Project Evaluation 2 Report - SnapIt

Rebooters:

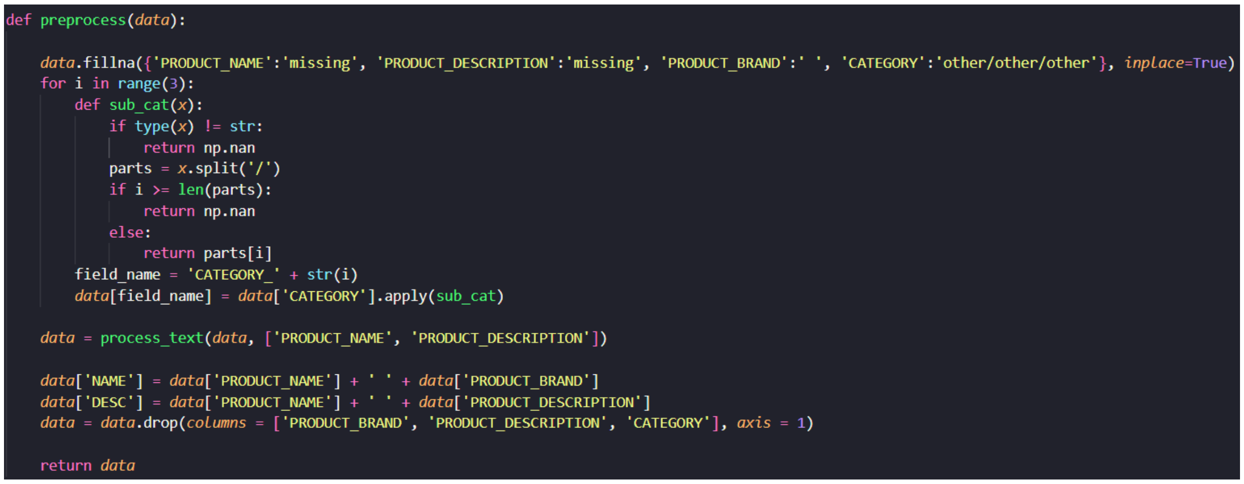
Mansih Reddy .K	IMT2020019
Prudhvi Nath Reddy .S	IMT2020082

Where we left of:

We had done some preprocessing and data analysis by the first evaluation but that was only preliminary. After the first evaluation we again started from scratch using the old notebook as a guide. We first resolved the text discrepancies in ‘PRODUCT_NAME’ and ‘PRODCUT_DESCRIPTION’, be it grammatical syntaxes or inconsequential words in the data. This was done using the libraries ‘Stopwords’ and ‘String.punctuations’. Contractions like “ I’m “ or “ can’t ” were decontracted in a grammatically sound fashion.

We had previously been hasty in our removal of the attribute ‘PRODUCT_BRAND’, as we had later realized these values could have a significant sway on the price as they do in real life. But we needed a way to remove any possible bias which could occur due to the fact that almost half the items don’t have a brand. Hence we decided to combine the attributes ‘PRODUCT_BRAND’ and ‘PRODUCT_NAME’ in a singular attribute ‘NAME’. Similarly ‘PRODUCT_NAME’ and ‘PRODUCT_DESCRIPTION’ were also combined into ‘DESC’ as it was observed that a lot of the item names had descriptive features in them (size, color, quantity, etc.). The missing values are also handled by filling them with ‘missing’ for ‘PRODUCT_NAME’ and ‘PRODUCT_DESCRIPTION’ , ‘ ‘ for ‘PRODUCT_BRAND’ and ‘other/other/other’ in case of ‘CATEGORY’.

Last time we were still on the edge regarding whether to split the attribute ‘CATEGORY’ into 3 and then use TF-IDF or one hot encode it first. But we realized working with more than unique values that are one hot encoded was too tedious and time consuming for the compiler. Hence we split it into ‘CATEGORY_0’,‘_1’ ,‘_2’ split by ‘/’



```
def preprocess(data):

    data.fillna({'PRODUCT_NAME':'missing', 'PRODUCT_DESCRIPTION':'missing', 'PRODUCT_BRAND':' ', 'CATEGORY':'other/other/other'}, inplace=True)
    for i in range(3):
        def sub_cat(x):
            if type(x) != str:
                return np.nan
            parts = x.split('/')
            if i >= len(parts):
                return np.nan
            else:
                return parts[i]
        field_name = 'CATEGORY_' + str(i)
        data[field_name] = data['CATEGORY'].apply(sub_cat)

    data = process_text(data, ['PRODUCT_NAME', 'PRODUCT_DESCRIPTION'])

    data['NAME'] = data['PRODUCT_NAME'] + ' ' + data['PRODUCT_BRAND']
    data['DESC'] = data['PRODUCT_NAME'] + ' ' + data['PRODUCT_DESCRIPTION']
    data = data.drop(columns = ['PRODUCT_BRAND', 'PRODUCT_DESCRIPTION', 'CATEGORY'], axis = 1)

    return data
```

The above is how the preprocessing was done.

Splitting the Data:

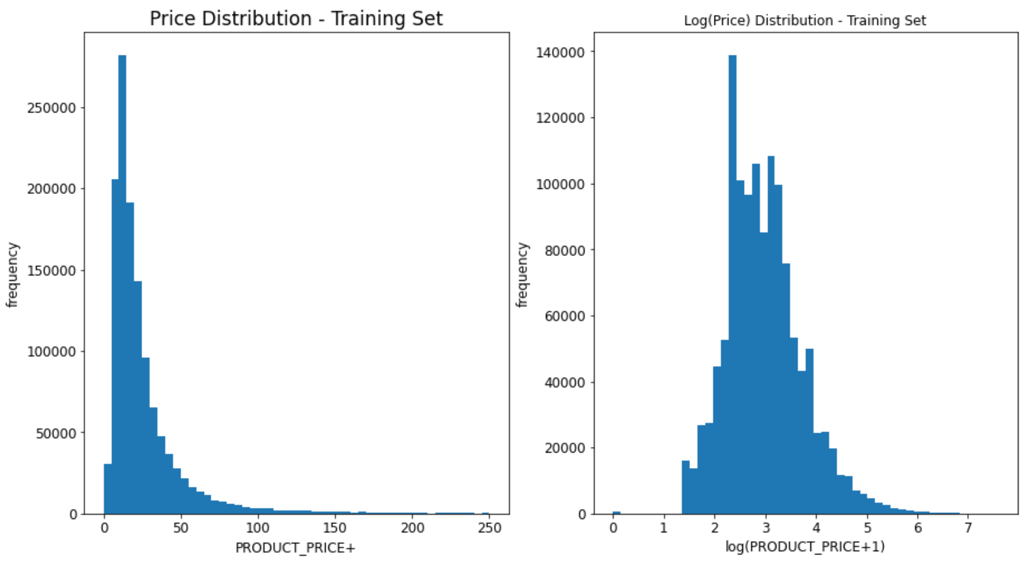
We also split the train data into X_train and X_valid with the attribute ‘PRODUCT_PRICE’ as the y_train and y_valid using train_test_split. This was done so as to have a dataset to find the accuracy of the model using cross validation between the predicted price and the price given. This is done as the test data does not have the attribute ‘PRODUCT_PRICE’ in it. We predict the prices for the test but cannot check how accurate those predictions are, hence we look at the given values for validation of the model.

Encoding:

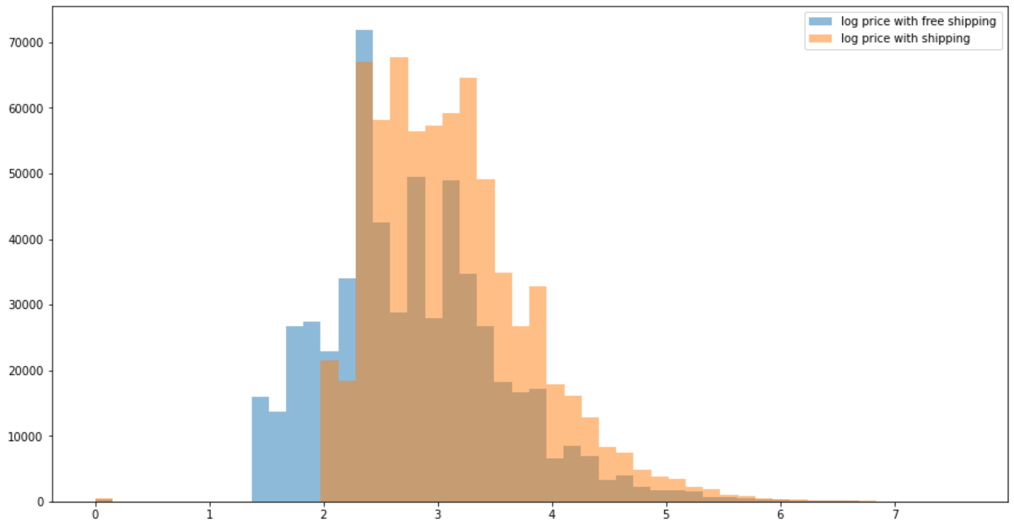
Next comes the encoding part, we used Countvectorizer to encode the three category attributes for the train, test and valid datasets. Then used Tfidfvectorizer to encode ‘NAME’ and ‘DESC’ for all three of them. The attributes ‘SHIPPING_CONDITION’ and ‘PRODUCT_CONDITION’ are compressed into sparse matrices using csr_martix. Then all these features are stacked into one array using hstack. We get 3 final outputs namely, train_data, test_data and valid_data. This is the crucial data needed to train the model.

Exploratory Data Analysis:

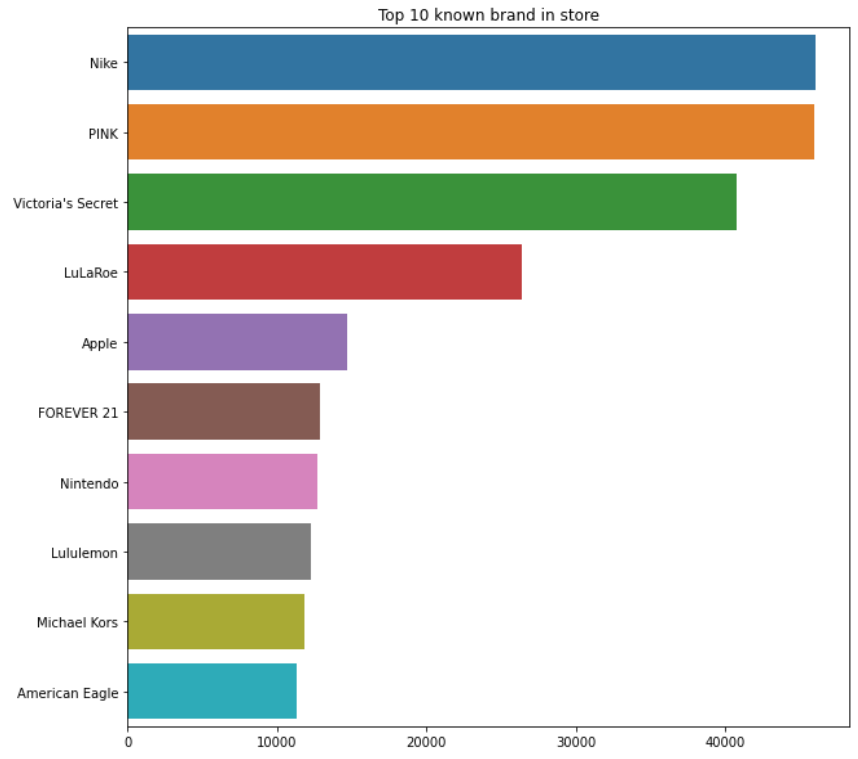
One of the first things analyzed was the price distribution in the dataset.



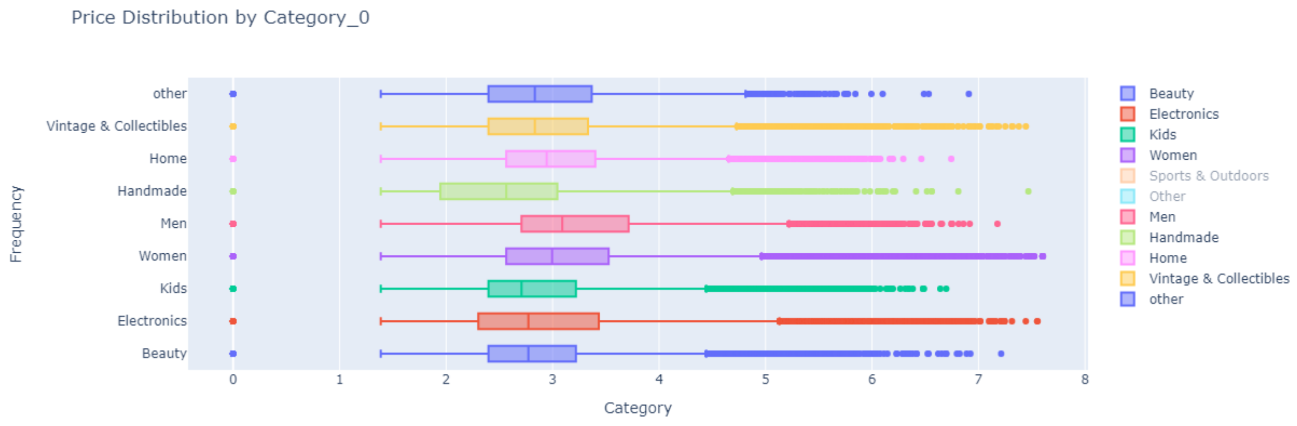
Then the price distribution with respect to the ‘SHIPPING_CONDITION’ was taken. While we assumed the price for items with free shipping to be high, that was proven to be false.



The top 10 brands in the items by popularity are the following.

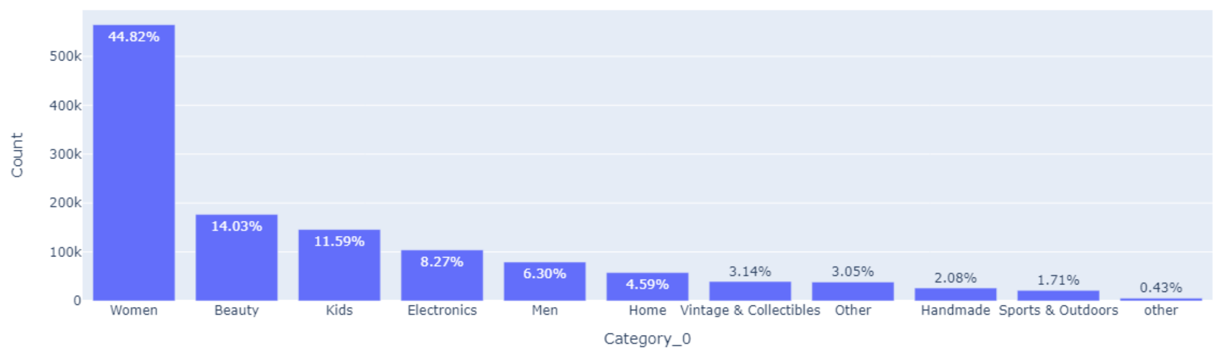


The following is the plot for price distribution based on 'CATEGORY_0'.

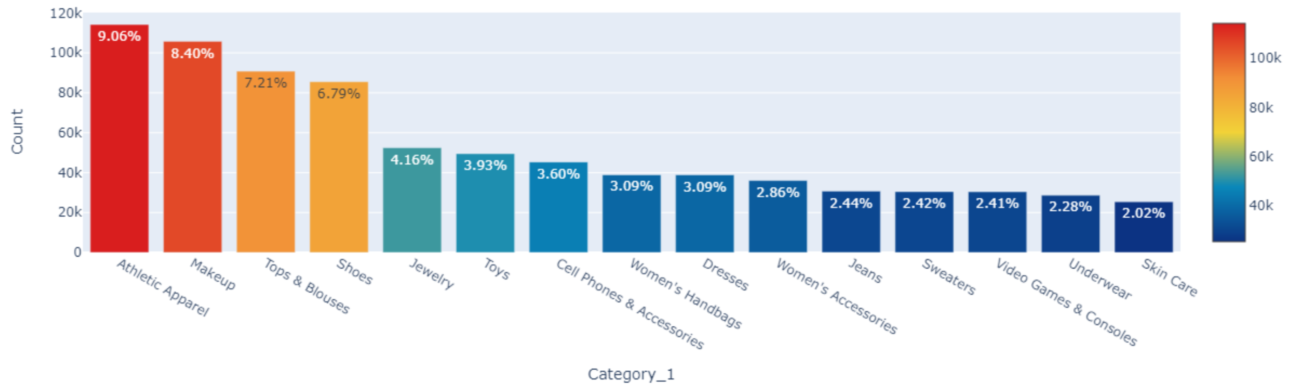


The item distribution based on 3 CATEGORY attributes are as follows:

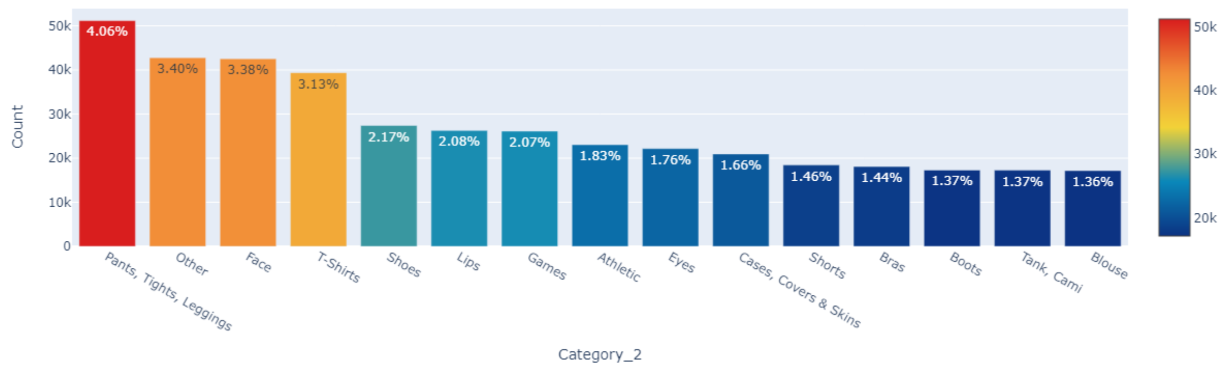
Item Distribution in Category_0



Item Distribution in Category 1 (Top 15)



Item Distribution in Category 2 (Top 15)



The Model:

The model we made is a Ridge regression model. Our previous attempt had a linear regression model too using gradient descent, but the RMSLE was much higher so it was abandoned. We import the Ridge functionality from sklearn library. We have the data for training, testing and for final predictions. We only need to find an optimal param alpha for acquiring as low of an error(rmsle) as possible. We used the solvers “auto” and “sag” for testing. While ‘sag’ was always much faster, there was a considerable gap in the rmsle when using ‘auto’ (lower rmsle). But again for optimizing alpha using ‘auto’ the time taken is also much higher.

What Next?

We are mostly done with the project, the only thing left to do is trying to decrease the rmsle for cross validation. We are looking into how to make that happen even now. We could decrease the train_test_split ratio, or give more precise alpha possibilities to the model. We tried the second method by making a function that works a bit similar to gradient descent function we made in the previous assignment by giving increasing alpha values and only stopping when there is a very small amount of decrease or it increases, but unfortunately, the rmsle values for this data are not consistent enough to facilitate that.