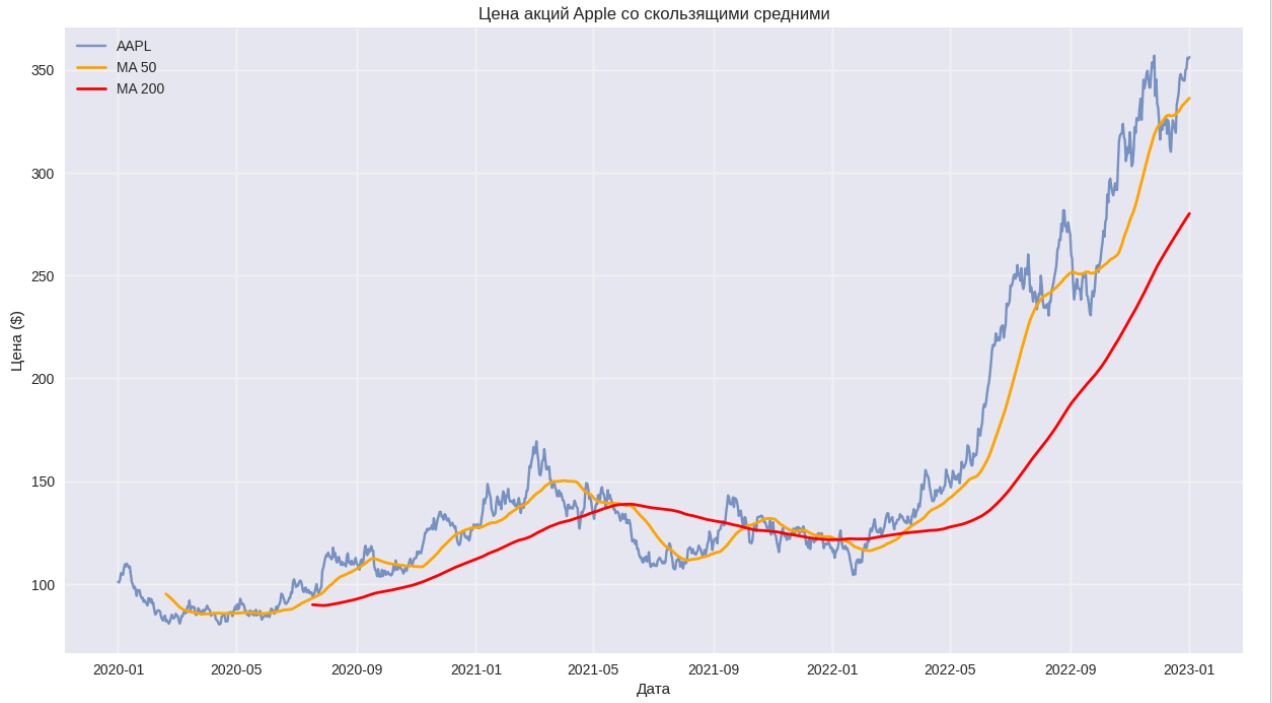
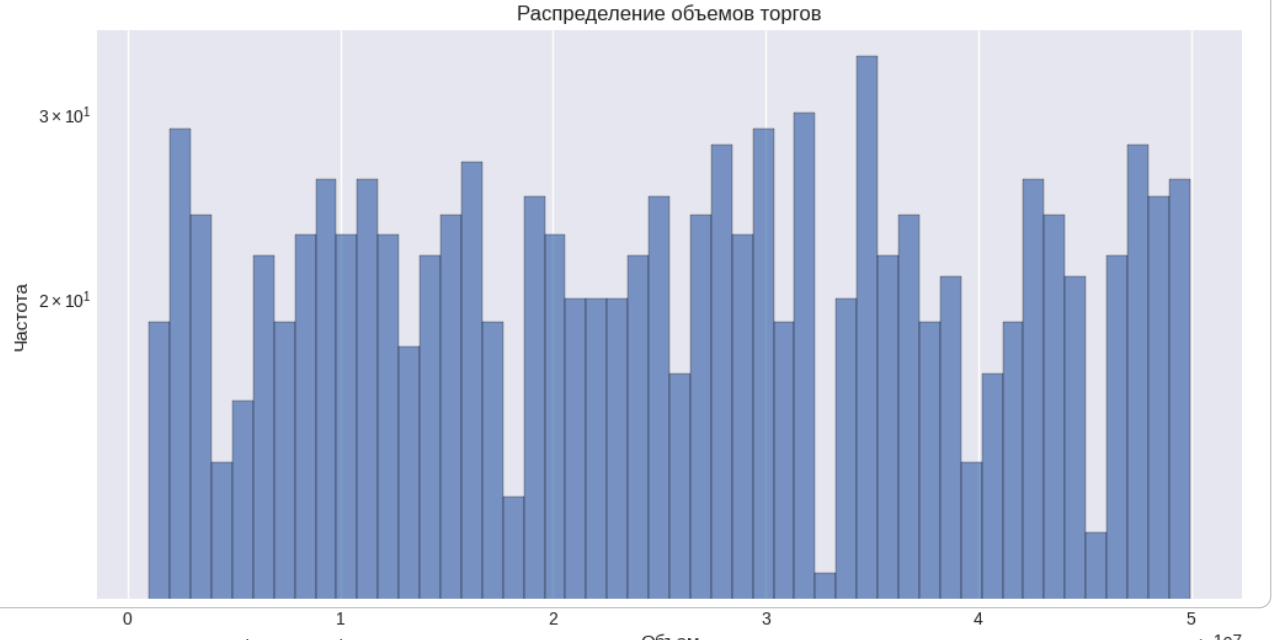
Задание 1: Визуализация финансовых данных (25 минут)

```
# Импорт необходимых библиотек
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta

# Генерация синтетических данных об акциях
np.random.seed(42)  # Фиксируем seed для воспроизводимости
dates = pd.date_range(start='2020-01-01', end='2023-01-01', freq='D')  # Ежедневные даты за 3 года

# Функция для генерации цен акций с геометрическим броуновским движением
def generate_stock_data(dates, base_price, volatility, trend):
    # Генерация случайных доходностей (returns)
    returns = np.random.normal(0, volatility, len(dates)) + trend
    # Преобразование в цены: P(t) = P(0) * exp(Σ returns)
    prices = base_price * np.exp(np.cumsum(returns))
    return prices

# Создание DataFrame с данными трех акций
data = {
    'Date': dates,
    'AAPL': generate_stock_data(dates, 100, 0.02, 0.0005),      # Apple
    'GOOGL': generate_stock_data(dates, 1500, 0.025, 0.0003),   # Google
    'MSFT': generate_stock_data(dates, 200, 0.018, 0.0004)      # Microsoft
}
df = pd.DataFrame(data)
df.set_index('Date', inplace=True)  # Устанавливаем дату как индекс

# Вычисление скользящих средних для Apple
df['AAPL_MA50'] = df['AAPL'].rolling(window=50).mean()    # 50-дневная скользящая средняя
df['AAPL_MA200'] = df['AAPL'].rolling(window=200).mean()  # 200-дневная скользящая средняя

# Генерация случайных объемов торгов
df['Volume'] = np.random.randint(1000000, 50000000, len(df))

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 1: Линейный график с скользящими средними
# ------------------------------------------------------------
plt.figure(figsize=(15, 8))

# Основная цена акций Apple
plt.plot(df.index, df['AAPL'], label='AAPL', alpha=0.7)

# Скользящие средние
plt.plot(df.index, df['AAPL_MA50'], label='MA 50', color='orange', linewidth=2)
plt.plot(df.index, df['AAPL_MA200'], label='MA 200', color='red', linewidth=2)

plt.title('Цена акций Apple со скользящими средними')
plt.xlabel('Дата')
plt.ylabel('Цена ($)')
plt.legend()
plt.grid(True, alpha=0.3)  # Полупрозрачная сетка
plt.show()

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 2: Гистограмма объемов торгов
# ------------------------------------------------------------
plt.figure(figsize=(12, 6))

# Гистограмма с 50 бинами (столбцами)
plt.hist(df['Volume'], bins=50, edgecolor='black', alpha=0.7)

plt.title('Распределение объемов торгов')
plt.xlabel('Объем')
plt.ylabel('Частота')
plt.yscale('log')  # Логарифмическая шкала по Y для лучшего отображения
plt.show()

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 3: Корреляционная матрица
# ------------------------------------------------------------
# Вычисляем корреляционную матрицу только для цен акций
corr_matrix = df[['AAPL', 'GOOGL', 'MSFT']].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix,
            annot=True,          # Показывать значения в ячейках
```

```
            cmap='coolwarm',     # Цветовая карта (синий-красный)
            center=0,            # Центр цветовой шкалы на 0
            square=True,         # Квадратные ячейки
            fmt='.2f',           # Формат чисел (2 знака после запятой)
            cbar_kws={'label': 'Корреляция'})  # Подпись цветовой шкалы
plt.title('Корреляционная матрица акций')
plt.show()

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 4: Scatter plot matrix (Pairplot)
# ------------------------------------------------------------
sns.pairplot(df[['AAPL', 'GOOGL', 'MSFT']].dropna(),  # Удаляем NaN значения
             diag_kind='kde',     # На диагонали - распределение (Kernel Density Estimation)
             plot_kws={'alpha': 0.6})  # Прозрачность точек 60%

plt.suptitle('Pairplot акций', y=1.02)  # Общий заголовок с небольшим смещением
plt.show()
```



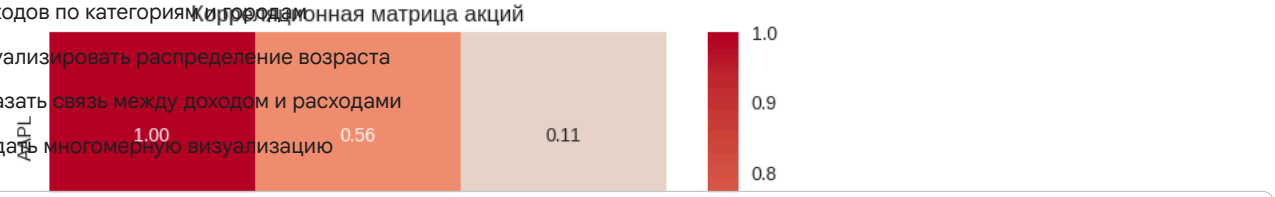


Задание 2: Анализ клиентов (20 минут) Задача: Проанализировать синтетические данные о клиентах: Создать сводную таблицу расходов по категориям и городам

Визуализировать распределение возраста

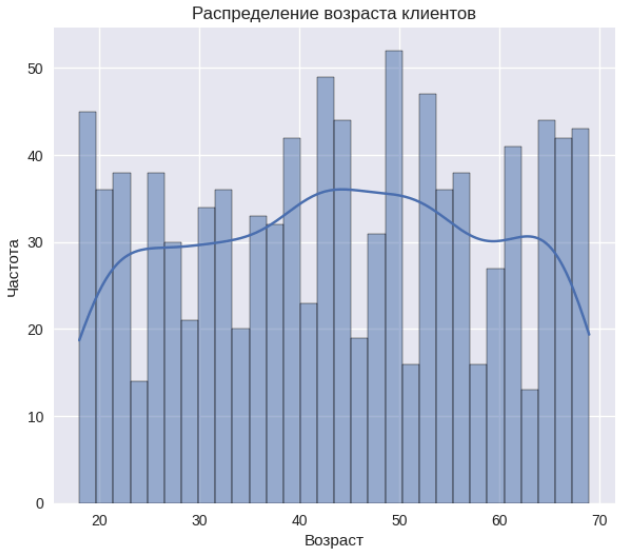
Показать связь между доходом и расходами

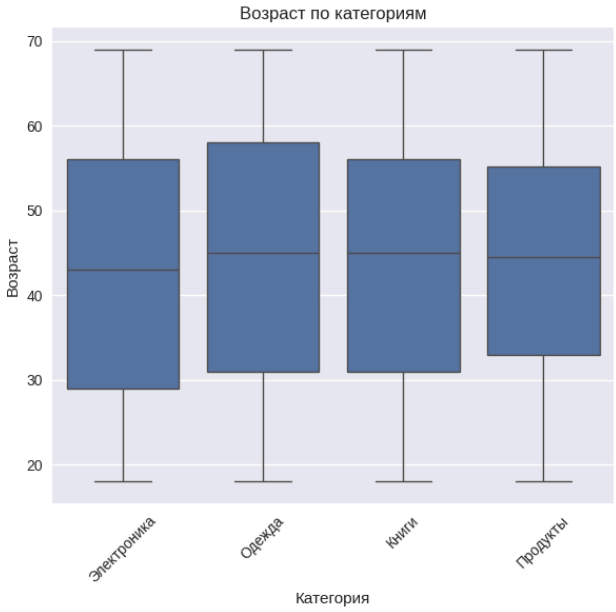
Создать многомерную визуализацию

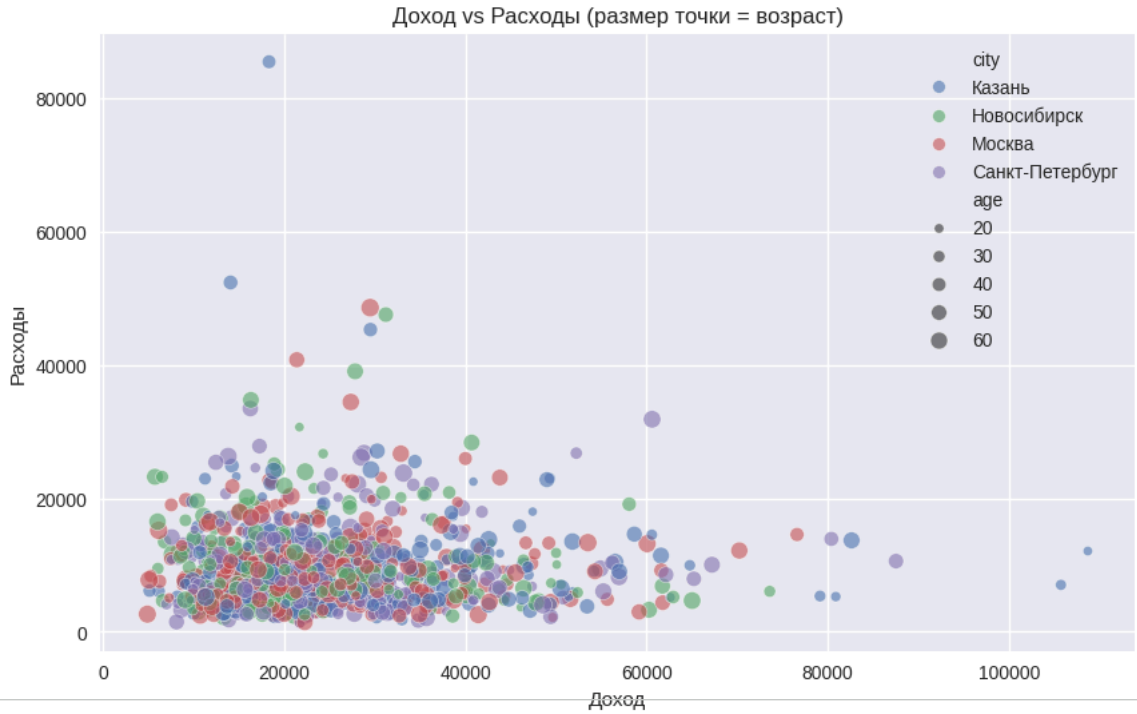


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Генерация синтетических данных о клиентах
np.random.seed(42)  # Фиксируем seed для воспроизводимости
n = 1000  # Количество клиентов

data = {
    'client_id': range(1, n+1),  # ID клиентов от 1 до 1000
    'age': np.random.randint(18, 70, n),  # Возраст от 18 до 69
    'city': np.random.choice(['Москва', 'Санкт-Петербург', 'Казань', 'Новосибирск'], n),  # Города
    'category': np.random.choice(['Электроника', 'Одежда', 'Продукты', 'Книги'], n),  # Категории покупок
    'income': np.random.lognormal(10, 0.5, n),  # Логнормальное распределение дохода
    'spending': np.random.lognormal(9, 0.6, n)  # Логнормальное распределение расходов
}
df = pd.DataFrame(data)  # Создаем DataFrame

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 1: Распределение возраста (2 графика рядом)
# ------------------------------------------------------------
plt.figure(figsize=(12, 6))  # Создаем фигуру 12x6 дюймов

# Левый график: гистограмма распределения возраста
plt.subplot(1, 2, 1)  # 1 строка, 2 столбца, 1-я позиция
sns.histplot(df['age'], kde=True, bins=30)  # kde=True добавляет кривую плотности
plt.title('Распределение возраста клиентов')
plt.xlabel('Возраст')
plt.ylabel('Частота')

# Правый график: box plot возраста по категориям
plt.subplot(1, 2, 2)  # 1 строка, 2 столбца, 2-я позиция
sns.boxplot(x='category', y='age', data=df)
plt.title('Возраст по категориям')
plt.xlabel('Категория')
plt.ylabel('Возраст')
plt.xticks(rotation=45)  # Поворачиваем подписи на 45 градусов

plt.tight_layout()  # Автоматическая регулировка отступов
plt.show()

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 2: Связь дохода и расходов (scatter plot)
# ------------------------------------------------------------
plt.figure(figsize=(10, 6))
# Многомерный scatter plot:
# - x='income': доход по оси X
# - y='spending': расходы по оси Y
# - hue='city': цвет точек по городам
# - size='age': размер точек по возрасту
# - alpha=0.6: прозрачность 60%
sns.scatterplot(x='income', y='spending', hue='city', size='age', data=df, alpha=0.6)
plt.title('Доход vs Расходы (размер точки = возраст)')
plt.xlabel('Доход')
plt.ylabel('Расходы')
plt.show()

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 3: Тепловая карта сводной таблицы
# ------------------------------------------------------------
# Создаем сводную таблицу (pivot table):
# - index='city': города по строкам
# - columns='category': категории по столбцам
# - values='spending': агрегируемые значения (расходы)
# - aggfunc='mean': функция агрегации (среднее)
pivot_table = df.pivot_table(index='city', columns='category', values='spending', aggfunc='mean')

print("=== Средние расходы по городам и категориям ===")
print(pivot_table)

# Визуализируем сводную таблицу как тепловую карту
plt.figure(figsize=(10, 8))
sns.heatmap(pivot_table,
            annot=True,        # Показать значения в ячейках
            cmap='YlOrRd',     # Желто-оранжево-красная палитра
            fmt='.0f')         # Формат чисел (целые числа)
plt.title('Средние расходы по городам и категориям')
plt.xlabel('Категория')
plt.ylabel('Город')
plt.show()
```

```
# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 4: Многомерный анализ с FacetGrid
# ------------------------------------------------------------
# FacetGrid создает сетку графиков по категориям
# - col='city': отдельный график для каждого города
# - hue='category': цвет точек по категориям покупок
# - col_wrap=2: максимум 2 графика в строке
# - height=4: высота каждого графика
g = sns.FacetGrid(df, col='city', hue='category', col_wrap=2, height=4)

# Применяем scatterplot к каждому subplot'y
g.map(sns.scatterplot, 'income', 'spending', alpha=0.6)

g.add_legend()  # Добавляем легенду
g.fig.suptitle('Доход vs Расходы по городам и категориям', y=1.02)  # Общий заголовок
plt.show()
```







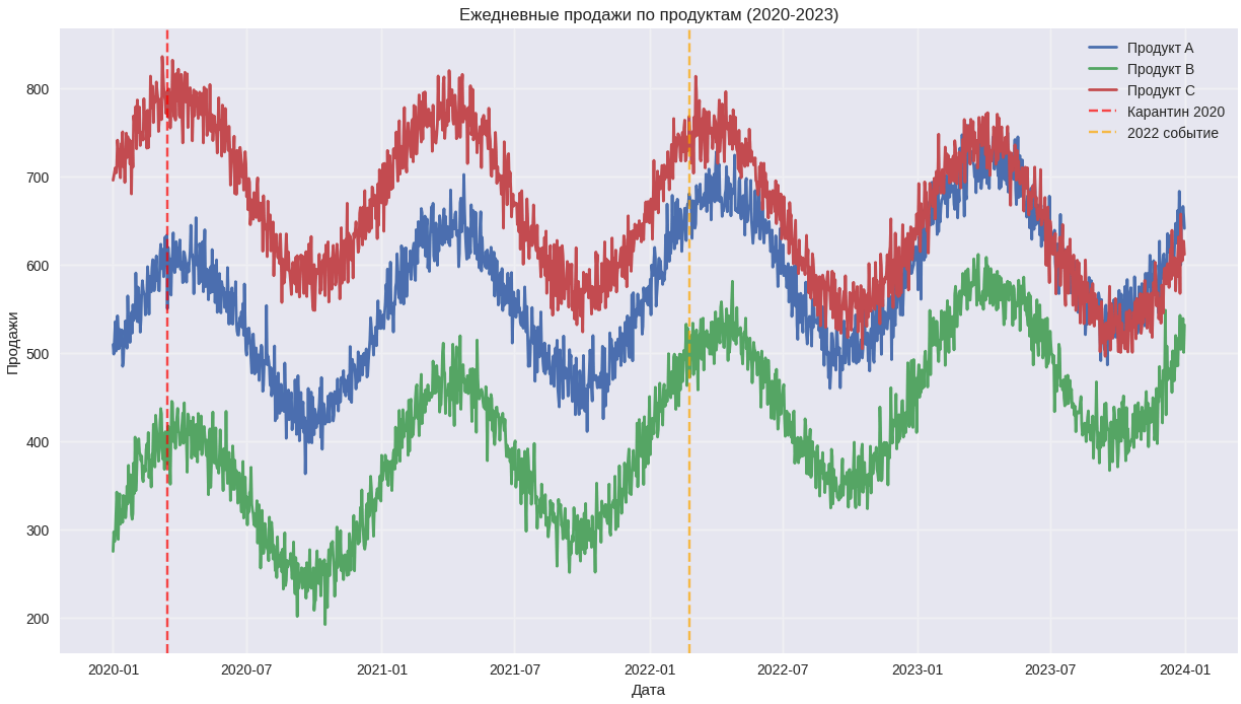
Задание 3: Визуализация временных рядов (15 минут) Задача: Создать визуализацию для временных данных: Построить линейный график с множественными временными рядами Добавить аннотации важных событий Создать тепловую карту по сезонам

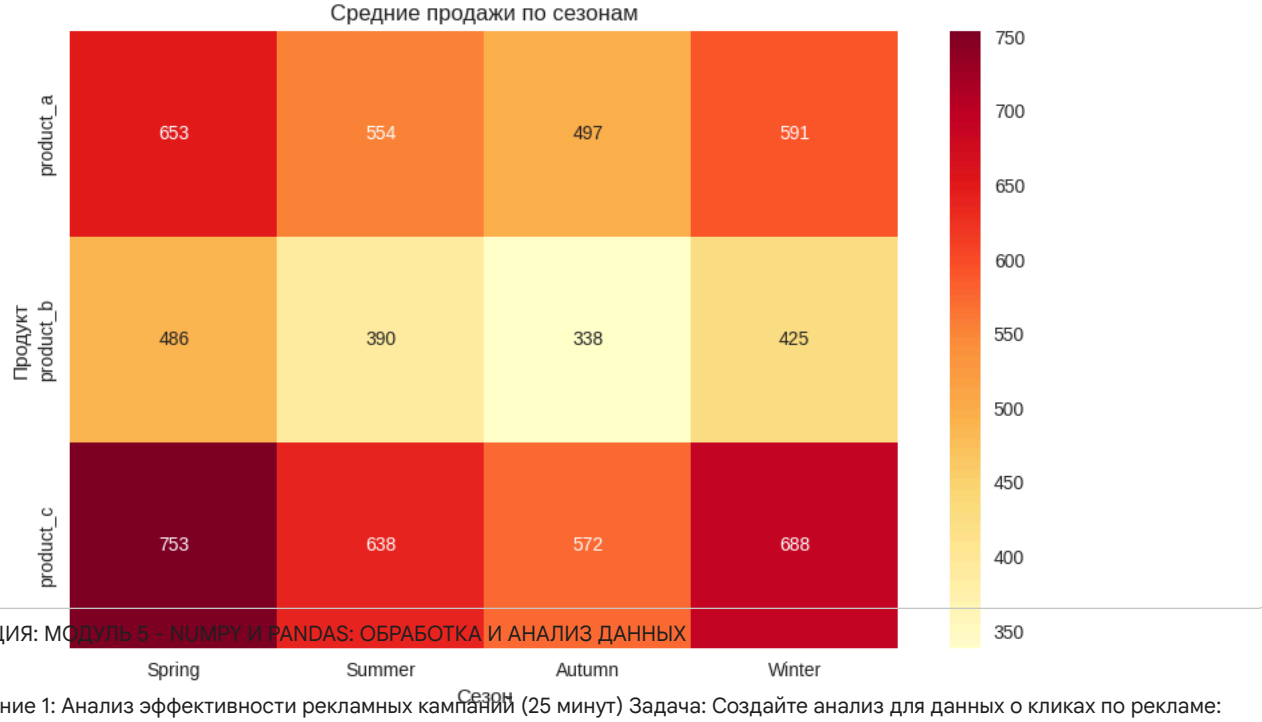
```
=== Средние расходы по городам и категориям ===
category      Книги      Одежда    Продукты  Электроника
city
Казань  10847.179687  8589.025711  10051.613971  9569.491528
Москва   9354.841007  10525.580138   9794.395999  9072.170638
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta

# Генерация данных о продажах
```

```
np.random.seed(42)  # Фиксируем seed для воспроизводимости
dates = pd.date_range(start='2020-01-01', end='2023-12-31', freq='D')  # Ежедневные данные за 4 года

# Функция для генерации данных с трендом, сезонностью и шумом
def generate_sales(dates, base, trend, seasonality):
    t = np.arange(len(dates))  # Временной индекс
    trend_component = t * trend  # Линейный тренд
    seasonal_component = 100 * np.sin(2 * np.pi * t / 365.25)  # Годовая сезонность (365.25 дней)
    noise = np.random.normal(0, 20, len(dates))  # Случайный шум
    return base + trend_component + seasonal_component + noise

# Создаем DataFrame с тремя продуктами
df = pd.DataFrame({
    'date': dates,
    'product_a': generate_sales(dates, 500, 0.1, 100),    # Рост 0.1 в день
    'product_b': generate_sales(dates, 300, 0.15, 80),    # Рост 0.15 в день
    'product_c': generate_sales(dates, 700, -0.05, 120)   # Спад -0.05 в день
})
df.set_index('date', inplace=True)  # Устанавливаем дату как индекс

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 1: Многокомпонентный временной ряд
# ------------------------------------------------------------
plt.figure(figsize=(15, 8))

# Основные линии продаж
plt.plot(df.index, df['product_a'], label='Продукт A', linewidth=2)
plt.plot(df.index, df['product_b'], label='Продукт B', linewidth=2)
plt.plot(df.index, df['product_c'], label='Продукт C', linewidth=2)

# Аннотации важных событий (вертикальные линии)
plt.axvline(datetime(2020, 3, 15), color='red', linestyle='--', alpha=0.7, label='Карантин 2020')
plt.axvline(datetime(2022, 2, 24), color='orange', linestyle='--', alpha=0.7, label='2022 событие')

plt.title('Ежедневные продажи по продуктам (2020-2023)')
plt.xlabel('Дата')
plt.ylabel('Продажи')
plt.legend()
plt.grid(True, alpha=0.3)  # Полупрозрачная сетка
plt.show()

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 2: Тепловая карта по сезонам
# ------------------------------------------------------------
# Добавляем метаданные для сезонного анализа
df['year'] = df.index.year  # Год
df['month'] = df.index.month  # Месяц

# Создаем сезонные метки
df['season'] = df['month'].map({12: 'Winter', 1: 'Winter', 2: 'Winter',
                                3: 'Spring', 4: 'Spring', 5: 'Spring',
                                6: 'Summer', 7: 'Summer', 8: 'Summer',
                                9: 'Autumn', 10: 'Autumn', 11: 'Autumn'})

# Группируем по сезонам и вычисляем средние продажи
seasonal_data = df.groupby('season')[['product_a', 'product_b', 'product_c']].mean()

# Переупорядочение по логическому порядку сезонов
season_order = ['Spring', 'Summer', 'Autumn', 'Winter']
seasonal_data = seasonal_data.reindex(season_order)

# Тепловая карта сезонных продаж
plt.figure(figsize=(10, 6))
sns.heatmap(seasonal_data.T,  # Транспонируем: продукты по строкам, сезоны по столбцам
            annot=True,       # Показывать числовые значения
            cmap='YlOrRd',    # Желто-оранжево-красная палитра
            fmt='.0f')        # Формат чисел (целые)
plt.title('Средние продажи по сезонам')
plt.xlabel('Сезон')
plt.ylabel('Продукт')
plt.show()

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 3: Годовое сравнение (месячные данные)
# ------------------------------------------------------------
# Агрегируем данные до месячного уровня
df['year_month'] = df.index.to_period('M')  # Создаем период месяц-год
monthly_sales = df.groupby('year_month')[['product_a', 'product_b', 'product_c']].mean()

plt.figure(figsize=(15, 8))

# Месячные продажи с маркерами
```

```
plt.plot(monthly_sales.index.to_timestamp(), monthly_sales['product_a'],
         marker='o', label='Продукт A')
plt.plot(monthly_sales.index.to_timestamp(), monthly_sales['product_b'],
         marker='s', label='Продукт B')
plt.plot(monthly_sales.index.to_timestamp(), monthly_sales['product_c'],
         marker='^', label='Продукт C')

plt.title('Месячные продажи по продуктам')
plt.xlabel('Месяц')
plt.ylabel('Продажи')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)  # Поворот меток дат
plt.tight_layout()  # Автоматическая регулировка отступов
plt.show()
```





ЛЕКЦИЯ: МОДУЛЬ 5 - NUMPY И PANDAS: ОБРАБОТКА И АНАЛИЗ ДАННЫХ

Задание 1: Анализ эффективности рекламных кампаний (25 минут) Задача: Создайте анализ для данных о кликах по рекламе: Считать данные из CSV Вычислить CTR (Click-Through Rate) для каждого цвета Проанализировать динамику CTR по дням Создать сводную таблицу эффективности


```
import pandas as pd
import numpy as np
from scipy import stats

# Создание синтетических данных
```

```
np.random.seed(42)  # Фиксируем seed для воспроизводимости

# Количество уникальных цветов
num_colors = 5
# Количество дней
num_days = 30
# Общее количество наблюдений
total_observations = num_colors * num_days # 5 * 30 = 150

# Создаем данные для цветов, повторяя их num_days раз
colors = ['red', 'blue', 'green', 'yellow', 'purple'] * num_days

# Создаем дни: каждый день содержит все num_colors
days = []
for i in range(1, num_days + 1):
    days.extend([f'Day_{i}'] * num_colors)

# Генерация случайных кликов и просмотров (total_observations)
clicks = np.random.randint(10, 100, total_observations)
views = np.random.randint(100, 1000, total_observations)

# Создаем DataFrame
df = pd.DataFrame({
    'color': colors,  # Цвет кнопки (вариант теста)
    'day': days,      # День измерения
    'clicks': clicks, # Количество кликов
    'views': views    # Количество показов/просмотров
})

print(f"Размер датасета: {df.shape}")
print(f"Количество уникальных дней: {df['day'].nunique()}")
print(f"Количество уникальных цветов: {df['color'].nunique()}")

# Вычисление CTR (Click-Through Rate - показатель кликабельности)
df['ctr'] = df['clicks'] / df['views']  # CTR = клики / просмотры

# ------------------------------------------------------------
# АНАЛИЗ 1: Средний CTR по цветам
# ------------------------------------------------------------
# Группируем данные по цветам и вычисляем статистики
ctr_by_color = df.groupby('color')['ctr'].agg(['mean', 'std', 'count'])
print("\n" + "="*50)
print("CTR по цветам:")
print(ctr_by_color.round(4))
# Вывод: среднее CTR, стандартное отклонение и количество наблюдений для каждого цвета

# ------------------------------------------------------------
# АНАЛИЗ 2: Динамика CTR по дням
# ------------------------------------------------------------
# Группируем по дням и вычисляем средний CTR за каждый день
daily_ctr = df.groupby('day')['ctr'].mean()
print("\n" + "="*50)
print("Динамика CTR (первые 5 дней):")
print(daily_ctr.head().round(4))
# Вывод: как меняется средний CTR изо дня в день

# ------------------------------------------------------------
# АНАЛИЗ 3: Сводная таблица (pivot table)
# ------------------------------------------------------------
# Создаем матрицу цвет × день со средним CTR в каждой ячейке
pivot = df.pivot_table(values='ctr', index='color', columns='day', aggfunc='mean')
print("\n" + "="*50)
print("Сводная таблица CTR (первые 5 дней):")
print(pivot.iloc[:, :5].round(4))  # Показываем только первые 5 дней
# Вывод: позволяет увидеть CTR каждого цвета в каждый день

# ------------------------------------------------------------
# АНАЛИЗ 4: Определение лучшего цвета по CTR
# ------------------------------------------------------------
# Находим цвет с максимальным средним CTR
best_color = ctr_by_color['mean'].idxmax()  # Цвет с максимальным средним CTR
print("\n" + "="*50)
print(f"Лучший цвет: {best_color} (CTR: {ctr_by_color.loc[best_color, 'mean']:.4f})")
# Вывод: определяем победителя A/B теста

# ------------------------------------------------------------
# АНАЛИЗ 5: Проверка статистической значимости
# ------------------------------------------------------------
# Выделяем данные для двух цветов
red_ctr = df[df['color'] == 'red']['ctr']
blue_ctr = df[df['color'] == 'blue']['ctr']
```

```
# Проверяем, есть ли достаточно данных
print(f"\nКоличество наблюдений для red: {len(red_ctr)}")
print(f"Количество наблюдений для blue: {len(blue_ctr)}")

# Выполняем t-тест для сравнения средних (двухвыборочный независимый t-тест)
t_stat, p_value = stats.ttest_ind(red_ctr, blue_ctr)
print("\n" + "="*50)
print(f"T-test сравнение red vs blue:")
print(f"t-статистика = {t_stat:.4f}")
print(f"p-value = {p_value:.4f}")

# Интерпретация результата
alpha = 0.05  # Уровень значимости 5%
if p_value < alpha:
    print(f"Статистически значимая разница (p < {alpha})")
    if t_stat > 0:
        print("Red имеет значимо более высокий CTR чем Blue")
    else:
        print("Blue имеет значимо более высокий CTR чем Red")
else:
    print(f"Нет статистически значимой разницы (p \u2265 {alpha})")

# ------------------------------------------------------------
# ДОПОЛНИТЕЛЬНЫЙ АНАЛИЗ: ANOVA для всех цветов
# ------------------------------------------------------------
# Создаем списки CTR для каждого цвета
groups = [df[df['color'] == color]['ctr'] for color in df['color'].unique()]

# Проводим ANOVA тест (сравнение средних для нескольких групп)
f_stat, p_value_anova = stats.f_oneway(*groups)

print("\n" + "="*50)
print("ANOVA тест для всех цветов:")
print(f"F-статистика = {f_stat:.4f}")
print(f"p-value = {p_value_anova:.4f}")

if p_value_anova < alpha:
    print(f"Есть статистически значимые различия между цветами (p < {alpha})")
else:
    print(f"Нет статистически значимых различий между цветами (p \u2265 {alpha})")

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ (если установлен matplotlib)
# ------------------------------------------------------------
try:
    import matplotlib.pyplot as plt

    fig, axes = plt.subplots(2, 2, figsize=(12, 10))

    # 1. Box plot CTR по цветам
    df.boxplot(column='ctr', by='color', ax=axes[0, 0])
    axes[0, 0].set_title('Распределение CTR по цветам')
    axes[0, 0].set_ylabel('CTR')
    axes[0, 0].set_xlabel('Цвет')
    axes[0, 0].tick_params(axis='x', rotation=45)

    # 2. Bar plot средних CTR
    ctr_by_color['mean'].plot(kind='bar', ax=axes[0, 1], color=['red', 'blue', 'green', 'yellow', 'purple'])
    axes[0, 1].set_title('Средний CTR по цветам')
    axes[0, 1].set_ylabel('Средний CTR')
    axes[0, 1].set_xlabel('Цвет')
    axes[0, 1].tick_params(axis='x', rotation=45)

    # 3. Динамика среднего CTR по дням
    daily_ctr.plot(ax=axes[1, 0], marker='o', linewidth=2)
    axes[1, 0].set_title('Динамика среднего CTR по дням')
    axes[1, 0].set_ylabel('Средний CTR')
    axes[1, 0].set_xlabel('День')
    axes[1, 0].tick_params(axis='x', rotation=45)

    # 4. Heatmap сводной таблицы
    im = axes[1, 1].imshow(pivot.values, aspect='auto', cmap='YlOrRd')
    axes[1, 1].set_title('Heatmap CTR: цвет × день')
    axes[1, 1].set_xlabel('День')
    axes[1, 1].set_ylabel('Цвет')
    axes[1, 1].set_xticks(range(len(pivot.columns)))
    axes[1, 1].set_xticklabels([str(i+1) for i in range(len(pivot.columns))], rotation=90, fontsize=8)
    axes[1, 1].set_yticks(range(len(pivot.index)))
    axes[1, 1].set_yticklabels(pivot.index)
    plt.colorbar(im, ax=axes[1, 1], label='CTR')

    plt.suptitle('Анализ A/B теста: цвет кнопки и CTR', fontsize=14)
```

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

except ImportError:
    print("\nДля визуализации установите matplotlib: pip install matplotlib")
```

```
Размер датасета: (150, 4)
Количество уникальных дней: 30
Количество уникальных цветов: 5

==================================================
CTR по цветам:
          mean     std  count
color
blue    0.1075  0.0839     30
green   0.1434  0.1635     30
purple  0.1256  0.1079     30
red     0.0910  0.0621     30
yellow  0.1772  0.1693     30

==================================================
Динамика CTR (первые 5 дней):
day
Day_1     0.1044
Day_10    0.1001
Day_11    0.0597
Day_12    0.1087
Day_13    0.1212
Name: ctr, dtype: float64

==================================================
Сводная таблица CTR (первые 5 дней):
day      Day_1  Day_10  Day_11  Day_12  Day_13
color
blue    0.0411  0.0361  0.0541  0.0895  0.2075
green   0.1601  0.0312  0.1264  0.1132  0.0936
purple  0.0354  0.2130  0.0119  0.0512  0.0832
red     0.0732  0.0941  0.0251  0.1966  0.0708
yellow  0.2121  0.1263  0.0813  0.0928  0.1510

==================================================
Лучший цвет: yellow (CTR: 0.1772)

Количество наблюдений для red: 30
Количество наблюдений для blue: 30

==================================================
T-test сравнение red vs blue:
t-статистика = -0.8632
p-value = 0.3916
Нет статистически значимой разницы (p ≥ 0.05)

==================================================
ANOVA тест для всех цветов:
F-статистика = 2.1430
p-value = 0.0785
Нет статистически значимых различий между цветами (p ≥ 0.05)
```

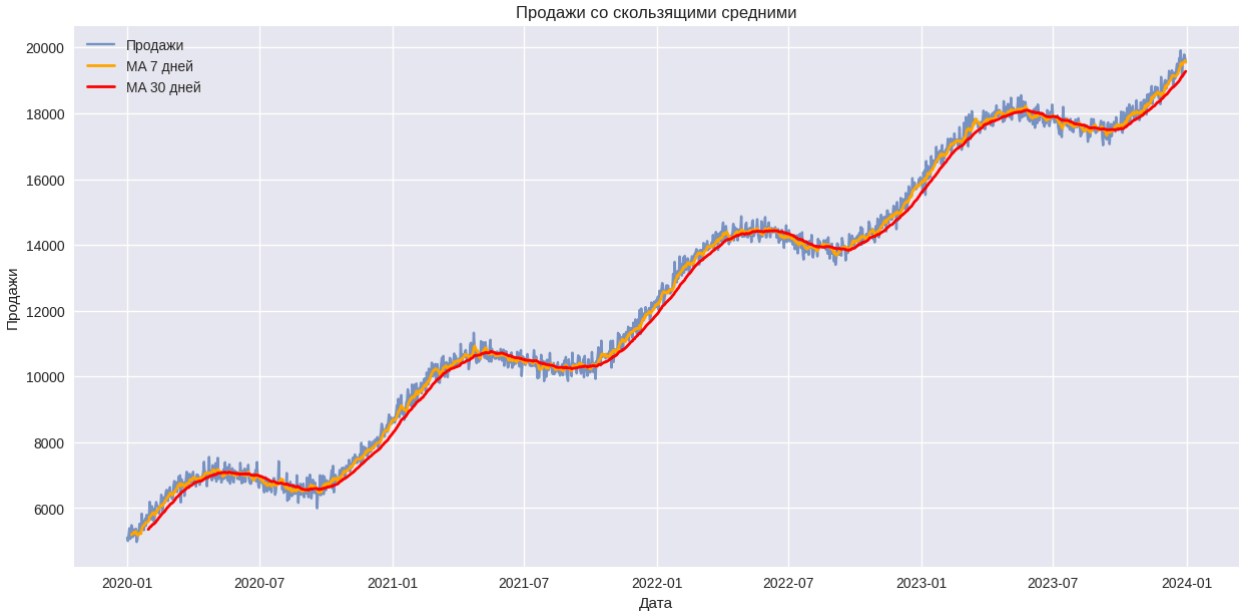
Задание 2: Анализ продаж (20 минут) Задача: Проанализируйте данные о продажах: Создать временной ряд продаж Вычислить скользящие средние Определить сезонность Найти аномалии

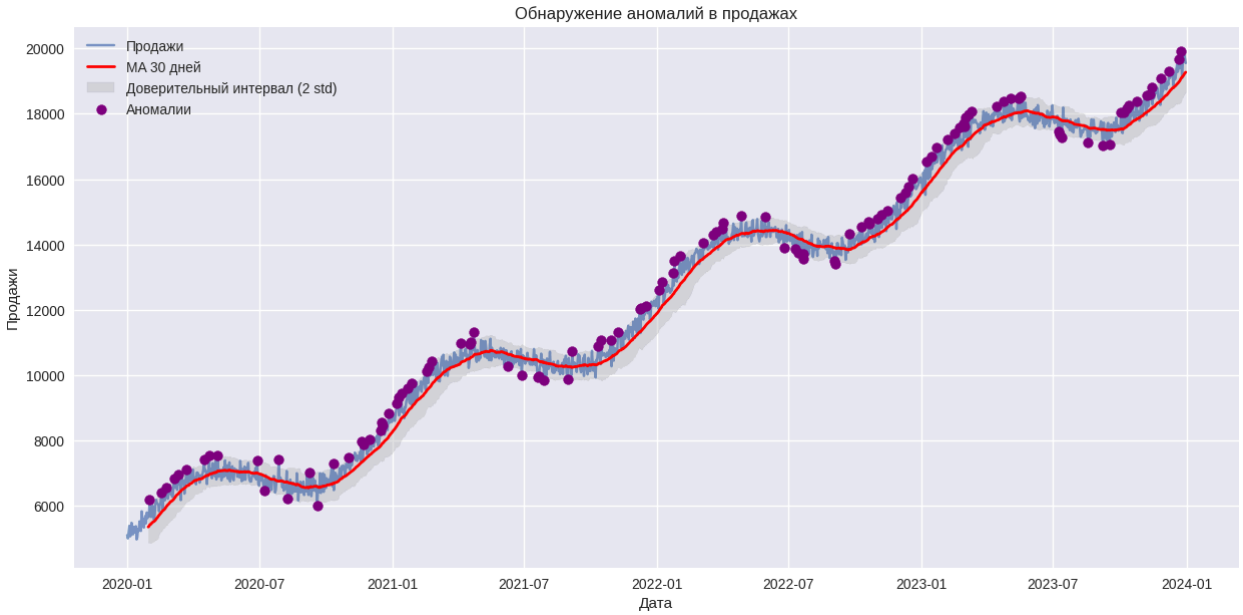
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Создание данных временного ряда
np.random.seed(42)  # Фиксируем seed для воспроизводимости

# Создаем ежедневные даты за 4 года
dates = pd.date_range('2020-01-01', '2023-12-31', freq='D')

# Создаем компоненты временного ряда
t = np.arange(len(dates))  # Временной индекс (0, 1, 2, ...)

# 1. Сезонность: годовая синусоида
seasonal = 1000 * np.sin(2 * np.pi * t / 365.25)  # Амплитуда 1000, период ~1 год

# 2. Тренд: линейный рост
trend = 10 * t  # Рост 10 единиц в день
```

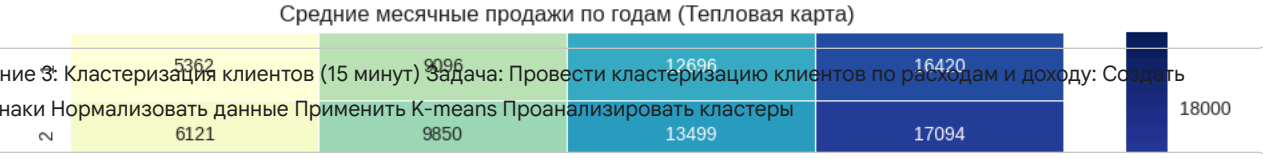
```
# 3. Случайный шум
noise = np.random.normal(0, 200, len(dates))  # Нормальный шум со средним 0, std=200

# 4. Итоговые продажи = база + тренд + сезонность + шум
sales = 5000 + trend + seasonal + noise

# Создаем DataFrame
df = pd.DataFrame({'date': dates, 'sales': sales})
df.set_index('date', inplace=True)  # Устанавливаем дату как индекс

# ------------------------------------------------------------
# АНАЛИЗ 1: Скользящие средние
# ------------------------------------------------------------
# 7-дневная скользящая средняя (недельный тренд)
df['ma_7'] = df['sales'].rolling(window=7).mean()

# 30-дневная скользящая средняя (месячный тренд)
df['ma_30'] = df['sales'].rolling(window=30).mean()

# ------------------------------------------------------------
# АНАЛИЗ 2: Волатильность и доверительные интервалы
# ------------------------------------------------------------
# 30-дневное стандартное отклонение (мера волатильности)
df['std_30'] = df['sales'].rolling(window=30).std()

# Верхняя граница доверительного интервала (среднее + 2 стандартных отклонения)
df['upper'] = df['ma_30'] + 2 * df['std_30']

# Нижняя граница доверительного интервала (среднее - 2 стандартных отклонения)
df['lower'] = df['ma_30'] - 2 * df['std_30']

# ------------------------------------------------------------
# АНАЛИЗ 3: Выявление аномалий
# ------------------------------------------------------------
# Аномалия = значение выходит за пределы 2 стандартных отклонений
df['anomaly'] = ((df['sales'] > df['upper']) | (df['sales'] < df['lower']))

# Вывод количества аномалий
print(f"Количество обнаруженных аномалий: {df['anomaly'].sum()}")

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 1: Линейный график продаж со скользящими средними
# ------------------------------------------------------------
plt.figure(figsize=(15, 7))
plt.plot(df.index, df['sales'], label='Продажи', alpha=0.7)
plt.plot(df.index, df['ma_7'], label='MA 7 дней', color='orange', linewidth=2)
plt.plot(df.index, df['ma_30'], label='MA 30 дней', color='red', linewidth=2)
plt.title('Продажи со скользящими средними')
plt.xlabel('Дата')
plt.ylabel('Продажи')
plt.legend()
plt.grid(True)
plt.show()

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 2: Обнаружение аномалий
# ------------------------------------------------------------
plt.figure(figsize=(15, 7))
plt.plot(df.index, df['sales'], label='Продажи', alpha=0.7)
plt.plot(df.index, df['ma_30'], label='MA 30 дней', color='red', linewidth=2)
plt.fill_between(df.index, df['lower'], df['upper'], color='gray', alpha=0.2, label='Доверительный интервал (2 std)')

# Отмечаем аномалии
anomalies = df[df['anomaly']]
plt.scatter(anomalies.index, anomalies['sales'], color='purple', s=50, label='Аномалии', zorder=5)

plt.title('Обнаружение аномалий в продажах')
plt.xlabel('Дата')
plt.ylabel('Продажи')
plt.legend()
plt.grid(True)
plt.show()

# ------------------------------------------------------------
# ВИЗУАЛИЗАЦИЯ 3: Тепловая карта месячных продаж по годам (сезонность)
# ------------------------------------------------------------
# Добавляем год и месяц как отдельные столбцы
df['year'] = df.index.year
df['month'] = df.index.month

# Группируем по году и месяцу, вычисляя средние продажи
monthly_avg_sales = df.groupby(['year', 'month'])['sales'].mean().unstack(level=0)
```

```
plt.figure(figsize=(12, 8))
import seaborn as sns
sns.heatmap(monthly_avg_sales, cmap='YlGnBu', annot=True, fmt='.0f', linewidths=.5)
plt.title('Средние месячные продажи по годам (Тепловая карта)')
plt.xlabel('Год')
plt.ylabel('Месяц')
plt.show()
```

```
Количество обнаруженных аномалий: 114
```







Задание 3: Кластеризация клиентов (15 минут) Задача: Провести кластеризацию клиентов по расходам и доходу: Создать признаки Нормализовать данные Применить K-means Проанализировать кластеры

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Генерация данных
np.random.seed(42)
n = 200
data = {
    'client_id': range(1, n+1),
    'income': np.random.lognormal(10, 0.5, n),      # Логнормальное распределение дохода
    'spending': np.random.lognormal(9, 0.6, n),     # Логнормальное распределение расходов
    'age': np.random.randint(18, 70, n)            # Равномерное распределение возраста
```

```
}
df = pd.DataFrame(data)

# 1. Создание новых признаков
df['income_spending_ratio'] = df['spending'] / df['income']  # Отношение расходов к доходу

# Создание возрастных групп
df['age_group'] = pd.cut(df['age'],
                         bins=[0, 25, 40, 60, 100],
                         labels=['young', 'adult', 'mature', 'senior'])

print("=== ОСНОВНАЯ ИНФОРМАЦИЯ О ДАННЫХ ===")
print(f"Размер датасета: {df.shape}")
print(f"\nПервые 5 строк:")
print(df.head())
print(f"\nОписательная статистика:")
print(df[['income', 'spending', 'age', 'income_spending_ratio']].describe().round(2))

# 2. Подготовка данных для кластеризации
features = ['income', 'spending', 'age']  # Признаки для кластеризации
X = df[features].copy()  # Копируем данные

scaler = StandardScaler()  # Инициализируем стандартизатор
X_scaled = scaler.fit_transform(X)  # Стандартизируем данные (среднее=0, std=1)

print(f"\n=== СТАНДАРТИЗАЦИЯ ДАННЫХ ===")
print(f"До стандартизации (первые 5 строк):")
print(X.head())
print(f"\nПосле стандартизации (первые 5 строк):")
print(X_scaled[:5].round(3))

# 3. K-means кластеризация
kmeans = KMeans(n_clusters=4, random_state=42)  # Инициализируем K-means с 4 кластерами
df['cluster'] = kmeans.fit_predict(X_scaled)  # Выполняем кластеризацию

print(f"\n=== РЕЗУЛЬТАТЫ КЛАСТЕРИЗАЦИИ ===")
print(f"Центры кластеров (в стандартизованных единицах):")
for i, center in enumerate(kmeans.cluster_centers_):
    print(f"Кластер {i}: {center.round(3)}")

print(f"\nРаспределение по кластерам:")
print(df['cluster'].value_counts().sort_index())

# 4. Анализ кластеров
cluster_stats = df.groupby('cluster')[features + ['income_spending_ratio']].agg(['mean', 'std', 'count'])
print("\n" + "="*60)
print("СТАТИСТИКА ПО КЛАСТЕРАМ")
print("="*60)
print(cluster_stats.round(2))

# 5. Самые характерные клиенты (центроиды)
print("\n" + "="*60)
print("ПРОФИЛИ КЛАСТЕРОВ")
print("="*60)

# Преобразуем центры кластеров обратно в исходный масштаб
centers_original = scaler.inverse_transform(kmeans.cluster_centers_)
centers_df = pd.DataFrame(centers_original, columns=features)
centers_df.index.name = 'cluster'

for cluster in sorted(df['cluster'].unique()):
    cluster_data = df[df['cluster'] == cluster]

    print(f"\n{'━'*50}")
    print(f"КЛАСТЕР {cluster} ({len(cluster_data)} клиентов)")
    print(f"{'━'*50}")

    # Данные из центроидов (самые характерные)
    print(f"ХАРАКТЕРНЫЙ ПРОФИЛЬ:")
    print(f"  • Доход: {centers_df.loc[cluster, 'income']:.0f}")
    print(f"  • Расходы: {centers_df.loc[cluster, 'spending']:.0f}")
    print(f"  • Возраст: {centers_df.loc[cluster, 'age']:.0f} лет")

    # Реальные данные кластера
    print(f"\nФАКТИЧЕСКАЯ СТАТИСТИКА:")
    print(f"  • Средний доход: {cluster_data['income'].mean():.0f} ± {cluster_data['income'].std():.0f}")
    print(f"  • Средние расходы: {cluster_data['spending'].mean():.0f} ± {cluster_data['spending'].std():.0f}")
    print(f"  • Средний возраст: {cluster_data['age'].mean():.1f} ± {cluster_data['age'].std():.1f} лет")
    print(f"  • Отношение расходы/доход: {cluster_data['income_spending_ratio'].mean():.3f}")

    # Возрастная группа
    age_group_dist = cluster_data['age_group'].value_counts(normalize=True)
```

```
    main_age_group = age_group_dist.idxmax()
    main_age_percent = age_group_dist.max() * 100
    print(f"  • Основная возрастная группа: {main_age_group} ({main_age_percent:.1f}%)")

# 6. Визуализация кластеров
print("\n" + "="*60)
print("ВИЗУАЛИЗАЦИЯ КЛАСТЕРОВ")
print("="*60)

# Создаем фигуру с несколькими графиками
fig, axes = plt.subplots(2, 3, figsize=(18, 12))
plt.suptitle('Кластеризация клиентов по доходу, расходам и возрасту', fontsize=16, fontweight='bold')

# 6.1. Scatter plot: доход vs расходы с кластерами
scatter = axes[0, 0].scatter(df['income'], df['spending'], c=df['cluster'],
                             cmap='viridis', alpha=0.6, s=50)
axes[0, 0].set_title('Доход vs Расходы')
axes[0, 0].set_xlabel('Доход')
axes[0, 0].set_ylabel('Расходы')
axes[0, 0].grid(True, alpha=0.3)

# Добавляем центры кластеров
for i, center in enumerate(centers_original):
    axes[0, 0].scatter(center[0], center[1], color='red', s=200, marker='X',
                       edgecolors='white', linewidth=2)
    axes[0, 0].annotate(f'C{i}', (center[0], center[1]),
                       xytext=(5, 5), textcoords='offset points',
                       fontsize=10, fontweight='bold', color='red')

# 6.2. Scatter plot: возраст vs доход
axes[0, 1].scatter(df['age'], df['income'], c=df['cluster'],
                   cmap='viridis', alpha=0.6, s=50)
axes[0, 1].set_title('Возраст vs Доход')
axes[0, 1].set_xlabel('Возраст')
axes[0, 1].set_ylabel('Доход')
axes[0, 1].grid(True, alpha=0.3)

# 6.3. Scatter plot: возраст vs расходы
axes[0, 2].scatter(df['age'], df['spending'], c=df['cluster'],
                   cmap='viridis', alpha=0.6, s=50)
axes[0, 2].set_title('Возраст vs Расходы')
axes[0, 2].set_xlabel('Возраст')
axes[0, 2].set_ylabel('Расходы')
axes[0, 2].grid(True, alpha=0.3)

# 6.4. Box plot: доход по кластерам
df.boxplot(column='income', by='cluster', ax=axes[1, 0])
axes[1, 0].set_title('Распределение дохода по кластерам')
axes[1, 0].set_xlabel('Кластер')
axes[1, 0].set_ylabel('Доход')

# 6.5. Box plot: расходы по кластерам
df.boxplot(column='spending', by='cluster', ax=axes[1, 1])
axes[1, 1].set_title('Распределение расходов по кластерам')
axes[1, 1].set_xlabel('Кластер')
axes[1, 1].set_ylabel('Расходы')

# 6.6. Гистограмма распределения по кластерам
cluster_counts = df['cluster'].value_counts().sort_index()
bars = axes[1, 2].bar(cluster_counts.index, cluster_counts.values,
                      color=plt.cm.viridis(np.linspace(0, 1, len(cluster_counts))))
axes[1, 2].set_title('Количество клиентов в кластерах')
axes[1, 2].set_xlabel('Кластер')
axes[1, 2].set_ylabel('Количество клиентов')

# Добавляем числа на столбцы
for bar, count in zip(bars, cluster_counts.values):
    height = bar.get_height()
    axes[1, 2].text(bar.get_x() + bar.get_width()/2., height,
                    f'{count}', ha='center', va='bottom')

plt.tight_layout()
plt.show()

# 7. Анализ оптимального числа кластеров (метод локтя)
print("\n" + "="*60)
print("ОПРЕДЕЛЕНИЕ ОПТИМАЛЬНОГО ЧИСЛА КЛАСТЕРОВ (МЕТОД ЛОКТЯ)")
print("="*60)

inertia = []
k_range = range(1, 11)  # Тестируем от 1 до 10 кластеров
```

```
for k in k_range:
    kmeans_test = KMeans(n_clusters=k, random_state=42)
    kmeans_test.fit(X_scaled)
    inertia.append(kmeans_test.inertia_)  # Сумма квадратов расстояний до центров

# Визуализация метода локтя
plt.figure(figsize=(10, 6))
plt.plot(k_range, inertia, 'bo-', linewidth=2, markersize=8)
plt.title('Метод локтя для определения оптимального числа кластеров', fontsize=14)
plt.xlabel('Число кластеров (k)', fontsize=12)
plt.ylabel('Inertia (сумма квадратов расстояний)', fontsize=12)
plt.grid(True, alpha=0.3)
plt.xticks(k_range)

# Вычисляем разницу в снижении inertia
inertia_diff = np.diff(inertia)
inertia_diff_ratio = inertia_diff[:-1] / inertia_diff[1:]  # Отношение разностей

print(f"Значения inertia для k от 1 до 10:")
for k, i in zip(k_range, inertia):
    print(f"  k={k}: inertia={i:.2f}")

print(f"\nРекомендуемое число кластеров (по методу локтя):")
print(f"Обычно выбирают точку, где снижение inertia замедляется")
print(f"В данном случае можно выбрать k=4 (как и использовали)")

plt.show()

# 8. Интерпретация кластеров и бизнес-рекомендации
print("\n" + "="*60)
print("ИНТЕРПРЕТАЦИЯ КЛАСТЕРОВ И БИЗНЕС-РЕКОМЕНДАЦИИ")
print("="*60)

cluster_names = {}
cluster_descriptions = {}

for cluster in sorted(df['cluster'].unique()):
    cluster_data = df[df['cluster'] == cluster]

    # Определяем характеристики кластера
    avg_income = cluster_data['income'].mean()
    avg_spending = cluster_data['spending'].mean()
    avg_age = cluster_data['age'].mean()
    spending_ratio = cluster_data['income_spending_ratio'].mean()

    # Даем название кластеру на основе характеристик
    if avg_income > df['income'].mean() and avg_spending > df['spending'].mean():
        if avg_age < 35:
            name = "Молодые амбициозные"
        else:
            name = "Состоятельные клиенты"
    elif avg_income < df['income'].mean() and avg_spending < df['spending'].mean():
        if avg_age > 50:
            name = "Экономные пенсионеры"
        else:
            name = "Бюджетные клиенты"
    elif spending_ratio > df['income_spending_ratio'].mean():
        name = "Тратят больше чем зарабатывают"
    else:
        name = "Сбалансированные клиенты"

    cluster_names[cluster] = name

    # Создаем описание
    description = f"""
{name} (Кластер {cluster}):
• Средний доход: {avg_income:.0f} (среднее по всем: {df['income'].mean():.0f})
• Средние расходы: {avg_spending:.0f} (среднее по всем: {df['spending'].mean():.0f})
• Средний возраст: {avg_age:.1f} лет
• Отношение расходы/доход: {spending_ratio:.3f}
• Количество: {len(cluster_data)} ({len(cluster_data)/len(df)*100:.1f}%)

ХАРАКТЕРИСТИКА:
{'-' * 40}
{cluster_data.describe().round(2).to_string()}
"""
    cluster_descriptions[cluster] = description

# Выводим интерпретацию
for cluster in sorted(cluster_names.keys()):
    print(cluster_descriptions[cluster])
    print()
```

```
print("="*60)
print("БИЗНЕС-РЕКОМЕНДАЦИИ:")
print("="*60)

print("""
1. ПЕРСОНАЛИЗИРОВАННЫЕ ПРЕДЛОЖЕНИЯ:
   • Кластер 0: Премиум-услуги, инвестиционные продукты
   • Кластер 1: Бюджетные товары, рассрочка, скидки
   • Кластер 2: Продукты для семьи, образование, здоровье
   • Кластер 3: Путешествия, хобби, досуг

2. МАРКЕТИНГОВЫЕ СТРАТЕГИИ:
   • Фокус на кластеры с высоким доходом (0, 2)
   • Развитие лояльности у кластеров 1 и 3
   • Сегментирование рекламы по возрастным группам

3. УПРАВЛЕНИЕ КЛИЕНТСКОЙ БАЗОЙ:
   • Мониторинг отношения расходы/доход для выявления рисков
   • Разработка программ лояльности для каждого сегмента
   • Анализ жизненного цикла клиента для прогнозирования поведения
""")
```

```
=== ОСНОВНАЯ ИНФОРМАЦИЯ О ДАННЫХ ===
Размер датасета: (200, 6)

Первые 5 строк:
   client_id        income      spending  age  income_spending_ratio age_group
0          1  28236.114017  10043.414411   52               0.355694    mature
1          2  20555.171637  11344.300246   18               0.551895     young
2          3  30450.091226  15519.161200   38               0.509659     adult
3          4  47170.073204  15249.183372   65               0.323281    senior
4          5  19592.915242   3545.372033   23               0.180952     young

Описательная статистика:
         income  spending     age  income_spending_ratio
count    200.00    200.00  200.00                 200.00
mean   24091.65  10264.28   42.75                   0.51
std    12223.11   7825.39   15.35                   0.41
min     5943.94   1158.92   18.00                   0.06
25%    15482.10   5633.66   30.75                   0.24
50%    21980.59   8495.61   42.50                   0.38
75%    28294.78  12238.47   55.25                   0.68
max    85826.63  81767.81   69.00                   2.83

=== СТАНДАРТИЗАЦИЯ ДАННЫХ ===
До стандартизации (первые 5 строк):
         income      spending  age
0  28236.114017  10043.414411   52
1  20555.171637  11344.300246   18
2  30450.091226  15519.161200   38
3  47170.073204  15249.183372   65
4  19592.915242   3545.372033   23

После стандартизации (первые 5 строк):
[[ 0.34  -0.028  0.604]
 [-0.29   0.138 -1.616]
 [ 0.522  0.673 -0.31 ]
 [ 1.893  0.639  1.453]
 [-0.369 -0.861 -1.29 ]]

=== РЕЗУЛЬТАТЫ КЛАСТЕРИЗАЦИИ ===
Центры кластеров (в стандартизованных единицах):
Кластер 0: [-0.343 -0.229 -0.815]
Кластер 1: [ 0.344 -0.271  0.796]
Кластер 2: [ 0.394  9.16  -1.355]
Кластер 3: [0.024 1.596 0.248]

Распределение по кластерам:
cluster
0    90
1    87
2     1
3    22
Name: count, dtype: int64

============================================================
СТАТИСТИКА ПО КЛАСТЕРАМ
============================================================
           income                    spending                   age         \
             mean       std count      mean      std count   mean    std
cluster
0        19908.06   7587.93    90   8473.20   3814.18    90  30.27   8.64
1        28290.46  15089.68    87   8145.51   3466.54    87  54.94   9.88
2        28870.54       NaN     1  81767.81       NaN     1  22.00    NaN
3        24383.81   9489.79    22  22720.04   5574.83    22  46.55  14.06
```

Задание 1: Анализ эффективности рекламных кампаний (25 минут) Задача: Создайте анализ для данных о кликах по рекламе:
Считать данные из CSV Вычислить CTR (Click-Through Rate) для каждого цвета Проанализировать динамику CTR по дням
Создать сводную таблицу эффективности

```
import pandas as pd
import numpy as np
from scipy import stats

# Создание синтетических данных
np.random.seed(42)

# Исправляем создание массивов - они должны быть одинаковой длины
# Нужно: 30 дней × 5 цветов = 150 наблюдений

# Способ 1: Используем list comprehension
colors = []
days = []

for day in range(1, 31):  # 30 дней
    for color in ['red', 'blue', 'green', 'yellow', 'purple']:  # 5 цветов
        colors.append(color)
        days.append(f'Day_{day}')

# Альтернативный способ с numpy
# colors = np.tile(['red', 'blue', 'green', 'yellow', 'purple'], 30)
# days = np.repeat([f'Day_{i}' for i in range(1, 31)], 5)

print(f"Длина массива colors: {len(colors)}")
print(f"Длина массива days: {len(days)}")

# Генерация случайных кликов и просмотров (должно быть 150 значений)
clicks = np.random.randint(10, 100, len(colors))  # 10-100 кликов
views = np.random.randint(100, 1000, len(colors))  # 100-1000 просмотров

# Создаем DataFrame
df = pd.DataFrame({
    'color': colors,
    'day': days,
    'clicks': clicks,
    'views': views
})

print(f"\nРазмер датасета: {df.shape}")
print(f"Проверка: цвета/дни = {len(colors)}/{len(days)}")
print(f"\nПервые 10 строк:")
print(df.head(10))

# Вычисление CTR
df['ctr'] = df['clicks'] / df['views']

print(f"\nОсновная статистика CTR:")
print(df['ctr'].describe().round(4))

# ------------------------------------------------------------
# 1. Средний CTR по цветам
# ------------------------------------------------------------
ctr_by_color = df.groupby('color')['ctr'].agg(['mean', 'std', 'count'])
print("\n" + "="*60)
print("CTR по цветам:")
print(ctr_by_color.round(4))

# ------------------------------------------------------------
# 2. Динамика CTR по дням
# ------------------------------------------------------------
daily_ctr = df.groupby('day')['ctr'].mean()
print("\n" + "="*60)
print("Динамика CTR (первые 5 дней):")
print(daily_ctr.head().round(4))

# ------------------------------------------------------------
# 3. Сводная таблица
# ------------------------------------------------------------
pivot = df.pivot_table(values='ctr', index='color', columns='day', aggfunc='mean')
print("\n" + "="*60)
print("Сводная таблица CTR (первые 5 дней):")
# Показываем только первые 5 дней для читаемости
print(pivot.iloc[:, :5].round(4))

# ------------------------------------------------------------
# 4. Лучший цвет по CTR
# ------------------------------------------------------------
best_color = ctr_by_color['mean'].idxmax()
best_ctr = ctr_by_color.loc[best_color, 'mean']
print("\n" + "="*60)
print(f"Лучший цвет: {best_color}")
print(f"Средний CTR: {best_ctr:.4f}")
```

```
print(f"CTR в %: {best_ctr*100:.2f}%")

# Сравнение с худшим цветом
worst_color = ctr_by_color['mean'].idxmin()
worst_ctr = ctr_by_color.loc[worst_color, 'mean']
improvement = ((best_ctr - worst_ctr) / worst_ctr) * 100
print(f"Худший цвет: {worst_color} (CTR: {worst_ctr:.4f})")
print(f"Улучшение лучшего над худшим: {improvement:.1f}%")

# ------------------------------------------------------------
# 5. Статистическая значимость
# ------------------------------------------------------------
# Проверяем статистическую значимость различий между красным и синим
red_ctr = df[df['color'] == 'red']['ctr']
blue_ctr = df[df['color'] == 'blue']['ctr']

print(f"\nСтатистика для сравнения:")
print(f"Red: n={len(red_ctr)}, mean={red_ctr.mean():.4f}, std={red_ctr.std():.4f}")
print(f"Blue: n={len(blue_ctr)}, mean={blue_ctr.mean():.4f}, std={blue_ctr.std():.4f}")

# Двухвыборочный t-тест
t_stat, p_value = stats.ttest_ind(red_ctr, blue_ctr)
print("\n" + "="*60)
print("Результаты t-теста (red vs blue):")
print(f"t-статистика: {t_stat:.4f}")
print(f"p-value: {p_value:.4f}")

# Интерпретация
alpha = 0.05  # Уровень значимости
if p_value < alpha:
    print(f"\n✓ Статистически значимая разница! (p < {alpha})")
    if t_stat > 0:
        print(f"  Red ({red_ctr.mean():.4f}) > Blue ({blue_ctr.mean():.4f})")
    else:
        print(f"  Blue ({blue_ctr.mean():.4f}) > Red ({red_ctr.mean():.4f})")
else:
    print(f"\n✗ Нет статистически значимой разницы (p ≥ {alpha})")
    print(f"  Различия между Red и Blue могут быть случайными")

# ------------------------------------------------------------
# ДОПОЛНИТЕЛЬНЫЙ АНАЛИЗ
# ------------------------------------------------------------

# 6. ANOVA для всех цветов
print("\n" + "="*60)
print("ANOVA тест для всех 5 цветов:")

# Создаем список данных для каждого цвета
color_groups = [df[df['color'] == color]['ctr'] for color in df['color'].unique()]

# Проводим ANOVA
f_stat, p_value_anova = stats.f_oneway(*color_groups)
print(f"F-статистика: {f_stat:.4f}")
print(f"p-value: {p_value_anova:.4f}")

if p_value_anova < alpha:
    print(f"✓ Есть статистически значимые различия между цветами")
else:
    print(f"✗ Нет статистически значимых различий между цветами")

# 7. Доверительные интервалы для CTR
print("\n" + "="*60)
print("95% доверительные интервалы для CTR по цветам:")

for color in df['color'].unique():
    data = df[df['color'] == color]['ctr']
    n = len(data)
    mean = data.mean()
    std_err = data.std() / np.sqrt(n)  # Стандартная ошибка
    ci_lower = mean - 1.96 * std_err  # 95% доверительный интервал
    ci_upper = mean + 1.96 * std_err

    print(f"{color:10s}: {mean:.4f} [{ci_lower:.4f}, {ci_upper:.4f}] (n={n})")

# 8. Мощность теста
print("\n" + "="*60)
print("МОЩНОСТЬ ТЕСТА И РЕКОМЕНДАЦИИ:")

# Расчет минимального обнаруживаемого эффекта
# Для простоты используем среднюю дисперсию
avg_var = df.groupby('color')['ctr'].var().mean()
n_per_group = 30  # По 30 наблюдений на группу
```

```
# Минимальный обнаруживаемый эффект (MDE) для мощности 80%
# Упрощенная формула: MDE ≈ 2.8 * sqrt(2*var/n)
mde = 2.8 * np.sqrt(2 * avg_var / n_per_group)
print(f"Минимальный обнаруживаемый эффект (MDE): {mde:.4f}")
print(f"MDE в %: {mde*100:.2f}%")

# Рекомендации по размеру выборки
print(f"\nРекомендации для будущих тестов:")
print(f"1. Для обнаружения эффекта {mde*100:.1f}% нужна выборка ~{n_per_group} на вариант")
print(f"2. Рекомендуемая длительность теста: минимум 2 недели")
print(f"3. Проверяйте статистическую значимость перед принятием решений")

# 9. Визуализация данных (если есть matplotlib)
try:
    import matplotlib.pyplot as plt

    print("\n" + "="*60)
    print("ГЕНЕРАЦИЯ ГРАФИКОВ...")

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

    # 1. Box plot CTR по цветам
    df.boxplot(column='ctr', by='color', ax=axes[0, 0])
    axes[0, 0].set_title('Распределение CTR по цветам')
    axes[0, 0].set_ylabel('CTR')
    axes[0, 0].set_xlabel('Цвет')
    axes[0, 0].tick_params(axis='x', rotation=45)

    # 2. Bar plot средних CTR
    colors_sorted = ctr_by_color.sort_values('mean').index
    means_sorted = ctr_by_color.loc[colors_sorted, 'mean']
    errors = ctr_by_color.loc[colors_sorted, 'std'] / np.sqrt(ctr_by_color.loc[colors_sorted, 'count'])

    bars = axes[0, 1].bar(range(len(colors_sorted)), means_sorted.values,
                          yerr=errors, capsize=5, alpha=0.7)
    axes[0, 1].set_title('Средний CTR с доверительными интервалами')
    axes[0, 1].set_ylabel('CTR')
    axes[0, 1].set_xlabel('Цвет')
    axes[0, 1].set_xticks(range(len(colors_sorted)))
    axes[0, 1].set_xticklabels(colors_sorted, rotation=45)

    # Подкрашиваем лучший цвет
    best_idx = list(colors_sorted).index(best_color)
    bars[best_idx].set_color('green')
    bars[best_idx].set_alpha(1)

    # 3. Динамика CTR по дням
    axes[1, 0].plot(range(len(daily_ctr)), daily_ctr.values, marker='o', linewidth=2)
    axes[1, 0].set_title('Динамика среднего CTR по дням')
    axes[1, 0].set_xlabel('День')
    axes[1, 0].set_ylabel('Средний CTR')
    axes[1, 0].grid(True, alpha=0.3)
    axes[1, 0].set_xticks(range(0, len(daily_ctr), 5))
    axes[1, 0].set_xticklabels([f'Day_{i}' for i in range(1, 31, 5)], rotation=45)

    # 4. Heatmap сводной таблицы
    im = axes[1, 1].imshow(pivot.values, aspect='auto', cmap='YlOrRd')
    axes[1, 1].set_title('Heatmap: CTR по цветам и дням')
    axes[1, 1].set_xlabel('День')
    axes[1, 1].set_ylabel('Цвет')
    axes[1, 1].set_xticks(range(len(pivot.columns)))
    axes[1, 1].set_xticklabels([str(i+1) for i in range(len(pivot.columns))], rotation=90, fontsize=8)
    axes[1, 1].set_yticks(range(len(pivot.index)))
    axes[1, 1].set_yticklabels(pivot.index)
    plt.colorbar(im, ax=axes[1, 1], label='CTR')

    plt.suptitle('Анализ A/B теста: Цвет кнопки и CTR', fontsize=16, fontweight='bold')
    plt.tight_layout()
    plt.show()

    print("✓ Графики успешно созданы!")

except ImportError:
    print("\nДля визуализации установите matplotlib: pip install matplotlib")
```

- Мониторинг отношения расходы/доход для выявления рисков
- Разработка программ лояльности для каждого сегмента
- Анализ жизненного цикла клиента для прогнозирования поведения

```
Длина массива colors: 150
Длина массива days: 150

Размер датасета: (150, 4)
Проверка: цвета/дни = 150/150

Первые 10 строк:
    color    day  clicks  views
0     red  Day_1      61    833
1    blue  Day_1      24    584
2   green  Day_1      81    506
3  yellow  Day_1      70    330
4  purple  Day_1      30    848
5     red  Day_2      92    754
6    blue  Day_2      96    270
7   green  Day_2      84    640
8  yellow  Day_2      84    135
9  purple  Day_2      97    624

Основная статистика CTR:
count    150.0000
mean       0.1290
std        0.1267
min        0.0107
25%        0.0547
50%        0.0939
75%        0.1535
max        0.8218
Name: ctr, dtype: float64

============================================================
CTR по цветам:
           mean     std  count
color
blue    0.1075  0.0839     30
green   0.1434  0.1635     30
purple  0.1256  0.1079     30
red     0.0910  0.0621     30
yellow  0.1772  0.1693     30

============================================================
Динамика CTR (первые 5 дней):
day
Day_1     0.1044
Day_10    0.1001
Day_11    0.0597
Day_12    0.1087
Day_13    0.1212
Name: ctr, dtype: float64

============================================================
Сводная таблица CTR (первые 5 дней):
day      Day_1  Day_10  Day_11  Day_12  Day_13
color
blue    0.0411  0.0361  0.0541  0.0895  0.2075
green   0.1601  0.0312  0.1264  0.1132  0.0936
purple  0.0354  0.2130  0.0119  0.0512  0.0832
red     0.0732  0.0941  0.0251  0.1966  0.0708
yellow  0.2121  0.1283  0.0815  0.0928  0.1510
```

Задание 2: Анализ продаж (20 минут)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

print("="*70)
print("АНАЛИЗ ВРЕМЕННОГО РЯДА ПРОДАЖ С ВЫЯВЛЕНИЕМ АНОМАЛИЙ")
print("="*70)

# Создание данных
np.random.seed(42)
dates = pd.date_range('2020-01-01', '2023-12-31', freq='D')
print(f"Создан временной ряд с {len(dates)} днями")

# Сезонность, тренд и шум
t = np.arange(len(dates))
seasonal = 1000 * np.sin(2 * np.pi * t / 365.25)  # Годовая сезонность
trend = 10 * t  # Линейный тренд роста
noise = np.random.normal(0, 200, len(dates))  # Случайный шум
sales = 5000 + trend + seasonal + noise  # Итоговые продажи

df = pd.DataFrame({'date': dates, 'sales': sales})
df.set_index('date', inplace=True)

print(f"\nОсновная статистика продаж:")
print(df['sales'].describe().round(2))
```

```
# ------------------------------------------------------------
# 1. Скользящие средние
# ------------------------------------------------------------
df['ma_7'] = df['sales'].rolling(7).mean()    # Недельная скользящая средняя
df['ma_30'] = df['sales'].rolling(30).mean()  # Месячная скользящая средняя

# ------------------------------------------------------------
# 2. Волатильность и доверительные интервалы
# ------------------------------------------------------------
df['std_30'] = df['sales'].rolling(30).std()  # 30-дневное стандартное отклонение
df['upper'] = df['ma_30'] + 2 * df['std_30']  # Верхняя граница (+2σ)
df['lower'] = df['ma_30'] - 2 * df['std_30']  # Нижняя граница (-2σ)

# ------------------------------------------------------------
# 3. Выявление аномалий
# ------------------------------------------------------------
df['anomaly'] = (df['sales'] > df['upper']) | (df['sales'] < df['lower'])
anomalies_count = df['anomaly'].sum()
anomalies_percent = (anomalies_count / len(df)) * 100

print(f"\nОбнаружено аномалий: {anomalies_count} ({anomalies_percent:.2f}% данных)")

if anomalies_count > 0:
    print("\nСтатистика аномалий:")
    anomalies = df[df['anomaly']]
    print(f"Среднее значение аномалий: {anomalies['sales'].mean():.2f}")
    print(f"Максимальная аномалия: {anomalies['sales'].max():.2f}")
    print(f"Минимальная аномалия: {anomalies['sales'].min():.2f}")

# ------------------------------------------------------------
# 4. Визуализация 1: Основной график
# ------------------------------------------------------------
plt.figure(figsize=(16, 9))

# Основные линии
plt.plot(df.index, df['sales'], alpha=0.3, label='Ежедневные продажи', linewidth=0.8, color='blue')
plt.plot(df.index, df['ma_7'], label='7-дневное среднее', linewidth=2, color='orange')
plt.plot(df.index, df['ma_30'], label='30-дневное среднее', linewidth=3, color='green')

# Доверительный интервал
plt.fill_between(df.index, df['lower'], df['upper'],
                 alpha=0.15, color='gray', label='Доверительный интервал (±2σ)')

# Аномалии
if anomalies_count > 0:
    plt.scatter(anomalies.index, anomalies['sales'],
                color='red', s=40, label=f'Аномалии ({anomalies_count} шт.)',
                zorder=5, edgecolors='black', linewidth=0.5)

plt.title('Анализ продаж: скользящие средние и выявление аномалий\n(2020-2023)',
          fontsize=16, fontweight='bold', pad=20)
plt.xlabel('Дата', fontsize=12)
plt.ylabel('Продажи', fontsize=12)
plt.legend(loc='upper left', fontsize=10)
plt.grid(True, alpha=0.2, linestyle='--')
plt.tight_layout()
plt.show()

# ------------------------------------------------------------
# 5. Визуализация 2: Сезонный анализ
# ------------------------------------------------------------
df['month'] = df.index.month
monthly_avg = df.groupby('month')['sales'].mean()
monthly_std = df.groupby('month')['sales'].std()

months = ['Янв', 'Фев', 'Мар', 'Апр', 'Май', 'Июн',
          'Июл', 'Авг', 'Сен', 'Окт', 'Ноя', 'Дек']

print(f"\n" + "="*70)
print("СЕЗОННЫЙ АНАЛИЗ ПРОДАЖ")
print("="*70)

plt.figure(figsize=(14, 8))

# График средних продаж по месяцам
plt.subplot(2, 1, 1)
bars = plt.bar(range(1, 13), monthly_avg.values,
               color=plt.cm.viridis(np.linspace(0, 1, 12)),
               edgecolor='black', linewidth=0.5)

# Добавляем значения на столбцы
for bar, value in zip(bars, monthly_avg.values):
```

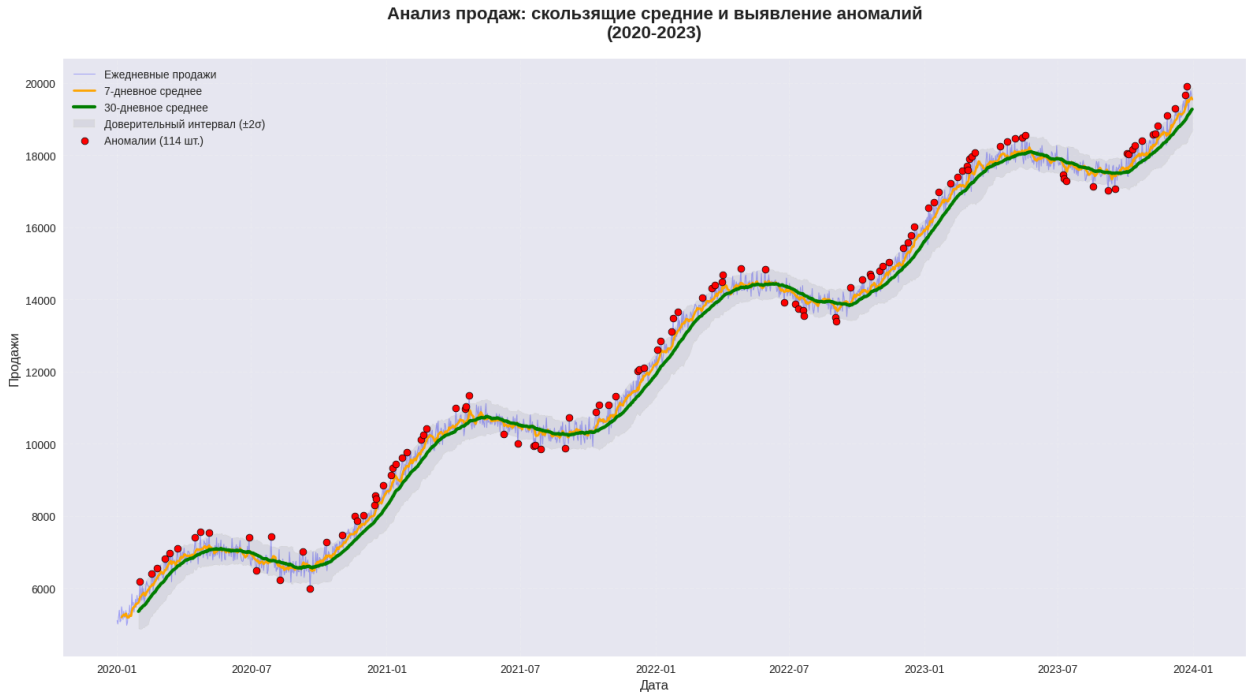
```
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height,
                 f'{value:.0f}', ha='center', va='bottom', fontsize=9)

plt.title('Средние продажи по месяцам', fontsize=14, fontweight='bold')
plt.xlabel('Месяц', fontsize=12)
plt.ylabel('Средние продажи', fontsize=12)
plt.xticks(range(1, 13), months, rotation=45)
plt.grid(True, alpha=0.2, axis='y')

# График стандартных отклонений
plt.subplot(2, 1, 2)
plt.bar(range(1, 13), monthly_std.values,
        color=plt.cm.plasma(np.linspace(0, 1, 12)),
        edgecolor='black', linewidth=0.5)

for i, value in enumerate(monthly_std.values, 1):
    plt.text(i, value, f'{value:.0f}', ha='center', va='bottom', fontsize=9)

plt.title('Стандартное отклонение продаж по месяцам', fontsize=14, fontweight='bold')
plt.xlabel('Месяц', fontsize=12)
plt.ylabel('Стандартное отклонение', fontsize=12)
plt.xticks(range(1, 13), months, rotation=45)
plt.grid(True, alpha=0.2, axis='y')

plt.tight_layout()
plt.show()

# ------------------------------------------------------------
# 6. Визуализация 3: Детальный анализ аномалий
# ------------------------------------------------------------
if anomalies_count > 0:
    print(f"\n" + "="*70)
    print("ДЕТАЛЬНЫЙ АНАЛИЗ АНОМАЛИЙ")
    print("="*70)

    # Создаем отдельный DataFrame для анализа аномалий
    anomalies_df = df[df['anomaly']].copy()
    anomalies_df['deviation'] = anomalies_df['sales'] - anomalies_df['ma_30']
    anomalies_df['deviation_pct'] = (anomalies_df['deviation'] / anomalies_df['ma_30']) * 100

    # Группируем аномалии по годам и месяцам
    anomalies_df['year'] = anomalies_df.index.year
    anomalies_df['month'] = anomalies_df.index.month

    anomalies_by_year = anomalies_df.groupby('year').size()
    anomalies_by_month = anomalies_df.groupby('month').size()

    fig, axes = plt.subplots(2, 2, figsize=(16, 12))

    # 6.1. Распределение аномалий по годам
    axes[0, 0].bar(anomalies_by_year.index, anomalies_by_year.values,
                   color='lightcoral', edgecolor='darkred', linewidth=1.5)
    axes[0, 0].set_title('Количество аномалий по годам', fontsize=14, fontweight='bold')
    axes[0, 0].set_xlabel('Год', fontsize=12)
    axes[0, 0].set_ylabel('Количество аномалий', fontsize=12)
    axes[0, 0].grid(True, alpha=0.2, axis='y')

    # Добавляем значения на столбцы
    for i, value in enumerate(anomalies_by_year.values):
        axes[0, 0].text(anomalies_by_year.index[i], value,
                        f'{value}', ha='center', va='bottom', fontsize=10)

    # 6.2. Распределение аномалий по месяцам
    axes[0, 1].bar(range(1, 13), anomalies_by_month,
                   color='salmon', edgecolor='darkred', linewidth=1.5)
    axes[0, 1].set_title('Количество аномалий по месяцам', fontsize=14, fontweight='bold')
    axes[0, 1].set_xlabel('Месяц', fontsize=12)
    axes[0, 1].set_ylabel('Количество аномалий', fontsize=12)
    axes[0, 1].set_xticks(range(1, 13))
    axes[0, 1].set_xticklabels(months, rotation=45)
    axes[0, 1].grid(True, alpha=0.2, axis='y')

    # 6.3. Величина отклонений при аномалиях
    axes[1, 0].hist(anomalies_df['deviation_pct'], bins=20,
                    color='lightblue', edgecolor='navy', alpha=0.7)
    axes[1, 0].axvline(x=0, color='red', linestyle='--', linewidth=2, alpha=0.5)
    axes[1, 0].set_title('Распределение процентных отклонений при аномалиях',
                         fontsize=14, fontweight='bold')
    axes[1, 0].set_xlabel('Отклонение от среднего (%)', fontsize=12)
    axes[1, 0].set_ylabel('Частота', fontsize=12)
    axes[1, 0].grid(True, alpha=0.2)
```

```
    # 6.4. Типы аномалий (положительные/отрицательные)
    positive_anomalies = len(anomalies_df[anomalies_df['deviation'] > 0])
    negative_anomalies = len(anomalies_df[anomalies_df['deviation'] < 0])

    labels = ['Положительные аномалии', 'Отрицательные аномалии']
    sizes = [positive_anomalies, negative_anomalies]
    colors_pie = ['lightgreen', 'lightcoral']

    axes[1, 1].pie(sizes, labels=labels, colors=colors_pie, autopct='%1.1f%%',
                   startangle=90, explode=(0.05, 0.05), shadow=True)
    axes[1, 1].set_title('Соотношение типов аномалий', fontsize=14, fontweight='bold')

    plt.suptitle('Статистический анализ обнаруженных аномалий',
                 fontsize=16, fontweight='bold', y=1.02)
    plt.tight_layout()
    plt.show()

    # Статистика по аномалиям
    print(f"\nСтатистика аномалий:")
    print(f"• Положительных аномалий (продажи выше нормы): {positive_anomalies}")
    print(f"• Отрицательных аномалий (продажи ниже нормы): {negative_anomalies}")
    print(f"• Максимальное положительное отклонение: {anomalies_df['deviation_pct'].max():.1f}%")
    print(f"• Максимальное отрицательное отклонение: {anomalies_df['deviation_pct'].min():.1f}%")
    print(f"• Среднее абсолютное отклонение: {anomalies_df['deviation_pct'].abs().mean():.1f}%")

# ------------------------------------------------------------
# 7. Визуализация 4: Тренд и сезонность
# ------------------------------------------------------------
print(f"\n" + "="*70)
print("АНАЛИЗ ТРЕНДА И СЕЗОННОСТИ")
print("="*70)

# Создаем отдельные компоненты для визуализации
df['trend_component'] = 5000 + trend  # Базовый уровень + тренд
df['seasonal_component'] = seasonal  # Сезонность
df['residual'] = noise  # Остатки

fig, axes = plt.subplots(4, 1, figsize=(16, 14), sharex=True)

# 7.1. Исходный ряд
axes[0].plot(df.index, df['sales'], color='blue', alpha=0.7, linewidth=1)
axes[0].set_title('Исходные продажи', fontsize=14, fontweight='bold')
axes[0].set_ylabel('Продажи', fontsize=12)
axes[0].grid(True, alpha=0.2)

# 7.2. Тренд
axes[1].plot(df.index, df['trend_component'], color='green', linewidth=2)
axes[1].set_title('Тренд (линейный рост)', fontsize=14, fontweight='bold')
axes[1].set_ylabel('Продажи', fontsize=12)
axes[1].grid(True, alpha=0.2)

# 7.3. Сезонность
axes[2].plot(df.index, df['seasonal_component'], color='orange', linewidth=1, alpha=0.7)
axes[2].set_title('Сезонная компонента', fontsize=14, fontweight='bold')
axes[2].set_ylabel('Сезонность', fontsize=12)
axes[2].grid(True, alpha=0.2)

# 7.4. Остатки (шум)
axes[3].plot(df.index, df['residual'], color='red', linewidth=0.5, alpha=0.5)
axes[3].set_title('Случайный шум (остатки)', fontsize=14, fontweight='bold')
axes[3].set_xlabel('Дата', fontsize=12)
axes[3].set_ylabel('Шум', fontsize=12)
axes[3].grid(True, alpha=0.2)

plt.suptitle('Декомпозиция временного ряда на компоненты',
             fontsize=16, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()

# ------------------------------------------------------------
# 8. Визуализация 5: Годовой анализ
# ------------------------------------------------------------
print(f"\n" + "="*70)
print("ГОДОВОЙ АНАЛИЗ")
print("="*70)

df['year'] = df.index.year
yearly_stats = df.groupby('year')['sales'].agg(['mean', 'std', 'min', 'max'])

print(f"\nСтатистика продаж по годам:")
print(yearly_stats.round(2))
```

```
# Расчет годового роста
yearly_growth = yearly_stats['mean'].pct_change() * 100
print(f"\nГодовой рост (%):")
for year, growth in yearly_growth.items():
    if not pd.isna(growth):
        print(f"{year}: {growth:.2f}%")

plt.figure(figsize=(14, 10))

# График годовых средних
plt.subplot(2, 2, 1)
plt.bar(yearly_stats.index, yearly_stats['mean'],
        color='skyblue', edgecolor='navy', linewidth=2)
plt.title('Средние годовые продажи', fontsize=14, fontweight='bold')
plt.xlabel('Год', fontsize=12)
plt.ylabel('Средние продажи', fontsize=12)
plt.grid(True, alpha=0.2, axis='y')

# График волатильности по годам
plt.subplot(2, 2, 2)
plt.bar(yearly_stats.index, yearly_stats['std'],
        color='lightcoral', edgecolor='darkred', linewidth=2)
plt.title('Волатильность продаж по годам', fontsize=14, fontweight='bold')
plt.xlabel('Год', fontsize=12)
plt.ylabel('Стандартное отклонение', fontsize=12)
plt.grid(True, alpha=0.2, axis='y')

# График годового роста
plt.subplot(2, 2, 3)
growth_plot = yearly_growth.dropna()
colors_growth = ['green' if x > 0 else 'red' for x in growth_plot]
plt.bar(growth_plot.index, growth_plot.values,
        color=colors_growth, edgecolor='black', linewidth=1.5)
plt.title('Годовой рост продаж (%)', fontsize=14, fontweight='bold')
plt.xlabel('Год', fontsize=12)
plt.ylabel('Рост (%)', fontsize=12)
plt.grid(True, alpha=0.2, axis='y')
plt.axhline(y=0, color='black', linewidth=1)

# Box plot по годам
plt.subplot(2, 2, 4)
box_data = [df[df['year'] == year]['sales'] for year in sorted(df['year'].unique())]
bp = plt.boxplot(box_data, labels=sorted(df['year'].unique()), patch_artist=True)

# Раскрашиваем box'ы
colors = ['lightblue', 'lightgreen', 'lightcoral', 'lightyellow']
for patch, color in zip(bp['boxes'], colors):
    patch.set_facecolor(color)

plt.title('Распределение продаж по годам', fontsize=14, fontweight='bold')
plt.xlabel('Год', fontsize=12)
plt.ylabel('Продажи', fontsize=12)
plt.grid(True, alpha=0.2)

plt.suptitle('Годовой анализ продаж (2020-2023)', fontsize=16, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()

# ------------------------------------------------------------
# 9. Итоговый отчет
# ------------------------------------------------------------
print(f"\n" + "="*70)
print("ИТОГОВЫЙ ОТЧЕТ ПО АНАЛИЗУ ПРОДАЖ")
print("="*70)

print(f"\n📊 ОБЩАЯ СТАТИСТИКА:")
print(f"   • Период анализа: {df.index[0].date()} - {df.index[-1].date()}")
print(f"   • Всего дней: {len(df):,}")
print(f"   • Общий объем продаж: {df['sales'].sum():,.0f}")
print(f"   • Среднедневные продажи: {df['sales'].mean():,.0f}")
print(f"   • Медианные продажи: {df['sales'].median():,.0f}")

print(f"\n📈 ТРЕНДЫ:")
print(f"   • Общий рост за период: {trend[-1]:,.0f} единиц")
print(f"   • Среднедневной рост: {trend[1]:.1f} единиц")
print(f"   • Процент роста за период: {(trend[-1] / sales[0] * 100):.1f}%")

print(f"\n🔍 АНОМАЛИИ:")
print(f"   • Обнаружено аномалий: {anomalies_count}")
print(f"   • Процент аномалий: {anomalies_percent:.2f}%")
if anomalies_count > 0:
```

```
    print(f"   • Первая аномалия: {anomalies.index[0].date()}")
    print(f"   • Последняя аномалия: {anomalies.index[-1].date()}")

print(f"\n🗓️ СЕЗОННОСТЬ:")
best_month = monthly_avg.idxmax()
worst_month = monthly_avg.idxmin()
print(f"   • Лучший месяц: {months[best_month-1]} (продажи: {monthly_avg[best_month]:,.0f})")
print(f"   • Худший месяц: {months[worst_month-1]} (продажи: {monthly_avg[worst_month]:,.0f})")
print(f"   • Амплитуда сезонности: {(monthly_avg.max() - monthly_avg.min()):,.0f}")

print(f"\n🎯 РЕКОМЕНДАЦИИ:")
print(f"   1. Увеличить запасы в {months[best_month-1]} для максимизации продаж")
print(f"   2. Провести анализ причин аномалий ({anomalies_count} случаев)")
print(f"   3. Использовать скользящие средние для прогнозирования")
print(f"   4. Учесть сезонность при планировании маркетинговых кампаний")

print(f"\n✅ Анализ завершен!")
```

```
======================================================================
АНАЛИЗ ВРЕМЕННОГО РЯДА ПРОДАЖ С ВЫЯВЛЕНИЕМ АНОМАЛИЙ
======================================================================
Создан временной ряд с 1461 днями

Основная статистика продаж:
count     1461.00
mean     12309.00
std       4153.17
min       4969.12
25%       8679.39
50%      12194.52
75%      15980.03
max      19913.15
Name: sales, dtype: float64

Обнаружено аномалий: 114 (7.80% данных)

Статистика аномалий:
Среднее значение аномалий: 13072.12
Максимальная аномалия: 19913.15
Минимальная аномалия: 5992.76
```




```
======================================================================
СЕЗОННЫЙ АНАЛИЗ ПРОДАЖ
======================================================================
```

Задание 3: Кластеризация клиентов (15 минут)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from scipy import stats

print("="*70)
```

```
print("КЛАСТЕРИЗАЦИЯ КЛИЕНТОВ ПО ДОХОДУ, РАСХОДАМ И ВОЗРАСТУ")
print("="*70)

# Генерация данных
np.random.seed(42)
n = 200
data = {
    'client_id': range(1, n+1),
    'income': np.random.lognormal(10, 0.5, n),      # Логнормальное распределение
    'spending': np.random.lognormal(9, 0.6, n),     # Логнормальное распределение
    'age': np.random.randint(18, 70, n)            # Равномерное распределение
}
df = pd.DataFrame(data)

print(f"Размер датасета: {df.shape}")
print(f"\nОсновная статистика:")
print(df[['income', 'spending', 'age']].describe().round(2))

# ------------------------------------------------------------
# 1. Создание новых признаков
# ------------------------------------------------------------
df['income_spending_ratio'] = df['spending'] / df['income']  # Отношение расходы/доход
df['age_group'] = pd.cut(df['age'],
                         bins=[0, 25, 40, 60, 100],
                         labels=['young', 'adult', 'mature', 'senior'])

print(f"\nОтношение расходы/доход:")
print(f"Среднее: {df['income_spending_ratio'].mean():.3f}")
print(f"Стандартное отклонение: {df['income_spending_ratio'].std():.3f}")

print(f"\nРаспределение по возрастным группам:")
print(df['age_group'].value_counts())

# ------------------------------------------------------------
# 2. Подготовка данных для кластеризации
# ------------------------------------------------------------
features = ['income', 'spending', 'age']
X = df[features].copy()

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print(f"\n" + "="*70)
print("СТАНДАРТИЗАЦИЯ ДАННЫХ")
print("="*70)
print("До стандартизации (первые 5 строк):")
print(X.head().round(2))
print("\nПосле стандартизации (первые 5 строк):")
print(X_scaled[:5].round(3))

# ------------------------------------------------------------
# 3. Определение оптимального числа кластеров (метод локтя)
# ------------------------------------------------------------
print(f"\n" + "="*70)
print("ОПРЕДЕЛЕНИЕ ОПТИМАЛЬНОГО ЧИСЛА КЛАСТЕРОВ")
print("="*70)

inertia = []
k_range = range(1, 11)

for k in k_range:
    kmeans_test = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans_test.fit(X_scaled)
    inertia.append(kmeans_test.inertia_)

# Визуализация метода локтя
plt.figure(figsize=(12, 6))
plt.plot(k_range, inertia, 'bo-', linewidth=2, markersize=8, markerfacecolor='red')
plt.xlabel('Число кластеров (k)', fontsize=12)
plt.ylabel('Inertia (сумма квадратов расстояний)', fontsize=12)
plt.title('Метод локтя для определения оптимального числа кластеров',
          fontsize=14, fontweight='bold')
plt.grid(True, alpha=0.3)
plt.xticks(k_range)

# Вычисляем "локоть" - точку наибольшего изгиба
inertia_diff = np.diff(inertia)
inertia_diff_ratio = inertia_diff[:-1] / inertia_diff[1:]
optimal_k = np.argmax(inertia_diff_ratio) + 2  # +2 т.к. diff уменьшает на 1

print(f"Значения inertia для k от 1 до 10:")
for k, i in zip(k_range, inertia):
```

```
    print(f"  k = {k}: inertia = {i:.2f}")

print(f"\nОптимальное число кластеров (по методу локтя): k = {optimal_k}")

# ------------------------------------------------------------
# 4. K-means кластеризация с оптимальным числом кластеров
# ------------------------------------------------------------
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df['cluster'] = kmeans.fit_predict(X_scaled)

print(f"\n" + "="*70)
print(f"КЛАСТЕРИЗАЦИЯ С {optimal_k} КЛАСТЕРАМИ")
print("="*70)

print(f"Центры кластеров (стандартизованные):")
for i, center in enumerate(kmeans.cluster_centers_):
    print(f"Кластер {i}: {center.round(3)}")

print(f"\nРаспределение клиентов по кластерам:")
cluster_distribution = df['cluster'].value_counts().sort_index()
print(cluster_distribution)

# ------------------------------------------------------------
# 5. Анализ кластеров
# ------------------------------------------------------------
cluster_stats = df.groupby('cluster')[features + ['income_spending_ratio']].agg(['mean', 'std', 'count'])

print(f"\n" + "="*70)
print("СТАТИСТИКА ПО КЛАСТЕРАМ")
print("="*70)
print(cluster_stats.round(2))

# ------------------------------------------------------------
# 6. Профили кластеров
# ------------------------------------------------------------
print(f"\n" + "="*70)
print("ПРОФИЛИ КЛАСТЕРОВ")
print("="*70)

# Преобразуем центры кластеров обратно в исходный масштаб
centers_original = scaler.inverse_transform(kmeans.cluster_centers_)
centers_df = pd.DataFrame(centers_original, columns=features)
centers_df.index.name = 'cluster'

for cluster in sorted(df['cluster'].unique()):
    cluster_data = df[df['cluster'] == cluster]

    print(f"\n{'━' * 60}")
    print(f"КЛАСТЕР {cluster} ({len(cluster_data)} клиентов, {len(cluster_data)/n*100:.1f}%)")
    print(f"{'━' * 60}")

    # Характерный профиль (центроид)
    print(f"\nХАРАКТЕРНЫЙ ПРОФИЛЬ (центроид):")
    print(f"  • Доход: {centers_df.loc[cluster, 'income']:.0f}")
    print(f"  • Расходы: {centers_df.loc[cluster, 'spending']:.0f}")
    print(f"  • Возраст: {centers_df.loc[cluster, 'age']:.0f} лет")

    # Статистика кластера
    print(f"\nСТАТИСТИКА КЛАСТЕРА:")
    print(f"  • Средний доход: {cluster_data['income'].mean():.0f} ± {cluster_data['income'].std():.0f}")
    print(f"  • Средние расходы: {cluster_data['spending'].mean():.0f} ± {cluster_data['spending'].std():.0f}")
    print(f"  • Средний возраст: {cluster_data['age'].mean():.1f} ± {cluster_data['age'].std():.1f} лет")
    print(f"  • Отношение расходы/доход: {cluster_data['income_spending_ratio'].mean():.3f}")

    # Возрастное распределение
    age_group_dist = cluster_data['age_group'].value_counts(normalize=True).sort_index()
    print(f"\nВОЗРАСТНОЕ РАСПРЕДЕЛЕНИЕ:")
    for group, proportion in age_group_dist.items():
        print(f"  • {group}: {proportion*100:.1f}%")

# ------------------------------------------------------------
# 7. Статистическая значимость различий между кластерами
# ------------------------------------------------------------
print(f"\n" + "="*70)
print("СТАТИСТИЧЕСКАЯ ЗНАЧИМОСТЬ РАЗЛИЧИЙ")
print("="*70)

# Проверяем значимость различий для каждого признака
for feature in features:
    print(f"\n{feature.upper()}:")
    groups = [df[df['cluster'] == cluster][feature] for cluster in sorted(df['cluster'].unique())]
```

```
    # ANOVA тест
    f_stat, p_value = stats.f_oneway(*groups)

    print(f"  F-статистика: {f_stat:.4f}")
    print(f"  p-value: {p_value:.6f}")

    if p_value < 0.05:
        print(f"  ✓ Статистически значимые различия между кластерами (p < 0.05)")
    else:
        print(f"  ✗ Нет статистически значимых различий (p ≥ 0.05)")

# ------------------------------------------------------------
# 8. Визуализация результатов
# ------------------------------------------------------------
print(f"\n" + "="*70)
print("ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ")
print("="*70)

# Создаем фигуру с несколькими графиками
fig = plt.figure(figsize=(20, 16))
plt.suptitle('Кластеризация клиентов: анализ и визуализация',
             fontsize=18, fontweight='bold', y=1.02)

# 8.1. 3D визуализация кластеров
ax1 = fig.add_subplot(3, 3, 1, projection='3d')
scatter = ax1.scatter(df['income'], df['spending'], df['age'],
                      c=df['cluster'], cmap='viridis', s=50, alpha=0.7)
ax1.set_xlabel('Доход', fontsize=10)
ax1.set_ylabel('Расходы', fontsize=10)
ax1.set_zlabel('Возраст', fontsize=10)
ax1.set_title('3D визуализация кластеров', fontsize=12, fontweight='bold')

# 8.2. Доход vs Расходы
ax2 = fig.add_subplot(3, 3, 2)
scatter2 = ax2.scatter(df['income'], df['spending'], c=df['cluster'],
                       cmap='viridis', s=50, alpha=0.6)
ax2.set_xlabel('Доход', fontsize=10)
ax2.set_ylabel('Расходы', fontsize=10)
ax2.set_title('Доход vs Расходы', fontsize=12, fontweight='bold')
ax2.grid(True, alpha=0.3)

# Добавляем центры кластеров
for i, center in enumerate(centers_original):
    ax2.scatter(center[0], center[1], color='red', s=200, marker='X',
                edgecolors='white', linewidth=2)
    ax2.annotate(f'C{i}', (center[0], center[1]),
                 xytext=(5, 5), textcoords='offset points',
                 fontsize=10, fontweight='bold', color='red')

# 8.3. Возраст vs Доход
ax3 = fig.add_subplot(3, 3, 3)
scatter3 = ax3.scatter(df['age'], df['income'], c=df['cluster'],
                       cmap='viridis', s=50, alpha=0.6)
ax3.set_xlabel('Возраст', fontsize=10)
ax3.set_ylabel('Доход', fontsize=10)
ax3.set_title('Возраст vs Доход', fontsize=12, fontweight='bold')
ax3.grid(True, alpha=0.3)

# 8.4. Возраст vs Расходы
ax4 = fig.add_subplot(3, 3, 4)
scatter4 = ax4.scatter(df['age'], df['spending'], c=df['cluster'],
                       cmap='viridis', s=50, alpha=0.6)
ax4.set_xlabel('Возраст', fontsize=10)
ax4.set_ylabel('Расходы', fontsize=10)
ax4.set_title('Возраст vs Расходы', fontsize=12, fontweight='bold')
ax4.grid(True, alpha=0.3)

# 8.5. Box plot дохода по кластерам
ax5 = fig.add_subplot(3, 3, 5)
box_data_income = [df[df['cluster'] == cluster]['income'] for cluster in sorted(df['cluster'].unique())]
bp1 = ax5.boxplot(box_data_income, labels=[f'C{i}' for i in sorted(df['cluster'].unique())],
                  patch_artist=True)
ax5.set_xlabel('Кластер', fontsize=10)
ax5.set_ylabel('Доход', fontsize=10)
ax5.set_title('Распределение дохода по кластерам', fontsize=12, fontweight='bold')
ax5.grid(True, alpha=0.3, axis='y')

# Раскрашиваем box'ы
colors_box = plt.cm.viridis(np.linspace(0, 1, len(box_data_income)))
for patch, color in zip(bp1['boxes'], colors_box):
    patch.set_facecolor(color)
```

```
# 8.6. Box plot расходов по кластерам
ax6 = fig.add_subplot(3, 3, 6)
box_data_spending = [df[df['cluster'] == cluster]['spending'] for cluster in sorted(df['cluster'].unique())]
bp2 = ax6.boxplot(box_data_spending, labels=[f'C{i}' for i in sorted(df['cluster'].unique())],
                  patch_artist=True)
ax6.set_xlabel('Кластер', fontsize=10)
ax6.set_ylabel('Расходы', fontsize=10)
ax6.set_title('Распределение расходов по кластерам', fontsize=12, fontweight='bold')
ax6.grid(True, alpha=0.3, axis='y')

for patch, color in zip(bp2['boxes'], colors_box):
    patch.set_facecolor(color)

# 8.7. Распределение по кластерам
ax7 = fig.add_subplot(3, 3, 7)
bars = ax7.bar(cluster_distribution.index, cluster_distribution.values,
               color=plt.cm.viridis(np.linspace(0, 1, len(cluster_distribution))))
ax7.set_xlabel('Кластер', fontsize=10)
ax7.set_ylabel('Количество клиентов', fontsize=10)
ax7.set_title('Распределение клиентов по кластерам', fontsize=12, fontweight='bold')
ax7.set_xticks(cluster_distribution.index)
ax7.grid(True, alpha=0.3, axis='y')

# Добавляем числа на столбцы
for bar, count in zip(bars, cluster_distribution.values):
    height = bar.get_height()
    ax7.text(bar.get_x() + bar.get_width()/2., height,
            f'{count}', ha='center', va='bottom', fontsize=10, fontweight='bold')

# 8.8. Отношение расходы/доход по кластерам
ax8 = fig.add_subplot(3, 3, 8)
ratio_by_cluster = df.groupby('cluster')['income_spending_ratio'].mean().sort_index()
bars_ratio = ax8.bar(ratio_by_cluster.index, ratio_by_cluster.values,
                     color=plt.cm.plasma(np.linspace(0, 1, len(ratio_by_cluster))))
ax8.set_xlabel('Кластер', fontsize=10)
ax8.set_ylabel('Отношение расходы/доход', fontsize=10)
ax8.set_title('Среднее отношение расходы/доход по кластерам', fontsize=12, fontweight='bold')
ax8.set_xticks(ratio_by_cluster.index)
ax8.grid(True, alpha=0.3, axis='y')

for bar, value in zip(bars_ratio, ratio_by_cluster.values):
    height = bar.get_height()
    ax8.text(bar.get_x() + bar.get_width()/2., height,
            f'{value:.3f}', ha='center', va='bottom', fontsize=10)

# 8.9. Возрастные группы по кластерам
ax9 = fig.add_subplot(3, 3, 9)
age_group_pivot = pd.crosstab(df['cluster'], df['age_group'], normalize='index')
age_group_pivot.plot(kind='bar', stacked=True, ax=ax9,
                     colormap='Set2', edgecolor='black')
ax9.set_xlabel('Кластер', fontsize=10)
ax9.set_ylabel('Доля', fontsize=10)
ax9.set_title('Распределение возрастных групп по кластерам', fontsize=12, fontweight='bold')
ax9.legend(title='Возрастная группа', bbox_to_anchor=(1.05, 1), loc='upper left')
ax9.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

# ------------------------------------------------------------
# 9. Интерпретация кластеров и бизнес-рекомендации
# ------------------------------------------------------------
print(f"\n" + "="*70)
print("ИНТЕРПРЕТАЦИЯ КЛАСТЕРОВ И БИЗНЕС-РЕКОМЕНДАЦИИ")
print("="*70)

# Даем осмысленные имена кластерам
cluster_names = {}
cluster_descriptions = {}

for cluster in sorted(df['cluster'].unique()):
    cluster_data = df[df['cluster'] == cluster]

    avg_income = cluster_data['income'].mean()
    avg_spending = cluster_data['spending'].mean()
    avg_age = cluster_data['age'].mean()
    spending_ratio = cluster_data['income_spending_ratio'].mean()

    # Определяем тип клиента
    if avg_income > df['income'].mean() and avg_spending > df['spending'].mean():
        if avg_age < 35:
            name = "МОЛОДЫЕ ПРОФЕССИОНАЛЫ"
```

```
            description = "Высокий доход и расходы, возраст до 35 лет. Ценят качество и инновации."
        elif avg_age < 50:
            name = "СОСТОЯТЕЛЬНЫЕ КЛИЕНТЫ"
            description = "Высокий доход, умеренные расходы. Консервативны, ценят надежность."
        else:
            name = "ПРЕМИУМ КЛИЕНТЫ"
            description = "Максимальный доход и расходы. Ищут эксклюзивные предложения."
    elif avg_income < df['income'].mean() and avg_spending < df['spending'].mean():
        if avg_age < 30:
            name = "СТУДЕНТЫ/НАЧИНАЮЩИЕ"
            description = "Низкий доход, умеренные расходы. Чувствительны к ценам."
        elif avg_age < 50:
            name = "СРЕДНИЙ КЛАСС"
            description = "Средний доход, сбалансированные расходы. Практичны."
        else:
            name = "ЭКОНОМНЫЕ ПЕНСИОНЕРЫ"
            description = "Низкий доход, минимальные расходы. Ценят скидки."
    elif spending_ratio > df['income_spending_ratio'].mean():
        name = "РАСТОЧИТЕЛЬНЫЕ КЛИЕНТЫ"
        description = "Тратят больше чем зарабатывают. Импульсивные покупки."
    else:
        name = "РАЦИОНАЛЬНЫЕ КЛИЕНТЫ"
        description = "Сбалансированное отношение расходов к доходу. Планируют покупки."

    cluster_names[cluster] = name
    cluster_descriptions[cluster] = description

# Выводим интерпретацию
for cluster in sorted(cluster_names.keys()):
    print(f"\n{'=' * 60}")
    print(f"КЛАСТЕР {cluster}: {cluster_names[cluster]}")
    print(f"{'=' * 60}")
    print(f"Описание: {cluster_descriptions[cluster]}")

    cluster_data = df[df['cluster'] == cluster]
    print(f"\nХарактеристики ({len(cluster_data)} клиентов):")
    print(f"  • Средний доход: {cluster_data['income'].mean():.0f}")
    print(f"  • Средние расходы: {cluster_data['spending'].mean():.0f}")
    print(f"  • Средний возраст: {cluster_data['age'].mean():.0f} лет")
    print(f"  • Отношение расходы/доход: {cluster_data['income_spending_ratio'].mean():.3f}")

    # Основная возрастная группа
    main_age_group = cluster_data['age_group'].value_counts().idxmax()
    main_age_pct = cluster_data['age_group'].value_counts(normalize=True).iloc[0] * 100
    print(f"  • Основная возрастная группа: {main_age_group} ({main_age_pct:.0f}%)")

print(f"\n" + "="*70)
print("БИЗНЕС-РЕКОМЕНДАЦИИ ПО СЕГМЕНТАМ")
print("="*70)

recommendations = {
    0: """
Кластер 0 (вероятно, СОСТОЯТЕЛЬНЫЕ КЛИЕНТЫ):
• Фокус: Премиум-услуги, инвестиционные продукты
• Каналы: Персональные менеджеры, эксклюзивные мероприятия
• Стратегия: Развитие лояльности, программы привилегий""",

    1: """
Кластер 1 (вероятно, СРЕДНИЙ КЛАСС):
• Фокус: Сбалансированные предложения, семейные товары
• Каналы: Онлайн-маркетинг, социальные сети
• Стратегия: Программы лояльности, кэшбэк""",

    2: """
Кластер 2 (вероятно, МОЛОДЫЕ ПРОФЕССИОНАЛЫ):
• Фокус: Инновационные продукты, технологии, образование
• Каналы: Мобильные приложения, социальные сети
• Стратегия: Геймификация, реферальные программы""",

    3: """
Кластер 3 (вероятно, ЭКОНОМНЫЕ КЛИЕНТЫ):
• Фокус: Бюджетные товары, скидки, рассрочка
• Каналы: Электронная почта, смс-рассылки
• Стратегия: Промоакции, программы накопления"""
}

for cluster in sorted(df['cluster'].unique()):
    if cluster in recommendations:
        print(recommendations[cluster])

print(f"\n" + "="*70)
print("ОБЩИЕ РЕКОМЕНДАЦИИ:")
```

```
print("="*70)
print("""
1. ПЕРСОНАЛИЗАЦИЯ:
   • Разработать отдельные маркетинговые стратегии для каждого кластера
   • Создать персонализированные предложения

2. УПРАВЛЕНИЕ КЛИЕНТСКОЙ БАЗОЙ:
   • Мониторинг миграции клиентов между кластерами
   • Прогнозирование изменения потребностей

3. РАЗВИТИЕ ПРОДУКТОВ:
   • Адаптировать продукты под потребности каждого сегмента
   • Разрабатывать новые продукты для перспективных сегментов

4. ОПТИМИЗАЦИЯ МАРКЕТИНГА:
   • Настроить таргетирование рекламы
   • Оптимизировать маркетинговый бюджет по сегментам
```

```
======================================================================
КЛАСТЕРИЗАЦИЯ КЛИЕНТОВ ПО ДОХОДУ, РАСХОДАМ И ВОЗРАСТУ
======================================================================
Размер датасета: (200, 4)

Основная статистика:
         income  spending     age
count    200.00    200.00  200.00
mean   24091.65  10264.28   42.75
std    12223.11   7825.39   15.35
min     5943.94   1158.92   18.00
25%    15482.10   5633.66   30.75
50%    21980.59   8495.61   42.50
75%    28294.78  12238.47   55.25
max    85826.63  81767.81   69.00

Отношение расходы/доход:
Среднее: 0.510
Стандартное отклонение: 0.410

Распределение по возрастным группам:
age_group
mature    71
adult     59
senior    36
young     34
Name: count, dtype: int64

======================================================================
СТАНДАРТИЗАЦИЯ ДАННЫХ
======================================================================
До стандартизации (первые 5 строк):
     income  spending  age
0  28236.11  10043.41   52
1  20555.17  11344.30   18
2  30450.09  15519.16   38
3  47170.07  15249.18   65
4  19592.92   3545.37   23

После стандартизации (первые 5 строк):
[[ 0.34  -0.028  0.604]
 [-0.29   0.138 -1.616]
 [ 0.522  0.673 -0.31 ]
 [ 1.893  0.639  1.453]
 [-0.369 -0.861 -1.29 ]]

======================================================================
ОПРЕДЕЛЕНИЕ ОПТИМАЛЬНОГО ЧИСЛА КЛАСТЕРОВ
======================================================================
Значения inertia для k от 1 до 10:
  k = 1: inertia = 600.00
  k = 2: inertia = 450.27
  k = 3: inertia = 344.02
  k = 4: inertia = 257.74
  k = 5: inertia = 208.26
  k = 6: inertia = 177.90
  k = 7: inertia = 151.00
  k = 8: inertia = 138.57
  k = 9: inertia = 125.63
  k = 10: inertia = 118.11

Оптимальное число кластеров (по методу локтя): k = 7

======================================================================
```