**СОДЕРЖАНИЕ**

[ЦЕЛЬ ЗАДАНИЯ 2](#_Toc202869707)

[СПОСОБ РЕШЕНИЯ № 1 2](#_Toc202869708)

[СБОР И ПОДГОТОВКА ДАННЫХ 2](#_Toc202869709)

[МАТЕМАТИЧЕСКАЯ МОДЕЛЬ 3](#_Toc202869710)

[РЕАЛИЗАЦИЯ 4](#_Toc202869711)

[МОДЕЛИРОВАНИЕ 4](#_Toc202869712)

[РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 6](#_Toc202869713)

[ЛИСТИНГ 8](#_Toc202869714)

[СПОСОБ РЕШЕНИЯ № 2 11](#_Toc202869715)

[СБОР И ПОДГОТОВКА ДАННЫХ 11](#_Toc202869716)

[ОБУЧЕНИЕ МОДЕЛИ 12](#_Toc202869717)

[ПРИМЕНЕНИЕ МОДЕЛИ 12](#_Toc202869718)

[РЕАЛИЗАЦИЯ 13](#_Toc202869719)

[РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 13](#_Toc202869720)

[ЛИСТИНГ 14](#_Toc202869721)

[ПРИЛОЖЕНИЕ А – ДОКУМЕНТАЦИЯ (СПОСОБ 1) 16](#_Toc202869722)

[ПРИЛОЖЕНИЕ Б – ДОКУМЕНТАЦИЯ (СПОСОБ 2) 17](#_Toc202869723)

# 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).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

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:

Введенная пользователем сумма ≥ 0,8× 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:
   1. 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;
   2. если у вклада есть максимальный срок (*term\_max>0*), то вклад получает 20 баллов, если:

*желаемый срок пользователя ≤ term\_max вклада*

* 1. в остальных случаях вклад получает 0 баллов по этому параметру.

1. **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.

1. **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).

## М**ОДЕЛИРОВАНИЕ**

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

Изображение выглядит как текст, снимок экрана, чек, документ

Автоматически созданное описание

## РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

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

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

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

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

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 Х – 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).*

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

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).

1. Критерии остановки:

* достигнута максимальная глубина (max\_depth=6);
* в узле осталось мало данных (например, меньше 2 примеров).

1. 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

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

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.

1. **Требование к системе**

ОС: windows 10 и новее, Linux.

Python: версия 3.8 и новее

Память: минимум 512 МБ

1. **Установка компонентов**

Необходимые зависимости: pip install numpy, pandas.

1. **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)

1. **Запуск программы**

Открыть командную строку, выполнить команду: python AI\_deposit.py

1. **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.

1. **Требование к системе**

ОС: windows 10 и новее, Linux.

Python: версия 3.8 и новее

Память: минимум 512 МБ

1. **Установка компонентов**

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

1. **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)

1. **Запуск программы**

Открыть командную строку, выполнить команду: python AI\_deposit.py

1. **Использование агента:**

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

2 - получить рекомендации.