Item Pricing Recommendation

1. Domain Background

In e-commerce, one of the main features wanted is to be able to recommend sellers a price for the items they intend to sell. The main reason behind it is that the customers are not always aware what would be a reasonable price (neither too high, nor too low, thus increasing the probability of selling their items). Besides the suggestion, the feature may be thought of as an automatic price option. Also it may be a first step against unrealistic intentionally prices (a common problem in e-commerce apps). Last but not least, it can be a traction factor for users (new or already present).

There are specialised companies that offer this feature for well defined product niches, but it is always an advantage when capable of creating oneself a customised pricing recommender. It has to be pointed out that generally the recommendation is subject of a specific category for the different possible price ranges, and may take into account the exclusion of existing outliers.

Personally I find the challenge interesting and useful also as a capstone application of the present ML Engineer nanodegree program.

2. Problem Statement

The problem is a regression problem (predict the price of an item). The models will take as input the following features: item name, item description, item brand, item category, item condition and shipping mode, and will produce as output the predicted price.

3. Datasets and Inputs

The dataset is provided from the Kaggle competition link
(https://www.kaggle.com/c/mercari-price-suggestion-challenge/data)
and consists essentially of:

- a *training* dataset:

train_id	item id
name	item title
item_condition_id	item condition
category_name	item category (enumeration of a hierarchy of categories separated by slashes)
brand_name	item brand name
price	the price the item was sold for [target variable]
shipping	1 or 0 (1 shipping fee paid by seller, 0 by buyer)
item_description	item full description

- a *test* dataset: same fields save for the price that is the target variable to be predicted for the items in this dataset.

Below there is a sample of the training data:

```
df.head(5)
```

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	1	No description yet
1	1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	0	This keyboard is in great condition and works ...
2	2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	1	Adorable top with a hint of lace and a key hol...
3	3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...
4	4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44.0	0	Complete with certificate of authenticity

All of the 1482535 items are different.

```
df.shape
```

```
(1482535, 8)
```

```
df['train_id'].nunique()
```

```
1482535
```

There are four features with null values (the three below plus the equivalent "No description yet" in *item_description*). Only the 6237 items with no category may be removed.

```
df.columns[df.isnull().any()]
```

```
Index(['category_name', 'brand_name', 'item_description'], dtype='object')
```

```
df[df['category_name'].isnull()].count()['train_id']
```

```
6327
```

It is interesting to analyse the hierarchy structure of the categories, we get there are at most 5 levels

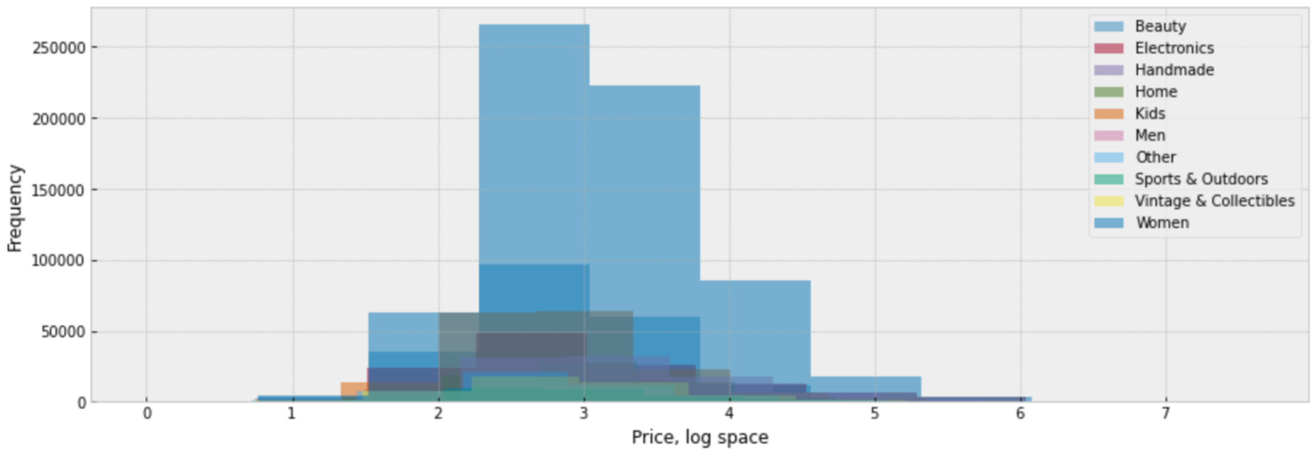
```
ddf = df.copy()
ddf = ddf.join(ddf['category_name'].str.split('/', expand=True).add_prefix('cat_').fillna(np.nan
ddf.head()
```

dition_id	category_name	brand_name	price	shipping	item_description	cat_0	cat_1	cat_2	cat_3	cat_4
3	Men/Tops/T-shirts	NaN	10.00000	1	No description yet	Men	Tops	T-shirts	NaN	NaN
3	Electronics/Computers & Tablets/Components & P...	Razer	52.00000	0	This keyboard is in great condition and works ...	Electronics	Computers & Tablets	Components & Parts	NaN	NaN
1	Women/Tops & Blouses/Blouse	Target	10.00000	1	Adorable top with a hint of lace and a key hol...	Women	Tops & Blouses	Blouse	NaN	NaN
1	Home/Home Décor/Home Décor Accents	NaN	35.00000	1	New with tags. Leather horses. Retail for [rm]...	Home	Home Décor	Home Décor Accents	NaN	NaN
1	Women/Jewelry/Necklaces	NaN	44.00000	0	Complete with certificate of authenticity	Women	Jewelry	Necklaces	NaN	NaN

with the following number of elements:

```
cat_0                        10
cat_1                       113
cat_2                       870
cat_3                         6
cat_4                         2
```

The first level category (cat_0) is too coarse regarding the price distribution to be used alone, as the below histogram shows:



The idea is to go deeper to the 2nd or 3rd level.
Regarding the categorical features *name* and *description*, it has to be seen their correlation degree.

4. Solution Statement

To build the pricing recommender I will use two different implementations: an extreme gradient boosting (**XGBoost**) model (the implementation provided by AWS) that will represent the comparison basis for the second implementation based on an **RNN** built with **PyTorch**. The deployment and testing will be performed in both cases using SageMaker.

5. Benchmark Model

The **XGBoost** model mentioned will be used when comparing the results with the **RNN** solution.

6. Evaluation Metrics

I am thinking of the usual evaluation metrics, beginning with the *Root Mean Squared Logarithmic Error* (RMSLE) that will allow the comparison of my results with those of the Kaggle competition, too.

$$\sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(log(\widehat{p}_i+1)\ -\ log(p_i+1))^2}$$

Here and below, *n* = number of observations in the dataset, *p-hat* = price prediction, and *p* - actual price.

I will also include the *Mean Absolute Error* (MAE) and the *Mean Absolute Percentage Error* (MAPE).

MAE $$\frac{1}{n}\sum_{i=1}^{n}|\widehat{p}_i-p_i|$$

MAPE $$\frac{1}{n}\sum_{i=1}^{n}\frac{|\widehat{p}_i-p_i|}{p_i}$$

7. Project Design

The first step implies the *data preparation* and *preprocessing* (sketched at point 3).

Further the data will be split into *training* and *test*.

In the next step, the *train method* will be defined. And for the RNN with PyTorch the *model* built, too.

Next, the *estimator* will be defined and fitted, varying the *hyperparameters*.

Afterwards, the model will be *deployed* for testing. For this, the predictor has to be prior defined.

The *test dataset* (see point 3) can now be used and the *metrics* (point 6) evaluated.