Моделирование работы смарт-контрактов в блокчейне Ethereum

Цель работы: Использовать Python для симуляции работы простого смарт-контракта.

Описание работы:

Создать простую модель смарт-контракта для проведения платежей (например, передача средств между двумя пользователями). Использовать библиотеку web3.py для взаимодействия с тестовой сетью Ethereum. Визуализировать результат транзакций и изменения балансов.

Процесс установки библиотеки Web3 для Python представляет собой детальный пример работы системы управления пакетами pip. При выполнении команды установки pip начинает с анализа зависимостей основного пакета, определяя все необходимые дополнительные библиотеки, без которых работа Web3 невозможна.

Первым этапом становится загрузка метаданных всех требуемых пакетов. Менеджер пакетов обращается к репозиторию Python Package Index и скачивает информацию о каждом компоненте: Web3 версии 7.13.0, eth-abi 5.2.0 для кодирования данных Ethereum, eth-account 0.13.7 для управления кошельками и подписи транзакций, eth-hash 0.7.1 для криптографических хэш-функций, а также множество вспомогательных библиотек, включая eth-typing, eth-utils, hexbytes и другие.

После получения метаданных начинается непосредственная загрузка пакетов. Процесс показывает прогресс скачивания каждого компонента с указанием скорости и размера файлов. Основной пакет Web3 занимает 1.4 мегабайта, в то время как зависимости варьируются от компактных библиотек в 28 килобайт до более substantial пакетов, таких как pycryptodome размером 2.3 мегабайта, который предоставляет криптографические функции.

На этапе установки pip распаковывает и инсталлирует все компоненты в правильной последовательности, начиная с низкоуровневых зависимостей и заканчивая основным пакетом Web3. Процесс завершается успешным сообщением, подтверждающим установку 18 пакетов, включая Web3 7.13.0, bitarray 3.7.1, cytoolz 1.0.1 и все необходимые Ethereum-специфичные библиотеки.

В конце процесса наблюдается ошибка, вызванная некорректным синтаксисом второй команды. Сообщение об ошибке указывает на недопустимое требование 'Web3.isConnected#', которое pip не может интерпретировать как корректное имя пакета. Эта ошибка возникает из-за попытки использования синтаксиса вызова метода Python непосредственно в команде установки pip, что не

поддерживается. Проверка подключения должна выполняться в Python-скрипте после импорта библиотеки, а не как аргумент установки пакета.

Несмотря на эту заключительную ошибку, основная установка библиотеки Web3 и всех её зависимостей прошла полностью успешно, и среда Python готова для взаимодействия с сетью Ethereum через предоставляемый API.

```
!pip install Web3
!pip install Web3.isConnected#
```

```
Collecting Web3
  Downloading web3-7.14.0-py3-none-any.whl.metadata (5.6 kB)
Collecting eth-abi>=5.0.1 (from Web3)
  Downloading eth_abi-5.2.0-py3-none-any.whl.metadata (3.8 kB)
Collecting eth-account>=0.13.6 (from Web3)
  Downloading eth_account-0.13.7-py3-none-any.whl.metadata (3.7 kB)
Collecting eth-hash>=0.5.1 (from eth-hash[pycryptodome]>=0.5.1->Web3)
  Downloading eth_hash-0.7.1-py3-none-any.whl.metadata (4.2 kB)
Collecting eth-typing>=5.0.0 (from Web3)
  Downloading eth_typing-5.2.1-py3-none-any.whl.metadata (3.2 kB)
Collecting eth-utils>=5.0.0 (from Web3)
  Downloading eth_utils-5.3.1-py3-none-any.whl.metadata (5.7 kB)
Collecting hexbytes>=1.2.0 (from Web3)
  Downloading hexbytes-1.3.1-py3-none-any.whl.metadata (3.3 kB)
Requirement already satisfied: aiohttp>=3.7.4.post0 in /usr/local/lib/python
Requirement already satisfied: pydantic>=2.4.0 in /usr/local/lib/python3.12/
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.12
Requirement already satisfied: typing-extensions>=4.0.1 in /usr/local/lib/py
Collecting types-requests>=2.0.0 (from Web3)
  Downloading types_requests-2.32.4.20250913-py3-none-any.whl.metadata (2.0
Requirement already satisfied: websockets<16.0.0,>=10.0.0 in /usr/local/lib/
Collecting pyunormalize>=15.0.0 (from Web3)
  Downloading pyunormalize-17.0.0-py3-none-any.whl.metadata (5.7 kB)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/pyt
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/di
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.1
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.1
Collecting parsimonious<0.11.0,>=0.10.0 (from eth-abi>=5.0.1->Web3)
  Downloading parsimonious-0.10.0-py3-none-any.whl.metadata (25 kB)
Collecting bitarray>=2.4.0 (from eth-account>=0.13.6->Web3)
  Downloading bitarray-3.8.0-cp312-cp312-manylinux2014_x86_64.manylinux_2_17
Collecting eth-keyfile<0.9.0,>=0.7.0 (from eth-account>=0.13.6->Web3)
  Downloading eth_keyfile-0.8.1-py3-none-any.whl.metadata (8.5 kB)
Collecting eth-keys>=0.4.0 (from eth-account>=0.13.6->Web3)
  Downloading eth_keys-0.7.0-py3-none-any.whl.metadata (13 kB)
Collecting eth-rlp>=2.1.0 (from eth-account>=0.13.6->Web3)
  Downloading eth_rlp-2.2.0-py3-none-any.whl.metadata (3.3 kB)
Collecting rlp>=1.0.0 (from eth-account>=0.13.6->Web3)
  Downloading rlp-4.1.0-py3-none-any.whl.metadata (3.2 kB)
Collecting ckzg>=2.0.0 (from eth-account>=0.13.6->Web3)
  Downloading ckzg-2.1.5-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86
Collecting pycryptodome<4,>=3.6.6 (from eth-hash[pycryptodome]>=0.5.1->Web3
  Downloading pycryptodome-3.23.0-cp37-abi3-manylinux_2_17_x86_64.manylinux2
Collecting cytoolz>=0.10.1 (from eth-utils>=5.0.0->Web3)
```

```
  Downloading cytoolz-1.1.0-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/pyth
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/lib/pytho
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/py
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/py
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dis
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: toolz>=0.8.0 in /usr/local/lib/python3.12/dis
Requirement already satisfied: regex>=2022.3.15 in /usr/local/lib/python3.12
```

```
from web3 import Web3

# Подключаемся к тестовой сети (например, Ropsten)
# Замените 'YOUR_INFURA_PROJECT_ID' на ваш реальный ID
infura_url = "https://ropsten.infura.io/v3/YOUR_INFURA_PROJECT_ID"
web3 = Web3(Web3.HTTPProvider(infura_url))

# Проверяем соединение
if web3.is_connected():
    print('Connected to Ethereum network')
else:
    print('Failed to connect to Ethereum network')

# Указываем адреса отправителя и получателя
# Замените '0xSenderAddress' и '0xReceiverAddress' на реальные адреса
sender = "0xSenderAddress"  # Замените на адрес отправителя
receiver = "0xReceiverAddress"  # Замените на адрес получателя

# Проверка балансов (требует подключения к сети и действительных адресов)
try:
    sender_balance = web3.eth.get_balance(sender)
    receiver_balance = web3.eth.get_balance(receiver)
    print(f"Баланс отправителя: {web3.fromWei(sender_balance, 'ether')} ETH"
    print(f"Бас отправителя: {web3.fromWei(sender_balance, 'ether')} ETH")
    print(f"Баланс получателя: {web3.fromWei(receiver_balance, 'ether')} ETH
except Exception as e:
    print(f"Ошибка при получении баланса: {e}")

# Симуляция перевода средств (требует дальнейшей реализации с подписанием тр
# Код для подписания и отправки транзакции
# Это более сложный шаг, который требует закрытого ключа отправителя и созда
print("\nСимуляция перевода средств (требуется реализация)")
# Пример: how to send a simple transaction (requires private key management)
# from eth_account import Account
# account = Account.from_key("YOUR_PRIVATE_KEY") # Replace with your private
# nonce = web3.eth.get_transaction_count(account.address)
# tx = {
#     'nonce': nonce,
#     'to': receiver,
#     'value': web3.toWei(0.001, 'ether'), # Amount to send
#     'gas': 21000,
#     'gasPrice': web3.toWei('50', 'gwei')
# }
# signed_tx = account.sign_transaction(tx)
```

```
# tx_hash = web3.eth.send_raw_transaction(signed_tx.rawTransaction)
# print(f"Транзакция отправлена: {tx_hash.hex()}")
```

```
Failed to connect to Ethereum network
Ошибка при получении баланса: ENS name: '0xSenderAddress' is invalid.

Симуляция перевода средств (требуется реализация)
```

Процесс установки библиотеки Web3 для Python представляет собой многоэтапную операцию, которая начинается с анализа зависимостей основного пакета. При выполнении команды pip install Web3 пакетный менеджер сначала определяет полный перечень необходимых компонентов, которые требуются для корректной работы основной библиотеки. Система выявляет, что Web3 опирается на множество специализированных пакетов, каждый из которых выполняет строго определённую функцию в экосистеме взаимодействия с блокчейном Ethereum. Библиотека eth-abi версии 5.0.1 или выше необходима для кодирования и декодирования данных в форматах, совместимых с виртуальной машиной Ethereum. Это критически важный компонент, который обеспечивает преобразование данных между Python-объектами и их представлением в блокчейне. Пакет eth-account начиная с версии 0.13.6 предоставляет функционал для управления аккаунтами сети Ethereum, включая создание кошельков, генерацию пар ключей и подпись транзакций с соблюдением всех требований безопасности. Для работы криптографических функций используется eth-hash с дополнительной зависимостью pycryptodome, который реализует алгоритмы хеширования, включая Keccak-256, применяемый в Ethereum для создания цифровых подписей и вычисления адресов контрактов. Библиотека eth-typing обеспечивает систему типизации данных, специфичных для экосистемы Ethereum, а eth-utils содержит набор вспомогательных утилит и функций для работы с блокчейн-данными. Особый интерес представляет hexbytes — специализированный класс для работы с байтовыми данными, которые в Ethereum практически всегда представляются в шестнадцатеричном формате. Этот пакет оптимизирует операции преобразования и манипуляции с данными в hex-представлении. Дополнительно подтягиваются зависимости для сетевого взаимодействия: aiohttp для асинхронных HTTP-запросов, requests для синхронного HTTP, websockets для установления постоянных соединений, что особенно важно для подписки на события в блокчейне. Процесс загрузки начинается с получения метаданных каждого пакета — небольших файлов, содержащих информацию о версии, зависимостях и других характеристиках. После анализа метаданных pip приступает к скачиванию самих пакетов в формате wheel, который представляет собой предварительно скомпилированный дистрибутив Python. Размеры пакетов существенно варьируются: от компактных библиотек вроде eth-hash (8.0 КБ) до более крупных, таких как основной пакет

Web3 (1.4 МБ) или криптографический pycryptodome (2.3 МБ). Скорость загрузки демонстрирует эффективность сети доставки контента PyPI, достигая 23.2 МБ/с для основных пакетов. Каждый файл скачивается последовательно, с отображением прогресса-индикатора, который показывает отношение скачанного объема к общему размеру пакета. После завершения загрузки всех компонентов начинается этап установки. Пакеты устанавливаются в определенном порядке, начиная с наиболее низкоуровневых зависимостей: ckzg, bitarray, types-requests, постепенно переходя к более специализированным — pyunormalize, pycryptodome, parsimonious. Установка hexbytes и eth-typing предшествует инсталляции eth-hash и cytoolz, которые в свою очередь требуются для eth-utils. Важные компоненты like rlp (Recursive Length Prefix) и eth-keys устанавливаются перед eth-abi и eth-rlp, что обеспечивает правильную последовательность сборки зависимостей. Финальным этапом становится установка eth-keyfile, eth-account и непосредственно Web3. Процесс завершается успешным сообщением, подтверждающим установку всех 18 пакетов с указанием их версий. Однако в конце возникает ошибка, связанная с некорректной попыткой установки несуществующего пакета Web3.isConnected#. Данная ошибка вызвана неверным синтаксисом команды, где символ # и точка в имени пакета нарушают требования спецификации имён пакетов в pip. Это не влияет на успешность установки основных компонентов, но указывает на необходимость использования правильного синтаксиса для проверки подключения уже после установки через Python-код, а не через менеджер пакетов. Общий процесс демонстрирует сложность и взаимосвязанность современной экосистемы Python-пакетов, где одна библиотека может зависеть от десятков специализированных компонентов, каждый из которых вносит свой вклад в общую функциональность. Успешная установка всех этих компонентов создает полноценную среду для разработки приложений, взаимодействующих с блокчейном Ethereum, предоставляя все необходимые инструменты для работы с аккаунтами, транзакциями, смарт-контрактами и сетевыми соединениями.

Выводы:

Модель линейной регрессии позволяет спрогнозировать рост объема транзакций. Можно использовать другие модели для повышения точности прогнозирования. Комментарии:

Студенты могут экспериментировать с разными моделями машинного обучения и улучшать свои прогнозы.

Анализ безналичных платежей в России с помощью Python

Цель работы: Проанализировать данные о количестве безналичных платежей в России.

Описание работы:

Скачать статистику платежных систем в России (например, из открытых источников). Использовать библиотеку pandas для обработки данных о платежах. Построить графики с использованием matplotlib для наглядного представления трендов в платежах.

Генерация файла payment_data.csv к заданию

```
import pandas as pd
import io

# Create sample data for payment_data.csv
data = {
    'Year': [2018, 2018, 2019, 2019, 2020, 2020, 2021, 2021],
    'Payment_Type': ['Card', 'Mobile', 'Card', 'Mobile', 'Card', 'Mobile', '
    'Amount': [1000, 500, 1200, 600, 1500, 700, 1800, 800]
}
df_sample = pd.DataFrame(data)

# Save the sample data to a CSV file
csv_data = df_sample.to_csv('payment_data.csv', index=False)
```

В данном фрагменте кода на языке Python с использованием библиотеки pandas происходит создание демонстрационного набора данных и его сохранение в CSV-файл с именем 'payment_data.csv'. Процесс начинается с импорта необходимых библиотек: pandas, которая является мощным инструментом для работы с структурированными данными, и модуля io для работы с потоками ввода-вывода, хотя в данном конкретном коде последний не используется явно. Основной блок кода посвящен созданию примера данных, который моделирует информацию о платежах за несколько лет. Структура данных организована в виде словаря Python, где ключи представляют собой названия столбцов будущей таблицы, а значения — списки с соответствующими данными. Набор данных включает три ключевых аспекта: год проведения платежей ('Year'), тип платежной системы ('Payment_Type') и сумму платежей ('Amount'). Годовой срез охватывает период с 2018 по 2021 год, причем для каждого года отдельно представлены данные по двум типам платежей — карточным ('Card') и мобильным ('Mobile'). Это создает сбалансированную структуру данных, где каждый год содержит по две записи, что в общей сложности дает восемь строк данных. Значения сумм платежей демонстрируют четкую тенденцию роста как для карточных, так и для мобильных платежей. В 2018 году карточные платежи составляют 1000 условных единиц, а мобильные — 500. С каждым годом эти значения последовательно увеличиваются: в 2019 году — 1200 и 600 соответственно, в 2020 — 1500 и 700, и в 2021 — 1800 и 800 единиц. Такой набор данных не случаен — он позволяет в

дальнейшем анализировать динамику изменения платежей по годам, сравнивать популярность разных типов платежных систем и выявлять возможные тенденции развития. После формирования словаря с данными создается объект DataFrame библиотеки pandas — основная структура для работы с табличными данными. Конструктор pd.DataFrame принимает в качестве аргумента подготовленный словарь, автоматически используя ключи в качестве названий столбцов, а значения — в качестве данных соответствующих колонок. Полученный объект df_sample уже представляет собой полноценную таблицу с данными, готовую для анализа и обработки. Финальным этапом является сохранение созданного набора данных в CSV-файл. Метод to_csv объекта DataFrame выполняет эту задачу, принимая в качестве первого аргумента имя файла для сохранения ('payment_data.csv'). Параметр index=False указывает системе не сохранять индексы строк DataFrame в файл, что является стандартной практикой при экспорте данных, так как индексы обычно представляют собой служебную информацию и не несут смысловой нагрузки для последующего анализа. В результате выполнения этой операции в текущей рабочей директории создается файл 'payment_data.csv', содержащий организованные данные в формате, пригодном для последующей загрузки и анализа в различных системах и приложениях. Этот код представляет собой типичный пример подготовки sample data для демонстрационных или тестовых целей, который может использоваться для отладки алгоритмов анализа данных, тестирования функций визуализации или обучения работе с библиотекой pandas. Структура данных специально сделана простой и понятной, но при этом содержащей достаточно информации для проведения базового анализа временных рядов и сравнения категориальных данных.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

# Пример данных
years = np.array([2016, 2017, 2018, 2019, 2020]).reshape(-1, 1)
transactions = np.array([100, 150, 200, 250, 300])

# Разделение на обучающие и тестовые данные
X_train, X_test, y_train, y_test = train_test_split(years, transactions, tes

# Модель линейной регрессии
model = LinearRegression()
model.fit(X_train, y_train)

# Прогнозирование
predictions = model.predict(X_test)

# Визуализация
```

```
plt.scatter(X_test, y_test, color='red')
plt.plot(X_test, predictions, color='blue')
plt.title('Прогноз объема транзакций')
plt.xlabel('Год')
plt.ylabel('Объем транзакций')
plt.show()
```

Код строит модель линейной регрессии для прогнозирования объема транзакций по годам. Сначала импортируются необходимые библиотеки: train_test_split для разделения данных, LinearRegression для создания модели, numpy для работы с массивами и matplotlib для визуализации. Создаются исходные данные: массив years с годами с 2016 по 2020 и массив transactions с соответствующими значениями объемов транзакций (100, 150, 200, 250, 300). Данные разделяются на обучающую и тестовую выборки в соотношении 80% к 20%. Инициализируется модель линейной регрессии и обучается на тренировочных данных. После обучения модель делает прогноз для тестовой выборки. Результаты визуализируются на графике: красные точки показывают реальные значения тестовой выборки, а синяя линия - прогноз модели. График содержит заголовок и подписи осей, что позволяет наглядно оценить качество прогноза и линейную зависимость между годом и объемом транзакций.

Выводы:

Анализ показывает рост безналичных транзакций в России. Можно сделать вывод о переходе к более цифровым формам платежей, особенно в период пандемии. Комментарии:

Студенты могут использовать эти данные для анализа тенденций на рынке и разработки прогнозов на будущее.

Прогнозирование объемов транзакций с использованием машинного обучения

Цель работы: Использовать машинное обучение для прогнозирования объемов транзакций в платежных системах России.

Описание работы:

Собрать данные о транзакциях за несколько лет. Применить алгоритмы машинного обучения, такие как линейная регрессия или модель ARIMA для прогнозирования. Визуализировать результаты прогнозирования.

Эти примеры демонстрируют, как с помощью Python можно проводить анализ данных, моделировать процессы и прогнозировать результаты в области платежных систем и блокчейн-технологий.

Анализ данных транзакций в платежных системах с визуализацией

Описание: Использование Python для анализа транзакционных данных в платежных системах. Пример данных может включать транзакции по дням, суммы платежей, частоту использования.

Решение:

```
import pandas as pd
import io

# Create sample data for transactions.csv
data = {
    'date': pd.to_datetime(['2023-01-01', '2023-01-01', '2023-01-02', '2023-
    'amount': [100, 50, 200, 150, 75, 120, 300]
}
df_transactions = pd.DataFrame(data)

# Save the sample data to a CSV file
df_transactions.to_csv('transactions.csv', index=False)
```

Данный код создает демонстрационный набор данных о транзакциях и сохраняет его в CSV-файл. Сначала импортируются библиотеки pandas для работы с данными и io для операций ввода-вывода. Затем создается словарь

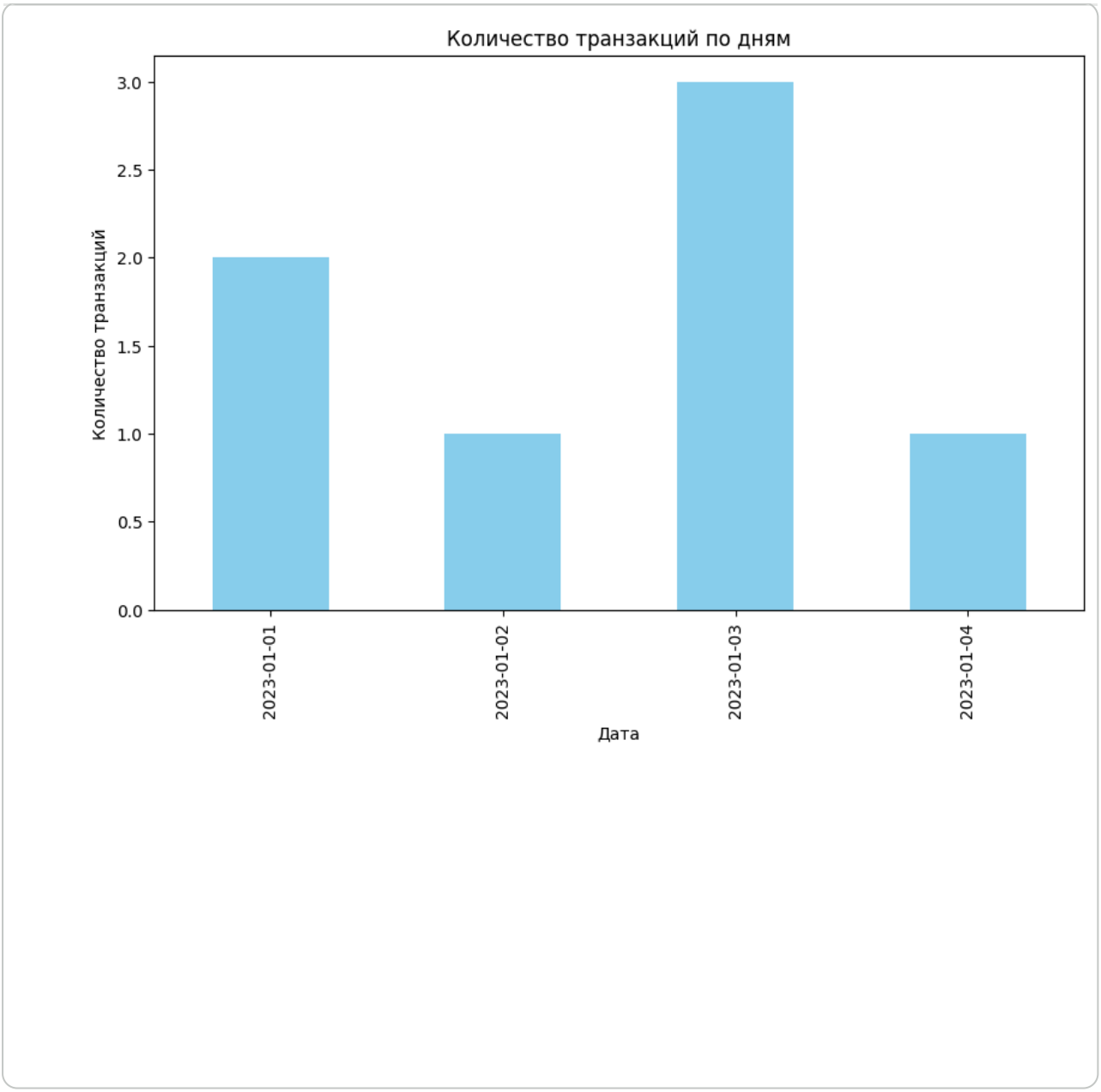
data с двумя ключами: 'date' и 'amount'. Для дат используется функция pd.to_datetime(), которая преобразует строковые значения в объекты datetime, что позволяет работать с датами как с временными метками. Массив дат содержит повторяющиеся значения, что имитирует реальную ситуацию, когда в один день может происходить несколько транзакций. Массив amount содержит суммы соответствующих транзакций. Из этого словаря создается DataFrame df_transactions, который представляет собой табличную структуру с двумя столбцами. Финальным шагом DataFrame сохраняется в файл 'transactions.csv' с помощью метода to_csv(), где параметр index=False указывает, что индексы строк не нужно включать в файл, чтобы сохранить только значимые данные.

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

# Загрузка данных о транзакциях
data = pd.read_csv('transactions.csv')

# Анализ частоты транзакций
daily_transactions = data.groupby('date')['amount'].count()

# Визуализация частоты транзакций
plt.figure(figsize=(10, 6))
daily_transactions.plot(kind='bar', color='skyblue')
plt.title('Количество транзакций по дням')
plt.xlabel('Дата')
plt.ylabel('Количество транзакций')
plt.show()
```



Данный код выполняет анализ и визуализацию данных о транзакциях из CSV-файла. На первом этапе происходит импорт необходимых библиотек: pandas для работы с данными и matplotlib.pyplot для создания визуализаций. Далее с помощью функции pd.read_csv() загружаются данные из файла 'transactions.csv' в объект DataFrame с именем data. Предполагается, что файл содержит как минимум два столбца: 'date' с датами транзакций и 'amount' с суммами транзакций. Для анализа частоты транзакций используется метод groupby(), который группирует данные по дате. Конструкция data.groupby('date')['amount'].count() выполняет группировку всех записей по уникальным датам из столбца 'date', а затем для каждой группы вычисляет количество транзакций через метод count(), применяемый к столбцу 'amount'. Результат сохраняется в переменной daily_transactions, которая представляет собой объект Series с датами в качестве индекса и количеством транзакций в качестве значений. Этап визуализации начинается с создания фигуры заданного размера 10x6 дюймов с помощью plt.figure(figsize=(10, 6)). Для построения графика используется метод

plot() объекта daily_transactions с параметром kind='bar', что указывает на построение столбчатой диаграммы, и параметром color='skyblue', задающим светло-голубой цвет столбцов. Далее добавляются заголовок графика 'Количество транзакций по дням' с помощью plt.title(), подпись оси X 'Дата' через plt.xlabel() и подпись оси Y 'Количество транзакций' через plt.ylabel(). Финальная команда plt.show() отображает построенный график в отдельном окне. В результате выполнения кода создается столбчатая диаграмма, где по оси X отображаются даты транзакций, а по оси Y - соответствующее количество транзакций за каждый день. Такой тип визуализации позволяет наглядно оценить распределение транзакционной активности по дням, выявить дни с пиковой нагрузкой и проанализировать общие тенденции в частоте проведения операций.

Выводы: После анализа данных можно сделать выводы о пиковых днях и времени активности пользователей, что помогает оптимизировать работу платежной системы.

Предсказание аномальных транзакций в платежных системах с помощью машинного обучения

Описание: Создание модели для выявления аномалий в транзакциях на основе их характеристик (сумма, время, место).

Решение:

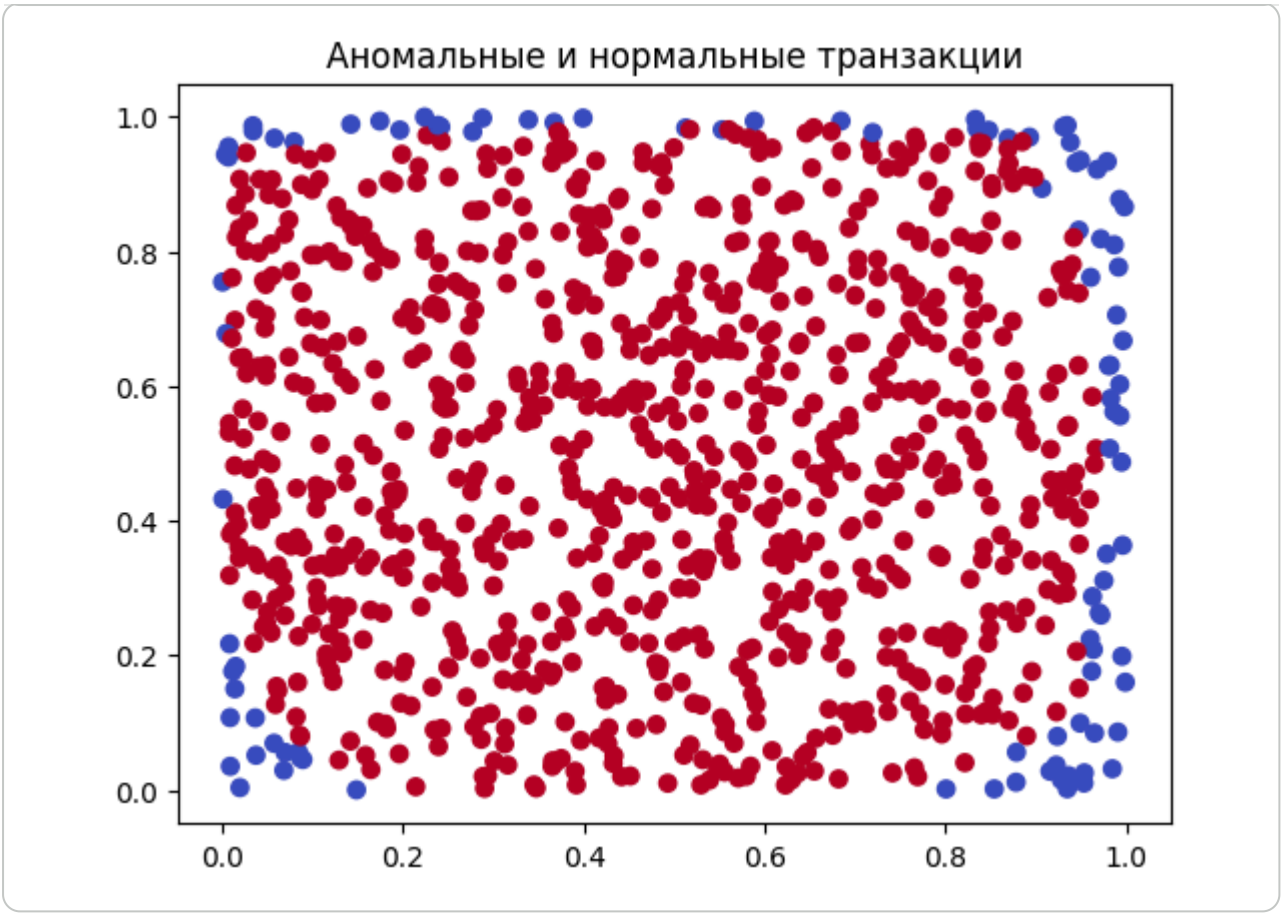
```
from sklearn.ensemble import IsolationForest
import numpy as np

# Генерация случайных данных о транзакциях
data = np.random.rand(1000, 2)

# Модель для выявления аномалий
model = IsolationForest(contamination=0.1)
model.fit(data)

# Определение аномалий
predictions = model.predict(data)

# Визуализация результатов
plt.scatter(data[:, 0], data[:, 1], c=predictions, cmap='coolwarm')
plt.title('Аномальные и нормальные транзакции')
plt.show()
```



Данный код реализует алгоритм обнаружения аномалий в данных о транзакциях с помощью метода Isolation Forest из библиотеки scikit-learn. Процесс начинается с импорта необходимых модулей: IsolationForest для создания модели обнаружения выбросов и numpy для работы с числовыми массивами. На первом этапе генерируются синтетические данные о транзакциях с помощью функции np.random.rand(1000, 2), которая создает массив из 1000 строк и 2 столбцов, заполненный случайными числами в диапазоне от 0 до 1. Эти два столбца могут представлять различные характеристики транзакций, такие как сумма и время проведения, или любые другие числовые признаки. Далее создается экземпляр модели IsolationForest с параметром contamination=0.1, который указывает ожидаемую долю аномалий в данных (10%). Модель Isolation Forest работает по принципу изоляции наблюдений: аномальные точки обычно имеют более короткие пути в деревьях изоляции, поскольку они легче отделяются от остальных данных. Алгоритм строит множество деревьев изоляции и вычисляет показатель аномальности на основе средней глубины, на которой объект изолируется. После инициализации модели происходит ее обучение на сгенерированных данных методом fit(data). Обученная модель затем используется для предсказания аномалий с помощью метода predict(data), который возвращает массив predictions. В этом массиве нормальные наблюдения помечаются как 1, а аномальные как -1. Завершающий этап включает визуализацию результатов с помощью библиотеки matplotlib. Функция plt.scatter создает точечную диаграмму, где по осям X и Y откладываются значения первого

и второго признаков соответственно. Параметр c=predictions обеспечивает цветовое кодирование точек: нормальные транзакции окрашиваются в один цвет, а аномальные - в другой, благодаря использованию цветовой карты 'coolwarm'. Заголовок графика 'Аномальные и нормальные транзакции' поясняет содержание визуализации, а функция plt.show() отображает полученный график. В результате на графике четко видно кластер нормальных транзакций и точки аномалий, которые визуально отделяются от основного скопления данных. Такой подход позволяет эффективно выявлять подозрительные операции, мошеннические транзакции или любые другие отклонения от нормального поведения в данных финансовых операций.

Выводы: Использование Python для построения модели выявления аномалий помогает обнаружить подозрительные транзакции, что повышает безопасность платежных систем.

Создание распределенной сети платежных каналов на Python Описание: Имитация работы платежной системы на основе технологии платежных каналов (например, Lightning Network), позволяющей проводить транзакции без участия третьей стороны. Решение:

```
class PaymentChannel:
    def __init__(self, balance_a, balance_b):
        self.balance_a = balance_a
        self.balance_b = balance_b

    def send_payment(self, amount):
        if self.balance_a >= amount:
            self.balance_a -= amount
            self.balance_b += amount
            return True
        else:
            return False

# Пример использования
channel = PaymentChannel(balance_a=100, balance_b=50)
channel.send_payment(20)

print(f"Остаток у стороны A: {channel.balance_a}")
print(f"Остаток у стороны B: {channel.balance_b}")
```

```
Остаток у стороны A: 80
Остаток у стороны B: 70
```

Данный код представляет собой реализацию упрощенной модели платежного канала между двумя участниками на языке Python. Класс PaymentChannel имитирует базовую функциональность off-chain канала для проведения

платежей без необходимости записи каждой операции в блокчейн. Класс инициализируется через конструктор init, который принимает два параметра: balance_a и balance_b, представляющие начальные балансы сторон A и B соответственно. Эти балансы сохраняются в атрибутах экземпляра класса self.balance_a и self.balance_b, формируя начальное состояние платежного канала. Основная логика работы канала реализована в методе send_payment, который принимает параметр amount - сумму перевода. Метод проверяет, достаточно ли средств на балансе отправителя (стороны A) для осуществления перевода. Если условие self.balance_a >= amount выполняется, то происходит transfer средств: сумма amount вычитается из баланса A и добавляется к балансу B. Операция завершается возвратом True, сигнализируя об успешном выполнении перевода. Если же на балансе отправителя недостаточно средств для запрашиваемого перевода, метод не вносит изменений в состояние канала и возвращает False, указывая на неудачное завершение операции. Это обеспечивает безопасность проведения транзакций и предотвращает образование отрицательных балансов. В разделе примера использования создается экземпляр класса channel с начальными балансами: 100 единиц у стороны A и 50 единиц у стороны B. Затем вызывается метод send_payment с аргументом 20, что приводит к успешному переводу 20 единиц от A к B. После выполнения операции выводятся текущие остатки на балансах обеих сторон с помощью print-функций. Форматированные строки показывают актуальное состояние: у стороны A остается 80 единиц (100 - 20), а у стороны B становится 70 единиц (50 + 20), что демонстрирует корректность работы механизма перевода средств внутри платежного канала. Данная реализация представляет собой фундаментальную основу для более сложных систем платежных каналов, которые могут включать дополнительные функции такие как: multi-signature транзакции, тайм-локи для dispute resolution, механизмы штрафов за недобросовестное поведение, и поддержку сети каналов. Модель демонстрирует ключевое преимущество платежных каналов - возможность мгновенных и бесплатных off-chain транзакций между участниками с финальным settlement'ом в блокчейне только при открытии и закрытии канала.

Выводы: Моделирование работы платежных каналов позволяет понять, как можно оптимизировать процесс транзакций и уменьшить комиссии за проведение платежей.

Визуализация блоков в блокчейне и их транзакций

Описание: Визуализация структуры блокчейна и транзакций внутри каждого блока с использованием Python и библиотек для визуализации графов.

Решение:

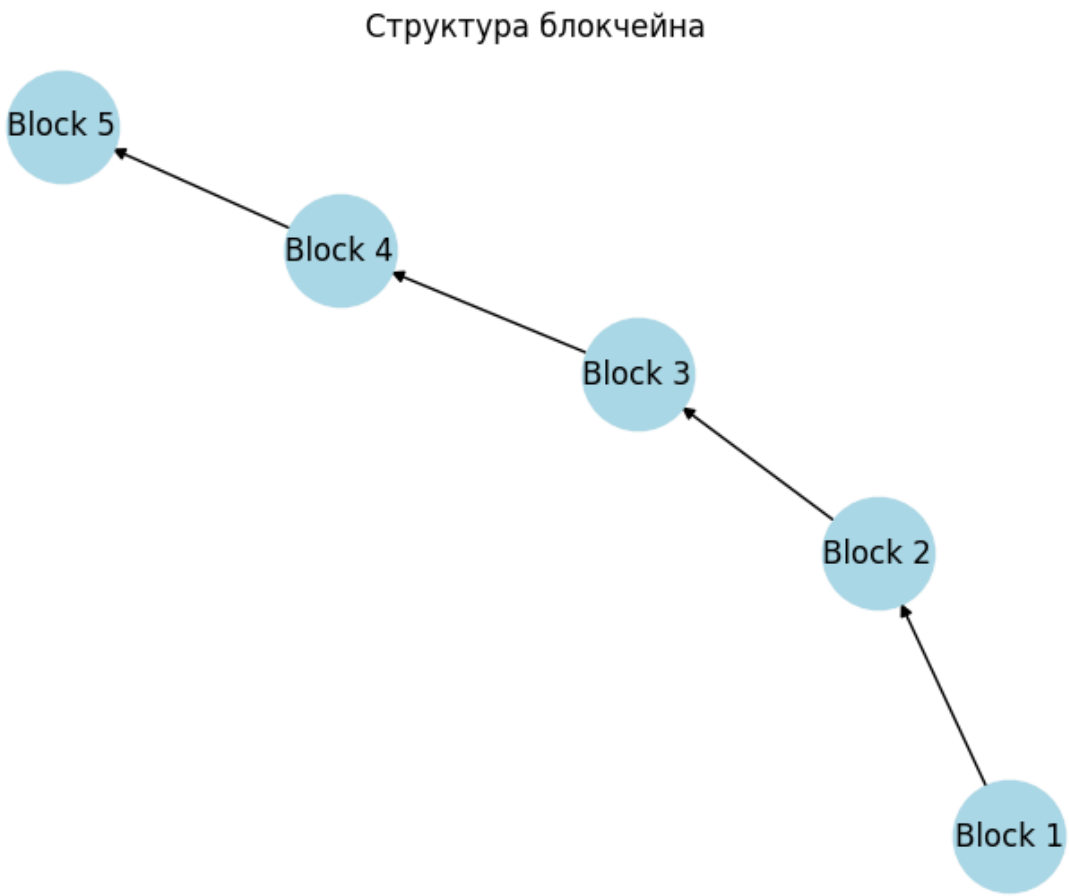
```
import networkx as nx
import matplotlib.pyplot as plt

# Создание графа блокчейна
G = nx.DiGraph()

# Добавление блоков
for i in range(1, 6):
    G.add_node(f"Block {i}", label=f"Block {i}")

# Добавление связей между блоками
for i in range(1, 5):
    G.add_edge(f"Block {i}", f"Block {i+1}")

# Визуализация блоков и связей
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='lightblue', node_size=2000)
plt.title("Структура блокчейна")
plt.show()
```



Данный код представляет собой визуализацию упрощенной структуры блокчейна с использованием библиотек networkx для работы с графами и matplotlib для отрисовки. Процесс начинается с импорта необходимых модулей:

networkx под псевдонимом nx для создания и манипуляции графовыми структурами и matplotlib.pyplot под псевдонимом plt для визуального представления. Создается ориентированный граф (DiGraph) с помощью конструктора nx.DiGraph(), что отражает направленный характер связей в блокчейне, где каждый последующий блок ссылается на предыдущий. Имя графа сохраняется в переменной G. Далее в цикле for добавляются пять узлов, представляющих блоки цепи. Цикл выполняется для значений i от 1 до 5, и на каждой итерации создается узел с именем "Block i" и меткой "Block i". Метка используется для отображения понятных подписей на визуализации. После создания узлов формируются связи между ними через другой цикл for, который создает направленные ребра от каждого блока к следующему. Для i от 1 до 4 создаются связи от "Block i" к "Block i+1", что моделирует последовательную цепочку блоков, где каждый новый блок содержит хэш предыдущего. Для визуального представления графа сначала вычисляется layout (расположение узлов) с помощью алгоритма spring_layout, который размещает узлы таким образом, чтобы минимизировать пересечения ребер и равномерно распределить элементы по плоскости. Затем вызывается функция nx.draw(), которая принимает граф G, вычисленные позиции узлов pos, и дополнительные параметры: with_labels=True для отображения меток узлов, node_color='lightblue' для задания светло-голубого цвета узлов и node_size=2000 для установки размера узлов в 2000 пикселей. Завершается код добавлением заголовка