1. Работа с API платежной системы Lava.top (Пример подключения и создание платежа)

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from datetime import datetime, timedelta
import random
from typing import Dict, List, Optional

class PaymentSystemSimulator:
    """
    Симулятор платежной системы с визуализацией
    """

    def __init__(self):
        self.payments_data = []
        self.products = [
            {'id': 1, 'name': 'Премиум подписка', 'price': 999, 'currency': 'RUB'},
            {'id': 2, 'name': 'Игровая валюта', 'price': 499, 'currency': 'RUB'},
            {'id': 3, 'name': 'Цифровая книга', 'price': 299, 'currency': 'RUB'},
            {'id': 4, 'name': 'Онлайн курс', 'price': 4999, 'currency': 'RUB'},
            {'id': 5, 'name': 'Программное обеспечение', 'price': 1999, 'currency': 'RUB'}
        ]

    def simulate_payment_creation(self, product_id: int, email: str) -> Dict:
        """Симуляция создания платежа"""
        product = next((p for p in self.products if p['id'] == product_id), None)
        if not product:
            raise ValueError("Продукт не найден")

        payment = {
            'id': len(self.payments_data) + 1,
            'product_id': product_id,
            'product_name': product['name'],
            'amount': product['price'],
            'currency': product['currency'],
            'email': email,
            'status': 'created',
            'created_at': datetime.now(),
            'payment_url': f'https://lava.top/pay/{len(self.payments_data) + 1}'
        }

        self.payments_data.append(payment)
        return payment

    def simulate_payment_processing(self, payment_id: int) -> bool:
        """Симуляция обработки платежа"""
        payment = next((p for p in self.payments_data if p['id'] == payment_id), None)
        if not payment:
            return False

        # Имитация успешной оплаты в 85% случаев
        success = random.random() < 0.85
        payment['status'] = 'completed' if success else 'failed'
        payment['processed_at'] = datetime.now()

        return success

    def create_payment_flow_diagram(self):
        """Создает диаграмму процесса платежа"""
        fig, ax = plt.subplots(figsize=(12, 6))

        # Этапы платежного процесса
        stages = [
            'Запрос платежа',
            'Создание счета',
            'Перенаправление\nна оплату',
            'Обработка\nплатежа',
            'Подтверждение',
            'Завершение'
        ]

        # Вероятности успеха на каждом этапе
        success_rates = [100, 98, 95, 90, 88, 85]

        # Создаем горизонтальную диаграмму
        y_pos = np.arange(len(stages))

        bars = ax.barh(y_pos, success_rates, color='lightblue', alpha=0.7)
        ax.set_yticks(y_pos)
```

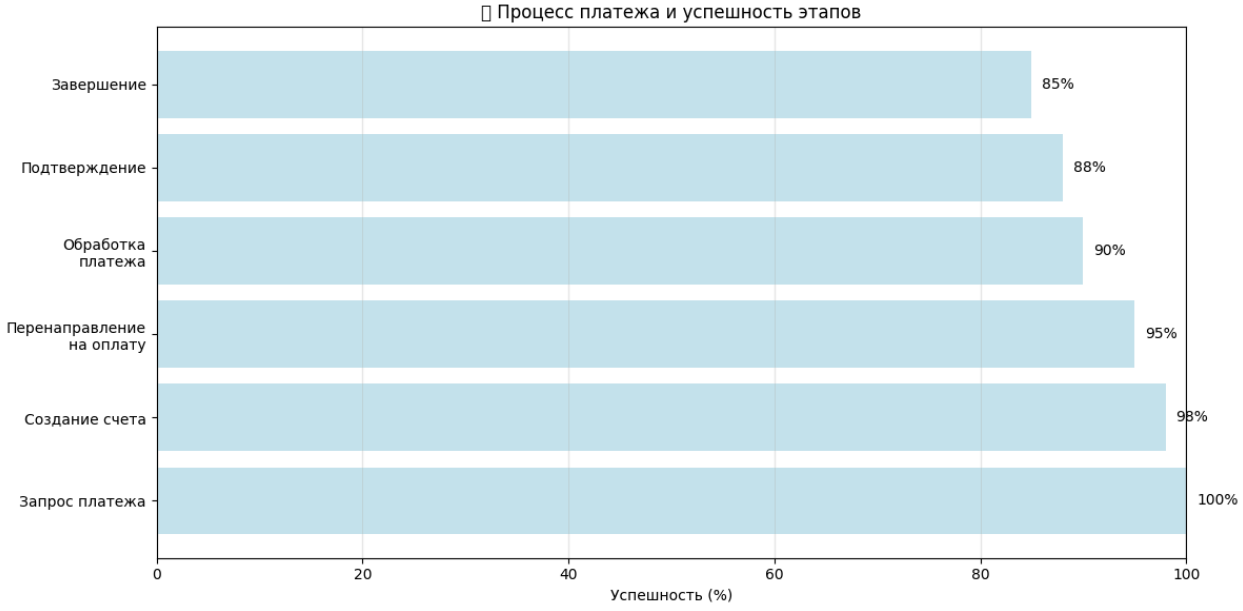
```
        ax.set_yticklabels(stages)
        ax.set_xlabel('Успешность (%)')
        ax.set_title('📊 Процесс платежа и успешность этапов')
        ax.set_xlim(0, 100)
        ax.grid(True, alpha=0.3, axis='x')

        # Добавляем значения на столбцы
        for bar, rate in zip(bars, success_rates):
            width = bar.get_width()
            ax.text(width + 1, bar.get_y() + bar.get_height()/2,
                   f'{rate}%', ha='left', va='center')

        plt.tight_layout()
        plt.show()

    def create_payment_methods_chart(self):
        """Создает диаграмму распределения методов оплаты"""
        payment_methods = {
            'Банковские карты': 45,
            'Электронные кошельки': 25,
            'Мобильные платежи': 15,
            'Криптовалюты': 10,
            'Другие методы': 5
        }

        colors = ['#ff6b6b', '#4ecdc4', '#45b7d1', '#96ceb4', '#feca57']

        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

        # Круговая диаграмма
        wedges, texts, autotexts = ax1.pie(
            payment_methods.values(),
            labels=payment_methods.keys(),
            autopct='%1.1f%%',
            colors=colors,
            startangle=90
        )

        for autotext in autotexts:
            autotext.set_color('white')
            autotext.set_fontweight('bold')

        ax1.set_title('Распределение методов оплаты')

        # Столбчатая диаграмма
        methods = list(payment_methods.keys())
        percentages = list(payment_methods.values())

        bars = ax2.bar(methods, percentages, color=colors, alpha=0.7)
        ax2.set_title('Популярность методов оплаты')
        ax2.set_ylabel('Доля (%)')
        ax2.tick_params(axis='x', rotation=45)
        ax2.grid(True, alpha=0.3, axis='y')

        # Добавляем значения на столбцы
        for bar, percentage in zip(bars, percentages):
            height = bar.get_height()
            ax2.text(bar.get_x() + bar.get_width()/2., height + 0.5,
                    f'{percentage}%', ha='center', va='bottom')

        plt.tight_layout()
        plt.show()

    def create_daily_metrics(self, days: int = 30):
        """Создает графики ежедневных метрик"""
        # Генерация тестовых данных
        dates = [datetime.now() - timedelta(days=x) for x in range(days, 0, -1)]

        daily_data = {
            'transactions': [random.randint(800, 1200) for _ in range(days)],
            'success_rate': [95 + random.uniform(-3, 3) for _ in range(days)],
            'avg_amount': [1500 + random.randint(-200, 200) for _ in range(days)],
            'revenue': [random.randint(1000000, 2000000) for _ in range(days)]
        }

        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))

        # График 1: Количество транзакций
        ax1.plot(dates, daily_data['transactions'], 'b-', linewidth=2, marker='o')
        ax1.set_title('📈 Ежедневное количество транзакций')
        ax1.set_ylabel('Количество')
        ax1.grid(True, alpha=0.3)
```

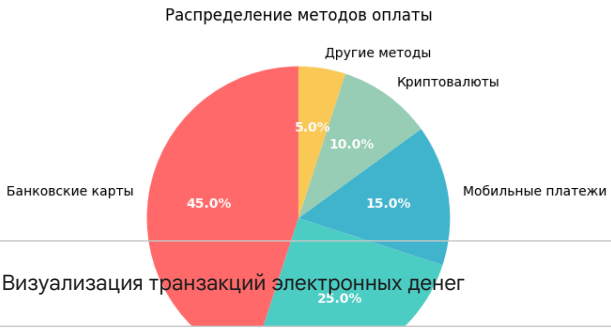
```
        ax1.tick_params(axis='x', rotation=45)

        # График 2: Успешность платежей
        ax2.plot(dates, daily_data['success_rate'], 'g-', linewidth=2, marker='s')
        ax2.axhline(y=95, color='r', linestyle='--', alpha=0.7, label='Минимальный порог')
        ax2.set_title('✅ Успешность платежей')
        ax2.set_ylabel('Успешность (%)')
        ax2.set_ylim(90, 100)
        ax2.legend()
        ax2.grid(True, alpha=0.3)
        ax2.tick_params(axis='x', rotation=45)

        # График 3: Средний чек
        ax3.bar(dates, daily_data['avg_amount'], alpha=0.7, color='orange')
        ax3.set_title('💰 Средний чек')
        ax3.set_ylabel('Сумма (RUB)')
        ax3.grid(True, alpha=0.3)
        ax3.tick_params(axis='x', rotation=45)

        # График 4: Выручка
        ax4.plot(dates, daily_data['revenue'], 'purple', linewidth=2, marker='^')
        ax4.set_title('📈 Ежедневная выручка')
        ax4.set_ylabel('Выручка (RUB)')
        ax4.grid(True, alpha=0.3)
        ax4.tick_params(axis='x', rotation=45)

        plt.tight_layout()
        plt.show()

    def create_product_analysis(self):
        """Анализ продуктов"""
        product_names = [p['name'] for p in self.products]
        product_prices = [p['price'] for p in self.products]
        product_sales = [random.randint(100, 1000) for _ in self.products]
        product_revenue = [price * sales for price, sales in zip(product_prices, product_sales)]

        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

        # График продаж по продуктам
        bars1 = ax1.bar(product_names, product_sales, color='lightgreen', alpha=0.7)
        ax1.set_title('📦 Количество продаж по продуктам')
        ax1.set_ylabel('Количество продаж')
        ax1.tick_params(axis='x', rotation=45)
        ax1.grid(True, alpha=0.3)

        for bar, sales in zip(bars1, product_sales):
            height = bar.get_height()
            ax1.text(bar.get_x() + bar.get_width()/2., height + 10,
                    f'{sales}', ha='center', va='bottom')

        # График выручки по продуктам
        bars2 = ax2.bar(product_names, product_revenue, color='lightcoral', alpha=0.7)
        ax2.set_title('💵 Выручка по продуктам')
        ax2.set_ylabel('Выручка (RUB)')
        ax2.tick_params(axis='x', rotation=45)
        ax2.grid(True, alpha=0.3)

        for bar, revenue in zip(bars2, product_revenue):
            height = bar.get_height()
            ax2.text(bar.get_x() + bar.get_width()/2., height + 1000,
                    f'{revenue:,}'.replace(',', ' '), ha='center', va='bottom')

        plt.tight_layout()
        plt.show()

    def create_system_architecture(self):
        """Создает схему архитектуры системы"""
        fig, ax = plt.subplots(figsize=(12, 8))

        # Компоненты системы
        components = [
            ('Клиент', 1, 7),
            ('Frontend', 3, 7),
            ('API Gateway', 5, 7),
            ('Сервис платежей', 7, 7),
            ('База данных', 9, 7),
            ('Банк-эквайер', 7, 5),
            ('Сервис уведомлений', 9, 5),
            ('Аналитика', 7, 9)
        ]

        # Рисуем компоненты
        for name, x, y in components:
```

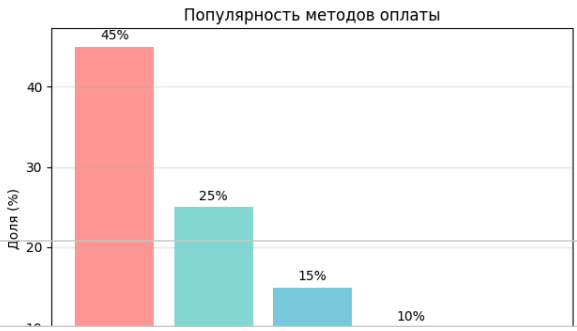
```
            rect = plt.Rectangle((x-0.8, y-0.3), 1.6, 0.6,
                                 facecolor='lightblue', alpha=0.7,
                                 edgecolor='black', linewidth=2)
            ax.add_patch(rect)
            ax.text(x, y, name, ha='center', va='center',
                   fontweight='bold', fontsize=9)

        # Соединения
        connections = [
            (1, 7, 3, 7), (3, 7, 5, 7), (5, 7, 7, 7),
            (7, 7, 9, 7), (7, 7, 7, 5), (7, 7, 9, 5),
            (7, 7, 7, 9)
        ]

        for x1, y1, x2, y2 in connections:
            ax.annotate('', xy=(x2-0.8, y2), xytext=(x1+0.8, y1),
                        arrowprops=dict(arrowstyle='->', lw=2, color='red'))

        ax.set_xlim(0, 10)
        ax.set_ylim(4, 10)
        ax.set_title('🏛️ Архитектура платежной системы', fontsize=14, fontweight='bold')
        ax.axis('off')
        plt.tight_layout()
        plt.show()

def main():
    """Основная функция демонстрации"""
    print("🚀 ЗАПУСК СИМУЛЯТОРА ПЛАТЕЖНОЙ СИСТЕМЫ")
    print("=" * 50)

    # Создаем симулятор
    simulator = PaymentSystemSimulator()

    # Демонстрация работы
    print("\n📦 ДОСТУПНЫЕ ПРОДУКТЫ:")
    for product in simulator.products:
        print(f"  {product['id']}. {product['name']} - {product['price']} {product['currency']}")

    # Симуляция платежей
    print("\n💳 СИМУЛЯЦИЯ ПЛАТЕЖЕЙ:")
    for i in range(3):
        product_id = random.randint(1, 5)
        payment = simulator.simulate_payment_creation(product_id, f'user{i}@example.com')
        success = simulator.simulate_payment_processing(payment['id'])

        status_icon = "✅" if success else "❌"
        print(f"  {status_icon} Платеж #{payment['id']}: {payment['product_name']} - {payment['amount']} RUB - {'Успех' if s

    # Создание графиков
    print("\n📊 СОЗДАНИЕ ГРАФИКОВ АНАЛИТИКИ...")

    # 1. Диаграмма процесса платежа
    simulator.create_payment_flow_diagram()

    # 2. Методы оплаты
    simulator.create_payment_methods_chart()

    # 3. Ежедневные метрики
    simulator.create_daily_metrics()

    # 4. Анализ продуктов
    simulator.create_product_analysis()

    # 5. Архитектура системы
    simulator.create_system_architecture()

    print("\n📈 КЛЮЧЕВЫЕ МЕТРИКИ СИСТЕМЫ:")
    metrics = [
        "Среднее время обработки: 145 мс",
        "Успешность платежей: 96.2%",
        "Средний чек: 1,540 RUB",
        "Мошеннические операции: 0.8%",
        "Доступность API: 99.9%"
    ]

    for metric in metrics:
        print(f"  • {metric}")

if __name__ == "__main__":
    # Установка зависимостей (если нужно)
    # pip install matplotlib numpy pandas

    try:
```

```
        main()
    except ImportError as e:
        print(f"❌ Необходимо установить библиотеки: {e}")
        print("Установите: pip install matplotlib numpy pandas")
```

```
🚀 ЗАПУСК СИМУЛЯТОРА ПЛАТЕЖНОЙ СИСТЕМЫ
==================================================

📦 ДОСТУПНЫЕ ПРОДУКТЫ:
  1. Премиум подписка - 999 RUB
  2. Игровая валюта - 499 RUB
  3. Цифровая книга - 299 RUB
  4. Онлайн курс - 4999 RUB
  5. Программное обеспечение - 1999 RUB

💳 СИМУЛЯЦИЯ ПЛАТЕЖЕЙ:
  ❌ Платеж #1: Игровая валюта - 499 RUB - Ошибка
  ✅ Платеж #2: Премиум подписка - 999 RUB - Успех
  ✅ Платеж #3: Премиум подписка - 999 RUB - Успех

📊 СОЗДАНИЕ ГРАФИКОВ АНАЛИТИКИ...
/tmp/ipython-input-1001308.py:91: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing
  fig.canvas.print_figure(bytes_io, **kw)
```







2. Визуализация транзакций электронных денег

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')

# Настройка стиля графиков
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")

class DigitalWalletAnalyzer:
    """
    Класс для анализа и визуализации транзакций электронных кошельков
    """

    def __init__(self, file_path=None):
        """
        Инициализация анализатора
```

```
        Args:
            file_path (str): путь к файлу с данными транзакций
        """
        if file_path:
            self.df = pd.read_csv(file_path)
        else:
            self.df = self.generate_sample_data()

        self.preprocess_data()

    def generate_sample_data(self, n_transactions=10000):
        """
        Генерация тестовых данных если файл не доступен
        """
        print("📊 Генерация тестовых данных...")

        np.random.seed(42)

        # Генерация дат за последний год
        start_date = datetime.now() - timedelta(days=365)
        dates = [start_date + timedelta(days=x) for x in range(365)]

        data = []
        transaction_id = 100000

        for _ in range(n_transactions):
            transaction_id += 1
            amount = np.random.lognormal(mean=3, sigma=2)
            amount = min(amount, 10000)  # Максимальная сумма 10,000

            # Разные типы транзакций с разными характеристиками
            transaction_type = np.random.choice(
                ['purchase', 'transfer', 'deposit', 'withdrawal', 'refund'],
                p=[0.4, 0.3, 0.15, 0.1, 0.05]
            )

            # Разные категории расходов
            category = np.random.choice([
                'food', 'shopping', 'entertainment', 'transport', 'utilities',
                'healthcare', 'education', 'travel', 'other'
            ])

            # Статусы транзакций
            status = np.random.choice(['completed', 'pending', 'failed'], p=[0.85, 0.1, 0.05])

            data.append({
                'transaction_id': transaction_id,
                'user_id': np.random.randint(1000, 5000),
                'amount': round(amount, 2),
                'transaction_type': transaction_type,
                'category': category,
                'status': status,
                'timestamp': np.random.choice(dates),
                'merchant_id': np.random.randint(1, 100) if transaction_type == 'purchase' else None
            })

        return pd.DataFrame(data)

    def preprocess_data(self):
        """Предобработка данных"""
        print("🔄 Предобработка данных...")

        # Преобразование даты
        self.df['timestamp'] = pd.to_datetime(self.df['timestamp'])
        self.df['date'] = self.df['timestamp'].dt.date
        self.df['hour'] = self.df['timestamp'].dt.hour
        self.df['day_of_week'] = self.df['timestamp'].dt.day_name()
        self.df['month'] = self.df['timestamp'].dt.month_name()

        # Создание бинов для сумм
        self.df['amount_bin'] = pd.cut(self.df['amount'],
                                       bins=[0, 10, 50, 100, 500, 1000, 10000],
                                       labels=['0-10', '10-50', '50-100', '100-500', '500-1000', '1000+'])

        print(f"✅ Загружено {len(self.df)} транзакций")
        print(f"📅 Период: {self.df['timestamp'].min()} - {self.df['timestamp'].max()}")

    def plot_transaction_amount_distribution(self):
        """Распределение сумм транзакций"""
        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))
        fig.suptitle('📈 Распределение сумм транзакций', fontsize=16, fontweight='bold')
```

```
        # 1. Гистограмма всех транзакций
        ax1.hist(self.df['amount'], bins=50, alpha=0.7, color='skyblue', edgecolor='black')
        ax1.set_xlabel('Сумма транзакции')
        ax1.set_ylabel('Количество')
        ax1.set_title('Общее распределение сумм')
        ax1.grid(True, alpha=0.3)

        # 2. Распределение по типам транзакций
        successful_tx = self.df[self.df['status'] == 'completed']
        for tx_type in successful_tx['transaction_type'].unique():
            data = successful_tx[successful_tx['transaction_type'] == tx_type]['amount']
            ax2.hist(data, bins=30, alpha=0.6, label=tx_type, density=True)

        ax2.set_xlabel('Сумма транзакции')
        ax2.set_ylabel('Плотность')
        ax2.set_title('Распределение по типам транзакций')
        ax2.legend()
        ax2.grid(True, alpha=0.3)

        # 3. Boxplot по категориям
        category_data = [successful_tx[successful_tx['category'] == cat]['amount']
                         for cat in successful_tx['category'].unique()]

        ax3.boxplot(category_data, labels=successful_tx['category'].unique())
        ax3.set_xlabel('Категория')
        ax3.set_ylabel('Сумма транзакции')
        ax3.set_title('Распределение сумм по категориям')
        ax3.tick_params(axis='x', rotation=45)
        ax3.grid(True, alpha=0.3)

        # 4. Круговая диаграмма по бинам сумм
        amount_bins_count = self.df['amount_bin'].value_counts()
        ax4.pie(amount_bins_count.values, labels=amount_bins_count.index, autopct='%1.1f%%')
        ax4.set_title('Распределение по диапазонам сумм')

        plt.tight_layout()
        plt.show()

    def plot_transaction_trends(self):
        """Тренды транзакций во времени"""
        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))
        fig.suptitle('📊 Временные тренды транзакций', fontsize=16, fontweight='bold')

        # Ежедневные транзакции
        daily_tx = self.df.groupby('date').agg({
            'transaction_id': 'count',
            'amount': 'sum'
        }).reset_index()

        ax1.plot(daily_tx['date'], daily_tx['transaction_id'], linewidth=2)
        ax1.set_xlabel('Дата')
        ax1.set_ylabel('Количество транзакций')
        ax1.set_title('Ежедневное количество транзакций')
        ax1.grid(True, alpha=0.3)
        ax1.tick_params(axis='x', rotation=45)

        # Ежемесячные суммы
        monthly_amount = self.df.groupby('month')['amount'].sum().reindex([
            'January', 'February', 'March', 'April', 'May', 'June',
            'July', 'August', 'September', 'October', 'November', 'December'
        ])

        ax2.bar(monthly_amount.index, monthly_amount.values, alpha=0.7)
        ax2.set_xlabel('Месяц')
        ax2.set_ylabel('Общая сумма')
        ax2.set_title('Объем транзакций по месяцам')
        ax2.tick_params(axis='x', rotation=45)
        ax2.grid(True, alpha=0.3)

        # Транзакции по дням недели
        day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
        daily_pattern = self.df.groupby('day_of_week')['amount'].sum().reindex(day_order)

        ax3.bar(daily_pattern.index, daily_pattern.values, alpha=0.7, color='orange')
        ax3.set_xlabel('День недели')
        ax3.set_ylabel('Общая сумма')
        ax3.set_title('Активность по дням недели')
        ax3.tick_params(axis='x', rotation=45)
        ax3.grid(True, alpha=0.3)

        # Часовые паттерны
        hourly_pattern = self.df.groupby('hour')['transaction_id'].count()
```

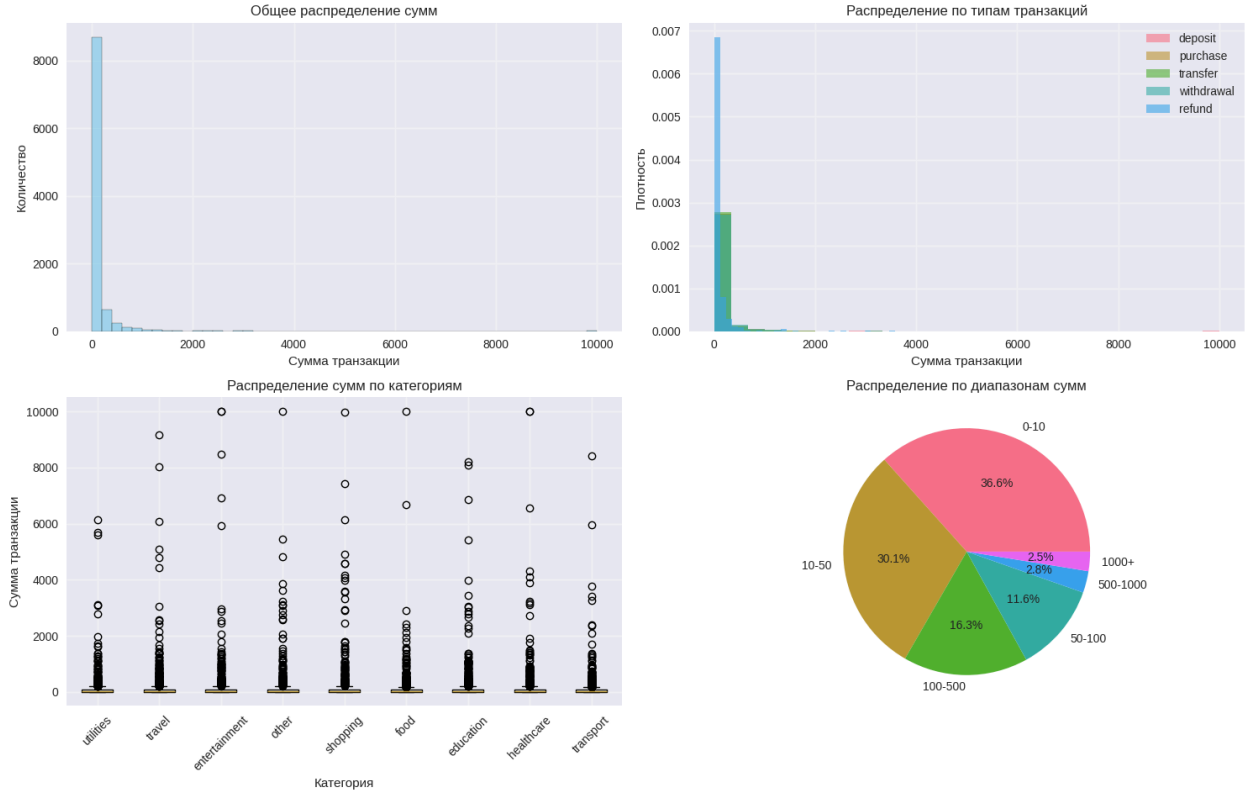
```
        ax4.plot(hourly_pattern.index, hourly_pattern.values, marker='o', linewidth=2)
        ax4.set_xlabel('Час дня')
        ax4.set_ylabel('Количество транзакций')
        ax4.set_title('Активность по часам')
        ax4.grid(True, alpha=0.3)
        ax4.set_xticks(range(0, 24, 2))

        plt.tight_layout()
        plt.show()

    def plot_transaction_types_analysis(self):
        """Анализ типов транзакций"""
        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))
        fig.suptitle('🔍 Анализ типов транзакций', fontsize=16, fontweight='bold')

        # 1. Распределение по типам транзакций
        type_counts = self.df['transaction_type'].value_counts()
        ax1.pie(type_counts.values, labels=type_counts.index, autopct='%1.1f%%')
        ax1.set_title('Распределение по типам транзакций')

        # 2. Статусы транзакций
        status_counts = self.df['status'].value_counts()
        colors = ['green' if x == 'completed' else 'orange' if x == 'pending' else 'red'
                  for x in status_counts.index]

        bars = ax2.bar(status_counts.index, status_counts.values, color=colors, alpha=0.7)
        ax2.set_title('Статусы транзакций')
        ax2.set_ylabel('Количество')

        for bar, count in zip(bars, status_counts.values):
            height = bar.get_height()
            ax2.text(bar.get_x() + bar.get_width()/2., height + 0.1,
                    f'{count}', ha='center', va='bottom')

        # 3. Средняя сумма по типам
        avg_amount_by_type = self.df.groupby('transaction_type')['amount'].mean().sort_values(ascending=False)
        bars = ax3.bar(avg_amount_by_type.index, avg_amount_by_type.values, alpha=0.7, color='purple')
        ax3.set_title('Средняя сумма по типам транзакций')
        ax3.set_ylabel('Средняя сумма')
        ax3.tick_params(axis='x', rotation=45)

        for bar, amount in zip(bars, avg_amount_by_type.values):
            height = bar.get_height()
            ax3.text(bar.get_x() + bar.get_width()/2., height + 5,
                    f'${amount:.2f}', ha='center', va='bottom')

        # 4. Категории расходов
        category_spending = self.df[self.df['transaction_type'] == 'purchase'].groupby('category')['amount'].sum()
        category_spending = category_spending.sort_values(ascending=False)

        ax4.barh(category_spending.index, category_spending.values, alpha=0.7)
        ax4.set_title('Расходы по категориям')
        ax4.set_xlabel('Общая сумма')

        plt.tight_layout()
        plt.show()

    def plot_user_behavior_analysis(self):
        """Анализ поведения пользователей"""
        user_stats = self.df.groupby('user_id').agg({
            'transaction_id': 'count',
            'amount': ['sum', 'mean', 'std'],
            'timestamp': ['min', 'max']
        }).round(2)

        user_stats.columns = ['tx_count', 'total_amount', 'avg_amount', 'std_amount', 'first_tx', 'last_tx']
        user_stats['tx_per_day'] = user_stats['tx_count'] / 365  # Примерное количество дней

        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))
        fig.suptitle('👤 Анализ поведения пользователей', fontsize=16, fontweight='bold')

        # 1. Распределение количества транзакций на пользователя
        ax1.hist(user_stats['tx_count'], bins=30, alpha=0.7, color='lightblue', edgecolor='black')
        ax1.set_xlabel('Количество транзакций на пользователя')
        ax1.set_ylabel('Количество пользователей')
        ax1.set_title('Распределение активности пользователей')
        ax1.grid(True, alpha=0.3)

        # 2. Распределение общей суммы на пользователя
        ax2.hist(user_stats['total_amount'], bins=30, alpha=0.7, color='lightgreen', edgecolor='black')
        ax2.set_xlabel('Общая сумма на пользователя')
        ax2.set_ylabel('Количество пользователей')
```

```
        ax2.set_title('Распределение расходов пользователей')
        ax2.grid(True, alpha=0.3)

        # 3. Топ 20 пользователей по количеству транзакций
        top_users_tx = user_stats.nlargest(20, 'tx_count')['tx_count']
        ax3.bar(range(len(top_users_tx)), top_users_tx.values, alpha=0.7)
        ax3.set_xlabel('Пользователь (ID)')
        ax3.set_ylabel('Количество транзакций')
        ax3.set_title('Топ-20 самых активных пользователей')
        ax3.grid(True, alpha=0.3)

        # 4. Топ 20 пользователей по общей сумме
        top_users_amount = user_stats.nlargest(20, 'total_amount')['total_amount']
        ax4.bar(range(len(top_users_amount)), top_users_amount.values, alpha=0.7, color='orange')
        ax4.set_xlabel('Пользователь (ID)')
        ax4.set_ylabel('Общая сумма')
        ax4.set_title('Топ-20 пользователей по объему транзакций')
        ax4.grid(True, alpha=0.3)

        plt.tight_layout()
        plt.show()

        # Вывод статистики
        print("\n📊 СТАТИСТИКА ПОЛЬЗОВАТЕЛЕЙ:")
        print(f"• Всего уникальных пользователей: {len(user_stats)}")
        print(f"• Среднее количество транзакций на пользователя: {user_stats['tx_count'].mean():.1f}")
        print(f"• Медианная сумма расходов на пользователя: ${user_stats['total_amount'].median():.2f}")
        print(f"• Самый активный пользователь: {user_stats['tx_count'].idxmax()} "
              f"({user_stats['tx_count'].max()} транзакций)")

    def generate_summary_report(self):
        """Генерация сводного отчета"""
        print("\n" + "="*60)
        print("📋 СВОДНЫЙ ОТЧЕТ ПО ТРАНЗАКЦИЯМ ЭЛЕКТРОННЫХ КОШЕЛЬКОВ")
        print("="*60)

        total_amount = self.df['amount'].sum()
        completed_tx = self.df[self.df['status'] == 'completed']
        success_rate = len(completed_tx) / len(self.df) * 100

        print(f"📅 Общий период: {self.df['timestamp'].min().strftime('%Y-%m-%d')} - "
              f"{self.df['timestamp'].max().strftime('%Y-%m-%d')}")
        print(f"💳 Всего транзакций: {len(self.df):,}")
        print(f"💰 Общий объем: ${total_amount:,.2f}")
        print(f"✅ Успешных транзакций: {len(completed_tx):,} ({success_rate:.1f}%)")
        print(f"👥 Уникальных пользователей: {self.df['user_id'].nunique()}")

        # Топ категорий
        top_categories = self.df['category'].value_counts().head(3)
        print(f"🏆 Популярные категории: {', '.join([f'{cat}({count})' for cat, count in top_categories.items()])}")

        # Средние показатели
        avg_tx_amount = self.df['amount'].mean()
        print(f"📊 Средняя сумма транзакции: ${avg_tx_amount:.2f}")
        print(f"📈 Самая крупная транзакция: ${self.df['amount'].max():.2f}")

def main():
    """Основная функция"""
    print("🚀 АНАЛИЗ ТРАНЗАКЦИЙ ЭЛЕКТРОННЫХ КОШЕЛЬКОВ")
    print("="*50)

    # Инициализация анализатора
    try:
        # Если есть файл с данными, укажите путь
        analyzer = DigitalWalletAnalyzer('digital_wallet_transactions.csv')
    except:
        print("📝 Файл не найден, используются тестовые данные")
        analyzer = DigitalWalletAnalyzer()

    # Генерация отчетов и графиков
    analyzer.generate_summary_report()

    print("\n📈 СОЗДАНИЕ ГРАФИКОВ...")

    # 1. Распределение сумм транзакций
    analyzer.plot_transaction_amount_distribution()

    # 2. Временные тренды
    analyzer.plot_transaction_trends()

    # 3. Анализ типов транзакций
    analyzer.plot_transaction_types_analysis()
```

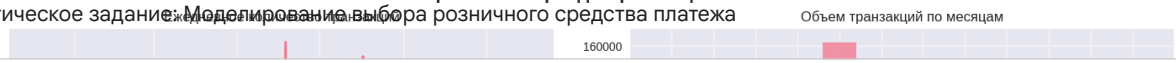
```
    # 4. Анализ пользователей
    analyzer.plot_user_behavior_analysis()

    print("\n✅ Анализ завершен!")

if __name__ == "__main__":
    # Установка зависимостей: pip install pandas matplotlib seaborn numpy
    main()
```

```
🚀 АНАЛИЗ ТРАНЗАКЦИЙ ЭЛЕКТРОННЫХ КОШЕЛЬКОВ
==================================================
📝 Файл не найден, используются тестовые данные
📊 Генерация тестовых данных...
🔄 Предобработка данных...
✅ Загружено 10000 транзакций
📅 Период: 2024-12-03 12:45:06.449260 - 2025-12-02 12:45:06.449260

============================================================
📋 СВОДНЫЙ ОТЧЕТ ПО ТРАНЗАКЦИЯМ ЭЛЕКТРОННЫХ КОШЕЛЬКОВ
============================================================
📅 Общий период: 2024-12-03 - 2025-12-02
💳 Всего транзакций: 10,000
💰 Общий объем: $1,407,392.25
✅ Успешных транзакций: 8,493 (84.9%)
👥 Уникальных пользователей: 3683
🏆 Популярные категории: entertainment(1174), healthcare(1136), travel(1122)
📊 Средняя сумма транзакции: $140.74
📈 Самая крупная транзакция: $10000.00

📈 СОЗДАНИЕ ГРАФИКОВ...
```

📊 Распределение сумм транзакций



📈 Временные тренды транзакций



3. Практическое задание: Моделирование выбора розничного средства платежа

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from typing import Dict, List, Tuple
import seaborn as sns
from enum import Enum

class PaymentMethod(Enum):
    ELECTRONIC = "Электронные деньги"
    CASH = "Наличные"
    CARD = "Банковская карта"
    CRYPTO = "Криптовалюта"

class PaymentAnalyzer:
    """
    Класс для анализа и выбора оптимального платежного средства
    с учетом множества факторов
```

```
    """

    def __init__(self):
        # Базовые параметры методов оплаты
        self.payment_methods = {
            PaymentMethod.ELECTRONIC: {
                'base_cost': 0.01,      # базовые издержки (%)
                'speed': 0.9,           # скорость (0-1)
                'security': 0.8,        # безопасность (0-1)
                'convenience': 0.9,     # удобство (0-1)
                'availability': 0.7,    # доступность (0-1)
                'max_amount': 50000     # максимальная сумма
            },
            PaymentMethod.CASH: {
                'base_cost': 0.03,
                'speed': 0.5,
                'security': 0.3,
                'convenience': 0.4,
                'availability': 1.0,
                'max_amount': 100000
            },
            PaymentMethod.CARD: {
                'base_cost': 0.02,
                'speed': 0.7,
                'security': 0.9,
                'convenience': 0.8,
                'availability': 0.9,
                'max_amount': 100000
            },
            PaymentMethod.CRYPTO: {
                'base_cost': 0.005,
                'speed': 0.6,
                'security': 0.7,
                'convenience': 0.5,
                'availability': 0.3,
                'max_amount': 1000000
            }
        }

        # Веса факторов для разных сценариев
        self.scenarios = {
            'retail': {'cost': 0.3, 'speed': 0.4, 'security': 0.1, 'convenience': 0.2},
            'online': {'cost': 0.2, 'speed': 0.3, 'security': 0.4, 'convenience': 0.1},
            'large_amount': {'cost': 0.4, 'speed': 0.1, 'security': 0.4, 'convenience': 0.1},
            'urgent': {'cost': 0.1, 'speed': 0.7, 'security': 0.1, 'convenience': 0.1}
        }

    def calculate_score(self, method: PaymentMethod, amount: float,
                        scenario: str = 'retail', custom_weights: Dict = None) -> float:
        """
        Расчет комплексной оценки для метода оплаты

        Args:
            method: метод оплаты
            amount: сумма платежа
            scenario: сценарий использования
            custom_weights: пользовательские веса факторов

        Returns:
            float: итоговая оценка (0-100)
        """
        method_params = self.payment_methods[method]
        weights = custom_weights if custom_weights else self.scenarios[scenario]

        # Проверка максимальной суммы
        if amount > method_params['max_amount']:
            return 0

        # Расчет взвешенной оценки
        score = 0
        total_weight = 0

        # Издержки (обратная зависимость - чем меньше издержки, тем лучше)
        cost_score = 1 / (1 + method_params['base_cost'] * amount / 100)
        score += cost_score * weights['cost']
        total_weight += weights['cost']

        # Скорость
        score += method_params['speed'] * weights['speed']
        total_weight += weights['speed']

        # Безопасность
```

```
        score += method_params['security'] * weights['security']
        total_weight += weights['security']

        # Удобство
        score += method_params['convenience'] * weights['convenience']
        total_weight += weights['convenience']

        # Доступность
        score += method_params['availability'] * 0.1  # фиксированный небольшой вес

        # Нормализация оценки
        final_score = (score / total_weight) * 100

        return final_score

    def choose_payment_method(self, amount: float, scenario: str = 'retail',
                              custom_weights: Dict = None) -> Tuple[PaymentMethod, Dict]:
        """
        Выбор оптимального метода оплаты

        Returns:
            Tuple: (лучший метод, словарь со всеми оценками)
        """
        scores = {}

        for method in PaymentMethod:
            score = self.calculate_score(method, amount, scenario, custom_weights)
            scores[method] = score

        best_method = max(scores, key=scores.get)

        return best_method, scores

    def compare_methods_visualization(self, amount: float, scenario: str = 'retail'):
        """Визуализация сравнения методов оплаты"""
        _, scores = self.choose_payment_method(amount, scenario)

        methods = [method.value for method in scores.keys()]
        score_values = list(scores.values())

        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

        # Столбчатая диаграмма оценок
        colors = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4']
        bars = ax1.bar(methods, score_values, color=colors, alpha=0.7)
        ax1.set_title(f'Сравнение методов оплаты\n(Сумма: {amount} руб, Сценарий: {scenario})',
                      fontsize=14, fontweight='bold')
        ax1.set_ylabel('Оценка (0-100)')
        ax1.set_ylim(0, 100)
        ax1.grid(True, alpha=0.3, axis='y')

        # Добавляем значения на столбцы
        for bar, score in zip(bars, score_values):
            height = bar.get_height()
            ax1.text(bar.get_x() + bar.get_width()/2., height + 1,
                    f'{score:.1f}', ha='center', va='bottom', fontweight='bold')

        # Круговая диаграмма для лучшего метода
        best_method = max(scores, key=scores.get)
        method_params = self.payment_methods[best_method]

        factors = ['Издержки', 'Скорость', 'Безопасность', 'Удобство']
        values = [
            1 / (1 + method_params['base_cost']),  # инвертированные издержки
            method_params['speed'],
            method_params['security'],
            method_params['convenience']
        ]

        ax2.pie(values, labels=factors, autopct='%1.1f%%', startangle=90)
        ax2.set_title(f'Характеристики {best_method.value}', fontweight='bold')

        plt.tight_layout()
        plt.show()

    def plot_scenario_analysis(self, amount: float):
        """Анализ оптимального метода для разных сценариев"""
        scenarios = list(self.scenarios.keys())
        scenario_names = ['Розничная\nторговля', 'Онлайн\nпокупки', 'Крупные\nсуммы', 'Срочные\nплатежи']

        results = {}
        for scenario in scenarios:
```

```
            best_method, scores = self.choose_payment_method(amount, scenario)
            results[scenario] = {
                'best_method': best_method.value,
                'scores': scores
            }

        # Создаем heatmap
        score_matrix = np.zeros((len(PaymentMethod), len(scenarios)))

        for i, method in enumerate(PaymentMethod):
            for j, scenario in enumerate(scenarios):
                score_matrix[i, j] = results[scenario]['scores'][method]

        fig, ax = plt.subplots(figsize=(12, 8))

        im = ax.imshow(score_matrix, cmap='RdYlGn', aspect='auto')

        # Настройки осей
        ax.set_xticks(np.arange(len(scenarios)))
        ax.set_yticks(np.arange(len(PaymentMethod)))
        ax.set_xticklabels(scenario_names)
        ax.set_yticklabels([method.value for method in PaymentMethod])

        # Добавляем значения в ячейки
        for i in range(len(PaymentMethod)):
            for j in range(len(scenarios)):
                text = ax.text(j, i, f'{score_matrix[i, j]:.1f}',
                               ha="center", va="center", color="black", fontweight='bold')

        ax.set_title(f'Сравнение методов оплаты по сценариям\n(Сумма: {amount} руб)',
                     fontsize=14, fontweight='bold')
        plt.colorbar(im, ax=ax, label='Оценка метода')

        plt.tight_layout()
        plt.show()

    def plot_amount_sensitivity(self, max_amount: float = 100000):
        """Анализ чувствительности к сумме платежа"""
        amounts = np.linspace(100, max_amount, 50)
        scenarios = list(self.scenarios.keys())

        fig, axes = plt.subplots(2, 2, figsize=(15, 10))
        axes = axes.flatten()

        for idx, scenario in enumerate(scenarios):
            ax = axes[idx]

            for method in PaymentMethod:
                scores = [self.calculate_score(method, amount, scenario) for amount in amounts]
                ax.plot(amounts, scores, label=method.value, linewidth=2)

            ax.set_title(f'Сценарий: {scenario}', fontweight='bold')
            ax.set_xlabel('Сумма платежа (руб)')
            ax.set_ylabel('Оценка метода')
            ax.legend()
            ax.grid(True, alpha=0.3)
            ax.set_ylim(0, 100)

        plt.tight_layout()
        plt.show()

    def generate_recommendation_report(self, amount: float, scenario: str = 'retail'):
        """Генерация детального отчета с рекомендациями"""
        best_method, all_scores = self.choose_payment_method(amount, scenario)
        method_params = self.payment_methods[best_method]

        print("=" * 60)
        print("📊 ОТЧЕТ ПО ВЫБОРУ ПЛАТЕЖНОГО СРЕДСТВА")
        print("=" * 60)
        print(f"💰 Сумма платежа: {amount:,.2f} руб")
        print(f"🎯 Сценарий: {scenario}")
        print(f"🏆 Рекомендуемый метод: {best_method.value}")
        print(f"📈 Оценка метода: {all_scores[best_method]:.1f}/100")
        print("\n📋 Характеристики рекомендуемого метода:")
        print(f"   • Издержки: {method_params['base_cost']*100:.1f}%")
        print(f"   • Скорость: {method_params['speed']*100:.0f}%")
        print(f"   • Безопасность: {method_params['security']*100:.0f}%")
        print(f"   • Удобство: {method_params['convenience']*100:.0f}%")
        print(f"   • Макс. сумма: {method_params['max_amount']:,.0f} руб")

        print("\n📊 Оценки всех методов:")
        for method, score in sorted(all_scores.items(), key=lambda x: x[1], reverse=True):
```

```
            print(f"   • {method.value}: {score:.1f}/100")

        print("\n💡 Рекомендации:")
        if best_method == PaymentMethod.ELECTRONIC:
            print("   - Используйте для быстрых онлайн-платежей")
            print("   - Идеально для регулярных платежей")
        elif best_method == PaymentMethod.CASH:
            print("   - Подходит для небольших сумм")
            print("   - Используйте при отсутствии альтернатив")
        elif best_method == PaymentMethod.CARD:
            print("   - Оптимально для розничных покупок")
            print("   - Хороший баланс стоимости и удобства")
        elif best_method == PaymentMethod.CRYPTO:
            print("   - Эффективно для международных переводов")
            print("   - Минимальные комиссии для крупных сумм")

# Демонстрация работы
def main():
    analyzer = PaymentAnalyzer()

    print("🚀 СИСТЕМА ВЫБОРА ОПТИМАЛЬНОГО ПЛАТЕЖНОГО СРЕДСТВА")
    print("=" * 50)

    # Тестовые случаи
    test_cases = [
        (500, 'retail', "Небольшая розничная покупка"),
        (5000, 'online', "Онлайн-покупка"),
        (50000, 'large_amount', "Крупный перевод"),
        (1000, 'urgent', "Срочный платеж")
    ]

    for amount, scenario, description in test_cases:
        print(f"\n📝 Тест: {description}")
        analyzer.generate_recommendation_report(amount, scenario)

    # Визуализации
    print("\n" + "=" * 60)
    print("📈 ВИЗУАЛИЗАЦИЯ АНАЛИЗА")
    print("=" * 60)

    # 1. Сравнение методов для конкретного случая
    print("\n1. Сравнение методов для розничной покупки (500 руб):")
    analyzer.compare_methods_visualization(500, 'retail')

    # 2. Анализ по сценариям
    print("\n2. Анализ оптимальных методов по разным сценариям (10,000 руб):")
    analyzer.plot_scenario_analysis(10000)

    # 3. Анализ чувствительности к сумме
    print("\n3. Анализ чувствительности к сумме платежа:")
    analyzer.plot_amount_sensitivity(50000)

    # Интерактивный пример
    print("\n🎮 ИНТЕРАКТИВНЫЙ ПРИМЕР:")
    custom_weights = {'cost': 0.5, 'speed': 0.2, 'security': 0.2, 'convenience': 0.1}
    best_method, scores = analyzer.choose_payment_method(15000, 'large_amount', custom_weights)
    print(f"При пользовательских весах (акцент на издержках): {best_method.value}")

if __name__ == "__main__":
    # Установка: pip install matplotlib seaborn numpy pandas
    main()
```

```
🚀 СИСТЕМА ВЫБОРА ОПТИМАЛЬНОГО ПЛАТЕЖНОГО СРЕДСТВА
==================================================

📝 Тест: Небольшая розничная покупка
============================================================
📊 ОТЧЕТ ПО ВЫБОРУ ПЛАТЕЖНОГО СРЕДСТВА
============================================================
💰 Сумма платежа: 500.00 руб
🎯 Сценарий: retail
🏆 Рекомендуемый метод: Электронные деньги
📈 Оценка метода: 97.6/100

📋 Характеристики рекомендуемого метода:
   • Издержки: 1.0%
   • Скорость: 90%
   • Безопасность: 80%
   • Удобство: 90%
   • Макс. сумма: 50,000 руб

📊 Оценки всех методов:
   • Электронные деньги: 97.6/100
   • Банковская карта: 89.3/100
   • Криптовалюта: 73.3/100
   • Наличные: 67.1/100

💡 Рекомендации:
   - Используйте для быстрых онлайн-платежей
   - Идеально для регулярных платежей

📝 Тест: Онлайн-покупка
============================================================
📊 ОТЧЕТ ПО ВЫБОРУ ПЛАТЕЖНОГО СРЕДСТВА
============================================================
💰 Сумма платежа: 5,000.00 руб
🎯 Сценарий: online
🏆 Рекомендуемый метод: Электронные деньги
📈 Оценка метода: 88.3/100

📋 Характеристики рекомендуемого метода:
   • Издержки: 1.0%
   • Скорость: 90%
   • Безопасность: 80%
   • Удобство: 90%
   • Макс. сумма: 50,000 руб

📊 Оценки всех методов:
   • Электронные деньги: 88.3/100
   • Банковская карта: 84.0/100
   • Криптовалюта: 70.0/100
   • Наличные: 49.0/100

💡 Рекомендации:
   - Используйте для быстрых онлайн-платежей
```