

задание 1

```

import asyncio
import aiohttp
import json
from enum import Enum
from typing import Dict, Any, Optional
import hmac
import hashlib
from datetime import datetime

class Currency(Enum):
    RUB = "RUB"
    USD = "USD"
    EUR = "EUR"
    UAH = "UAH"

class PaymentStatus(Enum):
    PENDING = "pending"
    SUCCESS = "success"
    FAILED = "failed"
    EXPIRED = "expired"

class PaymentSystem:
    """Универсальный класс для работы с платежными системами"""

    def __init__(self, api_key: str, secret_key: str, base_url: str):
        self.api_key = api_key
        self.secret_key = secret_key
        self.base_url = base_url
        self.session = None

    async def __aenter__(self):
        self.session = aiohttp.ClientSession()
        return self

    async def __aexit__(self, exc_type, exc_val, exc_tb):
        if self.session:
            await self.session.close()

    def _generate_signature(self, data: Dict) -> str:
        """Генерация подписи для запроса"""
        message = json.dumps(data, sort_keys=True, separators=(',', ':'))
        return hmac.new(
            self.secret_key.encode(),
            message.encode(),
            hashlib.sha256
        ).hexdigest()

    async def create_payment(self, amount: float, currency: Currency,
                           order_id: str, description: str = "") -> Dict[str, Any]:
        """Создание платежа"""
        payload = {
            "amount": amount,
            "currency": currency.value,
            "order_id": order_id,
            "description": description,
            "timestamp": int(datetime.now().timestamp())
        }

        payload["signature"] = self._generate_signature(payload)

        headers = {
            "Content-Type": "application/json",
            "X-API-Key": self.api_key
        }

        async with self.session.post(
            f"{self.base_url}/create-payment",
            json=payload,
            headers=headers
        ) as response:
            result = await response.json()
            return result

    async def check_payment_status(self, payment_id: str) -> Dict[str, Any]:
        """Проверка статуса платежа"""
        payload = {
            "payment_id": payment_id,
            "timestamp": int(datetime.now().timestamp())
        }

```

```

    }

    payload["signature"] = self._generate_signature(payload)

    headers = {
        "X-API-Key": self.api_key
    }

    async with self.session.get(
        f"{self.base_url}/payment-status/{payment_id}",
        headers=headers
    ) as response:
        result = await response.json()
        return result

class MockPaymentSystem:
    """Мок-платежная система для тестирования"""

    def __init__(self):
        self.payments = {}
        self.payment_id_counter = 1

    async def create_payment(self, amount: float, currency: Currency,
                           order_id: str, description: str = "") -> Dict[str, Any]:
        """Создание тестового платежа"""
        payment_id = f"pay_{self.payment_id_counter}"
        self.payment_id_counter += 1

        payment_data = {
            "id": payment_id,
            "amount": amount,
            "currency": currency.value,
            "order_id": order_id,
            "description": description,
            "status": PaymentStatus.PENDING.value,
            "created_at": datetime.now().isoformat(),
            "payment_url": f"https://mock-payment.com/pay/{payment_id}",
            "success_url": f"https://your-site.com/success/{order_id}",
            "fail_url": f"https://your-site.com/fail/{order_id}"
        }

        self.payments[payment_id] = payment_data
        return payment_data

    async def check_payment_status(self, payment_id: str) -> Dict[str, Any]:
        """Проверка статуса платежа (мок)"""
        if payment_id in self.payments:
            # Имитация изменения статуса
            import random
            payment = self.payments[payment_id]

            if payment["status"] == PaymentStatus.PENDING.value:
                # С вероятностью 70% платеж успешен, 30% -失败
                if random.random() < 0.7:
                    payment["status"] = PaymentStatus.SUCCESS.value
                else:
                    payment["status"] = PaymentStatus.FAILED.value

            return payment
        else:
            return {"error": "Payment not found"}

    async def main():
        """Пример использования платежной системы"""

        # Использование мок-системы для демонстрации
        payment_system = MockPaymentSystem()

        try:
            # Создание платежа
            payment = await payment_system.create_payment(
                amount=1000.00,
                currency=Currency.RUB,
                order_id="order_12345",
                description="Пополнение счета"
            )

            print("✅ Платеж создан:")
            print(f"ID: {payment['id']}")
            print(f"Сумма: {payment['amount']} {payment['currency']}")
            print(f"Статус: {payment['status']}")
            print(f"Ссылка для оплаты: {payment['payment_url']}")
        
```

```

# Имитация ожидания оплаты
print("\n⚠ Ожидаем оплату...")
await asyncio.sleep(2)

# Проверка статуса
status = await payment_system.check_payment_status(payment['id'])
print(f"\n📊 Статус платежа: {status['status']}")

if status['status'] == PaymentStatus.SUCCESS.value:
    print("✅ Платеж успешно завершен!")
elif status['status'] == PaymentStatus.FAILED.value:
    print("❌ Платеж не прошел")
else:
    print("⚠ Платеж в обработке")

except Exception as e:
    print(f"❗️ Ошибка: {e}")

# Реализация с вебхуками
class WebhookProcessor:
    """Обработчик вебхуков от платежной системы"""

    def __init__(self, secret_key: str):
        self.secret_key = secret_key

    def verify_webhook(self, payload: Dict, signature: str) -> bool:
        """Проверка подписи вебхука"""
        expected_signature = hmac.new(
            self.secret_key.encode(),
            json.dumps(payload, sort_keys=True, separators=(',', ':')).encode(),
            hashlib.sha256
        ).hexdigest()

        return hmac.compare_digest(expected_signature, signature)

    async def process_webhook(self, payload: Dict, signature: str):
        """Обработка входящего вебхука"""
        if not self.verify_webhook(payload, signature):
            raise ValueError("Invalid webhook signature")

        payment_id = payload.get('payment_id')
        status = payload.get('status')
        amount = payload.get('amount')

        print(f"⌚ Обработка вебхука: payment_id={payment_id}, status={status}")

        # Обработка разных статусов
        if status == PaymentStatus.SUCCESS.value:
            await self._handle_successful_payment(payment_id, amount)
        elif status == PaymentStatus.FAILED.value:
            await self._handle_failed_payment(payment_id)
        elif status == PaymentStatus.EXPIRED.value:
            await self._handle_expired_payment(payment_id)

    async def _handle_successful_payment(self, payment_id: str, amount: float):
        """Обработка успешного платежа"""
        print(f"✅ Платеж {payment_id} на сумму {amount} успешно завершен")
        # Здесь можно обновить статус в БД, начислить средства и т.д.

    async def _handle_failed_payment(self, payment_id: str):
        """Обработка неудачного платежа"""
        print(f"❌ Платеж {payment_id} не прошел")

    async def _handle_expired_payment(self, payment_id: str):
        """Обработка просроченного платежа"""
        print(f"⌚ Платеж {payment_id} просрочен")

# Пример использования вебхуков
async def webhook_example():
    webhook_processor = WebhookProcessor("your_secret_key")

    # Имитация вебхука
    webhook_payload = {
        "payment_id": "pay_123",
        "status": "success",
        "amount": 1000.00,
        "currency": "RUB",
        "timestamp": int(datetime.now().timestamp())
    }

    signature = hmac.new(
        "your_secret_key".encode(),

```

```

        json.dumps(webhook_payload, sort_keys=True, separators=(',', ':')).encode(),
        hashlib.sha256
    ).hexdigest()

    await webhook_processor.process_webhook(webhook_payload, signature)

# Запуск демонстрации платежной системы
print("👉 Запуск демонстрации платежной системы")
await main()

print("\n\n⌚ Демонстрация обработки вебхуков")
await webhook_example()

```

⌚ Запуск демонстрации платежной системы
✓ Платеж создан:
ID: pay_1
Сумма: 1000.0 RUB
Статус: pending
Ссылка для оплаты: https://mock-payment.com/pay/pay_1

🕒 Ожидаем оплату...

📊 Статус платежа: success
✓ Платеж успешно завершен!

⌚ Демонстрация обработки вебхуков
⌚ Обработка вебхука: payment_id=pay_123, status=success
✓ Платеж pay_123 на сумму 1000.0 успешно завершен

задание 2

```

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

# Настройка стиля для графиков
plt.style.use('default')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (10, 6)

print("📊 ВИЗУАЛИЗАЦИЯ ТРАНЗАКЦИЙ ЭЛЕКТРОННЫХ ДЕНЕГ")
print("=" * 50)

# Создание демо-датасета (если файл недоступен)
def create_sample_data():
    """Создает демонстрационный набор данных транзакций"""
    np.random.seed(42)
    n = 3000

    # Генерация реалистичных сумм транзакций
    amounts = np.concatenate([
        np.random.exponential(50, int(n * 0.6)),    # Мелкие транзакции
        np.random.normal(200, 80, int(n * 0.3)),    # Средние транзакции
        np.random.lognormal(5, 1, int(n * 0.1))     # Крупные транзакции
    ])
    amounts = np.abs(amounts)

    data = {
        'transaction_id': range(1, n + 1),
        'user_id': np.random.randint(1000, 1500, n),
        'transaction_amount': amounts,
        'transaction_type': np.random.choice(['purchase', 'transfer', 'withdrawal', 'deposit'], n),
        'merchant_category': np.random.choice(['retail', 'food', 'transport', 'entertainment', 'services'], n),
        'timestamp': pd.date_range('2024-01-01', periods=n, freq='H')
    }
    return pd.DataFrame(data)

# Загрузка данных
try:
    df = pd.read_csv('digital_wallet_transactions.csv')
    print("✓ Данные загружены из файла")
except:
    df = create_sample_data()
    print("✓ Создан демонстрационный dataset")

# Базовая информация
print(f"\n⌚ ОСНОВНАЯ СТАТИСТИКА:")

```

```

print(f"Количество транзакций: {len(df)}")
print(f"Общий объем: ${df['transaction_amount'].sum():,.2f}")
print(f"Средняя сумма: ${df['transaction_amount'].mean():.2f}")
print(f"Медианная сумма: ${df['transaction_amount'].median():.2f}")

# ГРАФИК 1: Распределение сумм транзакций
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.hist(df['transaction_amount'], bins=50, alpha=0.7, color='skyblue', edgecolor='black')
plt.title('Распределение сумм транзакций', fontweight='bold', fontsize=12)
plt.xlabel('Сумма транзакции ($)')
plt.ylabel('Количество')
plt.grid(True, alpha=0.3)
plt.axvline(df['transaction_amount'].mean(), color='red', linestyle='--',
            label=f'Среднее: ${df["transaction_amount"].mean():.2f}')
plt.legend()

# ГРАФИК 2: Распределение в логарифмической шкале
plt.subplot(2, 2, 2)
plt.hist(np.log1p(df['transaction_amount']), bins=40, alpha=0.7, color='lightgreen', edgecolor='darkgreen')
plt.title('Логарифмическое распределение сумм', fontweight='bold', fontsize=12)
plt.xlabel('ln(Сумма транзакции + 1)')
plt.ylabel('Количество')
plt.grid(True, alpha=0.3)

# ГРАФИК 3: Количество транзакций по типам
plt.subplot(2, 2, 3)
transaction_counts = df['transaction_type'].value_counts()
plt.bar(transaction_counts.index, transaction_counts.values, color='lightcoral', alpha=0.7)
plt.title('Количество транзакций по типам', fontweight='bold', fontsize=12)
plt.xlabel('Тип транзакции')
plt.ylabel('Количество')
plt.xticks(rotation=45)
# Добавляем значения на столбцы
for i, v in enumerate(transaction_counts.values):
    plt.text(i, v + 10, str(v), ha='center', fontweight='bold')

# ГРАФИК 4: Средняя сумма по типам транзакций
plt.subplot(2, 2, 4)
type_avg = df.groupby('transaction_type')['transaction_amount'].mean()
plt.bar(type_avg.index, type_avg.values, color='gold', alpha=0.7)
plt.title('Средняя сумма по типам транзакций', fontweight='bold', fontsize=12)
plt.xlabel('Тип транзакции')
plt.ylabel('Средняя сумма ($)')
plt.xticks(rotation=45)
# Добавляем значения на столбцы
for i, v in enumerate(type_avg.values):
    plt.text(i, v + 5, f'${v:.1f}', ha='center', fontweight='bold')

plt.tight_layout()
plt.show()

# ГРАФИК 5: Распределение по категориям мерчантов
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
category_counts = df['merchant_category'].value_counts()
plt.pie(category_counts.values, labels=category_counts.index, autopct='%.1f%%', startangle=90)
plt.title('Распределение по категориям мерчантов', fontweight='bold', fontsize=12)

# ГРАФИК 6: Boxplot по категориям
plt.subplot(2, 2, 2)
top_categories = df['merchant_category'].value_counts().head(4).index
filtered_df = df[df['merchant_category'].isin(top_categories)]
sns.boxplot(data=filtered_df, x='merchant_category', y='transaction_amount')
plt.title('Распределение сумм по категориям', fontweight='bold', fontsize=12)
plt.xlabel('Категория мерчанта')
plt.ylabel('Сумма транзакции ($)')
plt.xticks(rotation=45)

# ГРАФИК 7: Временной анализ - транзакции по часам
plt.subplot(2, 2, 3)
df['hour'] = pd.to_datetime(df['timestamp']).dt.hour
hourly_counts = df['hour'].value_counts().sort_index()
plt.plot(hourly_counts.index, hourly_counts.values, marker='o', linewidth=2, color='purple')
plt.title('Активность транзакций по часам', fontweight='bold', fontsize=12)
plt.xlabel('Час дня')
plt.ylabel('Количество транзакций')
plt.grid(True, alpha=0.3)
plt.xticks(range(0, 24, 2))

```

```

# ГРАФИК 8: Кумулятивная сумма транзакций
plt.subplot(2, 2, 4)
df_sorted = df.sort_values('timestamp')
df_sorted['cumulative_amount'] = df_sorted['transaction_amount'].cumsum()
plt.plot(df_sorted['timestamp'], df_sorted['cumulative_amount'], linewidth=2, color='red')
plt.title('Накопительный объем транзакций', fontweight='bold', fontsize=12)
plt.xlabel('дата')
plt.ylabel('Кумулятивная сумма ($)')
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()

# ГРАФИК 9: Сравнение распределений по типам транзакций
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
# Выбираем топ-3 типа транзакций для наглядности
top_types = df['transaction_type'].value_counts().head(3).index
filtered_types = df[df['transaction_type'].isin(top_types)]

for t in top_types:
    data = filtered_types[filtered_types['transaction_type'] == t]['transaction_amount']
    plt.hist(data, bins=30, alpha=0.6, label=t)

plt.title('Сравнение распределений по типам', fontweight='bold', fontsize=12)
plt.xlabel('Сумма транзакции ($)')
plt.ylabel('Количество')
plt.legend()
plt.grid(True, alpha=0.3)

# ГРАФИК 10: Scatter plot времени и суммы
plt.subplot(1, 2, 2)
df['hour_num'] = pd.to_datetime(df['timestamp']).dt.hour + pd.to_datetime(df['timestamp']).dt.minute / 60
plt.scatter(df['hour_num'], df['transaction_amount'], alpha=0.5, s=20, color='blue')
plt.title('Зависимость суммы от времени суток', fontweight='bold', fontsize=12)
plt.xlabel('Время суток (часы)')
plt.ylabel('Сумма транзакции ($)')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Дополнительная статистика и анализ
print("\n❸ ДЕТАЛЬНЫЙ АНАЛИЗ:")
print(f"\n❹ Статистика по типам транзакций:")
type_stats = df.groupby('transaction_type').agg({
    'transaction_amount': ['count', 'mean', 'sum', 'std'],
    'user_id': 'nunique'
}).round(2)
print(type_stats)

print(f"\n❺ Статистика по категориям мерчантов:")
category_stats = df.groupby('merchant_category').agg({
    'transaction_amount': ['count', 'mean', 'sum'],
    'user_id': 'nunique'
}).round(2)
print(category_stats)

# Анализ крупных транзакций
large_threshold = df['transaction_amount'].quantile(0.95)
large_transactions = df[df['transaction_amount'] > large_threshold]

print(f"\n❻ АНАЛИЗ КРУПНЫХ ТРАНЗАКЦИЙ (топ 5%):")
print(f"Порог крупных транзакций: ${large_threshold:.2f}")
print(f"Количество крупных транзакций: {len(large_transactions)}")
print(f"Доля от общего количества: {len(large_transactions)/len(df)*100:.1f}%")
print(f"Объем крупных транзакций: ${large_transactions['transaction_amount'].sum():,.2f}")

print(f"\n❽ ТОП-5 САМЫХ КРУПНЫХ ТРАНЗАКЦИЙ:")
top_5 = df.nlargest(5, 'transaction_amount')[['transaction_id', 'transaction_amount', 'transaction_type', 'merchant_category']]
print(top_5.to_string(index=False))

# Финальный график: Сравнение средних сумм по категориям
plt.figure(figsize=(12, 6))
category_avg = df.groupby('merchant_category')['transaction_amount'].mean().sort_values(ascending=False)
plt.bar(category_avg.index, category_avg.values, color='lightblue', alpha=0.7, edgecolor='navy')
plt.title('Сравнение средних сумм транзакций по категориям', fontweight='bold', fontsize=14)
plt.xlabel('Категория мерчанта')
plt.ylabel('Средняя сумма ($)')
plt.xticks(rotation=45)

```

```
plt.grid(True, alpha=0.3)

# Добавляем значения на столбцы
for i, v in enumerate(category_avg.values):
    plt.text(i, v + 5, f'{v:.1f}', ha='center', fontweight='bold')

plt.tight_layout()
plt.show()

print(f"\n✅ Визуализация завершена! Создано 11 различных графиков для анализа транзакций.")
```

задание 3

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

# Настройка стиля для красивых графиков
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")
plt.rcParams['font.size'] = 12
plt.rcParams['figure.figsize'] = (12, 8)

print("⌚ АНАЛИЗ ТРАНЗАКЦИЙ ЭЛЕКТРОННЫХ КОШЕЛЬКОВ")
```

```

print("=" * 60)

# Создание реалистичного демо-датасета
def create_digital_wallet_dataset():
    """Создает реалистичный dataset транзакций электронного кошелька"""
    np.random.seed(42)
    n = 5000

    # Генерация сумм с разными распределениями для реалистичности
    amounts = np.concatenate([
        np.random.exponential(50, int(n * 0.6)),      # Мелкие транзакции
        np.random.normal(300, 100, int(n * 0.3)),     # Средние транзакции
        np.random.lognormal(6, 1, int(n * 0.1))       # Крупные транзакции
    ])
    amounts = np.abs(amounts)

    data = {
        'transaction_id': [f'TX{i:06d}' for i in range(1, n + 1)],
        'user_id': np.random.randint(1000, 2000, n),
        'amount': amounts,
        'type': np.random.choice(['Покупка', 'Перевод', 'Пополнение', 'Снятие', 'Оплата услуг'],
                                 n, p=[0.4, 0.25, 0.15, 0.1, 0.1]),
        'category': np.random.choice(['Еда', 'Транспорт', 'Развлечения', 'Магазины', 'Услуги', 'Переводы'],
                                    n, p=[0.25, 0.2, 0.15, 0.15, 0.1, 0.15]),
        'date': pd.date_range('2024-01-01', periods=n, freq='H'),
        'status': np.random.choice(['Успешно', 'Ошибка', 'В обработке'],
                                   n, p=[0.92, 0.05, 0.03]),
        'device': np.random.choice(['Мобильный', 'Компьютер', 'Планшет'],
                                   n, p=[0.75, 0.2, 0.05]),
        'merchant': np.random.choice(['Amazon', 'Google', 'Uber', 'Netflix', 'Spotify', 'Local Store'], n)
    }

    return pd.DataFrame(data)

# Загрузка данных
try:
    df = pd.read_csv('digital_wallet_transactions.csv')
    print("✓ Данные загружены из файла")
except:
    df = create_digital_wallet_dataset()
    print("✓ Создан демонстрационный dataset")

# Базовая информация
print(f"\n📊 ОСНОВНАЯ СТАТИСТИКА:")
print(f"Всего транзакций: {len(df)}")
print(f"Период: {df['date'].min().strftime('%d.%m.%Y')} - {df['date'].max().strftime('%d.%m.%Y')}")
print(f"Общий объем: ${df['amount'].sum():,.2f}")
print(f"Средний чек: ${df['amount'].mean():.2f}")
print(f"Медианный чек: ${df['amount'].median():.2f}")

# ВИЗУАЛИЗАЦИЯ 1: Основное распределение сумм
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 12))

# 1.1 Гистограмма распределения сумм
ax1.hist(df['amount'], bins=50, alpha=0.7, color='skyblue', edgecolor='navy', linewidth=0.5)
ax1.set_title('🔴 Распределение сумм транзакций', fontsize=14, fontweight='bold')
ax1.set_xlabel('Сумма ($)')
ax1.set_ylabel('Количество транзакций')
ax1.grid(True, alpha=0.3)
ax1.axvline(df['amount'].mean(), color='red', linestyle='--', linewidth=2,
            label=f'Среднее: ${df["amount"].mean():.2f}')
ax1.legend()

# 1.2 Распределение в логарифмическом масштабе
ax2.hist(np.log1p(df['amount']), bins=40, alpha=0.7, color='lightgreen', edgecolor='darkgreen')
ax2.set_title('🟩 Логарифмическое распределение сумм', fontsize=14, fontweight='bold')
ax2.set_xlabel('ln(Сумма + 1)')
ax2.set_ylabel('Количество')
ax2.grid(True, alpha=0.3)

# 1.3 Boxplot для выбросов
ax3.boxplot(df['amount'], vert=True, patch_artist=True)
ax3.set_title('🟧 Boxplot сумм транзакций', fontsize=14, fontweight='bold')
ax3.set_xlabel('Сумма ($)')
ax3.grid(True, alpha=0.3)

# 1.4 Количество транзакций по статусам
status_counts = df['status'].value_counts()
colors = ['lightgreen', 'lightcoral', 'lightyellow']
ax4.bar(status_counts.index, status_counts.values, color=colors, alpha=0.7, edgecolor='black')
ax4.set_title('🟩 Статусы транзакций', fontsize=14, fontweight='bold')
ax4.set_xlabel('Количество')

```

```

for i, v in enumerate(status_counts.values):
    ax4.text(i, v + 10, str(v), ha='center', fontweight='bold')

plt.tight_layout()
plt.show()

# ВИЗУАЛИЗАЦИЯ 2: Анализ по типам и категориям
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 12))

# 2.1 Количество транзакций по типам
type_counts = df['type'].value_counts()
ax1.bar(type_counts.index, type_counts.values, color='lightcoral', alpha=0.7)
ax1.set_title('🕒 Распределение по типам транзакций', fontsize=14, fontweight='bold')
ax1.set_ylabel('Количество')
ax1.tick_params(axis='x', rotation=45)
for i, v in enumerate(type_counts.values):
    ax1.text(i, v + 10, str(v), ha='center', fontweight='bold')

# 2.2 Средняя сумма по типам
type_avg = df.groupby('type')['amount'].mean().sort_values(ascending=False)
ax2.bar(type_avg.index, type_avg.values, color='gold', alpha=0.7)
ax2.set_title('💰 Средняя сумма по типам', fontsize=14, fontweight='bold')
ax2.set_ylabel('Средняя сумма ($)')
ax2.tick_params(axis='x', rotation=45)
for i, v in enumerate(type_avg.values):
    ax2.text(i, v + 5, f'${v:.1f}', ha='center', fontweight='bold')

# 2.3 Круговая диаграмма по категориям
category_counts = df['category'].value_counts()
ax3.pie(category_counts.values, labels=category_counts.index, autopct='%1.1f%%',
         startangle=90, colors=sns.color_palette("Set3"))
ax3.set_title('🛍️ Распределение по категориям', fontsize=14, fontweight='bold')

# 2.4 Топ мерчанты по объему
merchant_volume = df.groupby('merchant')['amount'].sum().nlargest(6)
ax4.bar(merchant_volume.index, merchant_volume.values, color='lightblue', alpha=0.7)
ax4.set_title('🛒 Топ мерчанты по объему', fontsize=14, fontweight='bold')
ax4.set_ylabel('Общий объем ($)')
ax4.tick_params(axis='x', rotation=45)
for i, v in enumerate(merchant_volume.values):
    ax4.text(i, v + 100, f'${v:.0f}', ha='center', fontweight='bold')

plt.tight_layout()
plt.show()

# ВИЗУАЛИЗАЦИЯ 3: Временной анализ
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 12))

# 3.1 Транзакции по дням недели
df['day_of_week'] = df['date'].dt.day_name()
df['day_num'] = df['date'].dt.dayofweek
daily_counts = df.groupby('day_num').size()
days = ['Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Вс']
ax1.plot(days, daily_counts, marker='o', linewidth=3, markersize=8, color='purple')
ax1.set_title('📅 Активность по дням недели', fontsize=14, fontweight='bold')
ax1.set_ylabel('Количество транзакций')
ax1.grid(True, alpha=0.3)
for i, v in enumerate(daily_counts):
    ax1.text(i, v + 5, str(v), ha='center', fontweight='bold')

# 3.2 Транзакции по часам
df['hour'] = df['date'].dt.hour
hourly_counts = df.groupby('hour').size()
ax2.bar(hourly_counts.index, hourly_counts.values, color='orange', alpha=0.7)
ax2.set_title('🕒 Активность по часам', fontsize=14, fontweight='bold')
ax2.set_xlabel('Час дня')
ax2.set_ylabel('Количество транзакций')
ax2.grid(True, alpha=0.3)

# 3.3 Еженедельная динамика объема
weekly_volume = df.groupby(pd.Grouper(key='date', freq='W'))['amount'].sum()
ax3.plot(weekly_volume.index, weekly_volume.values, linewidth=3, color='green', marker='s')
ax3.set_title('📅 Еженедельный объем транзакций', fontsize=14, fontweight='bold')
ax3.set_ylabel('Объем ($)')
ax3.grid(True, alpha=0.3)

# 3.4 Устройства для транзакций
device_counts = df['device'].value_counts()
ax4.bar(device_counts.index, device_counts.values, color=['lightblue', 'lightpink', 'lightgray'])
ax4.set_title('📱 Транзакции по устройствам', fontsize=14, fontweight='bold')
ax4.set_ylabel('Количество')
for i, v in enumerate(device_counts.values):
    ax4.text(i, v + 10, str(v), ha='center', fontweight='bold')

```

```

ax4.text(i, v + 5, str(v), ha='center', fontweight='bold')

plt.tight_layout()
plt.show()

# ВИЗУАЛИЗАЦИЯ 4: Детальный анализ категорий
plt.figure(figsize=(16, 10))

# 4.1 Boxplot по категориям
plt.subplot(2, 2, 1)
sns.boxplot(data=df, x='category', y='amount')
plt.title('▣ Распределение сумм по категориям', fontsize=14, fontweight='bold')
plt.xlabel('Категория')
plt.ylabel('Сумма ($)')
plt.xticks(rotation=45)

# 4.2 Heatmap корреляций (если есть числовые данные)
plt.subplot(2, 2, 2)
# Создаем числовые признаки для heatmap
df_numeric = df.copy()
df_numeric['hour'] = df['date'].dt.hour
df_numeric['day'] = df['date'].dt.day
df_numeric['month'] = df['date'].dt.month

numeric_cols = ['amount', 'hour', 'day', 'month']
correlation_matrix = df_numeric[numeric_cols].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0,
            square=True, linewidths=0.5)
plt.title('⌚ Матрица корреляций', fontsize=14, fontweight='bold')

# 4.3 Куммулятивная сумма транзакций
plt.subplot(2, 2, 3)
df_sorted = df.sort_values('date')
df_sorted['cumulative_amount'] = df_sorted['amount'].cumsum()
plt.plot(df_sorted['date'], df_sorted['cumulative_amount'], linewidth=2, color='red')
plt.title('⌚ Накопительный объем транзакций', fontsize=14, fontweight='bold')
plt.xlabel('Дата')
plt.ylabel('Куммулятивная сумма ($)')
plt.grid(True, alpha=0.3)

# 4.4 Распределение по типам устройств и статусам
plt.subplot(2, 2, 4)
cross_tab = pd.crosstab(df['device'], df['status'])
cross_tab.plot(kind='bar', ax=plt.gca(), alpha=0.7)
plt.title('📱 Статусы по устройствам', fontsize=14, fontweight='bold')
plt.xlabel('Устройство')
plt.ylabel('Количество')
plt.legend(title='Статус')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()

# ДЕТАЛЬНАЯ СТАТИСТИКА
print("\n⌚ ДЕТАЛЬНЫЙ АНАЛИЗ:")
print(f"\n▣ Статистика по типам транзакций:")
type_stats = df.groupby('type').agg({
    'amount': ['count', 'mean', 'sum', 'std'],
    'user_id': 'nunique'
}).round(2)
print(type_stats)

print(f"\n▣ Статистика по категориям:")
category_stats = df.groupby('category').agg({
    'amount': ['count', 'mean', 'sum'],
    'user_id': 'nunique'
}).round(2)
print(category_stats)

# Анализ аномалий
Q1 = df['amount'].quantile(0.25)
Q3 = df['amount'].quantile(0.75)
IQR = Q3 - Q1
outlier_threshold = Q3 + 1.5 * IQR
outliers = df[df['amount'] > outlier_threshold]

print(f"\n⚠️ АНАЛИЗ АНОМАЛИЙ:")
print(f"Порог для выбросов: ${outlier_threshold:.2f}")
print(f"Количество аномальных транзакций: {len(outliers)}")
print(f"Доля аномалий: {len(outliers)/len(df)*100:.2f}%")

print(f"\n⌚ ТОП-5 САМЫХ КРУПНЫХ ТРАНЗАКЦИЙ:")

```

```
top_transactions = df.nlargest(5, 'amount')[['transaction_id', 'amount', 'type', 'category', 'date']]  
print(top_transactions.to_string(index=False))  
  
print(f"\nSTATISTIKA PO USTROJSTVAM:")  
device_stats = df.groupby('device').agg({  
    'amount': ['count', 'mean', 'sum'],  
    'user_id': 'nunique'  
})
```