In the real world, before I tried to solve this problem, I would want to clarify a few points.

1. I would want to know whether they want only 2 sets of prices, one for users who aren't likely to spend money and one for those that are, or whether they want more sets of prices. In that latter case, a binary classification might not be the best approach, but I have assumed that they want to apply only 2 sets of prices.
2. I would also ask if all this data is appropriate, particularly the actions, or are the prices intended for new users who haven't yet done any actions. For this, I assumed all the data is appropriate to use, so the prices would be for things later in the game.
3. I would want to understand the relative downsides of misclassifying different types of users. Possibly it could be much worse to predict that a high spender is not going to spend (then you might lose a lot of revenue by giving them low prices) than predicting that someone that isn't going to spend is (then you may have minimal reduced revenue even with higher prices). Similarly, the loss may depend on the amount of money that a user would spend, so it would be worse to predict that a high spender is not going to spend than it is to predict that a low spender is not going to spend. Since I don't know how the price scheme would work, I just made model decisions to improve the F1 score.

Based on those assumptions I created a model that predicts whether a user will spend in the game (a binary yes/no value). So effectively, the general approach is:

(user + device + actions information) → model → (user will spend or not)

From the problem statement, it is clear that this model is for use on future users (it's not an analysis on past user behavior). However, since how this model would be used is not given, I created only a simple interface with minimal testing or error handling.

If this were a real problem I would want to better understand the non-accuracy constraints, like:

- Maximum allowable training time
- Maximum allowable serving/prediction time
- Should this fit with an existing platform/code
- Maximum allowable cost or what magnitude of cost is reasonable to run this (for example, if it is going to earn an extra $100, you don't want it to cost $1000)
- Are there support/maintenance considerations (like if no one else in the company is experienced with Spark, then Spark probably isn't a good choice)
- etc.

Then the data itself presents a few issues:

1. Data usefulness & cleanliness - I would ideally get clarification about the data itself or test it, but in place of that I made a few assumptions about the data:

- device_mem_i does not provide more information than device_mem_grouping_i
- The same for device_gmem_i and device_gmem_grouping_i
- The same for device_model_s and device_mapped_s

- device_os_s is less effective than just the base OS, so for example the entry "android os 9 / api-28 (ww_phone-201905061022/1.." will be converted to “android os 9 "
- The base OS is always the value before "/" in device_os_s
- Whenever game_stats_tutorial_complete is missing, the user did not complete the tutorial
- Whenever game_stats_xp[...] is missing, the user achieved 0 xp[...]
- A missing value for every other column will be replaced with the most common value of that column. I'm not using the average or median for numeric fields simply for simplicity
- The language itself doesn't provide information, just whether it is the most common language for that country

Based on those assumptions, I performed some data transformations, but otherwise the data that is input to the model is the full set provided.

2. Many of the data fields are categorical but have a high cardinality. There are several methods to try to handle them well, like binning values or taking the top K values and then one-hot-encoding or embedding the fields, but besides effectively binning one column I did not do that, since then you would want to test different bin sizes or Ks. Instead, I just used CatBoost to handle categorical fields.
3. Similarly, the provided data is highly imbalanced, with much more non-spenders than spenders. You could test out more advanced approaches to handle this imbalance, like SMOTE sampling, possibly a one-class classifier, or maybe a neural network that learns a distance function and uses the nearest neighbor. But mostly to save time, I just used random sampling in order to make the two classes have about the same number of instances.