Project Report: SNAPIT

Agastya Thoppur • IMT2020528
Anwit Damale • IMT2020532

Overview

Evaluations:

Evaluation 1

- Basic EDA
- Pre-Processing
- Applying Linear Regression

Evaluation 2

- Text to Numeric value conversion using TFIDF
- Applying Ridge Regression
- Applying Lasso Model

Evaluation 3

- Using Label Binarizer
- Applying Neural Networks

Evaluation 1

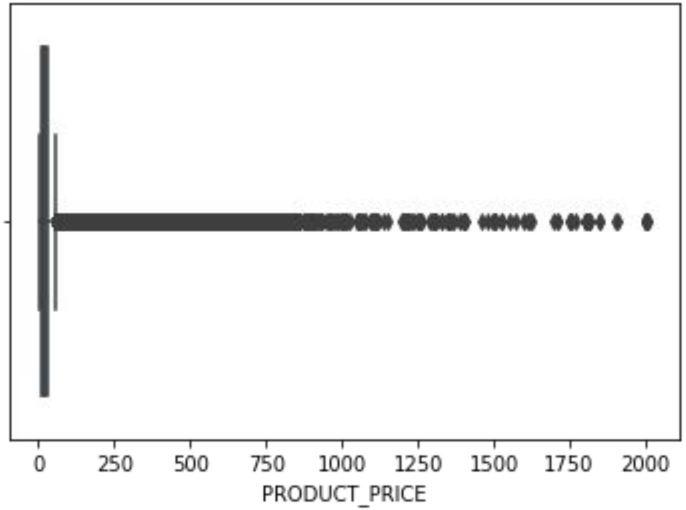
Pre-Processing Steps:-

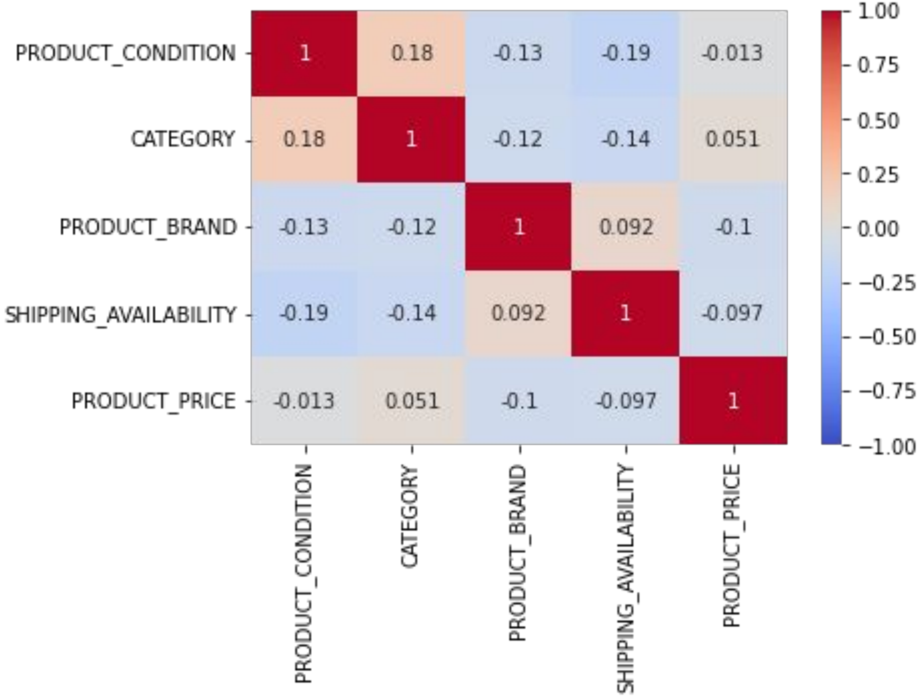
- Removal of Negative Product Prices.
- Assigning “Unbranded” and “Uncategorised” to the NULL values in the BRAND and CATEGORY columns.
- Using Label Encoding on the BRAND and CATEGORY columns.

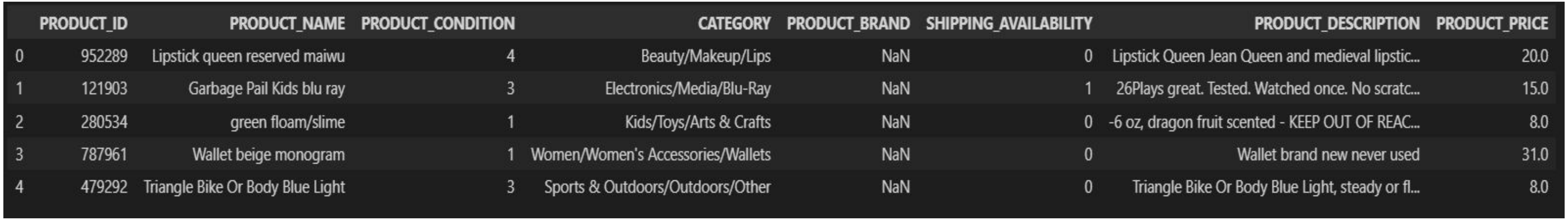
Evaluation 1

Pre-Processing Steps:-

- Eliminating the outliers based on product prices using min-max threshold values and box plots.
- Applying the Linear Regression Model.


0
250
500
750
1000
1250
1500
1750
2000
PRODUCT_PRICE


PRODUCT_CONDITION
CATEGORY
PRODUCT_BRAND
SHIPPING_AVAILABILITY
PRODUCT_PRICE
1
0.18
-0.13
-0.19
-0.013
0.18
1
-0.12
-0.14
0.051
-0.13
-0.12
1
0.092
-0.1
-0.19
-0.14
0.092
1
-0.097
-0.013
0.051
-0.1
-0.097
1
PRODUCT_CONDITION
CATEGORY
PRODUCT_BRAND
SHIPPING_AVAILABILITY
PRODUCT_PRICE
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



	PRODUCT_ID	PRODUCT_NAME	PRODUCT_CONDITION	CATEGORY	PRODUCT_BRAND	SHIPPING_AVAILABILITY	PRODUCT_DESCRIPTION	PRODUCT_PRICE
0	952289	Lipstick queen reserved maiwu	4	Beauty/Makeup/Lips	NaN	0	Lipstick Queen Jean Queen and medieval lipstic...	20.0
1	121903	Garbage Pail Kids blu ray	3	Electronics/Media/Blu-Ray	NaN	1	26Plays great. Tested. Watched once. No scratc...	15.0
2	280534	green floam/slime	1	Kids/Toys/Arts & Crafts	NaN	0	-6 oz, dragon fruit scented - KEEP OUT OF REAC...	8.0
3	787961	Wallet beige monogram	1	Women/Women's Accessories/Wallets	NaN	0	Wallet brand new never used	31.0
4	479292	Triangle Bike Or Body Blue Light	3	Sports & Outdoors/Outdoors/Other	NaN	0	Triangle Bike Or Body Blue Light, steady or fl...	8.0

```
 dataset = dataset[dataset['PRODUCT_PRICE'] > 0]
 dataset.describe()
```

```
dataset["PRODUCT_BRAND"].fillna("Unbranded", inplace=True)
dataset["CATEGORY"].fillna("Uncategorized", inplace=True)
test["PRODUCT_BRAND"].fillna("Unbranded", inplace=True)
test["CATEGORY"].fillna("Uncategorized", inplace=True)
```

```
label_encoder=LabelEncoder()
columns_to_label_encode=dataset["CATEGORY"]
label_encoded_columns=label_encoder.fit_transform(columns_to_label_encode)
dataset["CATEGORY"]=label_encoded_columns
columns_to_label_encode=dataset["PRODUCT_BRAND"]
label_encoded_columns=label_encoder.fit_transform(columns_to_label_encode)
dataset["PRODUCT_BRAND"]=label_encoded_columns
dataset.head()
```

```
y_train = dataset["PRODUCT_PRICE"].to_numpy().astype(np.float64)
x_train = dataset.drop(columns="PRODUCT_PRICE").to_numpy().astype(np.float64)


L=LinearRegression()
L.fit(x_train,y_train)
x_test = test.to_numpy().astype(np.float64)
new_y=L.predict(x_test)
```

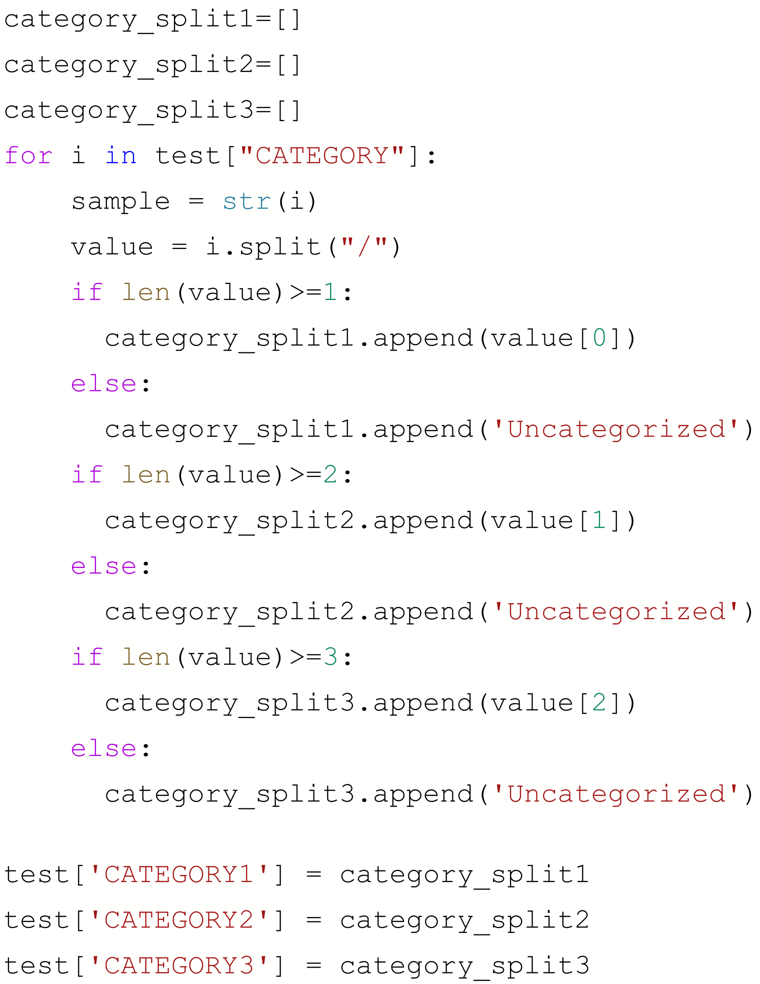
The value we got after this was around 0.77.

Goals for next Evaluation

1. See what we can do with the text columns like product description and product name.(TFIDF / Label Binarizer.)
2. Removal of Non-Important words from the Description column to improve accuracy.(Stopwords).
3. Applying better models.(Ridge Regression, Lasso Model)

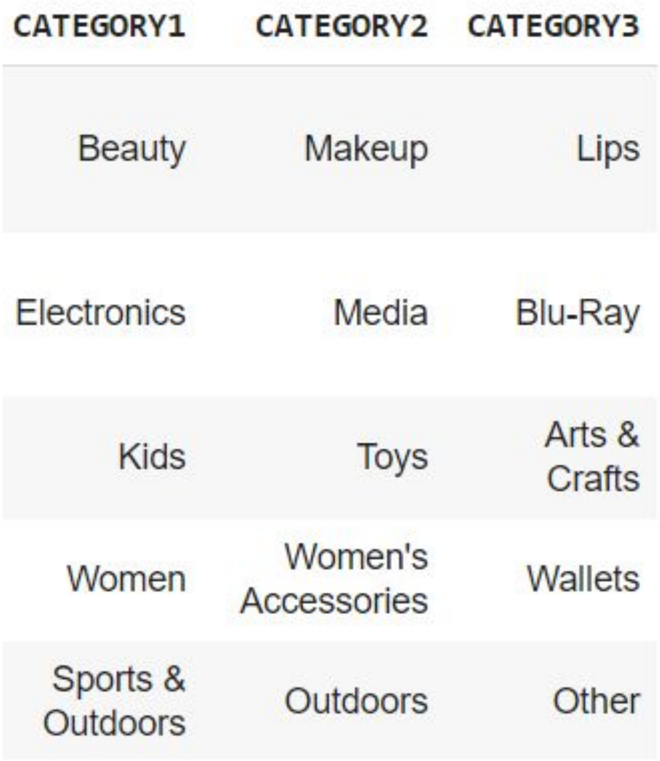
Evaluation 2

- Now, we split the Product Category column into 3 sub-category columns.
- Removing all the special characters and punctuations from the Description and Name columns.
- Now we remove the Stopwords from the Description Column.
- TFIDF Vectorization on the 3 sub-categories, Product Name, Brand Name and Description.



```
category_split1=[]
category_split2=[]
category_split3=[]
for i in test["CATEGORY"]:
    sample = str(i)
    value = i.split("/")
    if len(value)>=1:
      category_split1.append(value[0])
    else:
      category_split1.append('Uncategorized')
    if len(value)>=2:
      category_split2.append(value[1])
    else:
      category_split2.append('Uncategorized')
    if len(value)>=3:
      category_split3.append(value[2])
    else:
      category_split3.append('Uncategorized')

test['CATEGORY1'] = category_split1
test['CATEGORY2'] = category_split2
test['CATEGORY3'] = category_split3
```



CATEGORY1	CATEGORY2	CATEGORY3
Beauty	Makeup	Lips
Electronics	Media	Blu-Ray
Kids	Toys	Arts & Crafts
Women	Women's Accessories	Wallets
Sports & Outdoors	Outdoors	Other

```
product_name=[]
for i in dataset["PRODUCT_NAME"]:
  name = re.sub('[^A-Za-z0-9]+', ' ', i)
  product_name.append(name.lower())

dataset["PRODUCT_NAME"]=product_name

product_desc=[]

for i in dataset["PRODUCT_DESCRIPTION"]:
  desc = re.sub('[^A-Za-z0-9]+', ' ', i)
  product_desc.append(desc.lower().strip())

dataset["PRODUCT_DESCRIPTION"]=product_desc

#dataset["PRODUCT_NAME"].value_counts()[:20]
```

```
product_name=[]
for i in test["PRODUCT_NAME"]:
  name = re.sub('[^A-Za-z0-9]+', ' ', i)
  product_name.append(name.lower())

test["PRODUCT_NAME"]=product_name

product_desc=[]

for i in test["PRODUCT_DESCRIPTION"]:

  desc = re.sub('[^A-Za-z0-9]+', ' ', i)
  product_desc.append(desc.lower().strip())

test["PRODUCT_DESCRIPTION"]=product_desc
```

Removing the special characters and punctuations from the Description and Name columns.

```
from gensim.parsing.preprocessing import remove_stopwords, STOPWORDS
product_desc=dataset["PRODUCT_DESCRIPTION"]
new_product_desc=[]
for desc in product_desc:
  new_product_desc.append(remove_stopwords(desc))
dataset["PRODUCT_DESCRIPTION"]=new_product_desc

product_desc=test["PRODUCT_DESCRIPTION"]
new_product_desc=[]
for desc in product_desc:
  new_product_desc.append(remove_stopwords(desc))
test["PRODUCT_DESCRIPTION"]=new_product_desc
```

This is the code for the removal of STOPWORDS from the Description Column.

```
def vectorize_column(column, vectorizer):
    vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=100000)
    vectorizer.fit(column)
    transformed_column = vectorizer.transform(column)
    return transformed_column, vectorizer
```

Function for TFIDF Vectorization.

```
cat1, cat1temp = vectorize_column(x_train['CATEGORY1'].values.astype('U'), None)
cat2, cat2temp = vectorize_column(x_train['CATEGORY2'].values.astype('U'), None)
cat3, cat3temp = vectorize_column(x_train['CATEGORY3'].values.astype('U'), None)
brand, brandtemp = vectorize_column(x_train['PRODUCT_BRAND'].values.astype('U'), None)
name, nametemp = vectorize_column(x_train['PRODUCT_NAME'], None)
desc, desctemp = vectorize_column(x_train['PRODUCT_DESCRIPTION'], None)
```

Applying the TFIDF Vectorization on the 3 sub-category columns, Brand Name, Description and Product Name columns. This is better than one-hot encoding.

```
from sklearn.linear_model import RidgeCV

y_train = dataset["PRODUCT_PRICE"].to_numpy().astype(np.float64)

ridge_cv = RidgeCV(alphas=(0.01, 0.1, 1.0, 10.0), cv=3)
ridge_cv.fit(x_train_final, y_train)

y_pred = ridge_cv.predict(x_test_final)
```

When we apply Ridge CV, we get the error to be 0.50331.

Goals for next Evaluation

1. Guessing the brand names of the products based on the keywords used in the Product Description.
2. Predicting log (price) rather than price.
3. Using Label Binarizer.
4. Applying better models.(Neural Networks)

Evaluation 3

- Label Binarizer was used to encode the text columns other than name and description.
- Instead of using Ridge Regression, a custom Neural Networks model was built using the Keras library.
- This model predicts the scaled version of log (p+1), rather than predicting p.

```
updatedbrands=[]
brands = test['PRODUCT_BRAND'].unique()
brands=list(brands)
brands.remove('Unbranded')
for row in test.itertuples():
  flag=False
  if str(row.PRODUCT_BRAND) == 'Unbranded':
    for brand in brands:
      if str(brand) in str(row.PRODUCT_DESCRIPTION):
        flag=True
        updatedbrands.append(str(brand))
        break
    if flag==False:
      updatedbrands.append('Unbranded')
  else:
    updatedbrands.append(str(row.PRODUCT_BRAND))

test["PRODUCT_BRAND"]=updatedbrands
```

Since the PRODUCT_BRAND column had about 5 lakh empty entries, this column was filled by checking whether the product description column contained a brand name.

```
shippingavailability = lb_shipping.transform(x_test['SHIPPING_AVAILABILITY'])
productcondition = lb_item_condition_id.transform(x_test['PRODUCT_CONDITION'])
cat1 = lb_cat1.transform(x_test['CATEGORY1'])
cat2 = lb_cat2.transform(x_test['CATEGORY2'])
cat3 = lb_cat3.transform(x_test['CATEGORY3'])
brand = lb_brand.transform(x_test['PRODUCT_BRAND'])

name, nametemp1 = vectorize_column(x_test['PRODUCT_NAME'], nametemp)
desc, desctemp1 = vectorize_column(x_test['PRODUCT_DESCRIPTION'], desctemp)
```

Label Binarizer is a more efficient form of one-hot encoding that can be used for columns that do not have too many unique values. This can be used for category and brand, but not for name and description. It also transforms a column into CSR form. The TF-IDF vectorizer was used for name and description.

```
def nn_model(X, y):

    model_in = Input(shape = (X.shape[1],), dtype = 'float32', sparse = True)
    out=Dense(1024, activation='relu')(model_in)
    out=Dense(512, activation='relu')(out)
    out=Dense(256, activation='relu')(out)
    out=Dense(128, activation='relu')(out)
    out=Dense(64, activation='relu')(out)
    out=Dense(32,activation='relu')(out)
    out=Dense(16,activation='relu')(out)
    model_out=Dense(1)(out)

    model = Model(model_in, model_out)
    model.compile(loss = 'mean_squared_error', optimizer = keras.optimizers.Adam(3e-3))
    model.fit(X, y, batch_size = 2048, epochs = 1, verbose = 1)
    return model
```

Using Keras, we built a Neural Networks model that takes in [number of columns] as input and gives scaled(log(p+1)) as output. The model has 7 hidden layers.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
y_train = scaler.fit_transform(np.log1p(dataset['PRODUCT_PRICE'].values.reshape(-1, 1)))

model = nn_model(x_train_final, y_train)

y_pred = model.predict(x_test_final)
y_pred=np.expm1(scaler.inverse_transform(y_pred.reshape(-1, 1)))
```

The PRODUCT_PRICE is transformed as shown above. The process is inversed after the model predicts the values of scale(log(p+1)).

Conclusion

Our model yielded a Mean squared log error of 0.42375 on the public leaderboard and 0.42567 on the private leaderboard, which put team SixtyForty on 2nd place.