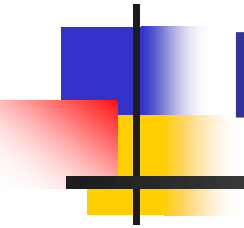
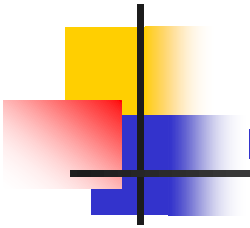


Kingsley Okoro

Absorb ML Challenge

Problem Statement



- Developing a predictive pricing model for items on an online platform.
- Dataset contained a mixture of structured and unstructured data(mainly text).
- Dataset is a table with product listing with each data point describing exactly one listing.

Dataset

```
#describe dataset
print (f"Dataset has {data.shape[0]} rows and {data.shape[1]} columns")
```

```
Dataset has 1482535 rows and 14 columns
```

```
#exploring the data
data.dtypes
```

```
id                       int64
title                   object
item_condition           int64
item_type               object
item_brand              object
price                  float64
shipping_category        int64
description             object
item_characteristic_n  float64
item_characteristic_p    int64
item_origin              int64
item_flag_available      int64
post_stats               int64
approved_poster           bool
dtype: object
```

- Dataset contains 1482535 data points and 14 features.
- Data set contains integer, float, string and bool data types.

Missing Data?

```
#how many missing data do we have in our data set?
total = data.isnull().sum().sort_values(ascending=False)
percent = (data.isnull().sum()/data.isnull().count()*100).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

	Total	Percent
item_brand	632682	42.675687
item_type	6327	0.426769
description	4	0.000270
approved_poster	0	0.000000
post_stats	0	0.000000
item_flag_available	0	0.000000
item_origin	0	0.000000
item_characteristic_p	0	0.000000
item_characteristic_n	0	0.000000
shipping_category	0	0.000000
price	0	0.000000
item_condition	0	0.000000
title	0	0.000000
id	0	0.000000

- Dataset contained missing values.
- The item_brand feature had most the missing values with (42%), item_type had 0.426% and description had only 4 observation missing.
- I decided to drop the item_brand feature (column) and observations for item_type and description

```
#dealing with missing data
data = data.drop((missing_data[missing_data['Total'] > 6327]).index,1)
data = data.drop(data.loc[data['item_type'].isnull()].index)
data = data.drop(data.loc[data['description'].isnull()].index)
print('Total number of missing data', data.isnull().sum().max()) #just
```

Categoriacal, Numerical and Discrete Observations

```
# find categorical variables
categorical = [var for var in data.columns if data[var].dtype=='O']
print('There are {} categorical variables'.format(len(categorical)))
# make a list of the numerical variables
numerical = [var for var in data.columns if data[var].dtype!='O']
print('There are {} numerical variables'.format(len(numerical)))
```

```
There are 5 categorical variables
There are 10 numerical variables
```

```
categorical
```

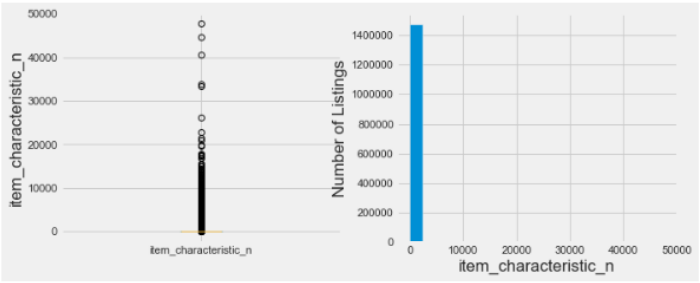
```
['title', 'description', 'item_type_a', 'item_type_b', 'item_type_c']
```

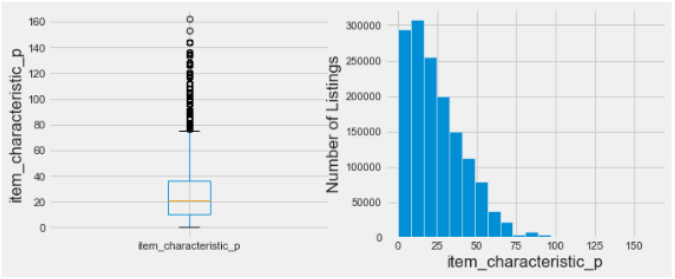
```
# let's visualise the values of the discrete variables
discrete = []

for var in numerical:
    if len(data[var].unique()) < 20:
        print(var, ' values: ', data[var].unique())
        discrete.append(var)
print()
print('There are {} discrete variables'.format(len(discrete)))
```

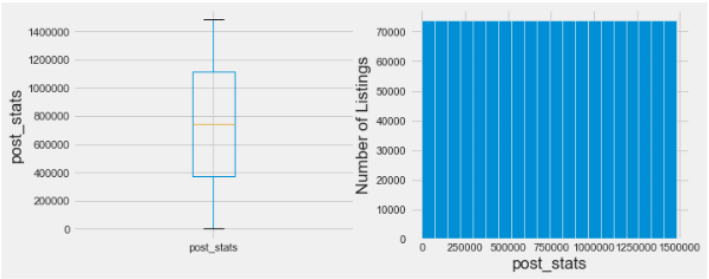
```
item_condition  values:  [1 2 3 4 5]
shipping_category  values:  [1 0]
item_origin  values:  [4 3 5 7 2 6 1 0 8]
item_flag_available  values:  [0 1]
approved_poster  values:  [0 1]

There are 5 discrete variables
```

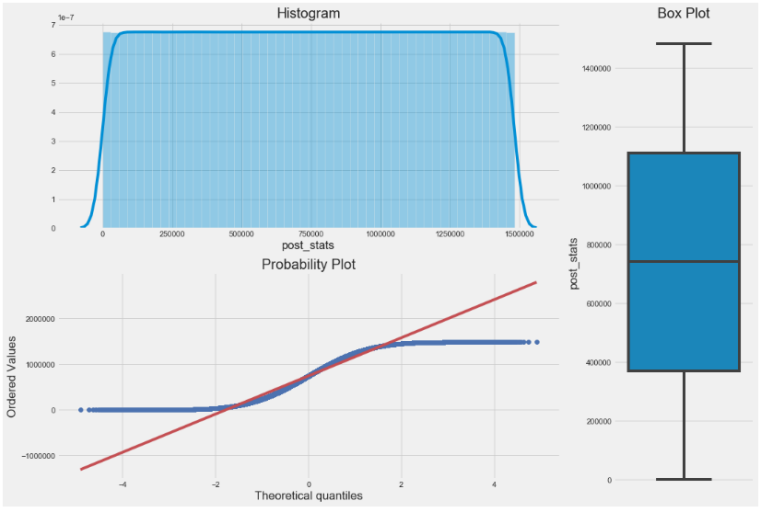
```
numerical = [var for var in numerical if var not in discrete and var not in [
    'id', 'price']]

print('There are {} numerical and continuous variables'.format(len(numerical)))
```

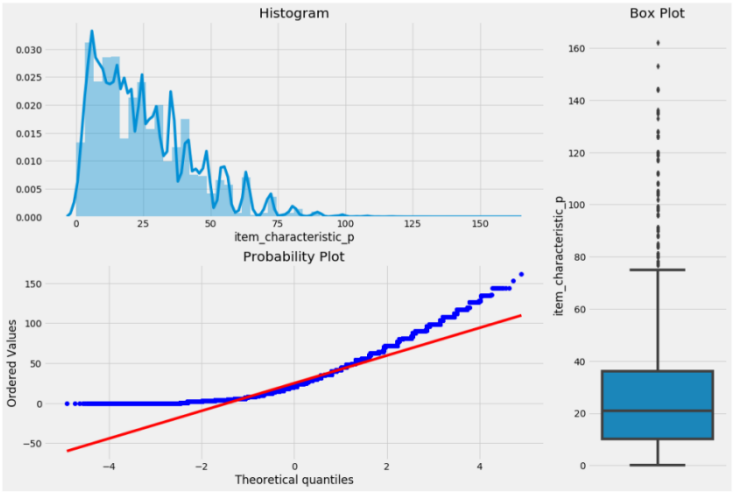
Exploratory Data Analysis (Visualizations)



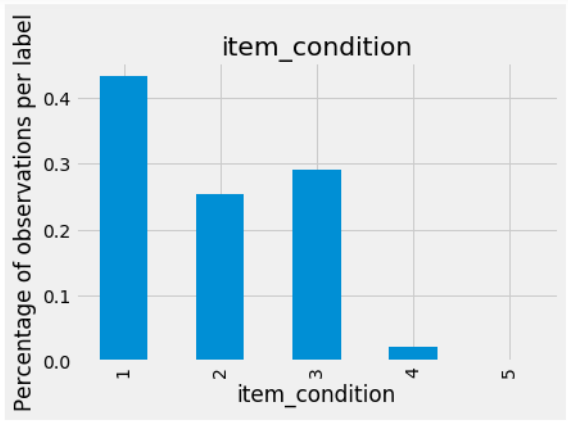


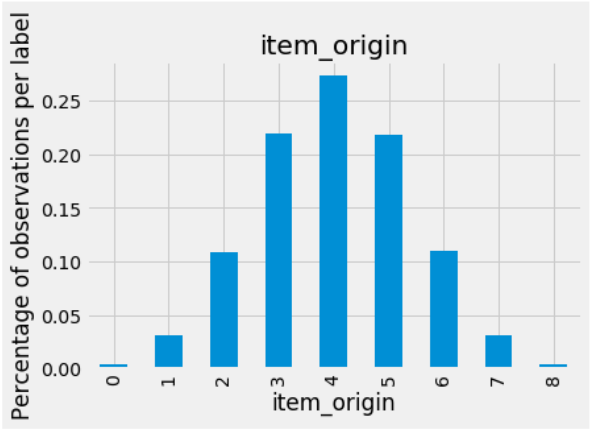


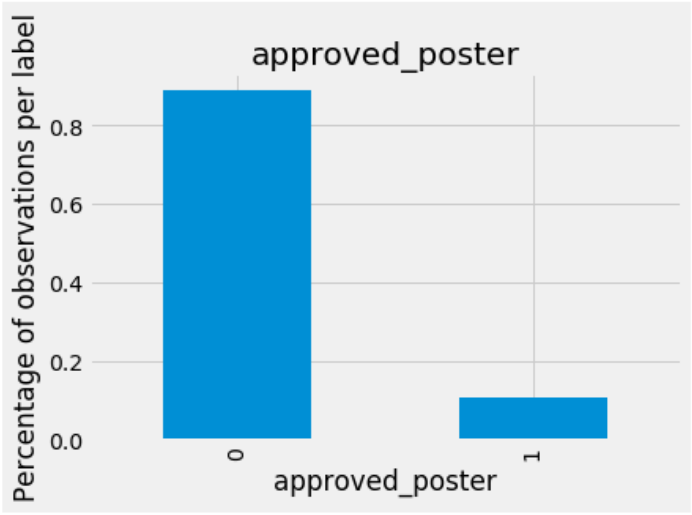




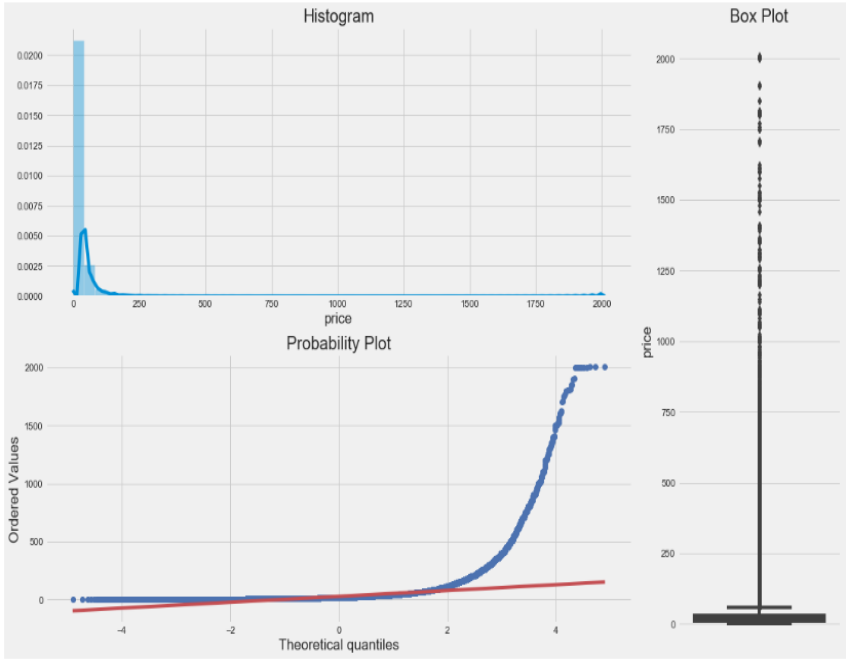
EDA Continued(other features Variables)







EDA continued(Target variable Price)



- Target variable price not normally distributed.
- Target variable is right skewed
- There are multiple outliers in the variable
- Linear models are based on the assumption that normality exists.

Feature Engineering

```
#tarnsforming the approved_poste column using integer encoding
data.approved_poster = data.approved_poster.apply(lambda x: 1 if x == True else 0)
```

Beauty/Makeup/Lips

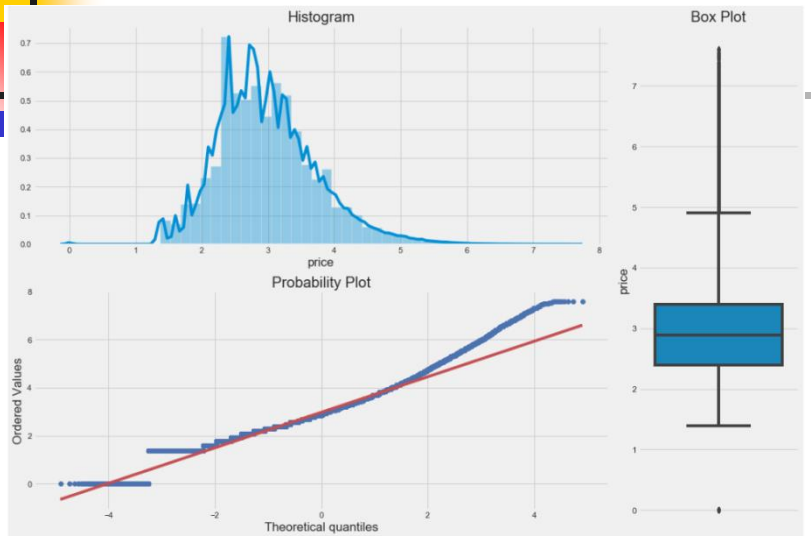
Electronics/Cameras & Photography/Camera & Pho...

Other/Books/Literature & Fiction

```
#split the item_type column into 3 seperate categories
new = data['item_type'].str.split("/", n = 2, expand = True)
#create new features
data['item_type_a'] = new[0]
data['item_type_b'] = new[1]
data['item_type_c'] = new[2]
data.drop(columns = ['item_type'], inplace = True )
```

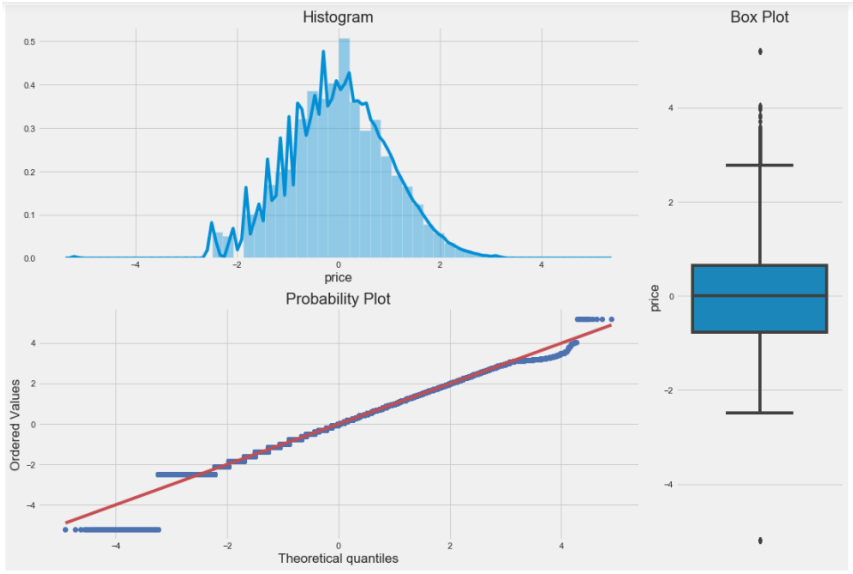
- Performed integer encoding
- Created new features from item_type column, item_type_a, item_type_b, item_type_a
- More feature engineering can be done leveraging domain knowledge.

Feature Engineering Cont'd



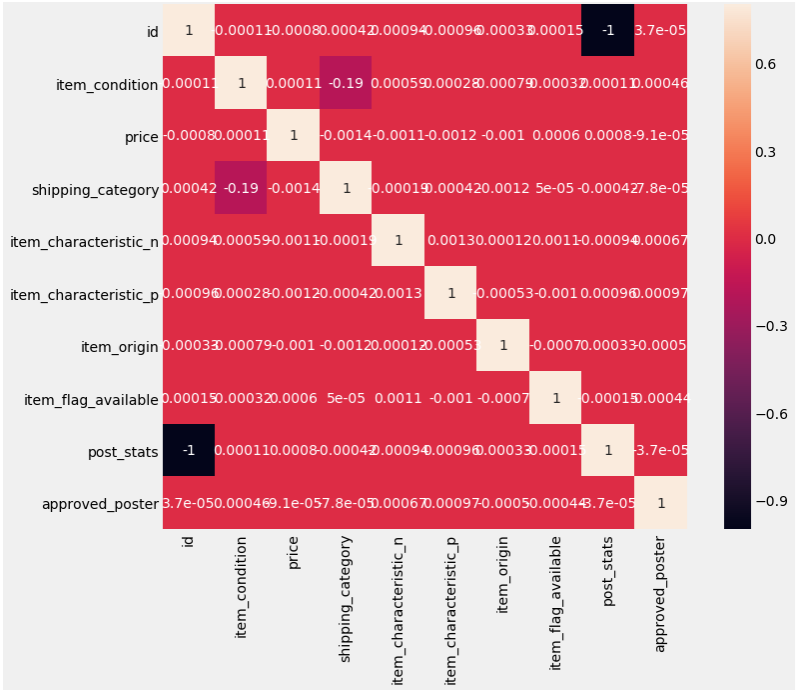

```
## trainsforming target variable using logrithimic transformation,
transformed_target = np.log1p(data["price"])
data_tdf = pd.DataFrame(transformed_target, columns = ['price'])
## Plotting the newly transformed response variable
plotting_3_chart(data_tdf, 'price')
```

```
#converting the target varaible to a normal distribution using Quantile Transformer
from sklearn.preprocessing import QuantileTransformer
transformed_target = data['price']
Transformer = QuantileTransformer(output_distribution='normal',random_state=42)
transformed_target = Transformer.fit_transform(np.array(transformed_target).reshape(-1,1))
transformed_target = np.ravel(transformed_target)
data_tdf = pd.DataFrame(transformed_target, columns = ['price'])
plotting_3_chart(data_tdf, 'price')
```



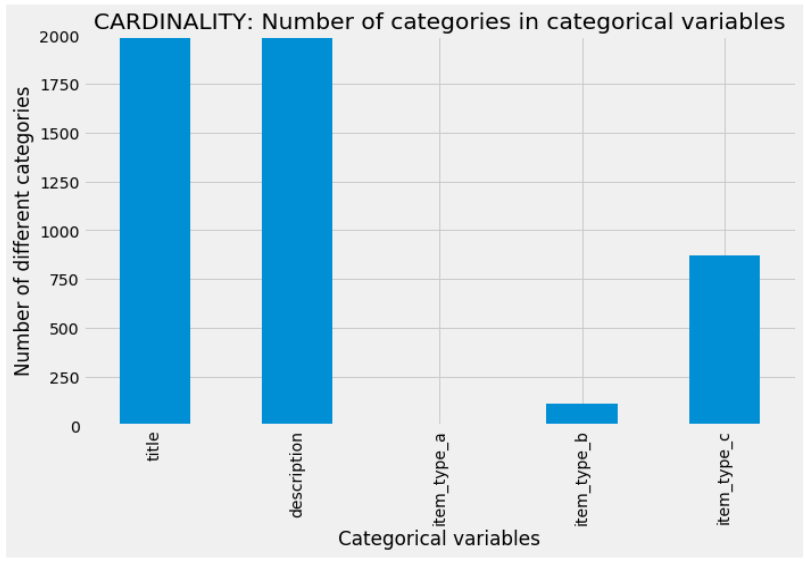
- Tried using logarithmic transformation and quantile transformer to transform data to follow a gaussian distribution .

Co-Linearity?



- Heatmap with pearson correlation to investigate correlation between independent variables.

Cardinality




```
print("Cardinality")
print(f' Item_type_a : {len(data.item_type_a.unique())}')
print(f' title : {len(data.title.unique())}')
print(f' description : {len(data.description.unique())}')
print(f' Item_type_b : {len(data.item_type_b.unique())}')
print(f' Item_type_c : {len(data.item_type_c.unique())}')
```

```
Cardinality
 Item_type_a : 10
 title : 1220177
 description : 1276149
 Item_type_b : 113
 Item_type_c : 871
```

- Drop features with high cardinality as they distort linear models

Final feature Selection

```
#Categorical column encoding using Pd . get Dummies
final_features = pd.get_dummies(data).reset_index(drop=True)
final_features.shape
```

```
(1476204, 19)
```

```
#splitting variable into features and Target
features = final_features.drop('price', axis=1)
target = transformed_target
```

```
features.shape, target.shape
```

```
((1476204, 18), (1476204,))
```

- Use Pd.get_dummies similar to one_hot or Lable Encoding to handle categorical data.

Model Training(simple Linear Regression)

```
## Train test s
from sklearn.model_selection import train_test_split
## Train test split follows this distinguished code pattern and helps creating train and test set to build machine learning.
X_train, X_test, y_train, y_test = train_test_split(features, target,
                                                    test_size = 0.33,
                                                    random_state = 0)
```

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((989056, 33), (989056,), (487148, 33), (487148,))
```

```
listing_pipeline = Pipeline([
    # feature Scaling - section 10
    ('scaler', StandardScaler()),

    # regression
    ('linear regression', LinearRegression(normalize=True, n_jobs=-1))
])
```

```
# let's fit the pipeline
listing_pipeline.fit(X_train, y_train)

# let's get the predictions
X_train_preds = listing_pipeline.predict(X_train)
X_test_preds = listing_pipeline.predict(X_test)
```

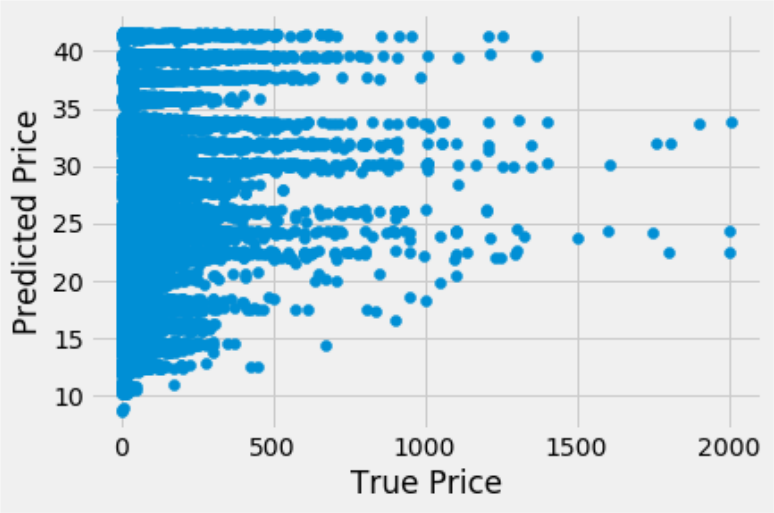
Model Evaluation(skewed target)

```
from math import sqrt
print('train mse: {}'.format(mean_squared_error(y_train, X_train_preds)))
print('train rmse: {}'.format(sqrt(mean_squared_error(y_train, X_train_preds))))
print('train r2: {}'.format(r2_score(y_train, X_train_preds)))
print()
print('test mse: {}'.format(mean_squared_error(y_test, X_test_preds)))
print('test rmse: {}'.format(sqrt(mean_squared_error(y_test, X_test_preds))))
print('test r2: {}'.format(r2_score(y_test, X_test_preds)))
```

```
train mse: 1449.3444840331595
train rmse: 38.07025720996851
train r2: 0.027441013870676323

test mse: 1450.1600542948488
test rmse: 38.08096708717951
test r2: 0.026962575311615833
```

```
plt.scatter(y_test,X_test_preds)
plt.xlabel('True Price')
plt.ylabel('Predicted Price')
```

```
Text(0, 0.5, 'Predicted Price')
```



Model Evaluation(normalized target)

```
from math import sqrt
print('train mse: {}'.format(mean_squared_error(y_train, X_train_preds)))
print('train rmse: {}'.format(sqrt(mean_squared_error(y_train, X_train_preds))))
print('train r2: {}'.format(r2_score(y_train, X_train_preds)))
print()
print('test mse: {}'.format(mean_squared_error(y_test, X_test_preds)))
print('test rmse: {}'.format(sqrt(mean_squared_error(y_test, X_test_preds))))
print('test r2: {}'.format(r2_score(y_test, X_test_preds)))
```

```
train mse: 0.9011830255741317
train rmse: 0.9493066025126612
train r2: 0.0986310997763663

test mse: 0.9018676133557288
test rmse: 0.9496671065987959
test r2: 0.09862261343834355
```

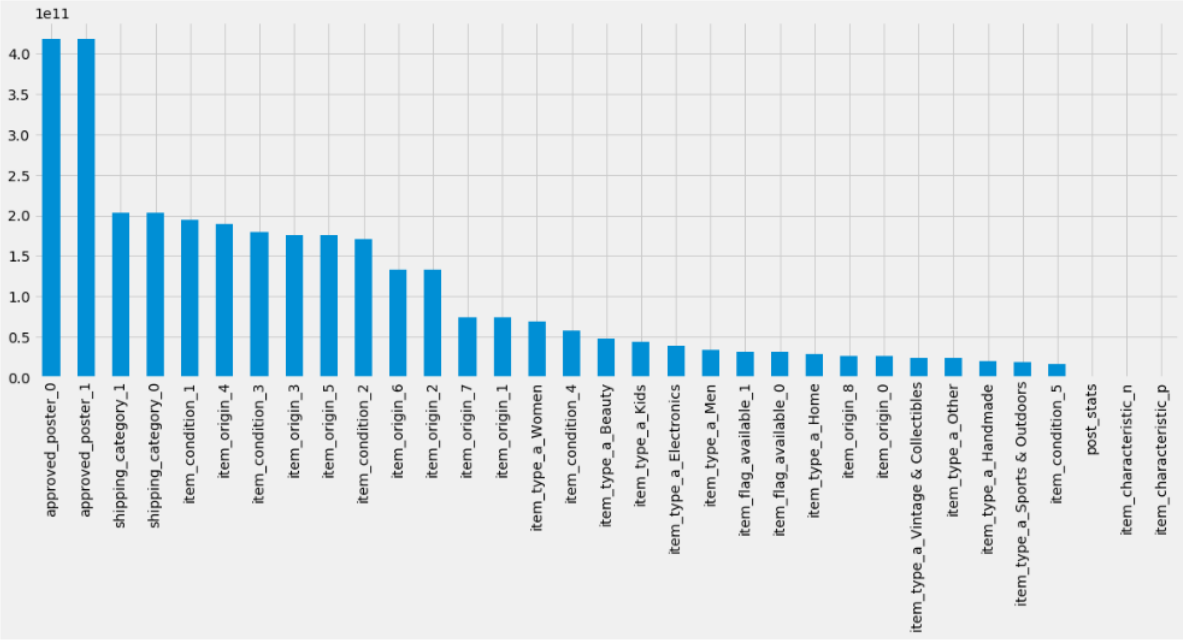
```
plt.scatter(y_test,X_test_preds)
plt.xlabel('True Price')
plt.ylabel('Predicted Price')
```

```
Text(0, 0.5, 'Predicted Price')
```

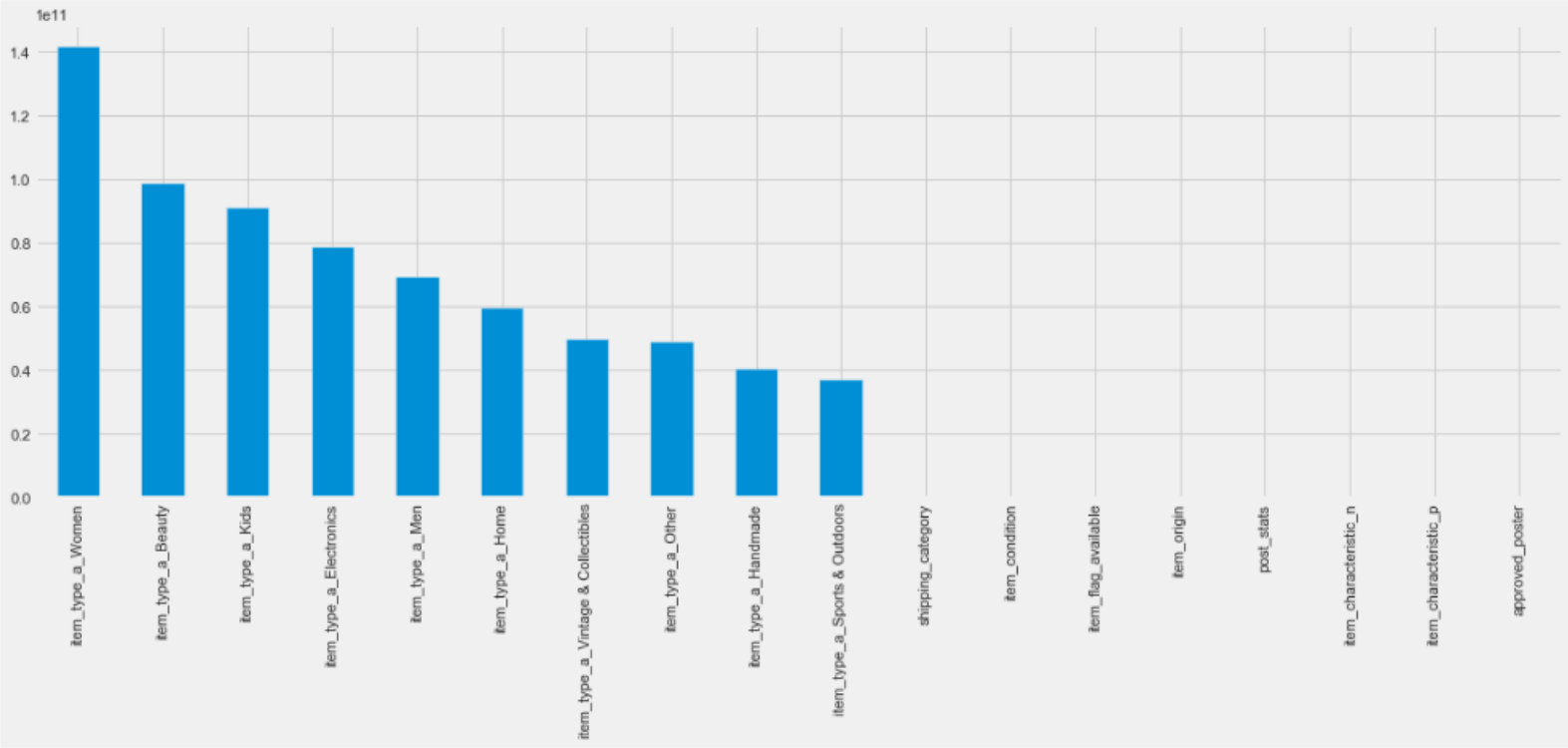
Feature Importance

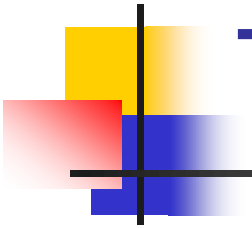
```
# let's explore the importance of the features
# the importance is given by the absolute value of the coefficient
# assigned by the Linear regression model

importance = pd.Series(np.abs(listing_pipeline.named_steps['linear regression'].coef_))
importance.index = list(X_train.columns)
importance.sort_values(inplace=True, ascending=False)
importance.plot.bar(figsize=(18,6))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7b50e860>
```



Feature Importance





Thank You

- Questions?