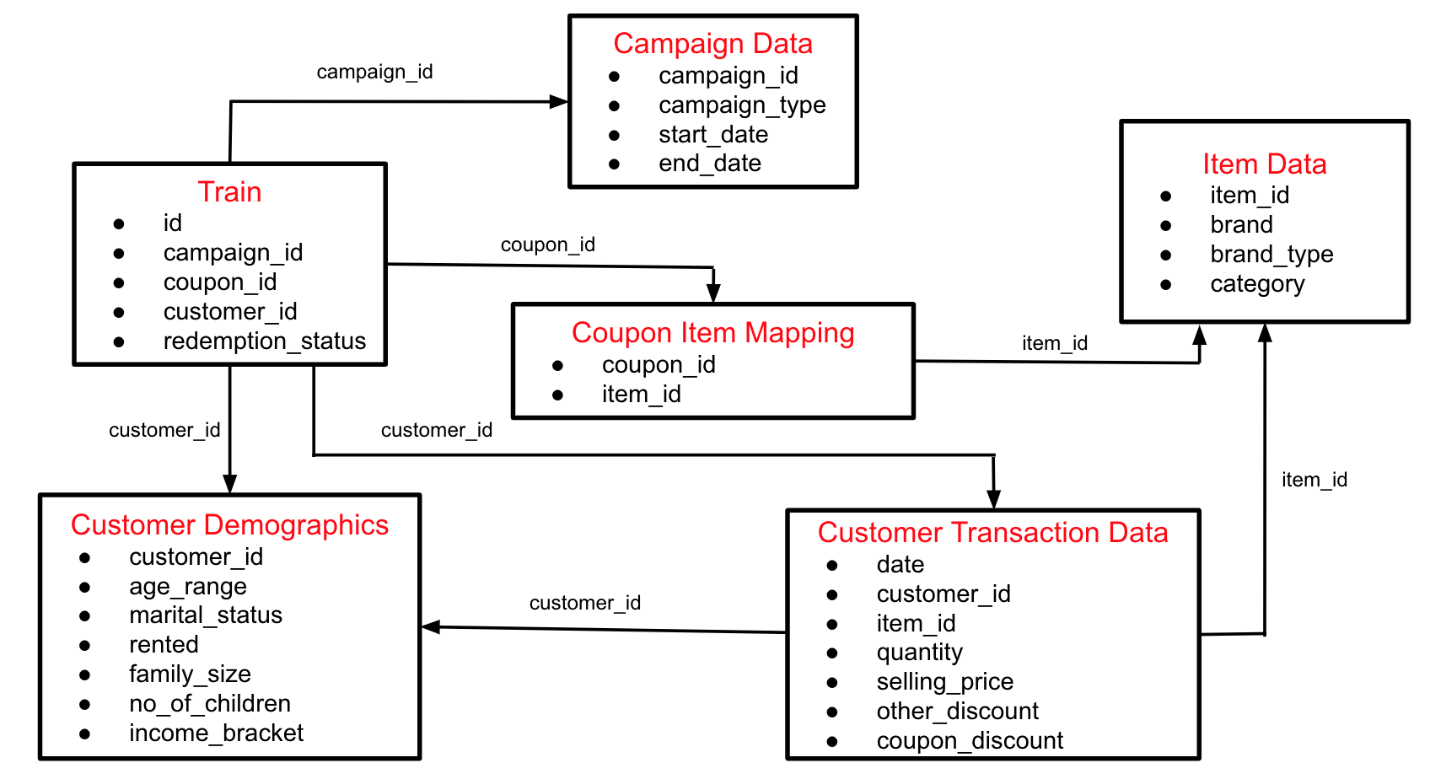
**Report**

This project was about figuring out how to predict the next best marketing action for customers using reinforcement learning. The dataset wasn’t small, it had campaigns, coupons, transactions, items, and customer demographics. I had to clean it, merge it, and then try different RL algorithms like multi-armed bandits and Q-learning

**The data set I used:**

I used the **Predicting Coupon Redemption dataset** from Kaggle. The dataset is composed of several related tables that describe campaigns, coupons, customers, items, and transactions.



train: links customers to campaigns and coupons

campaign data: has the campaign type and start/end dates

customer demographics: tells me age range, marital status, family size, income bracket

customer transaction data: all the purchases customers made (date, item, quantity, price, discounts)

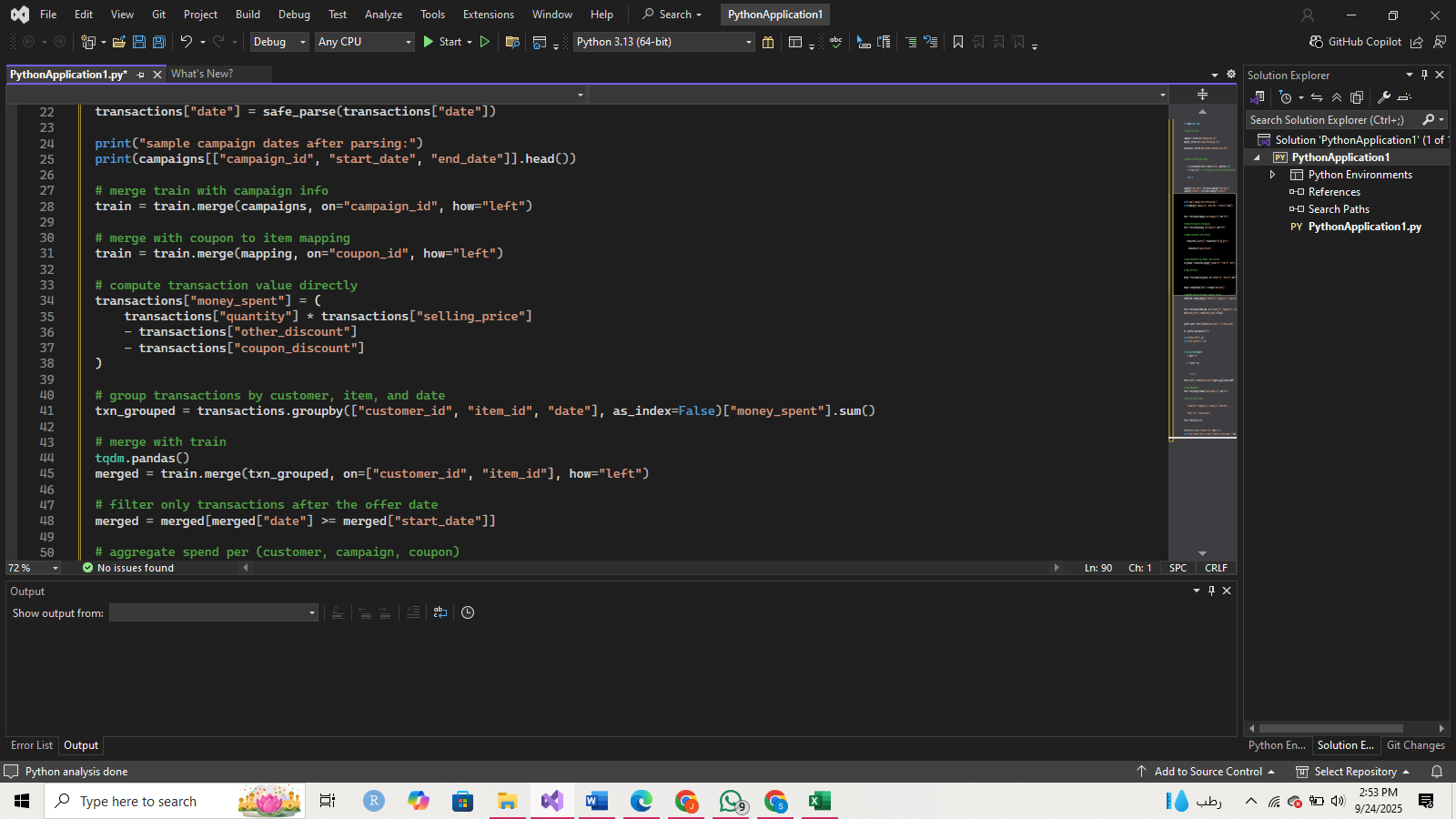
coupon item mapping: tells which items are linked to which coupons

item data: gives brand, brand\_type, and category for each item

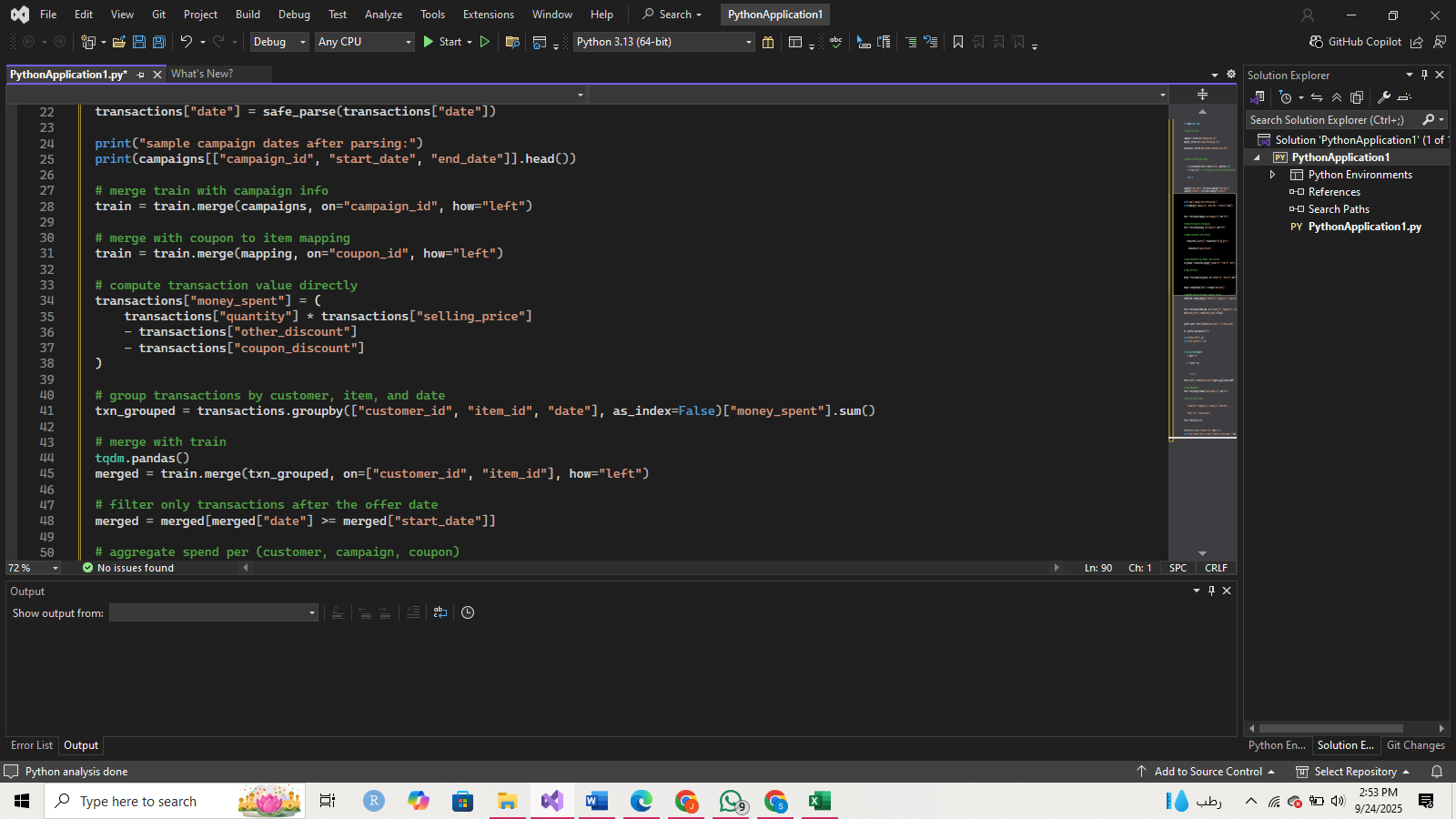
**Getting the data right:**

To prepare the dataset i combined the tables step by step:

* merged train with campaign data on campaign\_id so i know when offers were shown.

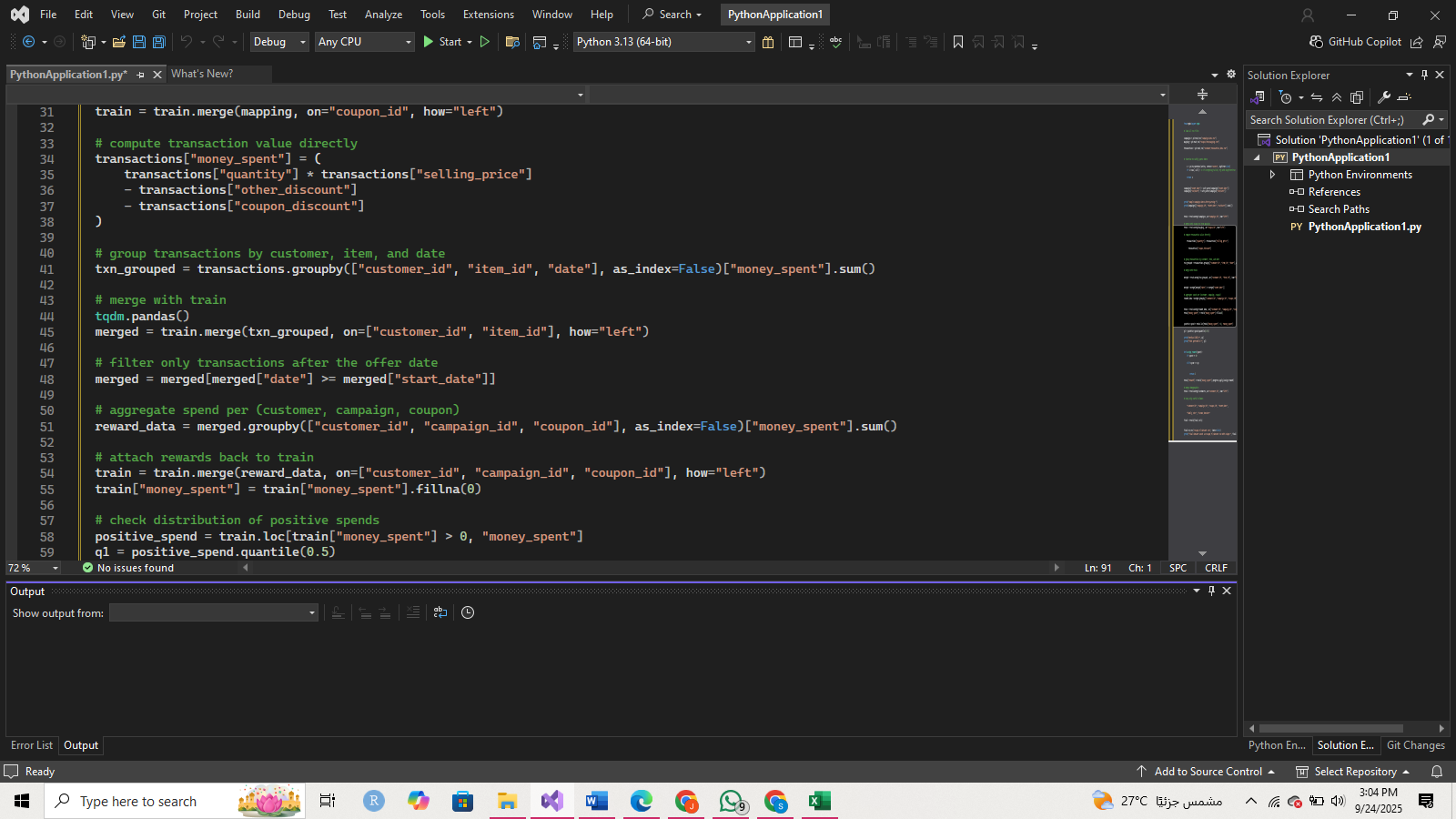


* then with coupon item mapping and item data so i can know the brand\_type and category of the couponed items

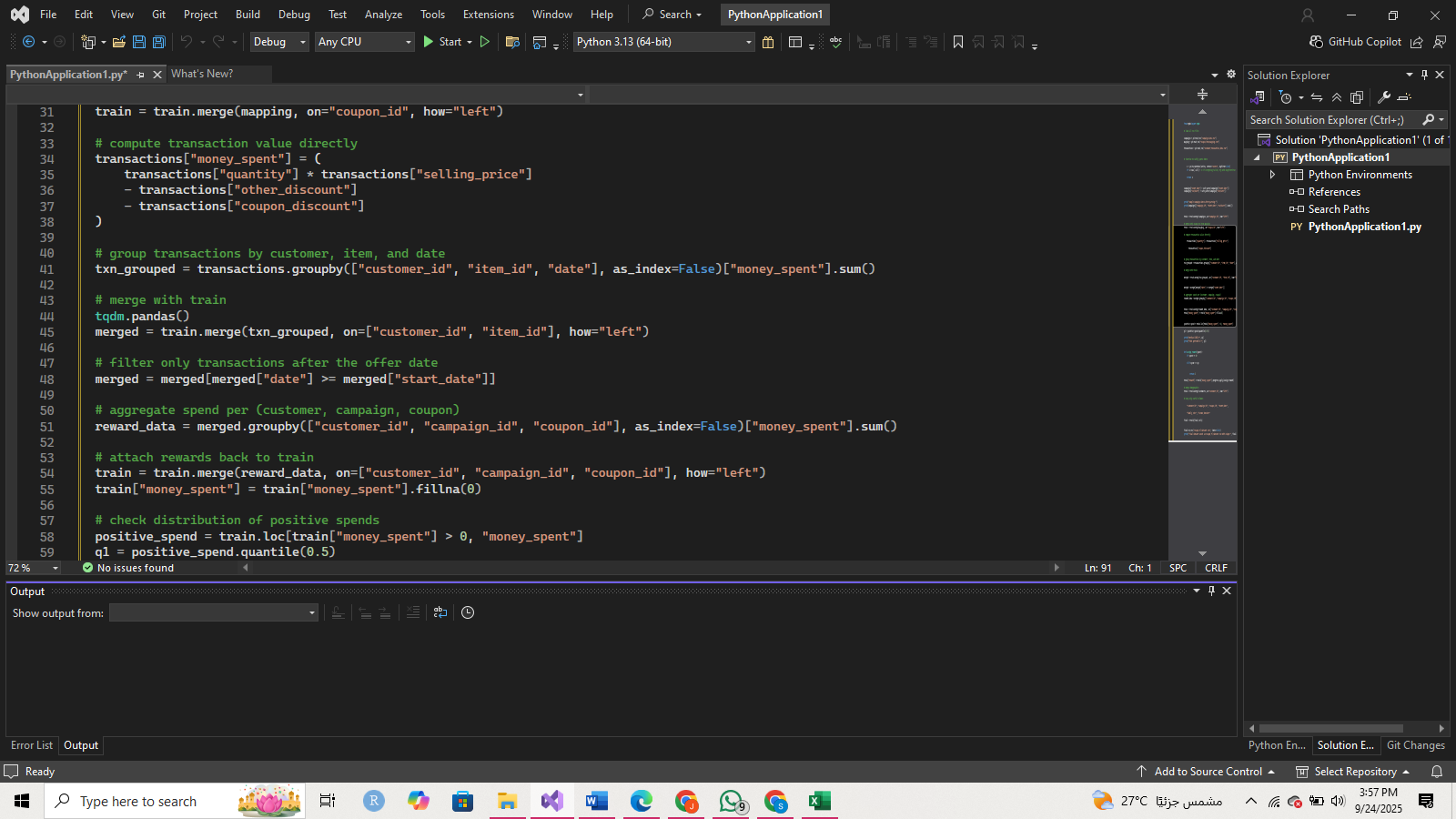


* I then prepped the transaction table before merging it with the rest:

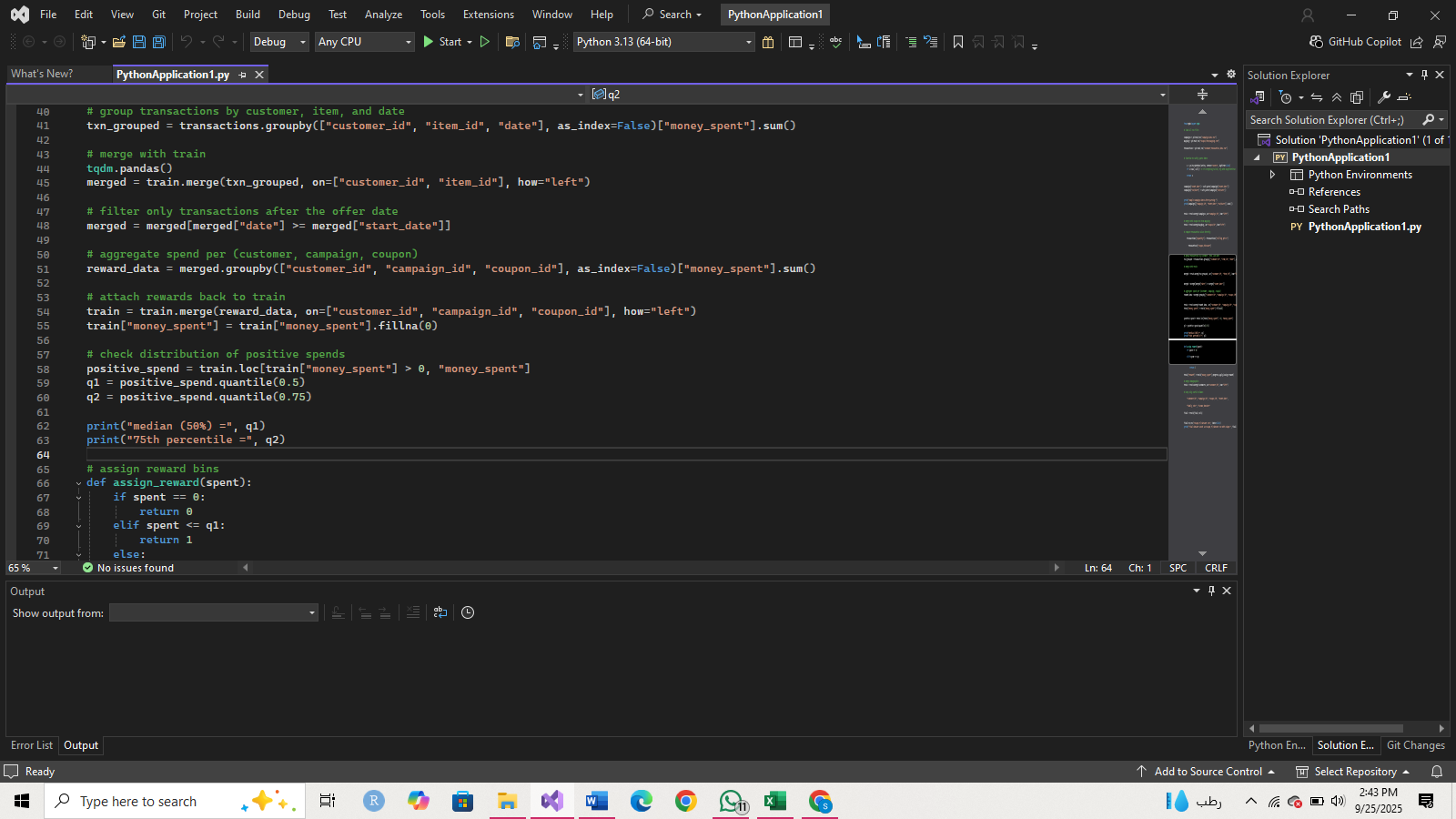
computed money spent in transactions, this creates a new column money\_spent in the transactions table by multiplying quantity and selling\_price, then subtracting discounts.



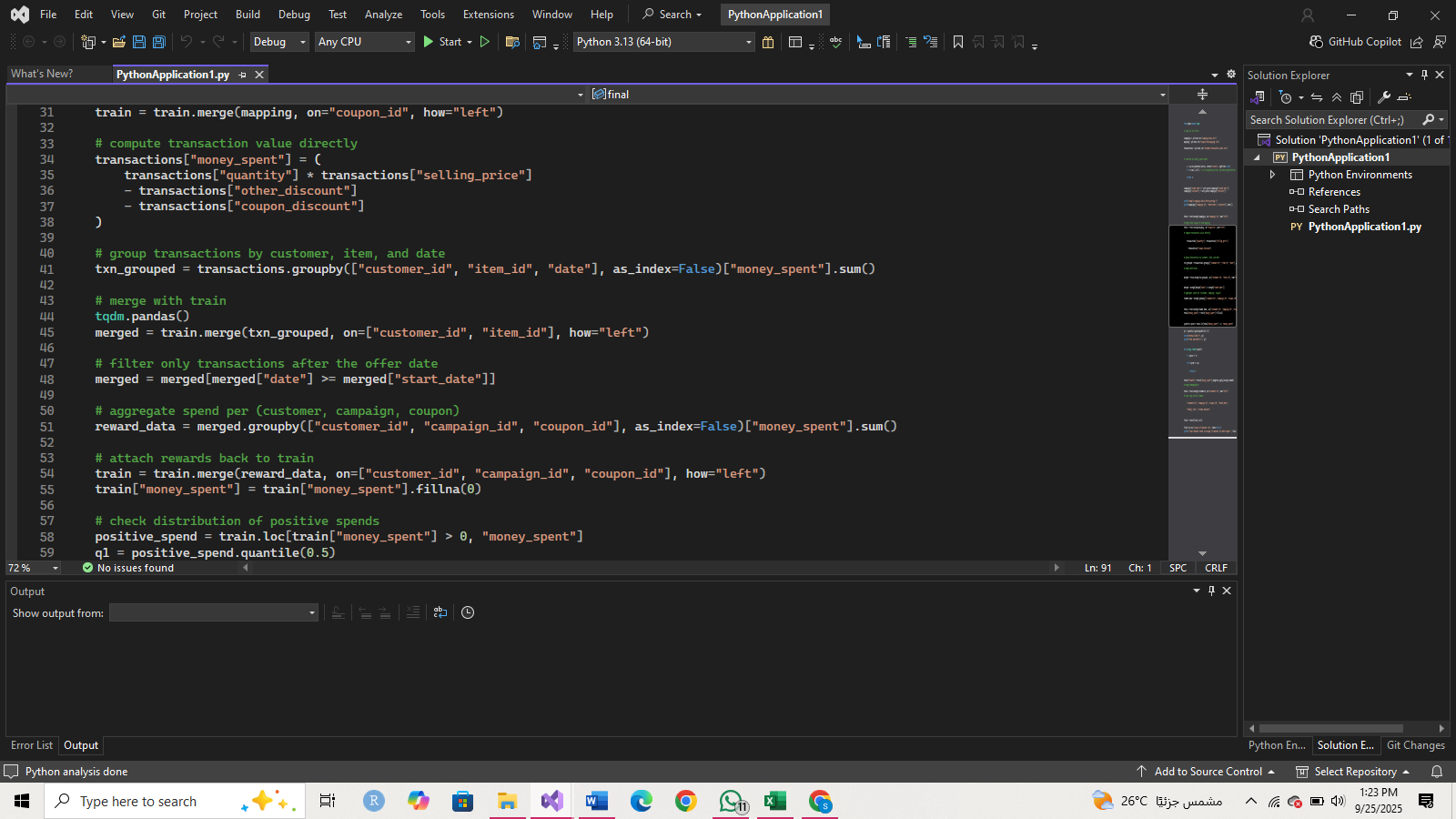
merged transactions with train, this joins the aggregated transactions with the train data so each customer–coupon entry has matching transaction data for the same item.



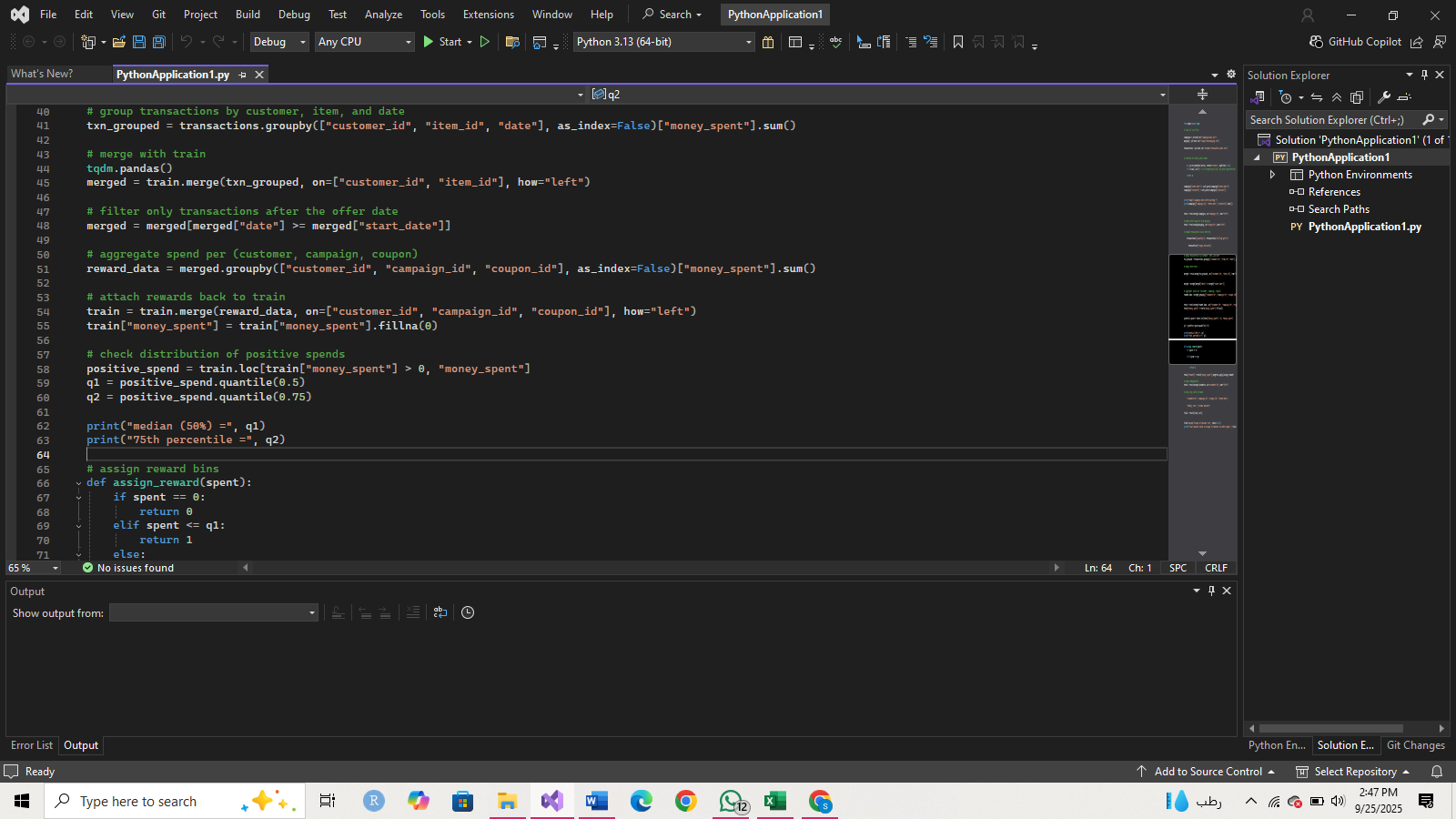
* merged with the train table, so every row has a money\_spent value. Missing values (no spend) are filled with 0s.



* filtered only transactions after offer date so we measure purchases only after the offer was shown (to clearly define rewards)



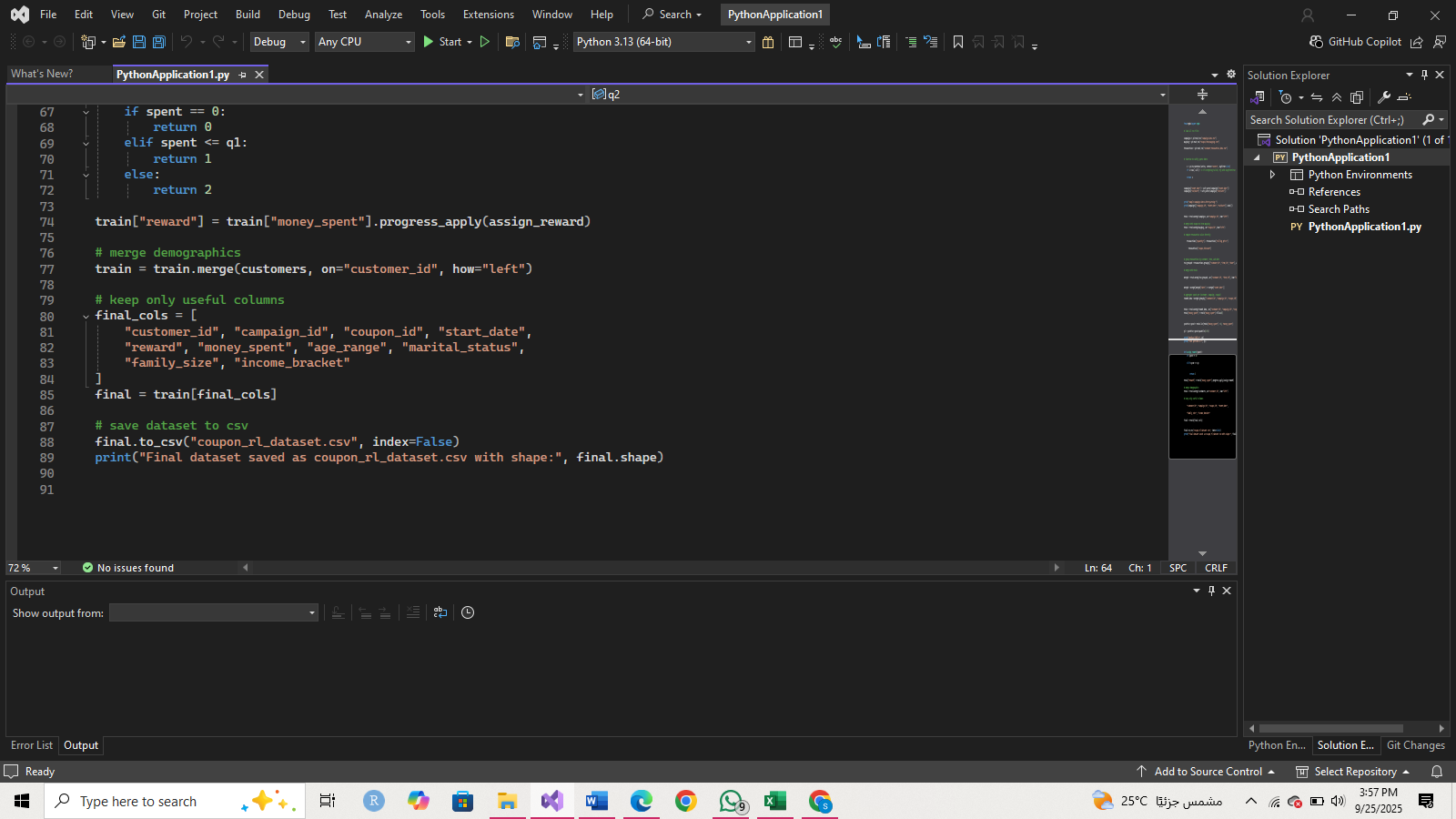
* calculated spend distribution this calculates the median (50%) and 75th percentile of all positive spending values that way we can bin the rewards.



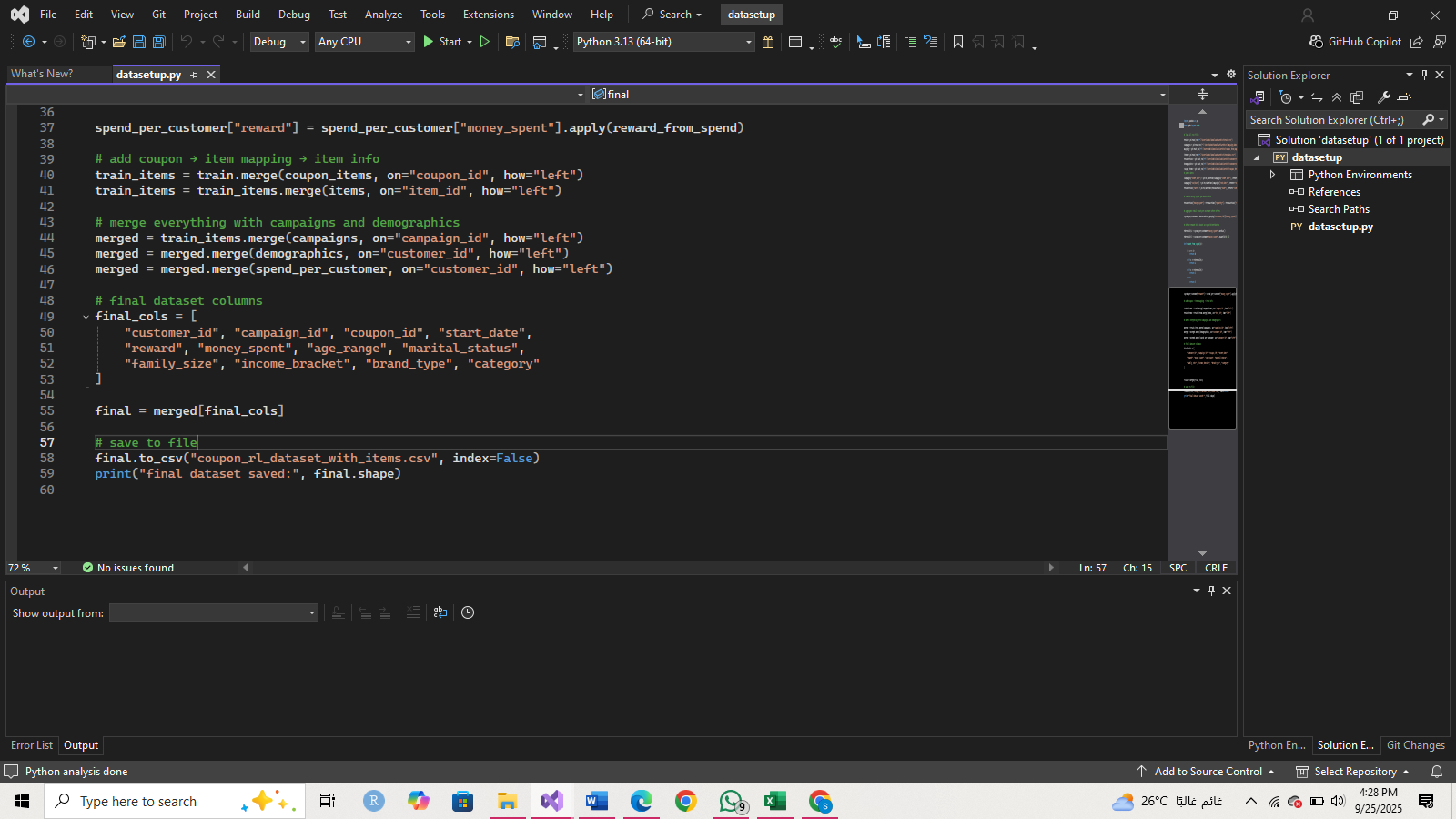
* assigned reward bins to create the reward column based on how much the customer spent.

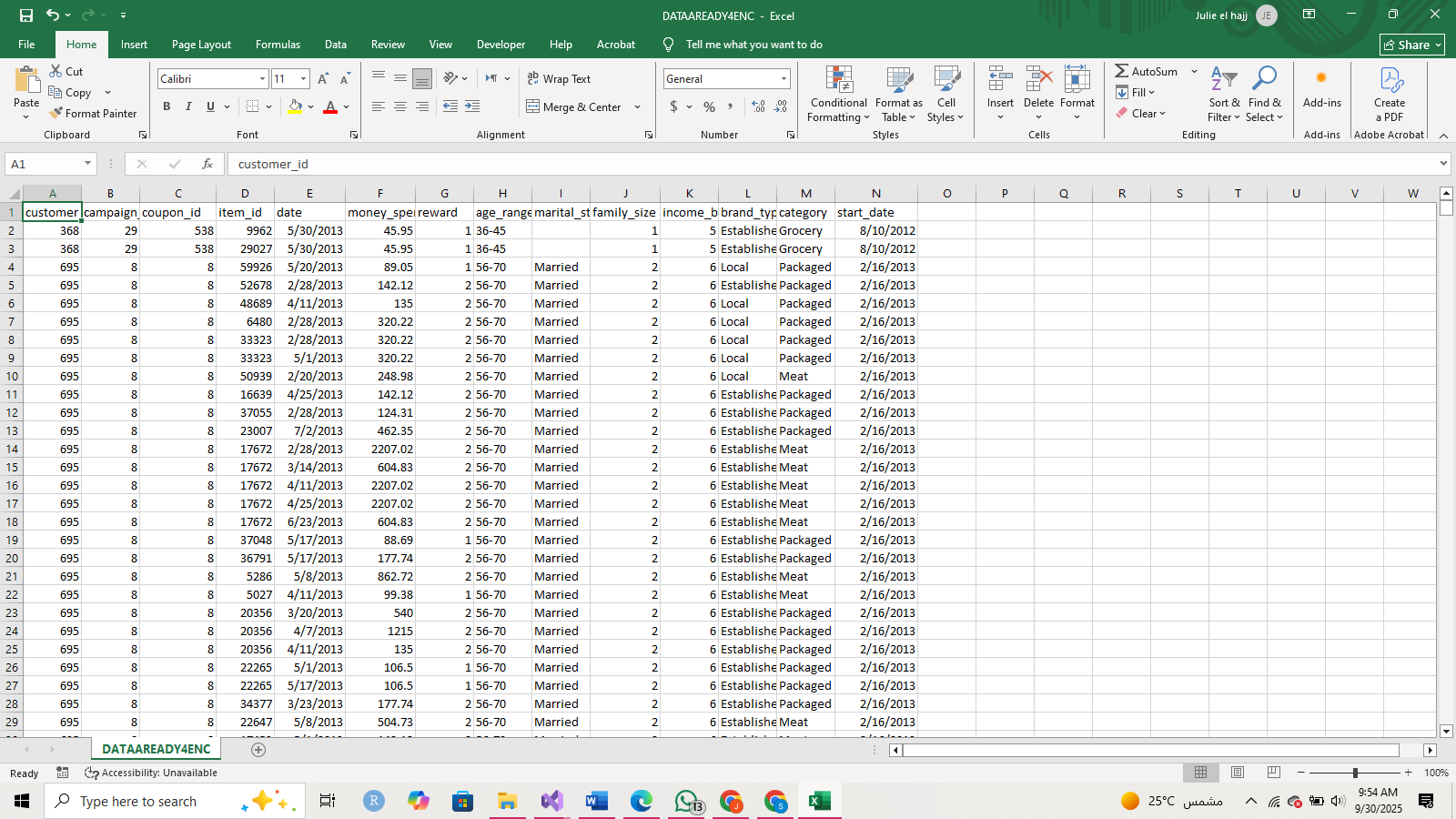


* merged costumer demographics to attach customer demographic info (age, marital status, family size, income) to the train table.



* Kept only useful columns

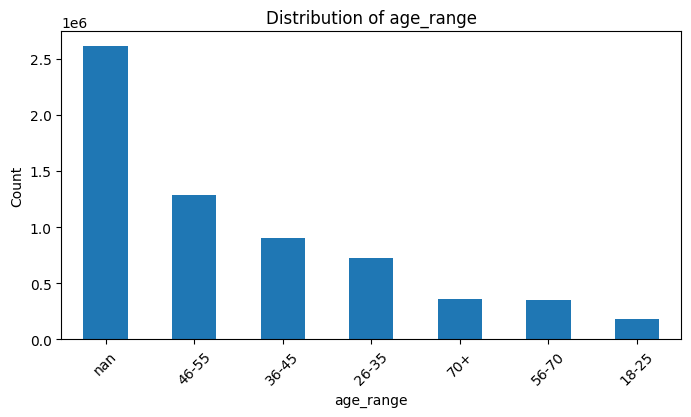




**Preprocessing & Encoding**

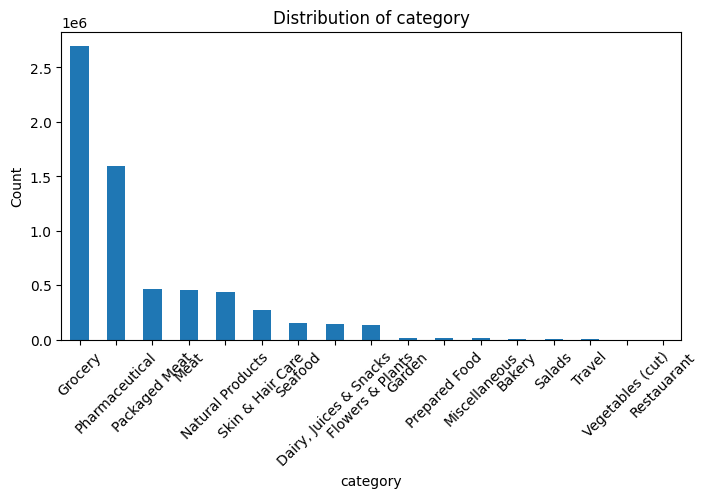
I first plotted the distributions of all categorical features (age\_range, marital\_status, brand\_type, and category) to better understand how the data is spread. These charts guided my preprocessing choices.

* Age:



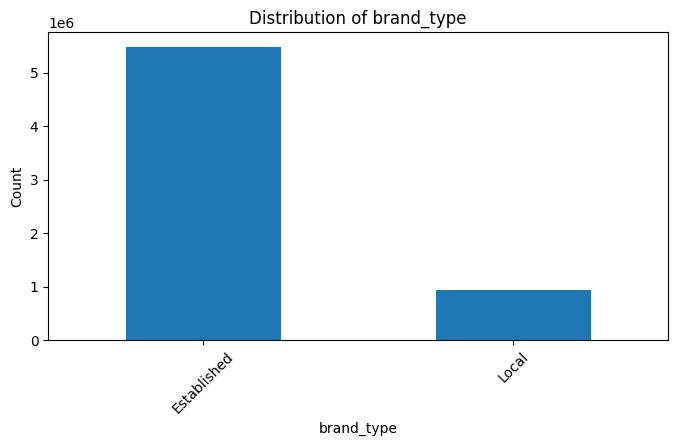
We can see that most customers fall in the 46–55, 36–45, and 26–35 groups, but there are also a lot of missing values (NaN). Instead of dropping these rows, I will treat “unknown” as its own category, because deleting them would mean losing too much data.

Category:



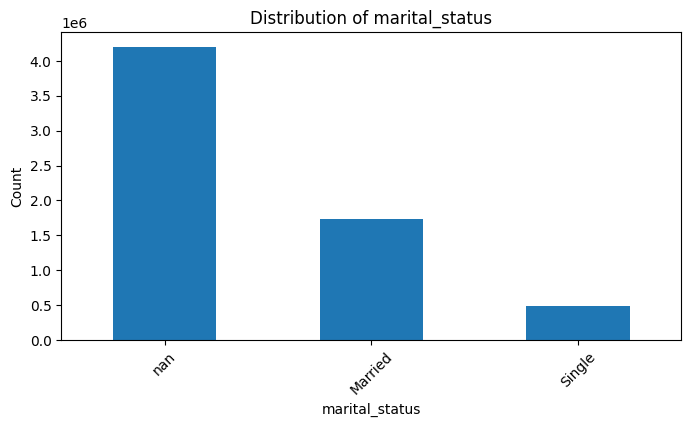
To make these categorical features usable for reinforcement learning, I applied one-hot encoding. This means I created a new column for each category, and each row is marked with a 1 if the customer belongs to that category and 0 otherwise. For example, if the coupon is for *Grocery*, then category\_Grocery = 1 and all the other category columns are 0.

The same logic was applied to other categorical variables like marital\_status, age\_range, and brand\_type. This transformation ensures that the RL algorithms can read these variables as numerical inputs while preserving the fact that the categories are distinct and not ordinal.



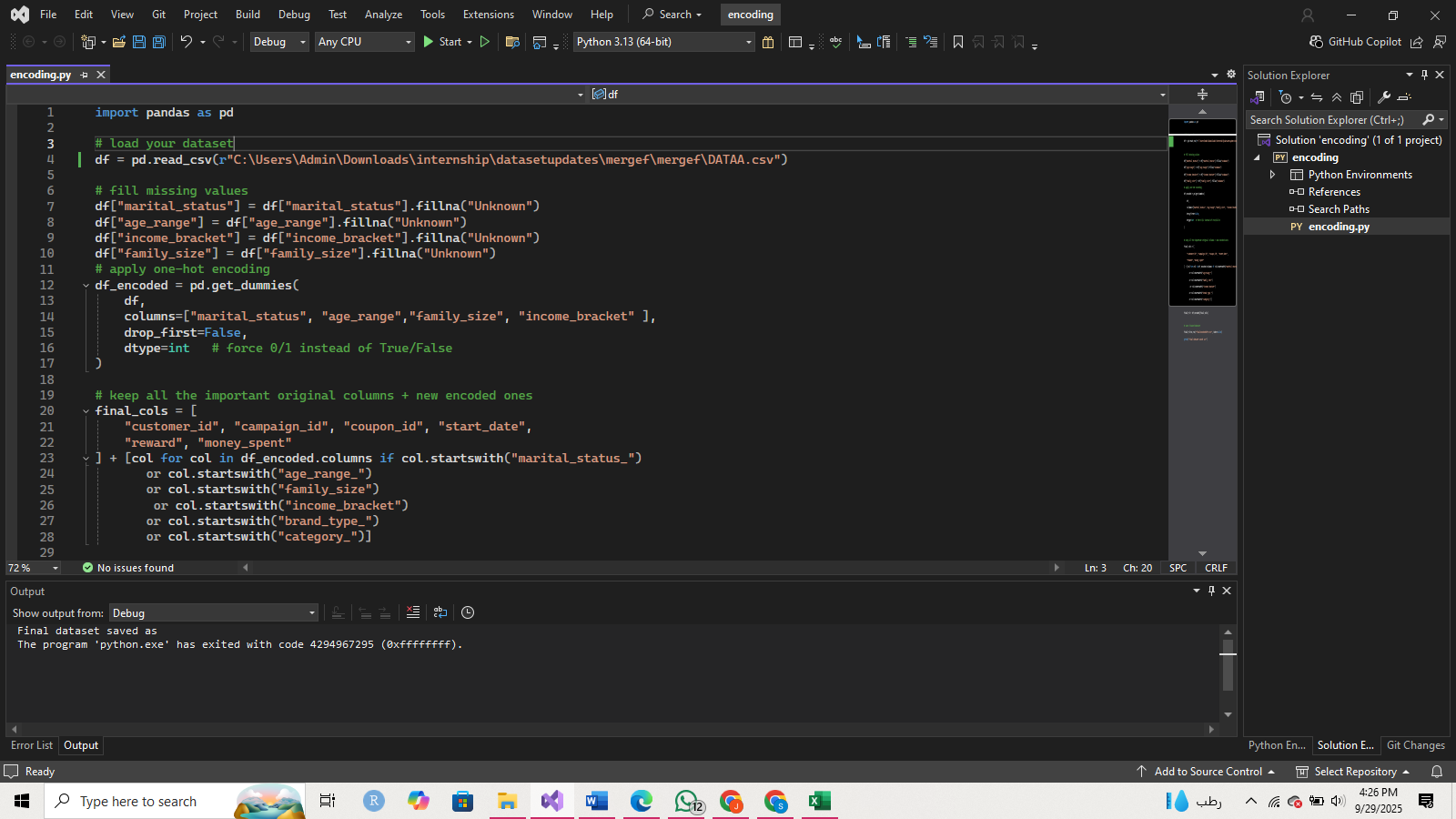
2 brand types: local or established

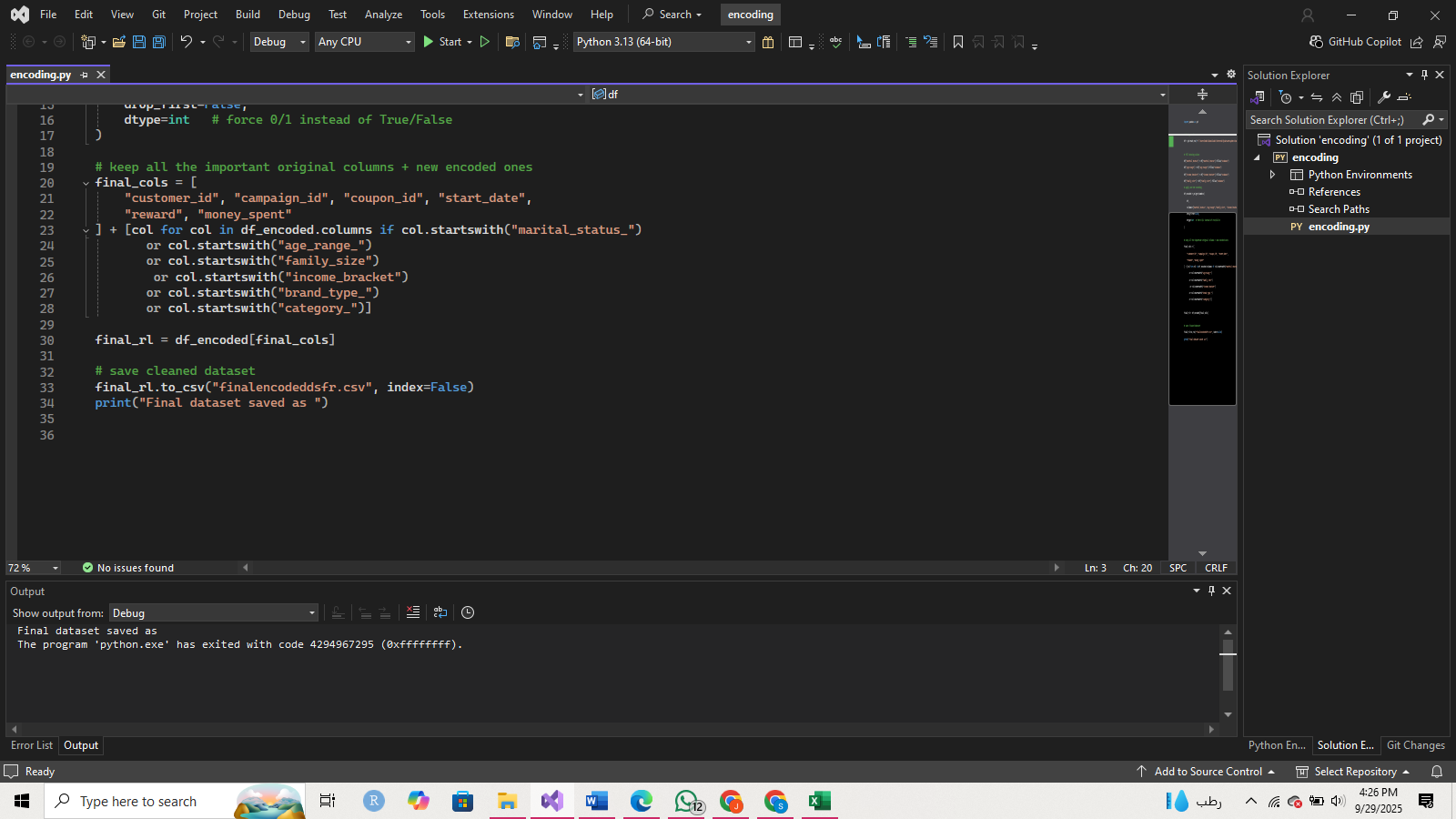
One-hot encoding.

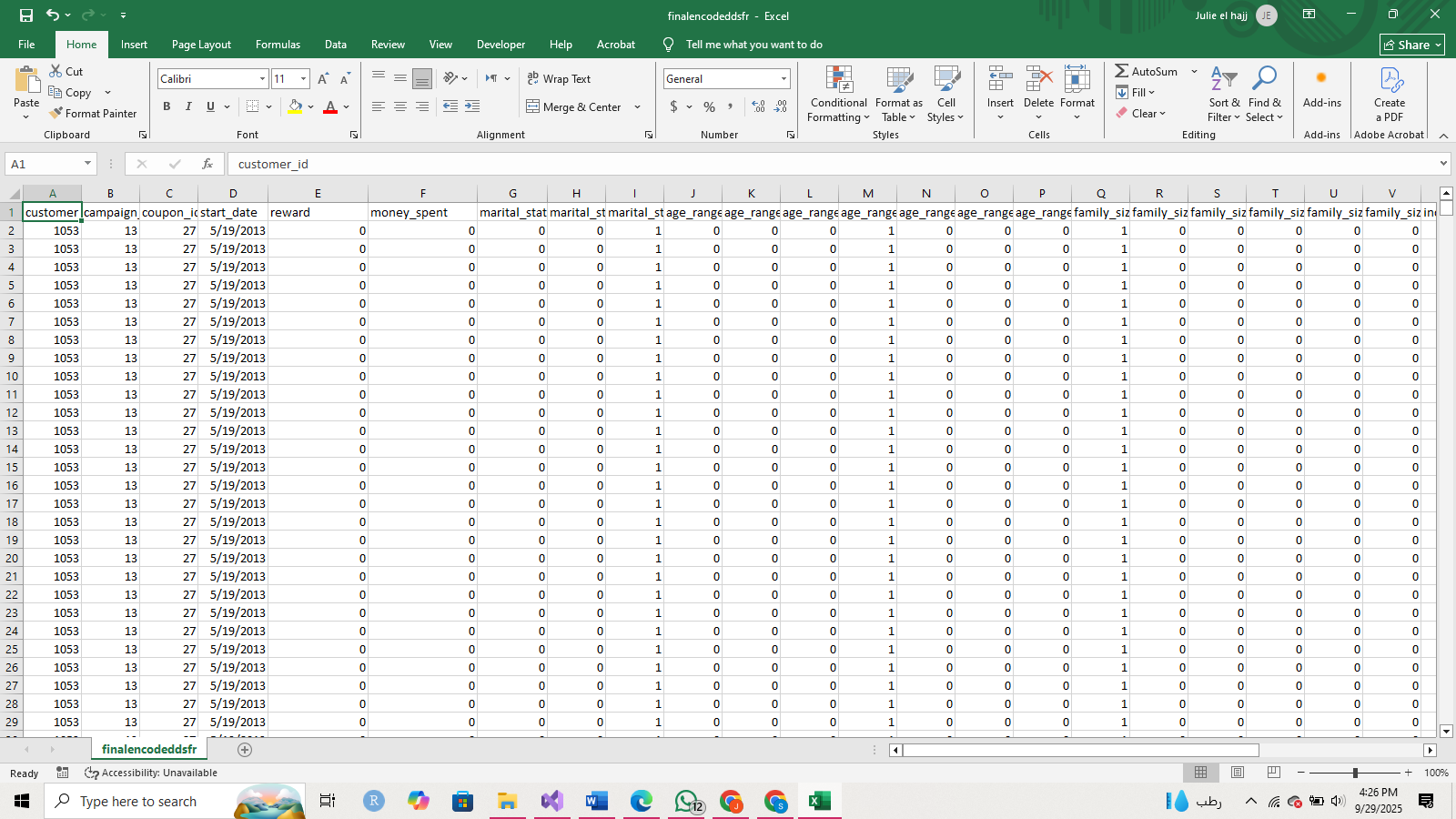


gonna replace nans with unknown.

one-hot encoding.







**RL Approaches**

***Non-Contextual Multi-Armed Bandit (NCMAB****)*

* Actions: available coupons (coupon\_id).
* Reward: binned spending after the coupon (0 = no spend, 1 = low spend, 2 = high spend).
* Environment description: At each round, the agent chooses a coupon to recommend.

The reward only depends on the coupon chosen, not the customer.

Over time, the agent should learn which coupons tend to give higher rewards *on average*.

* Goal: Find the coupon with the best expected redemption/spend rate across all customers.

**Contextual Multi-Armed Bandit (CMAB)**

* Context: customer features at the time of decision.

age\_range

marital\_status

family\_size

income\_bracket

* Actions: which of the available coupons (coupon\_id) to send.
* Reward: binned spending outcome (0–1–2).
* Goal: Personalize coupon selection by learning which actions work best for which types of customers.
* The agent learns that some coupons work better for certain demographic groups (for ex married 36–45 with income bracket X respond well to pharma coupons).

**3. Markov Decision Process (MDP) with Q-learning**

* State: sequential representation of the customer:

Demographics (same as CMAB).

Time step within the campaign (day relative to start\_date).

Money spent in the current state and money spent in the next state.

* Actions:

Which coupon to send next.

* Reward: based on the difference between the money spent in the current state and the next state.
* Next State:

Same customer, time advanced by one step.

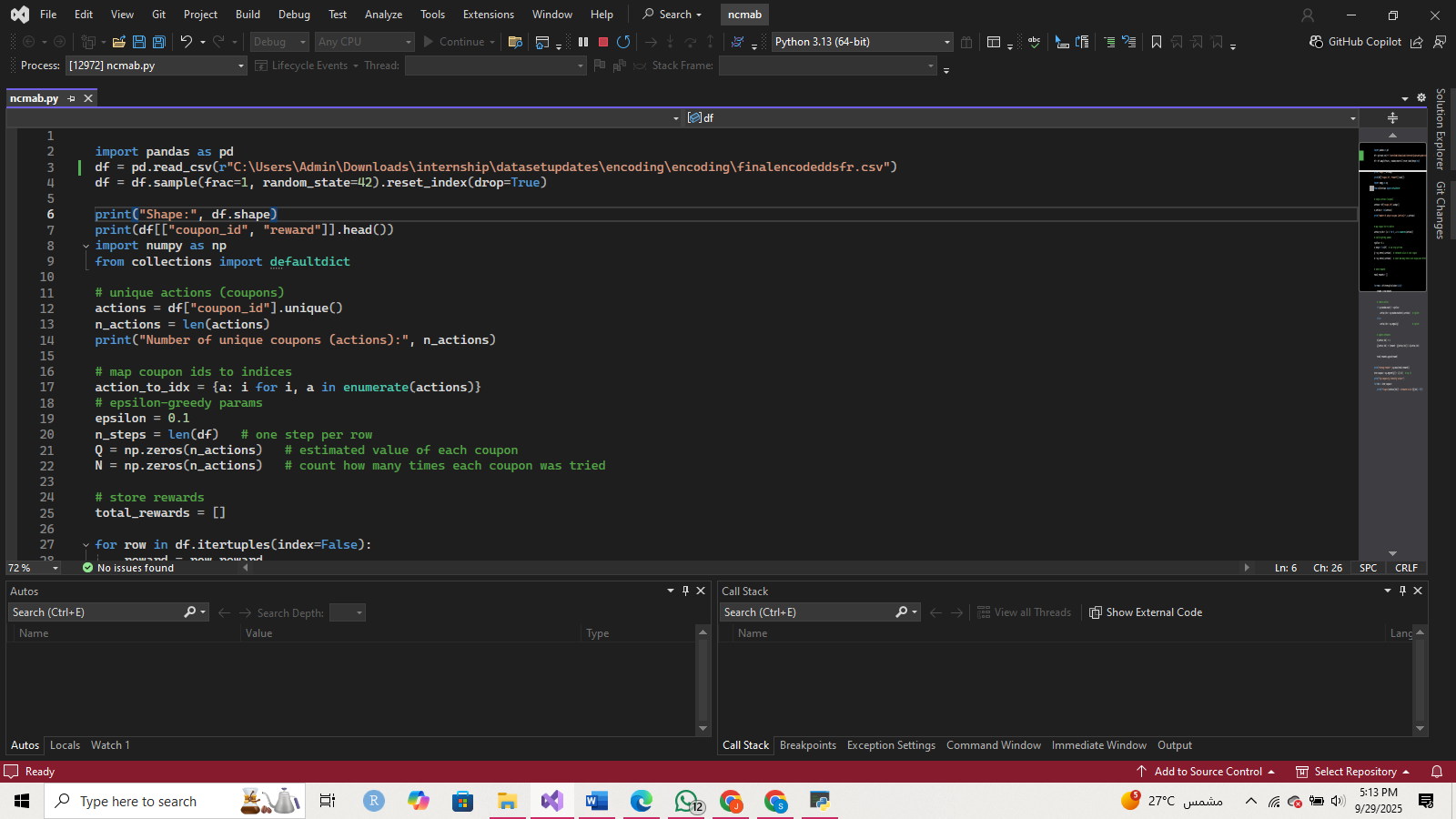
Money spent at this time.

* Environment description: The agent makes a sequence of coupon decisions for a customer. Each action affects both immediate spend (reward) and the customer’s future state (likelihood of buying again, time left in campaign).
* Goal: Maximize total cumulative reward across the campaign, not just single-step rewards.

**Application**

NCMAB:

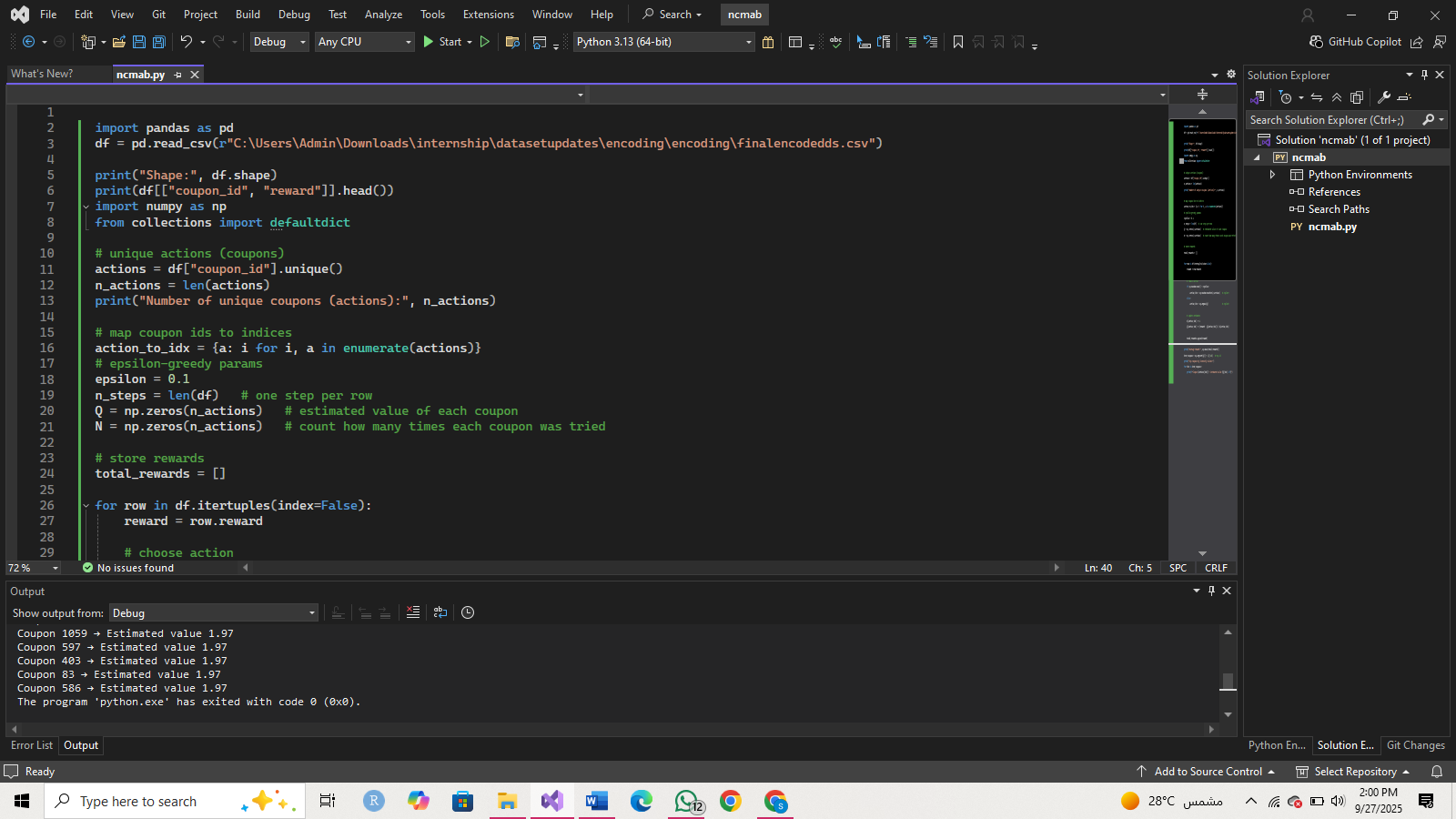
Loaded the encoded data set

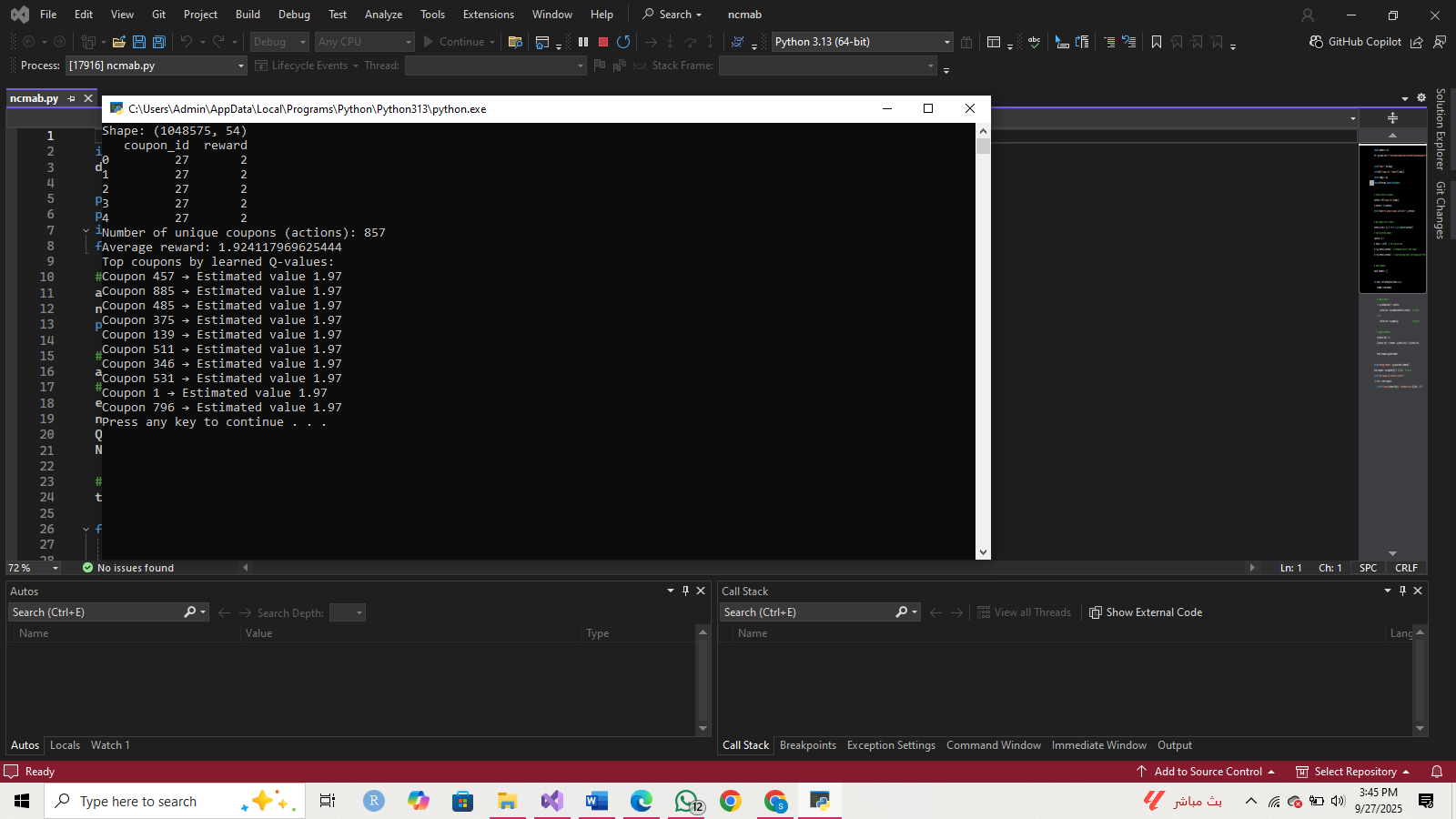


and took a look at it



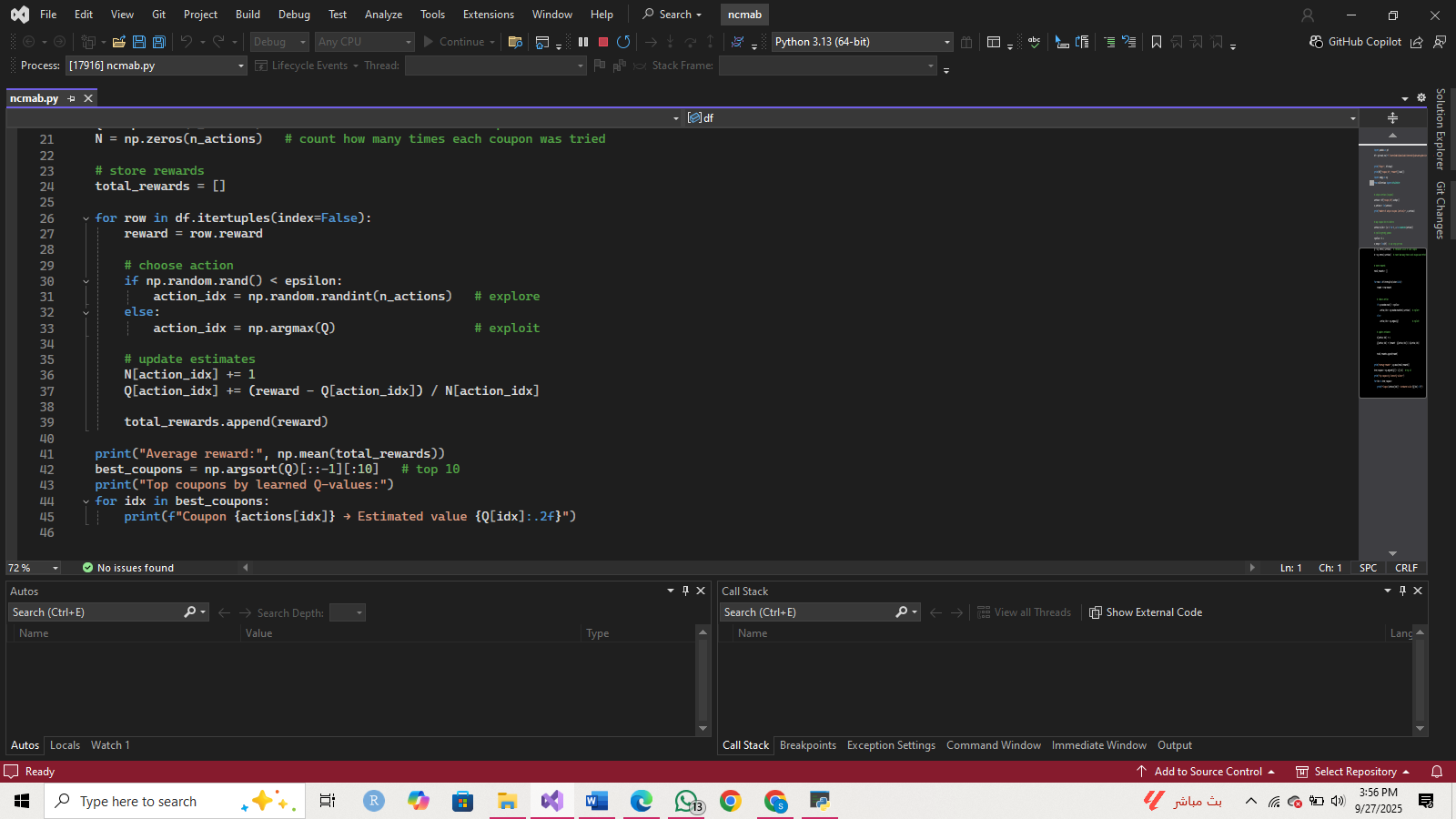
Defined the environment



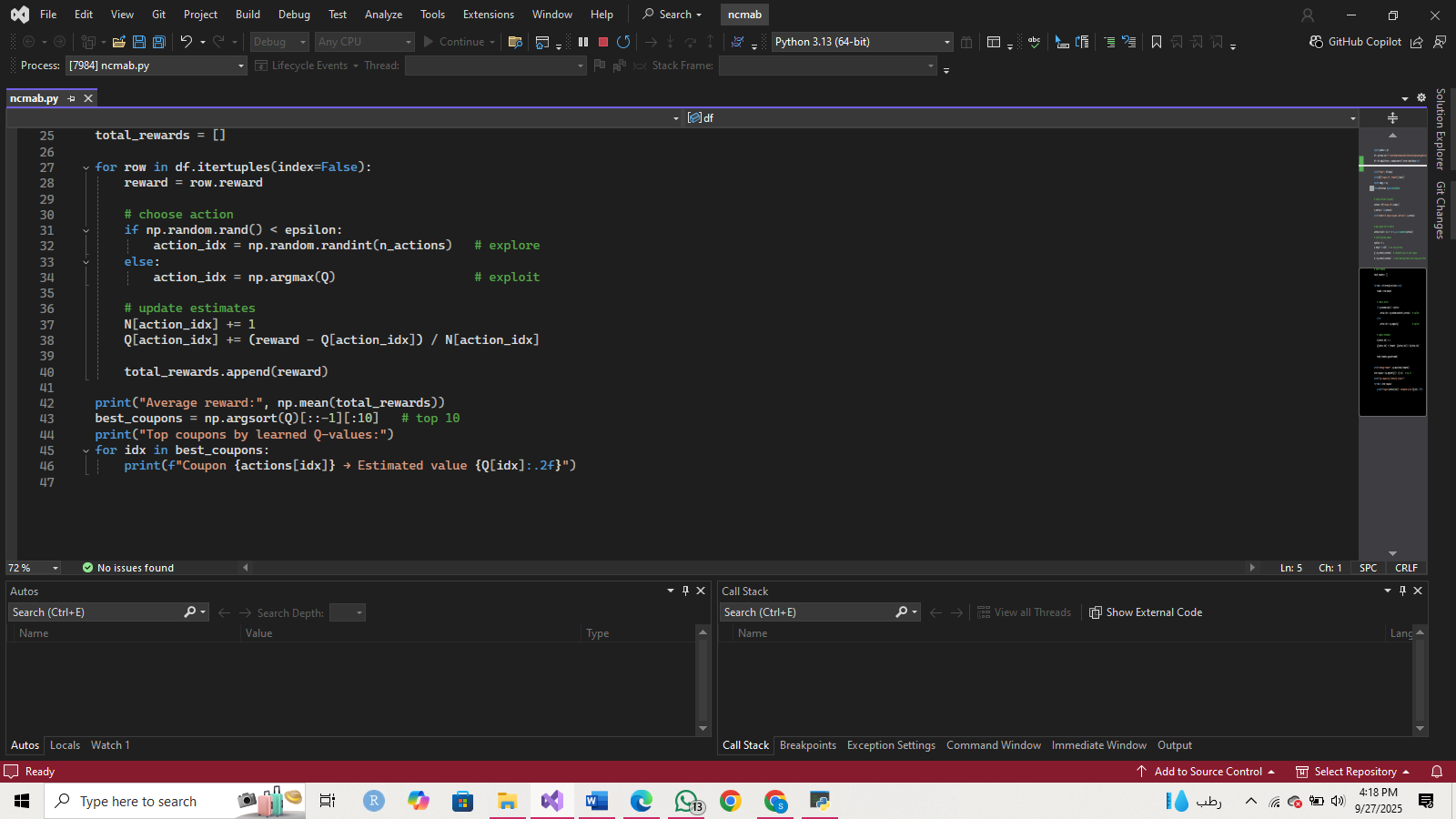


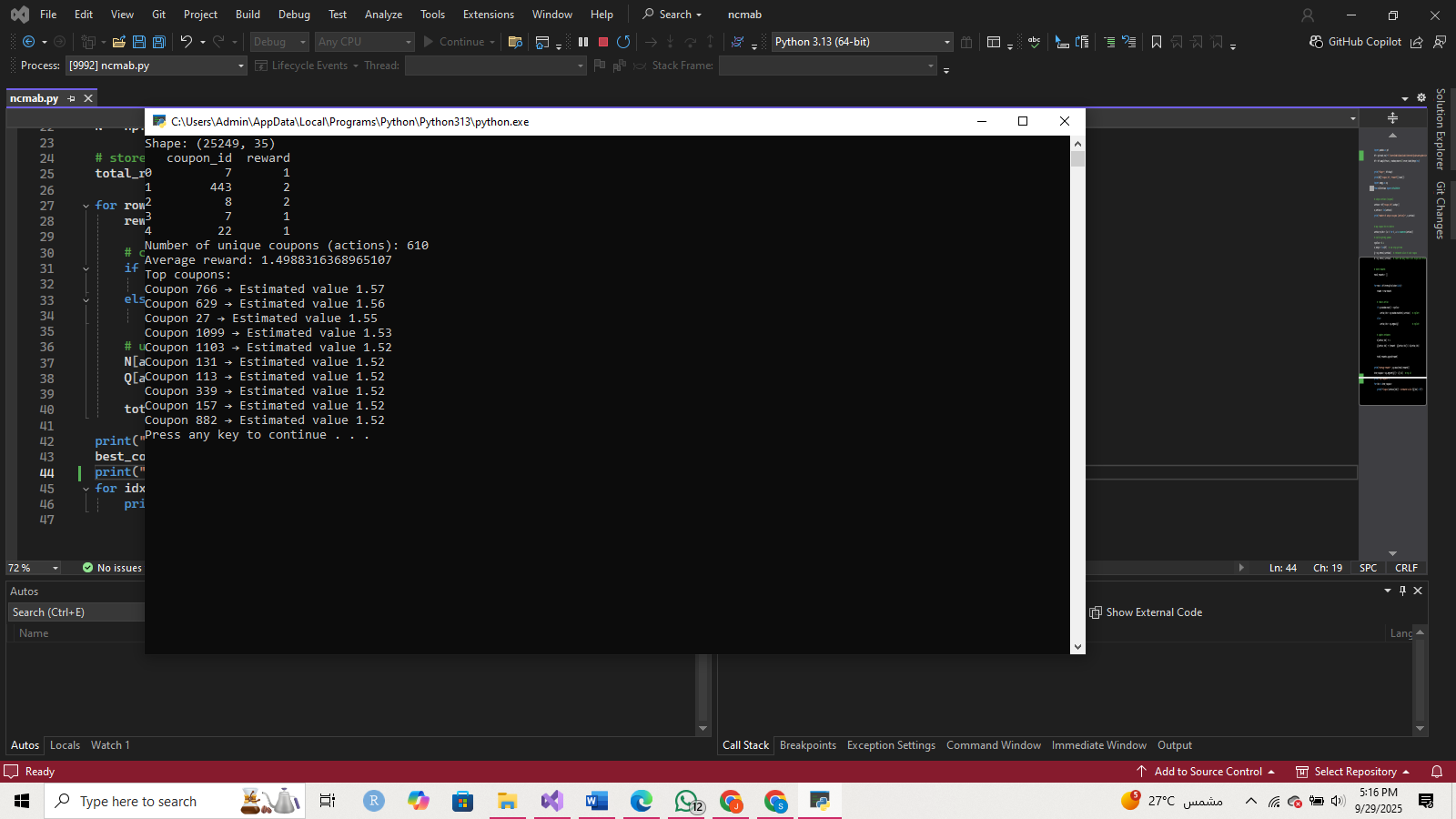
Implement epsilon-greedy bandit

This will explore randomly with prob epsilon, otherwise exploit the best-known coupon.



See which coupons are best

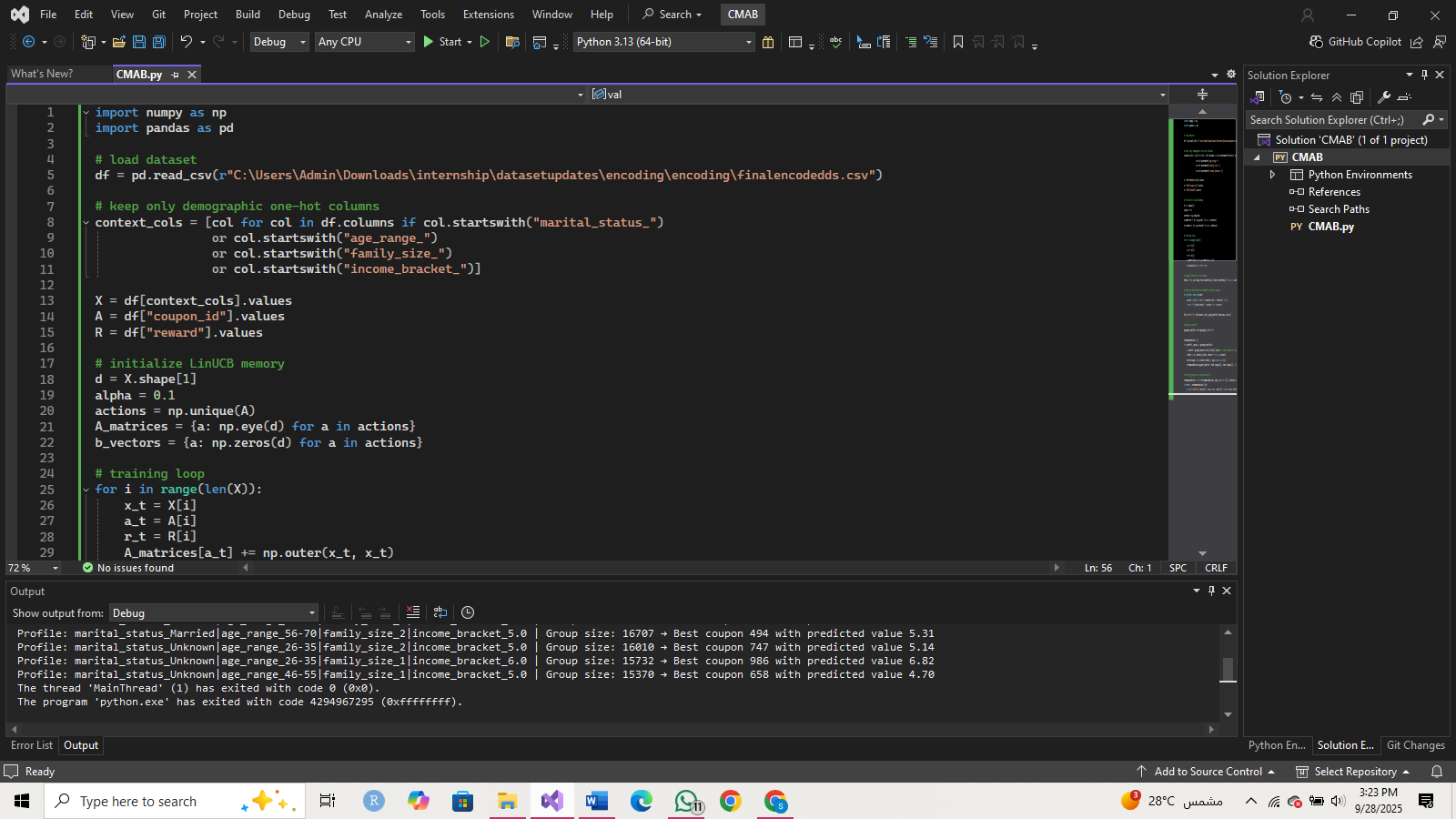




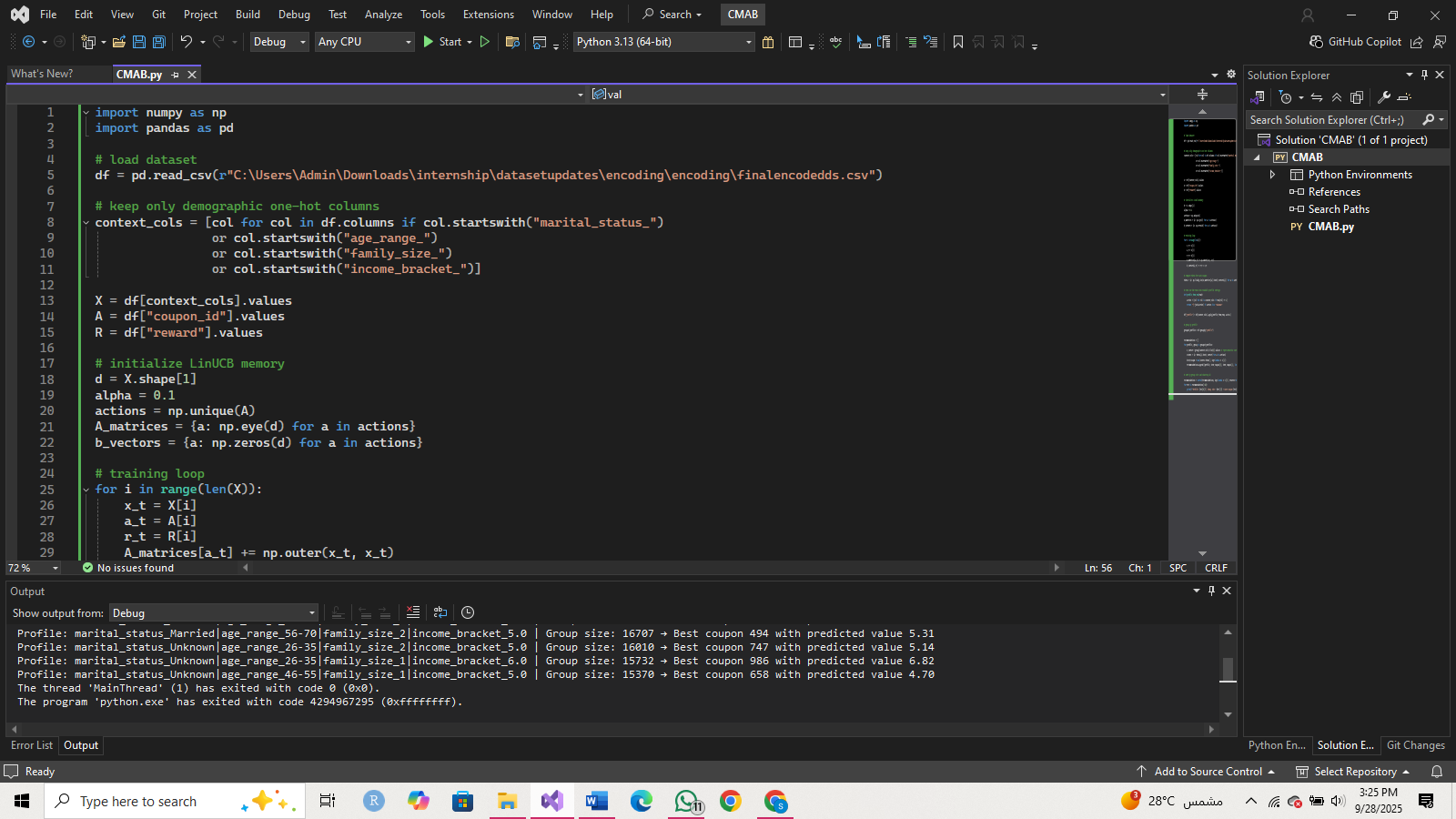
we treated each coupon as an independent action and estimated its expected reward. The model identified several coupons (like 663, 331, 11) with the highest Q-values (1.93), meaning these coupons are most effective on average across the population. However, since many coupons had similar estimated rewards, this suggests that personalization (via Contextual Bandits) is necessary to better differentiate coupon performance across customer groups.

CMAB:

**Loading dataset and select context features**



**Defining** context (X), actions (A = coupons), and rewards (R = spending level).



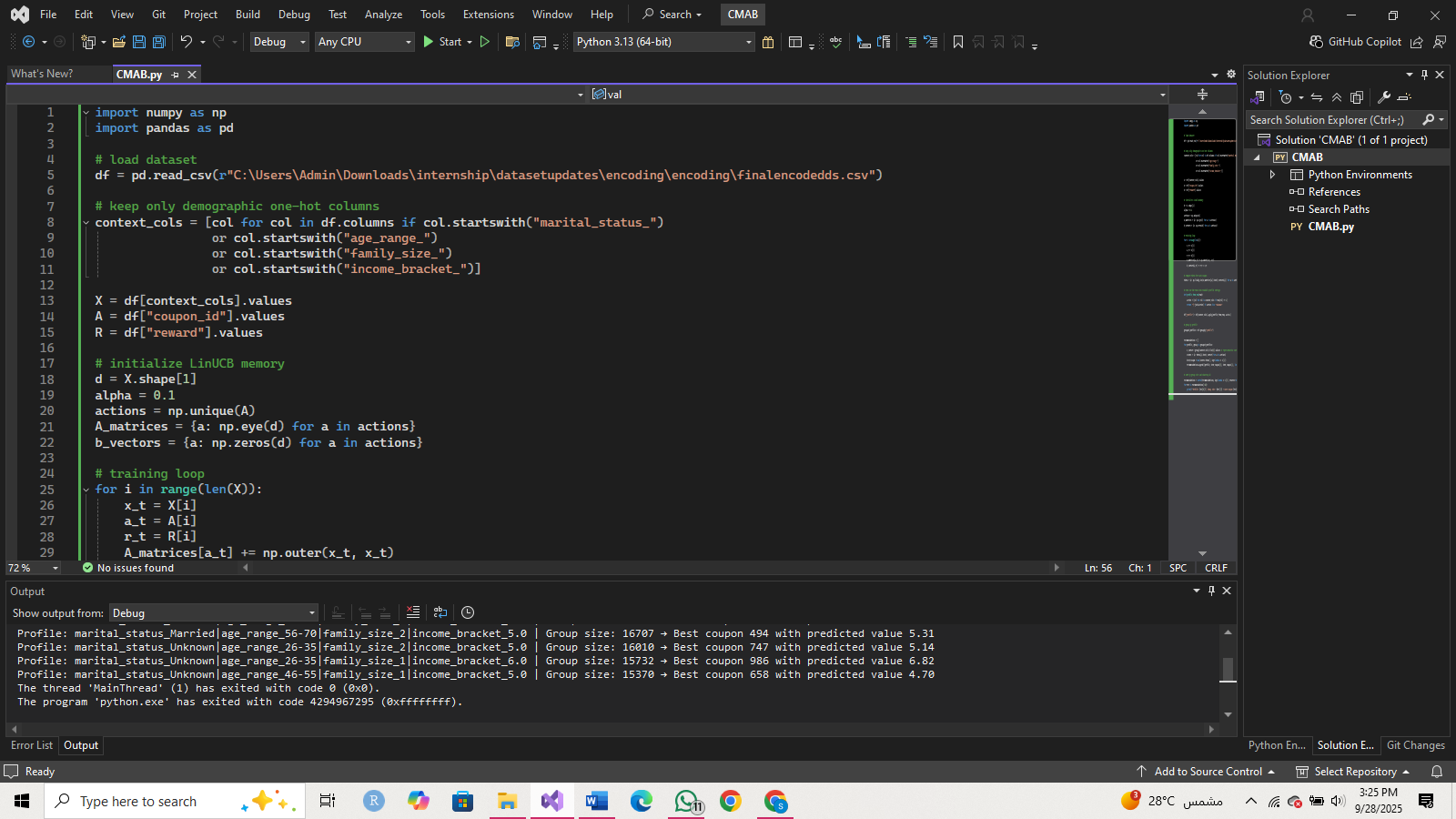
**Initializing LinUCB memory:**

We create

an A matrix that stores the history of features

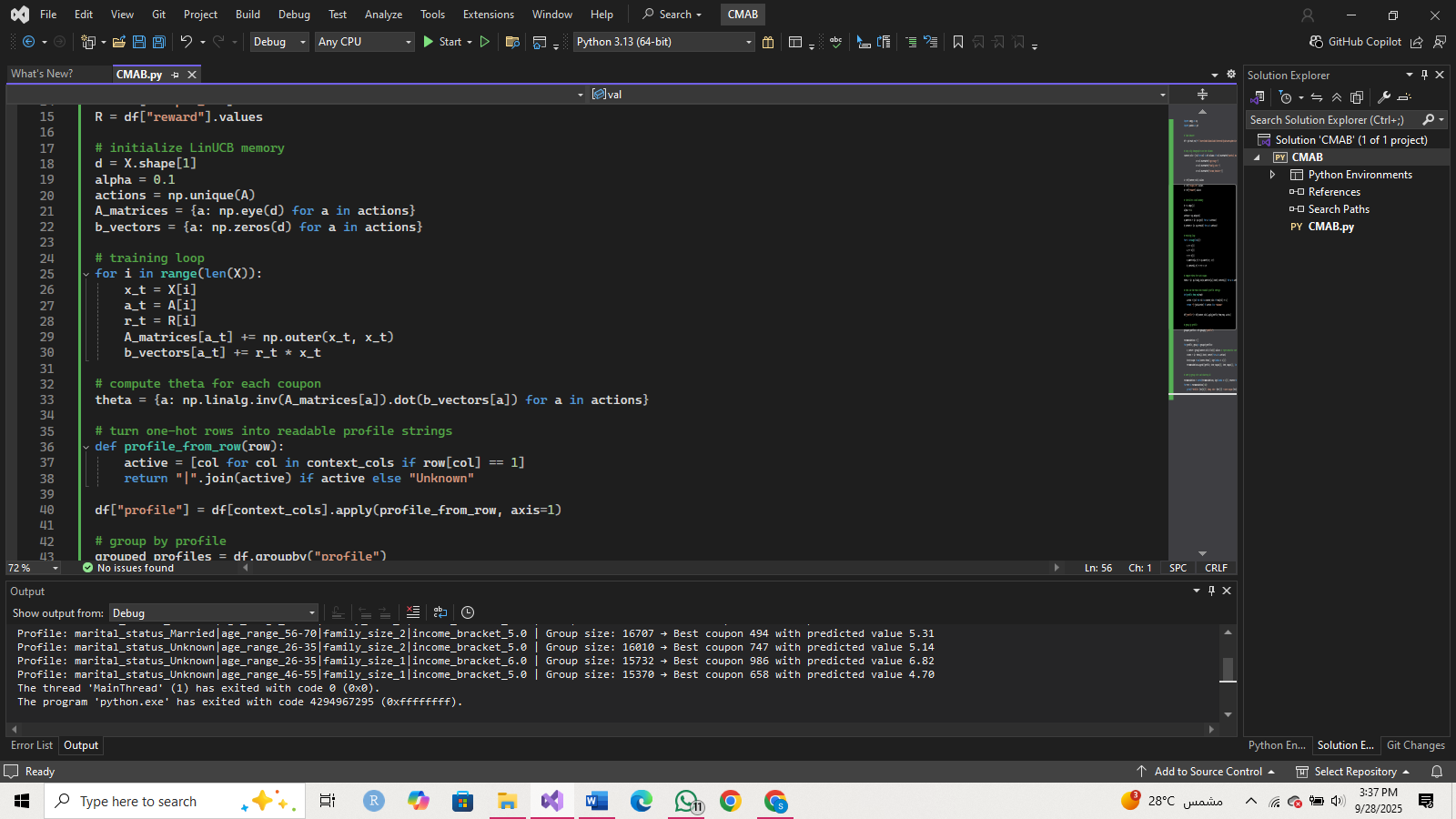
a b vector that stores the history of rewards.

for each coupon.



**Training loop**

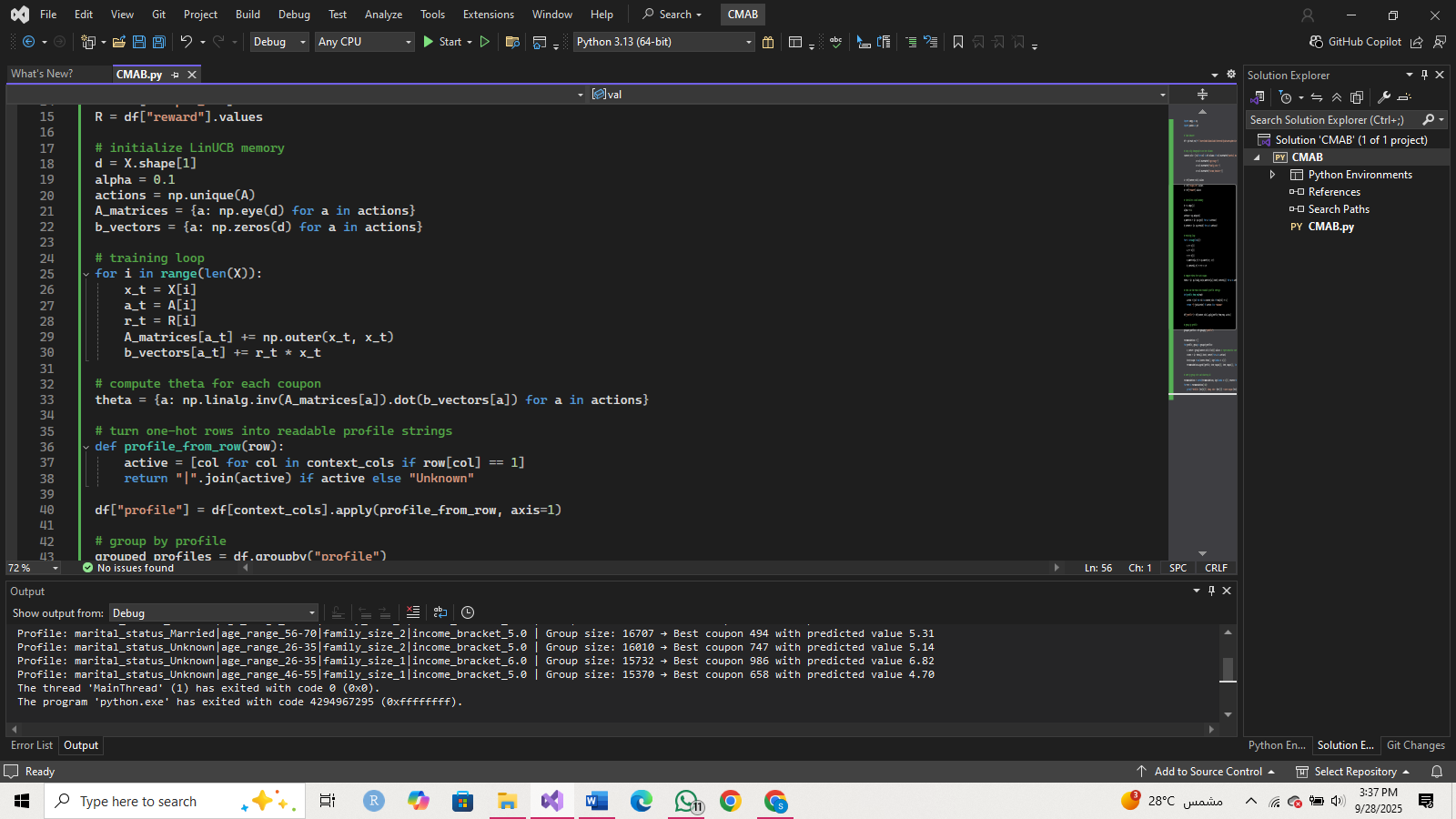
We loop over all rows (customer–coupon interactions) and update the memory for the coupon that was shown.



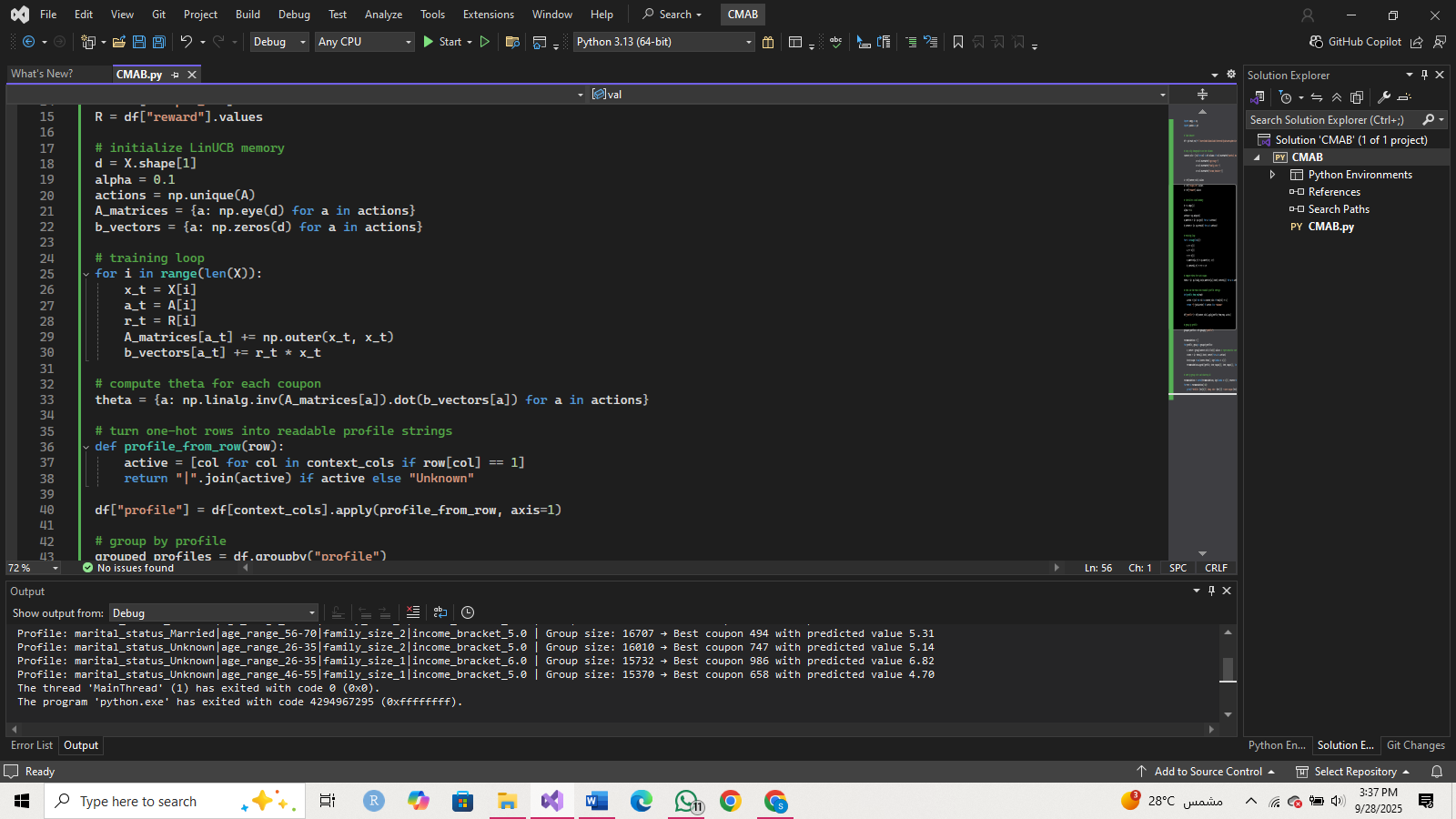
This makes coupons learn from every interaction: features get added into A, and reward contributions go into b.

**Compute θ (coupon weights)**

Once training is done, we calculate θ for each coupon. θ tells us how each feature contributes to the reward for that coupon.



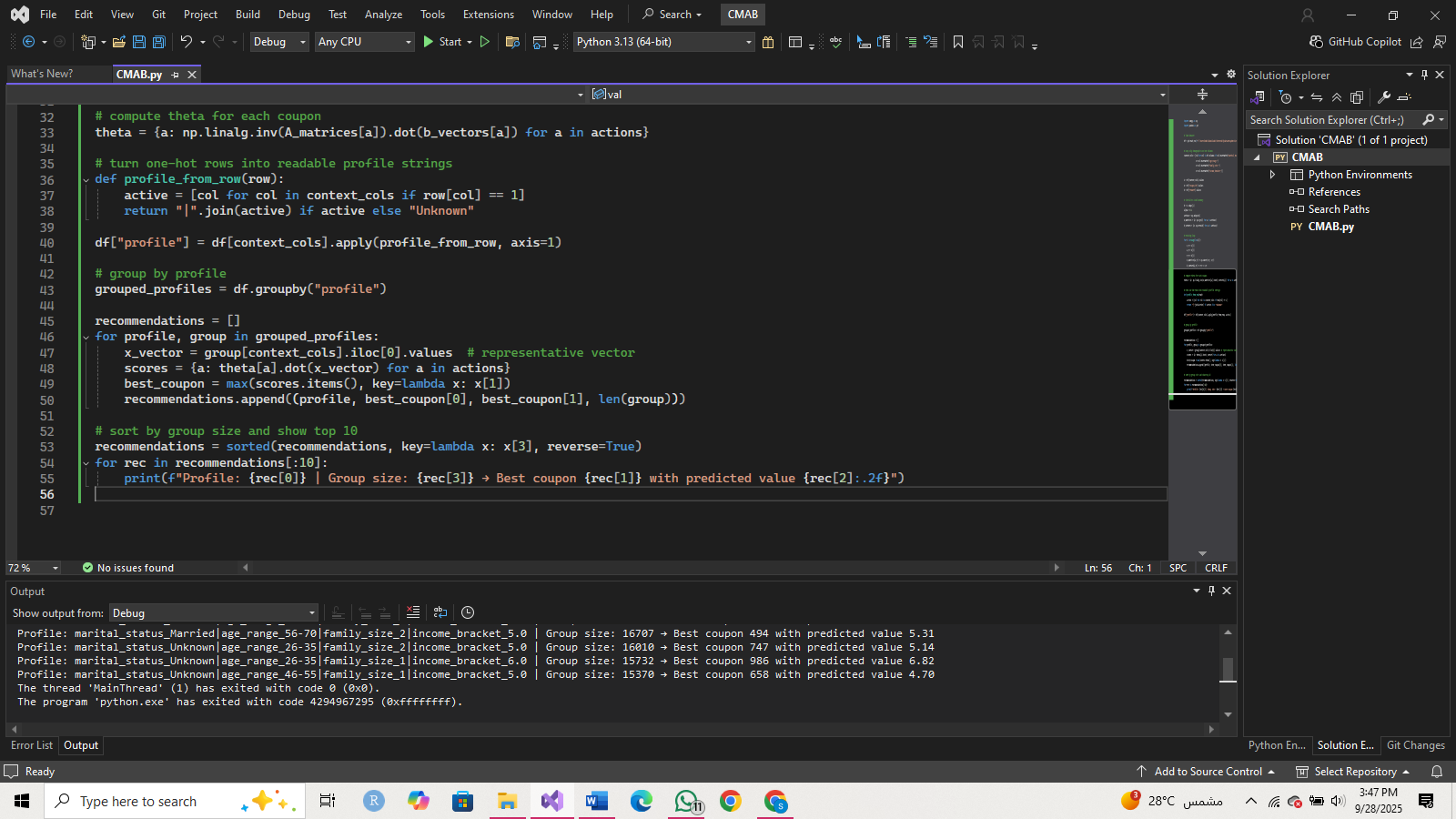
**Build human-readable profiles**  
Instead of keeping 0/1 feature rows, we convert them into strings listing only the active categories.



For ex marital\_status\_Married|age\_range\_26\_35|income\_bracket\_High|family\_size\_3

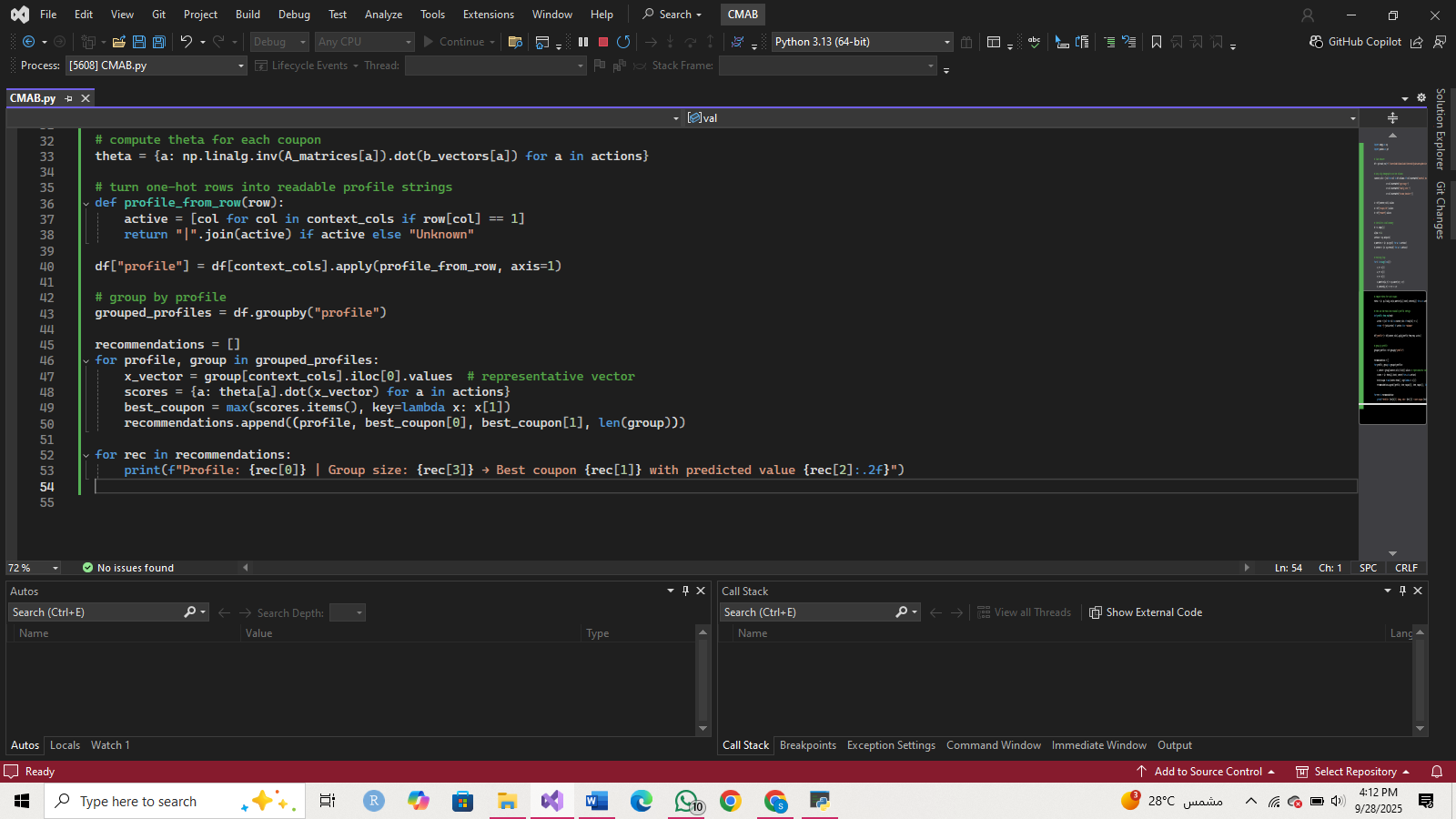
**Group by profile**

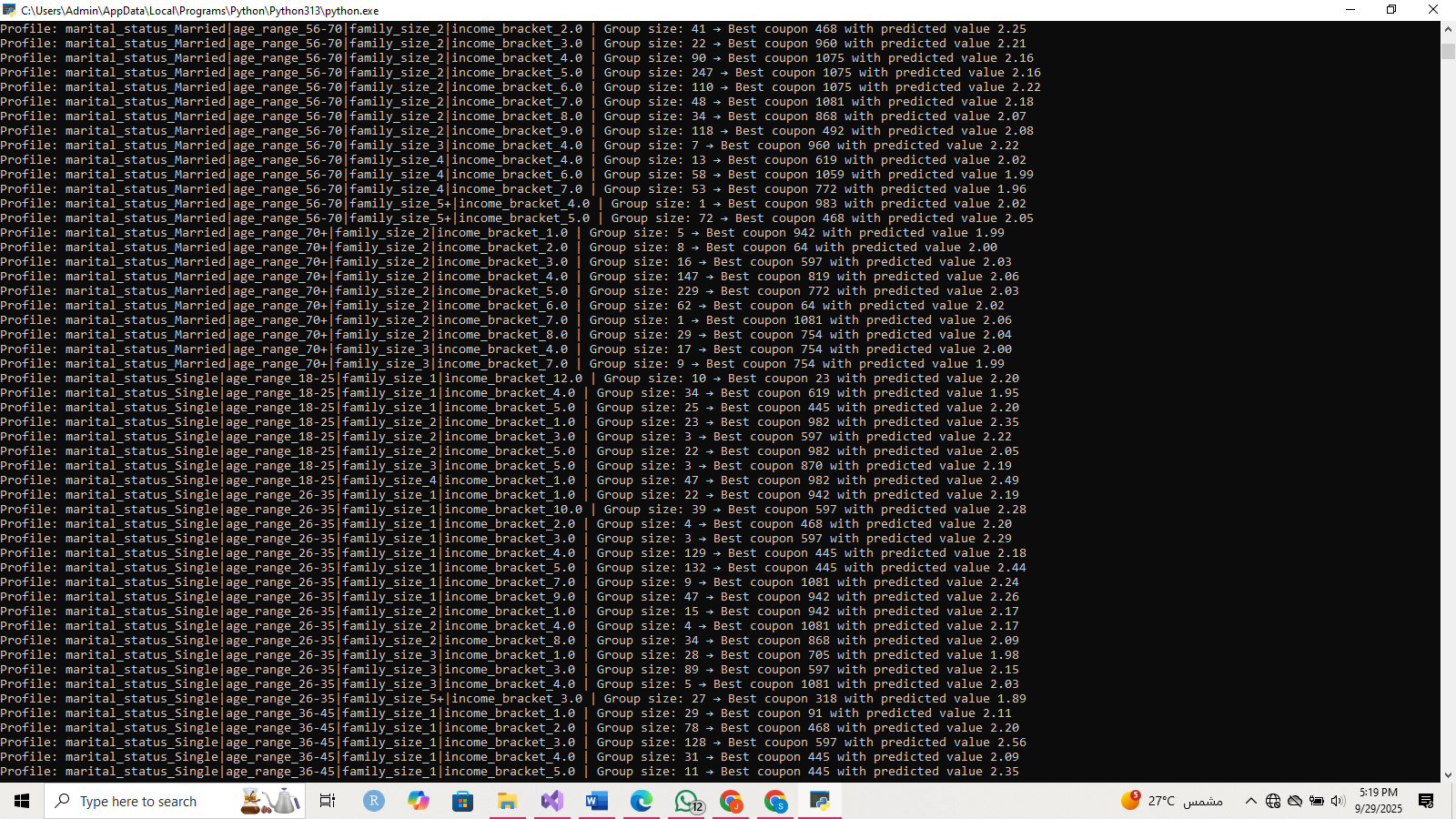
We put together all customers with the same profile and calculate the best coupon for each group.



Each group gets a dictionary of predicted scores for all coupons, and we keep the coupon with the highest value.

**Display results**





MDP:

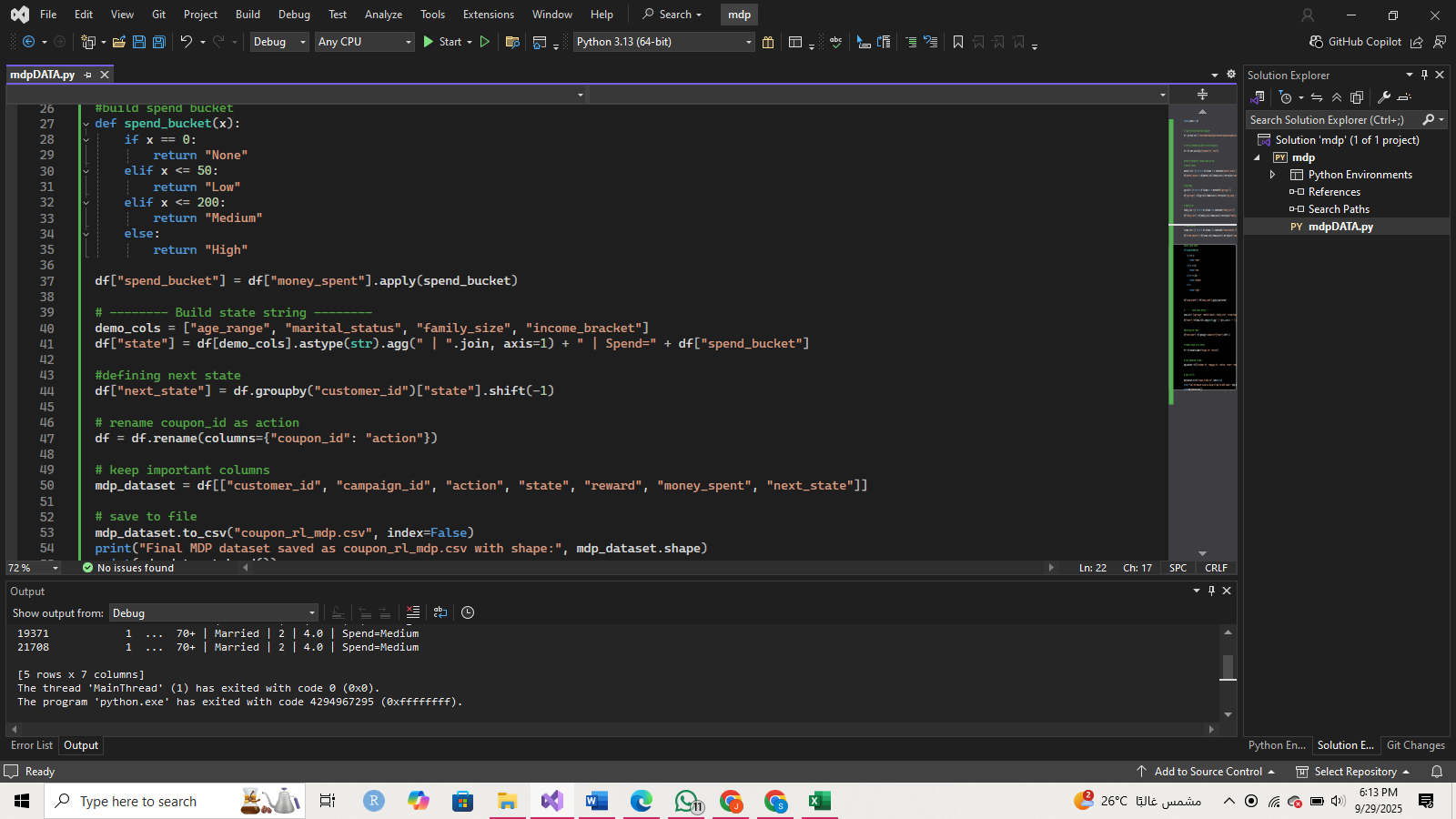
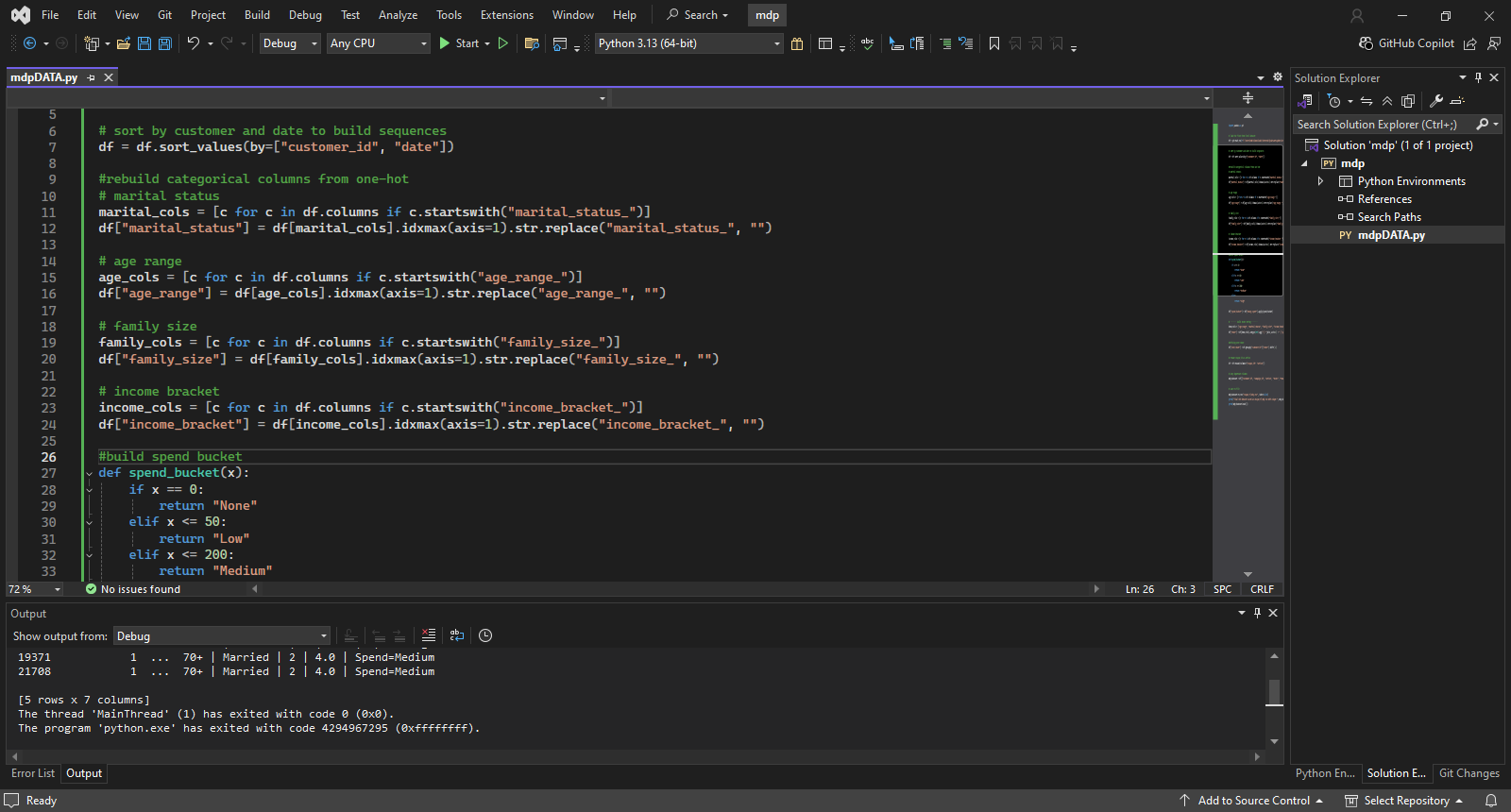
**State** = customer demographics + current spend level

**Action** = coupon shown (coupon\_id)

**Reward** = depends on spend (row-level, already assigned)

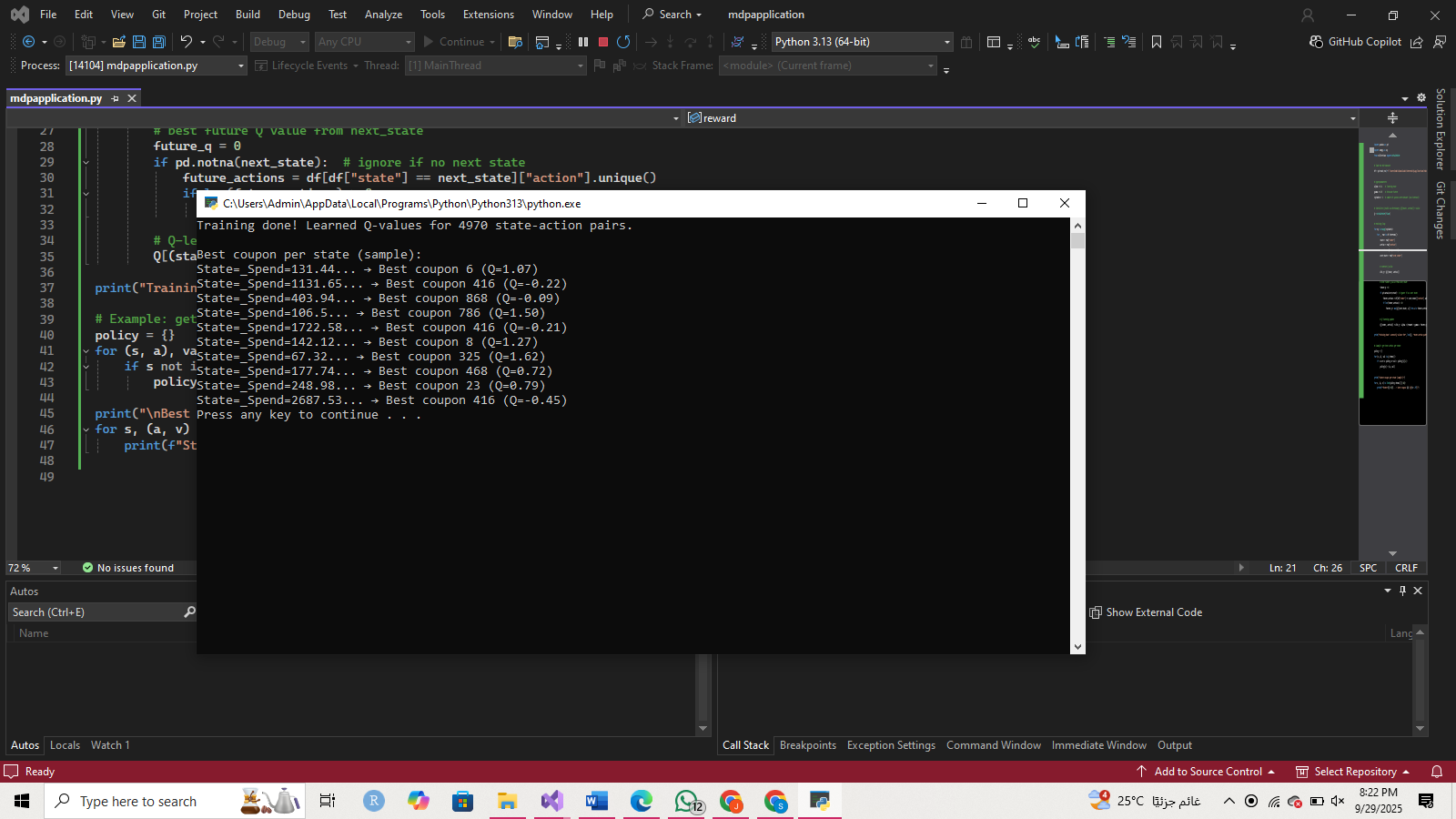
**Next state** = the following row for the same customer in chronological order

Built the data set:



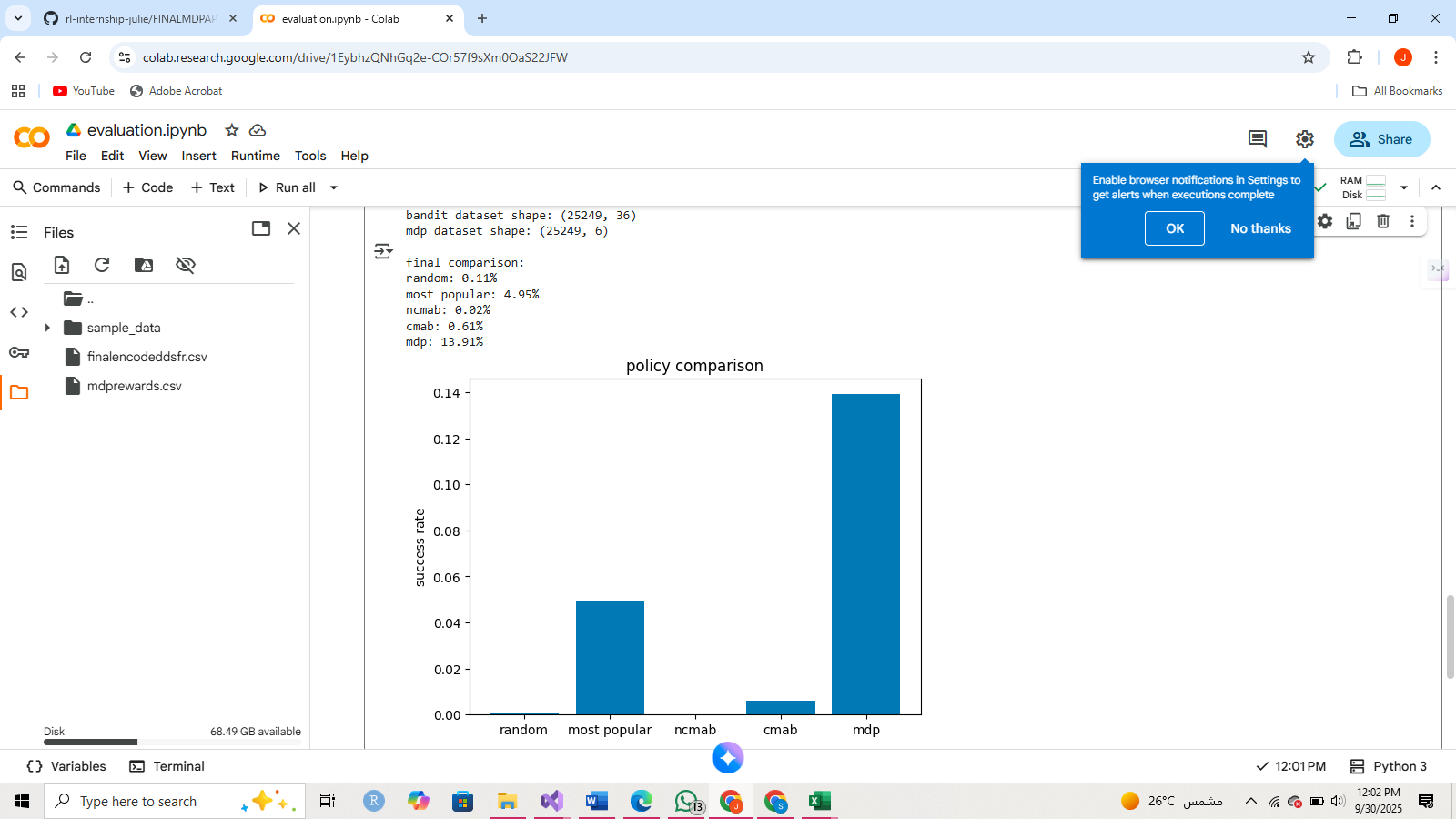
Applied q-learning where we generate a table where for each state we have the avg reward given by each action.

Because of that we can clearly define the best action for each state that also cares about the action after it.



**Evaluation**

I built a simple evaluation: if the action the algorithm says is best matches the actual action in the dataset and that action gave the max reward, I count it as (reward = 1). Then I measured average reward per approach. I also threw in baselines like “random coupon” and “most popular coupon.” That gave me something to compare against.



From this we can observe that:

* Random guessing is barely above 0 which is normal because it’s guessing different random actions each time.
* Most popular did a little better because one coupon dominates the dataset. But it’s still a dumb baseline.
* NCMAB came out really low and that’s because it doesn’t use context and just bets on average rewards across all customers.
* CMAB did a bit better at least it uses customer profiles, so it’s smarter than NCMAB.
* MDP scored the highest because it actually looks at transitions in spending and models customer behavior over time instead of just one-shot guessing.

**Conclusion**

* NCMAB found top coupons but wasn’t personalized.
* CMAB adapted to different customer types, so it felt more realistic.
* MDP gave the most flexible setup, but needed careful state design .
* Preprocessing and reward definition turned out to be the most important part. If those weren’t right, no algorithm made sense.

The project was basically about taking raw marketing data, shaping it into something that RL could understand, and then trying out different algorithms to see how they’d decide on offers.

The key lesson: most of the work is in cleaning and defining rewards. Once that’s done, the RL part is actually straightforward.