СОДЕРЖАНИЕ

ЦЕЛЬ ЗАДАНИЯ

Разработать ИИ агента, который будет анализировать доступные банковские вклады и рекомендовать наиболее подходящие варианты для пользователя на основе его финансовых целей, предпочтений и текущей экономической ситуации.

СПОСОБ РЕШЕНИЯ № 1

СБОР И ПОДГОТОВКА ДАННЫХ

Данные о банковских вкладах получены из официальных публичных источников (веб-сайт ПАО «Сбербанк») и произведен ручной сбор для сохранения информации в формате .csv (Рисунок 1).

name	rate	min_sum	term_min	term_max	replenishable	withdrawal	goal
"Лучший %"	18	100000	1	36	False	True	max_income
"СберВклад"	18	100000	1	36	True	True	savings
"Накопительный счёт"	16	0	0	0	True	True	passive_income
"Лучший % Лидер"	19	100000	1	36	False	False	max_income
"Управляй +"	15	50000	3	36	True	True	flexible
"Забота о будущем"	17	150000	12	60	True	False	long_term

Рисунок 1 – Данные о банковских вкладах

Итоговый датасет содержит информацию о шести вкладах и содержит семь параметров по каждому из них, включая:

- *ставку (rate);*
- *минимальную сумму (min_sum);*
- *срок от (term_min), мес.;*
- *срок до (term_max), мес.;*
- *возможность пополнения (replenishable);*
- *возможность снятия (withdrawal);*
- *цель вклада (goal).*

Для разработки ИИ агента выбран язык программирования Python 3.13, среда разработки – PyCharm 2025.1.3.

МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

Математическая модель решения заключается в использовании взвешенной системы оценки. Каждый параметр вклада имеет вес (баллы), который указывает на его важность для пользователя.

Перед тем, как оценивать важность каждого параметра, отсеиваются неподходящие варианты, которые не будут участвовать в дальнейшей оценке. Это делается на основе суммы, которую пользователь готов вложить.

Для каждого вклада программа проверяет условие:

$$\text{Введенная пользователем сумма} \geq 0{,}8 \times \text{min\_sum}$$

Например, если вклад «Лучший %» требует 100 000 рублей, а пользователь ввел 80 000, то программа будет проверять дальше этот вклад по критериям, но если введено 70 000, то отсеет вклад. Это гибкий порог, чтобы не отбрасывать вклады, которые почти подходят.

1. **Возможность пополнения и снятия** – по 10% веса (вклады, у которых условия пополнения совпадают с выбором пользователя получают 10 баллов, также и со снятием; если условия не совпадают – 0 баллов)
2. **Срок вклада** – 20% веса:
   a. если у вклада нет максимального срока (*term_max = 0*), это означает *бессрочный* и такой вклад всегда получает 20 баллов, так как подходит под любой срок, указанный пользователем;
   b. если у вклада есть максимальный срок (*term_max>0*), то вклад получает 20 баллов, если:

$$\textit{желаемый срок пользователя} \leq \textit{term\_max вклада}$$

   c. в остальных случаях вклад получает 0 баллов по этому параметру.
3. **Процентная ставка –** 30% веса

Отдается предпочтение тому вкладу, у которого ставка выше остальных. Балл для каждого вклада рассчитывается как отношение ставки вклада к максимальной из датасета, умноженное на 30.

4. **Финансовая цель –** 30% веса

Вклады, которые совпадают с целью пользователя, получают 30 баллов, остальные – 0.

Формирование рекомендаций

После того, как критерии каждого вклада получили веса (баллы), рассчитывается их сумма – общий балл. Пользователю рекомендуется вклад, набравший максимальный балл.

Если окажется, что сумма, которую готов вложить пользователь, меньше минимальной суммы для самого подходящего вклада, программа предлагает добавить сумму. Если пользователь не хочет добавить средства, программа предлагает альтернативу – лучший из доступных вариантов, который полностью соответствует его текущим возможностям.

РЕАЛИЗАЦИЯ

Для разработки ИИ агента выбран язык программирования Python 3.13, среда разработки – PyCharm 2025.1.3.

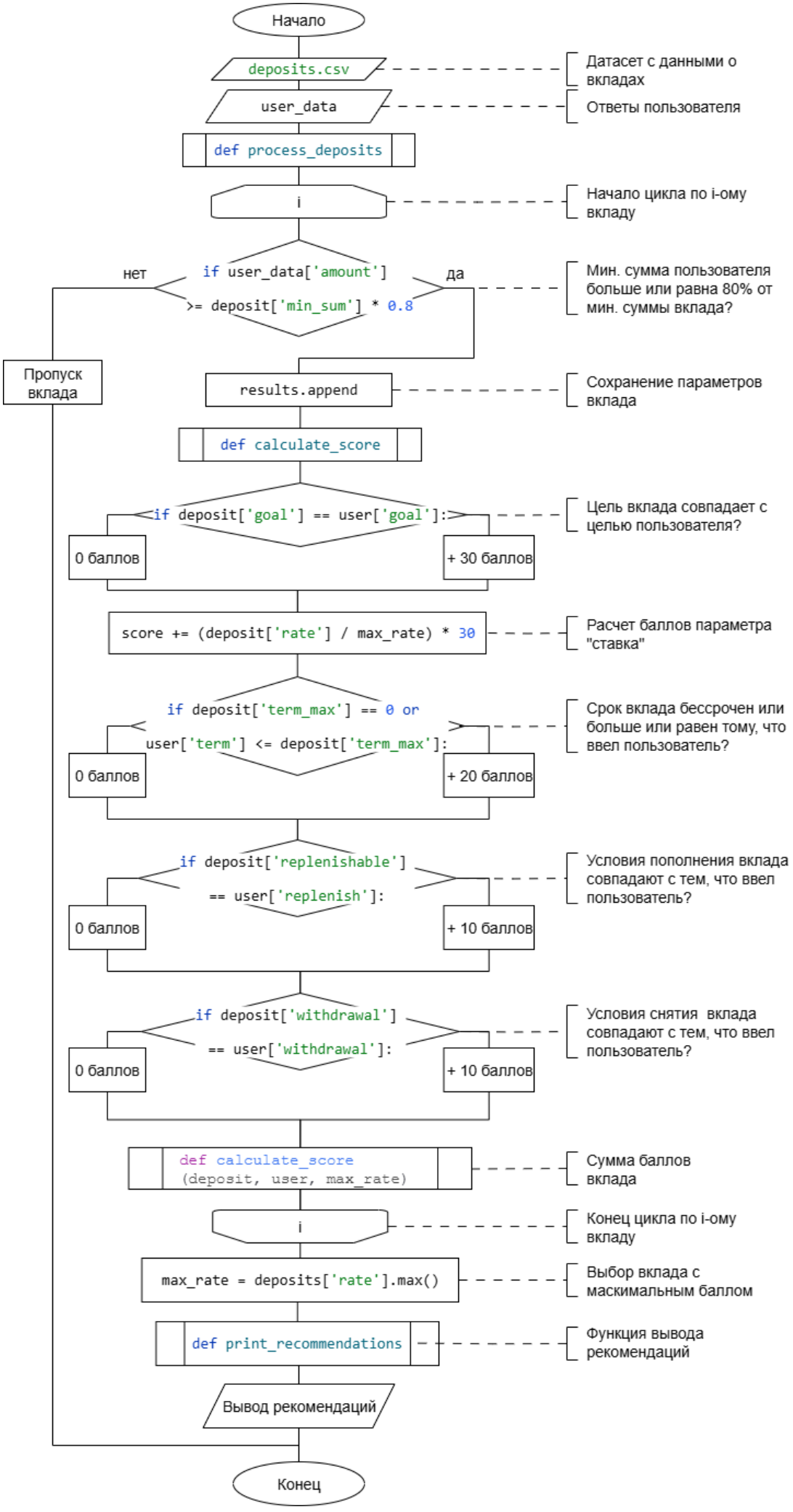
Чтобы масштабировать программу для работы с большими объемами данных, в ней предусмотрена параллельная обработка данных. Количество вкладов равномерно распределяется между процессами для независимой обработки, что ускоряет время работы программы.

Выбор библиотек

1. Pandas – для работы с таблицами (чтение csv-файла);
2. Numpy – работа с массивами, разделение данных на части для параллелизации;
3. Multiprocessing – создание пула процессов (mp.Pool) для обработки данных одновременно;
4. Time – работа со временем (измерить скорость работы программы).

МОДЕЛИРОВАНИЕ

Программа для рекомендаций реализована на основе правил (взвешенная система оценки). Ниже приведена алгоритмическая схема предлагаемого решения.


Начало
deposits.csv
Датасет с данными о вкладах
user_data
Ответы пользователя
def process_deposits
i
Начало цикла по i-ому вкладу
нет
if user_data['amount']
>= deposit['min_sum'] * 0.8
да
Мин. сумма пользователя больше или равна 80% от мин. суммы вклада?
Пропуск вклада
results.append
Сохранение параметров вклада
def calculate_score
if deposit['goal'] == user['goal']:
Цель вклада совпадает с целью пользователя?
0 баллов
+ 30 баллов
score += (deposit['rate'] / max_rate) * 30
Расчет баллов параметра "ставка"
if deposit['term_max'] == 0 or
user['term'] <= deposit['term_max']:
Срок вклада бессрочен или больше или равен тому, что ввел пользователь?
0 баллов
+ 20 баллов
if deposit['replenishable']
== user['replenish']:
Условия пополнения вклада совпадают с тем, что ввел пользователь?
0 баллов
+ 10 баллов
if deposit['withdrawal']
== user['withdrawal']:
Условия снятия вклада совпадают с тем, что ввел пользователь?
0 баллов
+ 10 баллов
def calculate_score
(deposit, user, max_rate)
Сумма баллов вклада
i
Конец цикла по i-ому вкладу
max_rate = deposits['rate'].max()
Выбор вклада с маскимальным баллом
def print_recommendations
Функция вывода рекомендаций
Вывод рекомендаций
Конец

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Интерфейс для взаимодействия с пользователем представляет из себя консольный ввод/вывод. Ниже приведены результаты тестирования ИИ агента.

```
C:\Users\Анастасия\PycharmProjects\PythonProject>python AI_deposit.py

Выберите финансовую цель:
1. Максимальный доход (прибыль в краткие сроки)
2. Накопление на цель, крупную покупку
3. Пассивный доход (частые выплаты процентов)
4. Гибкое управление (возможность снятия и пополнения без штрафов)
5. Долгосрочные сбережения (подушка безопасности)

Номер цели (1-5): 1

ПАРАМЕТРЫ ВКЛАДА
Сумма для вклада (руб): 90000
Срок (1-36 мес., 0 - бессрочно): 20
Нужно пополнение? (да/нет): нет
Нужно снятие? (да/нет): да
Выплата процентов:
1 - Ежемесячно
2 - В конце срока
3 - Без разницы
Ваш выбор (1-3): 1

РЕКОМЕНДАЦИИ
1. Совет: Доплатите 10000 ₽, чтобы выбрать вклад: "Лучший %"
   Ставка: 18% | Подходит Вам на: 98.4/100
2. Без доплат: "Накопительный счёт"
   Ставка: 16% | Подходит Вам на: 55.3/100

Время выполнения программы: 1.4061 секунд
```

Рисунок 2 – пример работы ИИ агента в командной строке Windows 10

```
Выберите финансовую цель:
1. Максимальный доход (прибыль в краткие сроки)
2. Накопление на цель, крупную покупку
3. Пассивный доход (частые выплаты процентов)
4. Гибкое управление (возможность снятия и пополнения без штрафов)
5. Долгосрочные сбережения (подушка безопасности)

Номер цели (1-5): 2

ПАРАМЕТРЫ ВКЛАДА
Сумма для вклада (руб): 200000
Срок (1-36 мес., 0 - бессрочно): 30
Нужно пополнение? (да/нет): да
Нужно снятие? (да/нет): да
Выплата процентов:
1 - Ежемесячно
2 - В конце срока
3 - Без разницы
Ваш выбор (1-3): 3

РЕКОМЕНДАЦИИ
1. "СберВклад"
   Ставка: 18% | Подходит Вам на: 98.4/100

Время выполнения программы: 2.8553 секунд
```

Рисунок 3 – пример работы ИИ агента в среде разработки PyCharm

.

ЛИСТИНГ

```
import pandas as pd
import numpy as np
import multiprocessing as mp
import time

deposits = pd.read_csv('deposits.csv', sep=';')

GOAL_CHOICES = {
    1: {'name': 'max_income', 'desc': "Максимальный доход (прибыль в
краткие сроки)"},
    2: {'name': 'savings', 'desc': "Накопление на цель, крупную
покупку"},
    3: {'name': 'passive_income', 'desc': "Пассивный доход (частые
выплаты процентов)"},
    4: {'name': 'flexible', 'desc': "Гибкое управление (возможность
снятия и пополнения без штрафов)"},
    5: {'name': 'long_term', 'desc': "Долгосрочные сбережения (подушка
безопасности)"}
}

def get_user_input():
    print("\u001b[1m\nВыберите финансовую цель:\u001b[0m")
    for num, goal in GOAL_CHOICES.items():
        print(f"{num}. {goal['desc']}")

    goal_num = int(input("\nНомер цели (1-5): "))
    print("\u001b[1m\nПАРАМЕТРЫ ВКЛАДА\u001b[0m")
    amount = int(input("Сумма для вклада (руб): "))
    term = int(input("Срок (1-36 мес., 0 - бессрочно): "))
    replenish = input("Нужно пополнение? (да/нет): ").lower() == 'да'
    withdrawal = input("Нужно снятие? (да/нет): ").lower() == 'да'
    print("Выплата процентов:")
    print("1 - Ежемесячно")
    print("2 - В конце срока")
    print("3 - Без разницы")
    payout = input("Ваш выбор (1-3): ")

    return {
        'goal': GOAL_CHOICES[goal_num]['name'],
        'goal_desc': GOAL_CHOICES[goal_num]['desc'],
        'amount': amount,
        'term': term,
        'replenish': replenish,
        'withdrawal': withdrawal,
        'payout': payout
    }

def calculate_score(deposit, user, max_rate):
    score = 0

    # 1. Совпадение цели (30 баллов, если совпадает)
    if deposit['goal'] == user['goal']:
        score += 30
```

```
    # 2. Ставка (30% веса)
    score += (deposit['rate'] / max_rate) * 30

    # 3. Срок (20% веса)
    if deposit['term_max'] == 0 or user['term'] <= deposit['term_max']:
        score += 20

    # 4. Пополнение (10% веса)
    if deposit['replenishable'] == user['replenish']:
        score += 10

    # 5. Снятие (10% веса)
    if deposit['withdrawal'] == user['withdrawal']:
        score += 10

    return round(score, 1)

def process_deposits(args):
    deposits_subset, user_data, max_rate = args
    results = []
    for _, deposit in deposits_subset.iterrows():
        if user_data['amount'] >= deposit['min_sum'] * 0.8:
            score = calculate_score(deposit, user_data, max_rate)
            results.append({
                'name': deposit['name'],
                'rate': deposit['rate'],
                'min_sum': deposit['min_sum'],
                'score': score,
                'goal': deposit['goal'],
                'needs_extra': user_data['amount'] < deposit['min_sum']
            })
    return results

def recommend_deposit_parallel(user_data, num_processes=4):
    max_rate = deposits['rate'].max()

    total_deposits = len(deposits)
    chunk_size = total_deposits // num_processes
    chunks = []

    for i in range(num_processes):
        start_idx = i * chunk_size
        end_idx = (i + 1) * chunk_size if i < num_processes - 1 else
total_deposits
        chunks.append((deposits.iloc[start_idx:end_idx], user_data,
max_rate))

    with mp.Pool(num_processes) as pool:
        results = pool.map(process_deposits, chunks)

    all_recommendations = []
    for chunk_result in results:
        all_recommendations.extend(chunk_result)
```

```
    all_recommendations.sort(key=lambda x: x['score'], reverse=True)
    return all_recommendations


def print_recommendations(rated_deposits, user_data):
    if not rated_deposits:
        print("\nНет подходящих вкладов")
        return

    print(f"\u001b[1m\nРЕКОМЕНДАЦИИ\u001b[0m")

    best = rated_deposits[0]
    if best['needs_extra']:
        missing = best['min_sum'] - user_data['amount']
        print(f"1. Совет: Доплатите {missing} ₽, чтобы выбрать вклад:
\u001b[1m{best['name']}\u001b[0m")
        print(f"   Ставка: {best['rate']}% | Подходит Вам на:
{best['score']}/100")

        for deposit in rated_deposits:
            if not deposit['needs_extra']:
                print(f"2. Без доплат:
\u001b[1m{deposit['name']}\u001b[0m")
                print(f"   Ставка: {deposit['rate']}% | Подходит Вам на:
{deposit['score']}/100")
                break
    else:
        print(f"1. {best['name']}")
        print(f"   Ставка: {best['rate']}% | Подходит Вам на:
{best['score']}/100")

def main():
    user_data = get_user_input()

    start_time = time.time()

    recommendations = recommend_deposit_parallel(user_data,
num_processes=4)

    end_time = time.time()
    execution_time = end_time - start_time

    print_recommendations(recommendations, user_data)
    print(f"\n\u001b[1mВремя выполнения программы: {execution_time:.4f}
секунд\u001b[0m")

if __name__ == "__main__":
    mp.freeze_support()
    main()
```

СПОСОБ РЕШЕНИЯ № 2

СБОР И ПОДГОТОВКА ДАННЫХ

Второй способ создания ИИ агента для рекомендаций вариантов вкладов основан на применении метода дерева решений. Он помогает определить, какие параметры (признаки) обычно важнее при выборе вклада.

Дерево решений – это алгоритм машинного обучения, который строит модель в виде древовидной структуры для принятия решений на основе входных данных.

Модель обучается на исторических данных, где X – это параметры (признаки), такие как ставка, срок, минимальная сумма, возможность пополнения, снятия, Y – выбранный вклад.

Был создан датасет с историческими данными о выборе вкладов пользователей, который включает:

- *минимальную сумму (min_sum);*
- *срок (term), мес.;*
- *возможность пополнения (replenish);*
- *возможность снятия (withdrawal);*
- *вклад, который в итоге был выбран пользователем (chosen).*

min_sum	term	replenish	withdrawal	chosen
100000	12	False	True	Лучший %
150000	24	True	True	СберВклад
50000	0	True	True	Накопительный счёт
200000	6	False	True	Лучший %
300000	36	True	False	СберВклад
70000	0	True	True	Накопительный счёт
120000	12	True	True	СберВклад
90000	18	False	True	Лучший %
250000	0	True	True	Накопительный счёт
180000	24	True	False	СберВклад
60000	12	False	True	Лучший %
350000	36	True	True	СберВклад
80000	0	True	True	Накопительный счёт
110000	6	False	True	Лучший %
400000	24	True	True	СберВклад

Рисунок 4 – Исторические данные о выборах вкладов пользователями

Так как модель работает только с числами, все текстовые значения преобразовываются (кодируются) в числа:

- True, False: 0, 1.
- Лучший %, СберВклад, Накопительный счет – 0, 1, 2.

ОБУЧЕНИЕ МОДЕЛИ

1. Все данные попадают в корневой узел.
2. Разбиение:

Для каждого признака (min_sum, term и т.д.) перебираются все возможные значения как пороги разбиения.

Например, для min_sum проверяются условия:

min_sum <= 100000 → левая ветвь,

min_sum > 100000 → правая ветвь.

Выбирается то разбиение, которое максимально разделяет данные (с минимальной ошибкой) по целевой переменной (вклад).

3. Критерии остановки:

- достигнута максимальная глубина (max_depth=6);
- в узле осталось мало данных (например, меньше 2 примеров).

4. Листья дерева:

В конечных узлах (листьях) возвращается предсказание – название вклада, которое чаще всего встречается в этой группе.

ПРИМЕНЕНИЕ МОДЕЛИ

После обучения модели мы можем использовать её для предсказания названия вклада для новых клиентов. В процессе предсказания модель сравнивает параметры каждого вклада с пороговыми значениями в узлах дерева и, следуя по дереву, получает предсказание для целевой переменной.

Таким образом:

1. модель строит дерево, выбирая лучшие параметры вклада для разбиения;

2. рекурсивно делит данные, пока не выполнится условие остановки (глубина дерева = 3 или мало данных в узле);

3. предсказывает, какой вклад подходит пользователю, проходя по дереву с новыми данными.

РЕАЛИЗАЦИЯ

Для разработки ИИ агента выбран язык программирования Python 3.13, среда разработки – PyCharm 2025.1.3.

Выбор библиотек

1. Pandas – для работы с таблицами (чтение csv-файла);
2. Scikit-learn – для машинного обучения:
   - DecisionTreeClassifier – для создания дерева решений;
   - LabelEncoder – для кодирования данных;
   - export_text – для текстового представления дерева.

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

```
C:\Users\Анастасия\PycharmProjects\PythonProject>python AI_deposit.py

1. Сумма для вклада (руб): 120000

2. Срок (1-36 мес., 0 - бессрочно): 30

3. Нужно пополнение? (да/нет): да
4. Нужно снятие? (да/нет): да

5. Выплата процентов:
   1 - Ежемесячно
   2 - В конце срока
   3 - Без разницы
Ваш выбор (1-3): 2

Дерево решений, использованное для рекомендации:

|--- replenish <= 0.50
|   |--- class: Лучший %
|--- replenish >  0.50
|   |--- term <= 6.00
|   |   |--- class: Накопительный счёт
|   |--- term >  6.00
|   |   |--- class: СберВклад


Рекомендуемый вклад: СберВклад

Условия вклада:
Гибкие условия (18%), с пополнением и снятием
```

Рисунок 5 – результаты тестирования ИИ агента

ЛИСТИНГ

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import export_text

history = pd.read_csv('data.csv', sep=';', encoding='utf-8-sig')

le = LabelEncoder()
history['chosen_deposit_encoded'] = le.fit_transform(history['chosen'])

X = history[['min_sum', 'term', 'replenish', 'withdrawal']]
y = history['chosen_deposit_encoded']

model = DecisionTreeClassifier(max_depth=6, random_state=42)
model.fit(X, y)

def get_user_input():

    amount = int(input("\n1. Сумма для вклада (руб): "))
    term = int(input("\n2. Срок (1-36 мес., 0 - бессрочно): "))
    replenish = input("\n3. Нужно пополнение? (да/нет): ").lower() ==
'да'
    withdrawal = input("4. Нужно снятие? (да/нет): ").lower() == 'да'
    print("\n5. Выплата процентов: ")
    print("   1 - Ежемесячно")
    print("   2 - В конце срока")
    print("   3 - Без разницы")
    payout_pref = int(input("Ваш выбор (1-3): "))

    return {
        'min_sum': amount,
        'term': term,
        'replenish': replenish,
        'withdrawal': withdrawal,
    }

def recommend_deposit(user_data):
    input_df = pd.DataFrame([user_data])

    print("\nДерево решений, использованное для рекомендации:")
    print(" ")
    tree_rules = export_text(model,
                             feature_names=['min_sum', 'term',
'replenish', 'withdrawal'],
                             class_names=le.classes_)
    print(tree_rules)
    print(" ")

       prediction = model.predict(input_df)
    return le.inverse_transform(prediction)[0]

user_data = get_user_input()
recommended_deposit = recommend_deposit(user_data)
```

```
print(f"Рекомендуемый вклад: {recommended_deposit}")

deposits_info = {
    "Лучший %": "Высокий процент (18%), без пополнения, снятие
разрешено",
    "СберВклад": "Гибкие условия (18%), с пополнением и снятием",
    "Накопительный счёт": "Бессрочный вклад (16%), пополнение и снятие"
}
```

ПРИЛОЖЕНИЕ А – ДОКУМЕНТАЦИЯ (СПОСОБ 1)

1. Назначение программы

Автоматизированный анализ доступных банковский вкладов и рекомендации наиболее подходящих вариантов.

2. Требование к системе

ОС: windows 10 и новее, Linux.

Python: версия 3.8 и новее

Память: минимум 512 МБ

3. Установка компонентов

Необходимые зависимости: pip install numpy, pandas.

4. Требования к входным данным

Файл deposits.csv, содержащий столбцы с информацией по вкладам:

- name - название вклада (текст);
- rate - ставка вклада (число);
- min_sum - минимальная сумма вклада (число);
- term_max - срок вклада (число);
- replenishabl - возможность пополнения (True/False)
- withdrawal - возможность снятия (True/False)
- goal - основная цель вклада (текст)

5. Запуск программы

Открыть командную строку, выполнить команду: python AI_deposit.py

6. Использование агента:

1 - выбрать финансовую цель;

2 - ввести ответы на вопросы агента (параметры желаемого вклада);

3 - получить рекомендации.

ПРИЛОЖЕНИЕ Б – ДОКУМЕНТАЦИЯ (СПОСОБ 2)

1. Назначение программы

Автоматизированный анализ доступных банковский вкладов и рекомендации наиболее подходящих вариантов на основе исторических данных о выборах вкладов пользователями.

2. Требование к системе

ОС: windows 10 и новее, Linux.

Python: версия 3.8 и новее

Память: минимум 512 МБ

3. Установка компонентов

Необходимые зависимости: pip install pandas, pip install scikit-learn.

4. Требования к входным данным

Файл data.csv, содержащий столбцы с информацией по вкладам:

- min_sum - минимальная сумма вклада (число);
- term - срок вклада (число);
- replenish - возможность пополнения (True/False)
- withdrawal - возможность снятия (True/False)
- chosen – выбранный в итоге вклад (текст)

5. Запуск программы

Открыть командную строку, выполнить команду: python AI_deposit.py

6. Использование агента:

1 - ввести ответы на вопросы агента (параметры желаемого вклада);

2 - получить рекомендации.