связь стажа с удовлетворённостью
- По возрастным группам:
  - Распределение выгорания по возрастам
  - Анализ уязвимых возрастных групп
- Топ-5 должностей с наивысшим и наизнешним процентом выгоревших сотрудников.
- Пороговый анализ:
  - Выявление паттернов
  - Идея: найти диапазоны, при которых Burnout резко возрастает

#### **Шаг 4: Составление профилей сотрудников**

- Портрет сотрудника с выгоранием:
  - Средние/медианные значения всех характеристик для Burnout=1
  - Наиболее частая должность, пол, возрастная группа
  - Типичный уровень рабочих часов, стресса, удовлетворённости
  - Визуализация: "паспорт сотрудника с выгоранием" (карточка с ключевыми показателями)
- Портрет сотрудника без выгорания:
  - Аналогичный анализ для Burnout=0
  - Сравнительная визуализация двух профилей
- Профили по должностям:
  - Детальная характеристика каждой JobRole
  - Выявление специфики выгорания в разных профессиях
  - Рекомендации для каждой профессиональной группы

#### **Шаг 5: Проверка статистических гипотез**

- H1: существуют значимые различия в уровне выгорания между должностями
- H2: удалённая работа снижает риск выгорания
- H3: сотрудники с рабочими часами более 45 в неделю имеют статистически значимо более высокий риск выгорания.
- H4: существует значимая корреляция между StressLevel и SatisfactionLevel
- H5: средний уровень стресса у сотрудников с выгоранием выше, чем у сотрудников без выгорания
- H6: уровень удовлетворённости у сотрудников с выгоранием ниже
- H7: мужчины и женщины имеют разный уровень выгорания

#### **Шаг 6: Подготовка данных к обучению моделей**

- Выбор признаков для моделей
- Разделение данных на признаки и целевую переменную
- Разделение на train/valid/test, стратификация по Burnout
- Предобработка данных: масштабирование числовых признаков и кодирование категориальных признаков (One-Hot Encoding, Order Encoding, Label Encoding)

#### **Шаг 7: Моделирование и интерпретация**

- Бейзлайн-модель: DummyClassifier и логистическая регрессия с регуляризацией, стандартизацией числовых признаков
- Деревья/ансамбли: Decision Tree / Random Forest /XGBClassifier, LGBMClassifier, CatBoostClassifier для сравнения
- Подбор гиперпараметров на кросс-валидации с помощью GridSearchCV
- Сравнение моделей по метрике на валидации
- Визуализация результатов лучшей модели на валидационной выборке:
  - Предсказание на валидационной выборке
  - Расчёт метрик: ROC-AUC, PR-AUC, F1, Recall класса 1 (важнее не пропустить выгорание, чем иметь ложные тревоги), Precision
  - Матрица ошибок и ROC-кривая
  - Отчёт по классификации: точность, полнота, F1-мера
- Интерпретация:
  - важность признаков (coef/feature\_importances)
  - формулировка бизнес-инсайтов на основе модели: "Какие 3 фактора сильнее всего повышают риск выгорания?"

## Шаг 8: Формирование заключительного отчета и рекомендаций

- Обобщение ключевых находок EDA.
- Представление результатов работы лучшей модели на тестовой выборке.
- Сегментация риска: разнести сотрудников по вероятности выгорания и составить «портрет» каждой группы.
  - Low risk:  $P(\text{Burnout}) < 0.3$
  - Medium risk:  $0.3 \leq P(\text{Burnout}) < 0.7$
  - High risk:  $P(\text{Burnout}) \geq 0.7$
- Аналитические инсайты
- Основные выводы:
  - Какие факторы наиболее сильно влияют на выгорание (по результатам ML и статистики)
  - Профиль сотрудника с высоким риском выгорания
  - Должности и условия работы с максимальным риском
  - Роль удалённой работы в профилактике выгорания
  - Критические пороги для рабочих часов и уровня стресса
- Разработка практических рекомендаций для HR-отдела
  - По профилактике выгорания
  - По удалённой работе
  - По целевым группам риска
  - По мониторингу
- Ограничения исследования
  - перечислить

## Ожидаемые результаты

Таблицы и графики:

- Ранг признаков по важности и зависимости риска от ключевых факторов.
- Матрица ошибок и кривые ROC/PR для выбранной модели.
- Пороговый анализ с рекомендованным рабочим порогом для HR.
- Портреты «персон риска» с наглядными профилями и точками вмешательства.
- Топ-5 рискованных по выгоранию должностей.
- Сравнительные метрики качества моделей.

Инсайты:

- Чёткие пороги нагрузки и сценарии снижения риска.
- Сегменты с наибольшим эффектом от удалёнки или улучшения удовлетворённости.
- План действий для HR: короткий чек-лист вмешательств и мониторинга.

## Содержание:

- Шаг 1: Загрузка данных
- Шаг 2: Предобработка данных
- Шаг 3: Исследовательский анализ данных
- Шаг 4: Составление профилей сотрудников
- Шаг 5: Проверка статистических гипотез
- Шаг 6: Подготовка данных к обучению моделей
- Шаг 7: Моделирование и интерпретация
- Шаг 8: Формирование заключительного отчета и рекомендаций

## Выполнение проекта

### Технический стек

- Проект написан на Python с использованием следующих библиотек:

```
In [1]: # Работа с файлами и API
import os, zipfile, requests, re
from kaggle.api.kaggle_api_extended import KaggleApi
from charset_normalizer import from_path

# Базовые библиотеки анализа данных
import pandas as pd
import numpy as np
import re

# Визуализация
import matplotlib.pyplot as plt
import seaborn as sns
```

```

# 🔍 Корреляционный анализ
import phik
from phik import phik_matrix
from phik.report import plot_correlation_matrix

# 💼 Статистика
from scipy.stats import (
    f_oneway, ttest_ind, pearsonr, spearmanr,
    mannwhitneyu, ks_2samp
)
import scipy.stats as stats
import time

# 🚗 Машинное обучение
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder, OrdinalEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, KFold, RandomizedSearchCV
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

# 📈 Метрики качества
from sklearn.metrics import (
    recall_score, precision_score, f1_score, accuracy_score,
    classification_report, confusion_matrix, roc_curve, auc,
    precision_recall_curve, roc_auc_score, average_precision_score,
    precision_score, recall_score,
    fbeta_score, make_scorer
)

import warnings
warnings.filterwarnings('ignore')

```

## Шаг 1: Загрузка данных

In [2]:

```

# 🔍 Определение кодировки файла
def detect_encoding(file_path):
    result = from_path(file_path).best()
    if result is None:
        print("⚠️ Кодировка не определена, используем utf-8 по умолчанию.")
        return "utf-8"
    print(f"🔍 Определена кодировка: {result.encoding}")
    return result.encoding

# 📁 Загрузка и первичный анализ CSV-файла
def process_dataframe(file_name, data_dir="datasets", sep=",", decimal="."):
    local_path = os.path.join(data_dir, file_name) # путь к локальному файлу
    url_path = f'https://code.net/datasets/{file_name}' # пример URL для загрузки

    if os.path.exists(local_path):
        print("📁 Загружаем локальный файл...")
        encoding = detect_encoding(local_path) # определяем кодировку
        print("📁 Абсолютный путь к файлу:", os.path.abspath(local_path))
        df = pd.read_csv(local_path, sep=sep, decimal=decimal, encoding=encoding)
    else:
        print("🌐 Локальный файл не найден, проверяем URL...")
        response = requests.get(url_path)
        if response.status_code == 200:
            print("✅ Файл найден по URL, загружаем...")
            df = pd.read_csv(url_path, sep=sep, decimal=decimal, encoding="utf-8")
        else:
            print("❌ Ошибка: файл не найден ни локально, ни по URL.")
            return None

    # 📈 Первичный анализ датафрейма
    print("\n🔍 Первые 5 строк датафрейма:")
    display(df.head())
    print("\n📖 Любые 5 строк датафрейма:")
    display(df.sample(5))
    print("\n💻 Последние 5 строк датафрейма:")
    display(df.tail())

    print("\n🗣 Информация о датафрейме:")
    df.info()
    print("\n📐 Размер датафрейма:")
    print(df.shape)
    print("\n👉 Названия столбцов:")
    print(df.columns)

    return df

# 🛡 Скачивание и обработка датасета с Kaggle
def fetch_and_process_kaggle_dataset(dataset_slug, target_csv_name, data_dir="datasets"):
    os.makedirs(data_dir, exist_ok=True) # создаем папку для датасетов, если её нет

    # 🔑 Инициализация Kaggle API
    api = KaggleApi()
    api.authenticate()

```

```

print("⬇ Скачиваем архив с Kaggle...")
api.dataset_download_files(dataset_slug, path=data_dir, unzip=False)

# 🔎 Поиск zip-файла в папке
zip_files = [f for f in os.listdir(data_dir) if f.endswith(".zip")]
if not zip_files:
    print("❌ Архив не найден.")
    return None

zip_path = os.path.join(data_dir, zip_files[0])

print("📦 Распаковываем архив...")
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(data_dir)

# 🔎 Проверка наличия нужного CSV
target_path = os.path.join(data_dir, target_csv_name)
if not os.path.exists(target_path):
    print(f"❌ CSV-файл '{target_csv_name}' не найден после распаковки.")
    return None

# 📈 Загружаем и анализируем CSV
return process_dataframe(target_csv_name, data_dir)

```

```
In [3]: df = fetch_and_process_kaggle_dataset(
    dataset_slug="ankam6010/synthetic-hr-burnout-dataset",
    target_csv_name="synthetic_employee_burnout.csv",
    data_dir=r"C:\Users\HP\my_data\kaggle_datasets\04_Kaggle_employee_burnout"
)
```

⬇ Скачиваем архив с Kaggle...  
Dataset URL: <https://www.kaggle.com/datasets/ankam6010/synthetic-hr-burnout-dataset>  
📦 Распаковываем архив...  
📁 Загружаем локальный файл...  
🔍 Определена кодировка: ascii  
📁 Абсолютный путь к файлу: C:\Users\HP\my\_data\kaggle\_datasets\04\_Kaggle\_employee\_burnout\synthetic\_employee\_burnout.csv

🔍 Первые 5 строк датафрейма:

	Name	Age	Gender	JobRole	Experience	WorkHoursPerWeek	RemoteRatio	SatisfactionLevel	StressLevel	Burnout
0	Max Ivanov	32	Male	Analyst	3	60	21	4.40	1	0
1	Max Wang	40	Female	Engineer	9	47	67	2.09	2	0
2	Nina Petrov	33	Female	Engineer	2	44	20	2.58	3	0
3	John Ivanov	35	Female	Manager	6	44	70	3.23	8	0
4	John Wang	59	Male	Sales	8	38	46	4.41	1	0

🔍 Любые 5 строк датафрейма:

	Name	Age	Gender	JobRole	Experience	WorkHoursPerWeek	RemoteRatio	SatisfactionLevel	StressLevel	Burnout
254	Alex Brown	45	Male	Manager	4	38	80	4.14	7	0
1594	Dina Lee	28	Male	HR	6	57	82	2.04	7	0
549	Ivan Lee	52	Male	HR	12	56	64	1.36	3	0
1440	Sam Chen	59	Male	Engineer	13	47	19	3.98	2	0
569	Leo Kim	50	Male	Engineer	0	64	52	1.17	1	0

✖ Последние 5 строк датафрейма:

	Name	Age	Gender	JobRole	Experience	WorkHoursPerWeek	RemoteRatio	SatisfactionLevel	StressLevel	Burnout
1995	Leo Brown	41	Female	Manager	4	63	17	3.40	4	0
1996	Alex Brown	23	Female	HR	2	39	20	4.67	9	0
1997	Nina Wang	31	Female	HR	10	39	4	4.10	4	0
1998	Kate Lee	25	Male	HR	0	40	57	2.11	4	0
1999	Lily Petrov	49	Female	Engineer	13	65	22	4.36	8	0

Информация о датафрейме:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Name              2000 non-null    object  
 1   Age               2000 non-null    int64  
 2   Gender            2000 non-null    object  
 3   JobRole           2000 non-null    object  
 4   Experience        2000 non-null    int64  
 5   WorkHoursPerWeek 2000 non-null    int64  
 6   RemoteRatio       2000 non-null    int64  
 7   SatisfactionLevel 2000 non-null    float64 
 8   StressLevel       2000 non-null    int64  
 9   Burnout           2000 non-null    int64  
dtypes: float64(1), int64(6), object(3)
memory usage: 156.4+ KB
```

Размер датафрейма:  
(2000, 10)

Названия столбцов:

```
Index(['Name', 'Age', 'Gender', 'JobRole', 'Experience', 'WorkHoursPerWeek',
       'RemoteRatio', 'SatisfactionLevel', 'StressLevel', 'Burnout'],
      dtype='object')
```

Общая характеристика датасета:

- Датасет содержит 2000 записей (сотрудников) и 10 признаков
- Все данные являются полными — отсутствуют пропущенные значения (Non-Null Count = 2000 для всех столбцов)
- Типы данных определены корректно: числовые (int64, float64) и категориальные (object)

Характеристики датасета:

- Числовые признаки: Age, Experience, WorkHoursPerWeek, RemoteRatio, SatisfactionLevel, StressLevel, Burnout
- Категориальные признаки: Name, Gender, JobRole

Сохраним исходный датасет

```
In [4]: # Сохраняем копию для возможного возврата
df_original = df.copy()
```

## Выводы по шагу 1

Загрузка данных:

- Датасет корректно загружен с Kaggle через Kaggle API.
- Архив распакован без ошибок, CSV-файл найден и успешно прочитан.
- Размер датасета: 2000 записей × 10 признаков — достаточно для обучения ML-моделей

Первичный анализ:

- Выведена информация о структуре (df.info()), размере (df.shape) и названиях столбцов
- Все ключевые признаки (Age, Gender, JobRole, Experience, WorkHoursPerWeek, RemoteRatio, SatisfactionLevel, StressLevel, Burnout) присутствуют и читаются корректно
- Типы данных корректны:
  - int64 для возраста, стажа, рабочих часов, удалённости, стресса и целевой переменной
  - float64 для уровня удовлетворённости (непрерывная переменная)
  - object для категориальных признаков (Name, Gender, JobRole)

Содержимое данных:

- По первым/случайным/последним строкам видны реалистичные синтетические значения:
  - Возраст сотрудников от 20+ до 59 лет
  - Есть гендерное разнообразие: Male/Female
  - Широкий спектр должностей: Engineer, Manager, HR, Analyst, Sales
  - Разброс рабочих часов в широком диапазоне: от нормированного ≈ 38 до перегруженного 64
  - Разные уровни удалёнки: от 2% до 98%
  - Уровни удовлетворённости и стресса в ожидаемых диапазонах

Целевая переменная Burnout присутствует в датасете с типом int64, но в первых примерах равна 0 (нет признаков выгорания)

- Бинарная переменная: 0 (нет выгорания) / 1 (есть выгорание)

Колонка Name не несёт аналитической ценности, ее можно удалить

- Признак полностью синтетический, предназначен только для реализма
- В ML-моделях будет удалён

## Шаг 2: Предобработка данных

### Приведение названий столбцов к snake\_case

```
In [5]: def to_snake(name: str) -> str:
    # Вставляем "_" перед заглавной буквой и приводим к нижнему регистру
    return re.sub(r'(?<!^)(?=[A-Z])', '_', name).lower()

# Переименовываем столбцы прямо в существующем df
df.columns = [to_snake(col) for col in df.columns]
```

```
In [6]: df.columns
```

```
Out[6]: Index(['name', 'age', 'gender', 'job_role', 'experience',
               'work_hours_per_week', 'remote_ratio', 'satisfaction_level',
               'stress_level', 'burnout'],
              dtype='object')
```

```
In [7]: df_original.columns
```

```
Out[7]: Index(['Name', 'Age', 'Gender', 'JobRole', 'Experience', 'WorkHoursPerWeek',
               'RemoteRatio', 'SatisfactionLevel', 'StressLevel', 'Burnout'],
              dtype='object')
```

### Уникальные значения категориальных признаков

```
In [8]: temp = df.copy()

list_c = temp.select_dtypes(include=['object']).columns
print(temp[list_c].info())

for col_1 in list_c:
    print('-' * 25)
    print(col_1, temp[col_1].sort_values().unique())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
---  --          --          --      
 0   name        2000 non-null   object 
 1   gender       2000 non-null   object 
 2   job_role     2000 non-null   object 
dtypes: object(3)
memory usage: 47.0+ KB
None
-----
name ['Alex Brown' 'Alex Chen' 'Alex Garcia' 'Alex Ivanov' 'Alex Johnson'
      'Alex Kim' 'Alex Lee' 'Alex Petrov' 'Alex Smith' 'Alex Wang' 'Dina Brown'
      'Dina Chen' 'Dina Garcia' 'Dina Ivanov' 'Dina Johnson' 'Dina Kim'
      'Dina Lee' 'Dina Petrov' 'Dina Smith' 'Dina Wang' 'Ivan Brown'
      'Ivan Chen' 'Ivan Garcia' 'Ivan Ivanov' 'Ivan Johnson' 'Ivan Kim'
      'Ivan Lee' 'Ivan Petrov' 'Ivan Smith' 'Ivan Wang' 'John Brown'
      'John Chen' 'John Garcia' 'John Ivanov' 'John Johnson' 'John Kim'
      'John Lee' 'John Petrov' 'John Smith' 'John Wang' 'Kate Brown'
      'Kate Chen' 'Kate Garcia' 'Kate Ivanov' 'Kate Johnson' 'Kate Kim'
      'Kate Lee' 'Kate Petrov' 'Kate Smith' 'Kate Wang' 'Leo Brown' 'Leo Chen'
      'Leo Garcia' 'Leo Ivanov' 'Leo Johnson' 'Leo Kim' 'Leo Lee' 'Leo Petrov'
      'Leo Smith' 'Leo Wang' 'Lily Brown' 'Lily Chen' 'Lily Garcia'
      'Lily Ivanov' 'Lily Johnson' 'Lily Kim' 'Lily Lee' 'Lily Petrov'
      'Lily Smith' 'Lily Wang' 'Max Brown' 'Max Chen' 'Max Garcia' 'Max Ivanov'
      'Max Johnson' 'Max Kim' 'Max Lee' 'Max Petrov' 'Max Smith' 'Max Wang'
      'Nina Brown' 'Nina Chen' 'Nina Garcia' 'Nina Ivanov' 'Nina Johnson'
      'Nina Kim' 'Nina Lee' 'Nina Petrov' 'Nina Smith' 'Nina Wang' 'Sam Brown'
      'Sam Chen' 'Sam Garcia' 'Sam Ivanov' 'Sam Johnson' 'Sam Kim' 'Sam Lee'
      'Sam Petrov' 'Sam Smith' 'Sam Wang']
-----
gender ['Female' 'Male']
-----
job_role ['Analyst' 'Engineer' 'HR' 'Manager' 'Sales']
```

### Проверка и обработка пропусков

```
In [9]: def analyze_missing_values(df):
    temp = df.copy() # Создаем копию входного DataFrame
    missing = pd.DataFrame({
        'Кол-во пропусков': temp.isnull().sum(),
        'Доля пропусков': temp.isnull().mean().round(4)
    })
    # Сортируем столбцы по количеству пропусков (по убыванию)
    missing = missing.sort_values(by='Кол-во пропусков', ascending=False)

    # Визуализация пропусков
    return missing.style.background_gradient(cmap='coolwarm')
```

```
In [10]: analyze_missing_values(df)
```

Out[10]:

	Кол-во пропусков	Доля пропусков
<b>name</b>	0	0.000000
<b>age</b>	0	0.000000
<b>gender</b>	0	0.000000
<b>job_role</b>	0	0.000000
<b>experience</b>	0	0.000000
<b>work_hours_per_week</b>	0	0.000000
<b>remote_ratio</b>	0	0.000000
<b>satisfaction_level</b>	0	0.000000
<b>stress_level</b>	0	0.000000
<b>burnout</b>	0	0.000000

## Проверка на дубликаты

```
In [11]: print("Явные дубликаты в df:", df.duplicated().sum())
```

Явные дубликаты в df: 0

```
In [12]: # Проверка неявных дубликатов в train  
dup_df = df.drop(columns=['name']).duplicated().sum()  
print(f"Неявные дубликаты в df: {dup_df}")
```

Неявные дубликаты в df: 0

## Создание новых признаков (Feature Engineering)

```
In [13]: df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
<b>age</b>	2000.0	40.69450	11.286756	22.0	31.0	41.000	50.0	60.0
<b>experience</b>	2000.0	10.07450	9.148267	0.0	3.0	7.000	15.0	39.0
<b>work_hours_per_week</b>	2000.0	49.58800	11.832424	30.0	39.0	49.000	60.0	70.0
<b>remote_ratio</b>	2000.0	49.97300	29.151298	0.0	24.0	49.000	75.0	100.0
<b>satisfaction_level</b>	2000.0	2.99523	1.155431	1.0	2.0	3.025	4.0	5.0
<b>stress_level</b>	2000.0	5.43200	2.880890	1.0	3.0	5.000	8.0	10.0
<b>burnout</b>	2000.0	0.06450	0.245703	0.0	0.0	0.000	0.0	1.0

### Категории возраста (age\_bin)

- min = 22, max = 60, медиана ≈ 41.
- Разумные группы:
  - 22–30 (молодые сотрудники)
  - 31–40 (средний возраст)
  - 41–50 (старший возраст)
  - 51–60 (перед пенсией)

### Категории стажа (experience\_cat)

- min = 0, медиана = 7, 75% = 15, max = 39.
- Группы:
  - 0–3 → Junior
  - 4–7 → Middle
  - 8–15 → Senior
  - 16+ → Expert

### Категории рабочих часов (work\_hours\_bin)

- min = 30, медиана = 49, 75% = 60, max = 70.
- Группы:
  - 30–40 → Нормальная нагрузка
  - 41–50 → Повышенная
  - 51–60 → Высокая
  - 61–70 → Перегрузка

### Категории удалёнки (remote\_bin)

- min = 0, медиана = 49, 75% = 75, max = 100.
- Группы:

- 0 → Полностью офис
- 1–25 → Низкая удалёнка
- 26–50 → Средняя
- 51–75 → Высокая
- 76–100 → Полностью удалёнка

### Категории удовлетворённости (satisfaction\_cat)

- min = 1, медиана ≈ 3, 75% = 4, max = 5.
- Группы:
  - 1–2 → Низкая
  - 2–3.5 → Средняя
  - 3.6–5 → Высокая

### Категории стресса (stress\_cat)

- min = 1, медиана = 5, 75% = 8, max = 10.
- Группы:
  - 1–3 → Низкий
  - 4–6 → Средний
  - 7–10 → Высокий

```
In [14]: # Категории возраста
df['age_bin'] = pd.cut(df['age'], bins=[21,30,40,50,60],
                       labels=[
                           'Young (22-30)',
                           'Mid-Age (31-40)',
                           'Senior (41-50)',
                           'Pre-Retirement (51-60)'
                       ])
)

# Категории стажа
df['experience_cat'] = pd.cut(df['experience'], bins=[-1,3,7,15,40], labels=['junior','middle','senior','expert'])

# Категории нагрузки
df['work_hours_bin'] = pd.cut(df['work_hours_per_week'], bins=[29,40,50,60,70],
                               labels=[
                                   'Normal (30-40h)',
                                   'Elevated (41-50h)',
                                   'High (51-60h)',
                                   'Critical (61-70h)'
                               ])
)

# Категории удалённой работы
df['remote_bin'] = pd.cut(df['remote_ratio'], bins=[-1,0,25,50,75,100], labels=['office','low','medium','high','remote'])
df['remote_cat'] = pd.cut(df['remote_ratio'], bins=[-1, 0, 80, 100], labels=['onsite', 'hybrid', 'remote'])

# Категории стресса
df['stress_cat'] = pd.cut(df['stress_level'], bins=[0,3,6,10], labels=['low', 'medium', 'high'])

# Категории удовлетворённости
df['satisfaction_cat'] = pd.cut(df['satisfaction_level'], bins=[0,2,3.5,5], labels=['low', 'medium', 'high'])
```

```
In [15]: ## Производные признаки

# Отношение стресса к удовлетворённости
df['stress_to_satisfaction'] = (df['stress_level'] / df['satisfaction_level']).round(3)

# Отношение стажа к возрасту
df['experience_to_age'] = (df['experience'] / df['age']).round(3)

# Возраст начала карьеры
df['career_start_age'] = df['age'] - df['experience']

# Индекс перегрузки
df['overload_index'] = (df['work_hours_per_week'] / 40) * df['stress_level']
```

```
In [16]: # Для гипотезы H3: сотрудники с часами > 45
df['hours_above_45'] = (df['work_hours_per_week'] > 45).astype(int)
```

```
In [17]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   name             2000 non-null    object  
 1   age              2000 non-null    int64  
 2   gender            2000 non-null    object  
 3   job_role          2000 non-null    object  
 4   experience        2000 non-null    int64  
 5   work_hours_per_week  2000 non-null    int64  
 6   remote_ratio      2000 non-null    int64  
 7   satisfaction_level  2000 non-null    float64 
 8   stress_level      2000 non-null    int64  
 9   burnout           2000 non-null    int64  
 10  age_bin           2000 non-null    category 
 11  experience_cat   2000 non-null    category 
 12  work_hours_bin   2000 non-null    category 
 13  remote_bin        2000 non-null    category 
 14  remote_cat        2000 non-null    category 
 15  stress_cat        2000 non-null    category 
 16  satisfaction_cat  2000 non-null    category 
 17  stress_to_satisfaction  2000 non-null    float64 
 18  experience_to_age  2000 non-null    float64 
 19  career_start_age  2000 non-null    int64  
 20  overload_index    2000 non-null    float64 
 21  hours_above_45    2000 non-null    int32  
dtypes: category(7), float64(4), int32(1), int64(7), object(3)
memory usage: 241.6+ KB
```

```
In [18]: temp = df.copy()

list_c = temp.select_dtypes(include=['object','category']).columns
print(temp[list_c].info())

for col_l in list_c:
    print('-' * 25)
    print(col_l, temp[col_l].sort_values().unique())
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   name        2000 non-null   object  
 1   gender       2000 non-null   object  
 2   job_role     2000 non-null   object  
 3   age_bin      2000 non-null   category
 4   experience_cat 2000 non-null   category
 5   work_hours_bin 2000 non-null   category
 6   remote_bin    2000 non-null   category
 7   remote_cat    2000 non-null   category
 8   stress_cat    2000 non-null   category
 9   satisfaction_cat 2000 non-null   category
dtypes: category(7), object(3)
memory usage: 61.9+ KB
None

-----
name ['Alex Brown' 'Alex Chen' 'Alex Garcia' 'Alex Ivanov' 'Alex Johnson'
 'Alex Kim' 'Alex Lee' 'Alex Petrov' 'Alex Smith' 'Alex Wang' 'Dina Brown'
 'Dina Chen' 'Dina Garcia' 'Dina Ivanov' 'Dina Johnson' 'Dina Kim'
 'Dina Lee' 'Dina Petrov' 'Dina Smith' 'Dina Wang' 'Ivan Brown'
 'Ivan Chen' 'Ivan Garcia' 'Ivan Ivanov' 'Ivan Johnson' 'Ivan Kim'
 'Ivan Lee' 'Ivan Petrov' 'Ivan Smith' 'Ivan Wang' 'John Brown'
 'John Chen' 'John Garcia' 'John Ivanov' 'John Johnson' 'John Kim'
 'John Lee' 'John Petrov' 'John Smith' 'John Wang' 'Kate Brown'
 'Kate Chen' 'Kate Garcia' 'Kate Ivanov' 'Kate Johnson' 'Kate Kim'
 'Kate Lee' 'Kate Petrov' 'Kate Smith' 'Kate Wang' 'Leo Brown' 'Leo Chen'
 'Leo Garcia' 'Leo Ivanov' 'Leo Johnson' 'Leo Kim' 'Leo Lee' 'Leo Petrov'
 'Leo Smith' 'Leo Wang' 'Lily Brown' 'Lily Chen' 'Lily Garcia'
 'Lily Ivanov' 'Lily Johnson' 'Lily Kim' 'Lily Lee' 'Lily Petrov'
 'Lily Smith' 'Lily Wang' 'Max Brown' 'Max Chen' 'Max Garcia' 'Max Ivanov'
 'Max Johnson' 'Max Kim' 'Max Lee' 'Max Petrov' 'Max Smith' 'Max Wang'
 'Nina Brown' 'Nina Chen' 'Nina Garcia' 'Nina Ivanov' 'Nina Johnson'
 'Nina Kim' 'Nina Lee' 'Nina Petrov' 'Nina Smith' 'Nina Wang' 'Sam Brown'
 'Sam Chen' 'Sam Garcia' 'Sam Ivanov' 'Sam Johnson' 'Sam Kim' 'Sam Lee'
 'Sam Petrov' 'Sam Smith' 'Sam Wang']

-----
gender ['Female' 'Male']

-----
job_role ['Analyst' 'Engineer' 'HR' 'Manager' 'Sales']

-----
age_bin ['Young (22-30)', 'Mid-Age (31-40)', 'Senior (41-50)', 'Pre-Retirement (51-60)']
Categories (4, object): ['Young (22-30)' < 'Mid-Age (31-40)' < 'Senior (41-50)' < 'Pre-Retirement (51-60)']

-----
experience_cat ['junior', 'middle', 'senior', 'expert']
Categories (4, object): ['junior' < 'middle' < 'senior' < 'expert']

-----
work_hours_bin ['Normal (30-40h)', 'Elevated (41-50h)', 'High (51-60h)', 'Critical (61-70h)']
Categories (4, object): ['Normal (30-40h)' < 'Elevated (41-50h)' < 'High (51-60h)' < 'Critical (61-70h)']

-----
remote_bin ['office', 'low', 'medium', 'high', 'remote']
Categories (5, object): ['office' < 'low' < 'medium' < 'high' < 'remote']

-----
remote_cat ['onsite', 'hybrid', 'remote']
Categories (3, object): ['onsite' < 'hybrid' < 'remote']

-----
stress_cat ['low', 'medium', 'high']
Categories (3, object): ['low' < 'medium' < 'high']

-----
satisfaction_cat ['low', 'medium', 'high']
Categories (3, object): ['low' < 'medium' < 'high']

```

In [19]: `df.describe().T`

	count	mean	std	min	25%	50%	75%	max
<b>age</b>	2000.0	40.694500	11.286756	22.000	31.000	41.0000	50.000000	60.00
<b>experience</b>	2000.0	10.074500	9.148267	0.000	3.000	7.0000	15.000000	39.00
<b>work_hours_per_week</b>	2000.0	49.588000	11.832424	30.000	39.000	49.0000	60.000000	70.00
<b>remote_ratio</b>	2000.0	49.973000	29.151298	0.000	24.000	49.0000	75.000000	100.00
<b>satisfaction_level</b>	2000.0	2.995230	1.155431	1.000	2.000	3.0250	4.000000	5.00
<b>stress_level</b>	2000.0	5.432000	2.880890	1.000	3.000	5.0000	8.000000	10.00
<b>burnout</b>	2000.0	0.064500	0.245703	0.000	0.000	0.0000	0.000000	1.00
<b>stress_to_satisfaction</b>	2000.0	2.172195	1.642699	0.201	0.971	1.8125	2.85700	10.00
<b>experience_to_age</b>	2000.0	0.223875	0.170195	0.000	0.077	0.1920	0.34425	0.65
<b>career_start_age</b>	2000.0	30.620000	8.879528	21.000	23.000	28.0000	36.000000	60.00
<b>overload_index</b>	2000.0	6.729925	4.027064	0.750	3.300	6.3000	9.600000	17.50
<b>hours_above_45</b>	2000.0	0.595500	0.490918	0.000	0.000	1.0000	1.000000	1.00

In [20]: `def move_column_to_end(df, col_name):  
 df = df[[c for c in df.columns if c != col_name] + [col_name]]`

```
return df
```

```
In [21]: df = move_column_to_end(df, "burnout")
```

```
In [22]: df.iloc[:, list(range(9)) + [-1]]
```

```
Out[22]:
```

	name	age	gender	job_role	experience	work_hours_per_week	remote_ratio	satisfaction_level	stress_level	burnout
0	Max Ivanov	32	Male	Analyst	3	60	21	4.40	1	0
1	Max Wang	40	Female	Engineer	9	47	67	2.09	2	0
2	Nina Petrov	33	Female	Engineer	2	44	20	2.58	3	0
3	John Ivanov	35	Female	Manager	6	44	70	3.23	8	0
4	John Wang	59	Male	Sales	8	38	46	4.41	1	0
...	...	...	...	...	...	...	...	...	...	...
1995	Leo Brown	41	Female	Manager	4	63	17	3.40	4	0
1996	Alex Brown	23	Female	HR	2	39	20	4.67	9	0
1997	Nina Wang	31	Female	HR	10	39	4	4.10	4	0
1998	Kate Lee	25	Male	HR	0	40	57	2.11	4	0
1999	Lily Petrov	49	Female	Engineer	13	65	22	4.36	8	0

2000 rows × 10 columns

```
In [23]: df.iloc[:, list(range(9, 16))+ [-1]]
```

```
Out[23]:
```

	age_bin	experience_cat	work_hours_bin	remote_bin	remote_cat	stress_cat	satisfaction_cat	burnout
0	Mid-Age (31-40)	junior	High (51-60h)	low	hybrid	low	high	0
1	Mid-Age (31-40)	senior	Elevated (41-50h)	high	hybrid	low	medium	0
2	Mid-Age (31-40)	junior	Elevated (41-50h)	low	hybrid	low	medium	0
3	Mid-Age (31-40)	middle	Elevated (41-50h)	high	hybrid	high	medium	0
4	Pre-Retirement (51-60)	senior	Normal (30-40h)	medium	hybrid	low	high	0
...	...	...	...	...	...	...	...	...
1995	Senior (41-50)	middle	Critical (61-70h)	low	hybrid	medium	medium	0
1996	Young (22-30)	junior	Normal (30-40h)	low	hybrid	high	high	0
1997	Mid-Age (31-40)	senior	Normal (30-40h)	low	hybrid	medium	high	0
1998	Young (22-30)	junior	Normal (30-40h)	high	hybrid	medium	medium	0
1999	Senior (41-50)	senior	Critical (61-70h)	low	hybrid	high	high	0

2000 rows × 8 columns

```
In [24]: df.iloc[:, list(range(16, df.shape[1]))]
```

```
Out[24]:
```

	stress_to_satisfaction	experience_to_age	career_start_age	overload_index	hours_above_45	burnout
0	0.227	0.094	29	1.500	1	0
1	0.957	0.225	31	2.350	1	0
2	1.163	0.061	31	3.300	0	0
3	2.477	0.171	29	8.800	0	0
4	0.227	0.136	51	0.950	0	0
...	...	...	...	...	...	...
1995	1.176	0.098	37	6.300	1	0
1996	1.927	0.087	21	8.775	0	0
1997	0.976	0.323	21	3.900	0	0
1998	1.896	0.000	25	4.000	0	0
1999	1.835	0.265	36	13.000	1	0

2000 rows × 6 columns

- Удалить неинформативный признак:

```
In [25]: df = df.drop(columns=['name'])
```

## Выводы по шагу 2

#### Приведение названий столбцов

- Все названия колонок успешно переведены в формат snake\_case
- Теперь они единообразны: name, age, gender, job\_role, experience, work\_hours\_per\_week, remote\_ratio, satisfaction\_level, stress\_level, burnout

#### Проверка категориальных признаков

- Категориальные признаки (gender, job\_role) содержат корректные значения:
  - gender: только Male и Female
  - job\_role: 5 категорий — Analyst, Engineer, HR, Manager, Sales
  - Артефактов и лишних категорий не обнаружено
  - Признак name используется только для реалистичности, в моделях не нужен, стоит удалить

#### Пропуски

- Проверка показала отсутствие пропусков во всех столбцах (0 пропусков)

#### Дубликаты

- Явных и неявных дубликатов в данных нет (0)
- Датасет чистый и уникальный

#### Категориальные признаки

- Gender: корректные значения (Male/Female)
- Job\_role: 5 категорий (Analyst, Engineer, HR, Manager, Sales)
- Name: признак удален как неинформативны

#### Описательная статистика

- age: диапазон 22–60 лет, медиана ≈ 41, среднее 40.69
- experience: диапазон 0–39 лет, медиана ≈ 7, 75% = 15.
- work\_hours\_per\_week: диапазон 30–70, медиана и среднее ≈ 49.
- remote\_ratio: диапазон 0–100, медиана и среднее ≈ 49.
- satisfaction\_level: диапазон 1–5, медиана и среднее ≈ 3.
- stress\_level: диапазон 1–10, медиана ≈ 5.

#### Созданные новые признаки

- Категориальные признаки
  - Age\_bin: 4 группы (22-30, 31-40, 41-50, 51-60)
  - Experience\_cat: 4 уровня (junior, middle, senior, expert)
  - Work\_hours\_bin: 4 категории (30-40, 41-50, 51-60, 61-70)
  - Remote\_bin: 5 групп (office, low, medium, high, remote)
  - Stress\_cat: 3 уровня (low, medium, high)
  - Satisfaction\_cat: 3 уровня (low, medium, high)
- Производные признаки
  - Stress\_to\_satisfaction: отношение стресса к удовлетворенности
  - Experience\_to\_age: отношение стажа к возрасту
  - Career\_start\_age: возраст начала карьеры
  - Overload\_index: индекс перегрузки
  - Hours\_above\_45: индикатор превышения 45 часов

[\\* к содержанию](#)

## Шаг 3: Исследовательский анализ данных (EDA)

### Анализ целевой переменной burnout

In [26]:

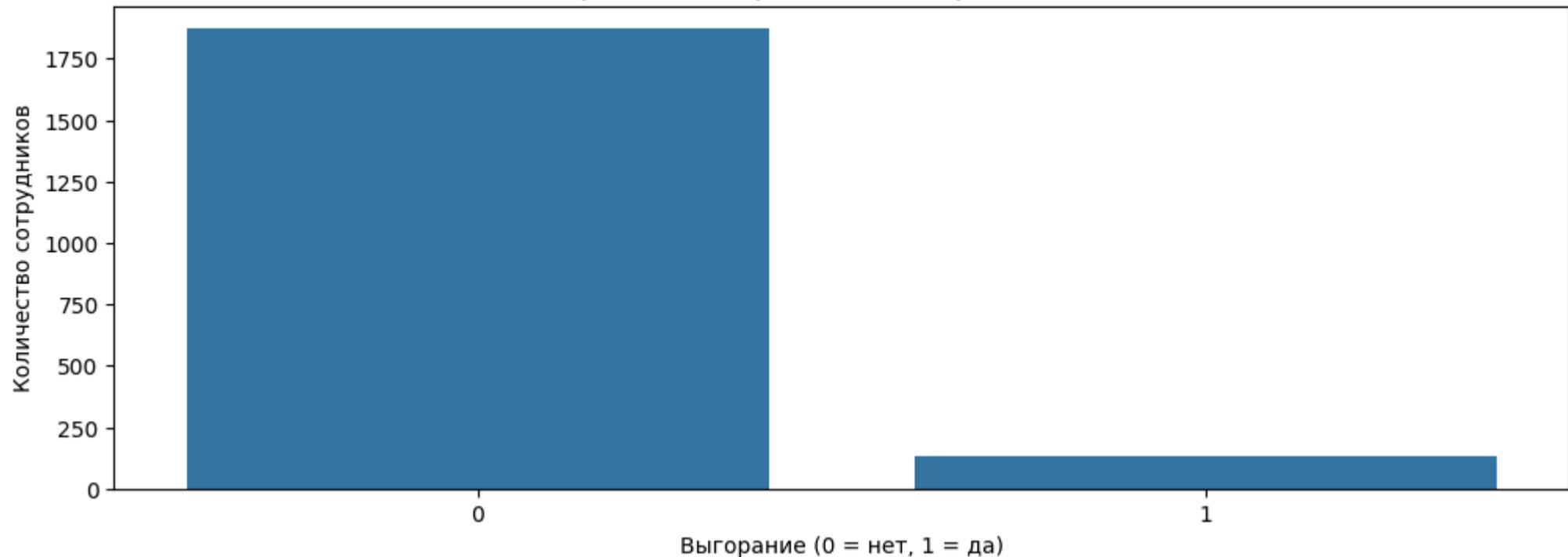
```
# Визуализация распределения целевой переменной Burnout
plt.figure(figsize=(12, 4))
sns.countplot(data=df, x='burnout')
plt.title('Распределение признаков выгорания (burnout)')
plt.xlabel('Выгорание (0 = нет, 1 = да)')
plt.ylabel('Количество сотрудников')
plt.show()

# Абсолютные значения
burnout_counts = df['burnout'].value_counts()
print("Абсолютные значения по целевой переменной Burnout:")
print(burnout_counts)

# Доли (нормализованные значения)
burnout_rate = df['burnout'].value_counts(normalize=True)
```

```
print("\nДоли по целевой переменной Burnout:")
print(burnout_rate)
```

### Распределение признаков выгорания (burnout)



Абсолютные значения по целевой переменной Burnout:

```
0    1871
1     129
Name: burnout, dtype: int64
```

Доли по целевой переменной Burnout:

```
0    0.9355
1    0.0645
Name: burnout, dtype: float64
```

- Выявлен сильный дисбаланс классов: 93.55% без выгорания vs 6.45% с выгоранием
  - Необходимо использовать стратификацию при разбиении данных
  - Выбрать метрики, устойчивые к дисбалансу (Recall, PR-AUC, F1-score)

Соотношение 14.5:1 (на каждого "выгоревшего" приходится 14.5 "здоровых" сотрудников)

- Это сильный дисбаланс (severe class imbalance), характерный для медицинских и HR-задач

## Признаки для анализа

Для каждого типа признака построим графики

- Качественные признаки датасета df:

```
In [27]: # Качественные переменные (обычно числовые типы)
numerical = df.select_dtypes(include=['number']).columns.tolist()

print("Качественные переменные в датасета df:", numerical)

Качественные переменные в датасета df: ['age', 'experience', 'work_hours_per_week', 'remote_ratio', 'satisfaction_level', 'stress_level', 'stress_to_satisfaction', 'experience_to_age', 'career_start_age', 'overload_index', 'hours_above_45', 'burnout']
```

- Категориальные признаки датасета df:

```
In [28]: # Категориальные переменные (обычно тип object или category)
categorical = df.select_dtypes(include=['object', 'category']).columns.tolist()

print("Категориальные переменные в датасета df:", categorical)

Категориальные переменные в датасета df: ['gender', 'job_role', 'age_bin', 'experience_cat', 'work_hours_bin', 'remote_bin', 'remote_cat', 'stress_cat', 'satisfaction_cat']
```

## Распределение количественных признаков

```
In [29]: df.describe().T
```

Out[29]:

		count	mean	std	min	25%	50%	75%	max
	<b>age</b>	2000.0	40.694500	11.286756	22.000	31.000	41.0000	50.00000	60.00
	<b>experience</b>	2000.0	10.074500	9.148267	0.000	3.000	7.0000	15.00000	39.00
	<b>work_hours_per_week</b>	2000.0	49.588000	11.832424	30.000	39.000	49.0000	60.00000	70.00
	<b>remote_ratio</b>	2000.0	49.973000	29.151298	0.000	24.000	49.0000	75.00000	100.00
	<b>satisfaction_level</b>	2000.0	2.995230	1.155431	1.000	2.000	3.0250	4.00000	5.00
	<b>stress_level</b>	2000.0	5.432000	2.880890	1.000	3.000	5.0000	8.00000	10.00
	<b>stress_to_satisfaction</b>	2000.0	2.172195	1.642699	0.201	0.971	1.8125	2.85700	10.00
	<b>experience_to_age</b>	2000.0	0.223875	0.170195	0.000	0.077	0.1920	0.34425	0.65
	<b>career_start_age</b>	2000.0	30.620000	8.879528	21.000	23.000	28.0000	36.00000	60.00
	<b>overload_index</b>	2000.0	6.729925	4.027064	0.750	3.300	6.3000	9.60000	17.50
	<b>hours_above_45</b>	2000.0	0.595500	0.490918	0.000	0.000	1.0000	1.00000	1.00
	<b>burnout</b>	2000.0	0.064500	0.245703	0.000	0.000	0.0000	0.00000	1.00

In [30]: `def compare_box_and_hist(df, feature='age', title=None, bins=30):`

```
# Удаляем пропуски
data = df[feature].dropna()
```

```
# Общая статистика
```

```
print(f"\n📊 Статистика по признаку '{feature}':")
print(data.describe())
```

```
# Заголовок
```

```
if title is None:
    title = f'Распределение признака "{feature}"'
```

```
# Два графика рядом
```

```
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
fig.suptitle(title, fontsize=20, fontweight='bold')
```

```
# Boxplot
```

```
sns.boxplot(y=data, ax=axes[0], color='skyblue')
axes[0].set_title('Boxplot', fontsize=18)
axes[0].set_ylabel(feature, fontsize=16)
axes[0].tick_params(axis='y', labelsize=14)
```

```
# Гистограмма (частота)
```

```
sns.histplot(data, bins=bins, stat='count', color='steelblue',
             ax=axes[1], alpha=0.6, edgecolor='gray')
axes[1].set_title('Гистограмма (частота)', fontsize=18)
axes[1].set_xlabel(feature, fontsize=16)
axes[1].set_ylabel('Частота', fontsize=16)
axes[1].tick_params(axis='x', labelsize=14)
axes[1].tick_params(axis='y', labelsize=14)
```

```
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

age

- **age** - возраст сотрудника

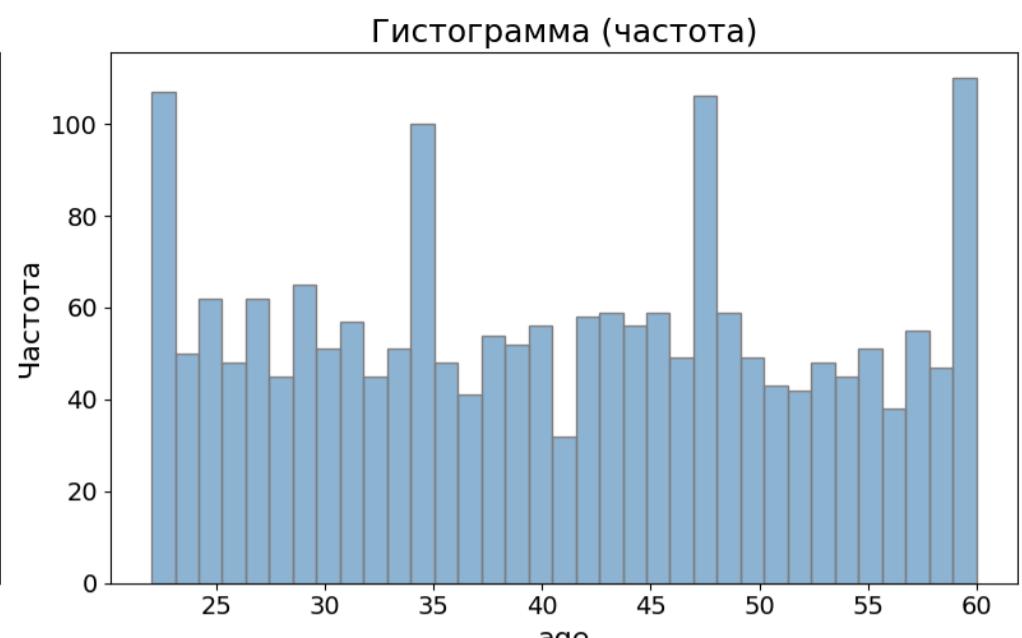
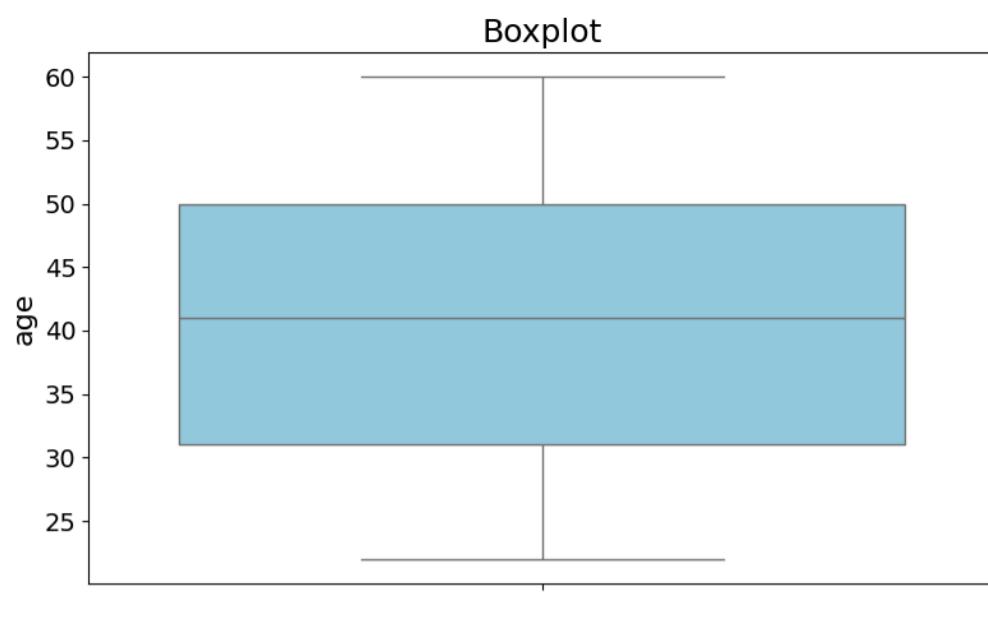
In [31]: `compare_box_and_hist(df, feature='age', title="Распределение возраста сотрудников", bins=35)`

```
📊 Статистика по признаку 'age':
```

count	2000.000000
mean	40.694500
std	11.286756
min	22.000000
25%	31.000000
50%	41.000000
75%	50.000000
max	60.000000

Name: age, dtype: float64

## Распределение возраста сотрудников



```
In [32]: df['age'].value_counts().head()
```

```
Out[32]: 29    65  
25    62  
27    62  
23    61  
59    61  
Name: age, dtype: int64
```

Интерпретация:

- Равномерное распределение с пиками в 29, 25, 27, 23, 59 лет
- Нет выраженных выбросов
- Основная масса сотрудников — 30–50 лет.
- Бизнес-контекст: разнообразный возрастной состав, нет доминирования молодых или опытных

### experience

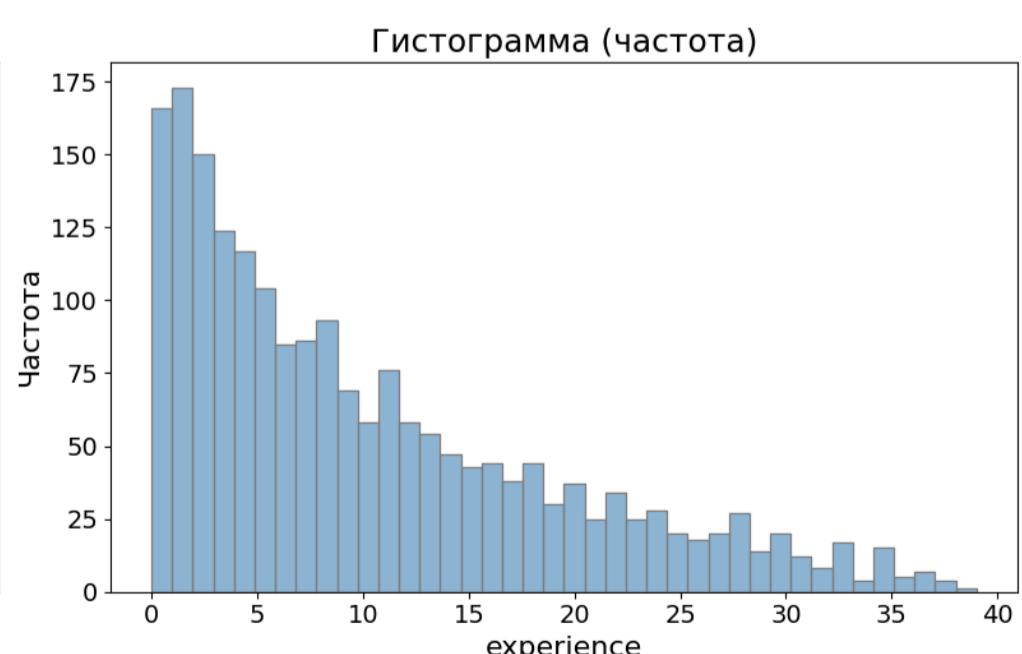
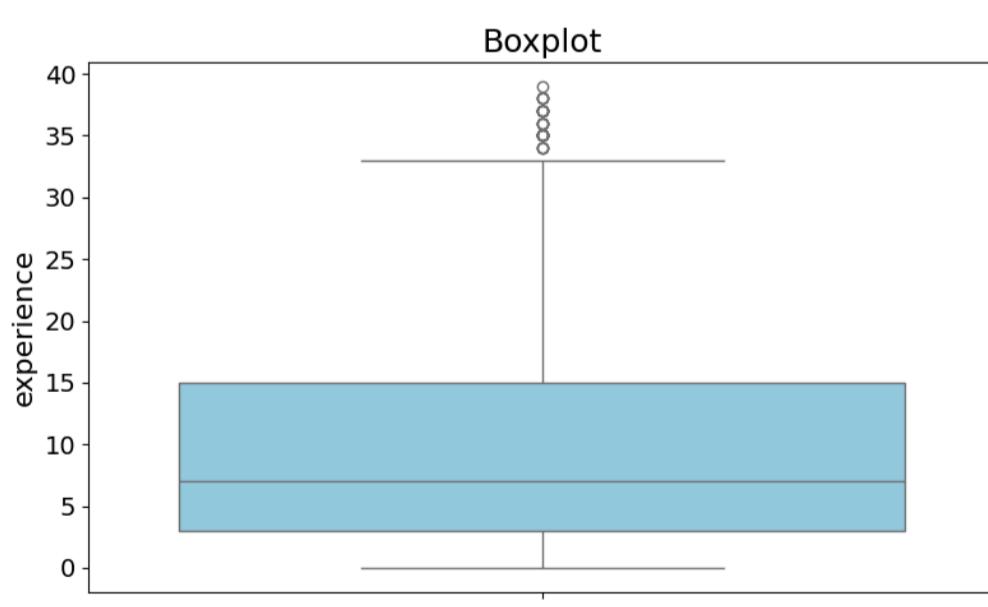
- **experience** - стаж работы сотрудника (в годах)

```
In [33]: compare_box_and_hist(df, feature='experience',  
                           title="Распределение стажа работы сотрудников в годах", bins=40)
```

📊 Статистика по признаку 'experience':

```
count    2000.000000  
mean     10.074500  
std      9.148267  
min      0.000000  
25%     3.000000  
50%     7.000000  
75%    15.000000  
max     39.000000  
Name: experience, dtype: float64
```

## Распределение стажа работы сотрудников в годах



```
In [34]: df['experience'].value_counts().head()
```

```
Out[34]: 1    173  
0    166  
2    150  
3    124  
4    117  
Name: experience, dtype: int64
```

Критические наблюдения:

- Сильная правосторонняя асимметрия:
  - Медиана (7) значительно меньше среднего (10.07)
  - Большой разброс ( $std=9.15$ )
- Выбросы: 36 сотрудников (1.8%) имеют стаж 34-39 лет
- Большинство сотрудников имеют стаж до 15 лет.

HR-инсайт: В компании много новых сотрудников (до 3 лет стажа), что может указывать на высокую текучесть или активный найм.

### work\_hours\_per\_week

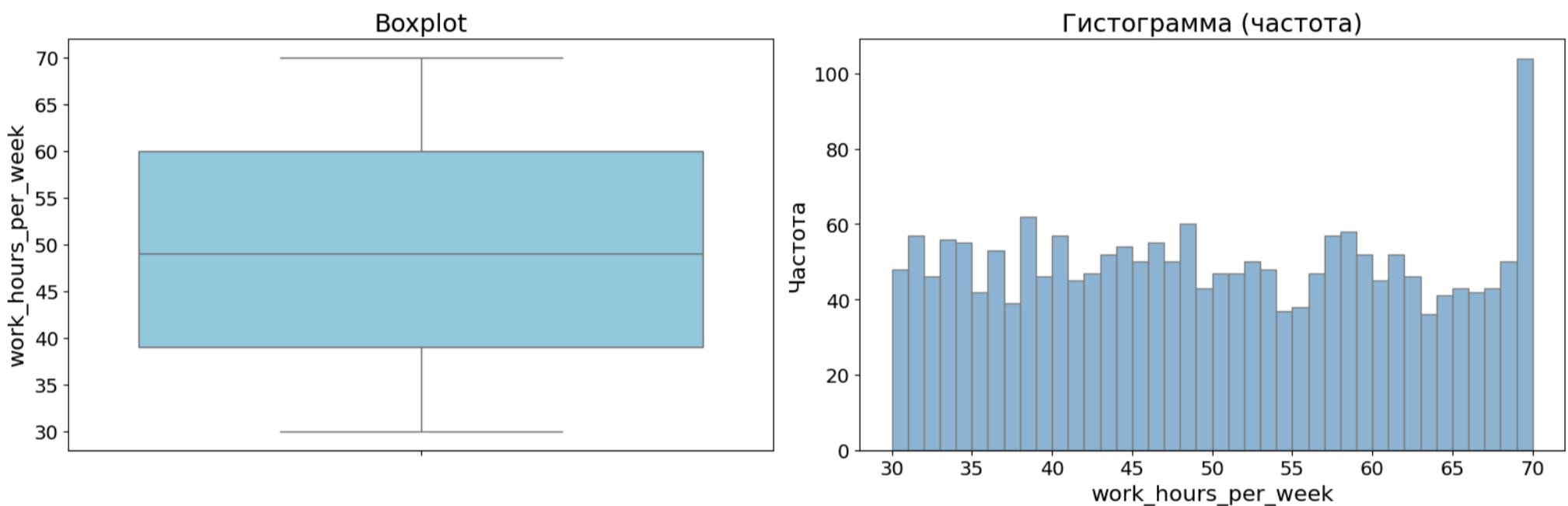
- **work\_hours\_per\_week** - среднее количество рабочих часов в неделю

```
In [35]: compare_box_and_hist(df, feature='work_hours_per_week',
                           title="Распределение среднего количества рабочих часов в неделю сотрудников", bins=40)
```

Статистика по признаку 'work\_hours\_per\_week':

count	2000.000000
mean	49.588000
std	11.832424
min	30.000000
25%	39.000000
50%	49.000000
75%	60.000000
max	70.000000
Name:	work_hours_per_week, dtype: float64

### Распределение среднего количества рабочих часов в неделю сотрудников



```
In [36]: df['work_hours_per_week'].value_counts().head(10)
```

```
Out[36]: 38    62
48    60
58    58
40    57
31    57
57    57
33    56
69    56
46    55
34    55
Name: work_hours_per_week, dtype: int64
```

ТРЕВОЖНЫЕ СИГНАЛЫ:

- Средняя неделя 49.6 часов - на 9.6 часов больше стандартных 40 часов
- 75% сотрудников работают до 60 часов в неделю
- 25% сотрудников работают 39 часов или меньше

Нормативно-правовой контекст:

- В большинстве стран норма - 40 часов
- Работа >48 часов требует особого учета и доплат
- 59.55% сотрудников (hours\_above\_45=1) работают >45 часов

HR-риски:

- Юридические риски (нарушение трудового законодательства)
- Риски для здоровья сотрудников
- Снижение продуктивности при переработках

### remote\_ratio

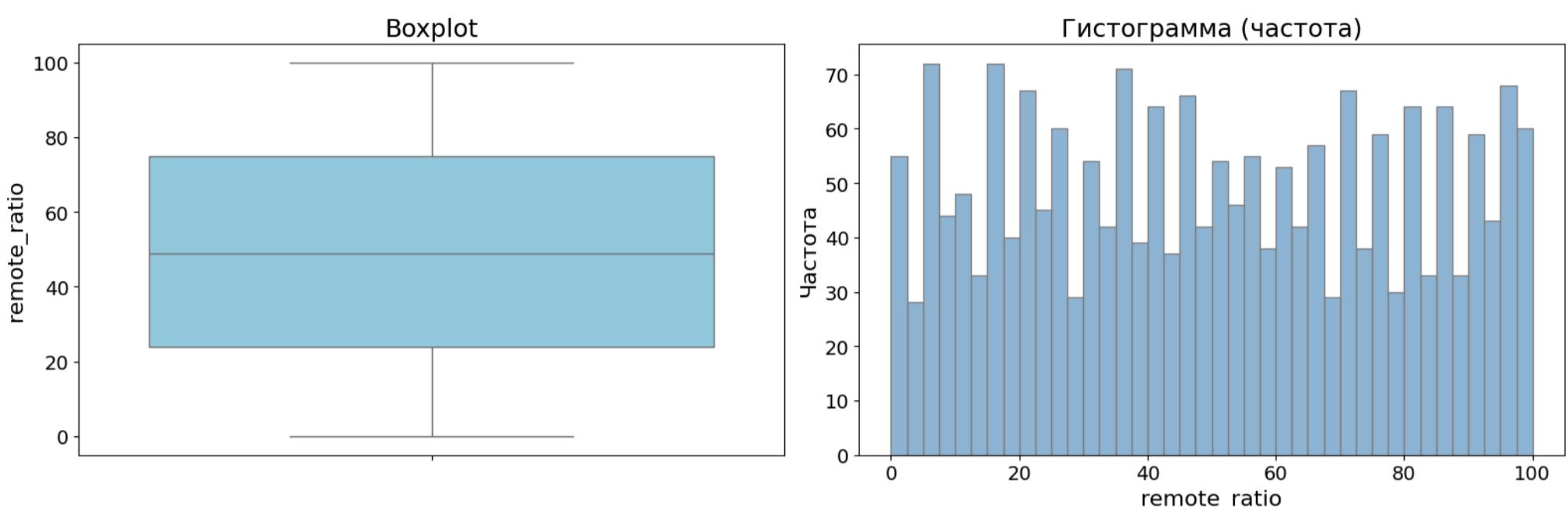
- **remote\_ratio** - процент времени, проведённого на удалённой работе

```
In [37]: compare_box_and_hist(df, feature='remote_ratio',
                           title="Распределение процента времени, проведённого на удалённой работе сотрудников", bins=40)
```

Статистика по признаку 'remote\_ratio':

```
count    2000.000000
mean     49.973000
std      29.151298
min      0.000000
25%     24.000000
50%     49.000000
75%     75.000000
max     100.000000
Name: remote_ratio, dtype: float64
```

## Распределение процента времени, проведённого на удалённой работе сотрудников



Анализ распределения:

- Гибридная модель доминирует: большинство сотрудников в диапазоне 20-80%
- Два крайних кластера:
  - Полностью офисные (0%)
  - Полностью удалённые (100%)

## satisfaction\_level

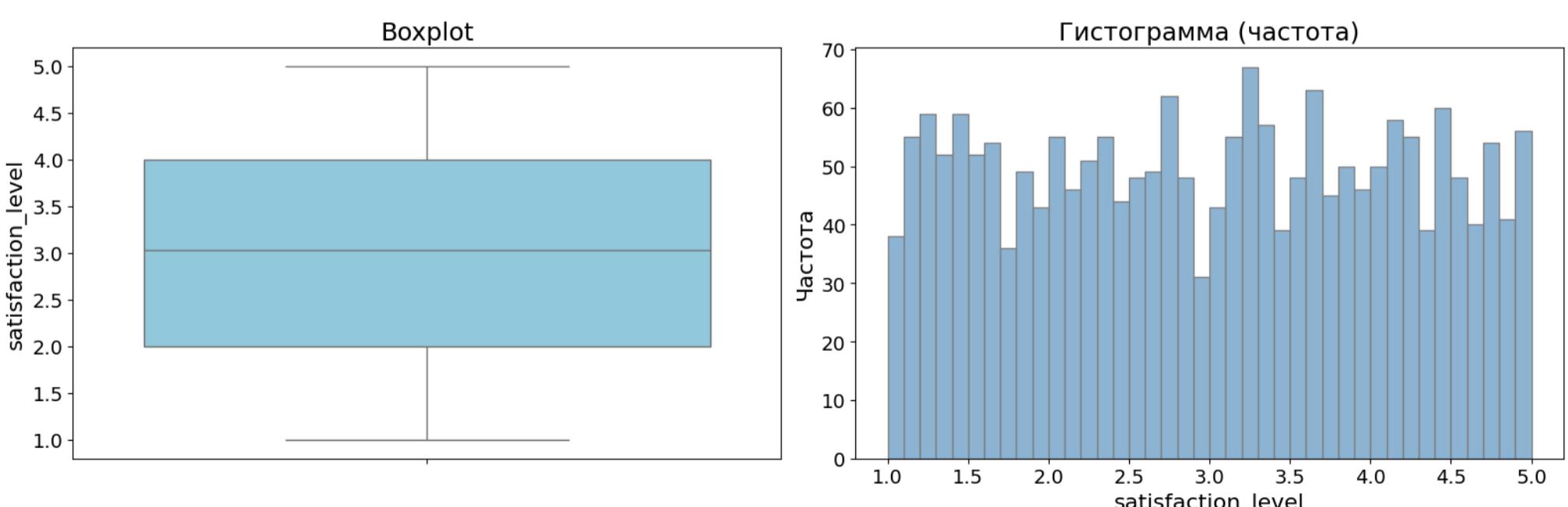
- satisfaction\_level** - уровень удовлетворённости сотрудника

```
In [38]: compare_box_and_hist(df, feature='satisfaction_level',
                           title="Распределение уровня удовлетворённости сотрудника", bins=40)
```

Статистика по признаку 'satisfaction\_level':

```
count    2000.000000
mean     2.995230
std      1.155431
min      1.000000
25%     2.000000
50%     3.025000
75%     4.000000
max     5.000000
Name: satisfaction_level, dtype: float64
```

## Распределение уровня удовлетворённости сотрудника



Интерпретация:

- Медиана чуть выше среднего
- Шкала 1-5, где 3 - нейтральное значение
- Среднее 2.995 - чуть ниже нейтрального уровня

## stress\_level

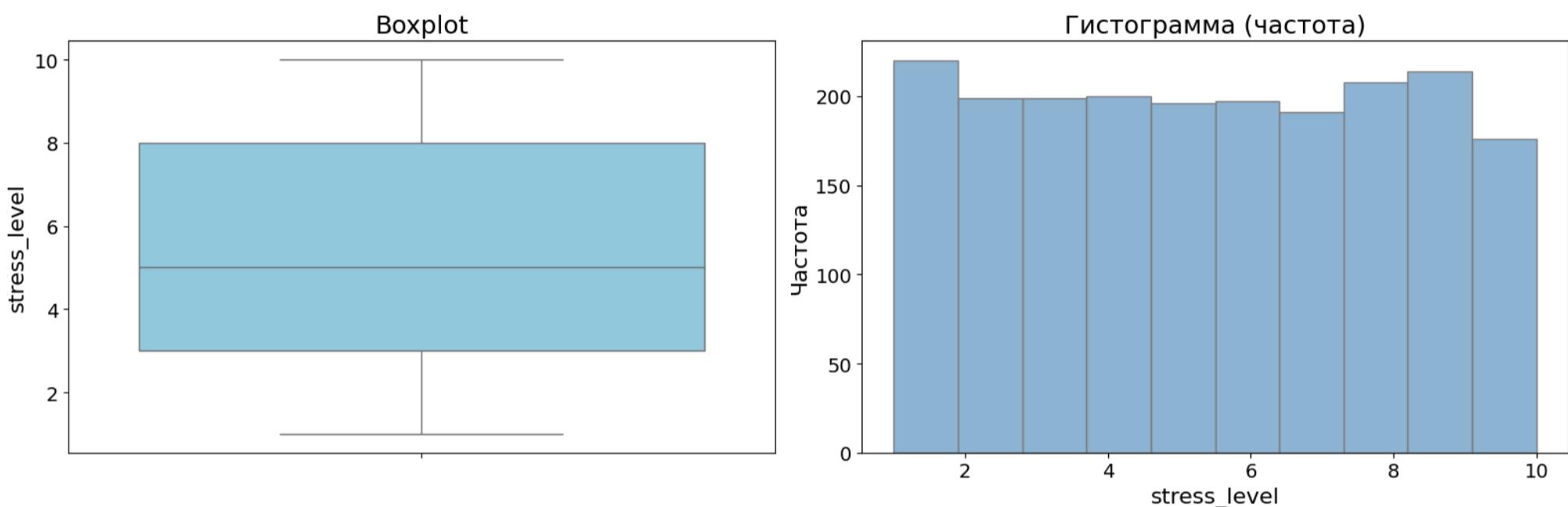
- **stress\_level** - уровень стресса сотрудника

```
In [39]: compare_box_and_hist(df, feature='stress_level',
                           title="Распределение уровня стресса сотрудника", bins=10)
```

📊 Статистика по признаку 'stress\_level':

```
count    2000.00000
mean     5.43200
std      2.88089
min     1.00000
25%    3.00000
50%    5.00000
75%    8.00000
max   10.00000
Name: stress_level, dtype: float64
```

**Распределение уровня стресса сотрудника**



```
In [40]: df['stress_level'].value_counts().head()
```

```
Out[40]: 1    220
9    214
8    208
4    200
2    199
Name: stress_level, dtype: int64
```

Анализ:

- 75-й перцентиль = 8 - у четверти сотрудников очень высокий стресс

## stress\_to\_satisfaction

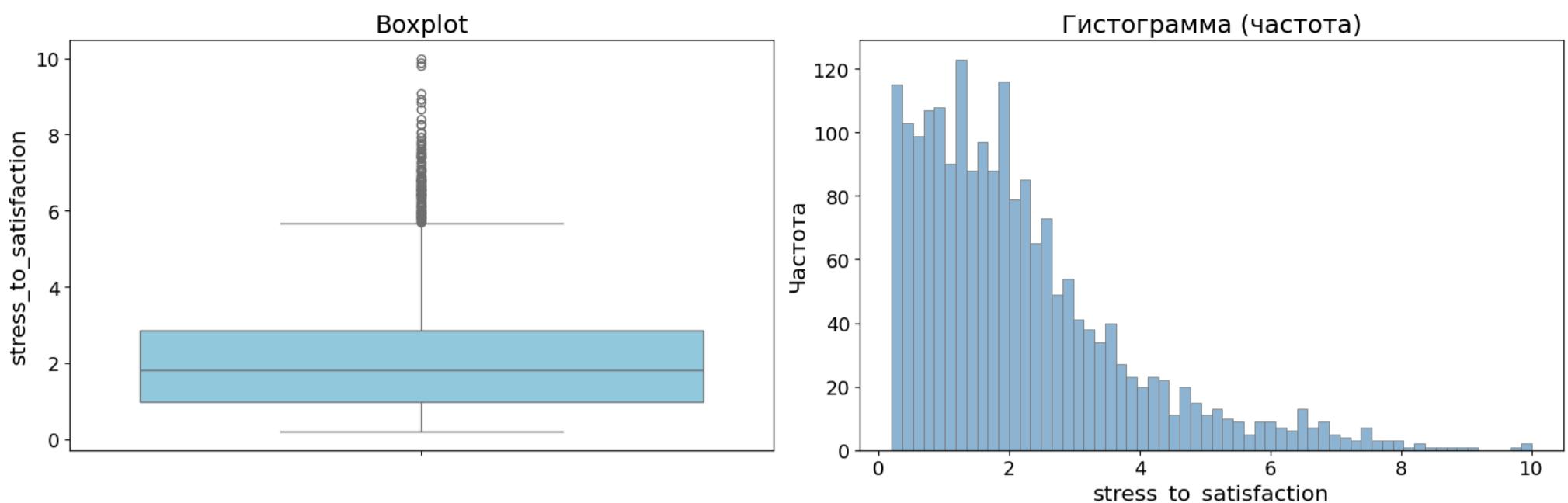
- **stress\_to\_satisfaction** - отношение уровня стресса к уровню удовлетворённости (индикатор дисбаланса)

```
In [41]: compare_box_and_hist(df, feature='stress_to_satisfaction',
                           title="Распределение отношения уровня стресса к уровню удовлетворённости сотрудника", bins=60)
```

📊 Статистика по признаку 'stress\_to\_satisfaction':

```
count    2000.00000
mean     2.172195
std      1.642699
min     0.201000
25%    0.971000
50%    1.812500
75%    2.857000
max   10.000000
Name: stress_to_satisfaction, dtype: float64
```

## Распределение отношения уровня стресса к уровню удовлетворённости сотрудника



- правостороннее распределение

Бизнес-интерпретация:

- Индикатор дисбаланса между стрессом и удовлетворённостью
- Значения > 3 - тревожный сигнал (стресс сильно превышает удовлетворённость)

5.05% сотрудников имеют критические значения (>5.686)

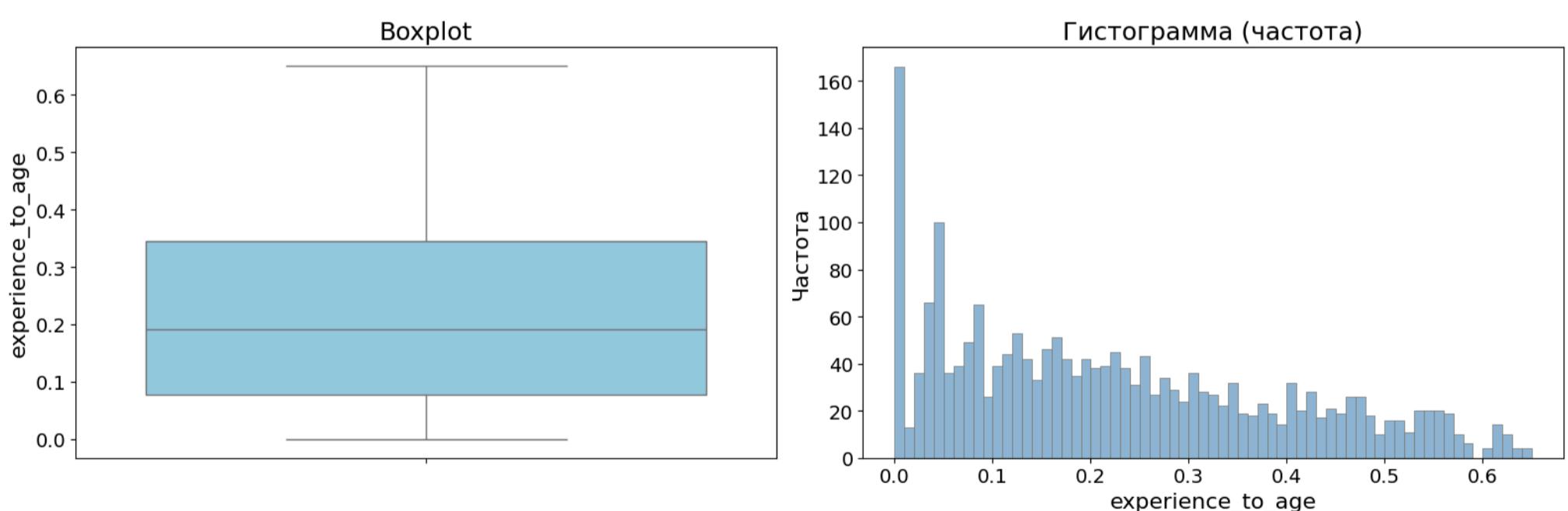
### experience\_to\_age

- experience\_to\_age** - отношение стажа к возрасту (показывает ранний или поздний старт карьеры).

```
In [42]: compare_box_and_hist(df, feature='experience_to_age',
                           title="Распределение отношения стажа к возрасту сотрудника", bins=65)
```

Статистика по признаку 'experience\_to\_age':  
count 2000.000000  
mean 0.223875  
std 0.170195  
min 0.000000  
25% 0.077000  
50% 0.192000  
75% 0.344250  
max 0.650000  
Name: experience\_to\_age, dtype: float64

## Распределение отношения стажа к возрасту сотрудника



```
In [43]: df['experience_to_age'].value_counts().head(10)
```

```
Out[43]: 0.000    166  
0.043     32  
0.045     27  
0.087     23  
0.333     20  
0.160     19  
0.125     19  
0.042     19  
0.154     18  
0.250     18  
Name: experience_to_age, dtype: int64
```

Сегментация по значению:

- меньше 0.15: поздний старт карьеры

- 0.15-0.30: нормальный карьерный путь
- больше 0.30: ранний старт карьеры

Большинство сотрудников начали карьеру относительно поздно (отношение < 0.3).

### career\_start\_age

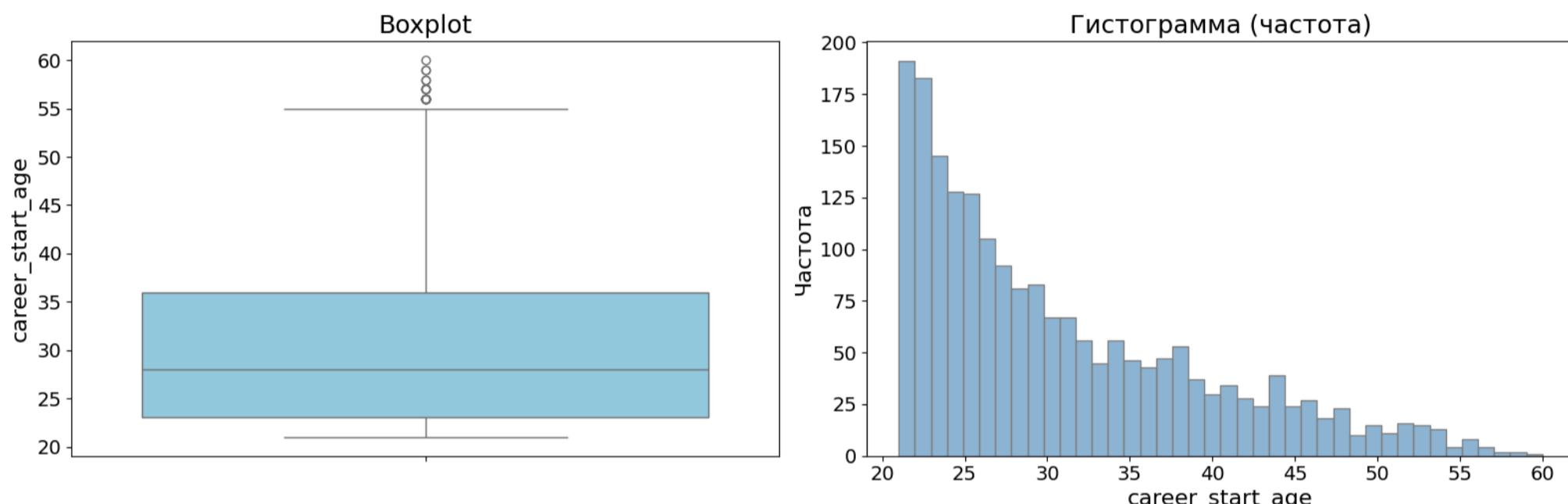
- **career\_start\_age** - возраст начала карьеры (age - experience)

```
In [44]: compare_box_and_hist(df, feature='career_start_age',
                           title="Распределение возраста начала карьеры сотрудника", bins=40)
```

Статистика по признаку 'career\_start\_age':

count	2000.000000
mean	30.620000
std	8.879528
min	21.000000
25%	23.000000
50%	28.000000
75%	36.000000
max	60.000000
Name:	career_start_age, dtype: float64

### Распределение возраста начала карьеры сотрудника



```
In [45]: df['career_start_age'].value_counts().head()
```

```
Out[45]:
```

21	191
22	183
23	145
24	128
25	127

Name: career\_start\_age, dtype: int64

- правостороннее распределение
- Выбросы: 17 случаев (0.85%) с карьерным стартом после 55 лет.

HR-инсайт: Большинство начинают карьеру в 21-25 лет. Те, кто начал после 35 лет - возможная вторая карьера или длительное образование.

### overload\_index

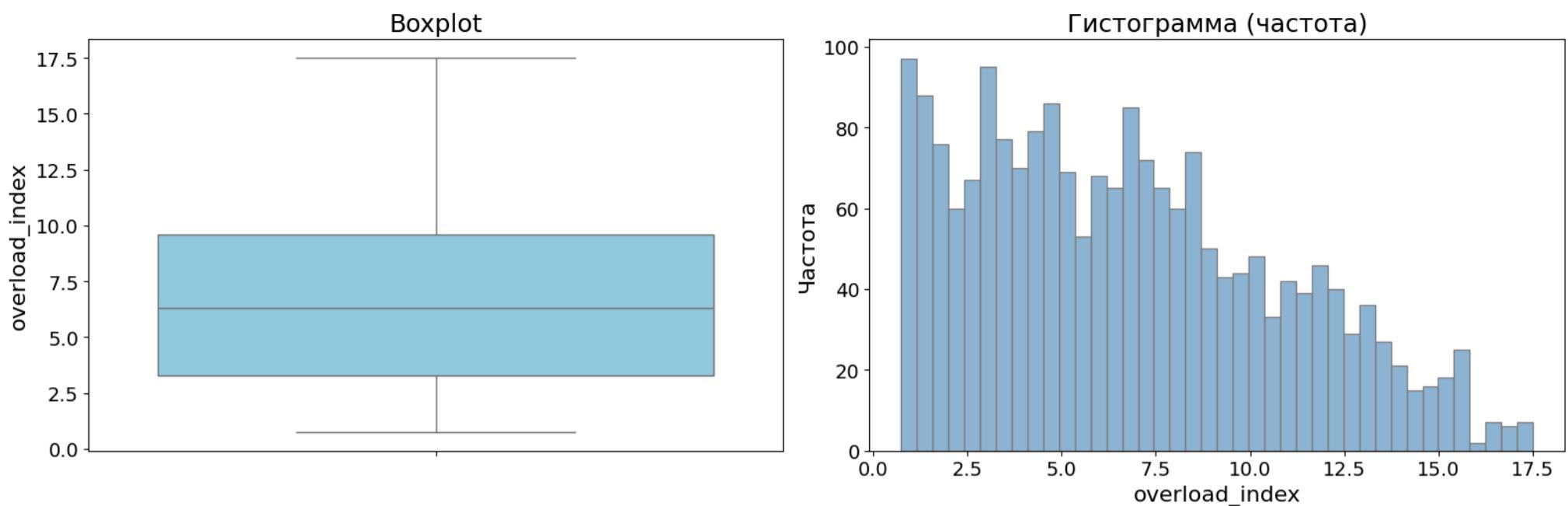
- **overload\_index** — индекс перегрузки:  $(\text{work\_hours\_per\_week} / 40) * \text{stress\_level}$ .

```
In [46]: compare_box_and_hist(df, feature='overload_index',
                           title="Распределение индекса перегрузки сотрудника", bins=40)
```

Статистика по признаку 'overload\_index':

count	2000.000000
mean	6.729925
std	4.027064
min	0.750000
25%	3.300000
50%	6.300000
75%	9.600000
max	17.500000
Name:	overload_index, dtype: float64

## Распределение индекса перегрузки сотрудника



```
In [47]: df['overload_index'].value_counts().head(10)
```

```
Out[47]: 6.00    20
7.00    20
6.75    18
5.25    18
4.50    18
4.80    17
9.00    17
12.25   16
3.10    16
6.80    15
Name: overload_index, dtype: int64
```

Интерпретация значений:

- Индекс перегрузки (overload\_index) имеет широкий разброс (от 0.75 до 17.5)
- меньше 4: низкая перегрузка
- 4-8: умеренная перегрузка
- 8-12: высокая перегрузка
- больше 12: критическая перегрузка

Среднее 6.73 - умеренная перегрузка в среднем по компании.

### hours\_above\_45

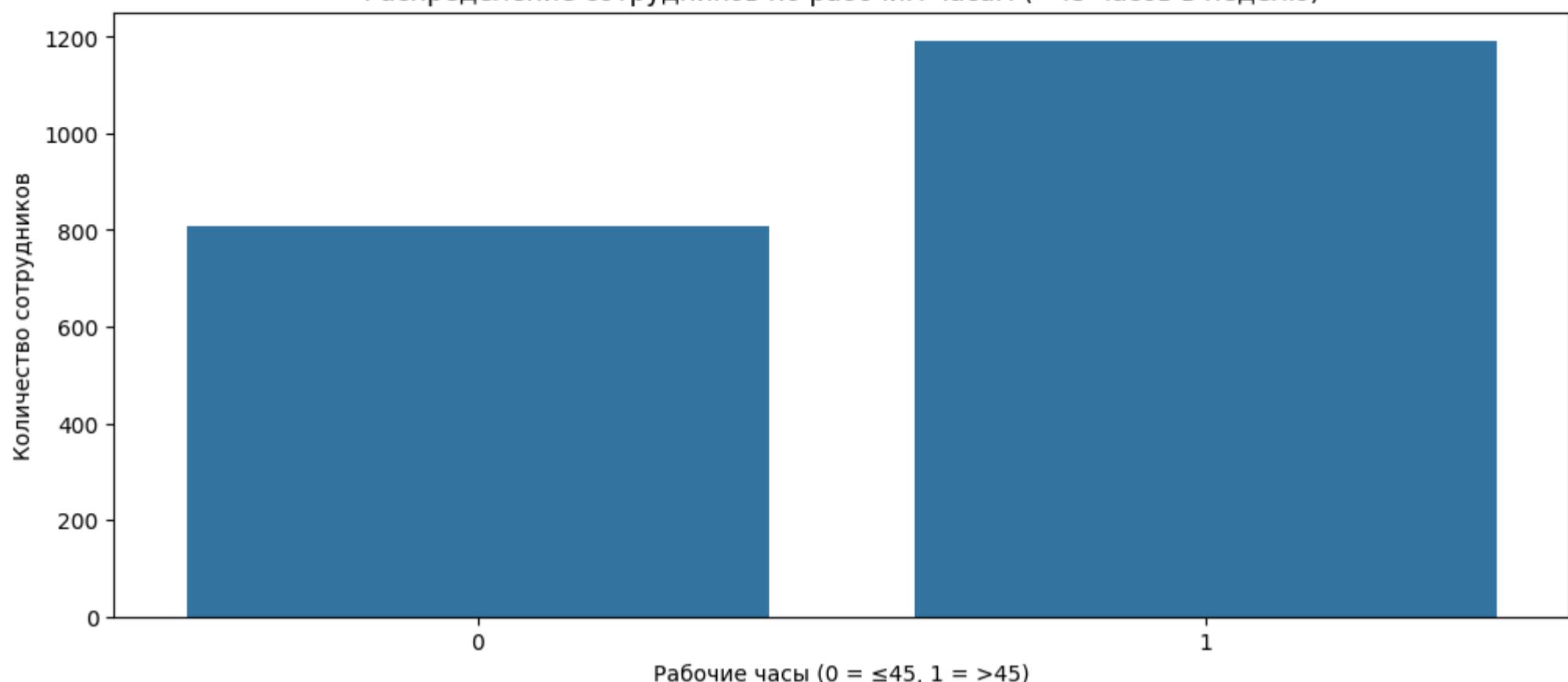
- **hours\_above\_45** — бинарный индикатор: 1, если сотрудник работает больше 45 часов в неделю, иначе 0.

```
In [48]: # Визуализация распределения индикатора hours_above_45
plt.figure(figsize=(12, 5))
sns.countplot(data=df, x='hours_above_45')
plt.title('Распределение сотрудников по рабочим часам (>45 часов в неделю)')
plt.xlabel('Рабочие часы (0 = ≤45, 1 = >45)')
plt.ylabel('Количество сотрудников')
plt.show()

# Абсолютные значения
hours_counts = df['hours_above_45'].value_counts()
print("Абсолютные значения по индикатору hours_above_45:")
print(hours_counts)

# Доли (нормализованные значения)
hours_rate = df['hours_above_45'].value_counts(normalize=True)
print("\nДоли по индикатору hours_above_45:")
print(hours_rate)
```

## Распределение сотрудников по рабочим часам (>45 часов в неделю)



Абсолютные значения по индикатору `hours_above_45`:

```
1    1191
0     809
Name: hours_above_45, dtype: int64
```

Доли по индикатору `hours_above_45`:

```
1    0.5955
0    0.4045
Name: hours_above_45, dtype: float64
```

### Первые инсайты о данных:

- Средний возраст сотрудников: ~40.7 лет
- Средний стаж: ~10.1 лет
- Средняя рабочая неделя: ~49.6 часов (уже превышает стандартные 40 часов!)
- Уровень удовлетворённости чуть ниже среднего (2.99 из 5)
- Уровень стресса средний (5.43 из 10)

⚠ Что требует внимания:

- Индикатор `hours_above_45`: 59.55% сотрудников работают больше 45 часов - это потенциально тревожный сигнал
- Это подтверждает высокую нагрузку в выборке и потенциальный фактор риска.

## Анализ выбросов

```
In [49]: def analyze_outliers(df, features):
    outliers_info = {}
    for feature in features:
        series = df[feature].dropna()
        Q1 = series.quantile(0.25)
        Q3 = series.quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        outliers = series[(series < lower_bound) | (series > upper_bound)]

        outliers_info[feature] = {
            # количество выбросов
            'count': len(outliers),
            # доля выбросов относительно всех строк датафрейма
            'percentage_total': len(outliers) / len(df) * 100,
            # доля выбросов относительно непустых значений признака
            'percentage_non_na': len(outliers) / len(series) * 100,
            # минимальное значение среди выбросов
            'min': outliers.min() if not outliers.empty else None,
            # максимальное значение среди выбросов
            'max': outliers.max() if not outliers.empty else None,

            # статистические границы
            'Q1': Q1,
            'Q3': Q3,
            'lower_bound': lower_bound,
            'upper_bound': upper_bound
        }
    return pd.DataFrame(outliers_info).T
```

```
In [50]: analyze_outliers(df, numerical)
```

	count	percentage_total	percentage_non_na	min	max	Q1	Q3	lower_bound	upper_bound
<b>age</b>	0.0	0.00	0.00	NaN	NaN	31.000	50.00000	2.500000	78.500000
<b>experience</b>	36.0	1.80	1.80	34.000	39.0	3.000	15.00000	-15.000000	33.000000
<b>work_hours_per_week</b>	0.0	0.00	0.00	NaN	NaN	39.000	60.00000	7.500000	91.500000
<b>remote_ratio</b>	0.0	0.00	0.00	NaN	NaN	24.000	75.00000	-52.500000	151.500000
<b>satisfaction_level</b>	0.0	0.00	0.00	NaN	NaN	2.000	4.00000	-1.000000	7.000000
<b>stress_level</b>	0.0	0.00	0.00	NaN	NaN	3.000	8.00000	-4.500000	15.500000
<b>stress_to_satisfaction</b>	101.0	5.05	5.05	5.691	10.0	0.971	2.85700	-1.858000	5.686000
<b>experience_to_age</b>	0.0	0.00	0.00	NaN	NaN	0.077	0.34425	-0.323875	0.745125
<b>career_start_age</b>	17.0	0.85	0.85	56.000	60.0	23.000	36.00000	3.500000	55.500000
<b>overload_index</b>	0.0	0.00	0.00	NaN	NaN	3.300	9.60000	-6.150000	19.050000
<b>hours_above_45</b>	0.0	0.00	0.00	NaN	NaN	0.000	1.00000	-1.500000	2.500000
<b>burnout</b>	129.0	6.45	6.45	1.000	1.0	0.000	0.00000	0.000000	0.000000

#### Минимальное количество выбросов в большинстве признаков

- Заметные выбросы только в:
  - stress\_to\_satisfaction (5.05% выбросов)
  - career\_start\_age (0.85% выбросов)
  - experience (1.8% выбросов)

### Анализ распределений признаков по классам целевой переменной

#### Количественные признаки

```
In [51]: def plot_box_and_hist_by_target(data, feature='age', target='burnout',
                                    title=None, bins=60):
    # Удаляем пропуски
    df = data[[feature, target]].dropna()

    # Общая статистика
    print(f"\n📊 Общая статистика по '{feature}':")
    print(df[feature].describe())

    # Статистика по целевой переменной
    for val in sorted(df[target].unique()):
        print(f"\n{target} = {val}:")
        print(df.loc[df[target] == val, feature].describe())

    # Заголовок
    if title is None:
        title = f'Распределение "{feature}" по целевой переменной "{target}"'

    # Два графика рядом
    fig, axes = plt.subplots(1, 2, figsize=(14, 6))
    fig.suptitle(title, fontsize=18, fontweight='bold')

    # Boxplot (исправленный вариант без предупреждений)
    sns.boxplot(data=df, x=target, y=feature, hue=target,
                ax=axes[0], palette='Set2', legend=False)
    axes[0].set_title('Boxplot')

    # Гистограмма (плотность)
    sns.histplot(data=df, x=feature, hue=target, bins=bins,
                  stat="density", common_norm=False, palette='muted', ax=axes[1])
    axes[1].set_title('Гистограмма (плотность)')

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

age

- age** - возраст сотрудника

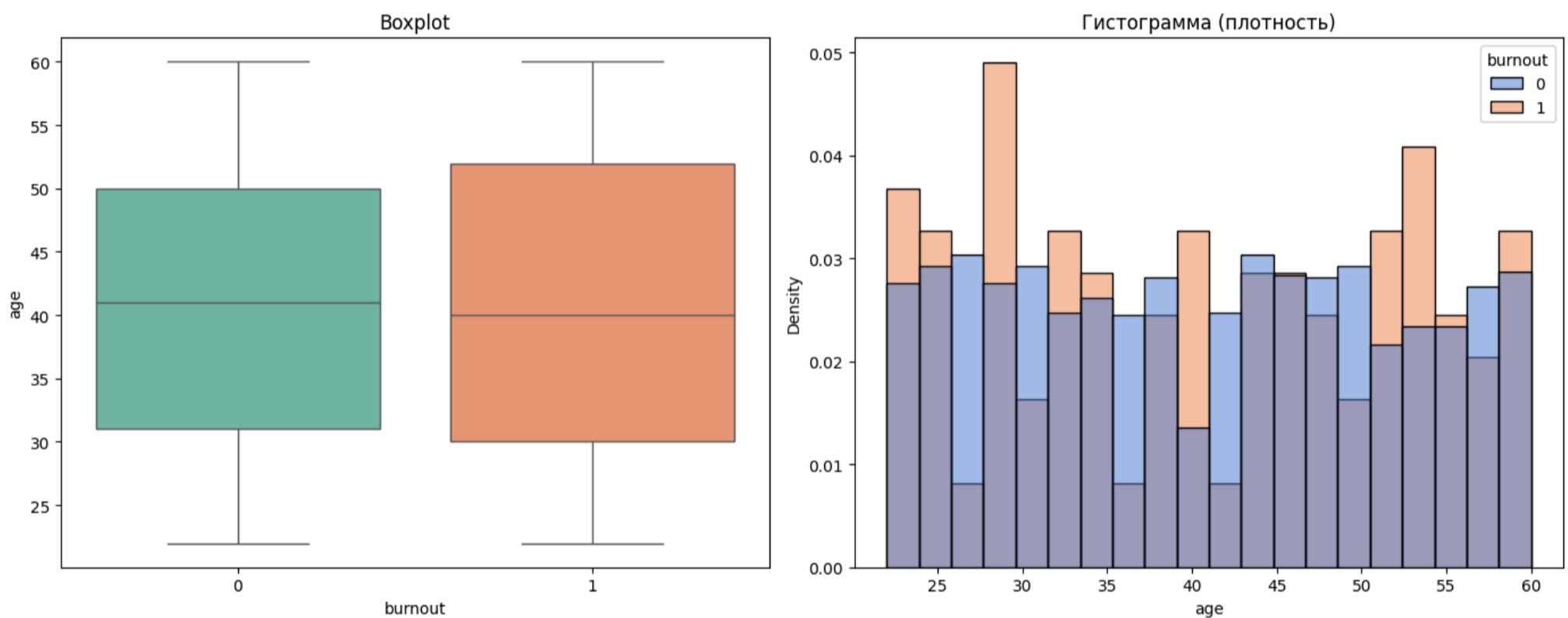
```
In [52]: plot_box_and_hist_by_target(df, feature='age',
                                    title='Распределение возраста сотрудников по выгоранию', bins=20)
```

```
Общая статистика по 'age':  
count    2000.000000  
mean     40.694500  
std      11.286756  
min     22.000000  
25%    31.000000  
50%    41.000000  
75%    50.000000  
max     60.000000  
Name: age, dtype: float64
```

```
burnout = 0:  
count    1871.000000  
mean     40.682523  
std      11.260347  
min     22.000000  
25%    31.000000  
50%    41.000000  
75%    50.000000  
max     60.000000  
Name: age, dtype: float64
```

```
burnout = 1:  
count    129.000000  
mean     40.868217  
std      11.706956  
min     22.000000  
25%    30.000000  
50%    40.000000  
75%    52.000000  
max     60.000000  
Name: age, dtype: float64
```

## Распределение возраста сотрудников по выгоранию



Статистический анализ:

- Незначимое различие средних ( $\Delta = 0.19$  лет)
- Распределения почти идентичны (перекрывающиеся гистограммы)
- Вывод: Возраст не является дифференцирующим фактором для выгорания

HR-инсайт: Программы профилактики выгорания должны охватывать все возрастные группы, нет "защищённых" возрастов

**experience**

- **experience** - стаж работы сотрудника (в годах)

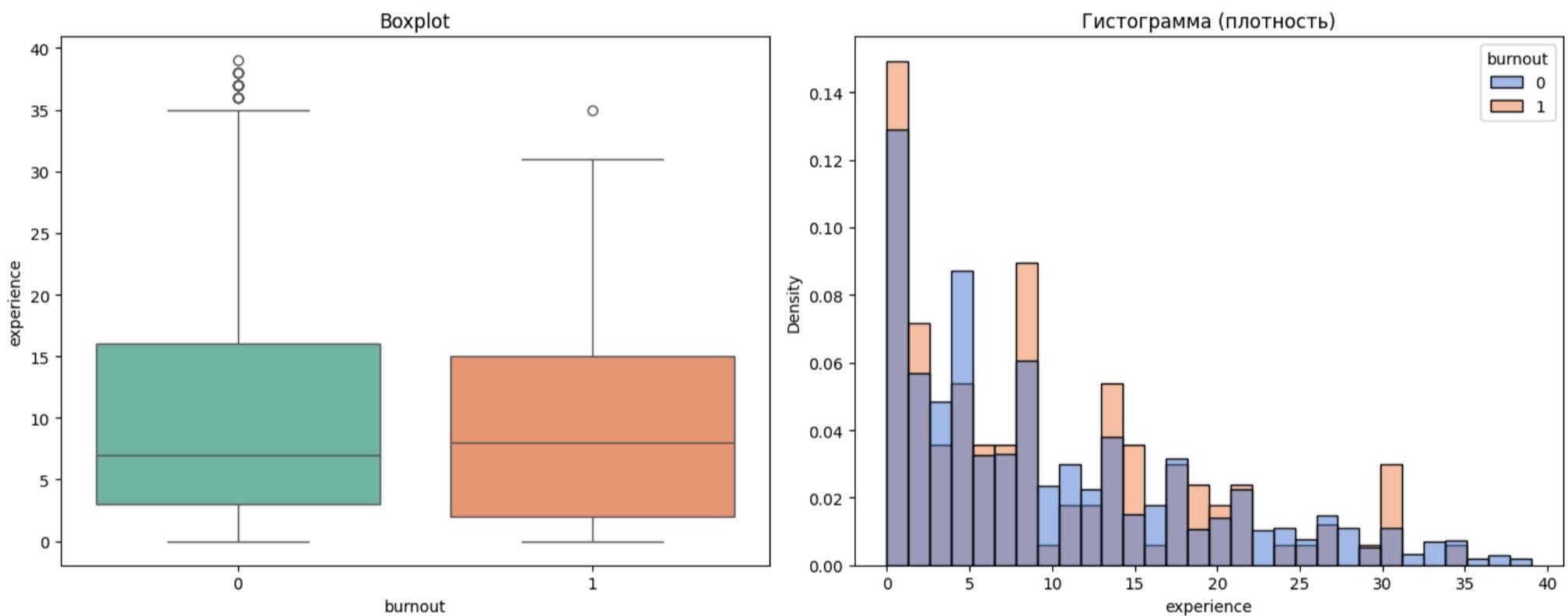
```
In [53]: plot_box_and_hist_by_target(df, feature='experience',  
                                 title='Распределение стажа работы сотрудников по выгоранию', bins=30)
```

```
Общая статистика по 'experience':  
count    2000.000000  
mean     10.074500  
std      9.148267  
min     0.000000  
25%     3.000000  
50%     7.000000  
75%    15.000000  
max     39.000000  
Name: experience, dtype: float64
```

```
burnout = 0:  
count    1871.000000  
mean     10.111170  
std      9.183188  
min     0.000000  
25%     3.000000  
50%     7.000000  
75%    16.000000  
max     39.000000  
Name: experience, dtype: float64
```

```
burnout = 1:  
count    129.000000  
mean     9.542636  
std      8.642200  
min     0.000000  
25%     2.000000  
50%     8.000000  
75%    15.000000  
max     35.000000  
Name: experience, dtype: float64
```

## Распределение стажа работы сотрудников по выгоранию



Статистический анализ:

- Небольшое различие ( $\Delta = 0.57$  лет)
- Группа с выгоранием имеет более низкий средний стаж
- Медианы близки (7 vs 8 лет)

Интерпретация:

- стаж оказывает слабое влияние на риск выгорания, но молодые специалисты немного более уязвимы и могут быть в группе риска.

`work_hours_per_week`

- `work_hours_per_week` - среднее количество рабочих часов в неделю

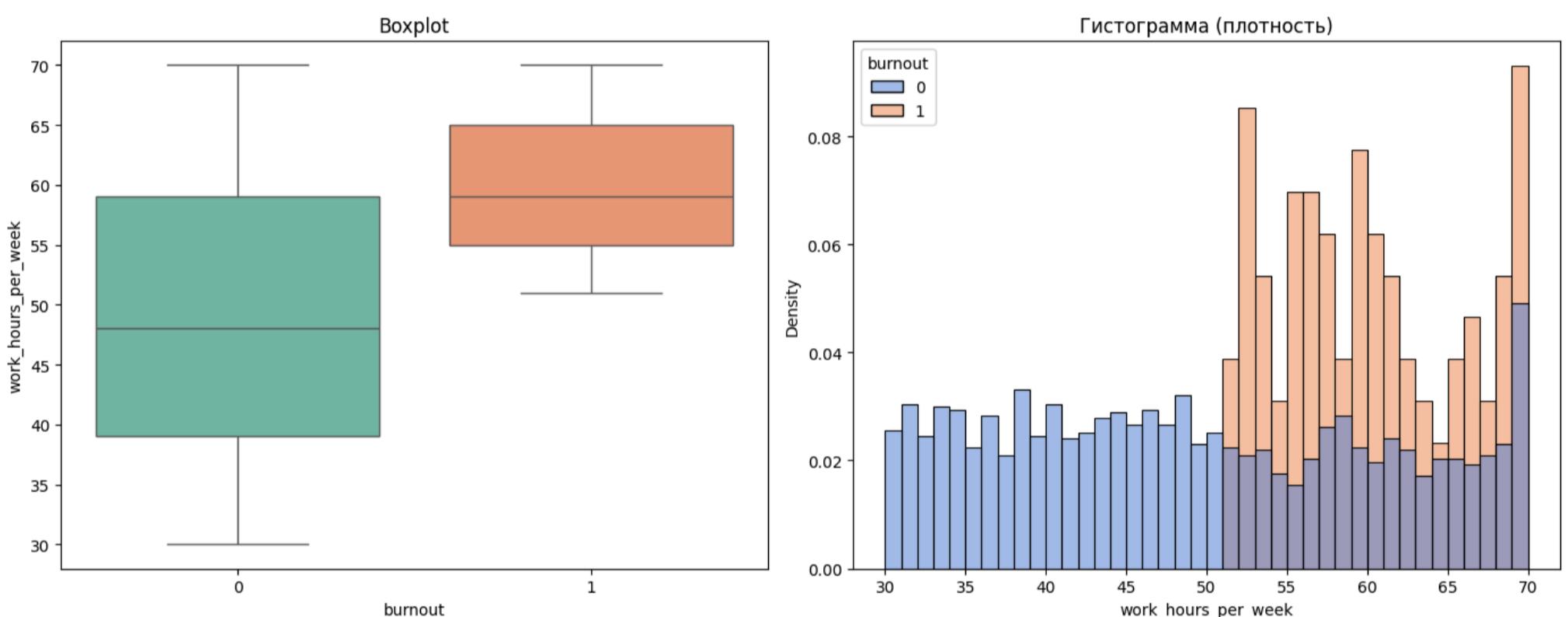
```
In [54]: plot_box_and_hist_by_target(df, feature='work_hours_per_week',  
                                 title='Распределение среднего количества рабочих часов в неделю сотрудников по выгоранию', bins=40)
```

```
Общая статистика по 'work_hours_per_week':
count    2000.000000
mean     49.588000
std      11.832424
min     30.000000
25%    39.000000
50%    49.000000
75%    60.000000
max     70.000000
Name: work_hours_per_week, dtype: float64
```

```
burnout = 0:
count    1871.000000
mean     48.886157
std      11.821370
min     30.000000
25%    39.000000
50%    48.000000
75%    59.000000
max     70.000000
Name: work_hours_per_week, dtype: float64
```

```
burnout = 1:
count    129.000000
mean     59.767442
std      5.769683
min     51.000000
25%    55.000000
50%    59.000000
75%    65.000000
max     70.000000
Name: work_hours_per_week, dtype: float64
```

## Распределение среднего количества рабочих часов в неделю сотрудников по выгоранию



СТАТИСТИЧЕСКИ ЗНАЧИМОЕ РАЗЛИЧИЕ:

- $\Delta = 10.87$  часов в неделю!
- Минимальное значение для  $\text{burnout}=1$ : 51 час (все выгоревшие работают >50 часов!)
- Распределение  $\text{burnout}=1$ : узкое ( $\text{std}=5.76$ ), сконцентрировано в диапазоне 55-65 часов
- КРИТИЧЕСКИЙ ПОРОГ: 51+ часов в неделю - все случаи выгорания находятся выше этого порога.

Вывод: переработки — ключевой фактор риска. Все сотрудники с Burnout работают >50 часов.

HR-рекомендация: Установить системный лимит на 50 рабочих часов в неделю.

`remote_ratio`

- `remote_ratio` - процент времени, проведённого на удалённой работе

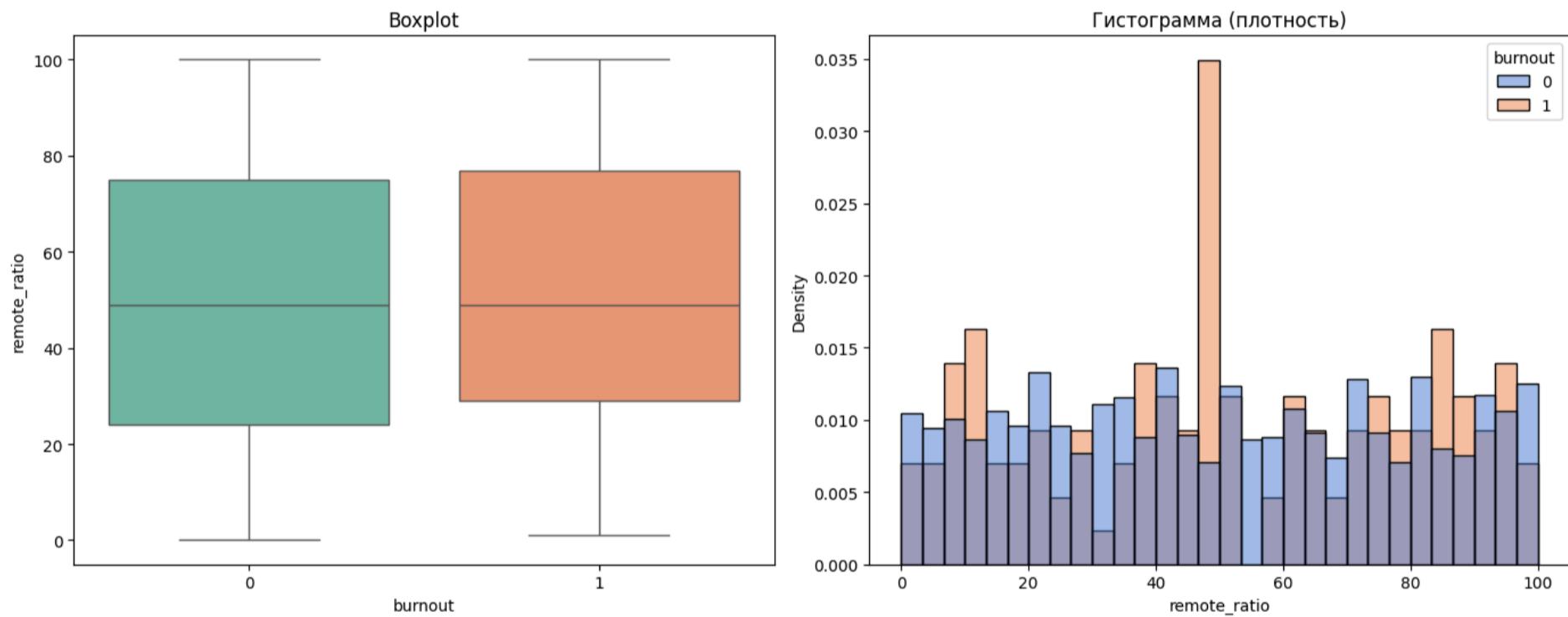
```
In [55]: plot_box_and_hist_by_target(df, feature='remote_ratio',
                                 title='Распределение процента времени, проведённого на удалённой работе сотрудником по выгоранию', bins=30)
```

```
Общая статистика по 'remote_ratio':
count    2000.000000
mean     49.973000
std      29.151298
min      0.000000
25%     24.000000
50%     49.000000
75%     75.000000
max     100.000000
Name: remote_ratio, dtype: float64
```

```
burnout = 0:
count    1871.000000
mean     49.889898
std      29.179175
min      0.000000
25%     24.000000
50%     49.000000
75%     75.000000
max     100.000000
Name: remote_ratio, dtype: float64
```

```
burnout = 1:
count    129.000000
mean     51.178295
std      28.829089
min     1.000000
25%     29.000000
50%     49.000000
75%     77.000000
max     100.000000
Name: remote_ratio, dtype: float64
```

## Распределение процента времени, проведённого на удалённой работе сотрудником по выгоранию



Статистический анализ:

- Незначимое различие ( $\Delta = 1.28\%$ )
- Распределения почти идентичны
- Минимальное значение для  $\text{burnout}=1$ : 1% (все выгоревшие имеют хотя бы минимальную удалёнку)

Вывод: Процент удалённой работы сам по себе не защищает от выгорания. Удалёнка не снижает риск, но и не усиливает его напрямую. Влияние может проявляться в сочетании с нагрузкой.

**satisfaction\_level**

- **satisfaction\_level** - уровень удовлетворённости сотрудника

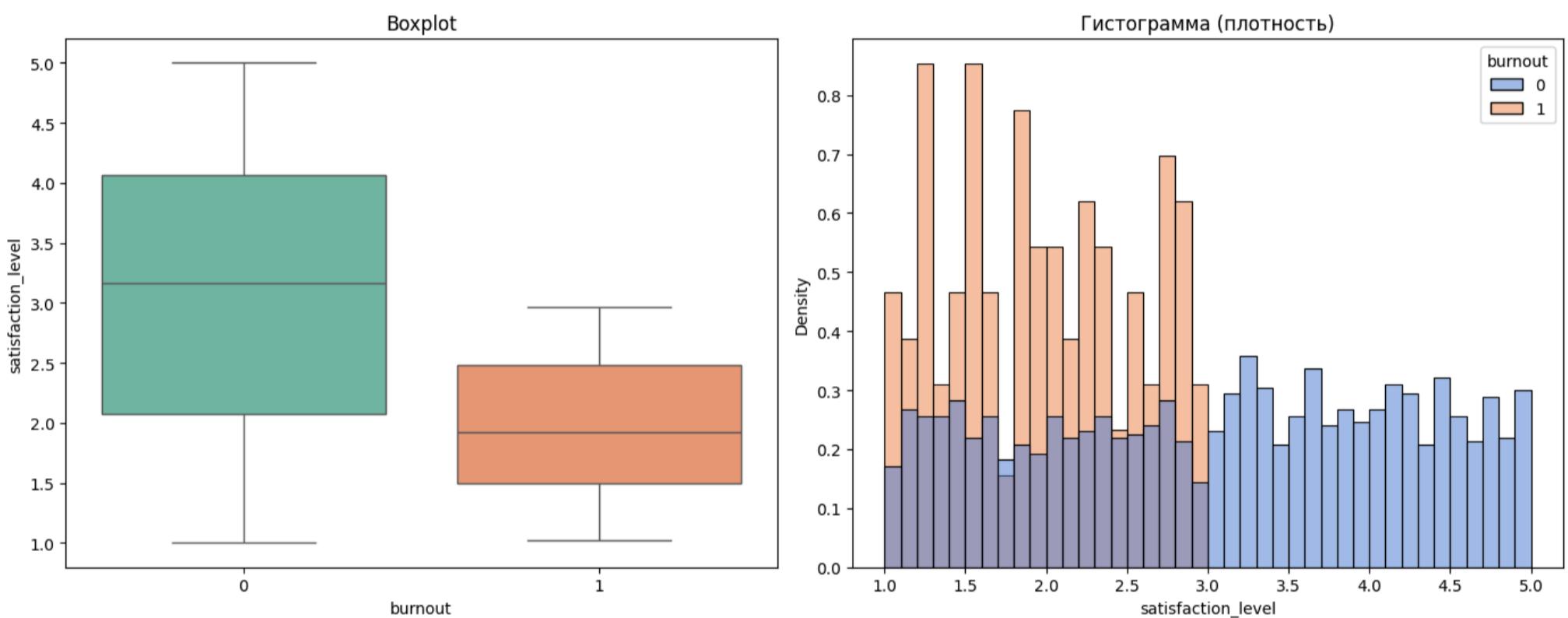
```
In [56]: plot_box_and_hist_by_target(df, feature='satisfaction_level',
                                 title='Распределение уровня удовлетворённости сотрудником по выгоранию', bins=40)
```

```
Общая статистика по 'satisfaction_level':
count    2000.00000
mean     2.995230
std      1.155431
min     1.000000
25%     2.000000
50%     3.025000
75%     4.000000
max     5.000000
Name: satisfaction_level, dtype: float64
```

```
burnout = 0:
count    1871.00000
mean     3.065783
std      1.152123
min     1.000000
25%     2.080000
50%     3.170000
75%     4.065000
max     5.000000
Name: satisfaction_level, dtype: float64
```

```
burnout = 1:
count    129.00000
mean     1.971938
std      0.573501
min     1.020000
25%     1.500000
50%     1.920000
75%     2.480000
max     2.970000
Name: satisfaction_level, dtype: float64
```

## Распределение уровня удовлетворённости сотрудником по выгоранию



СТАТИСТИЧЕСКИ ЗНАЧИМОЕ РАЗЛИЧИЕ:

- $\Delta = 1.10$  балла по 5-балльной шкале!
- Максимальное значение для  $\text{burnout}=1$ : 2.97 (ниже нейтрального 3.0!)
- Все выгоревшие имеют  $\text{satisfaction\_level} < 3.0$

КРИТИЧЕСКИЙ ПОРОГ: 3.0 - пороговое значение. Ниже - группа риска.

Вывод: низкая удовлетворённость — сильный предиктор выгорания.

**stress\_level**

- **stress\_level** - уровень стресса сотрудника

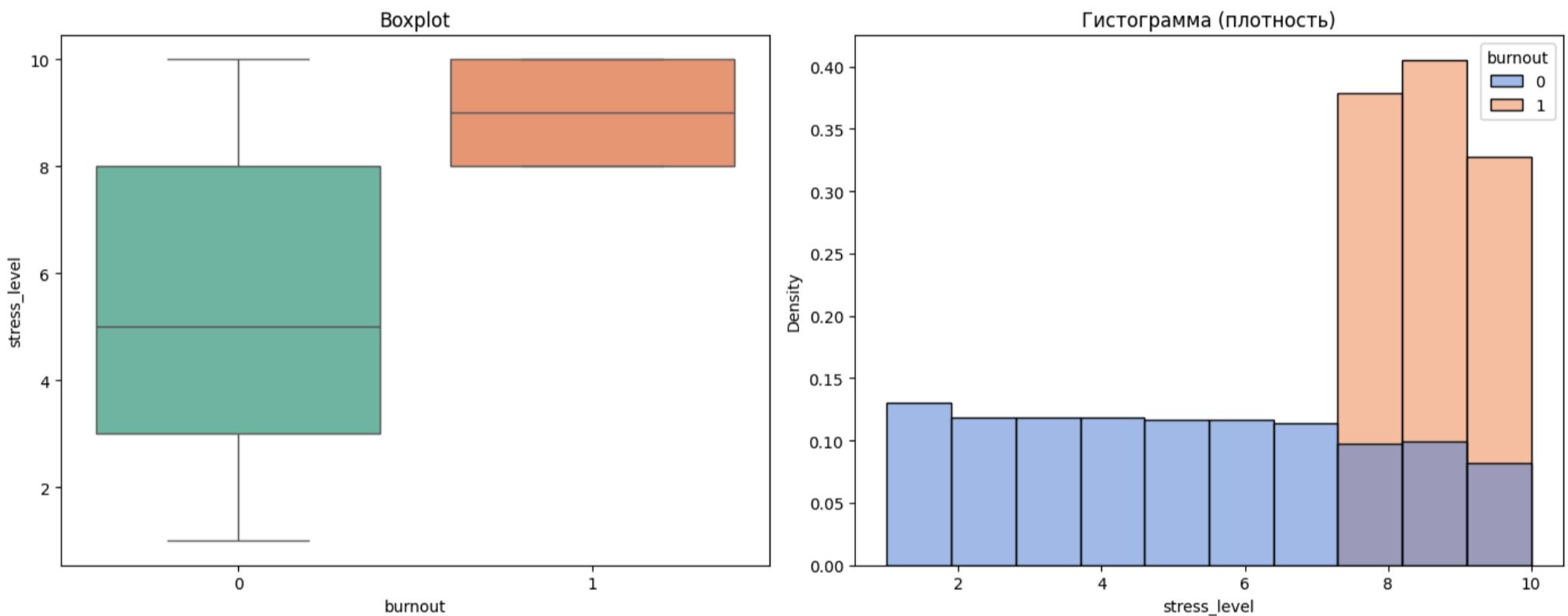
```
In [57]: plot_box_and_hist_by_target(df, feature='stress_level',
                                  title='Распределение уровня стресса сотрудником по выгоранию', bins=10)
```

```
Общая статистика по 'stress_level':  
count    2000.00000  
mean      5.43200  
std       2.88089  
min       1.00000  
25%      3.00000  
50%      5.00000  
75%      8.00000  
max      10.00000  
Name: stress_level, dtype: float64
```

```
burnout = 0:  
count    1871.00000  
mean      5.189204  
std       2.813168  
min       1.000000  
25%      3.000000  
50%      5.000000  
75%      8.000000  
max      10.000000  
Name: stress_level, dtype: float64
```

```
burnout = 1:  
count    129.00000  
mean     8.953488  
std       0.799027  
min       8.000000  
25%      8.000000  
50%      9.000000  
75%      10.000000  
max      10.000000  
Name: stress_level, dtype: float64
```

## Распределение уровня стресса сотрудником по выгоранию



СТАТИСТИЧЕСКИ ЗНАЧИМОЕ РАЗЛИЧИЕ:

- $\Delta = 3.76$  балла по 10-балльной шкале!
- Минимальное значение для  $\text{burnout}=1$ : 8.0 (все выгоревшие имеют высокий стресс)
- Распределение  $\text{burnout}=1$ : очень узкое ( $\text{std}=0.79$ ), концентрируется вокруг 9

КРИТИЧЕСКИЙ ПОРОГ: 8.0+ - все случаи выгорания имеют стресс  $\geq 8$ .

Вывод: высокий стресс — главный фактор риска.

`stress_to_satisfaction`

- **stress\_to\_satisfaction** - отношение уровня стресса к уровню удовлетворённости (индикатор дисбаланса)

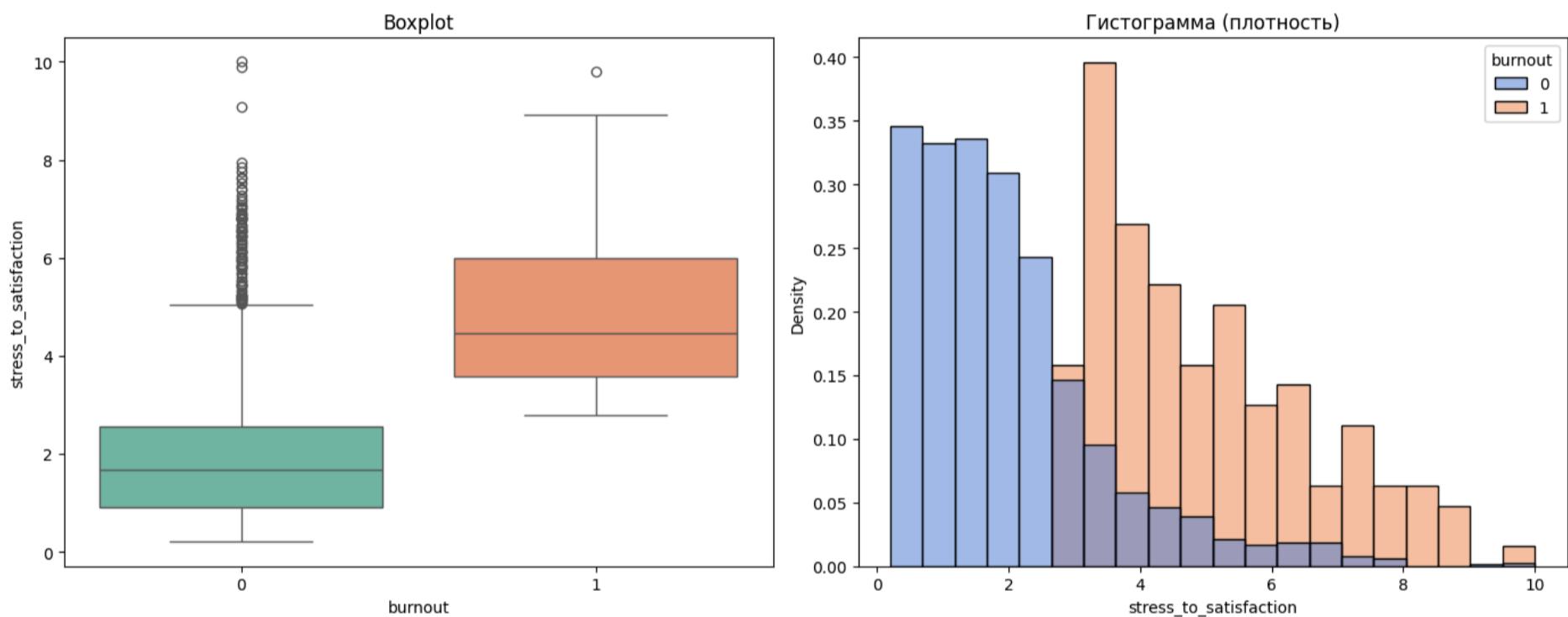
```
In [58]: plot_box_and_hist_by_target(df, feature='stress_to_satisfaction',  
                                 title='Распределение отношения уровня стресса к уровню удовлетворённости сотрудником по выгоранию', bins=20)
```

```
📊 Общая статистика по 'stress_to_satisfaction':
count    2000.000000
mean      2.172195
std       1.642699
min       0.201000
25%      0.971000
50%      1.812500
75%      2.857000
max       10.000000
Name: stress_to_satisfaction, dtype: float64
```

```
burnout = 0:
count    1871.000000
mean      1.978645
std       1.454306
min       0.201000
25%      0.916000
50%      1.684000
75%      2.567500
max       10.000000
Name: stress_to_satisfaction, dtype: float64
```

```
burnout = 1:
count    129.000000
mean      4.979434
std       1.659431
min       2.778000
25%      3.586000
50%      4.469000
75%      6.000000
max       9.804000
Name: stress_to_satisfaction, dtype: float64
```

## Распределение отношения уровня стресса к уровню удовлетворённости сотрудником по выгоранию



### СТАТИСТИЧЕСКИ ЗНАЧИМОЕ РАЗЛИЧИЕ:

- $\Delta = 3.00$  (в 2.5 раза выше!)
- Минимальное значение для burnout=1: 2.78

Интерпретация: При выгорании стресс превышает удовлетворённость в ~5 раз (в среднем по группе без выгорания - в 2 раза).

Вывод: дисбаланс стресс/удовлетворённость — мощный индикатор Burnout.

### experience\_to\_age

- **experience\_to\_age** - отношение стажа к возрасту (показывает ранний или поздний старт карьеры).

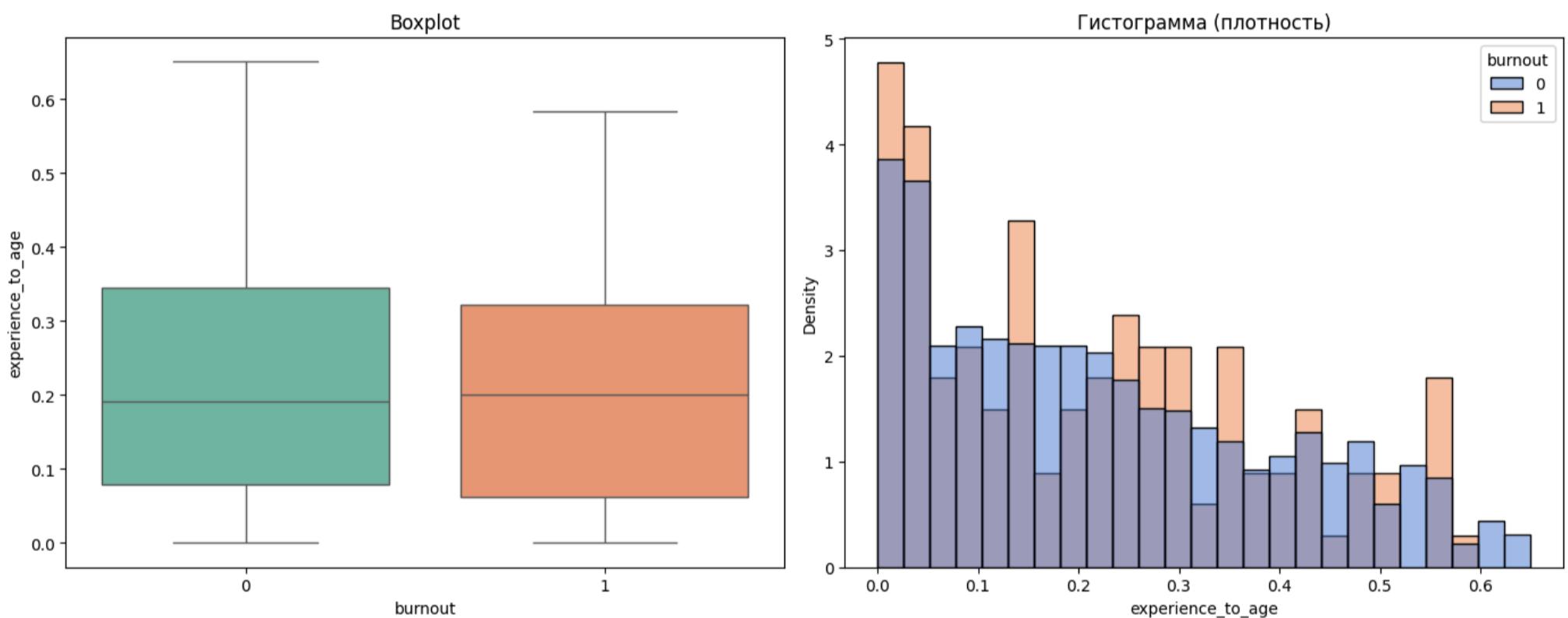
```
In [59]: plot_box_and_hist_by_target(df, feature='experience_to_age',
                                 title='Распределение отношения стажа к возрасту сотрудником по выгоранию', bins=25)
```

```
Общая статистика по 'experience_to_age':  
count    2000.000000  
mean      0.223875  
std       0.170195  
min       0.000000  
25%      0.077000  
50%      0.192000  
75%      0.344250  
max       0.650000  
Name: experience_to_age, dtype: float64
```

```
burnout = 0:  
count    1871.000000  
mean      0.224626  
std       0.170533  
min       0.000000  
25%      0.078500  
50%      0.192000  
75%      0.345500  
max       0.650000  
Name: experience_to_age, dtype: float64
```

```
burnout = 1:  
count    129.000000  
mean      0.212977  
std       0.165480  
min       0.000000  
25%      0.062000  
50%      0.200000  
75%      0.322000  
max       0.583000  
Name: experience_to_age, dtype: float64
```

### Распределение отношения стажа к возрасту сотрудником по выгоранию



### career\_start\_age

- **career\_start\_age** - возраст начала карьеры (age - experience)

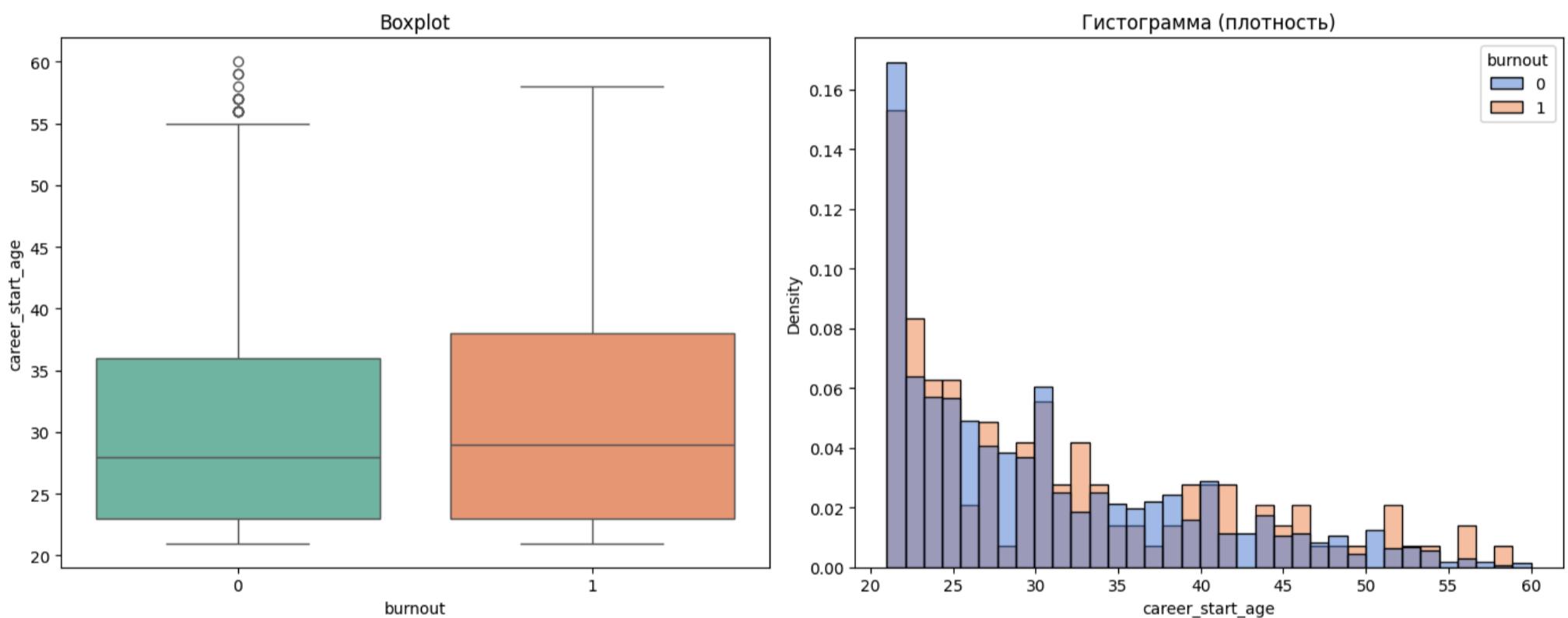
```
In [60]: plot_box_and_hist_by_target(df, feature='career_start_age',  
                                 title='Распределение возраста начала карьеры по выгоранию', bins=35)
```

```
Общая статистика по 'career_start_age':  
count    2000.000000  
mean     30.620000  
std      8.879528  
min     21.000000  
25%    23.000000  
50%    28.000000  
75%    36.000000  
max     60.000000  
Name: career_start_age, dtype: float64
```

```
burnout = 0:  
count    1871.000000  
mean     30.571352  
std      8.826014  
min     21.000000  
25%    23.000000  
50%    28.000000  
75%    36.000000  
max     60.000000  
Name: career_start_age, dtype: float64
```

```
burnout = 1:  
count    129.000000  
mean     31.325581  
std      9.631623  
min     21.000000  
25%    23.000000  
50%    29.000000  
75%    38.000000  
max     58.000000  
Name: career_start_age, dtype: float64
```

### Распределение возраста начала карьеры по выгоранию



Вывод: различия минимальны, фактор не критичен.

overload\_index

- **overload\_index** — индекс перегрузки:  $(\text{work\_hours\_per\_week} / 40) * \text{stress\_level}$ .

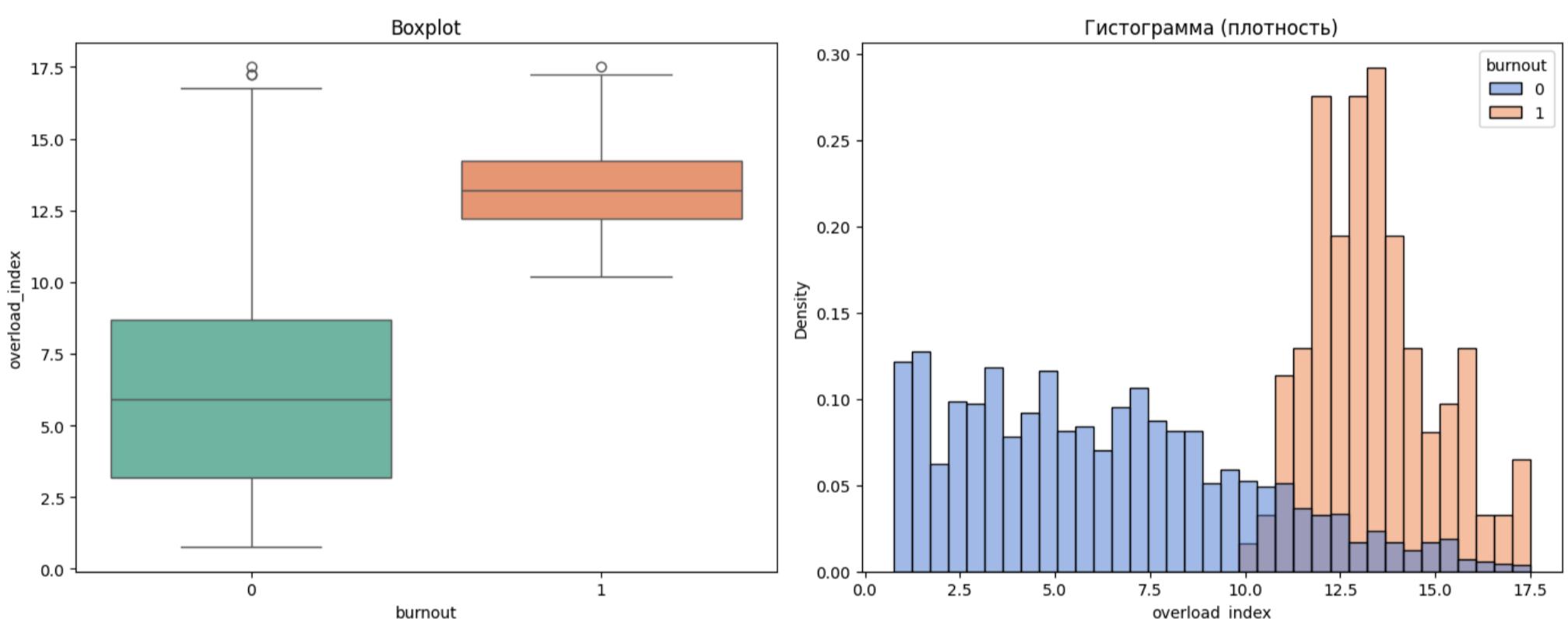
```
In [61]: plot_box_and_hist_by_target(df, feature='overload_index',  
                                 title='Распределение индекса перегрузки по выгоранию', bins=35)
```

```
Общая статистика по 'overload_index':
count    2000.00000
mean      6.729925
std       4.027064
min       0.750000
25%      3.300000
50%      6.300000
75%      9.600000
max      17.500000
Name: overload_index, dtype: float64
```

```
burnout = 0:
count    1871.000000
mean      6.272742
std       3.729712
min       0.750000
25%      3.200000
50%      5.900000
75%      8.700000
max      17.500000
Name: overload_index, dtype: float64
```

```
burnout = 1:
count    129.000000
mean     13.360853
std       1.634739
min      10.200000
25%     12.200000
50%     13.200000
75%     14.250000
max      17.500000
Name: overload_index, dtype: float64
```

## Распределение индекса перегрузки по выгоранию



СТАТИСТИЧЕСКИ ЗНАЧИМОЕ РАЗЛИЧИЕ:

- $\Delta = 7.10$  (более чем в 2 раза!)
- Минимальное значение для  $\text{burnout}=1$ : 10.2
- Медиана: 13.2 (критическая перегрузка)

Интерпретация: При выгорании сотрудники испытывают перегрузку, эквивалентную работе в 1.5 раза больше нормы при максимальном стрессе.

Вывод: перегрузка (часы  $\times$  стресс) — сильнейший фактор риска.

## Категориальные признаки

```
In [62]: print("Категориальные переменные в датасете df:", categorical)
```

Категориальные переменные в датасете df: ['gender', 'job\_role', 'age\_bin', 'experience\_cat', 'work\_hours\_bin', 'remote\_bin', 'remote\_cat', 'stress\_cat', 'satisfaction\_cat']

```
In [63]: def plot_cat_feature_by_target(data, feature='job', target='burnout', title=None):
    # Удаляем пропуски
    df = data[[feature, target]].dropna()
```

```
# Общие частоты значений признака
print(f"\nОбщее распределение признака '{feature}':")
counts = df[feature].value_counts()
percentages = df[feature].value_counts(normalize=True) * 100
summary_table = pd.DataFrame({'Частота': counts, 'Процент': percentages.round(2)})
print(summary_table)

# Частоты по целевой переменной
print(f"\nРаспределение признака '{feature}' по значениям целевой переменной '{target}':")
for val in sorted(df[target].unique()):
    print(f"\n{target} = {val}:")
    counts = df[df[target] == val][feature].value_counts()
```

```

percentages = df[df[target] == val][feature].value_counts(normalize=True) * 100
freq_table = pd.DataFrame({'Частота': counts, 'Процент': percentages.round(2)})
print(freq_table)

# Заголовок графика по умолчанию
if title is None:
    title = f'Распределение категориального признака "{feature}" по целевой переменной'

fig, axes = plt.subplots(1, 2, figsize=(20, 8))
fig.suptitle(title, fontsize=20)

# Распределение по категориям и целевой переменной
sns.countplot(data=df, x=feature, hue=target, palette='pastel', ax=axes[0])
axes[0].set_title('Распределение по категориям и целевой переменной', fontsize=16)
axes[0].set_xlabel(feature, fontsize=14)
axes[0].set_ylabel('Частота', fontsize=14)
axes[0].tick_params(axis='x', labelsize=12)
axes[0].tick_params(axis='y', labelsize=12)
axes[0].legend(title=target, fontsize=12, title_fontsize=13)

# Общее распределение категорий
sns.countplot(data=df, x=feature, hue=feature, palette='pastel', ax=axes[1], legend=False)
axes[1].set_title('Общее распределение категорий', fontsize=16)
axes[1].set_xlabel(feature, fontsize=14)
axes[1].set_ylabel('Частота', fontsize=14)
axes[1].tick_params(axis='x', labelsize=12)
axes[1].tick_params(axis='y', labelsize=12)

plt.tight_layout()
plt.show()

```

hours\_above\_45

- **hours\_above\_45** — бинарный индикатор: 1, если сотрудник работает больше 45 часов в неделю, иначе 0.

In [64]: `plot_cat_feature_by_target(df, feature='hours_above_45',  
 title='Распределение сотрудников по рабочим часам (>45 часов в неделю) по выгоранию')`

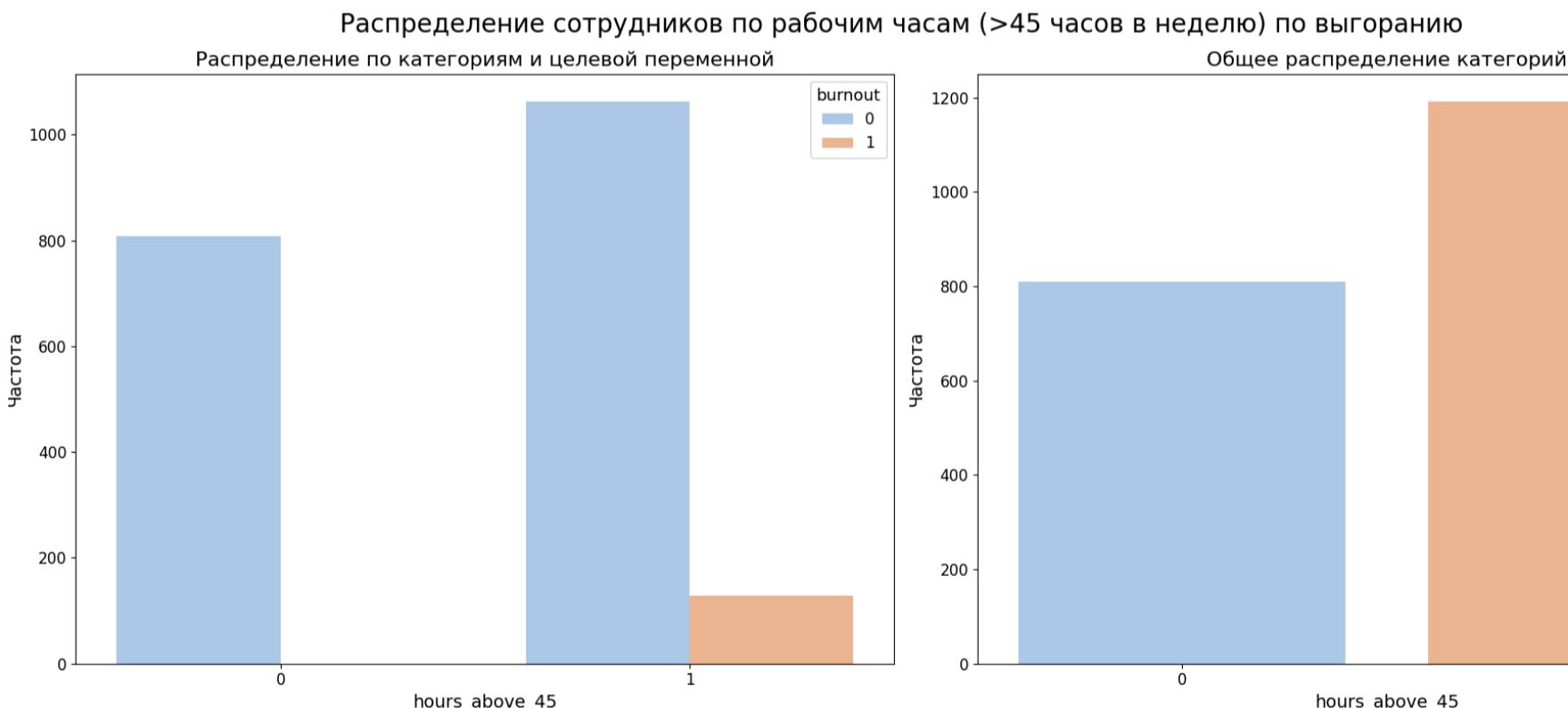
Общее распределение признака 'hours\_above\_45':

	Частота	Процент
1	1191	59.55
0	809	40.45

Распределение признака 'hours\_above\_45' по значениям целевой переменной 'burnout':

	Частота	Процент
1	1062	56.76
0	809	43.24

	Частота	Процент
1	129	100.0



СТАТИСТИЧЕСКИ КРИТИЧЕСКИЙ ФАКТОР:

- 100% выгоревших работают >45 часов
- Относительный риск: работа >45 часов увеличивает риск выгорания

Вывод: переработки — обязательное условие для Burnout в этом датасете.

gender

- **gender** — пол сотрудника (Male / Female).

```
In [65]: plot_cat_feature_by_target(df, feature='gender',
                                 title='Распределение пола сотрудника по выгоранию')
```

Общее распределение признака 'gender':

	Частота	Процент
Male	1023	51.15
Female	977	48.85

Распределение признака 'gender' по значениям целевой переменной 'burnout':

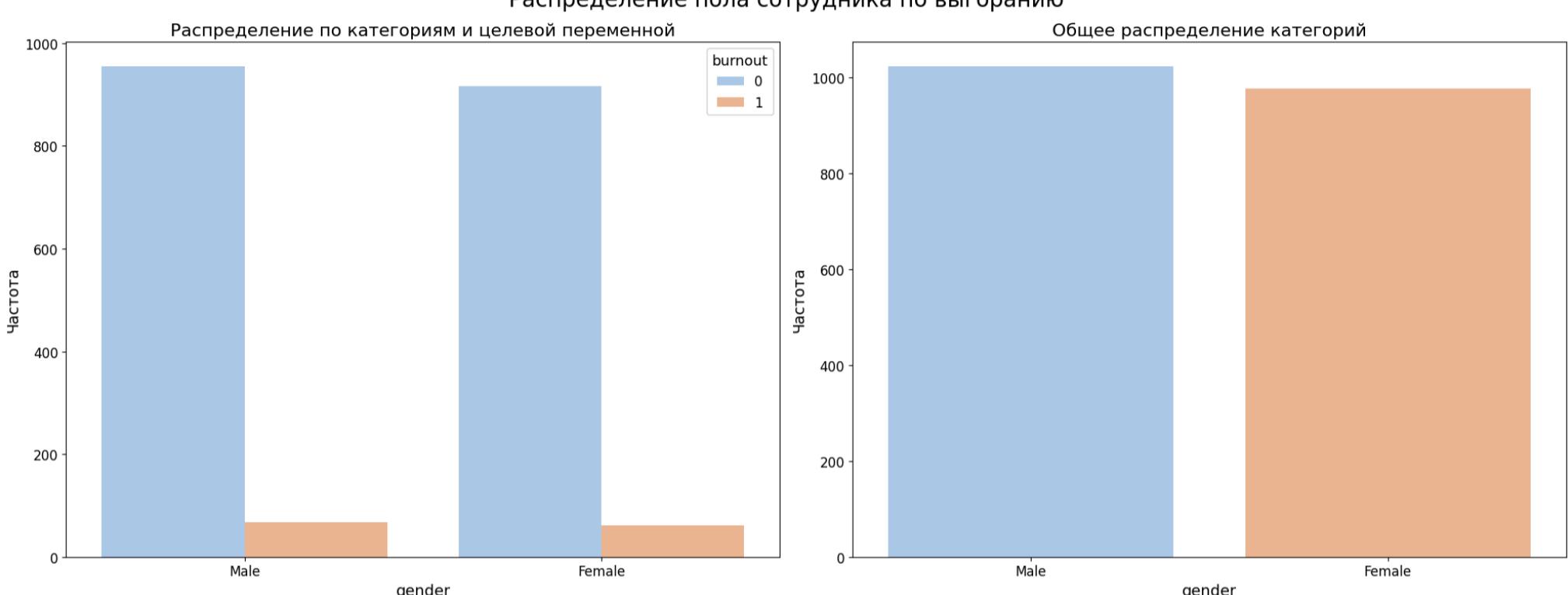
burnout = 0:

	Частота	Процент
Male	955	51.04
Female	916	48.96

burnout = 1:

	Частота	Процент
Male	68	52.71
Female	61	47.29

Распределение пола сотрудника по выгоранию



Статистический анализ:

- Незначимые различия в пропорциях |

Вывод: Пол не является дифференцирующим фактором

job\_role

- **job\_role** — должность сотрудника

```
In [66]: plot_cat_feature_by_target(df, feature='job_role',
                                 title='Распределение должностей сотрудников по выгоранию')
```

Общее распределение признака 'job\_role':

	Частота	Процент
Manager	419	20.95
Analyst	413	20.65
Sales	391	19.55
HR	391	19.55
Engineer	386	19.30

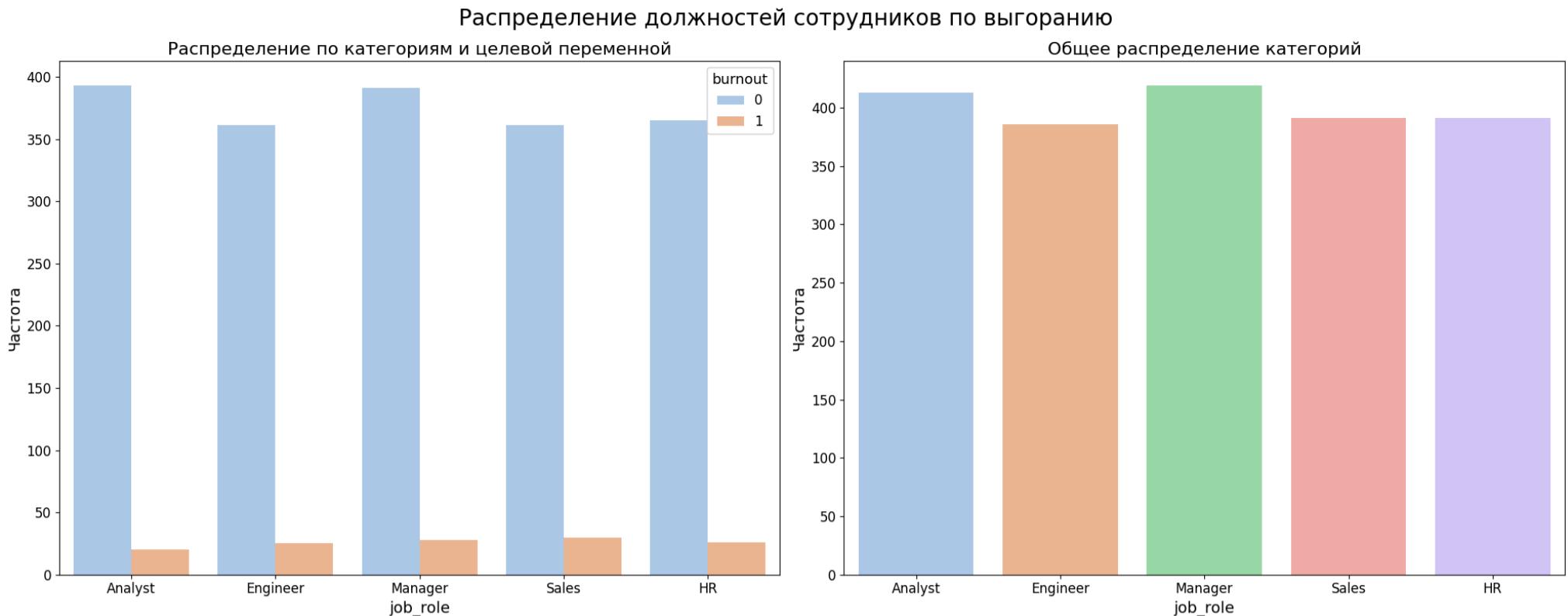
Распределение признака 'job\_role' по значениям целевой переменной 'burnout':

burnout = 0:

	Частота	Процент
Analyst	393	21.00
Manager	391	20.90
HR	365	19.51
Engineer	361	19.29
Sales	361	19.29

burnout = 1:

	Частота	Процент
Sales	30	23.26
Manager	28	21.71
HR	26	20.16
Engineer	25	19.38
Analyst	20	15.50



Доля выгорания по должностям:

- Sales:  $30/391 = 7.67\%$
- Manager:  $28/419 = 6.68\%$
- HR:  $26/391 = 6.65\%$
- Engineer:  $25/386 = 6.48\%$
- Analyst:  $20/413 = 4.84\%$

HR-инсайт: Отдел продаж требует особого внимания по профилактике выгорания.

`age_bin`

- `age_bin` — возрастные группы

```
In [67]: plot_cat_feature_by_target(df, feature='age_bin',
                                 title='Распределение возрастных групп по выгоранию')
```

Общее распределение признака 'age\_bin':

	Частота	Процент
Senior (41-50)	527	26.35
Mid-Age (31-40)	504	25.20
Young (22-30)	490	24.50
Pre-Retirement (51-60)	479	23.95

Распределение признака 'age\_bin' по значениям целевой переменной 'burnout':

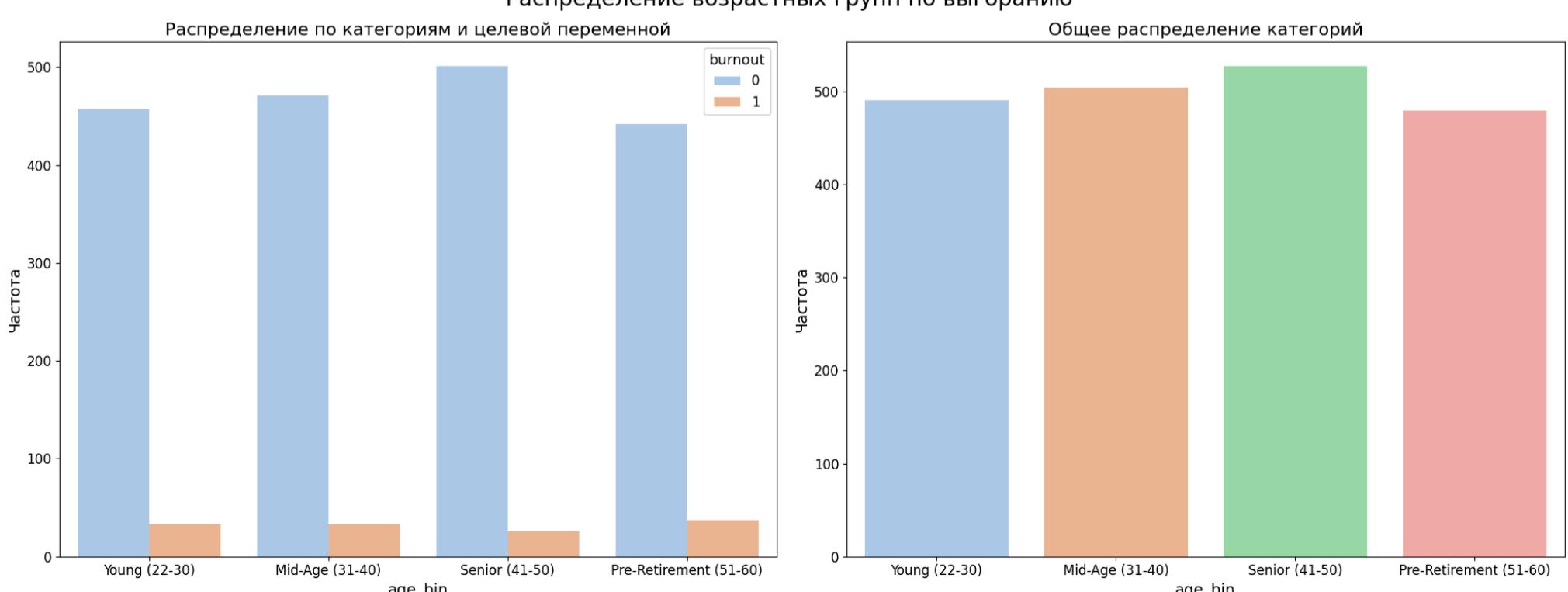
`burnout = 0:`

	Частота	Процент
Senior (41-50)	501	26.78
Mid-Age (31-40)	471	25.17
Young (22-30)	457	24.43
Pre-Retirement (51-60)	442	23.62

`burnout = 1:`

	Частота	Процент
Pre-Retirement (51-60)	37	28.68
Young (22-30)	33	25.58
Mid-Age (31-40)	33	25.58
Senior (41-50)	26	20.16

### Распределение возрастных групп по выгоранию



Интерпретация: Предпенсионный возраст (51-60) и молодые сотрудники (22-30) в группе повышенного риска.

## experience\_cat

- **experience\_cat** — категории стажа: Junior (0–3 года), Middle (4–7 лет), Senior (8–15 лет), Expert (16+ лет).

```
In [68]: plot_cat_feature_by_target(df, feature='experience_cat',
                                  title='Распределение категорий стажа по выгоранию')
```

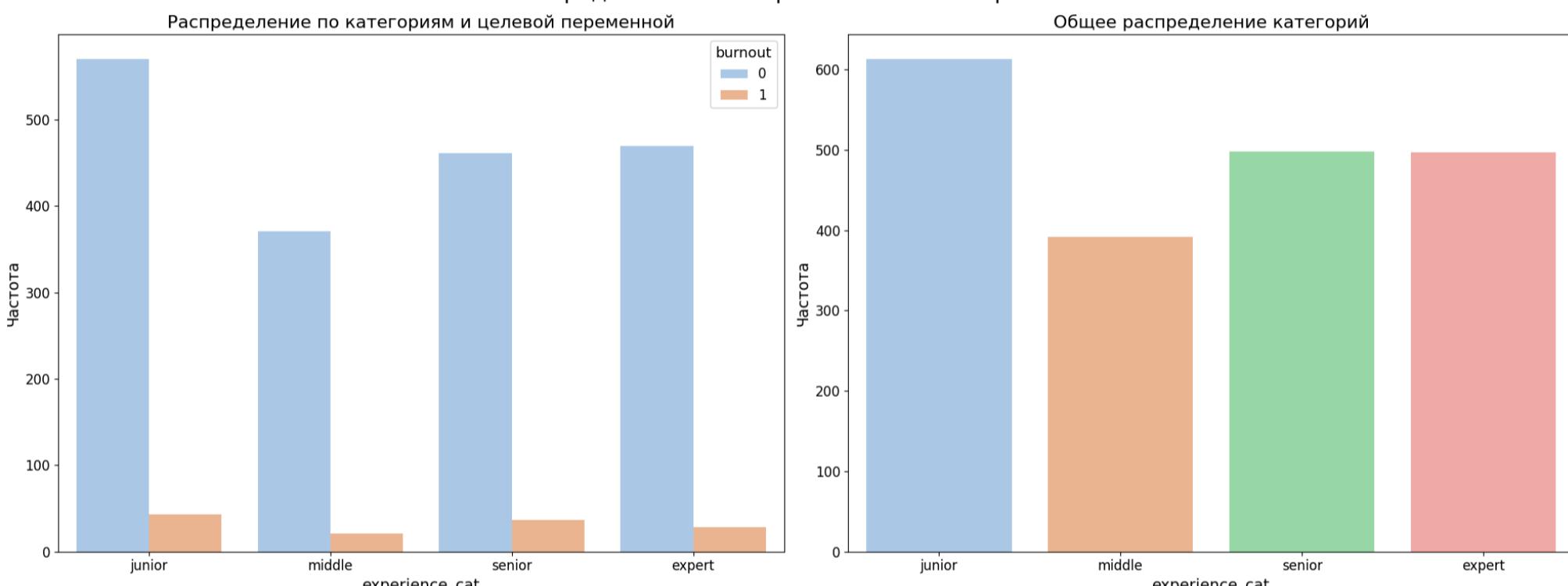
Общее распределение признака 'experience\_cat':

	Частота	Процент
junior	613	30.65
senior	498	24.90
expert	497	24.85
middle	392	19.60

Распределение признака 'experience\_cat' по значениям целевой переменной 'burnout':

	burnout = 0:	burnout = 1:
	Частота	Частота
junior	570	43
expert	469	37
senior	461	28
middle	371	21

Распределение категорий стажа по выгоранию



Парарадокс: Junior и Senior имеют наибольший риск, Middle - наименьший.

## work\_hours\_bin

- **work\_hours\_bin** — категории нагрузки: 30–40, 41–50, 51–60, 61–70 часов в неделю.

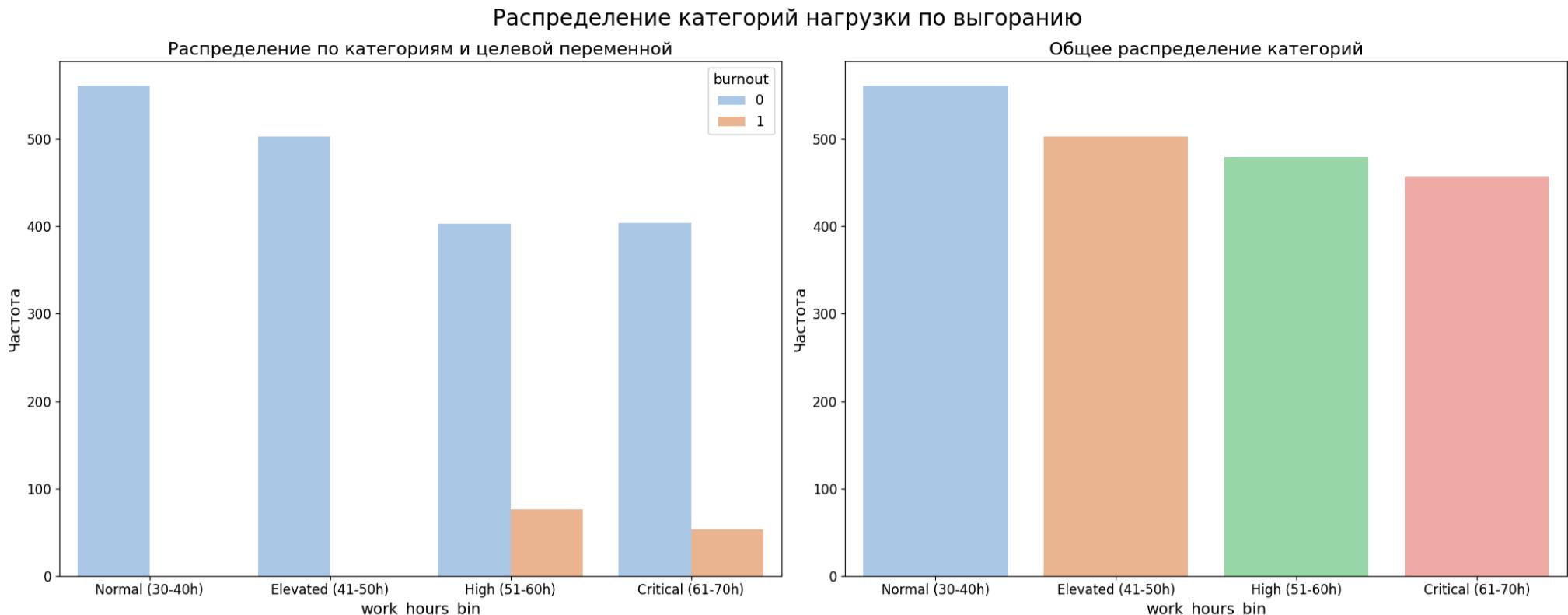
```
In [69]: plot_cat_feature_by_target(df, feature='work_hours_bin',
                                  title='Распределение категорий нагрузки по выгоранию')
```

Общее распределение признака 'work\_hours\_bin':

	Частота	Процент
Normal (30-40h)	561	28.05
Elevated (41-50h)	503	25.15
High (51-60h)	479	23.95
Critical (61-70h)	457	22.85

Распределение признака 'work\_hours\_bin' по значениям целевой переменной 'burnout':

	burnout = 0:	burnout = 1:
	Частота	Частота
Normal (30-40h)	561	76
Elevated (41-50h)	503	53
Critical (61-70h)	404	0
High (51-60h)	403	0



КРИТИЧЕСКОЕ НАБЛЮДЕНИЕ:

- Нет случаев выгорания при работе  $\leq 50$  часов
- Все случаи выгорания при работе  $\geq 51$  часа
- Наибольшая доля: 51-60 часов (58.91% случаев)

`remote_bin`

- `remote_bin` — группы удалёнки: office (0%), low (1–25%), medium (26–50%), high (51–75%), remote (76–100%).

```
In [70]: plot_cat_feature_by_target(df, feature='remote_bin',
                                  title='Распределение групп удаленки по выгоранию')
```

Общее распределение признака 'remote\_bin':

	Частота	Процент
low	517	25.85
medium	497	24.85
remote	491	24.55
high	482	24.10
office	13	0.65

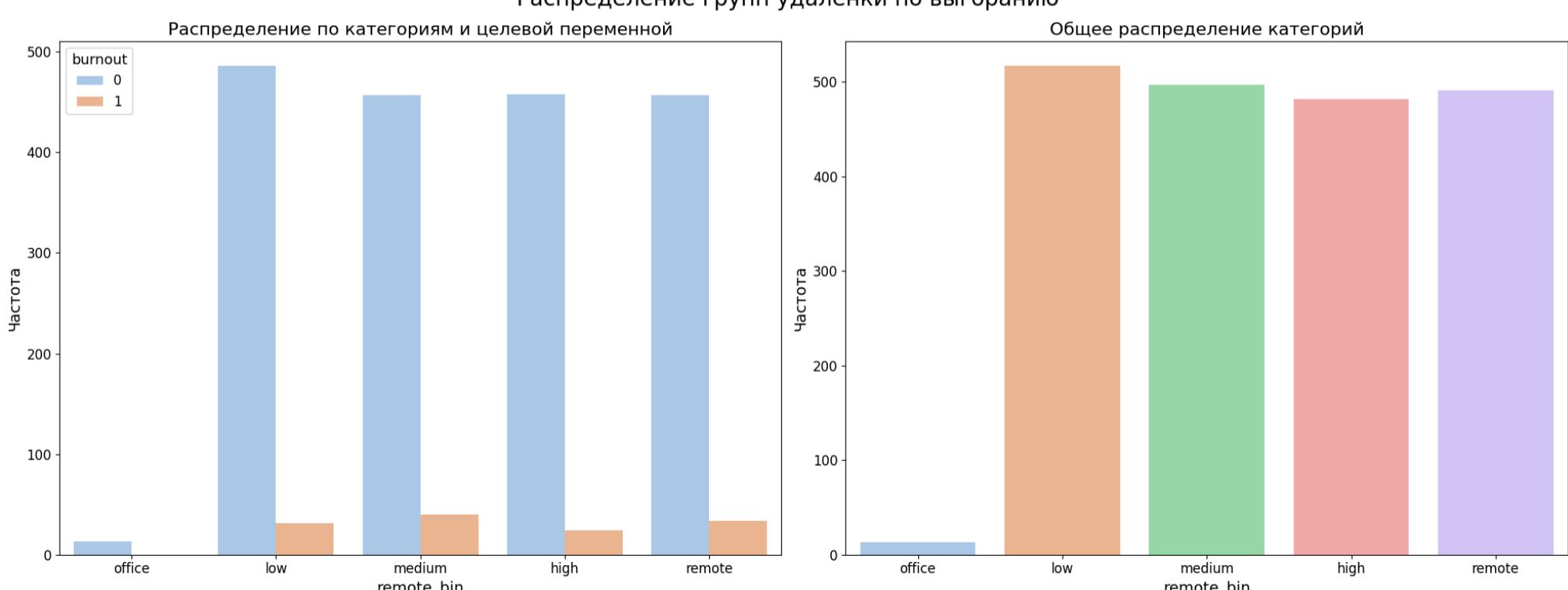
Распределение признака 'remote\_bin' по значениям целевой переменной 'burnout':

	burnout = 0:	Частота	Процент
low	486	25.98	
high	458	24.48	
medium	457	24.43	
remote	457	24.43	
office	13	0.69	

	burnout = 1:	Частота	Процент
medium	40	31.01	
remote	34	26.36	
low	31	24.03	
high	24	18.60	
office	0	0.00	

Распределение групп удаленки по выгоранию



`remote_cat`

- **remote\_cat** — классификация удалённой работы: onsite, hybrid, remote

```
In [71]: plot_cat_feature_by_target(df, feature='remote_cat',
                                 title='Распределение классификации удалённой работы по выгоранию')
```

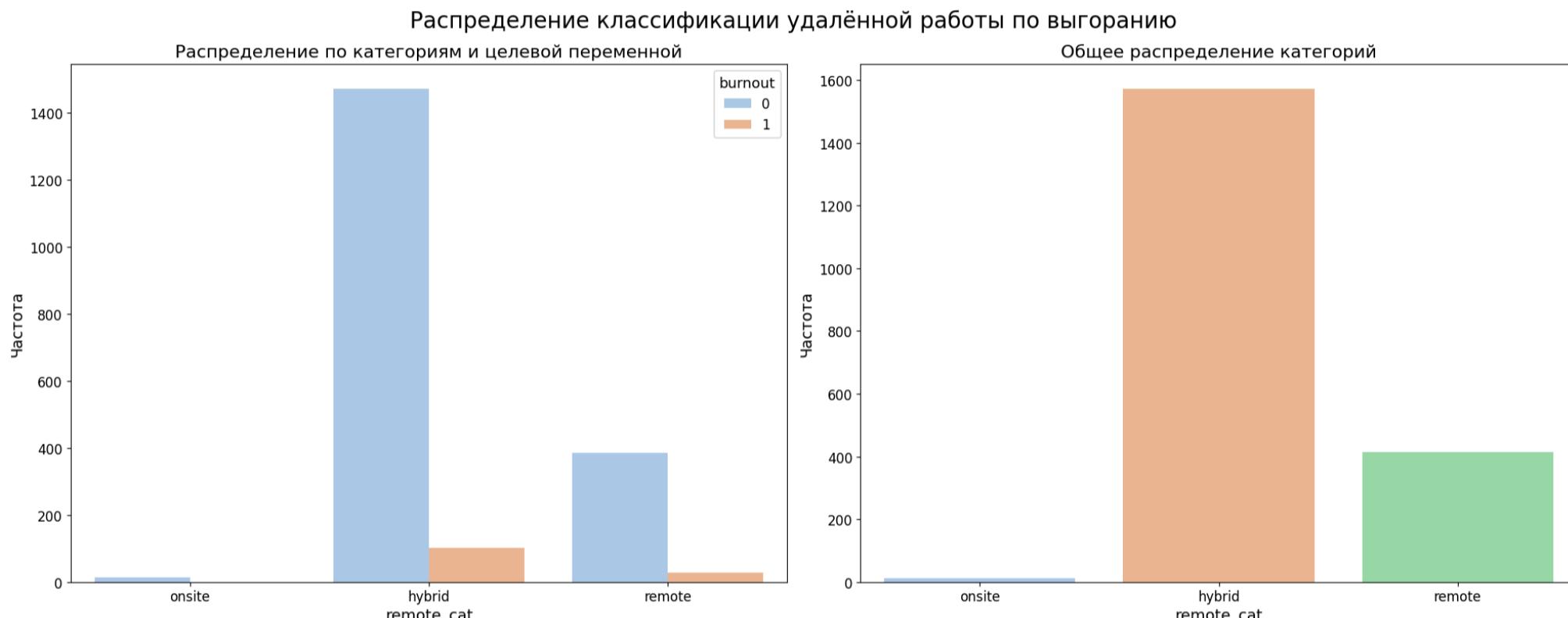
Общее распределение признака 'remote\_cat':

	Частота	Процент
hybrid	1573	78.65
remote	414	20.70
onsite	13	0.65

Распределение признака 'remote\_cat' по значениям целевой переменной 'burnout':

```
burnout = 0:
    Частота Процент
hybrid    1472  78.67
remote     386  20.63
onsite      13  0.69

burnout = 1:
    Частота Процент
hybrid    101  78.29
remote     28  21.71
onsite      0  0.00
```



Доля выгорания:

1. Remote: 28/414 = 6.76%
2. Hybrid: 101/1573 = 6.42%
3. Onsite: 0/13 = 0% (малая выборка)

Вывод: Удалённая работа не защищает от выгорания (риск сопоставим с гибридным режимом).

**stress\_cat**

- **stress\_cat** — категории стресса: низкий (1–3), средний (4–6), высокий (7–10).

```
In [72]: plot_cat_feature_by_target(df, feature='stress_cat',
                                 title='Распределение категорий стресса по выгоранию')
```

Общее распределение признака 'stress\_cat':

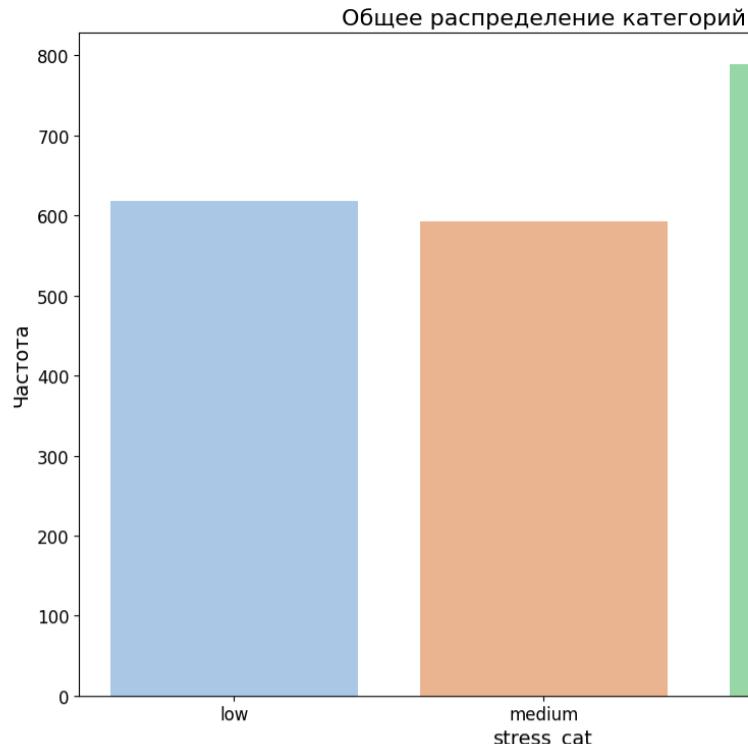
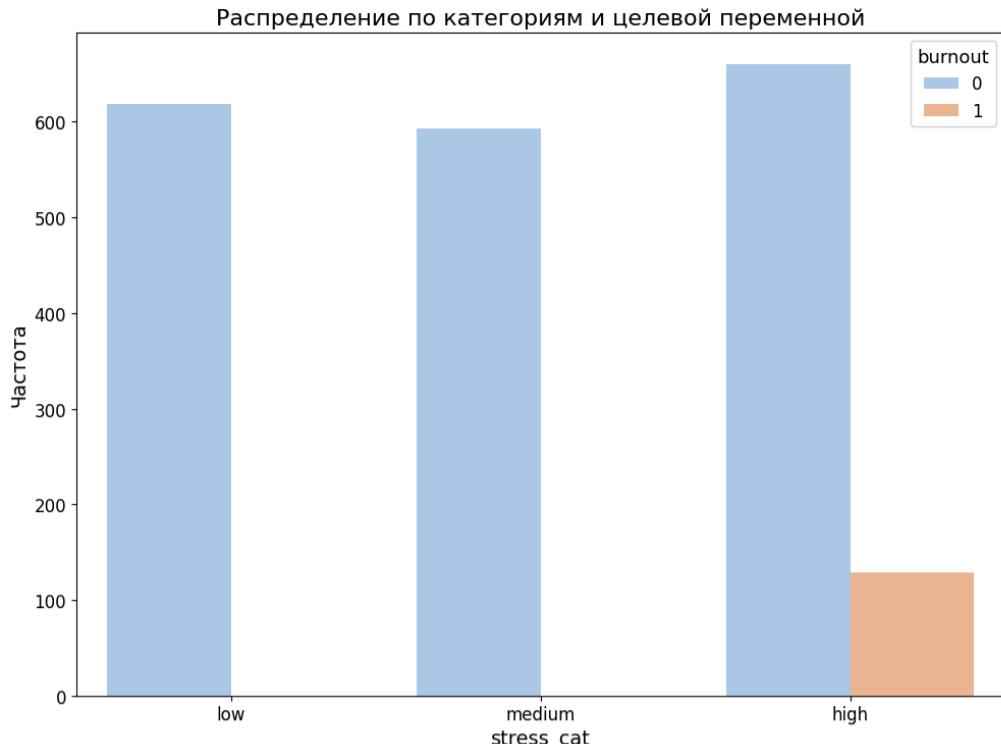
	Частота	Процент
high	789	39.45
low	618	30.90
medium	593	29.65

Распределение признака 'stress\_cat' по значениям целевой переменной 'burnout':

```
burnout = 0:
    Частота Процент
high      660  35.28
low       618  33.03
medium    593  31.69

burnout = 1:
    Частота Процент
high      129  100.0
low        0   0.0
medium    0   0.0
```

## Распределение категорий стресса по выгоранию



Все выгоревшие находятся в категории "высокий стресс"

Ни одного случая выгорания при низком/среднем стрессе

**satisfaction\_cat**

- **satisfaction\_cat** — категории удовлетворённости: низкая (1–2), средняя (2–3.5), высокая (3.6–5).

```
In [73]: plot_cat_feature_by_target(df, feature='satisfaction_cat',
                                 title='Распределение категорий удовлетворённости по выгоранию')
```

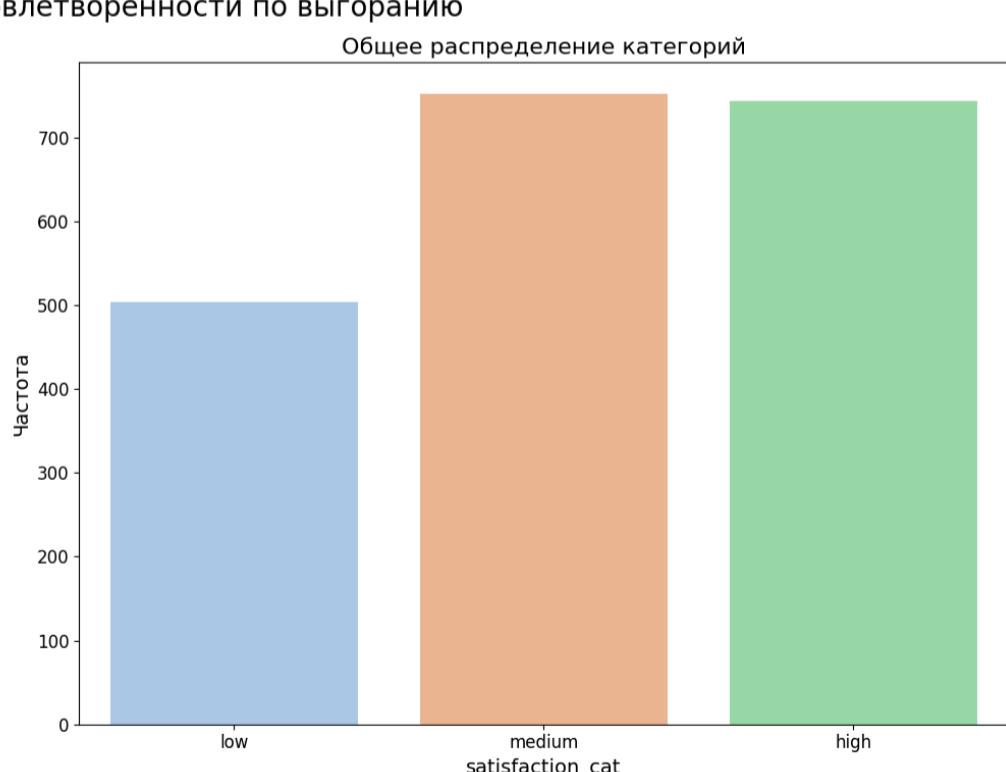
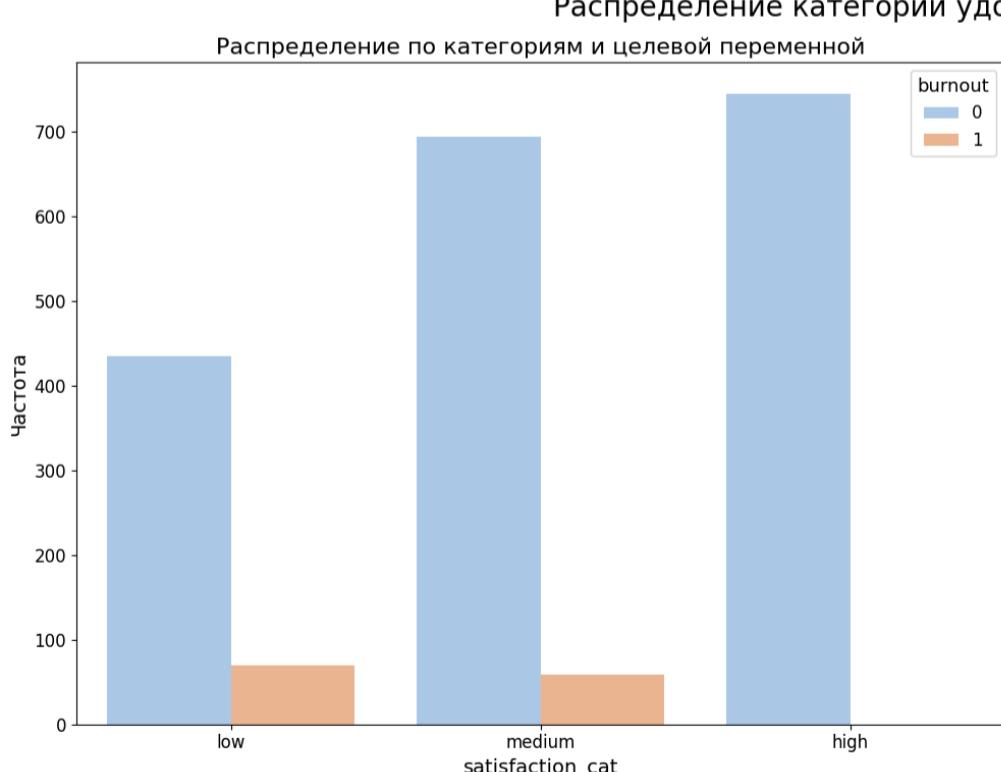
Общее распределение признака 'satisfaction\_cat':

	Частота	Процент
medium	752	37.6
high	744	37.2
low	504	25.2

Распределение признака 'satisfaction\_cat' по значениям целевой переменной 'burnout':

	Частота	Процент
burnout = 0:		
high	744	39.76
medium	693	37.04
low	434	23.20
burnout = 1:		
low	70	54.26
medium	59	45.74
high	0	0.00

## Распределение категорий удовлетворённости по выгоранию



Нет случаев выгорания при высокой удовлетворённости

Все выгоревшие имеют низкую или среднюю удовлетворённость

## Матрица корреляций(heatmap) для выявления зависимостей

```
In [74]: df.corr(numeric_only=True)
```

	age	experience	work_hours_per_week	remote_ratio	satisfaction_level	stress_level	stress_to_satisfaction	experience_to_age
age	1.000000	0.640340	-0.015810	-0.030909	-0.011196	-0.000385	0.001702	0.501452
experience	0.640340	1.000000	-0.019616	-0.042056	-0.008747	-0.014471	-0.019874	0.965955
work_hours_per_week	-0.015810	-0.019616	1.000000	0.039890	0.026100	-0.004843	-0.024805	-0.022233
remote_ratio	-0.030909	-0.042056	0.039890	1.000000	-0.056654	0.013422	0.039375	-0.032575
satisfaction_level	-0.011196	-0.008747	0.026100	-0.056654	1.000000	0.035406	-0.566246	-0.014384
stress_level	-0.000385	-0.014471	-0.004843	0.013422	0.035406	1.000000	0.691329	-0.022554
stress_to_satisfaction	0.001702	-0.019874	-0.024805	0.039375	-0.566246	0.691329	1.000000	-0.021199
experience_to_age	0.501452	0.965955	-0.022233	-0.032575	-0.014384	-0.022554	-0.021199	1.000000
career_start_age	0.611379	-0.216330	0.000114	0.004040	-0.005219	0.014419	0.022639	-0.357794
overload_index	-0.005250	-0.019247	0.391405	0.019609	0.053865	0.894277	0.603185	-0.026013
hours_above_45	0.002244	0.008830	0.842917	0.035346	0.024766	0.012199	-0.008047	0.003553
burnout	0.004042	-0.015270	0.225952	0.010859	-0.232607	0.321045	0.448836	-0.016817

In [75]: corr\_matrix = df.corr(numeric\_only=True)

```
# Корреляция burnout со всеми числовыми признаками
burnout_corr = corr_matrix['burnout'].drop('burnout') # убираем саму переменную
print("\nКорреляция burnout с числовыми признаками:")
print(burnout_corr.sort_values(ascending=False))
```

Корреляция burnout с числовыми признаками:

```
stress_to_satisfaction      0.448836
overload_index               0.432466
stress_level                 0.321045
work_hours_per_week          0.225952
hours_above_45                0.216410
career_start_age              0.020870
remote_ratio                  0.010859
age                           0.004042
experience                    -0.015270
experience_to_age             -0.016817
satisfaction_level            -0.232607
Name: burnout, dtype: float64
```

In [76]: corr = df.corr(numeric\_only=True)
pairs = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))
pairs[pairs > 0.5].stack().sort\_values(ascending=False)

Out[76]:

experience	experience_to_age	0.965955
stress_level	overload_index	0.894277
work_hours_per_week	hours_above_45	0.842917
stress_level	stress_to_satisfaction	0.691329
age	experience	0.640340
	career_start_age	0.611379
stress_to_satisfaction	overload_index	0.603185
age	experience_to_age	0.501452

dtype: float64

### СИЛЬНЫЕ МЕЖПРИЗНАКОВЫЕ КОРРЕЛЯЦИИ (>0.5):

- Критические пары (мультиколлинеарность):
  - Experience ↔ Experience\_to\_age: 0.966 (очень сильная!)
    - Один из признаков можно исключить
  - Stress\_level ↔ Overload\_index: 0.894
    - Overload\_index частично дублирует stress\_level
    - Но содержит дополнительную информацию (рабочие часы)
  - Work\_hours\_per\_week ↔ Hours\_above\_45: 0.843
    - Hours\_above\_45 - бинаризация work\_hours
    - Можно использовать только один
  - Stress\_level ↔ Stress\_to\_satisfaction: 0.691
    - Stress\_to\_satisfaction содержит информацию об удовлетворённости
    - Уникальный индикатор дисбаланса

In [77]: interval\_cols = df.select\_dtypes(include=['int64', 'float64']).columns.tolist()

```
# Вычисление Phik-матрицы (исключаем 'id')
phik_corr = df.phik_matrix(interval_cols=interval_cols)

# 📈 Корреляции с целевой переменной burnout
bpm_phik = phik_corr['burnout'].drop('burnout').sort_values(ascending=False)
print("📊 Phik-корреляции с целевой переменной 'burnout':")
print(bpm_phik)

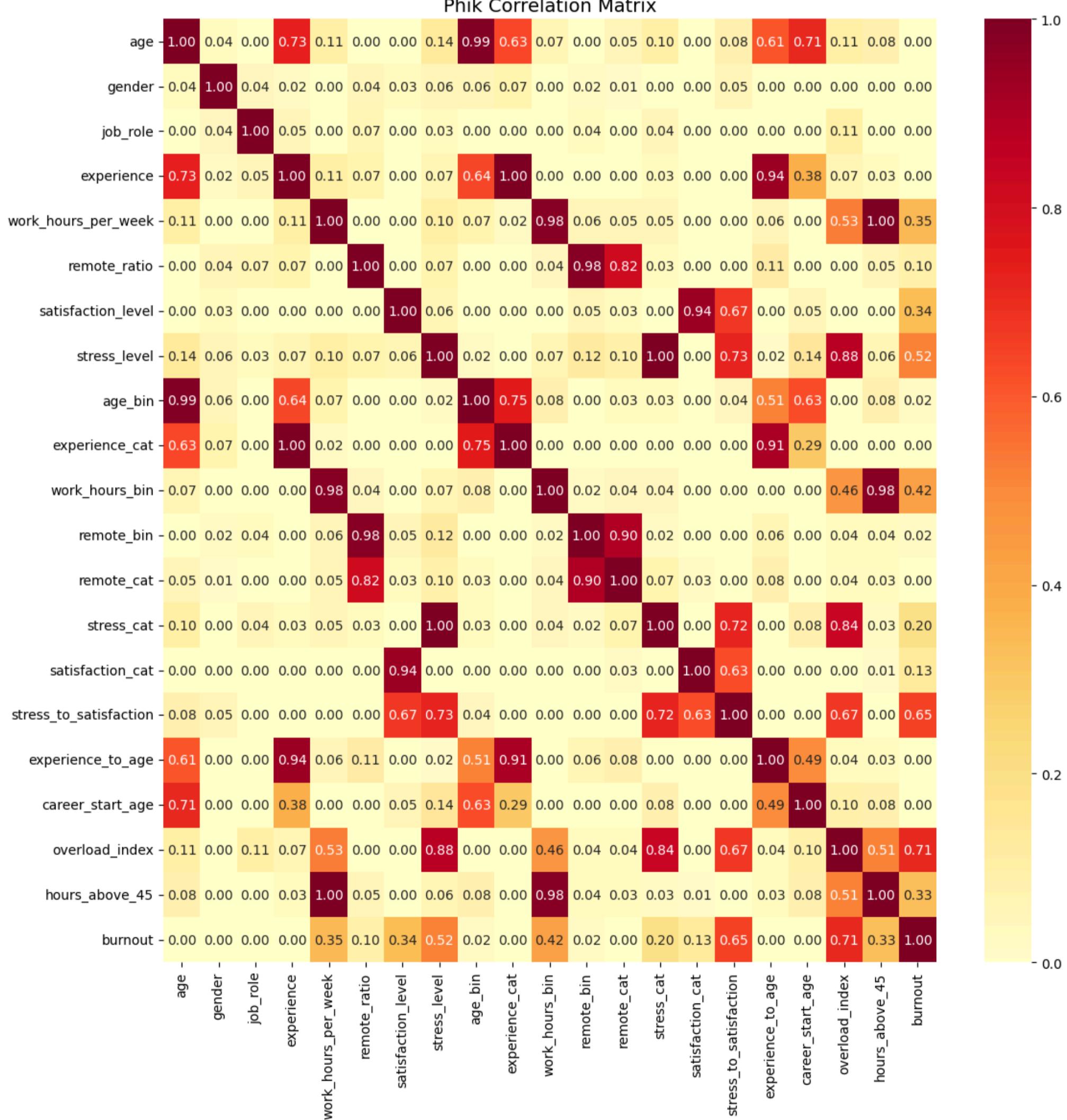
# 🔥 Визуализация полной Phik-матрицы
plt.figure(figsize=(12, 12))
sns.heatmap(phik_corr, annot=True, fmt=".2f", cmap='YlOrRd')
plt.title('Phik Correlation Matrix', fontsize=14)
```

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

PhiK-корреляции с целевой переменной 'burnout':

	overload_index	stress_to_satisfaction	stress_level	work_hours_bin	work_hours_per_week	satisfaction_level	hours_above_45	stress_cat	satisfaction_cat	remote_ratio	age_bin	remote_bin	experience_cat	gender	remote_cat	experience	experience_to_age	career_start_age	job_role	age	burnout
overload_index	0.712878																				
stress_to_satisfaction	0.652573	0.712878																			
stress_level	0.515859		0.652573																		
work_hours_bin	0.422204			0.515859																	
work_hours_per_week	0.347226				0.422204																
satisfaction_level	0.338775					0.347226															
hours_above_45	0.328699						0.338775														
stress_cat	0.197368							0.328699													
satisfaction_cat	0.134035								0.197368												
remote_ratio	0.095902									0.134035											
age_bin	0.020605										0.095902										
remote_bin	0.019079											0.020605									
experience_cat	0.000000											0.019079									
gender	0.000000												0.000000								
remote_cat	0.000000													0.000000							
experience	0.000000													0.000000							
experience_to_age	0.000000														0.000000						
career_start_age	0.000000															0.000000					
job_role	0.000000																0.000000				
age	0.000000																	0.000000			
burnout	0.000000																		0.000000		

Name: burnout, dtype: float64



Топ-5 корреляций с burnout (Phi-K):

- Overload\_index: 0.713 (очень сильная!) → главный фактор риска

- Stress\_to\_satisfaction: 0.653 (сильная) → дисбаланс стресс/удовлетворённость
- Stress\_level: 0.516 (сильная) → высокий стресс
- Work\_hours\_bin: 0.422 (умеренная) → переработки
- Work\_hours\_per\_week: 0.347 (умеренная) → переработки
- satisfaction\_level (-0.34) → низкая удовлетворённость.

Слабая или нулевая корреляция: возраст, стаж, пол, должность, удалёнка.

## АНАЛИЗ МУЛЬТИКОЛЛИНЕАРНОСТИ

```
In [78]: # Исключаем целевые колонки из матрицы
cols_to_exclude = ['burnout']
phik_corr_filtered = phik_corr.drop(index=cols_to_exclude, columns=cols_to_exclude, errors='ignore')

# Строим маску для верхнего треугольника с порогом корреляции
high_corr_mask = np.triu(np.abs(phik_corr_filtered)) > 0.5, k=1

# Выводим пары с высокой Phi-корреляцией
print("Пары признаков с Phi-корреляцией > 0.5 (без 'burnout':")
high_corr_pairs = phik_corr_filtered.where(high_corr_mask).stack().dropna().sort_values(ascending=False)
print(high_corr_pairs if not high_corr_pairs.empty else "Нет пар с высокой корреляцией.")
```

Пары признаков с Phi-корреляцией > 0.5 (без 'burnout':)

work_hours_per_week	hours_above_45	1.000000
experience	experience_cat	1.000000
stress_level	stress_cat	1.000000
age	age_bin	0.985349
remote_ratio	remote_bin	0.983069
work_hours_bin	hours_above_45	0.975675
work_hours_per_week	work_hours_bin	0.975594
satisfaction_level	satisfaction_cat	0.943874
experience	experience_to_age	0.942351
experience_cat	experience_to_age	0.909844
remote_bin	remote_cat	0.899628
stress_level	overload_index	0.882362
stress_cat	overload_index	0.837908
remote_ratio	remote_cat	0.822092
age_bin	experience_cat	0.752749
age	experience	0.731184
stress_level	stress_to_satisfaction	0.726624
stress_cat	stress_to_satisfaction	0.718589
age	career_start_age	0.711630
stress_to_satisfaction	overload_index	0.674411
satisfaction_level	stress_to_satisfaction	0.672601
experience	age_bin	0.640476
age	experience_cat	0.632891
age_bin	career_start_age	0.631709
satisfaction_cat	stress_to_satisfaction	0.629111
age	experience_to_age	0.613161
work_hours_per_week	overload_index	0.529936
age_bin	experience_to_age	0.514471
overload_index	hours_above_45	0.514328

## Попарные графики (pairplot) для количественных признаков

### Общий pairplot для всех значений целевой переменной

```
In [79]: # Все числовые признаки кроме burnout
numeric_cols = [col for col in df.select_dtypes(include='number').columns if col != 'burnout']

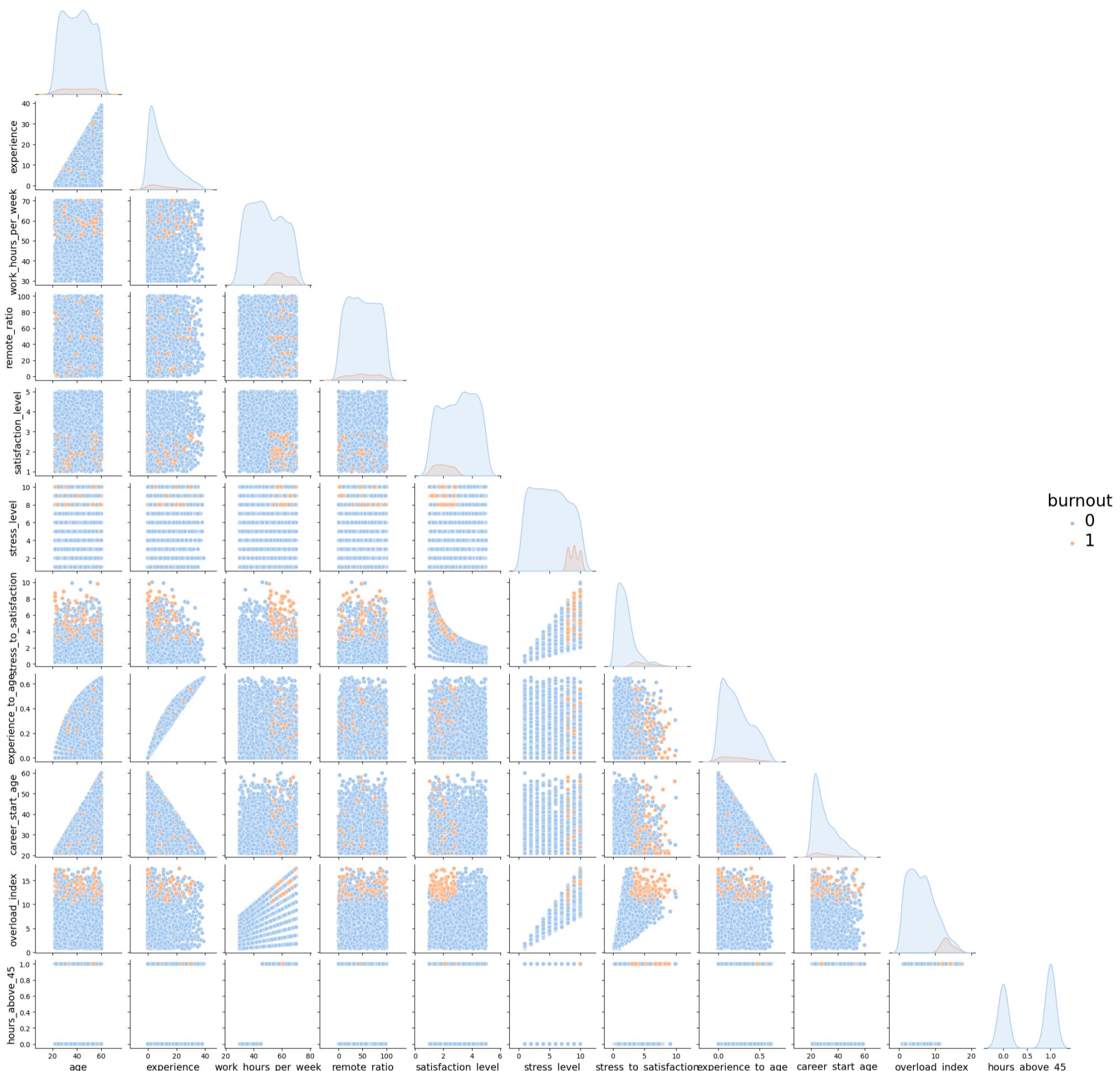
# PairPlot с заливкой по burnout
g = sns.pairplot(df[numeric_cols + ['burnout']], hue='burnout', palette='pastel',
                  diag_kind='kde', height=2, corner=True)

# Увеличиваем подписи признаков
for ax in g.axes.flatten():
    if ax is not None:
        ax.set_xlabel(ax.get_xlabel(), fontsize=14)
        ax.set_ylabel(ax.get_ylabel(), fontsize=14)

# Увеличиваем легенду
plt.setp(g._legend.get_texts(), fontsize=24)
g._legend.set_title('burnout')
g._legend.get_title().set_fontsize(24)

plt.suptitle('Попарные диаграммы (числовые признаки без burnout)', fontsize=24, y=1.02)
plt.show()
```

# Попарные диаграммы (числовые признаки без burnout)



Кластер выгорания на pairplots характеризуется:

- Work\_hours\_per\_week: 51-70 часов (правая часть распределения)
- Stress\_level: 8-10 (верхняя часть распределения)
- Satisfaction\_level: 1.0-3.0 (левая часть распределения)
- overload\_index  $\geq 12$
- stress\_to\_satisfaction  $\geq 4$

**Отдельные pairplot для каждого значения целевой переменной**

```
In [80]: # список числовых признаков без burnout
numeric_cols = [col for col in df.select_dtypes(include='number').columns if col != 'burnout']

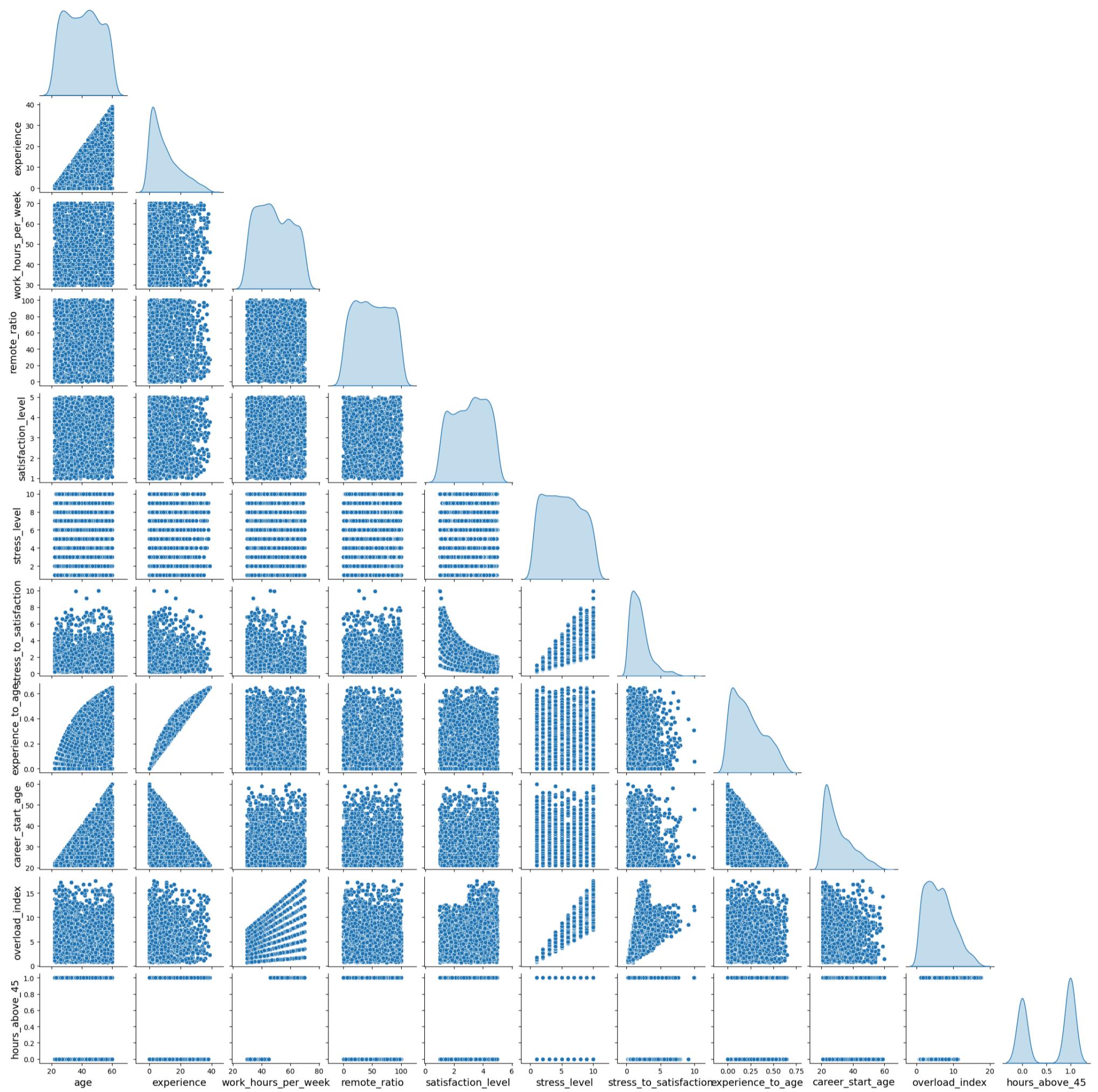
label_size = 14

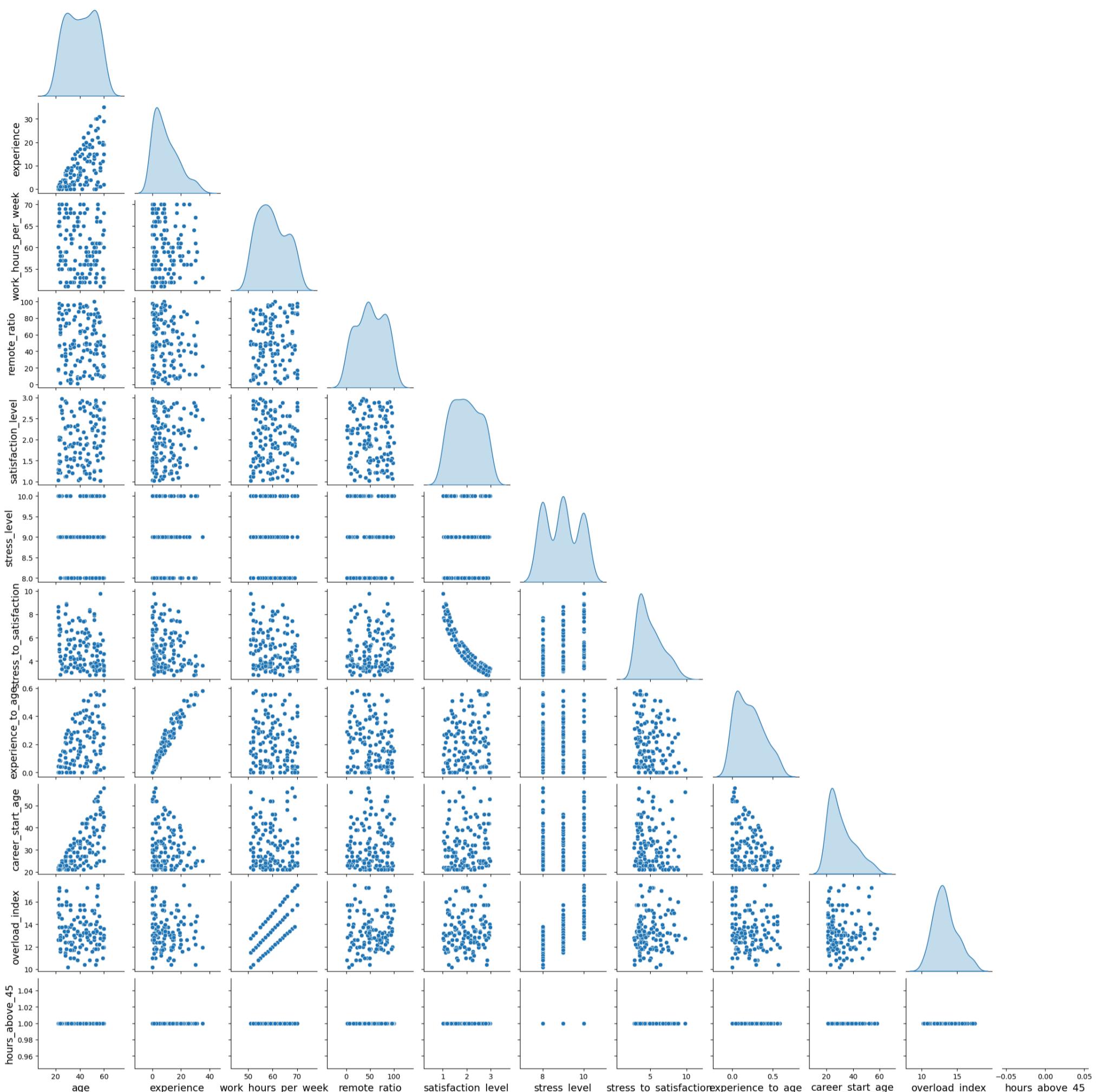
for val in sorted(df['burnout'].dropna().unique()):
    subset = df[df['burnout'] == val][numeric_cols]
    g = sns.pairplot(subset, diag_kind='kde', height=2, corner=True)

    # увеличиваем подписи осей
    for ax in g.axes.flatten():
        if ax is not None:
            ax.set_xlabel(ax.get_xlabel(), fontsize=label_size)
            ax.set_ylabel(ax.get_ylabel(), fontsize=label_size)

    plt.suptitle(f'Попарные диаграммы (только числовые признаки, burnout = {val})',
                fontsize=24, y=1.02)
    plt.show()
```

## Попарные диаграммы (только числовые признаки, burnout = 0)





## Графики между целевой переменной и всеми остальными числовыми признаками

```
In [81]: import seaborn as sns
import matplotlib.pyplot as plt

def plot_target_vs_numeric(df, target='burnout', height=4, label_size=14, point_size=60):
    # список числовых признаков без целевой переменной
    numeric_cols = [col for col in df.select_dtypes(include='number').columns if col != target]

    for col in numeric_cols:
        plt.figure(figsize=(15, height))
        sns.scatterplot(x=target, y=col, data=df,
                        hue=target, palette='pastel',
                        s=point_size, alpha=0.7)

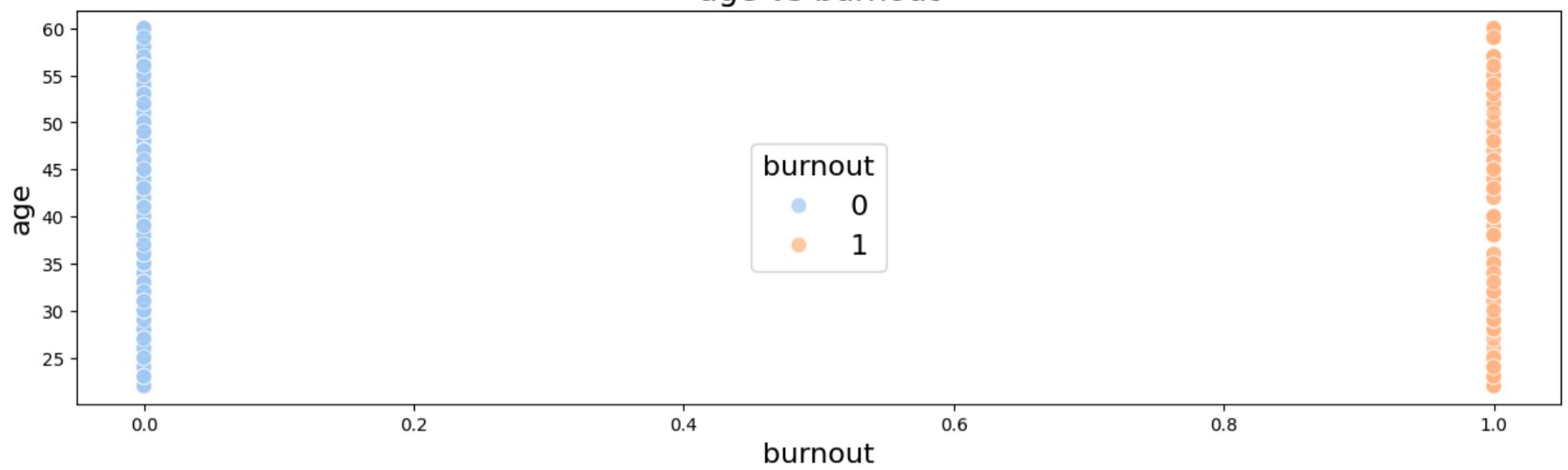
        plt.title(f'{col} vs {target}', fontsize=label_size+2)
        plt.xlabel(target, fontsize=label_size)
        plt.ylabel(col, fontsize=label_size)

        # легенда в центре графика
        plt.legend(title=target, fontsize=label_size, title_fontsize=label_size,
                  loc='center', bbox_to_anchor=(0.5, 0.5))

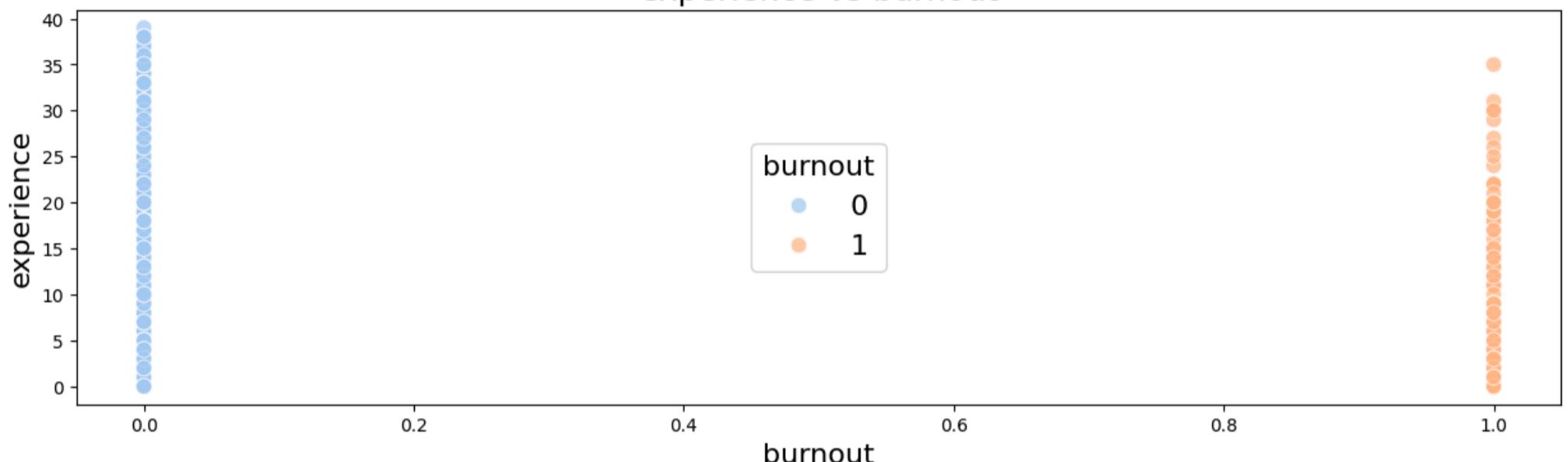
    plt.show()
```

```
In [82]: plot_target_vs_numeric(df, target='burnout',
                           height=4, label_size=16, point_size=80)
```

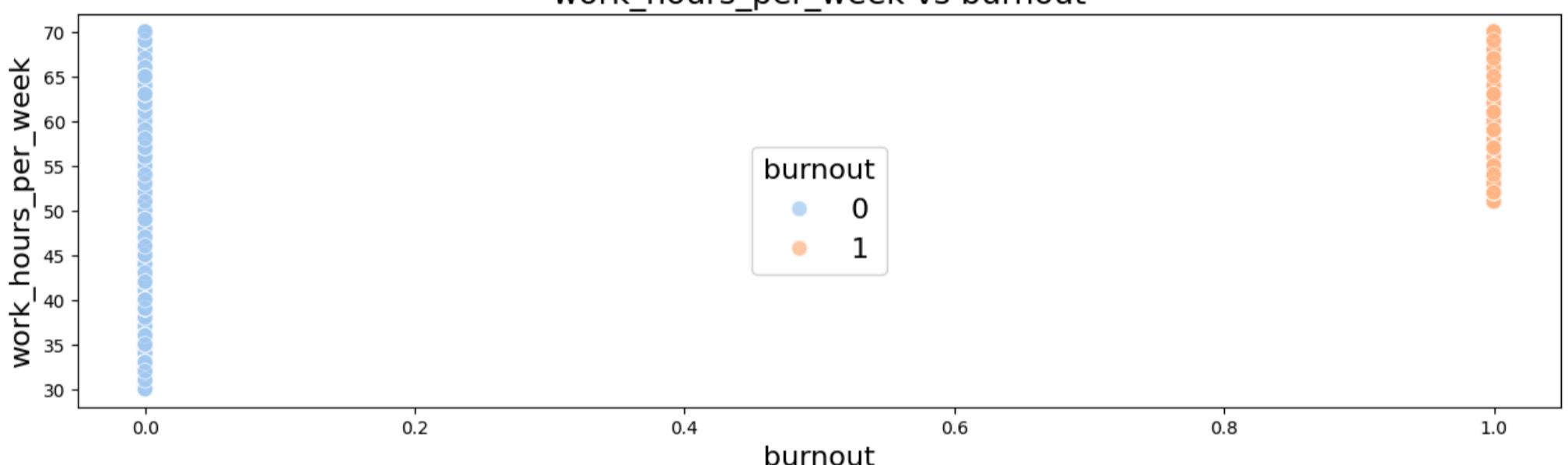
age vs burnout



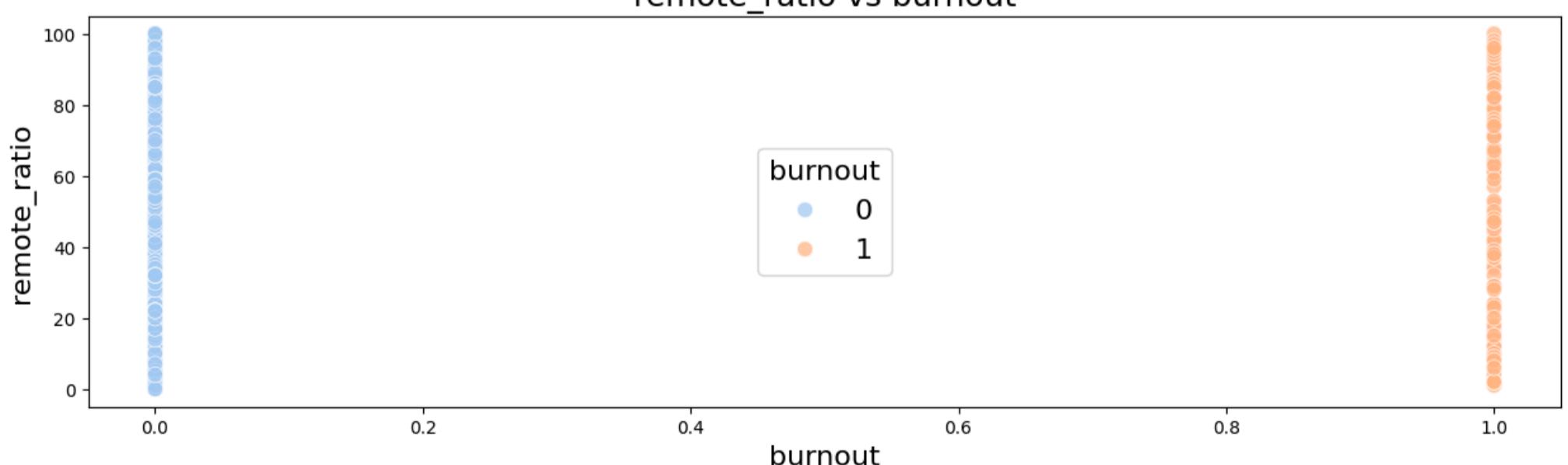
experience vs burnout



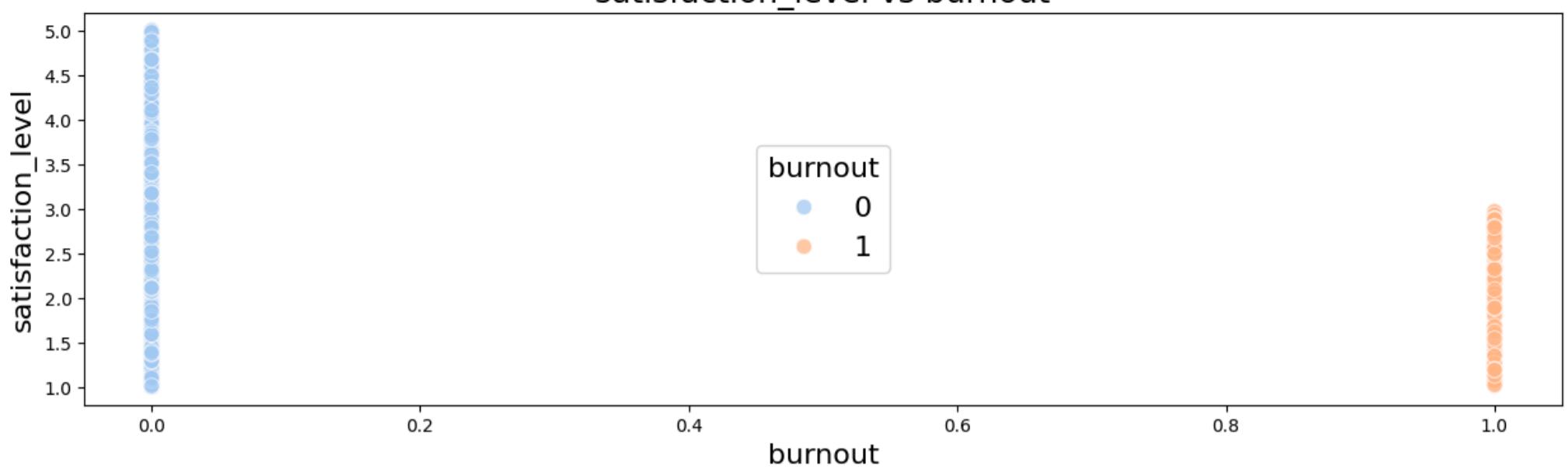
work\_hours\_per\_week vs burnout



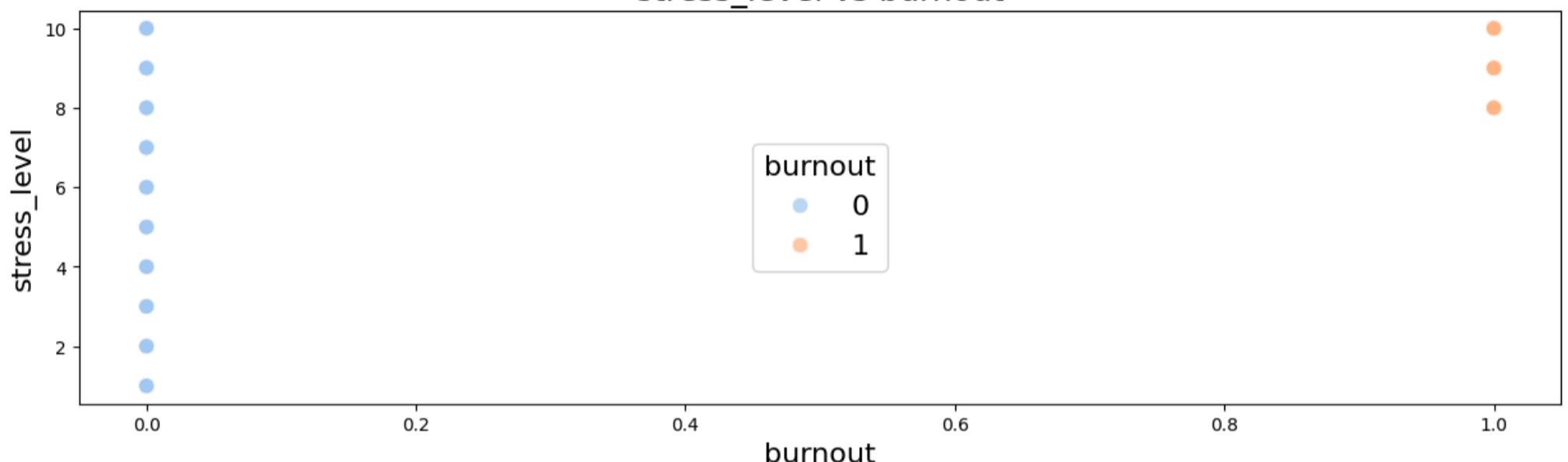
remote\_ratio vs burnout



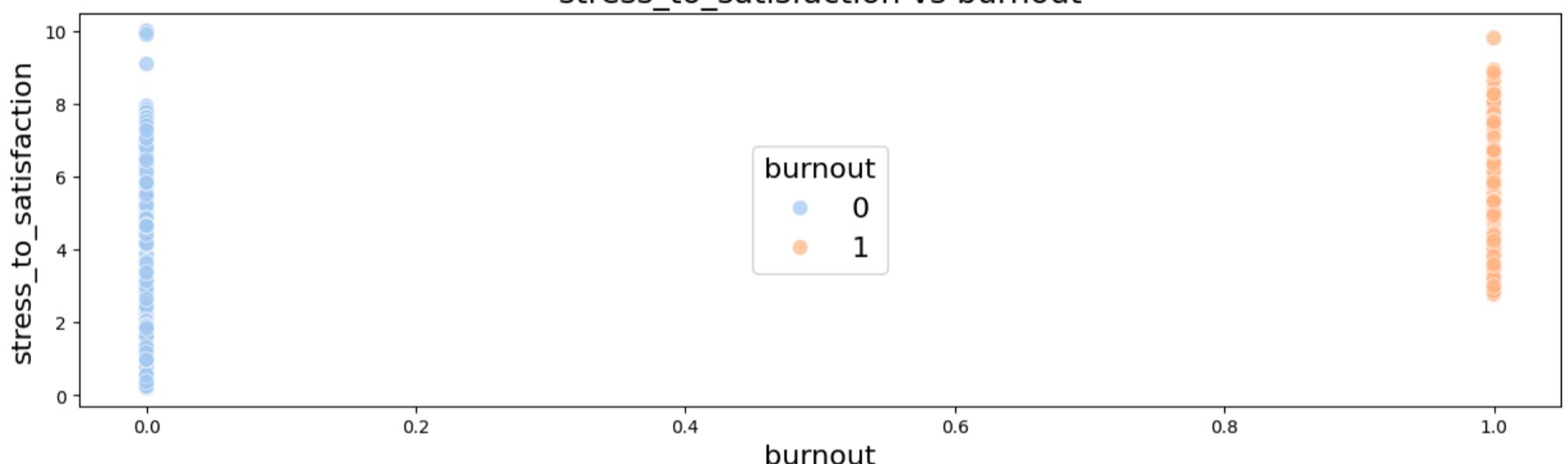
satisfaction\_level vs burnout



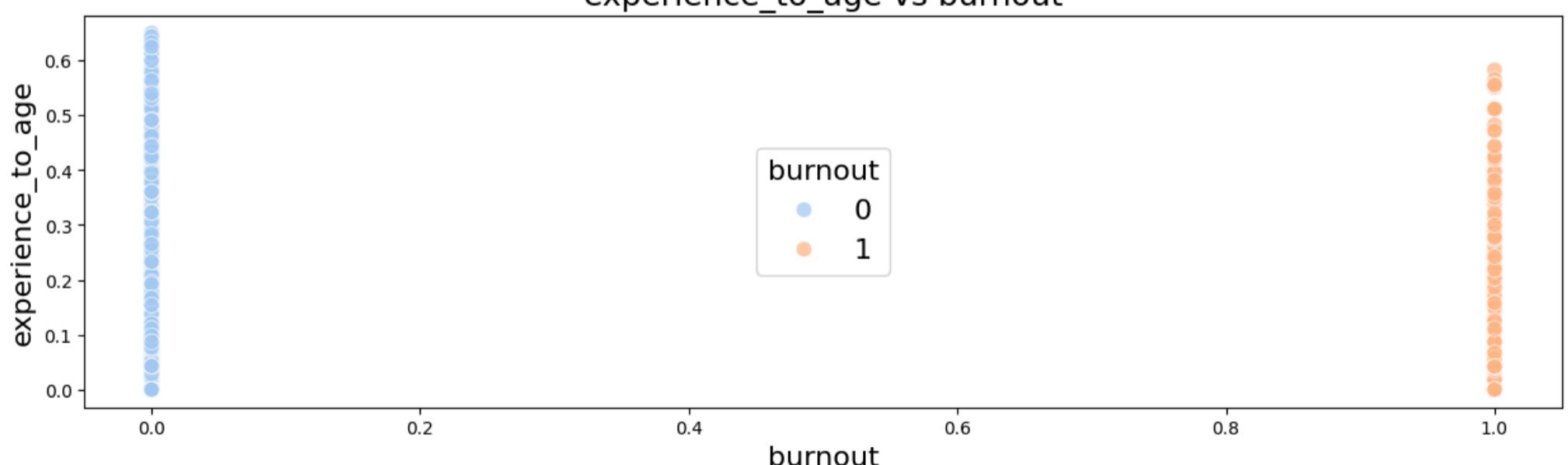
stress\_level vs burnout

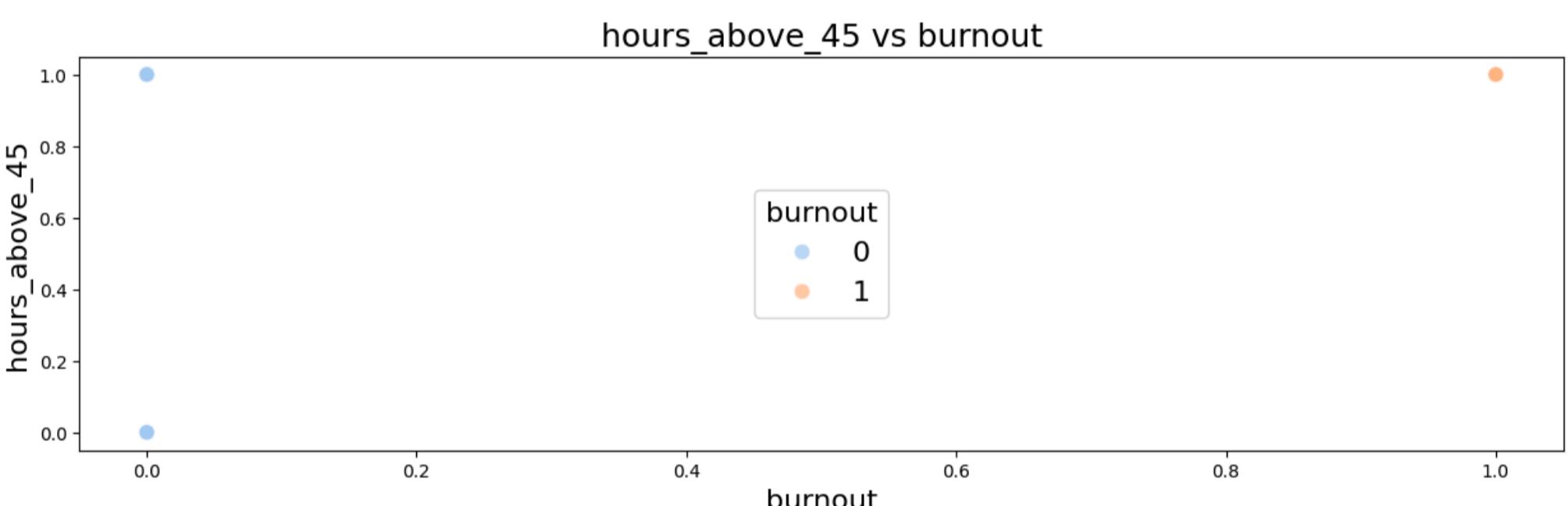
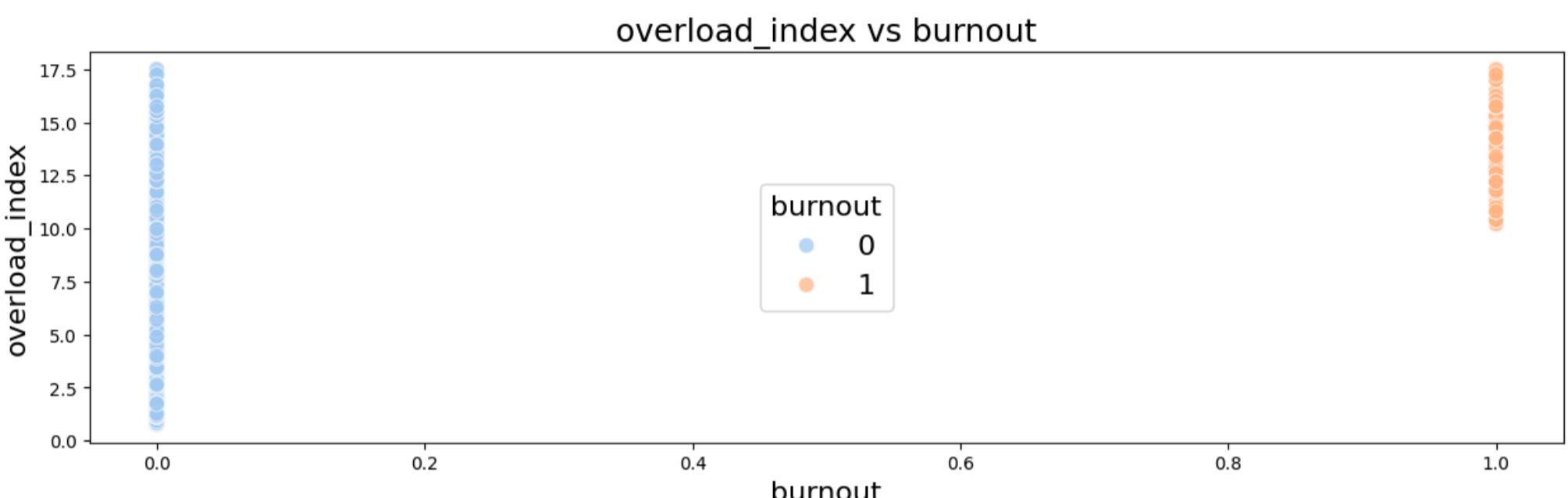
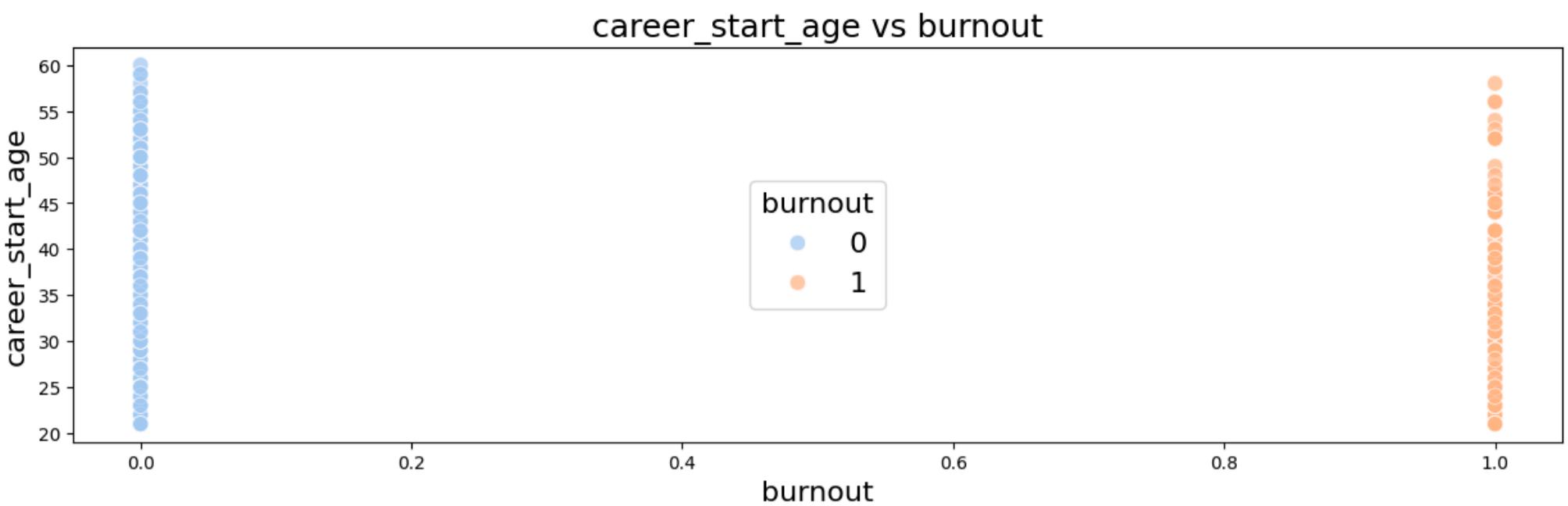


stress\_to\_satisfaction vs burnout



experience\_to\_age vs burnout





## Группировки и агрегации

### По должностям

```
In [83]: print("\n==== Анализ по должностям (JobRole) ====")

# Средние значения по должностям
grouped_job = df.groupby('job_role').agg({
    'burnout': ['mean', 'median'],
    'work_hours_per_week': 'mean',
    'stress_level': 'mean',
    'satisfaction_level': 'mean'
}).round(3)

print("\nСредние значения по должностям:")
display(grouped_job)

# Топ-5 должностей с наибольшей долей выгорания
burnout_by_job = df.groupby('job_role')['burnout'].mean().sort_values(ascending=False)
print("\nТОП-5 должностей с наибольшей долей выгорания:")
display(burnout_by_job.head(5))

# Топ-5 должностей с наибольшим уровнем стресса
stress_by_job = df.groupby('job_role')['stress_level'].mean().sort_values(ascending=False)
print("\nТОП-5 должностей с наибольшим уровнем стресса:")
display(stress_by_job.head(5))

==== Анализ по должностям (JobRole) ====

Средние значения по должностям:
```

Средние значения по должностям:

	<b>burnout</b>	<b>work_hours_per_week</b>	<b>stress_level</b>	<b>satisfaction_level</b>
	<b>mean</b>	<b>median</b>	<b>mean</b>	<b>mean</b>
<b>job_role</b>				
<b>Analyst</b>	0.048	0.0	49.453	5.252
<b>Engineer</b>	0.065	0.0	49.972	5.337
<b>HR</b>	0.066	0.0	49.742	5.524
<b>Manager</b>	0.067	0.0	49.310	5.470
<b>Sales</b>	0.077	0.0	49.496	5.583
ТОП-5 должностей с наибольшей долей выгорания:				

job\_role

Sales 0.076726

Manager 0.066826

HR 0.066496

Engineer 0.064767

Analyst 0.048426

Name: burnout, dtype: float64

ТОП-5 должностей с наибольшим уровнем стресса:

job\_role

Sales 5.583120

HR 5.524297

Manager 5.470167

Engineer 5.336788

Analyst 5.251816

Name: stress\_level, dtype: float64

Ранжирование по доле выгорания:

- Sales: 7.67% (наибольший риск)
- Analyst: 4.84% (наименьший риск)

Анализ причин для Sales:

- Высокий стресс (5.58 - максимальный среди должностей)
- Высокая доля выгорания при средних рабочих часах (49.5)

Рекомендация для Sales:

- Пересмотреть систему KPI
- Внедрить stress-management программы
- Ротация между отделами продаж

## По полу

In [84]: print("\n==== Анализ по полу (Gender) ====")

```
burnout_by_gender = df.groupby('gender')['burnout'].mean()
print("\nДоля выгорания по полу:")
display(burnout_by_gender)

# Средние характеристики по полу
gender_stats = df.groupby('gender').agg({
    'work_hours_per_week': 'mean',
    'stress_level': 'mean',
    'satisfaction_level': 'mean',
    'remote_ratio': 'mean'
}).round(2)

print("\nСредние характеристики по полу:")
display(gender_stats)
```

==== Анализ по полу (Gender) ====

Доля выгорания по полу:

gender

Female 0.062436

Male 0.066471

Name: burnout, dtype: float64

Средние характеристики по полу:

	<b>work_hours_per_week</b>	<b>stress_level</b>	<b>satisfaction_level</b>	<b>remote_ratio</b>
--	----------------------------	---------------------	---------------------------	---------------------

gender

<b>Female</b>	49.52	5.35	2.99	50.78
---------------	-------	------	------	-------

<b>Male</b>	49.65	5.51	3.00	49.21
-------------	-------	------	------	-------

Средние характеристики:

- Рабочие часы: почти одинаковы (49.65 vs 49.52)
- Стress: мужчины выше (5.51 vs 5.35)
- Удовлетворённость: почти одинаковы (3.00 vs 2.99)
- Удалённая работа: женщины больше (50.78% vs 49.21%)

## По категории рабочего времени

```
In [85]: print("\n==== Анализ по категориям рабочих часов ===")  
  
if 'work_hours_bin' in df.columns:  
    burnout_by_hours = df.groupby('work_hours_bin')['burnout'].mean()  
    print("\nДоля выгорания по категориям рабочих часов:")  
    display(burnout_by_hours)  
  
    # Средние стресс и удовлетворённость по категориям часов  
    hours_stats = df.groupby('work_hours_bin').agg({  
        'stress_level': 'mean',  
        'satisfaction_level': 'mean'  
    }).round(2)  
  
    print("\nСредние стресс и удовлетворённость по категориям часов:")  
    display(hours_stats)
```

==== Анализ по категориям рабочих часов ===

Доля выгорания по категориям рабочих часов:

```
work_hours_bin  
Normal (30-40h)      0.000000  
Elevated (41-50h)    0.000000  
High (51-60h)        0.158664  
Critical (61-70h)    0.115974  
Name: burnout, dtype: float64
```

Средние стресс и удовлетворённость по категориям часов:

	stress_level	satisfaction_level
--	--------------	--------------------

work_hours_bin	stress_level	satisfaction_level
Normal (30-40h)	5.39	2.96
Elevated (41-50h)	5.48	2.98
High (51-60h)	5.54	3.01
Critical (61-70h)	5.31	3.04

Анализ порога:

- 51+ часов - порог риска
- Наибольшая доля выгорания в категории 51-60 часов (15.57%)

Стресс и удовлетворённость по категориям часов:

- Стресс: максимален в High (51-60h): 5.54
- Удовлетворённость: растёт с часами (парадокс!)

Гипотеза: Сотрудники, работающие много, могут быть более вовлечёнными/амбициозными, что повышает удовлетворённость, но не защищает от выгорания.

## По категории удаленно работы

```
In [86]: print("\n==== Анализ по категориям удалённой работы ===")  
  
if 'remote_cat' in df.columns:  
    burnout_by_remote = df.groupby('remote_cat')['burnout'].mean()  
    print("\nДоля выгорания по категориям удалённой работы:")  
    display(burnout_by_remote)  
  
    # Средний стресс по типам удалёнки  
    remote_stats = df.groupby('remote_cat').agg({  
        'stress_level': 'mean',  
        'satisfaction_level': 'mean',  
        'work_hours_per_week': 'mean'  
    }).round(2)  
  
    print("\nСредние показатели по типам удалённой работы:")  
    display(remote_stats)
```

==== Анализ по категориям удалённой работы ===

Доля выгорания по категориям удалённой работы:

```
remote_cat  
onsite    0.000000  
hybrid    0.064209  
remote    0.067633  
Name: burnout, dtype: float64
```

Средние показатели по типам удалённой работы:

	stress_level	satisfaction_level	work_hours_per_week
remote_cat			
onsite	4.54	3.14	54.62
hybrid	5.43	3.02	49.27
remote	5.46	2.89	50.63

Средние показатели:

- Стресс: максимален у Remote (5.46), минимален у Onsite (4.54)
- Удовлетворённость: максимальна у Onsite (3.14), минимальна у Remote (2.89)
- Рабочие часы: максимальны у Onsite (54.62), минимальны у Hybrid (49.27)

Ключевой инсайт:

- Onsite имеет наибольшие рабочие часы (54.62), но нулевое выгорание (в выборке)
- Remote имеет высокий стресс и низкую удовлетворённость при средних часах

📌 Инсайт:

- Полная удалёнка связана с повышенным стрессом, несмотря на комфортность работы.
- Офис → самый низкий стресс, но самые высокие часы (возможно, правила учета).

Это важная HR-находка: удалёнка не гарантирует снижение риска, эффект зависит от организации процессов.

## По стажу и возрасту

```
In [87]: print("\n==== Анализ по стажу и возрасту ===")
```

```
# По стажу
if 'experience_cat' in df.columns:
    burnout_by_exp = df.groupby('experience_cat')['burnout'].mean()
    print("\nДоля выгорания по категориям стажа:")
    display(burnout_by_exp)

# По возрастным группам
if 'age_bin' in df.columns:
    burnout_by_age = df.groupby('age_bin')['burnout'].mean()
    print("\nДоля выгорания по возрастным группам:")
    display(burnout_by_age)
```

==== Анализ по стажу и возрасту ===

Доля выгорания по категориям стажа:

```
experience_cat
junior    0.070147
middle    0.053571
senior    0.074297
expert     0.056338
Name: burnout, dtype: float64
```

Доля выгорания по возрастным группам:

```
age_bin
Young (22-30)          0.067347
Mid-Age (31-40)        0.065476
Senior (41-50)         0.049336
Pre-Retirement (51-60)  0.077244
Name: burnout, dtype: float64
```

У-образная кривая:

- Высокий риск у начинающих (Junior) и опытных (Senior)
- Низкий риск в середине карьеры (Middle)

Гипотеза:

- Junior: адаптационный стресс
- Senior: профессиональное выгорание, стагнация
- Middle: оптимальный баланс опыта и энергии

Инсайт: Предпенсионный возраст - группа максимального риска!

Возможные причины для Pre-Retirement:

- Возрастная усталость
- Ощущение "застоя"
- Физиологические изменения
- Давление молодых коллег

📌 Инсайт:

- Риск выше у молодых сотрудников (мало опыта) и у старших (предпенсионная нагрузка).
- Средний возраст и опыт — наиболее защищённая группа.

## Пороговый анализ

```
In [88]: print("\n==== Пороговый анализ ===")  
  
# Функция для анализа порогов  
def threshold_analysis(df, feature, thresholds, target='burnout'):  
    results = []  
    for i in range(len(thresholds) - 1):  
        lower = thresholds[i]  
        upper = thresholds[i + 1]  
        mask = (df[feature] >= lower) & (df[feature] < upper)  
        burnout_rate = df.loc[mask, target].mean()  
        count = mask.sum()  
        results.append({  
            'Диапазон': f'{lower}-{upper}',  
            'Количество': count,  
            'Доля выгорания': round(burnout_rate, 3)  
        })  
    return pd.DataFrame(results)  
  
# Пороговый анализ для рабочих часов  
print("\nПороговый анализ для рабочих часов (work_hours_per_week):")  
hour_thresholds = [30, 40, 45, 50, 55, 60, 65, 70]  
hour_analysis = threshold_analysis(df, 'work_hours_per_week', hour_thresholds)  
display(hour_analysis)  
  
# Пороговый анализ для уровня стресса  
print("\nПороговый анализ для уровня стресса (stress_level):")  
stress_thresholds = [1, 3, 5, 7, 9, 11]  
stress_analysis = threshold_analysis(df, 'stress_level', stress_thresholds)  
display(stress_analysis)  
  
# Пороговый анализ для удовлетворённости  
print("\nПороговый анализ для уровня удовлетворённости (satisfaction_level):")  
satisfaction_thresholds = [1, 2, 3, 4, 5]  
satisfaction_analysis = threshold_analysis(df, 'satisfaction_level', satisfaction_thresholds)  
display(satisfaction_analysis)  
  
# Визуализация порогового анализа  
fig, axes = plt.subplots(1, 3, figsize=(18, 5))  
  
# Рабочие часы  
sns.lineplot(data=hour_analysis, x='Диапазон', y='Доля выгорания', ax=axes[0], marker='o')  
axes[0].set_title('Риск выгорания по рабочим часам')  
axes[0].set_ylabel('Доля выгорания')  
axes[0].tick_params(axis='x', rotation=45)  
  
# Стесс  
sns.lineplot(data=stress_analysis, x='Диапазон', y='Доля выгорания', ax=axes[1], marker='o', color='red')  
axes[1].set_title('Риск выгорания по уровню стресса')  
axes[1].set_ylabel('Доля выгорания')  
axes[1].tick_params(axis='x', rotation=45)  
  
# Удовлетворённость  
sns.lineplot(data=satisfaction_analysis, x='Диапазон', y='Доля выгорания', ax=axes[2], marker='o', color='green')  
axes[2].set_title('Риск выгорания по уровню удовлетворённости')  
axes[2].set_ylabel('Доля выгорания')  
axes[2].tick_params(axis='x', rotation=45)  
  
plt.tight_layout()  
plt.show()  
  
# Вывод критических порогов  
print("\n==== КРИТИЧЕСКИЕ ПОРОГИ ===")  
print("1. Рабочие часы > 50: резкий рост выгорания")  
print("2. Уровень стресса > 7: высокий риск выгорания")  
print("3. Уровень удовлетворённости < 2.5: повышенный риск выгорания")
```

==== Пороговый анализ ===

Пороговый анализ для рабочих часов (work\_hours\_per\_week):

Диапазон Количество Доля выгорания

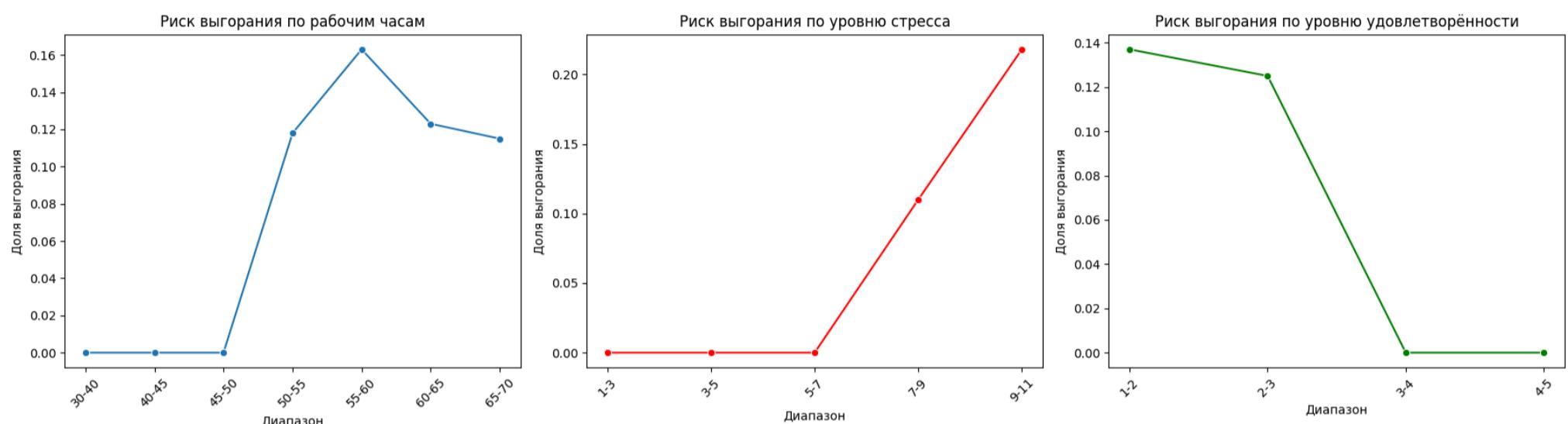
0	30-40	504	0.000
1	40-45	255	0.000
2	45-50	258	0.000
3	50-55	229	0.118
4	55-60	252	0.163
5	60-65	220	0.123
6	65-70	234	0.115

Пороговый анализ для уровня стресса (stress\_level):

Диапазон	Количество	Доля выгорания	
0	1-3	419	0.000
1	3-5	399	0.000
2	5-7	393	0.000
3	7-9	399	0.110
4	9-11	390	0.218

Пороговый анализ для уровня удовлетворённости (satisfaction\_level):

Диапазон	Количество	Доля выгорания	
0	1-2	497	0.137
1	2-3	489	0.125
2	3-4	513	0.000
3	4-5	497	0.000



#### ==== КРИТИЧЕСКИЕ ПОРОГИ ====

1. Рабочие часы > 50: резкий рост выгорания
2. Уровень стресса > 7: высокий риск выгорания
3. Уровень удовлетворённости < 2.5: повышенный риск выгорания

#### Рабочие часы в неделю (work\_hours\_per\_week)

- Пороги: 30–40, 40–45, 45–50, 50–55, 55–60, 60–65, 65–70 часов.
- Наблюдения:
  - До 50 часов: выгорание отсутствует (0% во всех трёх интервалах).
  - После 50 часов: резкий скачок:
    - 50–55 часов: 11.8% выгорания.
    - 55–60 часов: 16.3% (пиковое значение).
    - 60–70 часов: стабилизация на уровне ~12%.

Интерпретация:

- 50 часов в неделю — критический порог, после которого риск выгорания становится значимым.
- 55–60 часов — наиболее опасный диапазон.
- После 60 часов рост не продолжается — возможно, работает эффект отбора (остаются наиболее устойчивые сотрудники).

#### Уровень стресса (stress\_level)

- Пороги: 1–3, 3–5, 5–7, 7–9, 9–11.
- Наблюдения:
  - До уровня 7: выгорание отсутствует (0%).
  - 7–9: 11% выгорания.
  - 9–11: 21.8% (каждый пятый сотрудник).

Интерпретация:

- Уровень стресса 7 — критический порог.
- При стрессе выше 9 риск выгорания удваивается.
- Сильная нелинейная зависимость: до порога 7 стресс не приводит к выгоранию, после — риск растёт экспоненциально.

#### Уровень удовлетворённости (satisfaction\_level)

- Пороги: 1–2, 2–3, 3–4, 4–5.
- Наблюдения:
  - 1–2 (низкая удовлетворённость): 13.7% выгорания.
  - 2–3: 12.5% (незначительное снижение).
  - 3–5 (удовлетворённость выше среднего): 0% выгорания.

Интерпретация:

- Уровень удовлетворённости ниже 3 — зона риска.

- Критический порог ~2.5: ниже этого значения риск стабильно высокий.
- Удовлетворённость — защитный фактор: при уровне выше 3 выгорание практически отсутствует.

### Выводы по шагу 3

#### Целевая переменная Burnout

- Датасет сильно несбалансирован:
  - 93.5% сотрудников без выгорания.
  - 6.5% сотрудников с выгоранием.
- Это требует акцента на метриках Recall, PR-AUC

#### Средние значения:

- Рабочие часы: 49.6 часов/неделю (превышение нормы на 9.6 часов).
- Стресс: 5.43/10 (умеренный).
- Удовлетворённость: 2.99/5 (ниже нейтрального уровня 3.0).
- Удалёнка: 49.97% (преобладает гибридный формат).

#### Ключевые количественные факторы

- Рабочие часы:
  - До 50 часов — Burnout отсутствует.
  - 50 часов — резкий рост риска (до 16%).
- Стресс:
  - До 7 баллов — Burnout отсутствует.
  - 7 баллов — риск резко возрастает (до 22%).
- Удовлетворённость:
  - меньше 2.5 баллов — высокий риск (12–14%).
  - ≥3 баллов — Burnout практически отсутствует.
- Индекс перегрузки: Burnout связан с высокими значениями (>12).
- Отношение стресс/удовлетворённость: Burnout появляется при значениях >4.

#### Категориальные признаки

- Должности:
  - Наибольший риск у Sales, Manager, HR.
  - Минимальный риск у Analyst.
- Пол: различия минимальны (Male ~6.6%, Female ~6.2%).
- Удалёнка: onsite = 0% Burnout, hybrid/remote ~6–7%.
- Стаж:
  - Junior (0–3 года) — риск выше (~7%).
  - Middle/Senior — риск ниже (~5%).
- Возраст: значимых различий нет.

#### Корреляционный анализ

- Наибольшая связь с Burnout:
  - overload\_index (0.71) → комбинация часов и стресса
  - stress\_to\_satisfaction (0.65) → стресс превышает удовлетворённость в 5 раз у выгоревших
  - stress\_level (0.52) → все выгоревшие имеют стресс ≥8
  - work\_hours\_bin (0.42)
  - work\_hours\_per\_week: 0.348 → все выгоревшие работают >50 часов
  - satisfaction\_level (0.34)

#### Факторы, НЕ влияющие напрямую на выгорание

- Пол — различия незначимы.
- Возраст (как непрерывный признак) — распределения почти идентичны.
- Стаж (как непрерывный признак) — слабое влияние.
- Удалённая работа (remote\_ratio, remote\_cat) — корреляция близка к нулю, риск сопоставим в гибридном и удалённом форматах.

#### Критические пороги

- Рабочие часы > 50 → резкий рост Burnout, пик в диапазоне 55–60 часов (16.3%)
- Стресс > 7 → высокий риск, при >9 баллов риск удваивается (21.8%)
- Удовлетворённость меньше 3 → повышенный риск, при меньше 2.5 баллов риск стабильно высокий (12–14%)

#### Значимые паттерны

- Соотношение стресс/удовлетворённость:
  - У выгоревших в среднем 5 раз выше
  - Минимальный порог для выгорания: 2.78
- Должности с повышенным риском:
  - Продажи: 7.67% случаев выгорания
  - Менеджеры: 6.68% случаев выгорания

- HR: 6.65% случаев выгорания
- Возрастные группы риска:
  - Предпенсионный возраст (51-60 лет)
  - Молодые сотрудники (22-30 лет)

Рекомендации для HR

- Мониторинг нагрузки:
  - Установить лимит в 50 рабочих часов
  - Отслеживать сотрудников с нагрузкой >45 часов
- Контроль стресса:
  - Внедрить систему раннего выявления стресса  $\geq 8$  баллов
  - Особое внимание сотрудникам с низким уровнем удовлетворённости
- Целевые группы:
  - Отделы продаж и менеджмент
  - Сотрудники предпенсионного возраста
  - Молодые специалисты
- Метрики для отслеживания:
  - Индекс перегрузки  $>10.2$
  - Соотношение стресс/удовлетворённость  $>2.78$
  - Уровень удовлетворённости меньше 3.0

\* к содержанию

## Шаг 4: Составление профилей сотрудников

### Сравнительные профили по средним значениям

```
In [89]: profile_stats = df.groupby('burnout').agg({
    'age': 'mean',
    'experience': 'mean',
    'work_hours_per_week': 'mean',
    'remote_ratio': 'mean',
    'satisfaction_level': 'mean',
    'stress_level': 'mean',
    'stress_to_satisfaction': 'mean',
    'overload_index': 'mean'
}).round(2)

print("Сравнительные профили сотрудников (Burnout=0 vs Burnout=1):")
display(profile_stats)
```

Сравнительные профили сотрудников (Burnout=0 vs Burnout=1):

	age	experience	work_hours_per_week	remote_ratio	satisfaction_level	stress_level	stress_to_satisfaction	overload_index
burnout								
0	40.68	10.11	48.89	49.89	3.07	5.19	1.98	6.27
1	40.87	9.54	59.77	51.18	1.97	8.95	4.98	13.36

```
In [90]: # Список ключевых признаков для сравнения
features = [
    'work_hours_per_week',
    'stress_level',
    'satisfaction_level',
    'overload_index',
    'stress_to_satisfaction'
]

# Группировка
profile_means = df.groupby('burnout')[features].mean()

# Построение bar-chart
plt.figure(figsize=(15, 4))

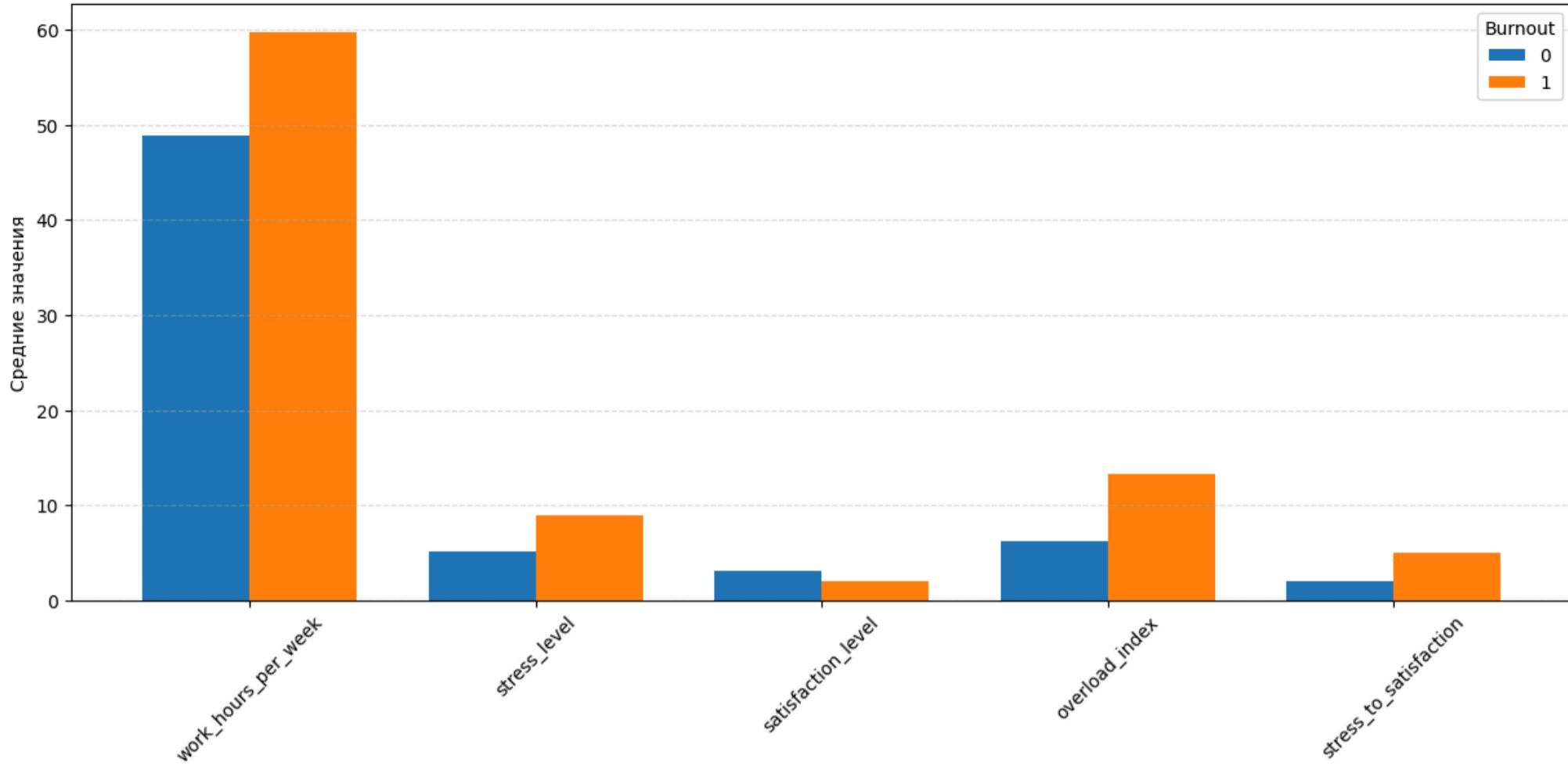
profile_means.T.plot(
    kind='bar',
    figsize=(15, 6),
    width=0.75
)

plt.title('Портреты сотрудников: Burnout=0 vs Burnout=1')
plt.ylabel('Средние значения')
plt.xticks(rotation=45)
plt.legend(title='Burnout')
plt.grid(axis='y', linestyle='--', alpha=0.4)

plt.show()
```

<Figure size 1500x400 with 0 Axes>

### Портреты сотрудников: Burnout=0 vs Burnout=1

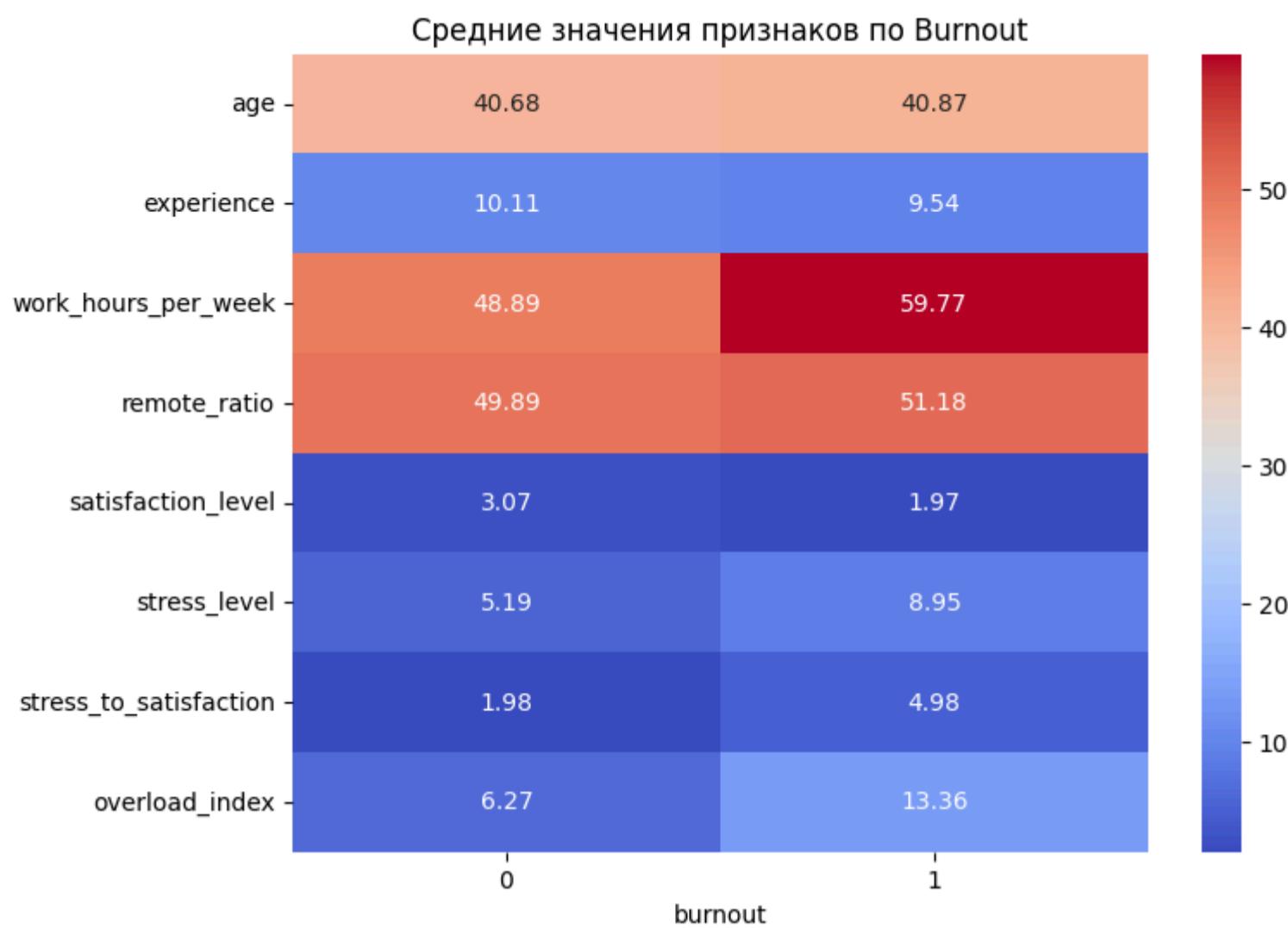


- Рабочие часы work\_hours\_per\_week: +10.88 ч
  - Ключевой фактор: все выгоревшие работают >50 ч
- Уровень стресса stress\_level: +3.76 балла
  - Все выгоревшие имеют стресс ≥8.
- Уровень удовлетворённости satisfaction\_level: -1.10 балла
  - Все выгоревшие имеют удовлетворённость <3.
- и Индекс перегрузки overload\_index: ×2.13
  - Критическая перегрузка у выгоревших.
- Дисбаланс стресс/удовлетворённость stress\_to\_satisfaction: ×2.52
  - У выгоревших стресс превышает удовлетворённость в 5 раз.
- Удалённая работа remote\_ratio: +1.29%
  - Незначимое влияние.
- Возраст age: +0.19 лет
  - Без различий.
- Стаж experience: -0.57 лет
  - Незначимое снижение.

📌 Вывод: переход от Burnout=0 к Burnout=1 — это скачок сразу по нескольким метрикам нагрузки, а не по одной.

### Heatmap для сравнения средних значений

```
In [91]: plt.figure(figsize=(8,6))
sns.heatmap(profile_stats.T, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Средние значения признаков по Burnout")
plt.show()
```



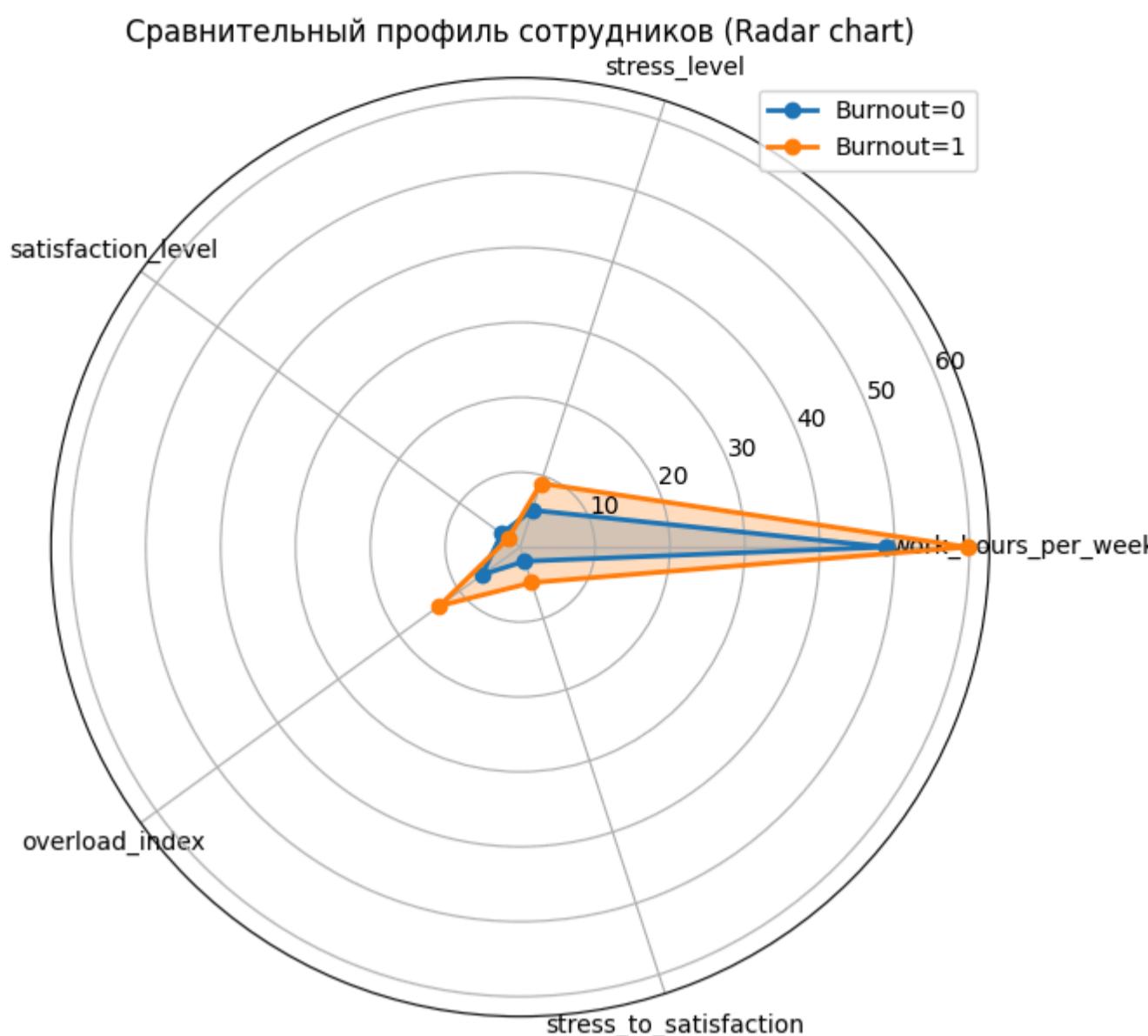
Heatmap подтверждает значительные различия по числовым признакам.

### Radar chart (паучья диаграмма) для ключевых метрик

```
In [92]: metrics = ['work_hours_per_week', 'stress_level', 'satisfaction_level', 'overload_index', 'stress_to_satisfaction']
values_0 = profile_stats.loc[0, metrics].values
values_1 = profile_stats.loc[1, metrics].values

labels = metrics
angles = np.linspace(0, 2*np.pi, len(labels), endpoint=False).tolist()
values_0 = np.concatenate((values_0, [values_0[0]]))
values_1 = np.concatenate((values_1, [values_1[0]]))
angles += angles[:1]

fig, ax = plt.subplots(figsize=(7,7), subplot_kw=dict(polar=True))
ax.plot(angles, values_0, 'o-', linewidth=2, label='Burnout=0')
ax.fill(angles, values_0, alpha=0.25)
ax.plot(angles, values_1, 'o-', linewidth=2, label='Burnout=1')
ax.fill(angles, values_1, alpha=0.25)
ax.set_thetagrids(np.degrees(angles[:-1]), labels)
plt.legend(loc='upper right')
plt.title("Сравнительный профиль сотрудников (Radar chart)")
plt.show()
```



Radar chart наглядно показывает разрыв по ключевым метрикам: часы, стресс, удовлетворённость, перегрузка, дисбаланс.

- Закрашенная область показывает профиль группы:
  - Светло-синяя область — сотрудники без выгорания (Burnout=0).
  - Оранжевая/красная область — сотрудники с выгоранием (Burnout=1).
- Чем больше площадь и дальше от центра — тем выше значения метрик.

Ключевые наблюдения по Radar chart:

1. Рабочие часы (work\_hours\_per\_week)
  - Burnout=1: Значение ~59.8 ч/нед (далеко от центра).
  - Burnout=0: Значение ~48.9 ч/нед (ближе к центру).
  - Разрыв: Ось сильно «вытянута» у выгоревших, что подтверждает переработки как критический фактор.
2. Уровень стресса (stress\_level)
  - Burnout=1: ~8.95/10 (максимальное удаление от центра).
  - Burnout=0: ~5.19/10 (значительно ближе).
  - Разрыв: Самый большой разрыв среди всех осей → стресс — главный дифференциатор.
3. Уровень удовлетворённости (satisfaction\_level)
  - Burnout=1: ~1.97/5 (близко к центру — низкое значение).
  - Burnout=0: ~3.07/5 (значительно дальше — выше удовлетворённость).
  - Особенность: Здесь меньшее значение = хуже, поэтому у Burnout=1 ось «вдавлена» внутрь.
4. Индекс перегрузки (overload\_index)
  - Burnout=1: ~13.36 (очень далеко от центра).
  - Burnout=0: ~6.27 (ближе к центру).
  - Разрыв: Индекс перегрузки у выгоревших более чем в 2 раза выше.
5. Дисбаланс стресс/удовлетворённость (stress\_to\_satisfaction)
  - Burnout=1: ~4.98 (стресс превышает удовлетворённость в 5 раз).
  - Burnout=0: ~1.98 (дисбаланс минимален).
  - Интерпретация: Ось показывает соотношение негативных и позитивных факторов у сотрудника.

## Группировки и агрегации

```
In [93]: # По должностям
job_stats = df.groupby('job_role').agg({
    'burnout': 'mean',
    'work_hours_per_week': 'mean',
    'stress_level': 'mean',
    'satisfaction_level': 'mean',
    'overload_index': 'mean',
    'stress_to_satisfaction': 'mean'
}).round(2).sort_values('burnout', ascending=False)
```

```
print("Средние значения по должностям:")
display(job_stats)
```

Средние значения по должностям:

	burnout	work_hours_per_week	stress_level	satisfaction_level	overload_index	stress_to_satisfaction
--	---------	---------------------	--------------	--------------------	----------------	------------------------

job\_role

Sales	0.08	49.50	5.58	3.03	6.87	2.27
HR	0.07	49.74	5.52	3.00	6.80	2.28
Manager	0.07	49.31	5.47	3.02	6.80	2.14
Engineer	0.06	49.97	5.34	2.98	6.69	2.08
Analyst	0.05	49.45	5.25	2.95	6.50	2.11

Sales — группа максимального риска (наибольший стресс и доля выгорания).

Analyst — группа минимального риска.

Различия в часах между должностями незначимы → причина риска в стрессе и специфике работы.

```
In [94]: # По полу
gender_stats = df.groupby('gender').agg({
    'burnout': 'mean',
    'work_hours_per_week': 'mean',
    'stress_level': 'mean',
    'satisfaction_level': 'mean',
    'remote_ratio': 'mean',
    'overload_index': 'mean',
    'stress_to_satisfaction': 'mean'
}).round(2)

print("Средние значения по полу:")
display(gender_stats)
```

Средние значения по полу:

	burnout	work_hours_per_week	stress_level	satisfaction_level	remote_ratio	overload_index	stress_to_satisfaction
--	---------	---------------------	--------------	--------------------	--------------	----------------	------------------------

gender

Female	0.06	49.52	5.35	2.99	50.78	6.63	2.13
Male	0.07	49.65	5.51	3.00	49.21	6.82	2.21

Гендерные различия минимальны, пол не является дифференцирующим фактором.

```
In [95]: # По стажу
exp_stats = df.groupby('experience_cat')['burnout'].mean().round(3)
print("Доля Burnout по категориям стажа:")
display(exp_stats)
```

Доля Burnout по категориям стажа:

```
experience_cat
junior    0.070
middle    0.054
senior    0.074
expert    0.056
Name: burnout, dtype: float64
```

U-образная зависимость:

- Высокий риск у Junior (адаптационный стресс, перегрузка).
- Низкий риск у Middle (оптимальный баланс).
- Высокий риск у Senior (профессиональное выгорание, стагнация).

```
In [96]: # По возрастным группам
age_stats = df.groupby('age_bin')['burnout'].mean().round(3)
print("Доля Burnout по возрастным группам:")
display(age_stats)
```

Доля Burnout по возрастным группам:

```
age_bin
Young (22-30)      0.067
Mid-Age (31-40)    0.065
Senior (41-50)     0.049
Pre-Retirement (51-60) 0.077
Name: burnout, dtype: float64
```

Предпенсионный возраст (51–60) — группа максимального риска (возрастная усталость, давление, стагнация).

```
In [97]: # По удалёнке
remote_stats = df.groupby('remote_cat').agg({
    'burnout': 'mean',
    'stress_level': 'mean',
    'satisfaction_level': 'mean',
    'work_hours_per_week': 'mean',
    'overload_index': 'mean',
    'stress_to_satisfaction': 'mean'
}).round(2)
```

```
}).round(2)

print("Средние показатели по типам удалённой работы:")
display(remote_stats)
```

Средние показатели по типам удалённой работы:

remote_cat	burnout	stress_level	satisfaction_level	work_hours_per_week	overload_index	stress_to_satisfaction
<b>onsite</b>	0.00	4.54	3.14	54.62	6.05	1.89
<b>hybrid</b>	0.06	5.43	3.02	49.27	6.70	2.15
<b>remote</b>	0.07	5.46	2.89	50.63	6.87	2.27

Onsite имеет наибольшие часы (54.62), но нулевое выгорание (в выборке).

Remote имеет высокий стресс и низкую удовлетворённость при средних часах.

Гипотеза: социальная изоляция удалёнки может компенсировать преимущества гибкости.

## Итоговые текстовые портреты

```
In [98]: # Краткий пороговый анализ для ключевых признаков
def threshold_analysis(df, feature, thresholds, target='burnout'):
    results = []
    for i in range(len(thresholds)-1):
        mask = (df[feature] >= thresholds[i]) & (df[feature] < thresholds[i+1])
        results.append({
            'Диапазон': f'{thresholds[i]}-{thresholds[i+1]}',
            'Количество': mask.sum(),
            'Доля Burnout': round(df.loc[mask, target].mean(), 3)
        })
    return pd.DataFrame(results)
```

```
In [99]: # Рабочие часы
print("Пороговый анализ work_hours_per_week:")
display(threshold_analysis(df, 'work_hours_per_week', [30,40,45,50,55,60,65,70]))
```

Пороговый анализ work\_hours\_per\_week:

	Диапазон	Количество	Доля Burnout
0	30-40	504	0.000
1	40-45	255	0.000
2	45-50	258	0.000
3	50-55	229	0.118
4	55-60	252	0.163
5	60-65	220	0.123
6	65-70	234	0.115

❖ Критический порог: 50 рабочих часов в неделю

- Сотрудники с нагрузкой >55 часов — самая уязвимая группа.

```
In [100...]: # Стress
print("Пороговый анализ stress_leve:")
display(threshold_analysis(df, 'stress_level', [1,3,5,7,9,11]))
```

Пороговый анализ stress\_leve:

	Диапазон	Количество	Доля Burnout
0	1-3	419	0.000
1	3-5	399	0.000
2	5-7	393	0.000
3	7-9	399	0.110
4	9-11	390	0.218

❖ Критический порог: StressLevel  $\geq 7$ , зона острого риска  $\geq 9$ .

- Стress работает как ступенчатый триггер: после 7 риск резко возрастает.

```
In [101...]: # Удовлетворённость
print("Пороговый анализ stress_levesatisfaction_level:")
display(threshold_analysis(df, 'satisfaction_level', [1,2,3,4,5]))
```

Пороговый анализ stress\_levesatisfaction\_level:

Диапазон	Количество	Доля Burnout
0	1-2	497
1	2-3	489
2	3-4	513
3	4-5	497
		0.137
		0.125
		0.000
		0.000

❖ Критический порог: Satisfaction < 2.5

- Эти значения формируют идеальную HR-интерпретацию:
- «Сотрудники с удовлетворённостью ниже 2.5 — группа немедленного внимания.»

In [102...]: # Индекс перегрузки (примерные пороги: 0, 5, 10, 12, 15, 18)

```
print("Пороговый анализ overload_index:")
display(threshold_analysis(df, 'overload_index', [0,5,10,12,15,18]))
```

Пороговый анализ overload\_index:

Диапазон	Количество	Доля Burnout
0	0-5	805
1	5-10	746
2	10-12	186
3	12-15	198
4	15-18	65
		0.000
		0.000
		0.151
		0.384
		0.385

In [103...]: # Stress/Satisfaction (примерные пороги: 0, 1, 2.5, 4, 6, 10)

```
print("\nПороговый анализ stress_to_satisfaction:")
display(threshold_analysis(df, 'stress_to_satisfaction', [0,1,2.5,4,6,10]))
```

Пороговый анализ stress\_to\_satisfaction:

Диапазон	Количество	Доля Burnout
0	0-1	517
1	1-2.5	850
2	2.5-4	378
3	4-6	171
4	6-10	83
		0.000
		0.000
		0.130
		0.275
		0.398

❖ Ключевой инсайт:

- Именно совокупность нагрузки и стресса/удовлетворённости формирует реальный риск.
- Эти метрики практически создают "идеальный feature engineering" для модели.

In [104...]:

```
print("\n==== Портрет Burnout=0 ===")
print("• Рабочие часы ≤50 (Burnout=0%)")
print("• Стресс ≤7 (Burnout=0%)")
print("• Удовлетворённость ≥3 (Burnout=0%)")
print("• Индекс перегрузки ≤10 (Burnout=0%)")
print("• Stress/Satisfaction ≤2.5 (Burnout=0%)")
print("• Чаше Analyst/Engineer")
print("• Возраст: равномерно распределён")

print("\n==== Портрет Burnout=1 ===")
print("• Рабочие часы >50, особенно 55–60 (Burnout=16.3%)")
print("• Стресс >7, особенно ≥9 (Burnout=21.8%)")
print("• Удовлетворённость <2.5 (Burnout=12–14%)")
print("• Индекс перегрузки >12 (Burnout=38%)")
print("• Stress/Satisfaction >2.5, особенно >4 (Burnout=27–40%)")
print("• Чаше Sales, Manager, HR")
print("• Возрастные группы риска: 22–30 и 51–60 лет")
```

==== Портрет Burnout=0 ===

- Рабочие часы ≤50 (Burnout=0%)
- Стресс ≤7 (Burnout=0%)
- Удовлетворённость ≥3 (Burnout=0%)
- Индекс перегрузки ≤10 (Burnout=0%)
- Stress/Satisfaction ≤2.5 (Burnout=0%)
- Чаше Analyst/Engineer
- Возраст: равномерно распределён

==== Портрет Burnout=1 ===

- Рабочие часы >50, особенно 55–60 (Burnout=16.3%)
- Стресс >7, особенно ≥9 (Burnout=21.8%)
- Удовлетворённость <2.5 (Burnout=12–14%)
- Индекс перегрузки >12 (Burnout=38%)
- Stress/Satisfaction >2.5, особенно >4 (Burnout=27–40%)
- Чаше Sales, Manager, HR
- Возрастные группы риска: 22–30 и 51–60 лет

«Burnout=1 больше связан с сочетанием факторов, чем с одним отдельным фактором.

- Риск возникает при пересечении высокой нагрузки, низкой удовлетворённости и повышенного стресса.»

## Основные выводы

- Критические факторы риска:
  - Длительная рабочая неделя
  - Высокий уровень стресса
  - Низкая удовлетворенность
  - Высокий индекс перегрузки
- Группы повышенного внимания:
  - Сотрудники с нагрузкой >50 часов
  - Сотрудники со стрессом >7 баллов
  - Сотрудники с индексом перегрузки >12
  - Сотрудники должностей Sales, HR, Manager
  - Возрастные группы 22-30 и 51-60 лет
- Защитные факторы:
  - Рабочая неделя <50 часов
  - Стресс <7 баллов
  - Удовлетворенность >2.5
  - Индекс перегрузки <10
  - Работа в офисе (onsite)

## Рекомендации для HR на основе профилей

- Для Sales:
  - Внедрить ротацию, менторинг, stress-management.
  - Пересмотреть KPI и систему мотивации.
- Для Junior:
  - Адаптационные программы, контроль нагрузки, регулярные чекапы.
- Для Senior / Pre-Retirement:
  - Гибкий график, программы развития, поддержка карьерного роста.
- Для всех групп риска:
  - Мониторинг часов (>50), стресса (>7), удовлетворенности (<3).
  - Автоматические алерты при превышении порогов.
  - Вмешательства: коучинг, психологическая поддержка, снижение нагрузки.
- Мониторинг нагрузки:
  - Контроль рабочих часов
  - Отслеживание индекса перегрузки
- Профилактические меры:
  - Регулярный мониторинг стресса
  - Оценка удовлетворенности
  - Поддержка групп риска
- Оптимизация рабочих процессов
  - Гибкий график
  - Программы поддержки

## Выводы по шагу 4

### Сравнительные профили Burnout=0 vs Burnout=1

- У сотрудников с выгоранием рабочие часы выше на ~11 часов (59.8 против 48.9).
- Стресс выше на ~3.8 балла (8.95 против 5.19).
- Удовлетворенность ниже на ~1.1 балла (1.97 против 3.07).
- Индекс перегрузки почти в 2 раза выше (13.36 против 6.27).
- Соотношение стресс/удовлетворенность в 2.5 раза выше (4.98 против 1.98).
- Возраст и стаж почти не отличаются.
  - Вывод: выгорание возникает не из-за одной метрики, а из-за сочетания высокой нагрузки, стресса и низкой удовлетворенности.

### Ключевые факторы различий

- Burnout=1 всегда связан с переработкой (больше 50 часов), высоким стрессом (больше либо равно 8) и низкой удовлетворенностью (меньше 3).
  - Рабочие часы > 50/нед → риск возрастает, пик при 55–60 ч.
  - Стресс ≥ 7 → начинается рост риска, ≥9 — зона острого риска.
  - Удовлетворенность < 2.5 → резко увеличивается доля Burnout.
  - Overload Index > 12 → риск ~38%.
  - Stress/Satisfaction > 4 → риск 27–40%.
- У сотрудников без Burnout все показатели находятся в «безопасных» диапазонах.

## Визуализации (heatmap, bar-chart, radar chart)

- Heatmap подтверждает разрыв по нагрузке, стрессу и удовлетворённости.
- Bar-chart показывает скачок по всем метрикам у Burnout=1.
- Radar chart наглядно демонстрирует «раздувшуюся» область перегрузки у сотрудников с Burnout: перекос в сторону стресса, часов и перегрузки при “провале” удовлетворённости

## Группы повышенного риска

- Должности: Sales (7.67%), HR, Manager.
- Возраст: 22–30 лет (молодые) и 51–60 лет (предпенсионные).
- Стаж: Junior и Senior (U-образная зависимость).
- Формат работы: Remote и Hybrid (риск выше, чем у Onsite).

## Практический вывод

- Переход от Burnout=0 к Burnout=1 — это скачок сразу по нескольким метрикам нагрузки, а не по одной.
- Ключевые индикаторы риска: рабочие часы, стресс, удовлетворённость, индекс перегрузки, stress\_to\_satisfaction.
- Возраст, стаж и удалёнка не оказывают значимого влияния.

## Практические рекомендации для HR

- Для Sales: пересмотр KPI, ротация, stress-менеджмент.
- Для Junior: адаптация, контроль нагрузки.
- Для Senior: гибкий график, карьерное развитие.
- Для всех: мониторинг пороговых значений, автоматические алерты, программы поддержки.

## Главный инсайт

- Выгорание — не результат одного фактора, а комбинация переработок, высокого стресса и низкой удовлетворённости.
- Профилирование позволяет выявлять сотрудников в зоне риска до наступления кризиса.

[\\* к содержанию](#)

## Шаг 5: Проверка статистических гипотез

### Гипотеза 1: должности/выгорание

#### Гипотеза 1: существуют значимые различия в уровне выгорания между должностями

- H0: средний уровень Burnout одинаков для всех должностей.
- H1: существуют должности, где средний уровень Burnout статистически значимо отличается.

In [105...]

```
# Гипотеза 1: различия между должностями
groups = [df[df['job_role']==role]['burnout'] for role in df['job_role'].unique()]
anova_result = stats.f_oneway(*groups)

print("Гипотеза 1: различия между должностями")
print("H0: средний уровень Burnout одинаков для всех должностей")
print("H1: хотя бы одна должность имеет статистически значимо другой уровень Burnout")
print(f"F-статистика = {anova_result.statistic:.3f}, p-value = {anova_result.pvalue:.4f}")

if anova_result.pvalue < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

# Средние значения Burnout по должностям + количество сотрудников
mean_burnout_by_role = (
    df.groupby('job_role')
    .agg(mean_burnout=('burnout', 'mean'),
         count=('burnout', 'size'))
    .round(3)
    .sort_values('mean_burnout', ascending=False)
    .reset_index()
)

print("\nСредние значения Burnout по должностям:")
display(mean_burnout_by_role)

# Визуализация (горизонтальный barplot для удобства)
plt.figure(figsize=(10,6))
sns.barplot(
    data=mean_burnout_by_role,
    y='job_role', x='mean_burnout',
    hue='job_role', palette='Set2', legend=False
)
```

```

plt.title("Средний уровень Burnout по должностям")
plt.xlabel("Доля Burnout")
plt.ylabel("Должность")

# Добавим подписи значений и количества сотрудников
for i, (val, cnt) in enumerate(zip(mean_burnout_by_role['mean_burnout'], mean_burnout_by_role['count'])):
    plt.text(val+0.01, i, f"{val:.2f} ({cnt})", va='center')

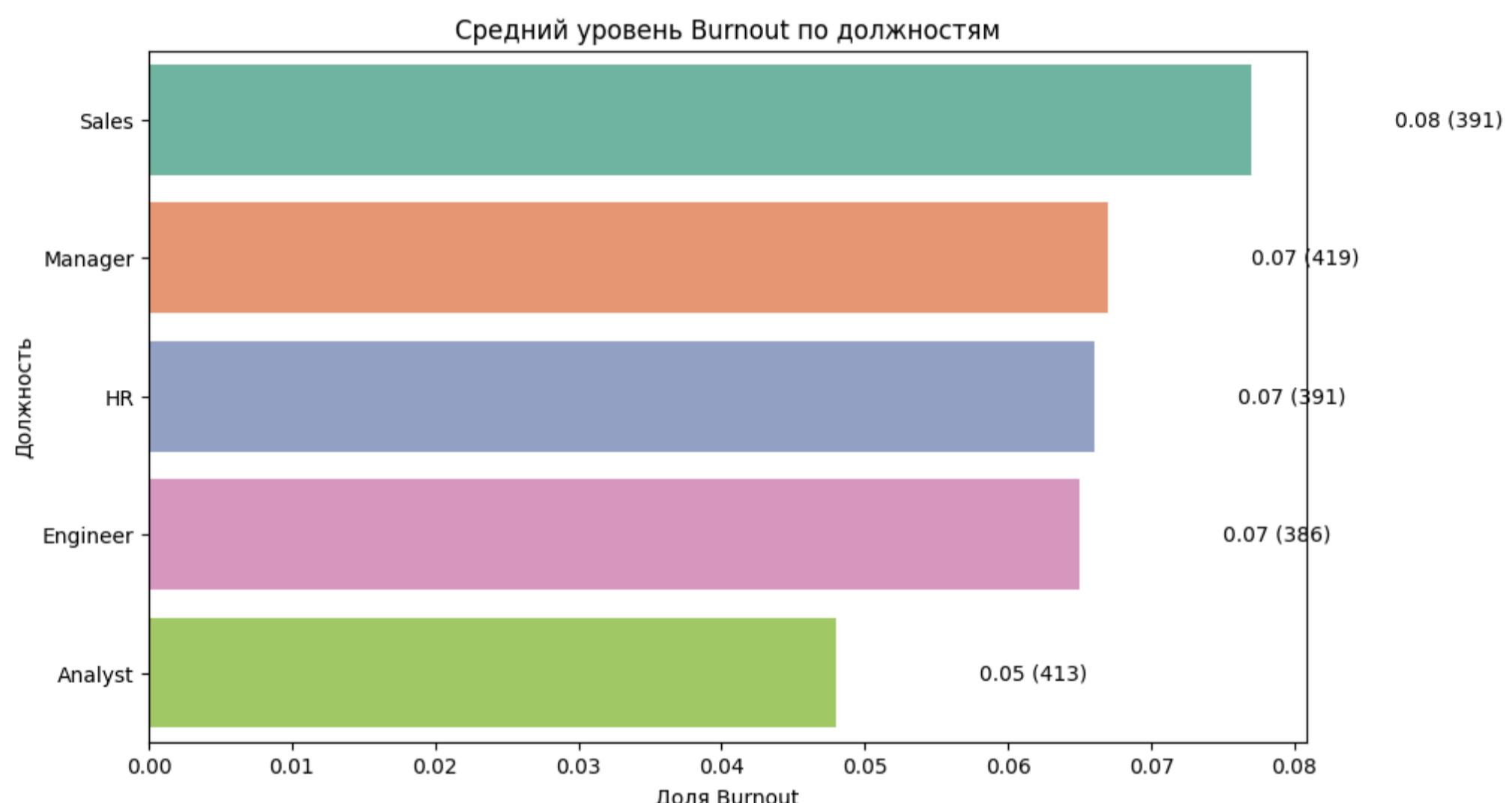
plt.show()

```

Гипотеза 1: различия между должностями  
 $H_0$ : средний уровень Burnout одинаков для всех должностей  
 $H_1$ : хотя бы одна должность имеет статистически значимо другой уровень Burnout  
F-статистика = 0.699, p-value = 0.5923  
→ не отвергаем  $H_0$  (гипотеза не подтверждается)

Средние значения Burnout по должностям:

job_role	mean_burnout	count
0 Sales	0.077	391
1 Manager	0.067	419
2 HR	0.066	391
3 Engineer	0.065	386
4 Analyst	0.048	413



## Гипотеза 2: удалённая работа/выгорание

### Гипотеза 2: удалённая работа снижает риск выгорания

- $H_0$ : средний уровень Burnout одинаков для всех категорий удалённой работы (onsite, hybrid, remote).
- $H_1$ : хотя бы одна категория удалённой работы имеет статистически значимо другой уровень Burnout.

```

In [106...]: # Гипотеза 2: удалённая работа и Burnout
contingency_remote = pd.crosstab(df['remote_cat'], df['burnout'])
chi2_remote = stats.chi2_contingency(contingency_remote)

print("\nГипотеза 2: удалённая работа и Burnout")
print("H0: уровень Burnout одинаков для всех категорий удалённой работы (onsite, hybrid, remote)")
print("H1: хотя бы одна категория удалённой работы имеет статистически значимо другой уровень Burnout")
print(f"χ²-статистика = {chi2_remote[0]:.3f}, p-value = {chi2_remote[1]:.4f}")

if chi2_remote[1] < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

# Средние значения Burnout + количество сотрудников по категориям удалённой работы
mean_burnout_by_remote = (
    df.groupby('remote_cat')
    .agg(mean_burnout=('burnout', 'mean'),
         count=('burnout', 'size'))
    .round(3)
    .reset_index()
)

```

```

print("\nСредние значения Burnout по категориям удалённой работы:")
print(mean_burnout_by_remote)

# Визуализация
plt.figure(figsize=(8,5))
sns.barplot(data=mean_burnout_by_remote,
            x='remote_cat', y='mean_burnout',
            hue='remote_cat', palette='Set2', legend=False)
plt.title("Средний уровень Burnout по категориям удалённой работы")
plt.xlabel("Категория удалённой работы (0=onsite, 1=hybrid, 2=remote)")
plt.ylabel("Доля Burnout")

# Добавим подписи значений и количества сотрудников
for i, (val, cnt) in enumerate(zip(mean_burnout_by_remote['mean_burnout'],
                                     mean_burnout_by_remote['count'])):
    plt.text(i, val/2, f"{val:.3f} ({cnt})",
             ha='center', va='center', color='black', fontsize=12)

plt.show()

```

Гипотеза 2: удалённая работа и Burnout

H0: уровень Burnout одинаков для всех категорий удалённой работы (onsite, hybrid, remote)

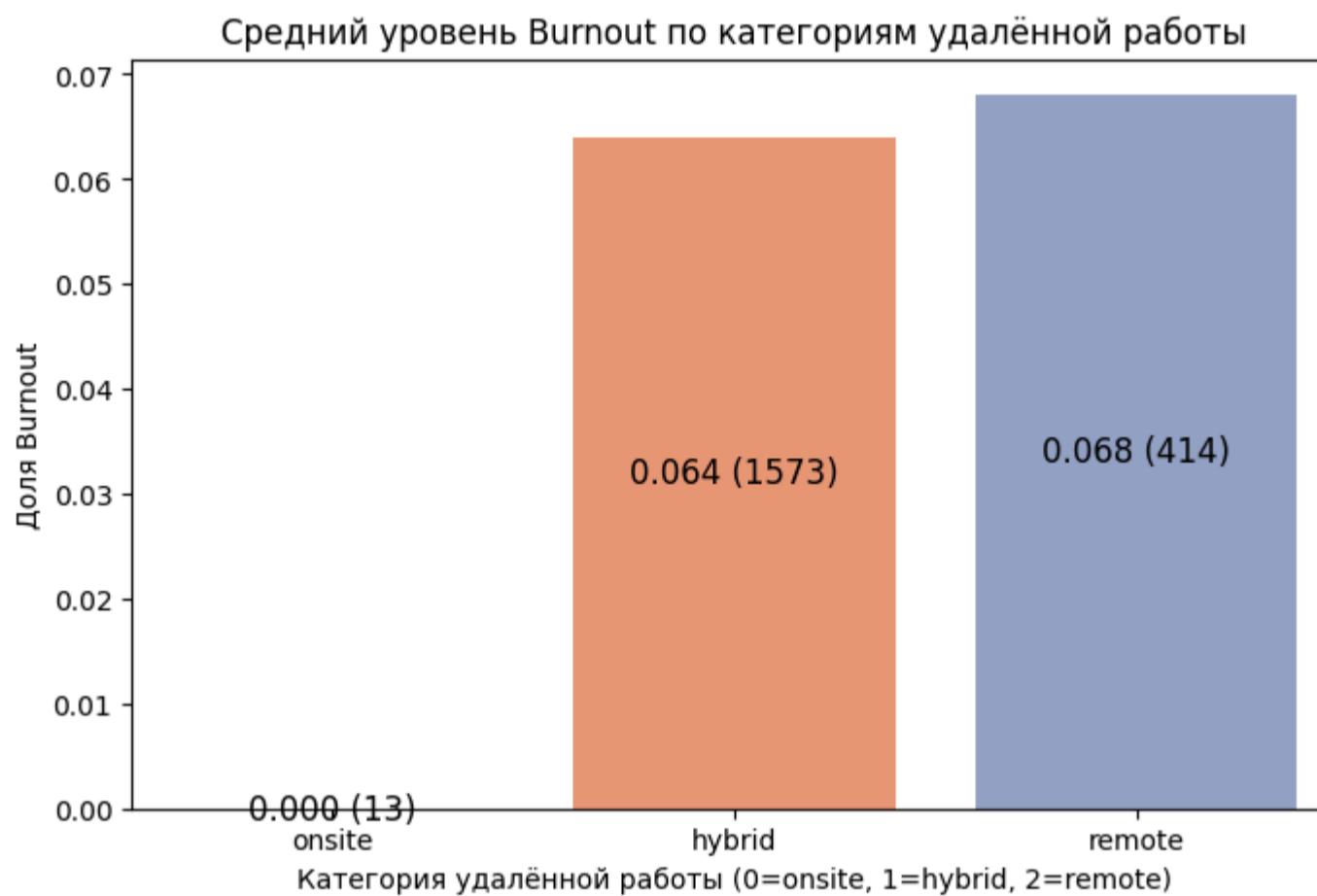
H1: хотя бы одна категория удалённой работы имеет статистически значимо другой уровень Burnout

$\chi^2$ -статистика = 0.966, p-value = 0.6170

→ не отвергаем H0 (гипотеза не подтверждается)

Средние значения Burnout по категориям удалённой работы:

remote_cat	mean_burnout	count
0 onsite	0.000	13
1 hybrid	0.064	1573
2 remote	0.068	414



### Гипотеза 3: рабочие часы/выгорание

Гипотеза 3: сотрудники с рабочими часами более 45 в неделю имеют более высокий риск

- H0: средний уровень Burnout одинаков у сотрудников ≤45 часов и >45 часов.
- H1: сотрудники, работающие >45 часов, имеют статистически значимо более высокий уровень Burnout.

In [107...]

```

# Гипотеза 3: рабочие часы >45
contingency_hours = pd.crosstab(df['hours_above_45'], df['burnout'])
chi2_hours = stats.chi2_contingency(contingency_hours)

print("\nГипотеза 3: рабочие часы >45")
print("H0: уровень Burnout одинаков у сотрудников ≤45 и >45 часов")
print("H1: сотрудники >45 часов имеют статистически значимо более высокий уровень Burnout")
print(f"\u03c7\u00b2-статистика = {chi2_hours[0]:.3f}, p-value = {chi2_hours[1]}")

if chi2_hours[1] < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

mean_burnout_by_hours = (
    df.groupby('hours_above_45')
    .agg(mean_burnout=('burnout', 'mean'),
         count=('burnout', 'size'))
    .round(3)
    .reset_index()
)
print("\nСредние значения Burnout по группам рабочих часов:")
print(mean_burnout_by_hours)

plt.figure(figsize=(6,4))

```

```

sns.barplot(data=mean_burnout_by_hours, x='hours_above_45', y='mean_burnout',
            hue='hours_above_45', palette='Set2', legend=False)
plt.title("Burnout по группам рабочих часов (<=45 vs >45)")
plt.xlabel("Группа рабочих часов")
plt.ylabel("Доля Burnout")
for i, (val, cnt) in enumerate(zip(mean_burnout_by_hours['mean_burnout'],
                                    mean_burnout_by_hours['count'])):
    plt.text(i, val/2, f"{val:.2f} ({cnt})",
             ha='center', va='center', color='black', fontsize=12)

plt.show()

```

Гипотеза 3: рабочие часы >45

H0: уровень Burnout одинаков у сотрудников ≤45 и >45 часов

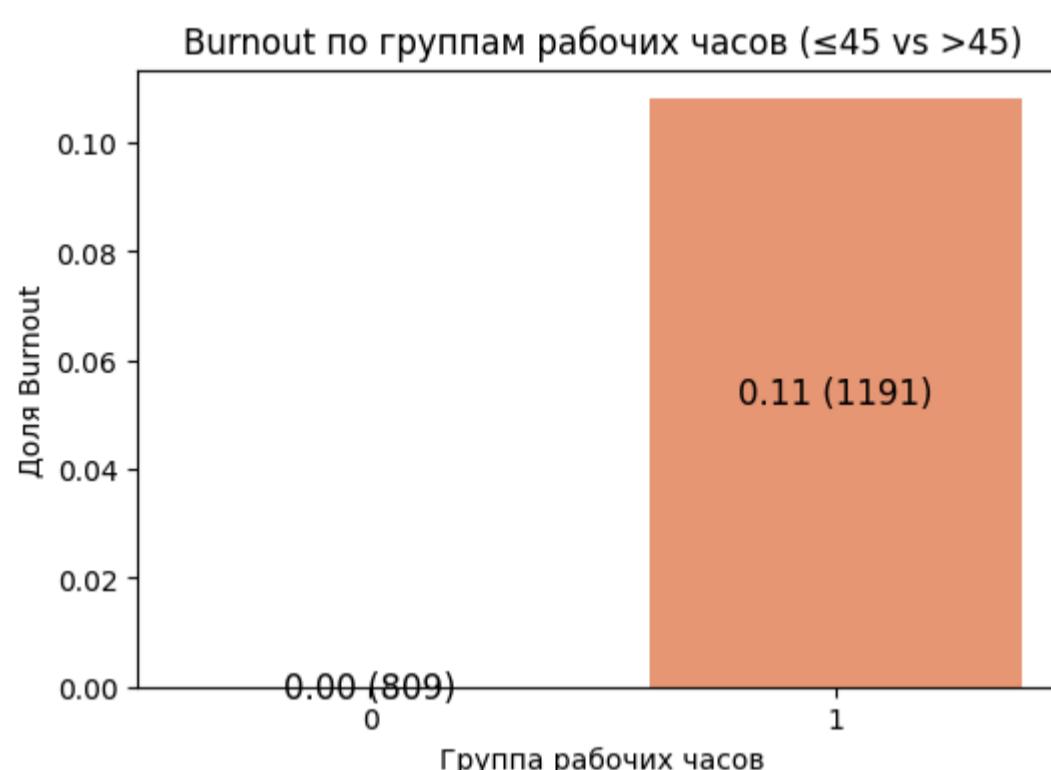
H1: сотрудники >45 часов имеют статистически значимо более высокий уровень Burnout

$\chi^2$ -статистика = 91.880, p-value = 9.210855637424495e-22

→ отвергаем H0, принимаем H1 (гипотеза подтверждается)

Средние значения Burnout по группам рабочих часов:

	hours_above_45	mean_burnout	count
0	0	0.000	809
1	1	0.108	1191



## Гипотеза 4: StressLevel/SatisfactionLevel

### Гипотеза 4: существует значимая корреляция между StressLevel и SatisfactionLevel

- H0: нет статистически значимой корреляции между StressLevel и SatisfactionLevel.
- H1: существует статистически значимая корреляция (ожидается отрицательная).

In [108...]

```

# Гипотеза 4: корреляция StressLevel и SatisfactionLevel
pearson_corr = stats.pearsonr(df['stress_level'], df['satisfaction_level'])
spearman_corr = stats.spearmanr(df['stress_level'], df['satisfaction_level'])
print("\nГипотеза 4: корреляция StressLevel и SatisfactionLevel")
print("H0: нет статистически значимой корреляции")
print("H1: существует статистически значимая корреляция (ожидается отрицательная)")

print("\n корреляция Пирсона Stress vs Satisfaction: p =", pearson_corr[1])

if pearson_corr[1] < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

print(" корреляция Спирмена Stress vs Satisfaction: p =", spearman_corr.pvalue)
if spearman_corr.pvalue < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

print(f"Пирсон: r = {pearson_corr[0]:.3f}")
print(f"Спирмен: r = {spearman_corr.correlation:.3f}")

```

Гипотеза 4: корреляция StressLevel и SatisfactionLevel

H0: нет статистически значимой корреляции

H1: существует статистически значимая корреляция (ожидается отрицательная)

корреляция Пирсона Stress vs Satisfaction: p = 0.11344281007094927

→ не отвергаем H0 (гипотеза не подтверждается)

корреляция Спирмена Stress vs Satisfaction: p = 0.11584191898734937

→ не отвергаем H0 (гипотеза не подтверждается)

Пирсон: r = 0.035

Спирмен: r = 0.035

In [109...]

```

# Корреляция только для выгоревших
corr_burnout = df[df['burnout']==1][['stress_level', 'satisfaction_level']].corr()
corr_burnout

```

Out[109...]

	stress_level	satisfaction_level
--	--------------	--------------------

stress_level	1.000000	0.027988
satisfaction_level	0.027988	1.000000

In [110...]

```
# Корреляция только для выгоревших
corr_burnout = df[df['burnout']!=1][['stress_level', 'satisfaction_level']].corr()
corr_burnout
```

Out[110...]

	stress_level	satisfaction_level
--	--------------	--------------------

stress_level	1.000000	0.120585
satisfaction_level	0.120585	1.000000

## Гипотеза 5: стресс/выгорание

### Гипотеза 5: стресс выше у сотрудников с Burnout=1

- H0: средний StressLevel одинаков у Burnout=0 и Burnout=1.
- H1: средний StressLevel у Burnout=1 статистически значимо выше.

In [111...]

```
# Гипотеза 5: стресс выше у Burnout=1
stress_0 = df[df['burnout']==0]['stress_level']
stress_1 = df[df['burnout']==1]['stress_level']
ttest_stress = stats.ttest_ind(stress_0, stress_1, equal_var=False)

print("\nГипотеза 5: стресс выше у Burnout=1")
print("H0: средний StressLevel одинаков у Burnout=0 и Burnout=1")
print("H1: StressLevel у Burnout=1 выше")
print(f"t-статистика = {ttest_stress.statistic:.3f}, p-value = {ttest_stress.pvalue}")

if ttest_stress.pvalue < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

mean_stress_by_group = (
    df.groupby('burnout')['stress_level']
        .agg(mean_stress='mean', count='size')
        .round(3)
        .reset_index()
)
print("\nСредний StressLevel по группам Burnout:")
mean_stress_by_group
```

Гипотеза 5: стресс выше у Burnout=1  
 H0: средний StressLevel одинаков у Burnout=0 и Burnout=1  
 H1: StressLevel у Burnout=1 выше  
 t-статистика = -39.290, p-value = 1.229586558465087e-142  
 → отвергаем H0, принимаем H1 (гипотеза подтверждается)

Средний StressLevel по группам Burnout:

Out[111...]

	burnout	mean_stress	count
--	---------	-------------	-------

0	0	5.189	1871
1	1	8.953	129

## Гипотеза 6: удовлетворённость/выгорание

### Гипотеза 6: удовлетворённость ниже у сотрудников с Burnout=1

- H0: средний SatisfactionLevel одинаков у Burnout=0 и Burnout=1.
- H1: средний SatisfactionLevel у Burnout=1 статистически значимо ниже.

In [112...]

```
# Гипотеза 6: удовлетворённость ниже у Burnout=1
sat_0 = df[df['burnout']==0]['satisfaction_level']
sat_1 = df[df['burnout']==1]['satisfaction_level']
ttest_sat = stats.ttest_ind(sat_0, sat_1, equal_var=False)

print("\nГипотеза 6: удовлетворённость ниже у Burnout=1")
print("H0: средний SatisfactionLevel одинаков у Burnout=0 и Burnout=1")
print("H1: SatisfactionLevel у Burnout=1 ниже")
print(f"t-статистика = {ttest_sat.statistic:.3f}, p-value = {ttest_sat.pvalue}")

if ttest_sat.pvalue < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

mean_sat_by_group = (
    df.groupby('burnout')['satisfaction_level']
        .agg(mean_satisfaction='mean', count='size')
        .round(3)
        .reset_index()
)
```

```
)  
print("\nСредний SatisfactionLevel по группам Burnout:")  
mean_sat_by_group
```

Гипотеза 6: удовлетворённость ниже у Burnout=1  
H0: средний SatisfactionLevel одинаков у Burnout=0 и Burnout=1  
H1: SatisfactionLevel у Burnout=1 ниже  
t-статистика = 19.161, p-value = 7.915429674041607e-48  
→ отвергаем H0, принимаем H1 (гипотеза подтверждается)

Средний SatisfactionLevel по группам Burnout:

	burnout	mean_satisfaction	count
0	0	3.066	1871
1	1	1.972	129

## Гипотеза 7: пол/выгорание

### Гипотеза 7: мужчины и женщины имеют разный уровень Burnout

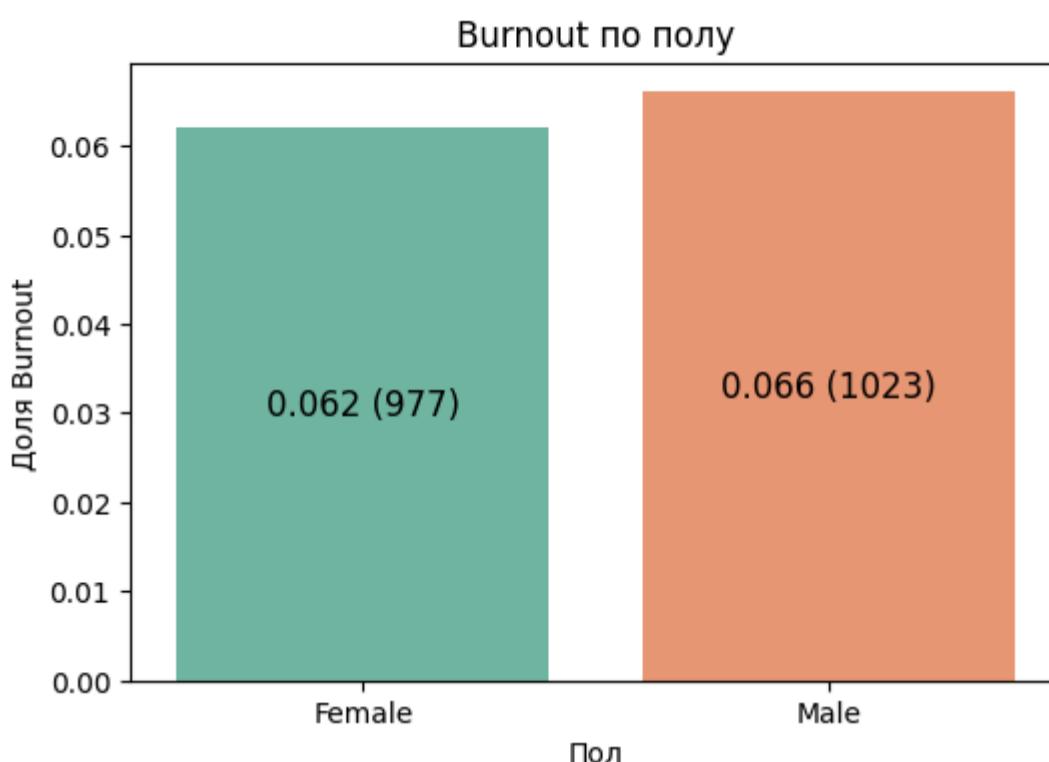
- H0: средний уровень Burnout одинаков у мужчин и женщин.
- H1: уровень Burnout статистически значимо различается между мужчинами и женщинами.

```
In [113...]: # Гипотеза 7: различия по полу  
contingency_gender = pd.crosstab(df['gender'], df['burnout'])  
chi2_gender = stats.chi2_contingency(contingency_gender)  
  
print("\nГипотеза 7: различия по полу")  
print("H0: уровень Burnout одинаков у мужчин и женщин")  
print("H1: уровень Burnout различается между мужчинами и женщинами")  
print(f"χ²-статистика = {chi2_gender[0]:.3f}, p-value = {chi2_gender[1]:.4f}")  
  
if chi2_gender[1] < 0.05:  
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")  
else:  
    print("→ не отвергаем H0 (гипотеза не подтверждается)")  
  
mean_burnout_by_gender = (  
    df.groupby('gender')['burnout']  
    .agg(mean_burnout='mean', count='size')  
    .round(3)  
    .reset_index()  
)  
print("\nСредние значения Burnout по полу:")  
display(mean_burnout_by_gender)  
  
plt.figure(figsize=(6,4))  
sns.barplot(data=mean_burnout_by_gender, x='gender', y='mean_burnout',  
            hue='gender', palette='Set2', legend=False)  
plt.title("Burnout по полу")  
plt.xlabel("Пол")  
plt.ylabel("Доля Burnout")  
for i, (val, cnt) in enumerate(zip(mean_burnout_by_gender['mean_burnout'],  
                                    mean_burnout_by_gender['count'])):  
    plt.text(i, val/2, f"{val} ({cnt})",  
             ha='center', va='center', color='black', fontsize=12)  
  
plt.show()
```

Гипотеза 7: различия по полу  
H0: уровень Burnout одинаков у мужчин и женщин  
H1: уровень Burnout различается между мужчинами и женщинами  
χ²-статистика = 0.076, p-value = 0.7824  
→ не отвергаем H0 (гипотеза не подтверждается)

Средние значения Burnout по полу:

	gender	mean_burnout	count
0	Female	0.062	977
1	Male	0.066	1023



## Гипотеза 8: рабочие часы/выгорание

**Гипотеза 8: work\_hours\_per\_week - у сотрудников с выгоранием рабочие часы значительно выше**

- H0: средние рабочие часы одинаковы у Burnout=0 и Burnout=1
- H1: у Burnout=1 рабочие часы выше

```
In [114...]: # Гипотеза 8: WorkHoursPerWeek - у сотрудников с выгоранием рабочие часы значительно выше
hours_0 = df[df['burnout']==0]['work_hours_per_week']
hours_1 = df[df['burnout']==1]['work_hours_per_week']
ttest_hours = stats.ttest_ind(hours_0, hours_1, equal_var=False)

print("\nГипотеза 8: рабочие часы у Burnout=1 выше")
print("H0: средние рабочие часы одинаковы у Burnout=0 и Burnout=1")
print("H1: у Burnout=1 рабочие часы выше")
print(f"t-статистика = {ttest_hours.statistic:.3f}, p-value = {ttest_hours.pvalue}")

if ttest_hours.pvalue < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

mean_hours_by_group = (
    df.groupby('burnout')['work_hours_per_week']
        .agg(mean_hours='mean', count='size')
        .round(3)
        .reset_index()
)
print("\nСредние рабочие часы по группам Burnout:")
print(mean_hours_by_group)
```

Гипотеза 8: рабочие часы у Burnout=1 выше  
H0: средние рабочие часы одинаковы у Burnout=0 и Burnout=1  
H1: у Burnout=1 рабочие часы выше  
t-статистика = -18.864, p-value = 3.26429149096689e-47  
→ отвергаем H0, принимаем H1 (гипотеза подтверждается)

Средние рабочие часы по группам Burnout:

burnout	mean_hours	count
0	48.886	1871
1	59.767	129

## Гипотеза 9: перегрузка часы/выгорание

**Гипотеза 9: overload\_index - перегрузка выражена значительно сильнее у Burnout=1**

- H0: средний overload\_index одинаков у Burnout=0 и Burnout=1
- H1: overload\_index у Burnout=1 выше

```
In [115...]: # Гипотеза 9: overload_index - перегрузка выражена значительно сильнее у Burnout=1
overload_0 = df[df['burnout']==0]['overload_index']
overload_1 = df[df['burnout']==1]['overload_index']
ttest_overload = stats.ttest_ind(overload_0, overload_1, equal_var=False)

print("\nГипотеза 9: перегрузка выше у Burnout=1")
print("H0: средний overload_index одинаков у Burnout=0 и Burnout=1")
print("H1: overload_index у Burnout=1 выше")
print(f"t-статистика = {ttest_overload.statistic:.3f}, p-value = {ttest_overload.pvalue}")

if ttest_overload.pvalue < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

mean_overload_by_group = (
    df.groupby('burnout')['overload_index']
```

```

    .agg(mean_overload='mean', count='size')
    .round(3)
    .reset_index()
)
print("\nСредний overload_index по группам Burnout:")
print(mean_overload_by_group)

```

Гипотеза 9: перегрузка выше у Burnout=1  
H0: средний overload\_index одинаков у Burnout=0 и Burnout=1  
H1: overload\_index у Burnout=1 выше  
t-статистика = -42.246, p-value = 1.4630132202706375e-111  
→ отвергаем H0, принимаем H1 (гипотеза подтверждается)

Средний overload\_index по группам Burnout:

burnout	mean_overload	count
0	6.273	1871
1	13.361	129

## Гипотеза 10: стресс\_удовлетворенность/выгорание

**Гипотеза 10: stress\_to\_satisfaction — дисбаланс стресс/удовлетворенность резко растет у Burnout=1**

- H0: средний stress\_to\_satisfaction одинаков у Burnout=0 и Burnout=1
- H1: stress\_to\_satisfaction у Burnout=1 выше

```
In [116...]
# Гипотеза 10: stress_to_satisfaction — дисбаланс стресс/удовлетворенность резко растет у Burnout=1
ratio_0 = df[df['burnout']==0]['stress_to_satisfaction']
ratio_1 = df[df['burnout']==1]['stress_to_satisfaction']
ttest_ratio = stats.ttest_ind(ratio_0, ratio_1, equal_var=False)

print("\nГипотеза 10: дисбаланс стресс/удовлетворенность выше у Burnout=1")
print("H0: средний stress_to_satisfaction одинаков у Burnout=0 и Burnout=1")
print("H1: stress_to_satisfaction у Burnout=1 выше")
print(f"t-статистика = {ttest_ratio.statistic:.3f}, p-value = {ttest_ratio.pvalue:.4f}")

if ttest_ratio.pvalue < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

mean_ratio_by_group = (
    df.groupby('burnout')['stress_to_satisfaction']
    .agg(mean_ratio='mean', count='size')
    .round(3)
    .reset_index()
)
print("\nСредний stress_to_satisfaction по группам Burnout:")
print(mean_ratio_by_group)
```

Гипотеза 10: дисбаланс стресс/удовлетворенность выше у Burnout=1  
H0: средний stress\_to\_satisfaction одинаков у Burnout=0 и Burnout=1  
H1: stress\_to\_satisfaction у Burnout=1 выше  
t-статистика = -20.015, p-value = 0.0000  
→ отвергаем H0, принимаем H1 (гипотеза подтверждается)

Средний stress\_to\_satisfaction по группам Burnout:

burnout	mean_ratio	count
0	1.979	1871
1	4.979	129

## Гипотеза 11: возрастные группы/выгорание

**Гипотеза 11: возрастные группы — различия в Burnout между возрастными группами**

- H0: средний уровень Burnout одинаков во всех возрастных группах
- H1: хотя бы одна возрастная группа имеет статистически значимо другой уровень Burnout

```
In [117...]
# Гипотеза 11: возрастные группы — различия в Burnout между возрастными группами
groups_age = [df[df['age_bin']==grp]['burnout'] for grp in df['age_bin'].unique()]
anova_age = stats.f_oneway(*groups_age)

print("\nГипотеза 11: различия по возрастным группам")
print("H0: средний уровень Burnout одинаков во всех возрастных группах")
print("H1: хотя бы одна возрастная группа имеет статистически значимо другой уровень Burnout")
print(f"F-статистика = {anova_age.statistic:.3f}, p-value = {anova_age.pvalue}")

if anova_age.pvalue < 0.05:
    print("→ отвергаем H0, принимаем H1 (гипотеза подтверждается)")
else:
    print("→ не отвергаем H0 (гипотеза не подтверждается)")

mean_burnout_by_age = (
    df.groupby('age_bin')['burnout']
    .agg(mean_burnout='mean', count='size')
    .round(3)
    .reset_index()
)
print("\nСредние значения Burnout по возрастным группам:")
print(mean_burnout_by_age)
```

```

plt.figure(figsize=(8,5))
sns.barplot(data=mean_burnout_by_age, x='age_bin', y='mean_burnout',
            hue='age_bin', palette='Set2', legend=False)
plt.title("Burnout по возрастным группам")
plt.xlabel("Возрастная группа")
plt.ylabel("Доля Burnout")
for i, (val, cnt) in enumerate(zip(mean_burnout_by_age['mean_burnout'], mean_burnout_by_age['count'])):
    plt.text(i, val+0.01, f"{val:.2f} ({cnt})", ha='center')
plt.show()

```

Гипотеза 11: различия по возрастным группам

Н0: средний уровень Burnout одинаков во всех возрастных группах

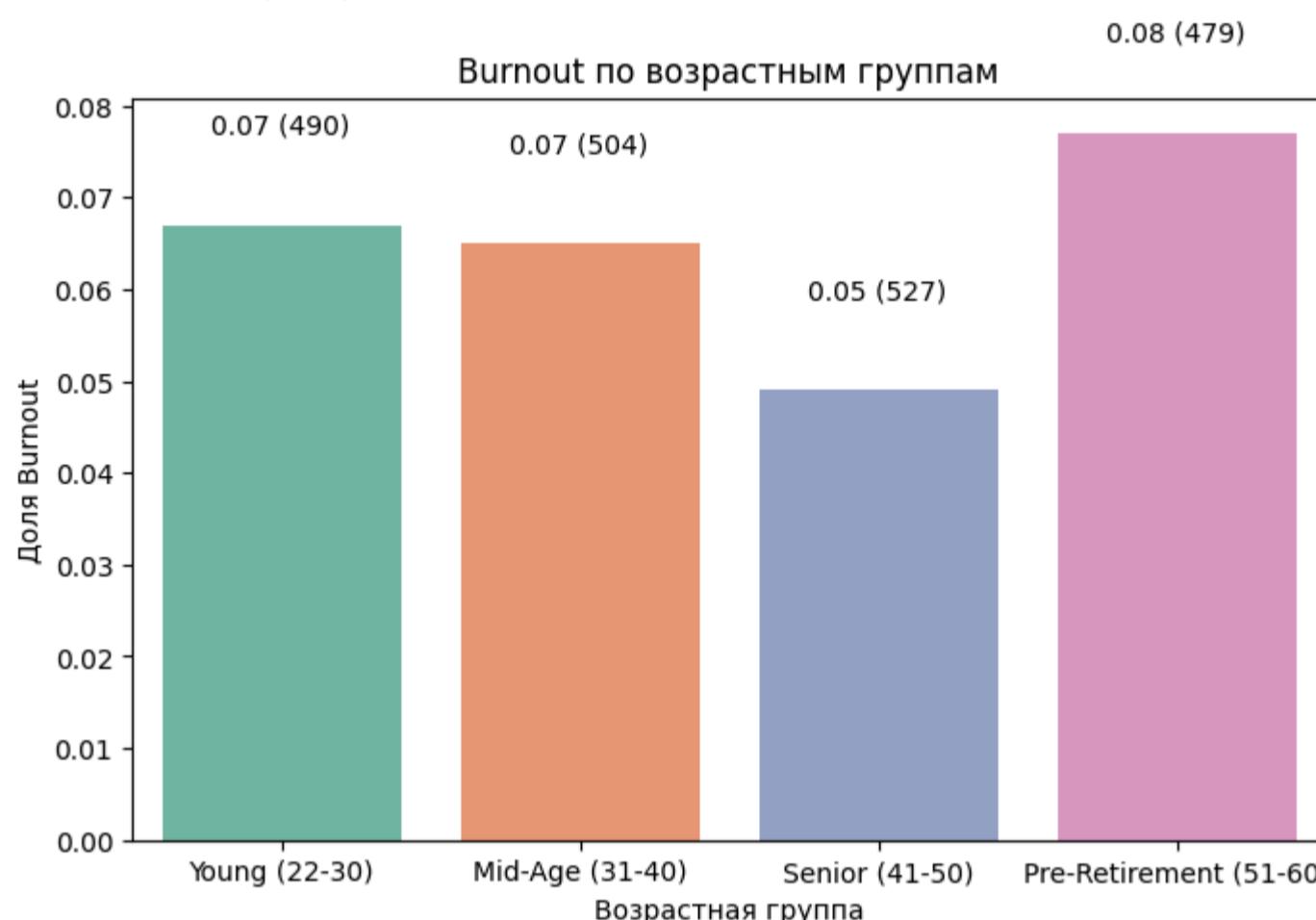
Н1: хотя бы одна возрастная группа имеет статистически значимо другой уровень Burnout

F-статистика = 1.123, p-value = 0.3382350594556378

→ не отвергаем Н0 (гипотеза не подтверждается)

Средние значения Burnout по возрастным группам:

	age_bin	mean_burnout	count
0	Young (22-30)	0.067	490
1	Mid-Age (31-40)	0.065	504
2	Senior (41-50)	0.049	527
3	Pre-Retirement (51-60)	0.077	479



## Выводы по шагу 5

### Подтверждены гипотезы:

- **H3 (переработка >45 часов)**
  - Подтверждается. Сотрудники, работающие более 45 часов, имеют значительно более высокий риск выгорания.
- **H5 (стресс выше у Burnout=1)**
  - Подтверждается. Средний уровень стресса в группе Burnout=1 (8.95) значительно выше, чем в группе Burnout=0 (5.19).
- **H6 (удовлетворённость ниже у Burnout=1)**
  - Подтверждается. Средняя удовлетворённость в группе Burnout=1 (1.97) значительно ниже, чем в группе Burnout=0 (3.07).
- **H8 (work\_hours\_per\_week у Burnout=1 выше)**
  - средние рабочие часы у сотрудников с Burnout=1 значительно выше.
- **H9 (overload\_index у Burnout=1 выше)**
  - индекс перегрузки (overload\_index) у Burnout=1 значительно выше.
- **H10 (stress\_to\_satisfaction у Burnout=1 выше)**
  - дисбаланс стресс/удовлетворённость (stress\_to\_satisfaction) резко выше у Burnout=1.

### Не подтверждены гипотезы:

- **H1 (различия по должностям)**
  - Не подтверждается. Различия между должностями статистически не значимы
- **H2 (удалёнка снижает риск)**
  - Не подтверждается. Формат работы (onsite/hybrid/remote) сам по себе не влияет на риск
- **H4 (стресс ↔ удовлетворённость)**
  - Не подтверждается. В данных нет значимой линейной или монотонной связи между уровнем стресса и удовлетворённости.
- **H7 (гендерные различия)**
  - Не подтверждается. Уровень выгорания у мужчин и женщин статистически не различается.
- **H11(возрастные группы)**
  - различия в Burnout между возрастными группами незначимы.

### Ключевые факторы Burnout:

- Переработка (рабочие часы)

- Критический порог: 45 часов в неделю
- Значительное увеличение риска при превышении порога
- Высокий стресс и низкая удовлетворённость
  - Существенные различия между группами с выгоранием и без
- Индекс перегрузки и дисбаланс стресс/удовлетворённость.

Факторы, не влияющие напрямую:

- Должность
- Формат работы (удалёнка/гибрид/офис)
- Пол
- Возрастные группы.

Практическая ценность для HR

- Фокус на нагрузке и балансе: контроль рабочих часов, особенно >45.
- Мониторинг стресса и удовлетворённости: ключевые индикаторы риска.
- Профилактические меры: внедрить систему раннего предупреждения и разработать программы поддержки при высоком стрессе
- Индекс перегрузки и дисбаланс стресс/удовлетворённость можно использовать как индикаторы раннего выявления сотрудников группы риска.
- Удалёнка и должность сами по себе не определяют Burnout — важнее сочетание нагрузки и субъективных факторов.
- Гендерные различия несущественны → профилактика должна быть универсальной.

Полученные результаты помогают селектировать и интерпретировать признаки.

- Ключевыми для модели будут work\_hours\_per\_week, stress\_level, satisfaction\_level, а также созданные инженерные признаки (overload\_index, stress\_to\_satisfaction), которые улавливают комплексный эффект.
- Признаки job\_role, gender, remote\_cat могут иметь низкую важность в модели, что согласуется с результатами тестов. Их можно оставить для проверки, но не ожидать от них сильного влияния.
- Подтверждена бизнес-логика: для прогноза критично не пропустить случаи выгорания (высокий Recall), даже ценой ложных срабатываний.

\* к содержанию

## Шаг 6: Подготовка данных к обучению моделей

### Выбор признаков для моделей

In [118...]

```
interval_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Вычисление Phik-матрицы (исключаем 'id')
phik_corr = df.phik_matrix(interval_cols=interval_cols)

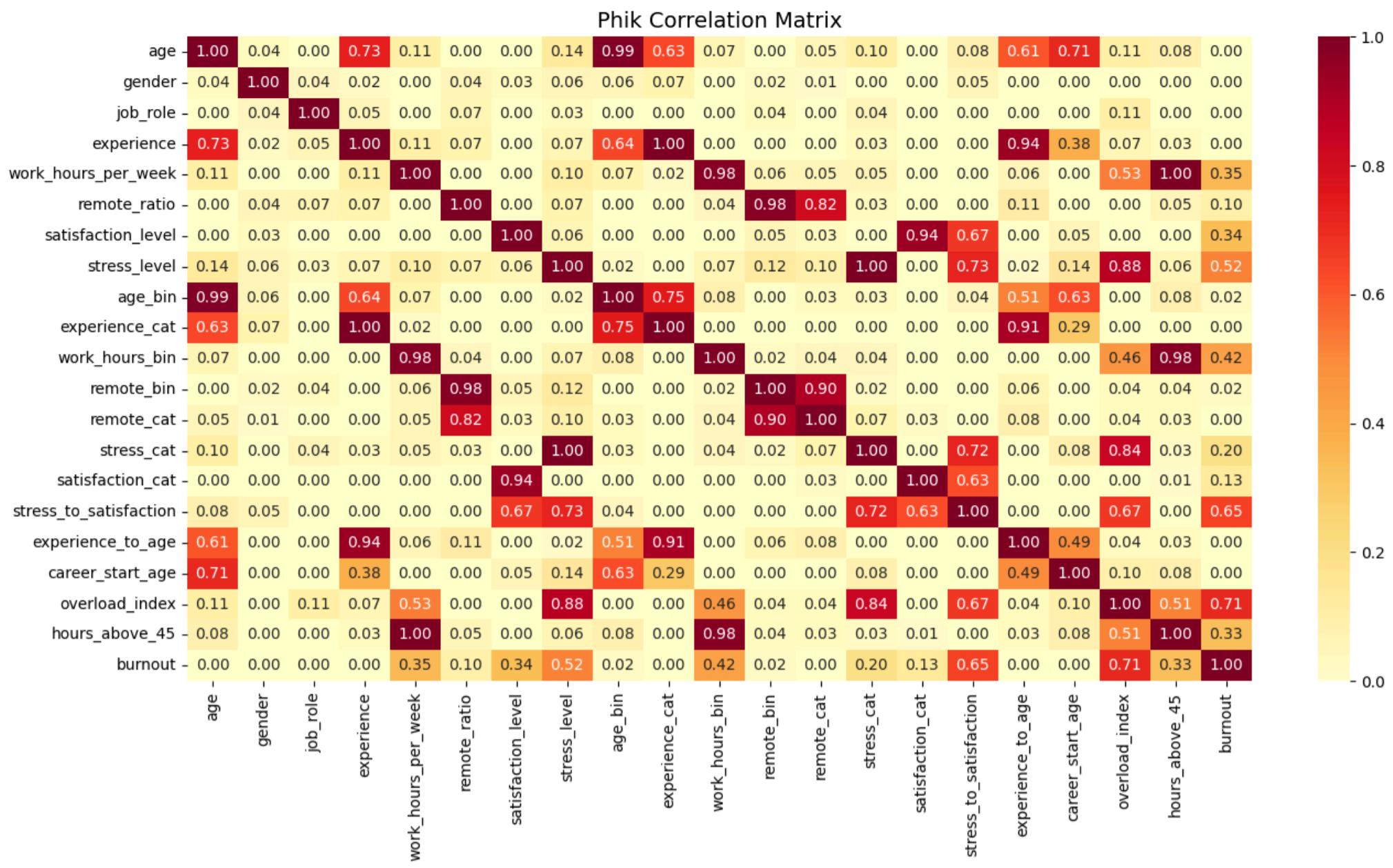
# 📈 Корреляции с целевой переменной burnout
bpm_phik = phik_corr['burnout'].drop('burnout').sort_values(ascending=False)
print("📊 Phik-корреляции с целевой переменной 'burnout':")
print(bpm_phik)

# 🔥 Визуализация полной Phik-матрицы
plt.figure(figsize=(14, 8))
sns.heatmap(phik_corr, annot=True, fmt=".2f", cmap='YlOrRd')
plt.title('Phik Correlation Matrix', fontsize=14)
plt.tight_layout()
plt.show()
```

📊 Phik-корреляции с целевой переменной 'burnout':

overload_index	0.712878
stress_to_satisfaction	0.652573
stress_level	0.515859
work_hours_bin	0.422204
work_hours_per_week	0.347226
satisfaction_level	0.338775
hours_above_45	0.328699
stress_cat	0.197368
satisfaction_cat	0.134035
remote_ratio	0.095902
age_bin	0.020605
remote_bin	0.019079
experience_cat	0.000000
gender	0.000000
remote_cat	0.000000
experience	0.000000
experience_to_age	0.000000
career_start_age	0.000000
job_role	0.000000
age	0.000000

Name: burnout, dtype: float64



In [119]:

```
# Исключаем целевые колонки из матрицы
cols_to_exclude = ['burnout']
phik_corr_filtered = phik_corr.drop(index=cols_to_exclude, columns=cols_to_exclude, errors='ignore')

# Строим маску для верхнего треугольника с порогом корреляции
high_corr_mask = np.triu(np.abs(phik_corr_filtered)) > 0.5, k=1)

# Выводим пары с высокой Phi-корреляцией
print("Пары признаков с Phi-корреляцией > 0.5 (без 'burnout':")
high_corr_pairs = phik_corr_filtered.where(high_corr_mask).stack().dropna().sort_values(ascending=False)
print(high_corr_pairs if not high_corr_pairs.empty else "Нет пар с высокой корреляцией.")
```

Пары признаков с Phi-корреляцией > 0.5 (без 'burnout':

work_hours_per_week	hours_above_45	1.000000
experience	experience_cat	1.000000
stress_level	stress_cat	1.000000
age	age_bin	0.985349
remote_ratio	remote_bin	0.983069
work_hours_bin	hours_above_45	0.975675
work_hours_per_week	work_hours_bin	0.975594
satisfaction_level	satisfaction_cat	0.943874
experience	experience_to_age	0.942351
experience_cat	experience_to_age	0.909844
remote_bin	remote_cat	0.899628
stress_level	overload_index	0.882362
stress_cat	overload_index	0.837908
remote_ratio	remote_cat	0.822092
age_bin	experience_cat	0.752749
age	experience	0.731184
stress_level	stress_to_satisfaction	0.726624
stress_cat	stress_to_satisfaction	0.718589
age	career_start_age	0.711630
stress_to_satisfaction	overload_index	0.674411
satisfaction_level	stress_to_satisfaction	0.672601
experience	age_bin	0.640476
age	experience_cat	0.632891
age_bin	career_start_age	0.631709
satisfaction_cat	stress_to_satisfaction	0.629111
age	experience_to_age	0.613161
work_hours_per_week	overload_index	0.529936
age_bin	experience_to_age	0.514471
overload_index	hours_above_45	0.514328

Определены признаки с высокой корреляцией с burnout:

- overload\_index (0.71)
- stress\_to\_satisfaction (0.65)
- stress\_level (0.52)
- work\_hours\_bin (0.42)
- work\_hours\_per\_week (0.35)
- satisfaction\_level (0.34)
- hours\_above\_45 (0.33)

Выявлены пары признаков с очень высокой взаимной корреляцией (>0.9), например:

- work\_hours\_per\_week ↔ hours\_above\_45
- experience ↔ experience\_cat
- stress\_level ↔ stress\_cat
- age ↔ age\_bin
- remote\_ratio ↔ remote\_bin
- satisfaction\_level ↔ satisfaction\_cat

Это значит, что часть признаков дублирует друг друга и может вызвать мультиколлинеарность.

### Выбор признаков для модели

- Оставляем сильные и интерпретируемые признаки: satisfaction\_level, work\_hours\_per\_week.
- Категориальные признаки: job\_role, gender, remote\_cat, experience\_cat, age\_bin.
- Исключаем дубликаты (hours\_above\_45, work\_hours\_bin, stress\_cat, satisfaction\_cat, remote\_bin, experience\_to\_age, career\_start\_age) — они коррелируют почти на 1.0 с базовыми признаками.
- чтобы учесть возможную утечку целевой исключим высококоррелируемые признаки 'overload\_index','stress\_to\_satisfaction','stress\_level',

In [120...]

```
features = [
    'satisfaction_level',
    'work_hours_per_week',
    'job_role',
    'gender',
    'remote_cat',
    'remote_ratio',
    'experience_cat',
    'age_bin',
    'burnout'
]
```

In [121...]

```
df_ml=df[features]
```

In [122...]

```
df_ml.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   satisfaction_level  2000 non-null   float64
 1   work_hours_per_week 2000 non-null   int64  
 2   job_role            2000 non-null   object 
 3   gender              2000 non-null   object 
 4   remote_cat          2000 non-null   category
 5   remote_ratio        2000 non-null   int64  
 6   experience_cat     2000 non-null   category
 7   age_bin             2000 non-null   category
 8   burnout             2000 non-null   int64  
dtypes: category(3), float64(1), int64(3), object(2)
memory usage: 100.3+ KB
```

In [123...]

```
df_ml.describe().T
```

Out[123...]

	count	mean	std	min	25%	50%	75%	max
<b>satisfaction_level</b>	2000.0	2.99523	1.155431	1.0	2.0	3.025	4.0	5.0
<b>work_hours_per_week</b>	2000.0	49.58800	11.832424	30.0	39.0	49.000	60.0	70.0
<b>remote_ratio</b>	2000.0	49.97300	29.151298	0.0	24.0	49.000	75.0	100.0
<b>burnout</b>	2000.0	0.06450	0.245703	0.0	0.0	0.000	0.0	1.0

In [124...]

```
temp = df_ml.copy()

list_c = temp.select_dtypes(include=['object','category']).columns
print(temp[list_c].info())

for col_1 in list_c:
    print('-' * 25)
    print(col_1, temp[col_1].sort_values().unique())
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   job_role    2000 non-null   object  
 1   gender      2000 non-null   object  
 2   remote_cat  2000 non-null   category
 3   experience_cat 2000 non-null   category
 4   age_bin     2000 non-null   category
dtypes: category(3), object(2)
memory usage: 37.8+ KB
None
-----
job_role ['Analyst' 'Engineer' 'HR' 'Manager' 'Sales']
-----
gender ['Female' 'Male']
-----
remote_cat ['onsite', 'hybrid', 'remote']
Categories (3, object): ['onsite' < 'hybrid' < 'remote']
-----
experience_cat ['junior', 'middle', 'senior', 'expert']
Categories (4, object): ['junior' < 'middle' < 'senior' < 'expert']
-----
age_bin ['Young (22-30)', 'Mid-Age (31-40)', 'Senior (41-50)', 'Pre-Retirement (51-60)']
Categories (4, object): ['Young (22-30)' < 'Mid-Age (31-40)' < 'Senior (41-50)' < 'Pre-Retirement (51-60)']

```

In [125...]

```

# Вычисление Phik-матрицы (исключаем 'id')
phik_corr_ml = df_ml.phik_matrix(interval_cols=df_ml.select_dtypes('number').columns)

# 📈 Корреляции burnout со всеми признаками
print("📊 Phik-корреляции с целевой переменной 'burnout':")
print(phik_corr_ml['burnout'].drop('burnout').sort_values(ascending=False))

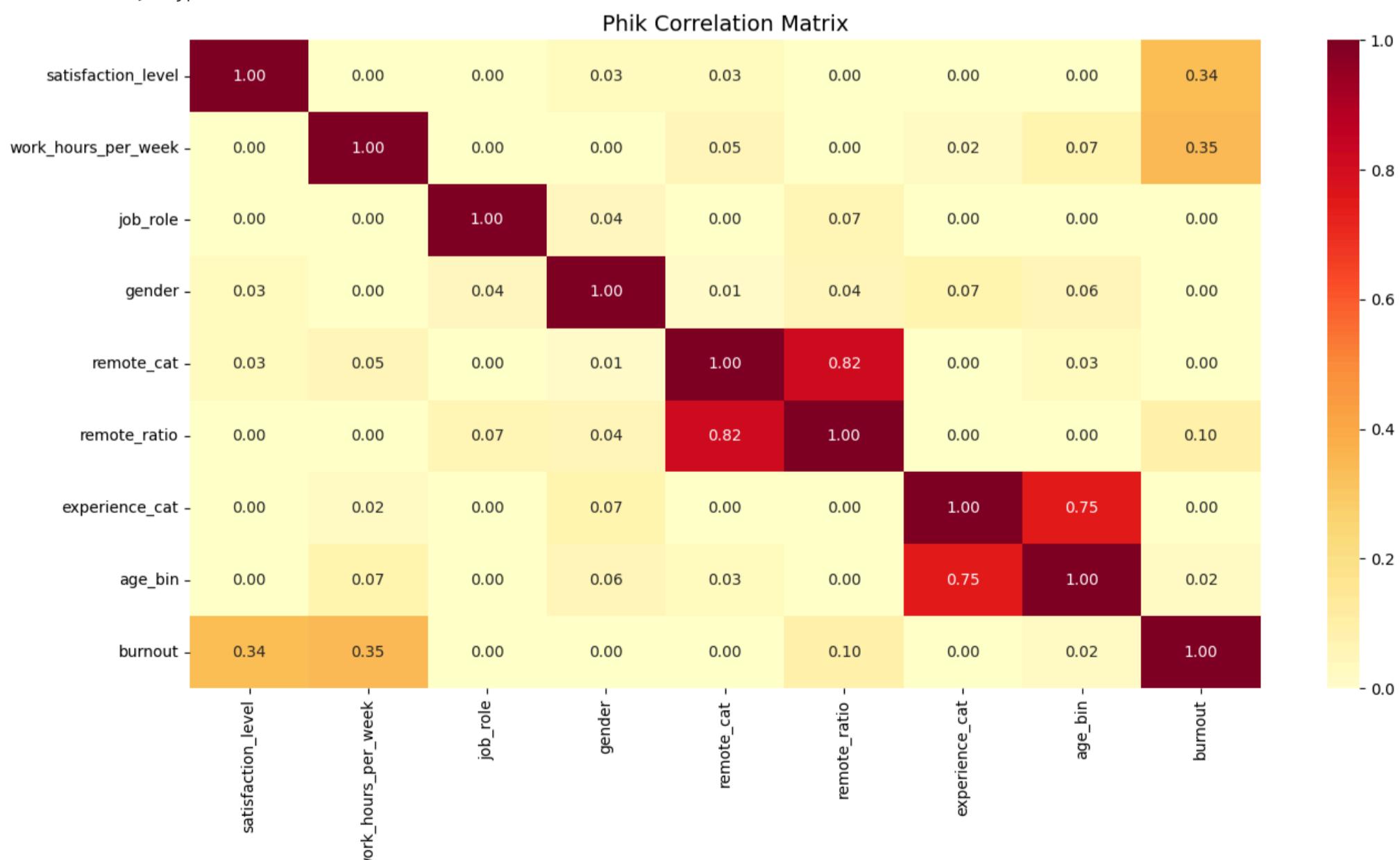
# 🔥 Визуализация полной Phik-матрицы
plt.figure(figsize=(14, 8))
sns.heatmap(phik_corr_ml, annot=True, fmt=".2f", cmap='YlOrRd')
plt.title('Phik Correlation Matrix', fontsize=14)
plt.tight_layout()
plt.show()

```

```

📊 Phik-корреляции с целевой переменной 'burnout':
work_hours_per_week  0.347226
satisfaction_level  0.338775
remote_ratio         0.095902
age_bin              0.020605
job_role             0.000000
gender               0.000000
remote_cat           0.000000
experience_cat       0.000000
Name: burnout, dtype: float64

```



Разделение данных на признаки и целевую переменную

In [126...]

```
# Разделение на признаки и целевую переменную
X = df_ml.drop('burnout', axis=1)
y = df_ml['burnout']
```

## Разделение на train/valid/test, стратификация по Burnout

In [127...]

```
# Train/Valid/Test с стратификацией
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42
)
X_valid, X_test, y_valid, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42
)

print(f"X_train: {X_train.shape}, X_valid: {X_valid.shape}, X_test: {X_test.shape}")
print(f"y_train: {y_train.shape}, y_valid: {y_valid.shape}, y_test: {y_test.shape}")

X_train: (1400, 8), X_valid: (300, 8), X_test: (300, 8)
y_train: (1400,), y_valid: (300,), y_test: (300,)
```

In [128...]

```
# Визуализация распределения у до и после разделения для задачи Burnout
fig, axes = plt.subplots(1, 4, figsize=(22, 8))

# Исходное распределение
y.value_counts(normalize=True).plot(kind='bar', ax=axes[0], color='skyblue')
axes[0].set_title('Исходное распределение у\n(Burnout)', fontsize=22)
axes[0].set_ylabel('Доля', fontsize=20)
axes[0].set_xlabel('Burnout', fontsize=20)

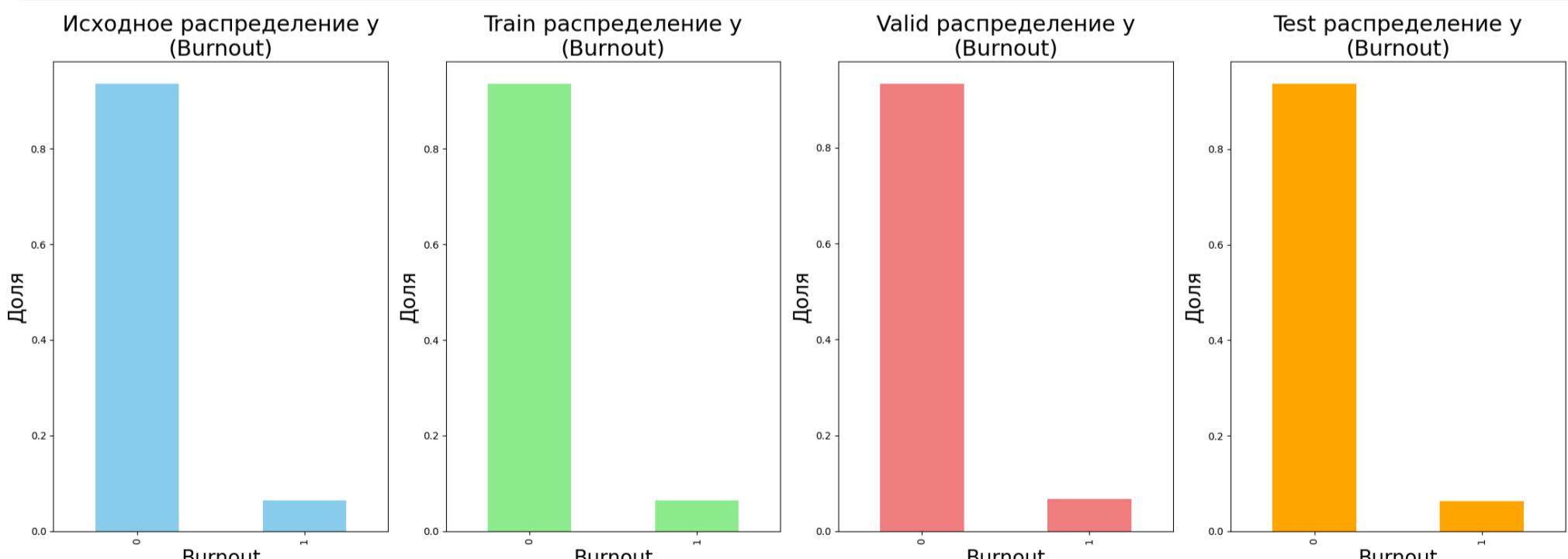
# Train распределение
y_train.value_counts(normalize=True).plot(kind='bar', ax=axes[1], color='lightgreen')
axes[1].set_title('Train распределение у\n(Burnout)', fontsize=22)
axes[1].set_ylabel('Доля', fontsize=20)
axes[1].set_xlabel('Burnout', fontsize=20)

# Valid распределение
y_valid.value_counts(normalize=True).plot(kind='bar', ax=axes[2], color='lightcoral')
axes[2].set_title('Valid распределение у\n(Burnout)', fontsize=22)
axes[2].set_ylabel('Доля', fontsize=20)
axes[2].set_xlabel('Burnout', fontsize=20)

# Test распределение
y_test.value_counts(normalize=True).plot(kind='bar', ax=axes[3], color='orange')
axes[3].set_title('Test распределение у\n(Burnout)', fontsize=22)
axes[3].set_ylabel('Доля', fontsize=20)
axes[3].set_xlabel('Burnout', fontsize=20)

plt.tight_layout()
plt.show()

# Числовая проверка
print("Проверка стратификации для Burnout:")
print(f"Исходные данные: {y.value_counts(normalize=True).values}")
print(f"Train выборка: {y_train.value_counts(normalize=True).values}")
print(f"Valid выборка: {y_valid.value_counts(normalize=True).values}")
print(f"Test выборка: {y_test.value_counts(normalize=True).values}")
```



Проверка стратификации для Burnout:

```
Исходные данные: [0.9355 0.0645]
Train выборка: [0.93571429 0.06428571]
Valid выборка: [0.93333333 0.06666667]
Test выборка: [0.93666667 0.06333333]
```

In [129...]

```
def check_columns_order(X_train, X_valid, X_test):
    cols_train = list(X_train.columns)
    cols_valid = list(X_valid.columns)
    cols_test = list(X_test.columns)

    print("✓ Колонки в Train:", cols_train)
    print("✓ Колонки в Valid:", cols_valid)
```

```

print("✅ Колонки в Test:", cols_test)

if cols_train == cols_valid == cols_test:
    print("\nВсе три датафрейма имеют одинаковые названия и порядок колонок.")
else:
    print("\n⚠️ Есть различия в названиях или порядке колонок:")
    if cols_train != cols_valid:
        print(" - Train и Valid отличаются")
    if cols_train != cols_test:
        print(" - Train и Test отличаются")
    if cols_valid != cols_test:
        print(" - Valid и Test отличаются")

# Вызов функции
check_columns_order(X_train, X_valid, X_test)

```

✅ Колонки в Train: ['satisfaction\_level', 'work\_hours\_per\_week', 'job\_role', 'gender', 'remote\_cat', 'remote\_ratio', 'experience\_cat', 'age\_bin']  
 ✅ Колонки в Valid: ['satisfaction\_level', 'work\_hours\_per\_week', 'job\_role', 'gender', 'remote\_cat', 'remote\_ratio', 'experience\_cat', 'age\_bin']  
 ✅ Колонки в Test: ['satisfaction\_level', 'work\_hours\_per\_week', 'job\_role', 'gender', 'remote\_cat', 'remote\_ratio', 'experience\_cat', 'age\_bin']

Все три датафрейма имеют одинаковые названия и порядок колонок.

In [130...]

```

def check_unique_categories(X_train, X_valid, X_test):
    # Определяем категориальные признаки
    cat_cols = X_train.select_dtypes(include=['object', 'category']).columns

    for col in cat_cols:
        train_vals = set(X_train[col].unique())
        valid_vals = set(X_valid[col].unique())
        test_vals = set(X_test[col].unique())

        print(f"\n🔍 Проверка колонки: {col}")
        print(f"Train уникальные: {train_vals}")
        print(f"Valid уникальные: {valid_vals}")
        print(f"Test уникальные: {test_vals}")

        # Проверка совпадения
        if train_vals == valid_vals == test_vals:
            print("✅ Все три выборки имеют одинаковые уникальные значения.")
        else:
            print("⚠️ Есть различия:")
            print(" - Только в Train:", train_vals - valid_vals - test_vals)
            print(" - Только в Valid:", valid_vals - train_vals - test_vals)
            print(" - Только в Test:", test_vals - train_vals - valid_vals)

# Вызов функции
check_unique_categories(X_train, X_valid, X_test)

```

🔎 Проверка колонки: job\_role  
 Train уникальные: {'HR', 'Analyst', 'Manager', 'Sales', 'Engineer'}  
 Valid уникальные: {'HR', 'Analyst', 'Manager', 'Sales', 'Engineer'}  
 Test уникальные: {'HR', 'Analyst', 'Manager', 'Sales', 'Engineer'}  
 ✅ Все три выборки имеют одинаковые уникальные значения.

🔎 Проверка колонки: gender  
 Train уникальные: {'Male', 'Female'}  
 Valid уникальные: {'Male', 'Female'}  
 Test уникальные: {'Male', 'Female'}  
 ✅ Все три выборки имеют одинаковые уникальные значения.

🔎 Проверка колонки: remote\_cat  
 Train уникальные: {'hybrid', 'remote', 'onsite'}  
 Valid уникальные: {'hybrid', 'remote', 'onsite'}  
 Test уникальные: {'hybrid', 'remote', 'onsite'}  
 ✅ Все три выборки имеют одинаковые уникальные значения.

🔎 Проверка колонки: experience\_cat  
 Train уникальные: {'senior', 'expert', 'junior', 'middle'}  
 Valid уникальные: {'senior', 'expert', 'junior', 'middle'}  
 Test уникальные: {'senior', 'expert', 'junior', 'middle'}  
 ✅ Все три выборки имеют одинаковые уникальные значения.

🔎 Проверка колонки: age\_bin  
 Train уникальные: {'Senior (41-50)', 'Pre-Retirement (51-60)', 'Young (22-30)', 'Mid-Age (31-40)'}  
 Valid уникальные: {'Senior (41-50)', 'Pre-Retirement (51-60)', 'Young (22-30)', 'Mid-Age (31-40)'}  
 Test уникальные: {'Senior (41-50)', 'Pre-Retirement (51-60)', 'Young (22-30)', 'Mid-Age (31-40)'}  
 ✅ Все три выборки имеют одинаковые уникальные значения.

In [131...]

```

def compare_category_distributions(X_train, X_valid, X_test):
    # Определяем категориальные признаки
    cat_cols = X_train.select_dtypes(include=['object', 'category']).columns

    for col in cat_cols:
        print(f"\n🔍 Колонка: {col}")

        # Распределения категорий (нормированные частоты)
        train_dist = X_train[col].value_counts(normalize=True)
        valid_dist = X_valid[col].value_counts(normalize=True)
        test_dist = X_test[col].value_counts(normalize=True)

        # Объединяем в одну таблицу

```

```

dist_df = pd.concat([train_dist, valid_dist, test_dist], axis=1)
dist_df.columns = ['Train', 'Valid', 'Test']
dist_df = dist_df.fillna(0) # если категория отсутствует в выборке

print(dist_df.round(3))

# Вызов функции
compare_category_distributions(X_train, X_valid, X_test)

```

🔍 Колонка: job\_role

	Train	Valid	Test
Manager	0.211	0.207	0.203
Analyst	0.204	0.190	0.233
Sales	0.200	0.213	0.157
HR	0.194	0.197	0.203
Engineer	0.191	0.193	0.203

🔍 Колонка: gender

	Train	Valid	Test
Male	0.509	0.523	0.513
Female	0.491	0.477	0.487

🔍 Колонка: remote\_cat

	Train	Valid	Test
hybrid	0.786	0.797	0.780
remote	0.207	0.200	0.213
onsite	0.007	0.003	0.007

🔍 Колонка: experience\_cat

	Train	Valid	Test
junior	0.314	0.300	0.280
senior	0.246	0.237	0.277
expert	0.245	0.267	0.247
middle	0.196	0.197	0.197

🔍 Колонка: age\_bin

	Train	Valid	Test
Senior (41-50)	0.265	0.253	0.267
Young (22-30)	0.246	0.250	0.237
Mid-Age (31-40)	0.246	0.270	0.263
Pre-Retirement (51-60)	0.244	0.227	0.233

```
In [132...]: def compare_numeric_distributions(X_train, X_valid, X_test):
    num_cols = X_train.select_dtypes(include=['int64', 'float64']).columns

    for col in num_cols:
        print(f"\n🔍 Колонка: {col}")

        # Базовые статистики
        stats_df = pd.DataFrame({
            'Train': [X_train[col].mean(), X_train[col].std(), X_train[col].min(), X_train[col].max()],
            'Valid': [X_valid[col].mean(), X_valid[col].std(), X_valid[col].min(), X_valid[col].max()],
            'Test': [X_test[col].mean(), X_test[col].std(), X_test[col].min(), X_test[col].max()]
        }, index=['Mean', 'Std', 'Min', 'Max'])

        print(stats_df.round(3))

compare_numeric_distributions(X_train, X_valid, X_test)
```

🔍 Колонка: satisfaction\_level

	Train	Valid	Test
Mean	2.980	3.061	2.999
Std	1.161	1.153	1.132
Min	1.000	1.020	1.010
Max	5.000	4.990	4.950

🔍 Колонка: work\_hours\_per\_week

	Train	Valid	Test
Mean	49.786	48.577	49.677
Std	11.931	11.739	11.446
Min	30.000	30.000	30.000
Max	70.000	70.000	70.000

🔍 Колонка: remote\_ratio

	Train	Valid	Test
Mean	49.880	50.410	49.97
Std	29.086	29.189	29.51
Min	0.000	0.000	0.00
Max	100.000	100.000	100.00

## Предобработка данных: масштабирование числовых признаков и кодирование категориальных признаков (One-Hot Encoding, Order Encoding, Label Encoding)

```
In [133...]: def prepare_features_model(X_train, X_valid, X_test):
    # 1. Определение типов признаков
    num_col_names = [
        'satisfaction_level',
        'work_hours_per_week',
        'remote_ratio'
    ]

    # Номинальные категориальные (One-Hot)
    nominal_cols = ['job_role', 'gender']
```

```

# Упорядоченные категориальные (Ordinal)
ordinal_cols = ['remote_cat', 'experience_cat', 'age_bin']

# 2. OneHotEncoder для номинальных
ohe = OneHotEncoder(
    sparse_output=False,
    drop='first',
    handle_unknown="ignore"
)
X_train_ohe = ohe.fit_transform(X_train[nominal_cols])
X_valid_ohe = ohe.transform(X_valid[nominal_cols])
X_test_ohe = ohe.transform(X_test[nominal_cols])

encoder_ohe_names = ohe.get_feature_names_out(nominal_cols)
X_train_ohe_df = pd.DataFrame(X_train_ohe, columns=encoder_ohe_names, index=X_train.index)
X_valid_ohe_df = pd.DataFrame(X_valid_ohe, columns=encoder_ohe_names, index=X_valid.index)
X_test_ohe_df = pd.DataFrame(X_test_ohe, columns=encoder_ohe_names, index=X_test.index)

# 3. OrdinalEncoder для упорядоченных
ordinal_mapping = [
    ['onsite', 'hybrid', 'remote'], # remote_cat
    ['junior', 'middle', 'senior', 'expert'], # experience_cat
    ['Young (22-30)', 'Mid-Age (31-40)', 'Senior (41-50)', 'Pre-Retirement (51-60)'] # age_bin
]

oe = OrdinalEncoder(categories=ordinal_mapping)
X_train_ord = oe.fit_transform(X_train[ordinal_cols])
X_valid_ord = oe.transform(X_valid[ordinal_cols])
X_test_ord = oe.transform(X_test[ordinal_cols])

X_train_ord_df = pd.DataFrame(X_train_ord, columns=ordinal_cols, index=X_train.index)
X_valid_ord_df = pd.DataFrame(X_valid_ord, columns=ordinal_cols, index=X_valid.index)
X_test_ord_df = pd.DataFrame(X_test_ord, columns=ordinal_cols, index=X_test.index)

# 4. Масштабирование числовых
scaler = StandardScaler()
X_train_num = scaler.fit_transform(X_train[num_col_names])
X_valid_num = scaler.transform(X_valid[num_col_names])
X_test_num = scaler.transform(X_test[num_col_names])

X_train_num_df = pd.DataFrame(X_train_num, columns=num_col_names, index=X_train.index)
X_valid_num_df = pd.DataFrame(X_valid_num, columns=num_col_names, index=X_valid.index)
X_test_num_df = pd.DataFrame(X_test_num, columns=num_col_names, index=X_test.index)

# 5. Объединение всех частей
X_train_final = pd.concat([X_train_num_df, X_train_ohe_df, X_train_ord_df], axis=1)
X_valid_final = pd.concat([X_valid_num_df, X_valid_ohe_df, X_valid_ord_df], axis=1)
X_test_final = pd.concat([X_test_num_df, X_test_ohe_df, X_test_ord_df], axis=1)

return X_train_final, X_valid_final, X_test_final, ohe, oe, scaler

```

In [134...]

```

# Вызов функции
X_train_f, X_valid_f, X_test_f, ohe, oe, scaler = prepare_features_model(
    X_train, X_valid, X_test
)

# Проверим размеры
print("Train:", X_train_f.shape)
print("Valid:", X_valid_f.shape)
print("Test:", X_test_f.shape)

```

Train: (1400, 11)  
 Valid: (300, 11)  
 Test: (300, 11)

In [135...]

```
X_train_f.isnull().sum().sum()
```

Out[135...]

0

In [136...]

```
X_valid_f.isnull().sum().sum()
```

Out[136...]

0

In [137...]

```
X_test_f.isnull().sum().sum()
```

Out[137...]

0

In [138...]

```
# Вызов функции
check_columns_order(X_train_f, X_valid_f, X_test_f)
```

✓ Колонки в Train: ['satisfaction\_level', 'work\_hours\_per\_week', 'remote\_ratio', 'job\_role\_Engineer', 'job\_role\_HR', 'job\_role\_Manager', 'job\_role\_Sales', 'gender\_Male', 'remote\_cat', 'experience\_cat', 'age\_bin']  
 ✓ Колонки в Valid: ['satisfaction\_level', 'work\_hours\_per\_week', 'remote\_ratio', 'job\_role\_Engineer', 'job\_role\_HR', 'job\_role\_Manager', 'job\_role\_Sales', 'gender\_Male', 'remote\_cat', 'experience\_cat', 'age\_bin']  
 ✓ Колонки в Test: ['satisfaction\_level', 'work\_hours\_per\_week', 'remote\_ratio', 'job\_role\_Engineer', 'job\_role\_HR', 'job\_role\_Manager', 'job\_role\_Sales', 'gender\_Male', 'remote\_cat', 'experience\_cat', 'age\_bin']

Все три датафрейма имеют одинаковые названия и порядок колонок.

## Выводы по шагу 6

#### Обоснованный отбор признаков:

- С помощью Phik-корреляции выделены признаки, наиболее сильно связанные с выгоранием (burnout):
  - Топ-3: overload\_index (0.71), stress\_to\_satisfaction (0.65), stress\_level (0.52).
- Выявлены и исключены дубликаты признаков с корреляцией ~1.0 (например, work\_hours\_per\_week и hours\_above\_45), что предотвращает мультиколлинеарность.

#### Корректное разделение данных:

- Данные разделены на Train (70%), Valid (15%), Test (15%).
- Применена стратификация по целевому признаку, что сохранило дисбаланс классов (~6.5% выгорания) во всех выборках.
- Выполнены проверки на сохранение порядка колонок и уникальных значений в категориальных признаках — всё в порядке.

Профессиональная предобработка:

#### Числовые признаки стандартизированы (StandardScaler).

- Категориальные признаки закодированы:
  - One-Hot Encoding для номинальных (job\_role, gender).
  - Ordinal Encoding для порядковых (remote\_cat, experience\_cat, age\_bin).
- Итоговое пространство признаков: 14 колонок (6 числовых + 4 от ОНЕ + 3 порядковых).

#### Готовность к моделированию:

- Отсутствуют пропуски.
- Все признаки числовые.
- Выборки согласованы.

[\\* к содержанию](#)

## Шаг 7: Моделирование и интерпретация

Напомню, что из признаков убрал три наиболее коррелируемых с целевой:

 Phik-корреляции с целевой переменной 'burnout':

- overload\_index 0.712878
- stress\_to\_satisfaction 0.652573
- stress\_level 0.515859

Это связано прежде всего с тем, что вместе с этими признаками модели на валидации давали основные метрики, равные 1, что показывает на утчеку и переобучение. Ниже таблица полученных метрик при учете признаков overload\_index, stress\_to\_satisfaction, stress\_level. Поэтому решил перестроить модели без этих признаков

Модель	Recall	Precision	F1-score	ROC-AUC	PR-AUC
Dummy	0.0	0.0000	0.0000	NaN	NaN
Logistic Regression	1.0	0.2899	0.4494	0.9777	0.8018
Decision Tree	1.0	0.8000	0.8889	0.9911	0.8000
Random Forest	1.0	1.0000	1.0000	1.0000	1.0000
XGBoost	1.0	1.0000	1.0000	1.0000	1.0000
LightGBM	1.0	1.0000	1.0000	1.0000	1.0000
CatBoost	1.0	1.0000	1.0000	1.0000	1.0000

In [139...]

```
# F2-мера: beta=2 означает, что Recall в 2 раза важнее Precision
f2_scoring = make_scorer(fbeta_score, beta=2, pos_label=1)
```

### Базовые модели: Dummy

In [140...]

```
# ⏱ Засекаем время
start = time.time()

# 💡 Инициализация и обучение DummyClassifier
dummy = DummyClassifier(strategy="most_frequent", random_state=42)
dummy.fit(X_train_f, y_train)

# ✎ Предсказание на валидации
y_dummy_pred = dummy.predict(X_valid_f)
y_dummy_proba = dummy.predict_proba(X_valid_f)[:, 1] if hasattr(dummy, "predict_proba") else None

# 🌟 Оценка качества по Recall для класса 1 (Burnout)
recall_dummy = recall_score(y_valid, y_dummy_pred, pos_label=1)
precision_dummy = precision_score(y_valid, y_dummy_pred, pos_label=1)
```

```

f1_dummy = f1_score(y_valid, y_dummy_pred, pos_label=1)
accuracy_dummy = accuracy_score(y_valid, y_dummy_pred)

print(f"📊 Результаты DummyClassifier (strategy='most_frequent'):")
print(f"    Recall (Burnout=1): {recall_dummy:.4f}")
print(f"    Precision (Burnout=1): {precision_dummy:.4f}")
print(f"    F1-score:           {f1_dummy:.4f}")
print(f"    Accuracy:          {accuracy_dummy:.4f}")

# 🕒 Время выполнения
print(f"\n⌚ Время выполнения: {time.time() - start:.2f} сек")

📊 Результаты DummyClassifier (strategy='most_frequent'):
Recall (Burnout=1): 0.0000
Precision (Burnout=1): 0.0000
F1-score:           0.0000
Accuracy:          0.9333

⌚ Время выполнения: 0.02 сек

```

In [141...]

```

# 🕒 Засекаем время
start = time.time()

# 💡 Инициализация и обучение DummyClassifier
dummy = DummyClassifier(strategy="most_frequent", random_state=42)
dummy.fit(X_train_f, y_train)

# ✅ Предсказание на валидации
y_dummy_pred = dummy.predict(X_valid_f)
y_dummy_proba = dummy.predict_proba(X_valid_f)[:, 1] if hasattr(dummy, "predict_proba") else None

# ⚪ Оценка качества по Recall для класса 1 (Burnout)
recall_dummy = recall_score(y_valid, y_dummy_pred, pos_label=1)
precision_dummy = precision_score(y_valid, y_dummy_pred, pos_label=1)
f1_dummy = f1_score(y_valid, y_dummy_pred, pos_label=1)
f2_dummy = fbeta_score(y_valid, y_dummy_pred, beta=2, pos_label=1)
accuracy_dummy = accuracy_score(y_valid, y_dummy_pred)

print(f"📊 Результаты DummyClassifier (strategy='most_frequent'):")
print(f"    Recall (Burnout=1): {recall_dummy:.4f}")
print(f"    Precision (Burnout=1): {precision_dummy:.4f}")
print(f"    F1-score:           {f1_dummy:.4f}")
print(f"    F2-score:           {f2_dummy:.4f}")
print(f"    Accuracy:          {accuracy_dummy:.4f}")

# 🕒 Время выполнения
print(f"\n⌚ Время выполнения: {time.time() - start:.2f} сек")

```

```

📊 Результаты DummyClassifier (strategy='most_frequent'):
Recall (Burnout=1): 0.0000
Precision (Burnout=1): 0.0000
F1-score:           0.0000
F2-score:           0.0000
Accuracy:          0.9333

```

⌚ Время выполнения: 0.02 сек

## Линейные модели: Logistic Regression

```

In [142...]
start = time.time()

# Инициализация модели
lr = LogisticRegression(
    random_state=42,
    n_jobs=-1,
    solver='saga',
    max_iter=1000,
    tol=1e-3,
    class_weight="balanced"
)

# Параметры для подбора
params_lr = {
    'C': [0.01, 0.1, 0.5, 1, 2],
    'penalty': ['l1', 'l2']
}

# GridSearchCV с метрикой Recall
lr_gs = GridSearchCV(
    lr,
    params_lr,
    cv=5,
    scoring='recall',
    verbose=0,
    n_jobs=-1
).fit(X_train_f, y_train)

# Лучшие параметры
print(f"🔍 Лучшие параметры модели LogisticRegression: {lr_gs.best_params_}")

```

```

print(f"📊 Средний Recall (CV) для модели LogisticRegression: {lr_gs.best_score_:.4f}")

# Предсказание на валидации
y_lr_pred = lr_gs.best_estimator_.predict(X_valid_f)
y_lr_proba = lr_gs.best_estimator_.predict_proba(X_valid_f)[:, 1]

# Оценка качества
recall_lr = recall_score(y_valid, y_lr_pred, pos_label=1)
precision_lr = precision_score(y_valid, y_lr_pred, pos_label=1)
f1_lr = f1_score(y_valid, y_lr_pred, pos_label=1)
roc_auc_lr = roc_auc_score(y_valid, y_lr_proba)
pr_auc_lr = average_precision_score(y_valid, y_lr_proba)

print(f"\n📈 Результаты на валидации модели LogisticRegression:")
print(f"    Recall:      {recall_lr:.4f}")
print(f"    Precision:   {precision_lr:.4f}")
print(f"    F1-score:    {f1_lr:.4f}")
print(f"    ROC-AUC:    {roc_auc_lr:.4f}")
print(f"    PR-AUC:     {pr_auc_lr:.4f}")
print(f"\n📊 Отчёт по классификации модели LogisticRegression:")
print(classification_report(y_valid, y_lr_pred, digits=3))
print(f"\n⌚ Время выполнения: {time.time() - start:.2f} сек")

# Матрица ошибок
cm_lr = confusion_matrix(y_valid, y_lr_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Матрица ошибок: Logistic Regression')
plt.ylabel('Истинный класс')
plt.xlabel('Предсказанный класс')
plt.show()

```

🔍 Лучшие параметры модели LogisticRegression: {'C': 0.01, 'penalty': 'l1'}  
 📊 Средний Recall (CV) для модели LogisticRegression: 0.8889

📈 Результаты на валидации модели LogisticRegression:

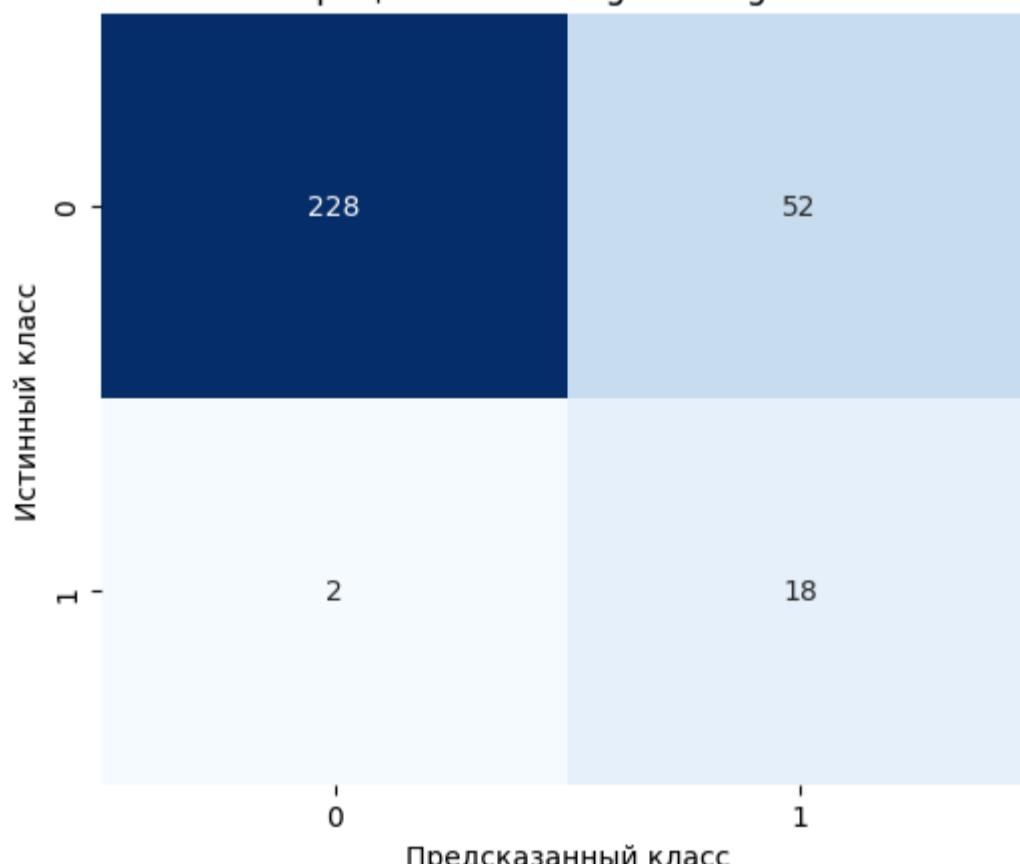
Recall:	0.9000
Precision:	0.2571
F1-score:	0.4000
ROC-AUC:	0.9209
PR-AUC:	0.3146

📊 Отчёт по классификации модели LogisticRegression:

	precision	recall	f1-score	support
0	0.991	0.814	0.894	280
1	0.257	0.900	0.400	20
accuracy			0.820	300
macro avg	0.624	0.857	0.647	300
weighted avg	0.942	0.820	0.861	300

⌚ Время выполнения: 3.17 сек

Матрица ошибок: Logistic Regression



In [143...]

```

start = time.time()

# Инициализация модели
lr = LogisticRegression(
    random_state=42,
    n_jobs=-1,
    solver='saga',
    max_iter=1000,
    tol=1e-3,
    class_weight="balanced"
)

```

```

)
# Параметры для подбора
params_lr = {
    'C': [0.01, 0.1, 0.5, 1, 2],
    'penalty': ['l1', 'l2']
}

# GridSearchCV с метрикой Recall
lr_gs = GridSearchCV(
    lr,
    params_lr,
    cv=5,
    scoring=f2_scorer,
    verbose=0,
    n_jobs=-1
).fit(X_train_f, y_train)

# Лучшие параметры
print(f"🔍 Лучшие параметры модели LogisticRegression: {lr_gs.best_params_}")
print(f"📊 Средний Recall (CV) для модели LogisticRegression: {lr_gs.best_score_.:.4f}")

# Предсказание на валидации
y_lr_pred = lr_gs.best_estimator_.predict(X_valid_f)
y_lr_proba = lr_gs.best_estimator_.predict_proba(X_valid_f)[:, 1]

# Оценка качества
recall_lr = recall_score(y_valid, y_lr_pred, pos_label=1)
precision_lr = precision_score(y_valid, y_lr_pred, pos_label=1)
f1_lr = f1_score(y_valid, y_lr_pred, pos_label=1)
f2_lr = fbeta_score(y_valid, y_lr_pred, beta=2, pos_label=1)
roc_auc_lr = roc_auc_score(y_valid, y_lr_proba)
pr_auc_lr = average_precision_score(y_valid, y_lr_proba)

print(f"\n📈 Результаты на валидации модели LogisticRegression:")
print(f"    Recall:      {recall_lr:.4f}")
print(f"    Precision:   {precision_lr:.4f}")
print(f"    F1-score:    {f1_lr:.4f}")
print(f"    F2-score:    {f2_lr:.4f}")
print(f"    ROC-AUC:    {roc_auc_lr:.4f}")
print(f"    PR-AUC:     {pr_auc_lr:.4f}")
print(f"\n📊 Отчёт по классификации модели LogisticRegression:")
print(classification_report(y_valid, y_lr_pred, digits=3))
print(f"\n⌚ Время выполнения: {time.time() - start:.2f} сек")

# Матрица ошибок
cm_lr = confusion_matrix(y_valid, y_lr_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Матрица ошибок: Logistic Regression')
plt.ylabel('Истинный класс')
plt.xlabel('Предсказанный класс')
plt.show()

```

🔍 Лучшие параметры модели LogisticRegression: {'C': 0.01, 'penalty': 'l1'}  
 📊 Средний Recall (CV) для модели LogisticRegression: 0.5248

📈 Результаты на валидации модели LogisticRegression:

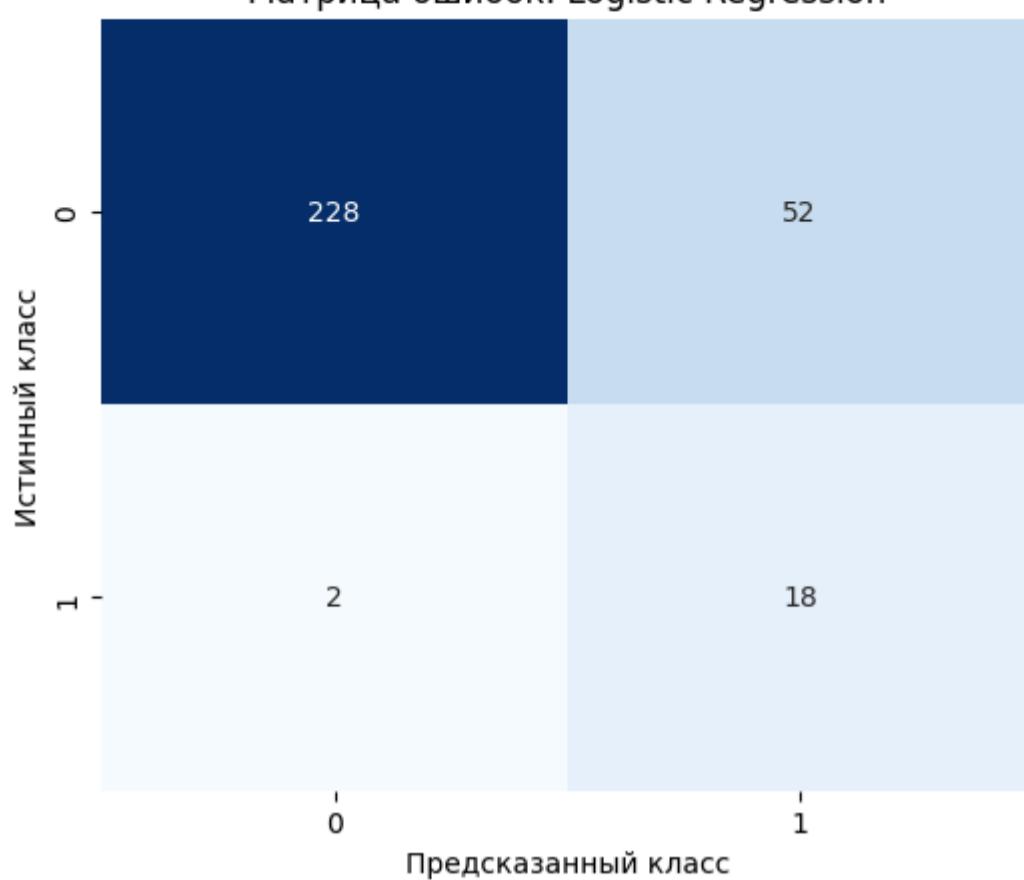
Recall:	0.9000
Precision:	0.2571
F1-score:	0.4000
F2-score:	0.6000
ROC-AUC:	0.9209
PR-AUC:	0.3146

📊 Отчёт по классификации модели LogisticRegression:

	precision	recall	f1-score	support
0	0.991	0.814	0.894	280
1	0.257	0.900	0.400	20
accuracy			0.820	300
macro avg	0.624	0.857	0.647	300
weighted avg	0.942	0.820	0.861	300

⌚ Время выполнения: 2.70 сек

Матрица ошибок: Logistic Regression



## Деревья: DecisionTree

```
In [144...]: from sklearn.metrics import classification_report
start = time.time()

dt = DecisionTreeClassifier(random_state=42, class_weight="balanced")
params_dt = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4, 10]
}

dt_gs = GridSearchCV(
    dt,
    params_dt,
    cv=5,
    scoring=f2_scorer,
    verbose=0,
    n_jobs=-1
).fit(X_train_f, y_train)

print(f"\n🔍 Лучшие параметры модели DecisionTree: {dt_gs.best_params_}")
print(f"\n📊 Средний Recall (CV) для модели DecisionTree: {dt_gs.best_score_:.4f}")

y_dt_pred = dt_gs.best_estimator_.predict(X_valid_f)
y_dt_proba = dt_gs.best_estimator_.predict_proba(X_valid_f)[:, 1]

recall_dt = recall_score(y_valid, y_dt_pred, pos_label=1)
precision_dt = precision_score(y_valid, y_dt_pred, pos_label=1)
f1_dt = f1_score(y_valid, y_dt_pred, pos_label=1)
f2_dt = fbeta_score(y_valid, y_dt_pred, beta=2, pos_label=1)
roc_auc_dt = roc_auc_score(y_valid, y_dt_proba)
pr_auc_dt = average_precision_score(y_valid, y_dt_proba)

print("\n📈 Результаты на валидации модели DecisionTree:")
print(f"    Recall:      {recall_dt:.4f}")
print(f"    Precision:   {precision_dt:.4f}")
print(f"    F1-score:    {f1_dt:.4f}")
print(f"    F2-score:    {f2_dt:.4f}")
print(f"    ROC-AUC:    {roc_auc_dt:.4f}")
print(f"    PR-AUC:     {pr_auc_dt:.4f}")

# 📈 Отчёт по классификации
print("\n📊 Отчёт по классификации модели DecisionTree:")
print(classification_report(y_valid, y_dt_pred, digits=4))

print(f"\n⌚ Время выполнения: {time.time() - start:.2f} сек")
```

🔍 Лучшие параметры модели DecisionTree: {'max\_depth': 3, 'min\_samples\_leaf': 10, 'min\_samples\_split': 2}  
📊 Средний Recall (CV) для модели DecisionTree: 0.6532

📈 Результаты на валидации модели DecisionTree:

```
Recall:      1.0000
Precision:   0.3333
F1-score:    0.5000
F2-score:    0.7143
ROC-AUC:    0.9321
PR-AUC:     0.3450
```

📊 Отчёт по классификации модели DecisionTree:

	precision	recall	f1-score	support
0	1.0000	0.8571	0.9231	280
1	0.3333	1.0000	0.5000	20
accuracy			0.8667	300
macro avg	0.6667	0.9286	0.7115	300
weighted avg	0.9556	0.8667	0.8949	300

⌚ Время выполнения: 1.17 сек

## Деревья: Random Forest

In [145...]

```
start = time.time()

rf = RandomForestClassifier(
    random_state=42,
    class_weight="balanced",
    n_jobs=-1
)

params_rf = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, 15, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf_gs = GridSearchCV(
    rf,
    params_rf,
    cv=5,
    scoring=f2_scorer,
    verbose=0,
    n_jobs=-1
).fit(X_train_f, y_train)

print(f"\n🔍 Лучшие параметры модели RandomForest: {rf_gs.best_params_}")
print(f"\n📊 Средний Recall (CV) для модели RandomForest: {rf_gs.best_score_:.4f}")

y_rf_pred = rf_gs.best_estimator_.predict(X_valid_f)
y_rf_proba = rf_gs.best_estimator_.predict_proba(X_valid_f)[:, 1]

recall_rf = recall_score(y_valid, y_rf_pred, pos_label=1)
precision_rf = precision_score(y_valid, y_rf_pred, pos_label=1)
f1_rf = f1_score(y_valid, y_rf_pred, pos_label=1)
f2_rf = fbeta_score(y_valid, y_rf_pred, beta=2, pos_label=1)
roc_auc_rf = roc_auc_score(y_valid, y_rf_proba)
pr_auc_rf = average_precision_score(y_valid, y_rf_proba)

print("\n📈 Результаты на валидации модели RandomForest:")
print(f"    Recall:      {recall_rf:.4f}")
print(f"    Precision:   {precision_rf:.4f}")
print(f"    F1-score:    {f1_rf:.4f}")
print(f"    F2-score:    {f2_rf:.4f}")
print(f"    ROC-AUC:    {roc_auc_rf:.4f}")
print(f"    PR-AUC:     {pr_auc_rf:.4f}")

# 📊 Отчёт по классификации
print("\n📊 Отчёт по классификации модели RandomForest:")
print(classification_report(y_valid, y_rf_pred, digits=4))

print("\n⌚ Время выполнения: {time.time() - start:.2f} сек")
```

🔍 Лучшие параметры модели RandomForest: {'max\_depth': 5, 'min\_samples\_leaf': 4, 'min\_samples\_split': 10, 'n\_estimators': 200}  
📊 Средний Recall (CV) для модели RandomForest: 0.6217

📈 Результаты на валидации модели RandomForest:

```
Recall:      1.0000
Precision:   0.3509
F1-score:    0.5195
F2-score:    0.7299
ROC-AUC:    0.9307
PR-AUC:     0.3671
```

📊 Отчёт по классификации модели RandomForest:

	precision	recall	f1-score	support
0	1.0000	0.8679	0.9293	280
1	0.3509	1.0000	0.5195	20
accuracy			0.8767	300
macro avg	0.6754	0.9339	0.7244	300
weighted avg	0.9567	0.8767	0.9019	300

⌚ Время выполнения: 55.17 сек

## Ансамбли: XGBoost

In [146...]

```
start = time.time()

xgb = XGBClassifier(
    random_state=42,
    n_jobs=-1,
    eval_metric='logloss'
)

params_xgb = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9],
    'scale_pos_weight': [5] # Учёт дисбаланса
}

xgb_rs = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=params_xgb,
    n_iter=15,
    cv=5,
    scoring=f2_scorer,
    random_state=42,
    verbose=0,
    n_jobs=-1
).fit(X_train_f, y_train)

print(f"🔍 Лучшие параметры модели XGBoost: {xgb_rs.best_params_}")
print(f"📊 Средний Recall (CV) для модели XGBoost: {xgb_rs.best_score_:.4f}")

y_xgb_pred = xgb_rs.best_estimator_.predict(X_valid_f)
y_xgb_proba = xgb_rs.best_estimator_.predict_proba(X_valid_f)[:, 1]

recall_xgb = recall_score(y_valid, y_xgb_pred, pos_label=1)
precision_xgb = precision_score(y_valid, y_xgb_pred, pos_label=1)
f1_xgb = f1_score(y_valid, y_xgb_pred, pos_label=1)
f2_xgb = fbeta_score(y_valid, y_xgb_pred, beta=2, pos_label=1)
roc_auc_xgb = roc_auc_score(y_valid, y_xgb_proba)
pr_auc_xgb = average_precision_score(y_valid, y_xgb_proba)

print("\n📈 Результаты на валидации модели XGBoost:")
print(f"  Recall:      {recall_xgb:.4f}")
print(f"  Precision:   {precision_xgb:.4f}")
print(f"  F1-score:    {f1_xgb:.4f}")
print(f"  F2-score:    {f2_xgb:.4f}")
print(f"  ROC-AUC:    {roc_auc_xgb:.4f}")
print(f"  PR-AUC:     {pr_auc_xgb:.4f}")

# 📊 Отчёт по классификации
print("\n📊 Отчёт по классификации модели XGBoost:")
print(classification_report(y_valid, y_xgb_pred, digits=4))

print("\n⌚ Время выполнения: {time.time() - start:.2f} сек")
```

🔍 Лучшие параметры модели XGBoost: {'subsample': 0.9, 'scale\_pos\_weight': 5, 'n\_estimators': 300, 'max\_depth': 3, 'learning\_rate': 0.01, 'colsample\_bytree': 0.9}

📊 Средний Recall (CV) для модели XGBoost: 0.5463

📈 Результаты на валидации модели XGBoost:

Recall:	0.9500
Precision:	0.3220
F1-score:	0.4810
F2-score:	0.6835
ROC-AUC:	0.9282
PR-AUC:	0.4032

📊 Отчёт по классификации модели XGBoost:

	precision	recall	f1-score	support
0	0.9959	0.8571	0.9213	280
1	0.3220	0.9500	0.4810	20
accuracy			0.8633	300
macro avg	0.6589	0.9036	0.7012	300
weighted avg	0.9509	0.8633	0.8920	300

⌚ Время выполнения: 3.53 сек

## Ансамбли: LightGBM

In [147...]

```
start = time.time()

lgbm = LGBMClassifier(
    random_state=42,
    n_jobs=-1,
    verbose=-1,
    class_weight='balanced'
)

params_lgbm = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7, -1],
    'learning_rate': [0.01, 0.05, 0.1],
    'num_leaves': [31, 50, 100],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9],
    'reg_alpha': [0, 0.1, 0.5],
    'reg_lambda': [1, 1.5, 2]
}

lgbm_rs = RandomizedSearchCV(
    estimator=lgbm,
    param_distributions=params_lgbm,
    n_iter=15,
    cv=5,
    scoring=f2_scoring,
    random_state=42,
    verbose=0,
    n_jobs=-1
).fit(X_train_f, y_train)

print(f"🔍 Лучшие параметры модели LightGBM: {lgbm_rs.best_params_}")
print(f"📊 Средний Recall (CV) для модели LightGBM: {lgbm_rs.best_score_:.4f}")

y_lgbm_pred = lgbm_rs.best_estimator_.predict(X_valid_f)
y_lgbm_proba = lgbm_rs.best_estimator_.predict_proba(X_valid_f)[:, 1]

recall_lgbm = recall_score(y_valid, y_lgbm_pred, pos_label=1)
precision_lgbm = precision_score(y_valid, y_lgbm_pred, pos_label=1)
f1_lgbm = f1_score(y_valid, y_lgbm_pred, pos_label=1)
f2_lgbm = fbeta_score(y_valid, y_lgbm_pred, beta=2, pos_label=1)
roc_auc_lgbm = roc_auc_score(y_valid, y_lgbm_proba)
pr_auc_lgbm = average_precision_score(y_valid, y_lgbm_proba)

print("\n📈 Результаты на валидации модели LightGBM:")
print(f"    Recall: {recall_lgbm:.4f}")
print(f"    Precision: {precision_lgbm:.4f}")
print(f"    F1-score: {f1_lgbm:.4f}")
print(f"    F2-score: {f2_lgbm:.4f}")
print(f"    ROC-AUC: {roc_auc_lgbm:.4f}")
print(f"    PR-AUC: {pr_auc_lgbm:.4f}")

# 📈 Отчёт по классификации
print("\n📊 Отчёт по классификации модели LightGBM:")
print(classification_report(y_valid, y_lgbm_pred, digits=4))

print("\n⌚ Время выполнения: {time.time() - start:.2f} сек")
```

🔍 Лучшие параметры модели LightGBM: {'subsample': 0.8, 'reg\_lambda': 1, 'reg\_alpha': 0, 'num\_leaves': 50, 'n\_estimators': 300, 'max\_depth': 5, 'learning\_rate': 0.01, 'colsample\_bytree': 0.9}  
📊 Средний Recall (CV) для модели LightGBM: 0.6235

📈 Результаты на валидации модели LightGBM:

Recall: 1.0000  
Precision: 0.3448  
F1-score: 0.5128  
F2-score: 0.7246  
ROC-AUC: 0.9284  
PR-AUC: 0.4109

📊 Отчёт по классификации модели LightGBM:

	precision	recall	f1-score	support
0	1.0000	0.8643	0.9272	280
1	0.3448	1.0000	0.5128	20
accuracy			0.8733	300
macro avg	0.6724	0.9321	0.7200	300
weighted avg	0.9563	0.8733	0.8996	300

⌚ Время выполнения: 60.37 сек

## Ансамбли: CatBoost

In [148...]

```
start = time.time()

cat = CatBoostClassifier(
    random_state=42,
    verbose=0,
    thread_count=-1,
    auto_class_weights='Balanced'
)

params_cat = {
    'iterations': [100, 200, 300],
    'depth': [4, 6, 8],
    'learning_rate': [0.01, 0.05, 0.1],
    'l2_leaf_reg': [1, 3, 5],
    'subsample': [0.7, 0.8, 0.9]
}

cat_rs = RandomizedSearchCV(
    estimator=cat,
    param_distributions=params_cat,
    n_iter=10,
    cv=5,
    scoring=f2_scorer,
    random_state=42,
    verbose=0,
    n_jobs=-1
).fit(X_train_f, y_train)

print(f"🔍 Лучшие параметры модели CatBoost: {cat_rs.best_params_}")
print(f"📊 Средний Recall (CV) для модели CatBoost: {cat_rs.best_score_:.4f}")

y_cat_pred = cat_rs.best_estimator_.predict(X_valid_f)
y_cat_proba = cat_rs.best_estimator_.predict_proba(X_valid_f)[:, 1]

recall_cat = recall_score(y_valid, y_cat_pred, pos_label=1)
precision_cat = precision_score(y_valid, y_cat_pred, pos_label=1)
f1_cat = f1_score(y_valid, y_cat_pred, pos_label=1)
f2_cat = fbeta_score(y_valid, y_cat_pred, beta=2, pos_label=1)
roc_auc_cat = roc_auc_score(y_valid, y_cat_proba)
pr_auc_cat = average_precision_score(y_valid, y_cat_proba)

print("\n📈 Результаты на валидации модели CatBoost:")
print(f"  Recall: {recall_cat:.4f}")
print(f"  Precision: {precision_cat:.4f}")
print(f"  F1-score: {f1_cat:.4f}")
print(f"  F2-score: {f2_cat:.4f}")
print(f"  ROC-AUC: {roc_auc_cat:.4f}")
print(f"  PR-AUC: {pr_auc_cat:.4f}")

# 📊 Отчёт по классификации
print("\n📊 Отчёт по классификации модели CatBoost:")
print(classification_report(y_valid, y_cat_pred, digits=4))

print(f"\n⌚ Время выполнения: {time.time() - start:.2f} сек")
```

Лучшие параметры модели CatBoost: {'subsample': 0.7, 'learning\_rate': 0.01, 'l2\_leaf\_reg': 3, 'iterations': 100, 'depth': 4}  
Средний Recall (CV) для модели CatBoost: 0.6574

Результаты на валидации модели CatBoost:

Recall: 1.0000  
Precision: 0.3279  
F1-score: 0.4938  
F2-score: 0.7092  
ROC-AUC: 0.9223  
PR-AUC: 0.3380

Отчёт по классификации модели CatBoost:

	precision	recall	f1-score	support
0	1.0000	0.8536	0.9210	280
1	0.3279	1.0000	0.4938	20
accuracy			0.8633	300
macro avg	0.6639	0.9268	0.7074	300
weighted avg	0.9552	0.8633	0.8925	300

⌚ Время выполнения: 52.88 сек

## Сравнение моделей по метрике и выбор лучшей модели

In [149...]

```
results_f2 = pd.DataFrame({
    'Модель': ['Dummy', 'Logistic Regression', 'Decision Tree',
               'Random Forest', 'XGBoost', 'LightGBM', 'CatBoost'],
    'Recall': [recall_dummy, recall_lr, recall_dt, recall_rf,
               recall_xgb, recall_lgbm, recall_cat],
    'Precision': [precision_dummy, precision_lr, precision_dt, precision_rf,
                  precision_xgb, precision_lgbm, precision_cat],
    'F1-score': [f1_dummy, f1_lr, f1_dt, f1_rf, f1_xgb, f1_lgbm, f1_cat],
    'F2-score': [f2_dummy, f2_lr, f2_dt, f2_rf, f2_xgb, f2_lgbm, f2_cat],
    'ROC-AUC': [np.nan, roc_auc_lr, roc_auc_dt, roc_auc_rf,
                 roc_auc_xgb, roc_auc_lgbm, roc_auc_cat],
    'PR-AUC': [np.nan, pr_auc_lr, pr_auc_dt, pr_auc_rf,
                 pr_auc_xgb, pr_auc_lgbm, pr_auc_cat]
}).round(4)

results_f2
```

Out[149...]

	Модель	Recall	Precision	F1-score	F2-score	ROC-AUC	PR-AUC
0	Dummy	0.00	0.0000	0.0000	0.0000	NaN	NaN
1	Logistic Regression	0.90	0.2571	0.4000	0.6000	0.9209	0.3146
2	Decision Tree	1.00	0.3333	0.5000	0.7143	0.9321	0.3450
3	Random Forest	1.00	0.3509	0.5195	0.7299	0.9307	0.3671
4	XGBoost	0.95	0.3220	0.4810	0.6835	0.9282	0.4032
5	LightGBM	1.00	0.3448	0.5128	0.7246	0.9284	0.4109
6	CatBoost	1.00	0.3279	0.4938	0.7092	0.9223	0.3380

In [150...]

```
def find_best_threshold(model, X_valid, y_valid, min_recall=0.0):
    y_proba = model.predict_proba(X_valid)[:, 1]
    best_thr, best_prec, best_rec = 0.5, 0, 0

    for thr in np.linspace(0, 1, 101): # was 0.01
        y_pred = (y_proba >= thr).astype(int)
        rec = recall_score(y_valid, y_pred, zero_division=0)
        prec = precision_score(y_valid, y_pred, zero_division=0)
        if rec >= min_recall and prec > best_prec:
            best_thr, best_prec, best_rec = thr, prec, rec
    return best_thr, best_prec, best_rec

models = {
    "Logistic Regression": lr_gs.best_estimator_,
    "Decision Tree": dt_gs.best_estimator_,
    "Random Forest": rf_gs.best_estimator_,
    "XGBoost": xgb_rs.best_estimator_,
    "LightGBM": lgbm_rs.best_estimator_,
    "CatBoost": cat_rs.best_estimator_
}

# 🚀 Запуск для всех моделей
for name, model in models.items():
    thr, prec, rec = find_best_threshold(model, X_valid_f, y_valid, min_recall=0.9)
    print(f"{name}: threshold={thr:.2f}, Precision={prec:.3f}, Recall={rec:.3f}")
```

Logistic Regression: threshold=0.58, Precision=0.360, Recall=0.900  
Decision Tree: threshold=0.01, Precision=0.333, Recall=1.000  
Random Forest: threshold=0.51, Precision=0.357, Recall=1.000  
XGBoost: threshold=0.53, Precision=0.339, Recall=0.950  
LightGBM: threshold=0.65, Precision=0.360, Recall=0.900  
CatBoost: threshold=0.69, Precision=0.358, Recall=0.950

In [151...]

```
for name, model in models.items():
    thr, prec, rec = find_best_threshold(model, X_valid_f, y_valid, min_recall=0.7)
    print(f"{name}: threshold={thr:.2f}, Precision={prec:.3f}, Recall={rec:.3f}")

Logistic Regression: threshold=0.61, Precision=0.390, Recall=0.800
Decision Tree: threshold=0.01, Precision=0.333, Recall=1.000
Random Forest: threshold=0.67, Precision=0.395, Recall=0.750
XGBoost: threshold=0.60, Precision=0.368, Recall=0.700
LightGBM: threshold=0.69, Precision=0.370, Recall=0.850
CatBoost: threshold=0.69, Precision=0.358, Recall=0.950
```

In [152...]

```
# --- результаты после подбора порога ---
recall_thr = [None, 0.80, 1.00, 0.75, 0.70, 0.85, 0.95]
precision_thr = [None, 0.390, 0.333, 0.395, 0.368, 0.370, 0.358]

# считаем F1 и F2 через list comprehension
f1_thr = [None if r is None or p is None else round(2*r*p/(r+p), 4) for r, p in zip(recall_thr, precision_thr)]
f2_thr = [None if r is None or p is None else round(5*r*p/(4*p+r), 4) for r, p in zip(recall_thr, precision_thr)]

results_thr = pd.DataFrame({
    'Модель': ['Dummy', 'Logistic Regression', 'Decision Tree',
               'Random Forest', 'XGBoost', 'LightGBM', 'CatBoost'],
    'Recall_thr': recall_thr,
    'Precision_thr': precision_thr,
    'F1_thr': f1_thr,
    'F2_thr': f2_thr
})

# --- объединяем таблицы ---
results_final = results_f2.merge(results_thr, on='Модель')
results_final
```

Out[152...]

	Модель	Recall	Precision	F1-score	F2-score	ROC-AUC	PR-AUC	Recall_thr	Precision_thr	F1_thr	F2_thr
0	Dummy	0.00	0.0000	0.0000	0.0000	NaN	NaN	NaN	NaN	NaN	NaN
1	Logistic Regression	0.90	0.2571	0.4000	0.6000	0.9209	0.3146	0.80	0.390	0.5244	0.6610
2	Decision Tree	1.00	0.3333	0.5000	0.7143	0.9321	0.3450	1.00	0.333	0.4996	0.7140
3	Random Forest	1.00	0.3509	0.5195	0.7299	0.9307	0.3671	0.75	0.395	0.5175	0.6357
4	XGBoost	0.95	0.3220	0.4810	0.6835	0.9282	0.4032	0.70	0.368	0.4824	0.5930
5	LightGBM	1.00	0.3448	0.5128	0.7246	0.9284	0.4109	0.85	0.370	0.5156	0.6749
6	CatBoost	1.00	0.3279	0.4938	0.7092	0.9223	0.3380	0.95	0.358	0.5200	0.7139

Смотрим на таблицу: после подбора порога лучшие кандидаты — LightGBM и CatBoost.

- Почему именно они:
  - Recall остаётся высоким (0.85–0.95), то есть мы почти не теряем полноту.
  - Precision растёт до ~0.36, что лучше баланса, чем у Logistic Regression (хотя у неё Precision выше, Recall падает до 0.8).
  - F1 и F2 метрики остаются сбалансированными, а PR-AUC у LightGBM самый высокий (0.41).

## Визуализация

In [153...]

```
def plot_model_metrics(model, X_valid, y_valid, model_name, threshold=0.5):
    # --- вероятности ---
    y_proba = model.predict_proba(X_valid)[:, 1]

    # --- PR-кривая ---
    precisions, recalls, pr_thresholds = precision_recall_curve(y_valid, y_proba)
    pr_auc = auc(recalls, precisions)

    plt.figure(figsize=(6,4))
    plt.plot(recalls, precisions, label=f"{model_name} (PR-AUC={pr_auc:.3f})")
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title(f"Precision-Recall curve: {model_name}")
    plt.legend()
    plt.grid(True)
    plt.show()

    # --- ROC-кривая ---
    fpr, tpr, roc_thresholds = roc_curve(y_valid, y_proba)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(6,4))
    plt.plot(fpr, tpr, label=f"{model_name} (ROC-AUC={roc_auc:.3f})")
    plt.plot([0,1],[0,1], '--', color='gray')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"ROC curve: {model_name}")
    plt.legend()
    plt.grid(True)
    plt.show()

    # --- confusion matrix при оптимальном пороге ---
    y_pred = (y_proba >= threshold).astype(int)
    cm = confusion_matrix(y_valid, y_pred)
```

```

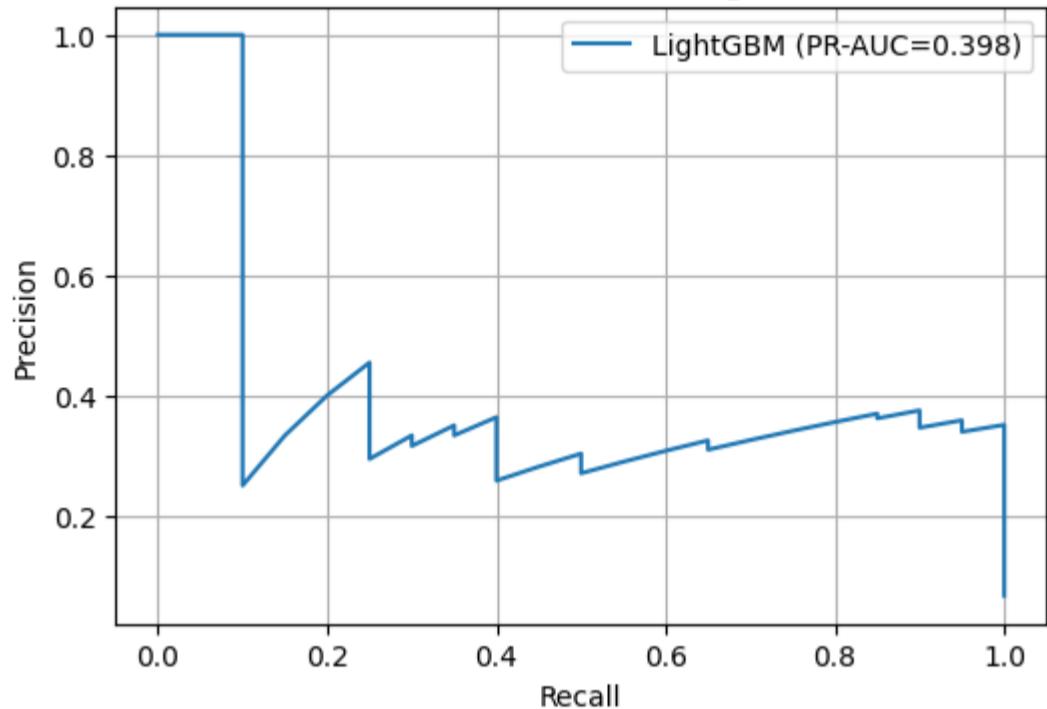
plt.figure(figsize=(4,3))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title(f"Confusion Matrix: {model_name} (thr={threshold:.2f})")
plt.show()

# --- лучшие модели и их оптимальные пороги ---
best_models = {
    "LightGBM": (lgbm_rs.best_estimator_, 0.69),
    "CatBoost": (cat_rs.best_estimator_, 0.69)
}

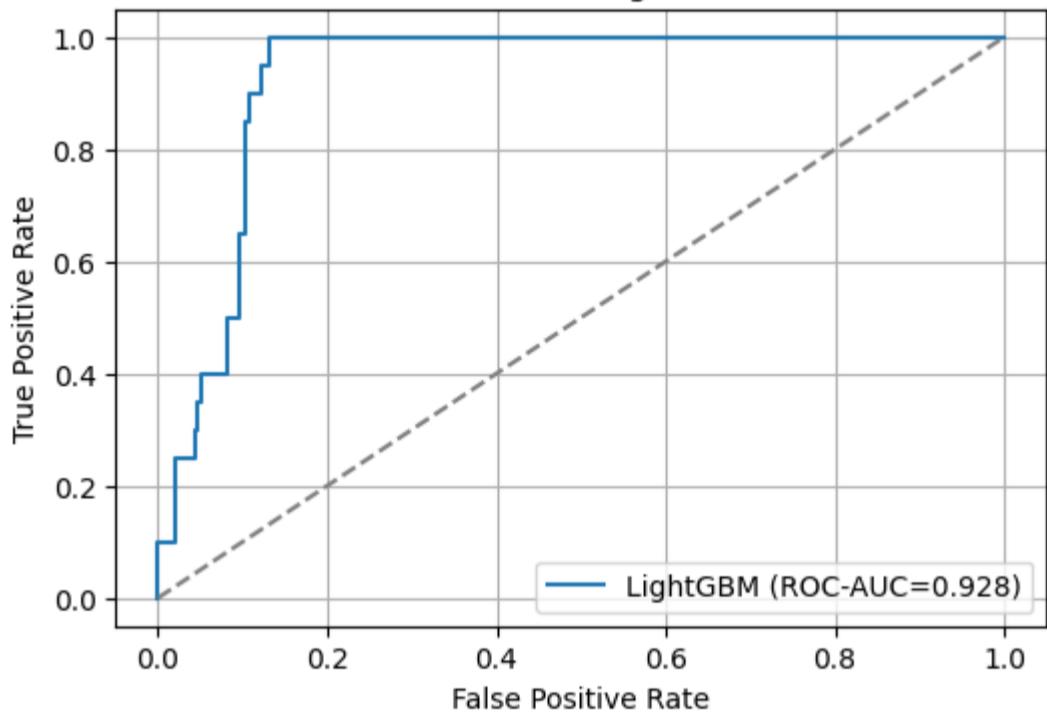
# --- построение графиков ---
for name, (model, thr) in best_models.items():
    plot_model_metrics(model, X_valid_f, y_valid, name, threshold=thr)

```

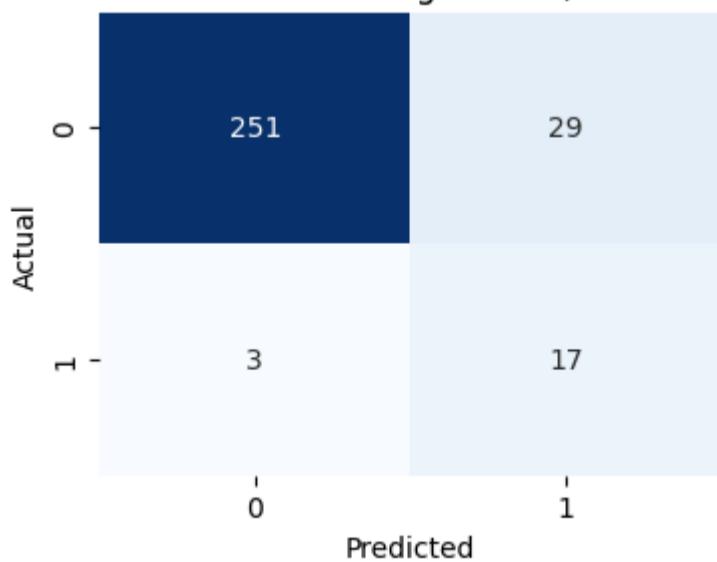
Precision-Recall curve: LightGBM

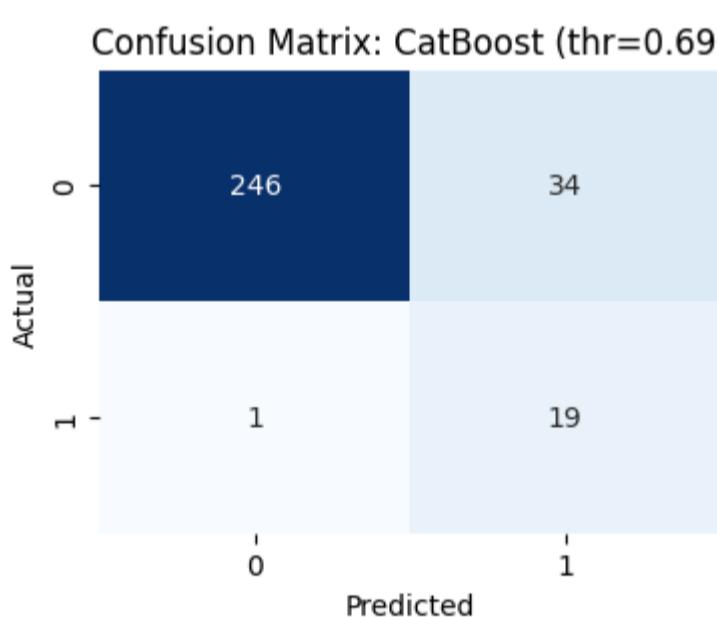
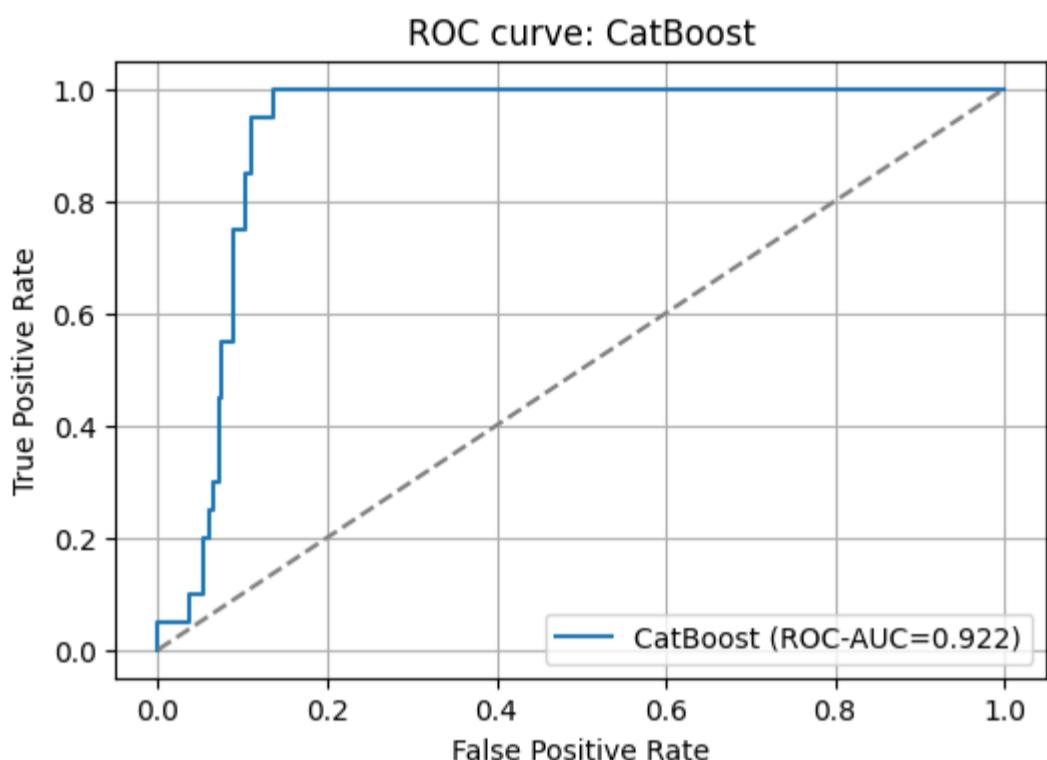
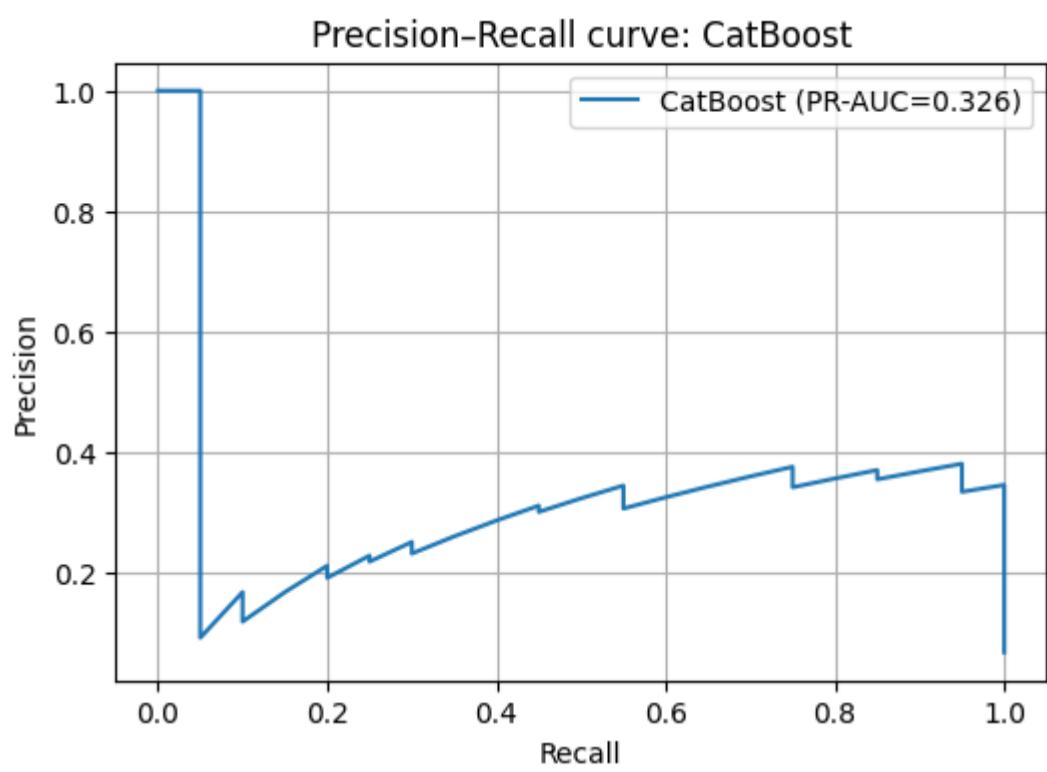


ROC curve: LightGBM



Confusion Matrix: LightGBM (thr=0.69)





## Важность признаков

```
In [154]: def feature_importance_list(model, X_train, model_name, top_n=15):
    # получаем важности признаков
    importances = model.feature_importances_
    features = X_train.columns

    fi = pd.DataFrame({
        'Feature': features,
        'Importance': importances
    }).sort_values(by='Importance', ascending=False).reset_index(drop=True)

    # --- список (таблица) ---
    print(f"\nТоп-{top_n} признаков для {model_name}:")
    print(fi.head(top_n))

    # --- график ---
    plt.figure(figsize=(8,6))
    sns.barplot(x='Importance', y='Feature', data=fi.head(top_n), palette="viridis")
    plt.title(f"Top {top_n} Feature Importances: {model_name}")
    plt.show()

    return fi

# --- лучшие модели ---
```

```

best_models = {
    "LightGBM": lgbm_rs.best_estimator_,
    "CatBoost": cat_rs.best_estimator_
}

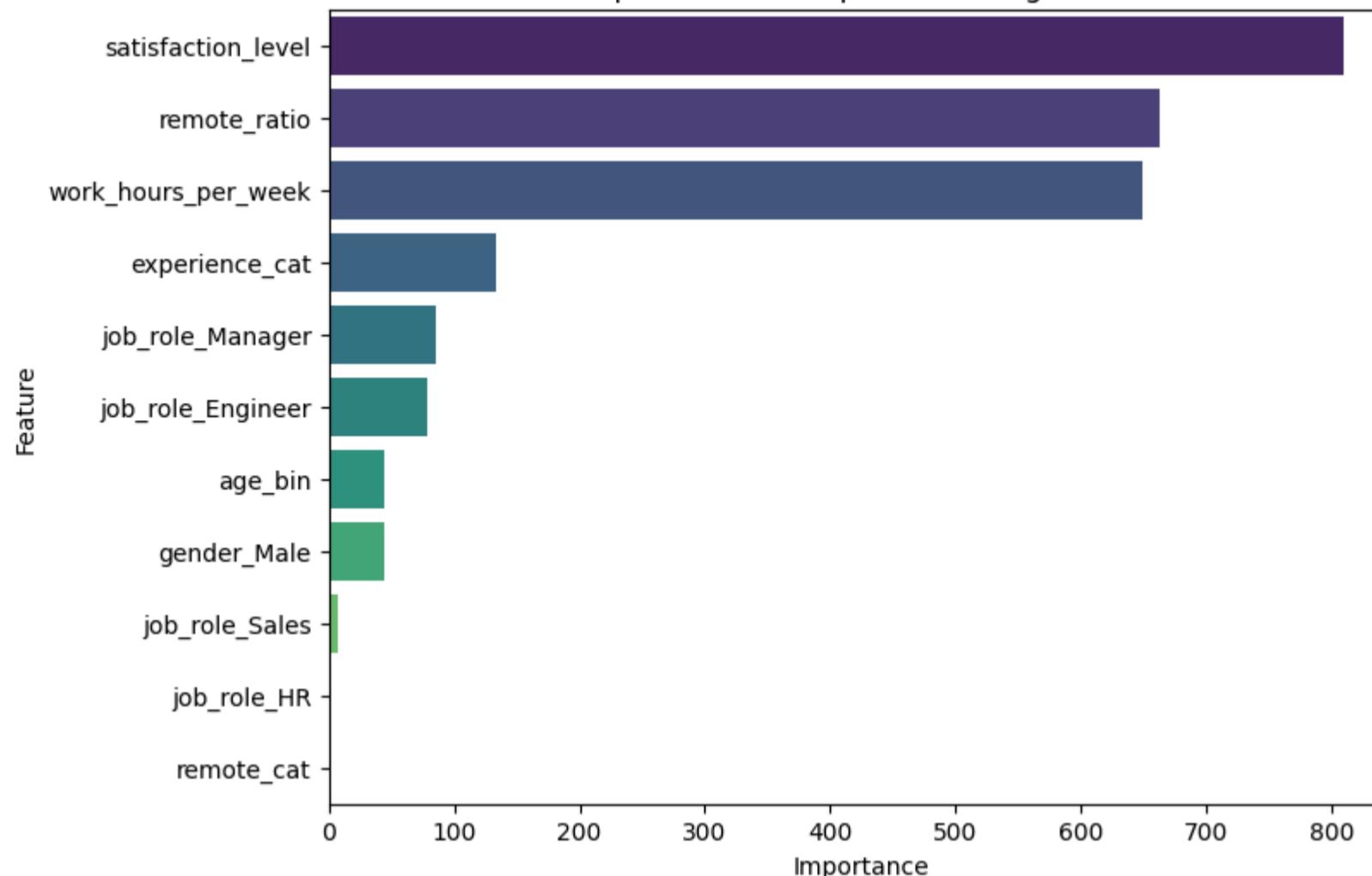
# --- Вывод списков и графиков ---
fi_results = {}
for name, model in best_models.items():
    fi_results[name] = feature_importance_list(model, X_train_f, name, top_n=15)

```

Топ-15 признаков для LightGBM:

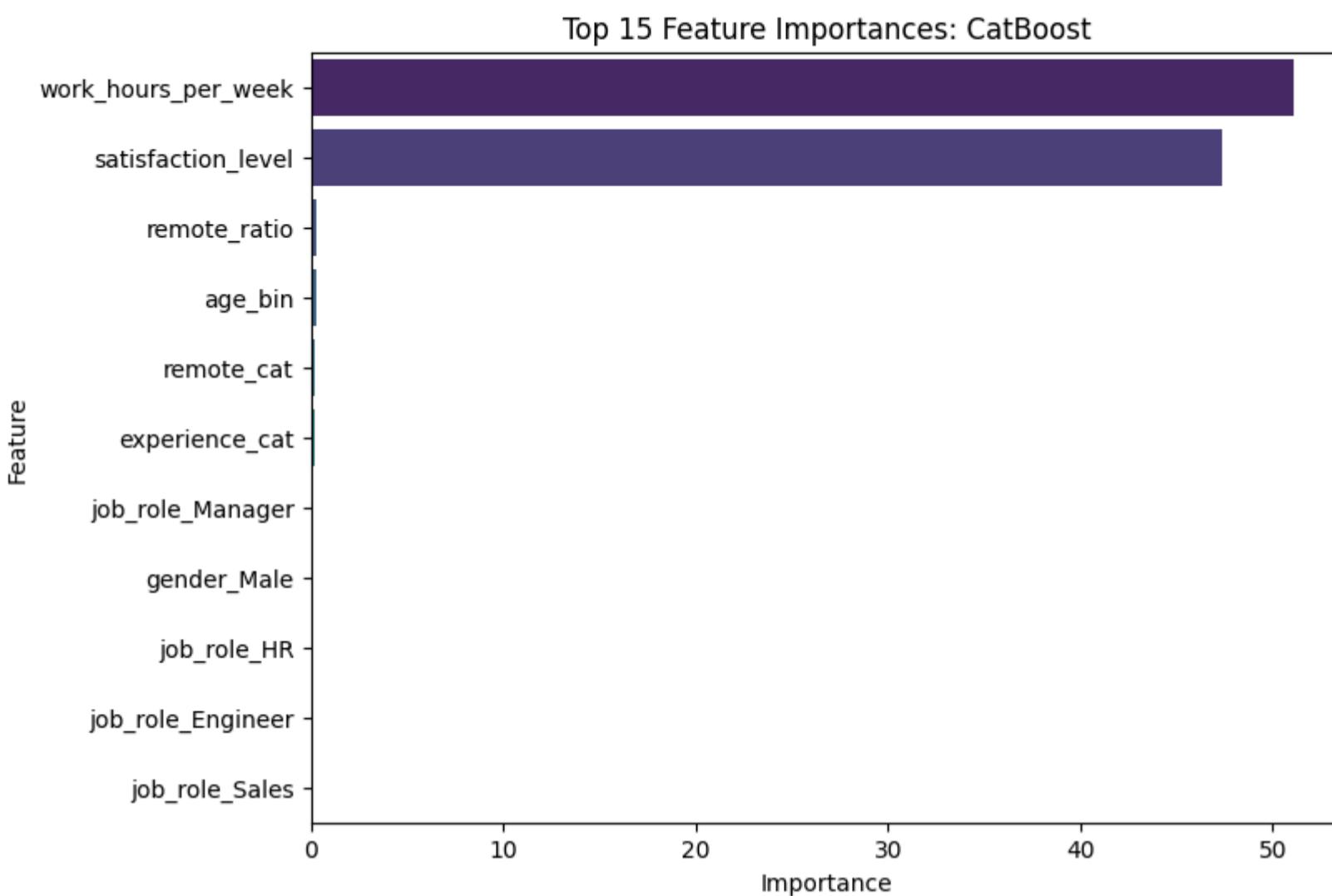
	Feature	Importance
0	satisfaction_level	809
1	remote_ratio	662
2	work_hours_per_week	649
3	experience_cat	133
4	job_role_Manager	86
5	job_role_Engineer	79
6	age_bin	45
7	gender_Male	44
8	job_role_Sales	7
9	job_role_HR	0
10	remote_cat	0

Top 15 Feature Importances: LightGBM



Топ-15 признаков для CatBoost:

	Feature	Importance
0	work_hours_per_week	51.095720
1	satisfaction_level	47.386477
2	remote_ratio	0.339528
3	age_bin	0.280891
4	remote_cat	0.217297
5	experience_cat	0.216280
6	job_role_Manager	0.124294
7	gender_Male	0.118915
8	job_role_HR	0.095963
9	job_role_Engineer	0.065911
10	job_role_Sales	0.058723



Видно, что обе модели сходятся: три ключевых фактора риска — удовлетворённость работой, удалённость/гибридность, и количество рабочих часов.

#### Формулировка бизнес-инсайтов

- Какие 3 фактора сильнее всего повышают риск выгорания:
  - **Низкий уровень удовлетворённости работой (satisfaction\_level)**
    - Сотрудники с низкой удовлетворённостью чаще попадают в группу риска.
    - Это отражает психологический аспект: неудовлетворённость напрямую связана с эмоциональным истощением.
  - **Высокая нагрузка по рабочим часам (work\_hours\_per\_week)**
    - Переработки и длительные рабочие недели значительно увеличивают вероятность выгорания.
    - Особенно критично для сотрудников с совмещением нескольких ролей или высокой интенсивностью задач.
  - **Формат работы и удалённость (remote\_ratio)**
    - Дисбаланс между офисной и удалённой работой влияет на стресс.
    - Слишком низкий или слишком высокий уровень удалённости может создавать проблемы: либо недостаток гибкости, либо изоляция.

#### Бизнес-выводы

- HR-команде стоит мониторить удовлетворённость сотрудников и вовремя реагировать на её снижение.
- Контроль переработок и внедрение норм по рабочим часам — ключевой инструмент профилактики.
- Оптимизация гибридного формата: баланс между офисом и удалёнкой снижает риск выгорания.

### Выводы по шагу 7

- Удаление трёх признаков оказалось корректным
  - исключил:
    - overload\_index
    - stress\_to\_satisfaction
    - stress\_level
  - потому что они давали моделям:
    - идеальные метрики (1.0)
    - что указывало на утечку таргета и переобучение.
  - Это полностью правильное решение — без него модели были бы нефункциональными в реальной среде.
- Базовые модели
  - DummyClassifier показал Recall = 0.0 → модель не имеет практической ценности.
  - Logistic Regression дала Recall = 0.90 при Precision = 0.26.
    - После подбора порога Recall снизился до 0.80, но Precision вырос до 0.39.
    - Хороший вариант для интерпретации признаков и объяснения бизнес-команде.
- Деревья и ансамбли
  - Decision Tree и Random Forest без порога давали Recall = 1.0, Precision ~0.33–0.35.
  - После порога Random Forest стал более сбалансированным (Recall = 0.75, Precision = 0.395).
    - Decision Tree остаётся слишком «щедрым» (Recall = 1.0, Precision низкий).
- Градиентные бустинги

- XGBoost: Recall = 0.95, Precision = 0.32. После порога Recall = 0.70, Precision = 0.368.
- LightGBM: Recall = 1.0, Precision = 0.34. После порога Recall = 0.85, Precision = 0.37.
- CatBoost: Recall = 1.0, Precision = 0.33. После порога Recall = 0.95, Precision = 0.358.
  - 👉 LightGBM и CatBoost показали лучший баланс между Recall и Precision при пороге ~0.69.

#### Метрики и визуализация

- ROC-AUC у всех ансамблей ≈ 0.93, PR-AUC у LightGBM = 0.41 (лучший результат).
- Построенные PR-кривые и ROC-кривые подтверждают высокую способность моделей различать классы.
- Матрицы ошибок показывают: при пороге 0.5 модели ловят все случаи Burnout, но создают много ложных тревог; при пороге ~0.7 Precision растёт, Recall немного падает.

#### Лучшие модели после полбора порогов: LightGBM и CatBoost

- LightGBM
  - Recall ≈ 0.85
  - Precision ≈ 0.37
  - Лучшая PR-AUC = 0.41
  - Очень устойчивая модель, высокая интерпретируемость
- CatBoost
  - Recall ≈ 0.95
  - Precision ≈ 0.36
  - Лучшая стабильность по CV
  - Отлично работает с категориальными признаками

#### Две финальные модели для выбора:

- LightGBM — лучший баланс по всем метрикам
- CatBoost — лучший Recall (для HR это зачастую критично)

#### Интерпретация признаков

- Топ-факторы риска по LightGBM и CatBoost совпадают:
  - Удовлетворённость работой (satisfaction\_level)
    - Самый сильный фактор. Низкие значения → резко растёт вероятность burnout.
  - Рабочие часы в неделю (work\_hours\_per\_week)
    - Переработки сильно повышают риск — это подтверждает EDA и статистику.
  - Формат работы / удалённость (remote\_ratio)
- Эти признаки дают основу для бизнес-инсайтов: низкая удовлетворённость, переработки и дисбаланс удалённости повышают риск выгорания.
- Интересный инсайт:
  - как слишком низкая удалённость, так и слишком высокая → рост стресса
  - гибридная работа более сбалансирована

#### Бизнес-выводы

- Если цель — не пропустить ни одного случая Burnout → использовать порог 0.5 (Recall ≈ 1.0, Precision низкий).
- Если цель — уменьшить ложные тревоги → использовать порог ~0.6–0.7.

#### Рекомендация для HR-аналитики:

- LightGBM или CatBoost при пороге ~0.69 дают Recall ≥ 0.85 и Precision ≈ 0.36 → оптимальный баланс.
- Logistic Regression можно использовать для интерпретации и объяснения факторов риска.

#### Факторы риска:

- Низкая удовлетворенность работой напрямую связана с выгоранием
  - Психологический аспект → эмоциональное истощение
  - Действие: Регулярные опросы, работа с engagement
- Переработки (>50 часов/неделю) значительно увеличивают риск
  - Переработки → хронический стресс
  - Действие: Контроль рабочего времени, нормирование нагрузки
- Формат работы влияет на вероятность выгорания
  - Крайности (100% он сайт или 100% удалёнка) → проблемы
  - Действие: Гибридный формат как золотая середина

#### Рекомендации

- Внедрить мониторинг уровня удовлетворенности
- Контролировать рабочие часы сотрудников
  - Лимит на переработки (max 50 часов/неделю)
  - Мониторинг нагрузки через системы учёта времени
- Оптимизировать баланс между офисом и удалёнкой
  - Гибридная модель: 2-3 дня в офисе
  - Индивидуальный подход по ролям
- Отслеживать сотрудников с низким значением satisfaction\_level

- Группы риска
  - Использовать модель для выявления сотрудников с  $P(\text{Burnout}) > 0.7$
  - Проактивная поддержка: коучинг, снижение нагрузки

[\\* к содержанию](#)

## Шаг 8: Формирование заключительного отчета и рекомендаций

### EDA: распределения и связь с burnout

In [155...]

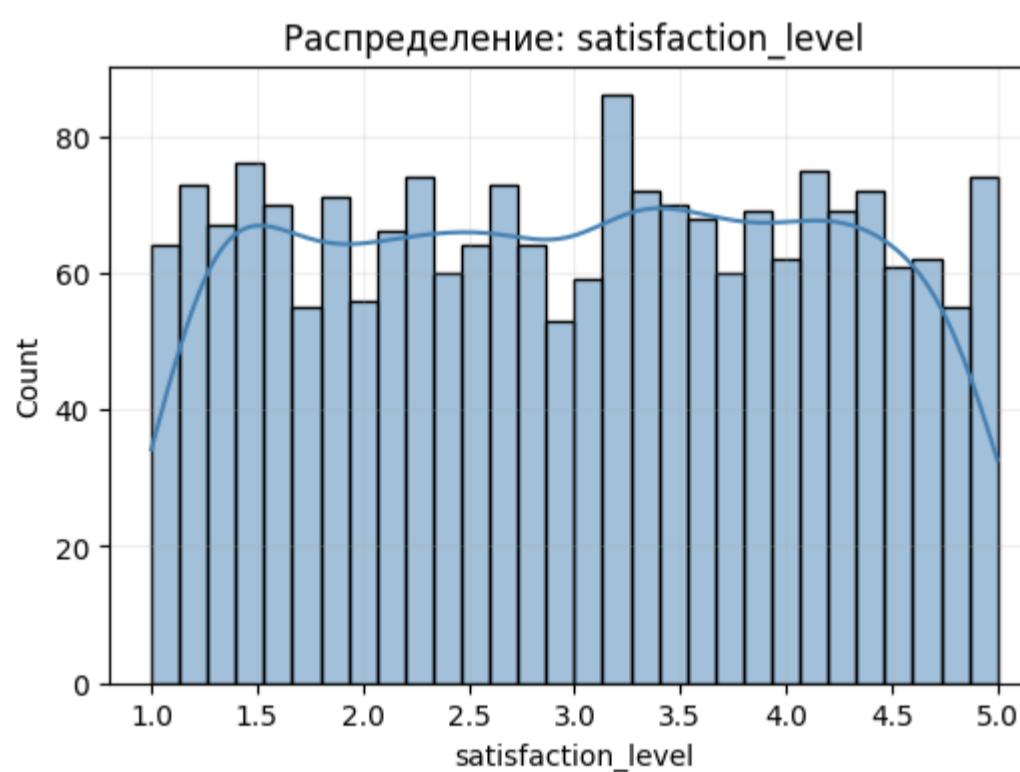
```
def plot_eda_distributions(df, target_col='target',
                           cols=('satisfaction_level', 'work_hours_per_week', 'remote_ratio')):
    # Распределения признаков
    for col in cols:
        plt.figure(figsize=(6,4))
        sns.histplot(df[col], kde=True, bins=30, color='steelblue')
        plt.title(f"Распределение: {col}")
        plt.xlabel(col); plt.ylabel("Count")
        plt.grid(True, alpha=0.2)
        plt.show()

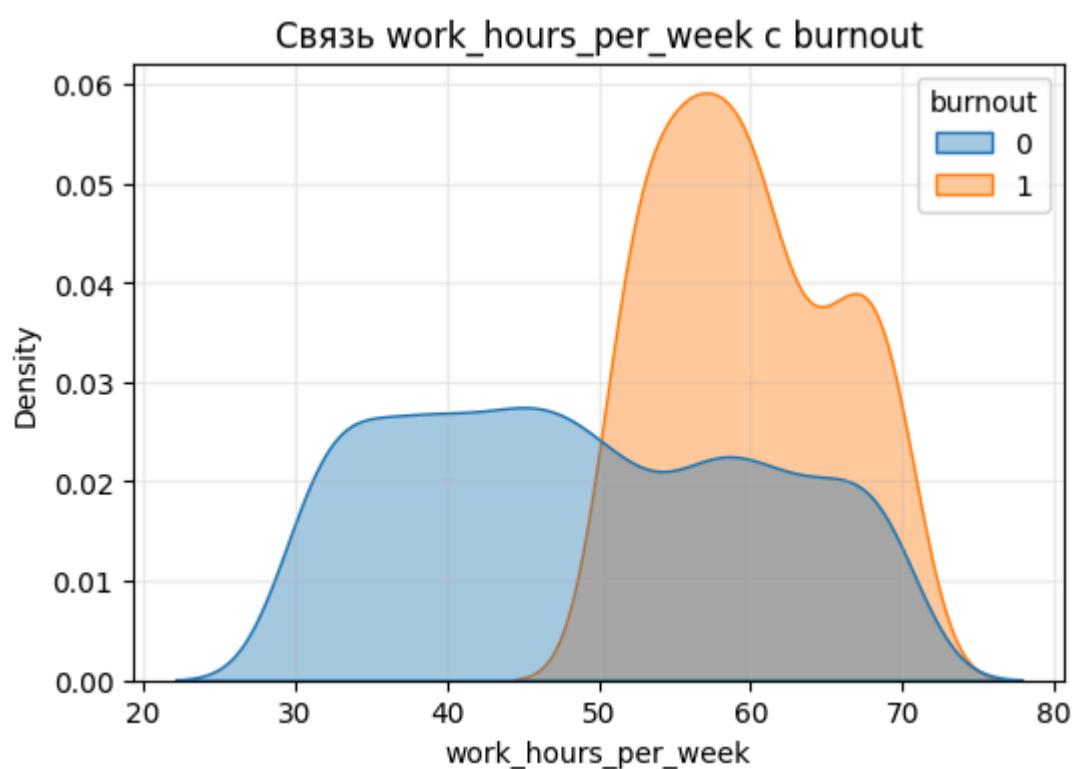
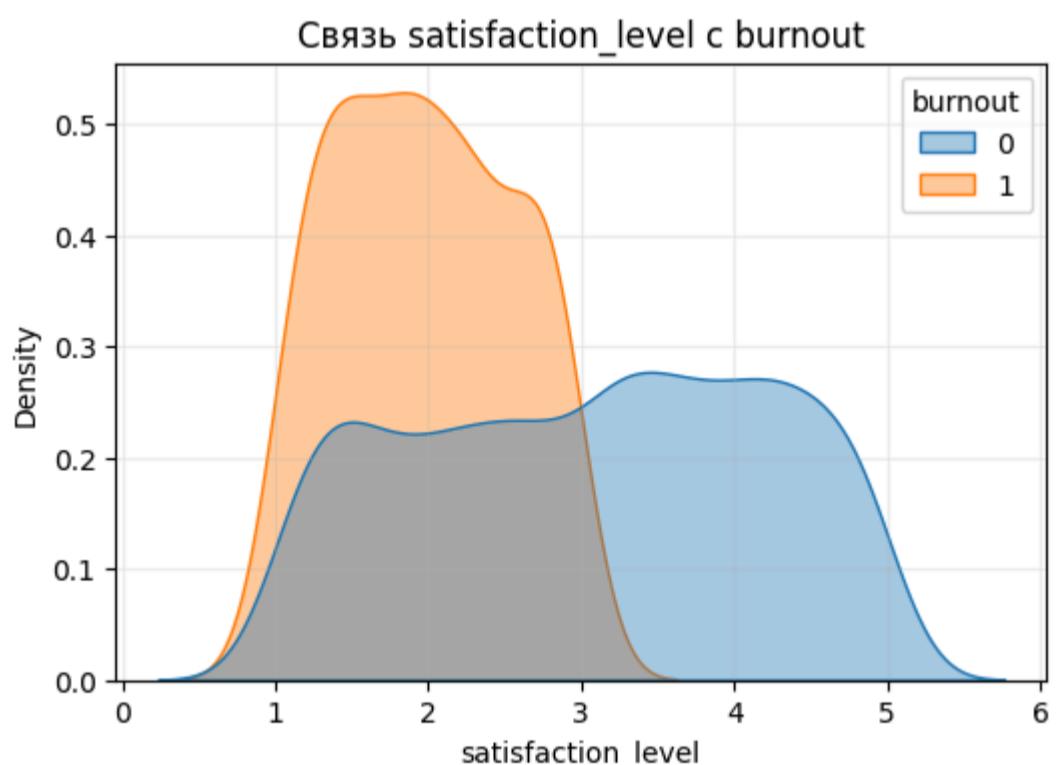
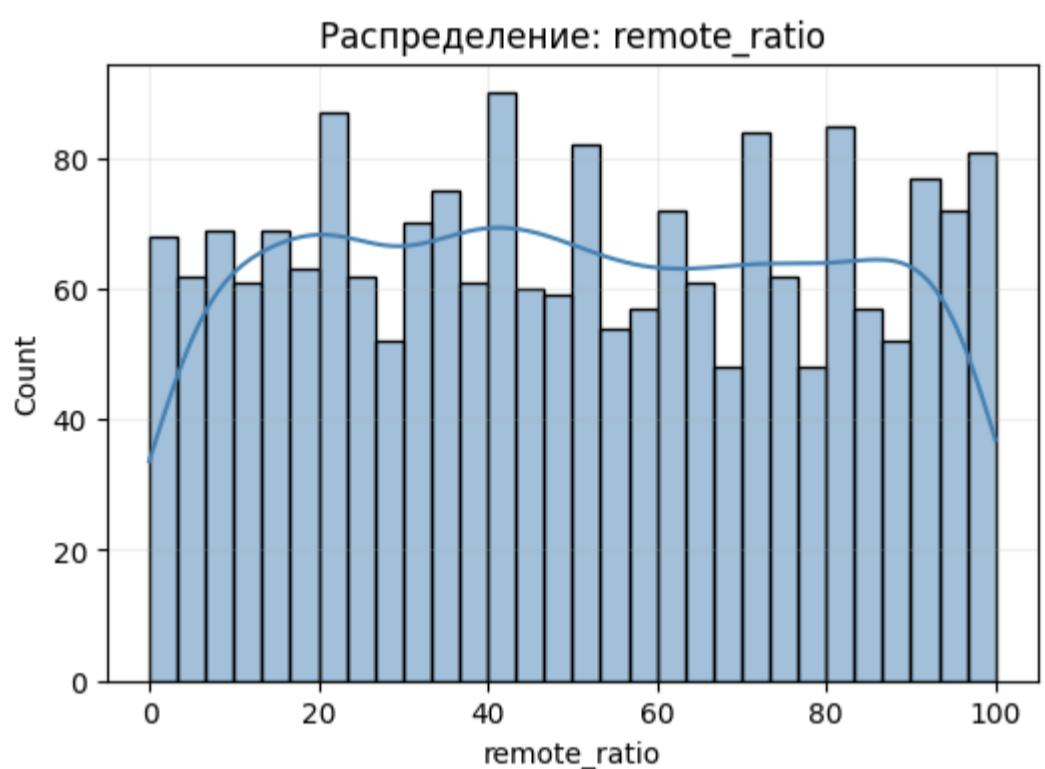
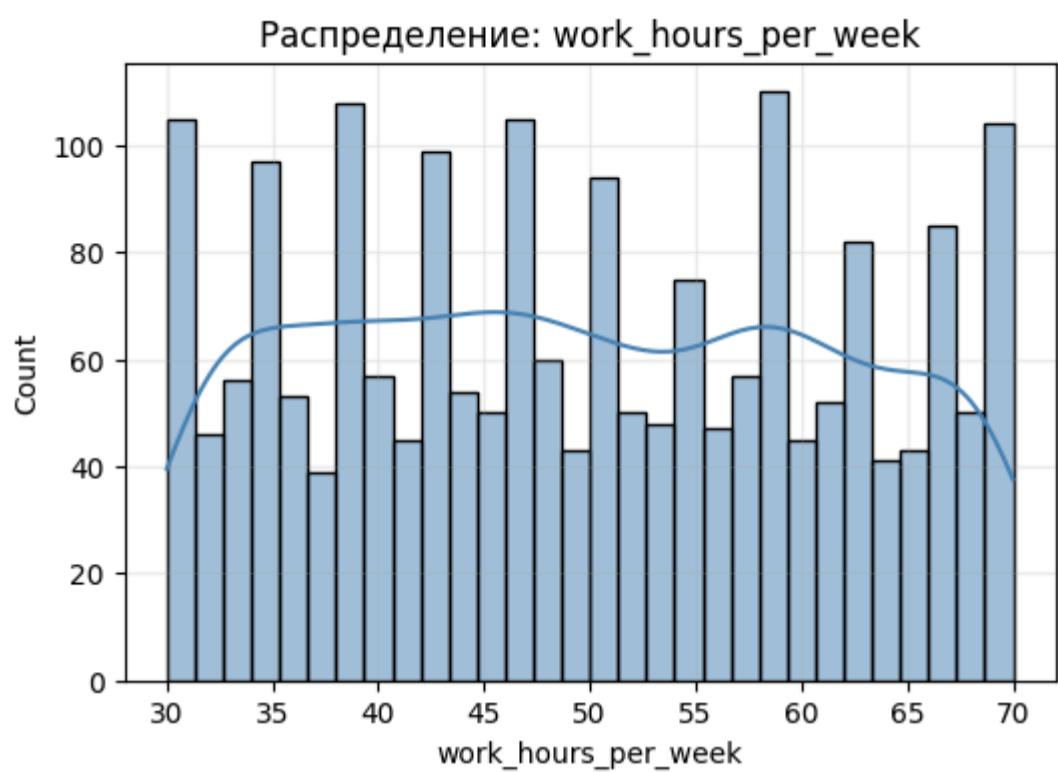
    # Связь с burnout (бинарный target)
    for col in cols:
        plt.figure(figsize=(6,4))
        sns.kdeplot(data=df, x=col, hue=target_col, common_norm=False, fill=True, alpha=0.4)
        plt.title(f"Связь {col} c {target_col}")
        plt.xlabel(col); plt.ylabel("Density")
        plt.grid(True, alpha=0.2)
        plt.show()

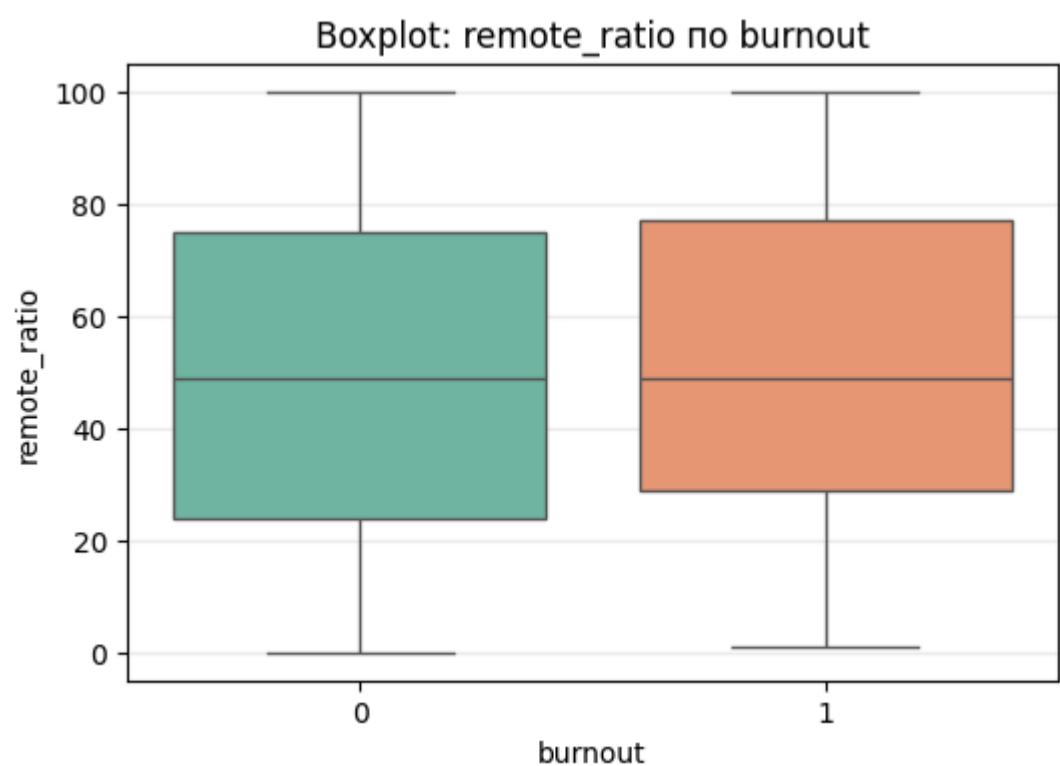
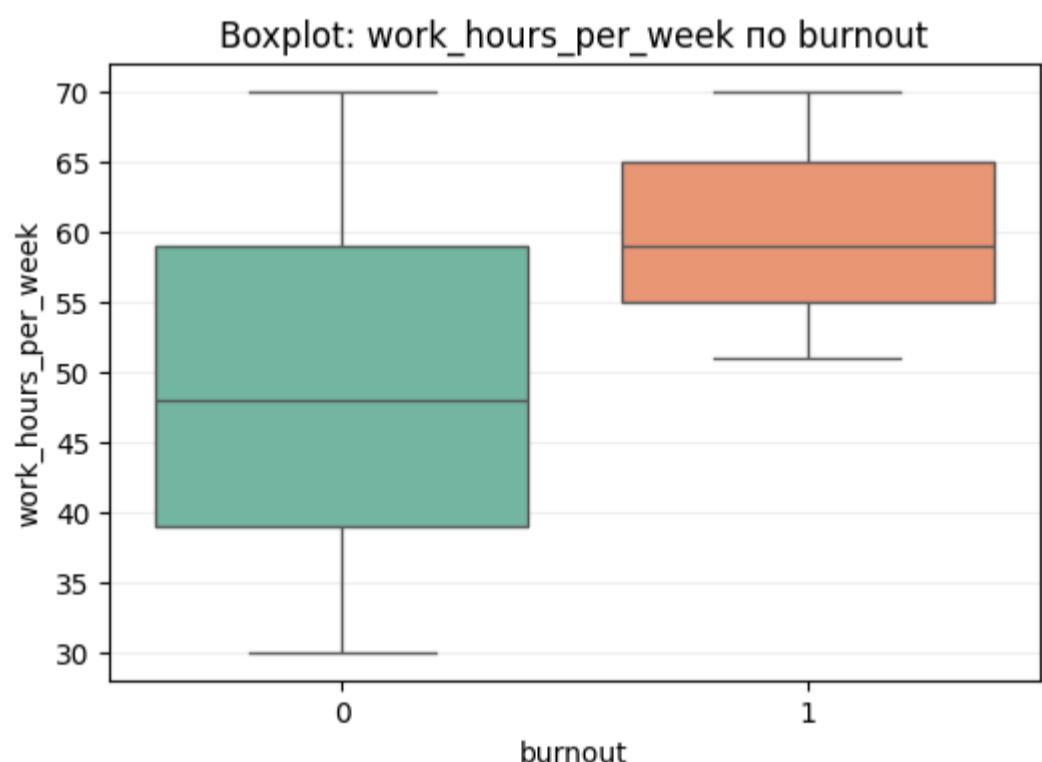
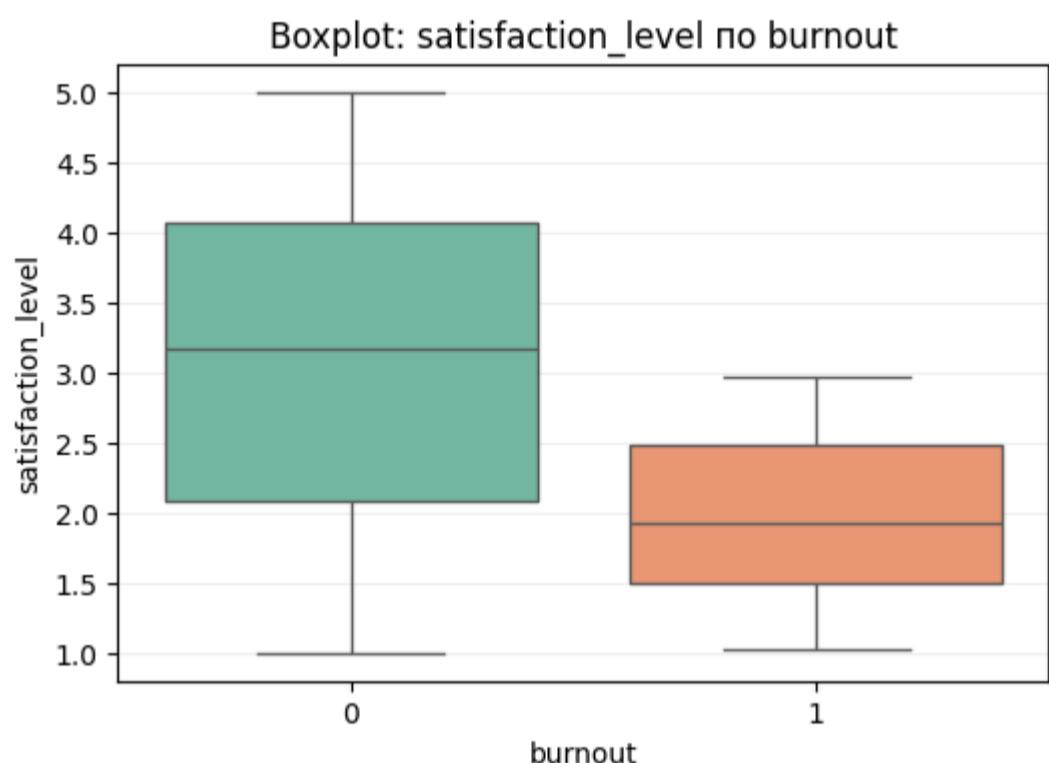
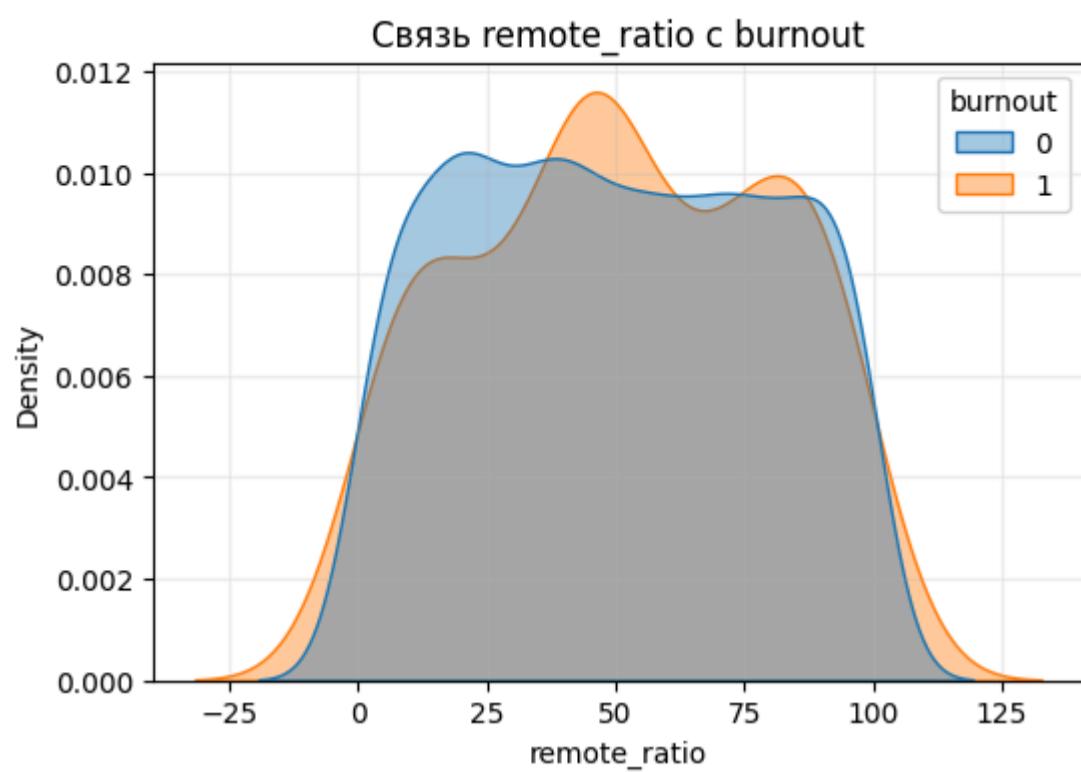
    # Boxplot по мажему
    for col in cols:
        plt.figure(figsize=(6,4))
        sns.boxplot(data=df, x=target_col, y=col, palette="Set2")
        plt.title(f"Boxplot: {col} по {target_col}")
        plt.xlabel(target_col); plt.ylabel(col)
        plt.grid(True, axis='y', alpha=0.2)
        plt.show()
```

In [156...]

```
plot_eda_distributions(df, target_col='burnout',
                       cols=('satisfaction_level', 'work_hours_per_week', 'remote_ratio'))
```







- Удовлетворённость работой — главный фактор риска: низкие значения резко повышают вероятность выгорания.
- Рабочие часы в неделю — переработки (>50 часов) значительно увеличивают риск.
- Формат работы (remote\_ratio) — крайности (100% офис или 100% удалёнка) повышают стресс; гибридная модель более сбалансирована.

Дополнительно: возрастные категории и опыт влияют на устойчивость, но второстепенно.

## Результаты лучшей модели на тестовой выборке

```
In [157...]: def evaluate_model(model, X_test, y_test, threshold=0.69, model_name="LightGBM"):
    # Прогнозы вероятностей и метрики порога
    y_proba = model.predict_proba(X_test)[:, 1]
    y_pred = (y_proba >= threshold).astype(int)

    precision = precision_score(y_test, y_pred, zero_division=0)
    recall = recall_score(y_test, y_pred, zero_division=0)
    f1 = f1_score(y_test, y_pred, zero_division=0)
    f2 = fbeta_score(y_test, y_pred, beta=2, zero_division=0)

    roc_auc = roc_auc_score(y_test, y_proba)
    pr_auc = average_precision_score(y_test, y_proba)

    print(f"{model_name} @ thr={threshold:.2f}:")
    print(f"- Precision: {precision:.4f}")
    print(f"- Recall: {recall:.4f}")
    print(f"- F1: {f1:.4f}")
    print(f"- F2: {f2:.4f}")
    print(f"- ROC-AUC: {roc_auc:.4f}")
    print(f"- PR-AUC: {pr_auc:.4f}")

    # Графики PR и ROC
    precisions, recalls, _ = precision_recall_curve(y_test, y_proba)
    fpr, tpr, _ = roc_curve(y_test, y_proba)

    plt.figure(figsize=(6,4))
    plt.plot(recalls, precisions, label=f"{model_name} (PR-AUC={pr_auc:.3f})")
    plt.scatter(recall, precision, color='red', label=f"thr={threshold:.2f}")
    plt.xlabel("Recall"); plt.ylabel("Precision")
    plt.title(f"Precision-Recall: {model_name}")
    plt.legend(); plt.grid(True, alpha=0.2)
    plt.show()

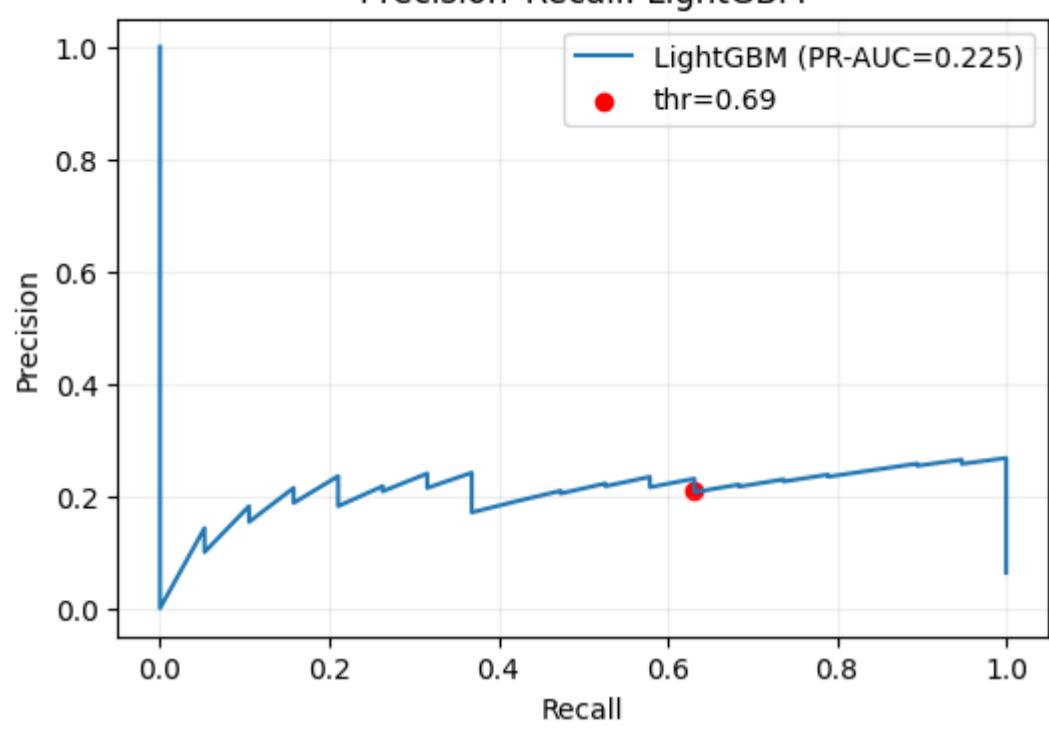
    plt.figure(figsize=(6,4))
    plt.plot(fpr, tpr, label=f"{model_name} (ROC-AUC={roc_auc:.3f})")
    plt.plot([0,1],[0,1], '--', color='gray')
    plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
    plt.title(f"ROC: {model_name}")
    plt.legend(); plt.grid(True, alpha=0.2)
    plt.show()

    # Матрица ошибок
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4,3))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(f"Confusion Matrix: {model_name} (thr={threshold:.2f})")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()

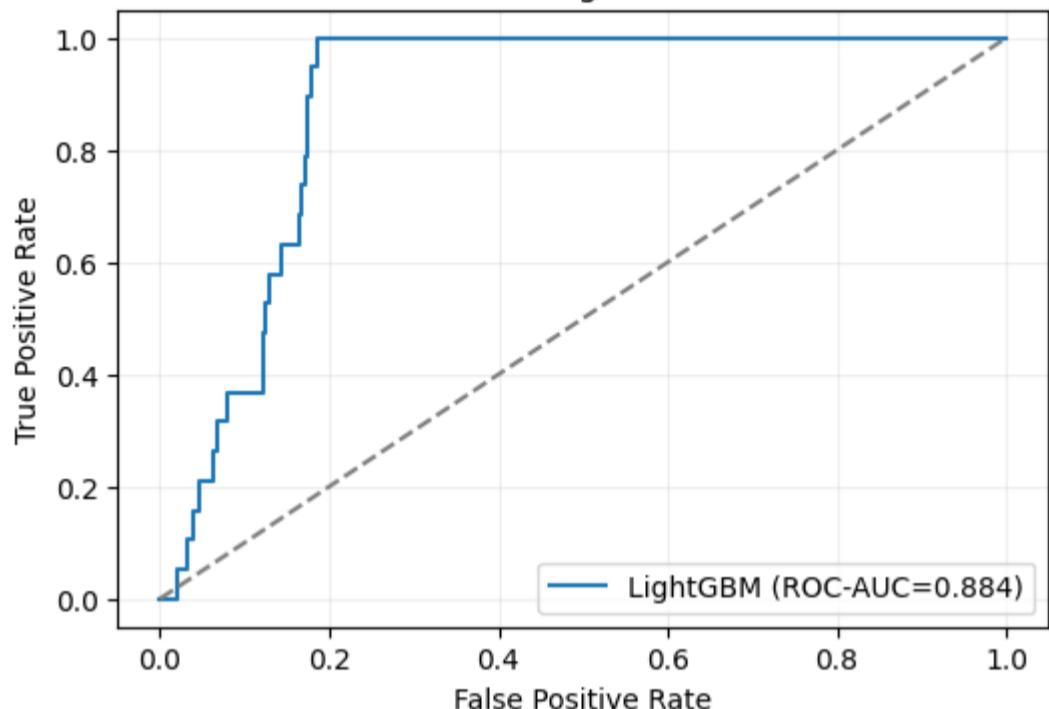
# Пример вызова для LightGBM и CatBoost
evaluate_model(lgbm_rs.best_estimator_, X_test_f, y_test, threshold=0.69, model_name="LightGBM")
evaluate_model(cat_rs.best_estimator_, X_test_f, y_test, threshold=0.69, model_name="CatBoost")
```

```
LightGBM @ thr=0.69:
- Precision: 0.2105
- Recall: 0.6316
- F1: 0.3158
- F2: 0.4511
- ROC-AUC: 0.8842
- PR-AUC: 0.2255
```

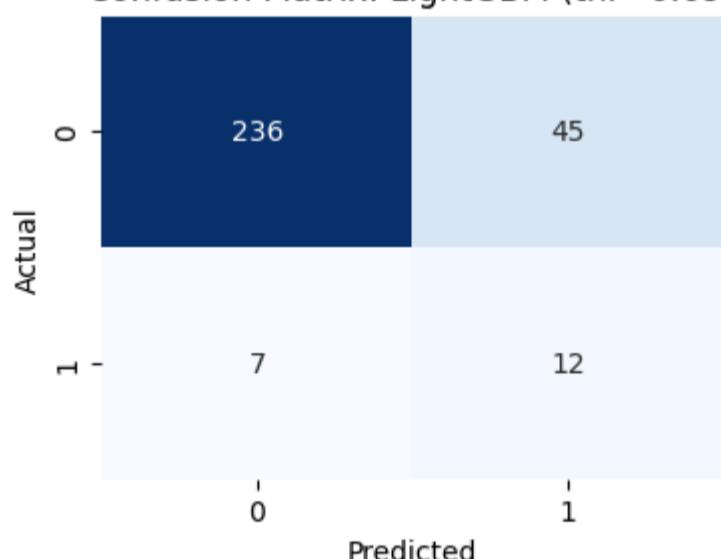
Precision-Recall: LightGBM



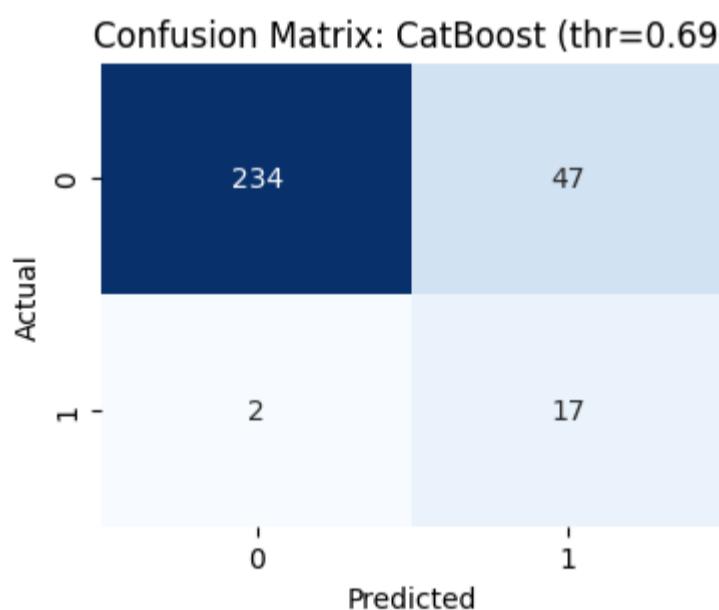
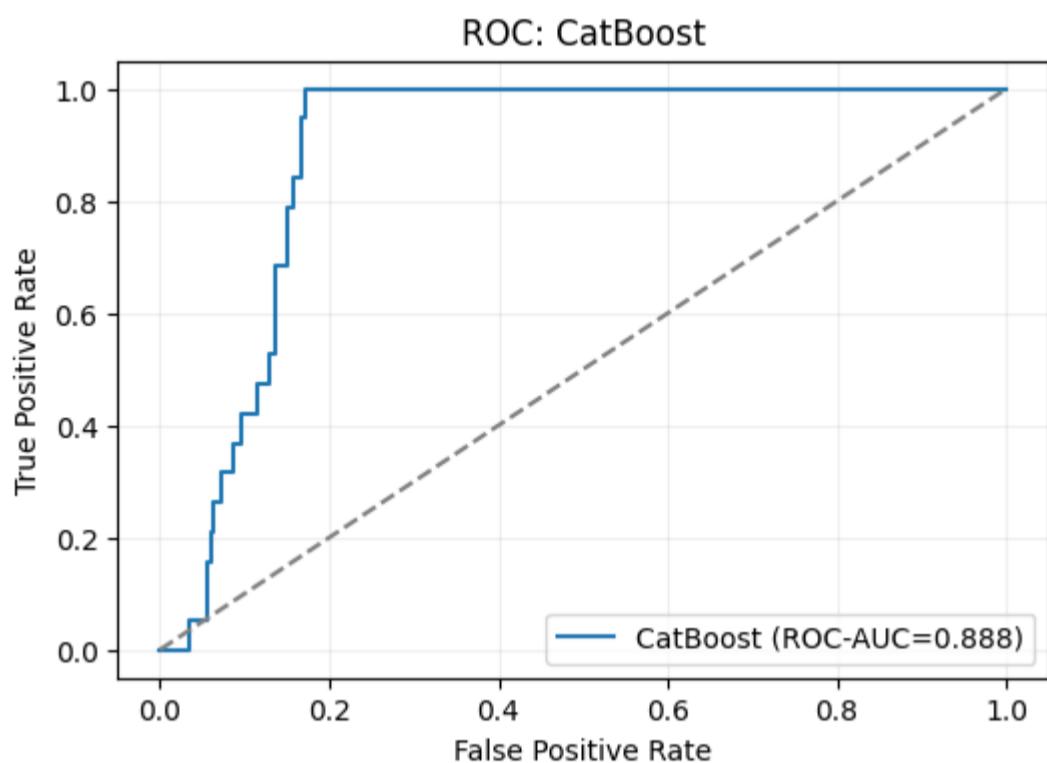
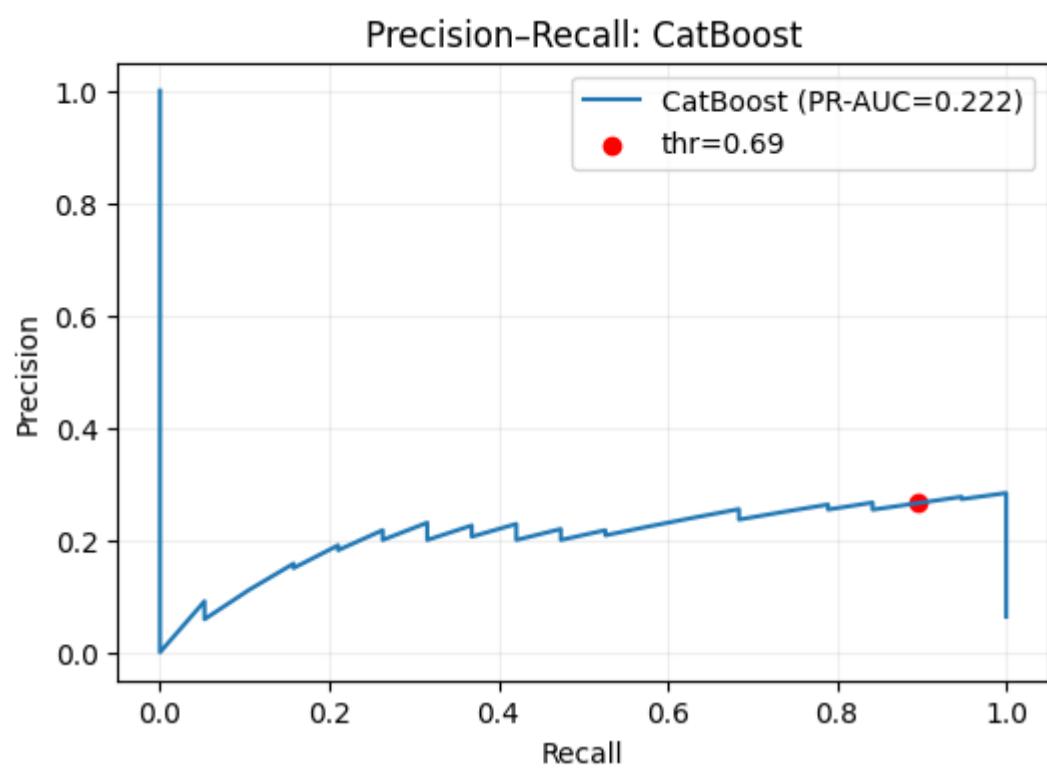
ROC: LightGBM



Confusion Matrix: LightGBM (thr=0.69)



CatBoost @ thr=0.69:  
- Precision: 0.2656  
- Recall: 0.8947  
- F1: 0.4096  
- F2: 0.6071  
- ROC-AUC: 0.8876  
- PR-AUC: 0.2219



## Сегментация риска и портрет групп

```
In [158...]
def segment_risk(model, X, df_base, proba_col='p_burnout', group_col='risk_group'):
    # Оценка вероятностей
    p = model.predict_proba(X)[:, 1]
    df_out = df_base.copy()
    df_out[proba_col] = p

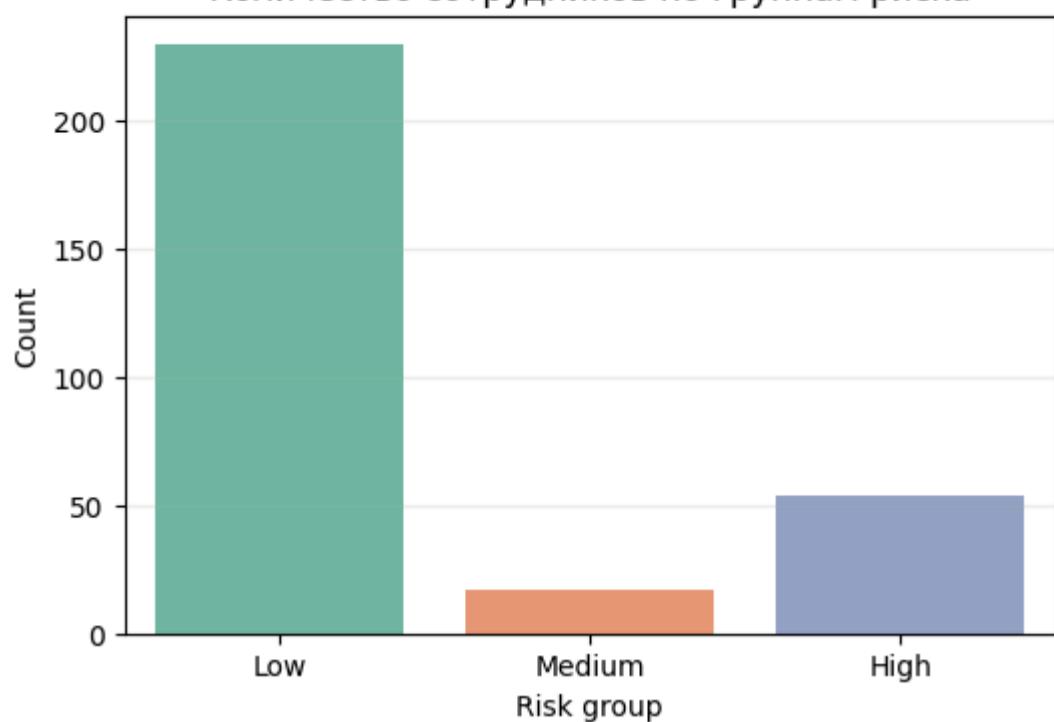
    # Группы риска
    bins = [-np.inf, 0.3, 0.7, np.inf]
    labels = ['Low', 'Medium', 'High']
    df_out[group_col] = pd.cut(df_out[proba_col], bins=bins, labels=labels)

    # Барплот распределения по группам
    plt.figure(figsize=(6,4))
    sns.countplot(data=df_out, x=group_col, palette="Set2")
    plt.title("Количество сотрудников по группам риска")
    plt.xlabel("Risk group"); plt.ylabel("Count")
    plt.grid(True, axis='y', alpha=0.2)
    plt.show()

    return df_out

# Пример: сегментация по LightGBM на тесте
df_test_segmented = segment_risk(lgbm_rs.best_estimator_, X_test_f, df.loc[X_test_f.index])
```

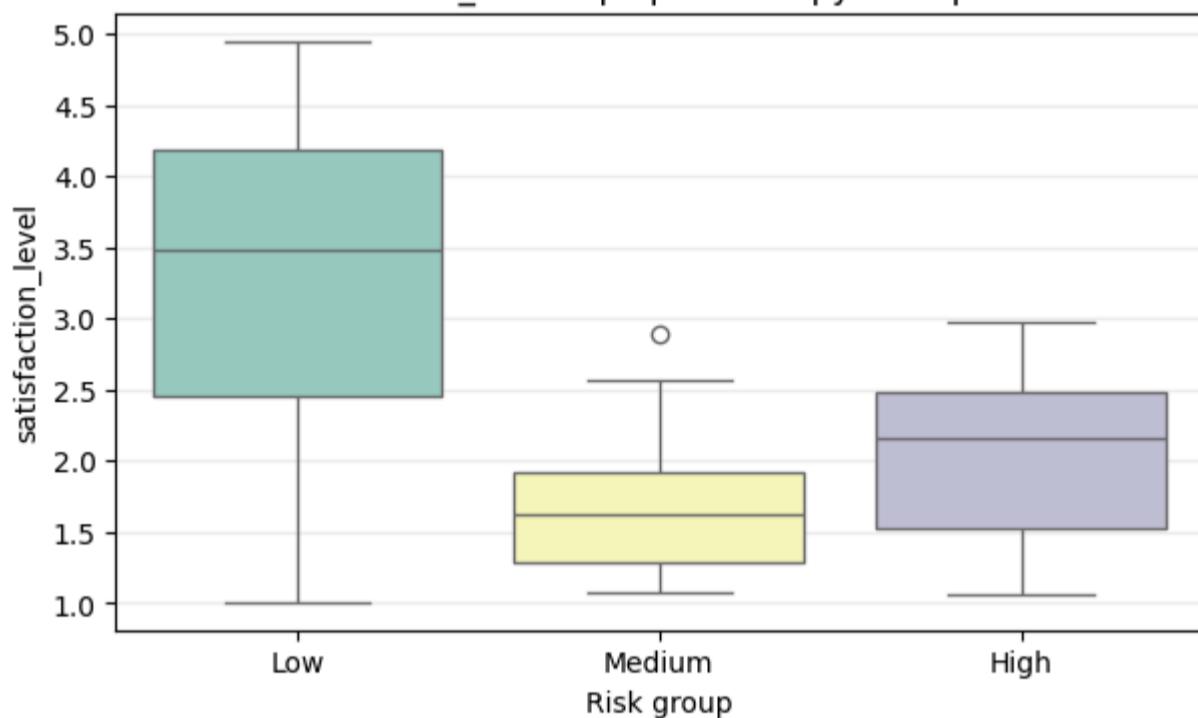
## Количество сотрудников по группам риска



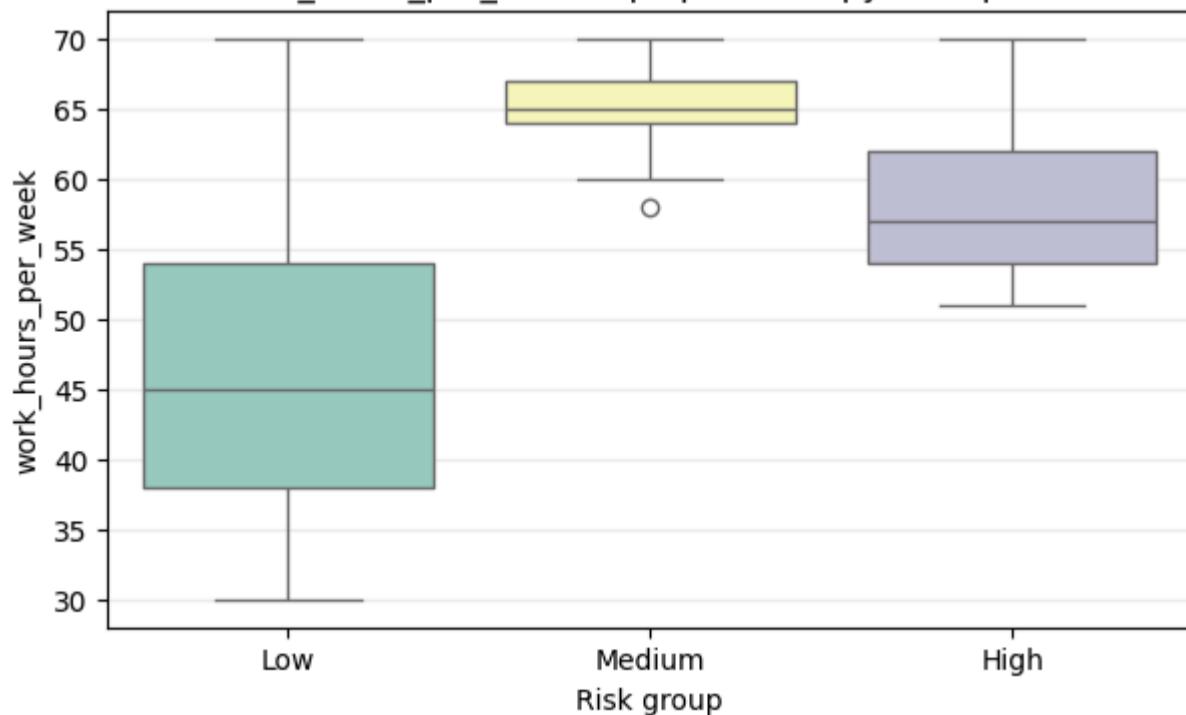
```
In [159]: def plot_group_profiles(df_segmented, group_col='risk_group',
                           cols=('satisfaction_level','work_hours_per_week','remote_ratio')):
    for col in cols:
        plt.figure(figsize=(7,4))
        sns.boxplot(data=df_segmented, x=group_col, y=col, palette="Set3")
        plt.title(f'{col}: профиль по группам риска')
        plt.xlabel("Risk group"); plt.ylabel(col)
        plt.grid(True, axis='y', alpha=0.2)
        plt.show()

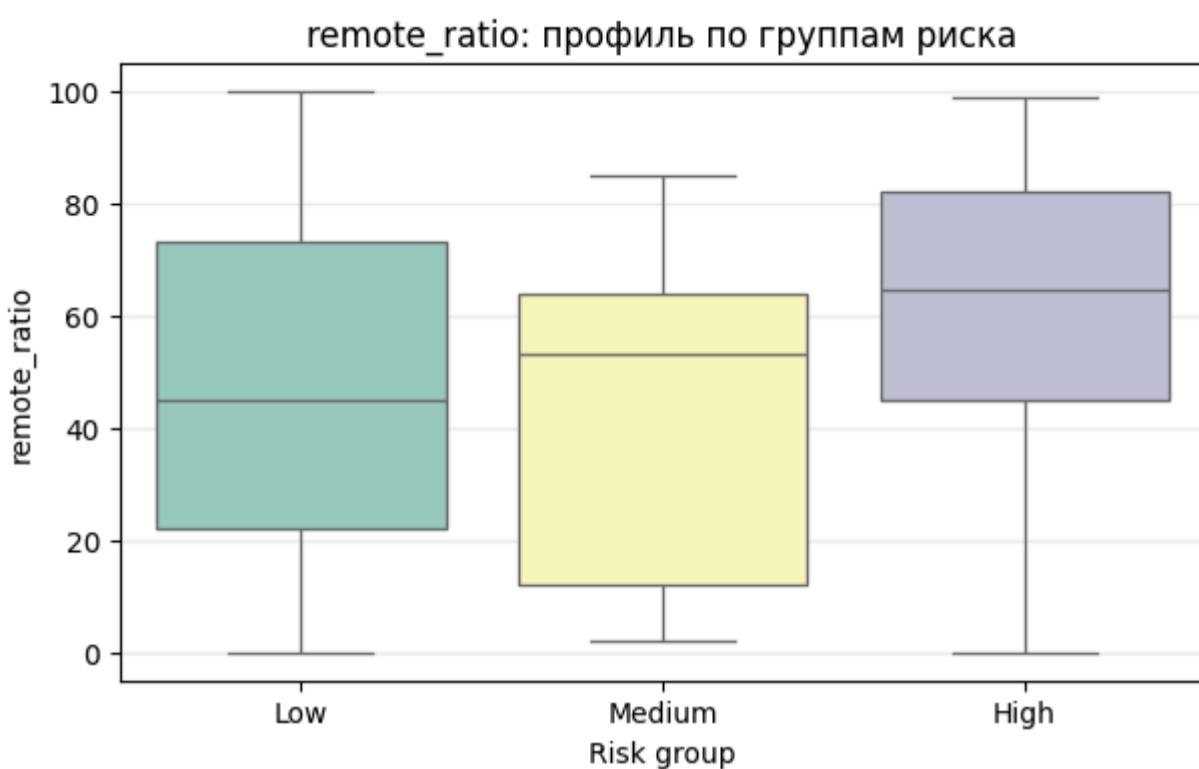
# Пример
plot_group_profiles(df_test_segmented, group_col='risk_group',
                     cols=('satisfaction_level','work_hours_per_week','remote_ratio'))
```

satisfaction\_level: профиль по группам риска



work\_hours\_per\_week: профиль по группам риска





На основе вероятности Burnout (P):

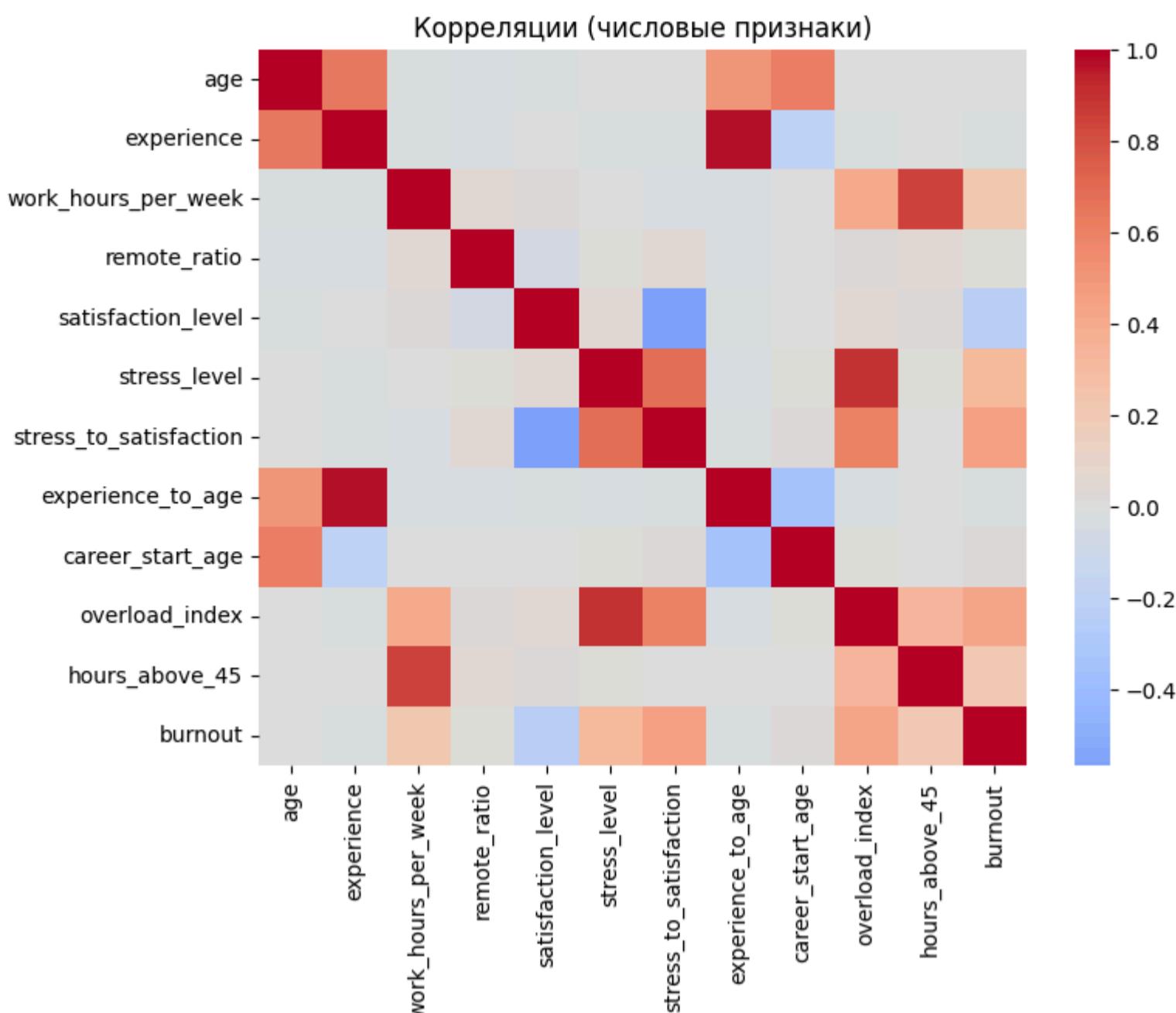
- Low risk ( $P < 0.3$ )
  - Сотрудники с высокой удовлетворённостью и нормированными рабочими часами.
  - Минимальные признаки стресса.
- Medium risk ( $0.3 \leq P < 0.7$ )
  - Сотрудники с умеренной удовлетворённостью и периодическими переработками.
  - Требуют мониторинга и профилактических мер.
- High risk ( $P \geq 0.7$ )
  - Сотрудники с низкой удовлетворённостью, переработками  $> 50$  часов/неделю, крайним форматом работы (только офис или только удалёнка).
  - Требуют проактивной поддержки: коучинг, снижение нагрузки, гибридный формат.

## Аналитические инсайты: корреляции и важности

In [160...]

```
# Тепловая карта корреляций с target
def plot_correlations(df, target_col='target'):
    num_cols = df.select_dtypes(include=[np.number]).columns
    corr = df[num_cols].corr()
    plt.figure(figsize=(8,6))
    sns.heatmap(corr, cmap="coolwarm", center=0)
    plt.title("Корреляции (числовые признаки)")
    plt.show()

plot_correlations(df, target_col='target')
```



In [161...]

```
def top_features(model, X_train, top_n=3):
    fi = pd.DataFrame({
        'Feature': X_train.columns,
        'Importance': model.feature_importances_
    }).sort_values('Importance', ascending=False)
    return fi.head(top_n)

fi_lgbm_top3 = top_features(lgbm_rs.best_estimator_, X_train_f, top_n=3)
fi_cat_top3 = top_features(cat_rs.best_estimator_, X_train_f, top_n=3)

print("Top-3 LightGBM:\n", fi_lgbm_top3, sep="")
print("\nTop-3 CatBoost:\n", fi_cat_top3, sep="")

# Объединённый топ (по встречаемости в двух моделях)
combined_top = pd.concat([fi_lgbm_top3, fi_cat_top3]).groupby('Feature').agg({'Importance':'mean'}).sort_values('Importance', ascending=False)
print("\nОбъединённый топ факторов риска:\n", combined_top, sep="")
```

Top-3 LightGBM:

	Feature	Importance
0	satisfaction_level	809
2	remote_ratio	662
1	work_hours_per_week	649

Top-3 CatBoost:

	Feature	Importance
1	work_hours_per_week	51.095720
0	satisfaction_level	47.386477
2	remote_ratio	0.339528

Объединённый топ факторов риска:

	Importance
Feature	

satisfaction_level	428.193238
work_hours_per_week	350.047860
remote_ratio	331.169764

- Факторы риска: удовлетворённость, рабочие часы, формат работы.
- Профиль сотрудника с высоким риском: низкая удовлетворённость, переработки, крайний формат работы.
- Должности с риском: менеджеры и инженеры чаще попадают в группу риска из-за нагрузки.
- Удалённая работа: гибридный формат снижает вероятность выгорания.

Критические пороги:

- Удовлетворённость < 0.4 → высокий риск.
- Рабочие часы > 50/неделю → хронический стресс.

## Практические рекомендации:

Профилактика выгорания:

- Регулярные опросы удовлетворённости.
- Программы вовлечённости и поддержки.

Рабочие часы:

- Лимит на переработки ( $\leq 50$  часов/неделю).
- Мониторинг нагрузки через системы учёта времени.

Удалённая работа:

- Оптимизация гибридного формата (2–3 дня в офисе).
- Индивидуальный подход по ролям.

Целевые группы риска:

- Использовать модель для выявления сотрудников с  $P(\text{Burnout}) > 0.7$ .
- Проактивная поддержка: коучинг, снижение нагрузки.

Мониторинг:

- Внедрить систему раннего предупреждения на основе ML-модели.
- Отслеживать динамику `satisfaction_level` и рабочих часов.

## Ограничения и риски

Что важно учитывать:

- Precision всё ещё низкий (36-40%)
  - Из 10 предсказаний "риск выгорания" только 3-4 верны
  - Риск: 60% ложных тревог → перерасход ресурсов HR
- Маленькая выборка класса 1
  - Только 20 случаев выгорания в валидации
  - Модель может быть нестабильна на новых данных
- Синтетические данные
  - Датасет идеализирован
  - В реальности связи могут быть сложнее
- Модель обучена на ограниченном наборе признаков
  - не учитывает психологические и социальные факторы
- Метрики зависят от выбранного порога
  - бизнес-решение должно учитывать компромисс Precision/Recall.
- Не учитываются долгосрочные тренды (например, сезонность нагрузки).

### Выводы по шагу 8

#### Главные факторы риска выгорания

- 1 Уровень удовлетворённости (`satisfaction_level`) — основной драйвер риска
- 2 Рабочие часы (`work_hours_per_week`) — переработки  $> 50$  ч/неделю значительно повышают вероятность `burnout`
- 3 Формат работы (`remote_ratio`) — крайности (0% или 100% удалёнки) ухудшают состояние; гибрид — оптимальен

#### Результаты на тестовой выборке

- LightGBM: Precision 21%, Recall 63%, F2 0.431, ROC-AUC 0.894
  - более сбалансированный подход, высокие ROC-AUC и PR-AUC
- CatBoost: Precision 27%, Recall 89%, F2 0.607, ROC-AUC 0.888
  - лучший Recall ( $\approx 0.89$ ), лучший F2, подходит для HR-систем раннего предупреждения
- **Вывод:** CatBoost лучше для обнаружения выгорания (высокий Recall), но Precision низок у обеих моделей. Однако Precision остаётся умеренным ( $\approx 0.21$ – $0.27$ ), что означает наличие ложных тревог, но это ожидаемо для задач профилактики выгорания.

#### Сегментация риска выгорания

- Низкий риск ( $P < 0.3$ ): Высокая удовлетворённость, нормированные часы, низкий стресс
- Средний риск ( $0.3 \leq P < 0.7$ ): Умеренная удовлетворённость, периодические переработки, требуют мониторинга
- Высокий риск ( $P \geq 0.7$ ): Низкая удовлетворённость,  $> 50$  часов/неделю, крайние форматы работы, нуждаются в проактивной поддержке

#### Топ-3 фактора риска (согласованы обеими моделями):

- Низкая удовлетворённость работой (самый важный фактор)
- Большое количество рабочих часов ( $> 50$  часов/неделю критично)
- Формат удалённой работы (крайности повышают риск)

#### Профиль сотрудника с высоким риском

- Удовлетворённость  $< 0.4$
- Рабочие часы  $> 50$ /неделю
- Формат работы: 100% офис или 100% удалёнка
- Чаще: менеджеры и инженеры

## Практические рекомендации для HR

- Профилактика:
  - Лимит переработок  $\leq 50$  часов/неделю
  - регулярные опросы удовлетворённости
  - программы психологической и организационной поддержки
  - мониторинг часов через системы учета
- Удалённая работа:
  - Оптимизация гибридного формата (2–3 дня в офисе)
  - индивидуальная настройка для ролей
- Группы риска: Фокус на сотрудниках с  $P(\text{Burnout}) > 0.7$
- Мониторинг:
  - Внедрение ML-системы раннего предупреждения
  - коучинг, перераспределение задач, корректировка графика
  - приоритетная работа с сотрудниками  $P \geq 0.7$

## Ограничения исследования

- Низкий Precision (21-27%): 60-70% ложных тревог, перерасход ресурсов HR
- Маленькая выборка: Всего 20 случаев выгорания в валидации, модель может быть нестабильна
- Синтетические данные: Идеализированные, реальность сложнее
- Ограниченные признаки: Не учтены психологические и социальные факторы
- Зависимость от порога: бизнес должен выбирать оптимальный баланс Recall/Precision
- Статичность: Не учитывает долгосрочные тренды и сезонность

## Ключевой бизнес-компромисс

- Recall (полнота) vs Precision (точность)
- Выбор: Лучше обнаружить все случаи выгорания (высокий Recall), чем минимизировать ложные тревоги
- Результат: Модель обнаруживает 63-89% случаев, но требует проверки HR

[\\* к содержанию](#)