Mercari's Price Suggestion and Modelling

A Project by

Raj Waje

Rohit Amalnerkar

Raghavendra Srinivasan

Neha Patil

Table of Contents

Declaration

We declare that Mercari's Price Suggestion and Modelling project represents our ideas in our own words and whatever we learnt in the Class. We have combined the knowledge gained through some projects and some online resources like Kaggle, Stack Overflow, Wikipedia, and YouTube for reference.

Raj Waje

Rohit Amalnerkar

Raghavendra Srinivasan

Neha Patil

Date: 18/06/2020

Acknowledgement

We would like to thank **Ms. Ritu Sekhon** for wonderful teachings in Data Science, **Ms. Prachi Samant** for her teachings in R programming and **Ms. Nikita Tandel** for her deep and practical insights which helped us in a better understanding of the technical and practical approach required to keep while solving real world problems and big tedious datasets.

List of Figures

CHAPTER 1

INTRODUCTION

1. Introduction:

The modern age is the age of machine intelligence. These are such buzzwords that has taken the world by storm and almost every avenue has enjoyed the flavor of Machine Learning in some way. In this report we are going to take you through a real-world data science problem which we have picked from Kaggle's live competition and will demonstrate our way of solving it.

This case study solves everything right from scratch. So, you will get to see each and every phase of how in the real world, a case study is solved. Before we talk about our approach of solving the problem, let us know what **Mercari** is.

About Mercari:

Mercari is an online shopping marketplace which is powered by one of the biggest community of Japan where users can sell pretty much anything. The community wants to offer price suggestions to the sellers but is a tough task as the sellers are enabled to put just about anything, or any bundle of things, on Mercari's marketplace

2. Problem Statement:

- It can be hard to know how much something's really worth. Small details can mean big differences in pricing. For example, one of these sweaters cost 335 dollars and the other cost 9.99 dollars. Can you guess which one's which?

Sweater A: Vince, Long sleeve, Turtle neck pullover sweater, Black, Size L, Great condition

Sweater B: St. John's Bay Long sleeve, Turtle neck Pullover sweater, Size L, Great condition

- Hence, it's harder to guess or decide what price which product should have
- Product pricing gets even harder at scale, considering just how many products are sold online. Clothing has strong seasonal pricing trends and is heavily influenced by brand names, while electronics have fluctuating prices based on product specs.

3. Challenges to solve:

- The task of this Project is to build an algorithm that suggests the right product prices for shopping app from product name, user inputted text descriptions of the product, category name, brand name, item condition, and shipping information
- The goal is about creating a model that would help sellers to price their products. Pricing should be intermediate between sellers and buyers
- Making it easy for the sellers to mark the prices of new products on the website, after taking into account the previously collected data
- Understand what the data is trying to tell us through EDA

4. Our Objectives:

1. • Data Cleaning • Pre-processing • Feature Engineering
2. Exploratory Data Analysis:
    - Gathering as many insights as possible through Exploratory Data Analysis.
    - Trying to see through Descriptive Statistics that what our data is trying to tell us and understand the significance of each and every variable.
3. Text Processing:
    - Natural Language Processing (NLP)
    - Tokenizing and tf-idf algorithm
    - K-means Clustering
4. The given problem is exactly about predicting the price, which is a real-valued value, thus it falls into the Regression Category
5. Trying to provide some interpretability and evolving our ways to handle the upcoming problems as we proceed through our project

5. Approach and Models used:

5.1 Approach:

- **Machine learning** is the fastest growing field in the world
- There exists a bunch of algorithms in ML incorporated for different scenarios. Some of them work on a specific set of problem and to some we have to fine tune to make it work
- There is no such ML algorithm that gives us a deterministic result no matter what the model is and how precisely we calibrate it
- However, the most important step before getting into ML and interpreting which model to apply, one must always try and understand what the data is trying to tell us. This is possible through EDA

5.2 Models used:

Basic Regression Algorithms:

1. Ordinary Least Squares Regression
2. Linear Regression

Algorithms that require Fine Tuning:

1. Ridge Regression
2. Stochastic Gradient Descent Regressor
3. Random Forest Regressor

6. Data Description:

- **train_id** - the id of the product
- **name** - the name of the product
- **item_condition_id** - the condition of the product provided by the seller

According to information available on Mercari Website

  - 1 stands for New
  - 2 stands for Fairly New
  - 3 stands for Good
  - 4 stands for Bad
  - 5 stands for Very Poor

- **category_name** - category of the product
- **brand_name** - brand name of the product
- **price** - the price that the product was sold for. (This is the target variable that we will predict) The unit is USD ($)
- **shipping** - 1 if shipping fee is paid by seller and 0 by buyer
- **item_description** - the full description of the item

Independent variables: train_id, name, item_condition_id, category_name, brand_name, shipping, item_description

Target Variable: price

We will build various supervised machine learning regression models and see which succeeds in solving the given mapping between the input variables and the price feature in the best way. Let's begin in a step-by-step manner

CHAPTER 2

PRE - PROCESSING

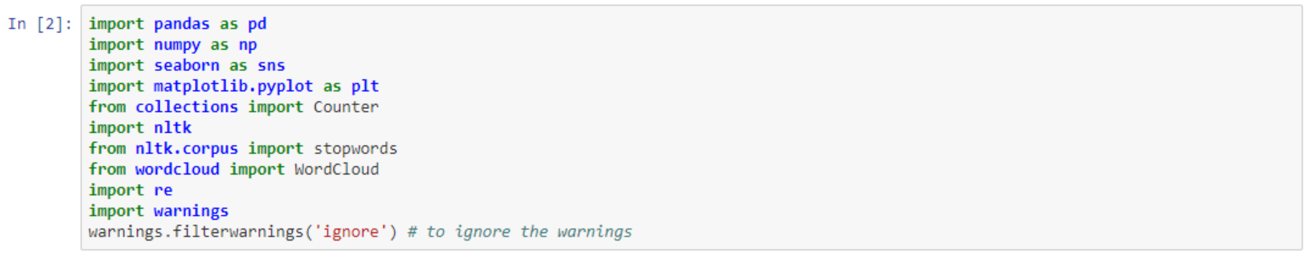
&

EXPLORATORY DATA ANALYSIS

Importance of EDA:

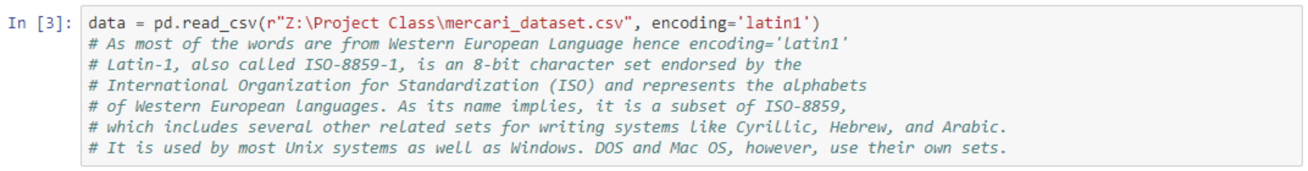
- The very first step in solving any case study in data science is to properly look and analyze the data you have
- It helps us gain valuable insights into the pattern and information it has to convey
- Statistical tools have a big role in proper visualization of the data
- Even though it is considered not a very important part of solving a problem, but successful data scientists and ML engineers spend maximum part of solving a problem by analyzing the data they have
- Proper EDA gives interesting features of your data which in turn influences our data preprocessing and model selection criterion as well

Step 1: Importing the important libraries and data



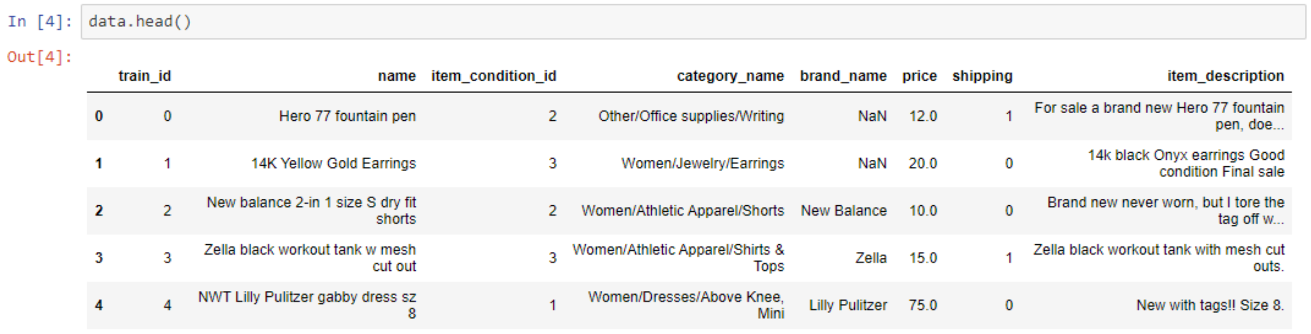
```
In [2]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        from collections import Counter
        import nltk
        from nltk.corpus import stopwords
        from wordcloud import WordCloud
        import re
        import warnings
        warnings.filterwarnings('ignore') # to ignore the warnings
```

Loading the data:



```
In [3]: data = pd.read_csv(r"Z:\Project Class\mercari_dataset.csv", encoding='latin1')
        # As most of the words are from Western European Language hence encoding='latin1'
        # Latin-1, also called ISO-8859-1, is an 8-bit character set endorsed by the
        # International Organization for Standardization (ISO) and represents the alphabets
        # of Western European languages. As its name implies, it is a subset of ISO-8859,
        # which includes several other related sets for writing systems like Cyrillic, Hebrew, and Arabic.
        # It is used by most Unix systems as well as Windows. DOS and Mac OS, however, use their own sets.
```

Step 2: Checking some important aspects of the data



```
In [4]: data.head()
```

Out[4]:

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	0	Hero 77 fountain pen	2	Other/Office supplies/Writing	NaN	12.0	1	For sale a brand new Hero 77 fountain pen, doe...
1	1	14K Yellow Gold Earrings	3	Women/Jewelry/Earrings	NaN	20.0	0	14k black Onyx earrings Good condition Final sale
2	2	New balance 2-in 1 size S dry fit shorts	2	Women/Athletic Apparel/Shorts	New Balance	10.0	0	Brand new never worn, but I tore the tag off w...
3	3	Zella black workout tank w mesh cut out	3	Women/Athletic Apparel/Shirts & Tops	Zella	15.0	1	Zella black workout tank with mesh cut outs.
4	4	NWT Lilly Pulitzer gabby dress sz 8	1	Women/Dresses/Above Knee, Mini	Lilly Pulitzer	75.0	0	New with tags!! Size 8.

Step 3: Checking some important aspects of the data

```
In [5]: # Data Dimensions
        data.shape
```

```
Out[5]: (148253, 8)
```

As the orignal data had over 1.4 million observations, we only take the 10% out of it in the csv file above

```
In [6]: # Data Types
        data.dtypes
```

```
Out[6]: train_id               int64
        name                  object
        item_condition_id      int64
        category_name         object
        brand_name            object
        price                float64
        shipping               int64
        item_description      object
        dtype: object
```

```
In [7]: # Columns in the data
        data.columns
```

```
Out[7]: Index(['train_id', 'name', 'item_condition_id', 'category_name', 'brand_name',
               'price', 'shipping', 'item_description'],
              dtype='object')
```

Step 4: Checking and Handling Missing Values

```
In [8]: # Checking the NAs in the data
        data.isnull().sum()
```

```
Out[8]: train_id                 0
        name                     0
        item_condition_id        0
        category_name          641
        brand_name           62991
        price                    0
        shipping                 0
        item_description         0
        dtype: int64
```

```
In [9]: # making a copy of the data as backup
        data1 = data.copy()
```

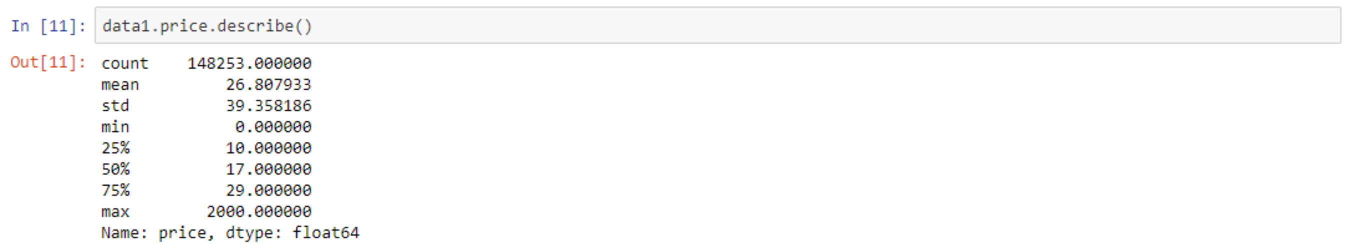
```
In [10]: for value in ['category_name']:
             data1[value].fillna(value='Other', inplace=True)
             # replacing by "Other" because there already exists a category by the same name

         for value in ['brand_name']:
             data1[value].fillna(value='Unknown', inplace=True)

         # Rechecking the null values
         data1.isnull().sum()
```

```
Out[10]: train_id             0
         name                 0
         item_condition_id    0
         category_name        0
         brand_name           0
         price                0
         shipping             0
         item_description     0
         dtype: int64
```

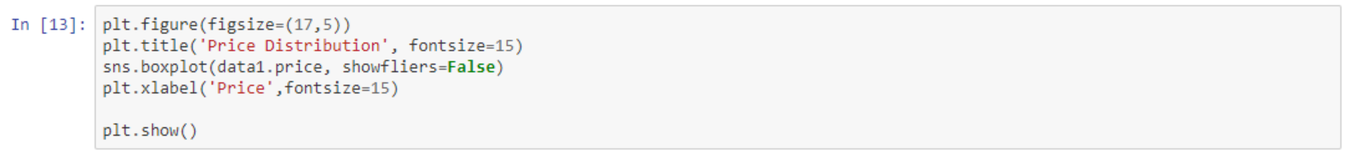
Analysis on each variable:

1 Price

This is our response variable, the value which we are trying to predict for the seller. According to a Linear Regression Assumption of Normality, our response/target variable should be Normally Distributed. 2nd important thing told by many ML experts is that while implementing ML models on this is we cannot remove the outliers from a target variable as the model should be able to justify them as well



```
In [11]: data1.price.describe()

Out[11]: count    148253.000000
         mean         26.807933
         std          39.358186
         min           0.000000
         25%          10.000000
         50%          17.000000
         75%          29.000000
         max        2000.000000
         Name: price, dtype: float64
```



```
In [13]: plt.figure(figsize=(17,5))
         plt.title('Price Distribution', fontsize=15)
         sns.boxplot(data1.price, showfliers=False)
         plt.xlabel('Price',fontsize=15)

         plt.show()
```

The **showfliers = False** command would ignore the outliers and plot the graph

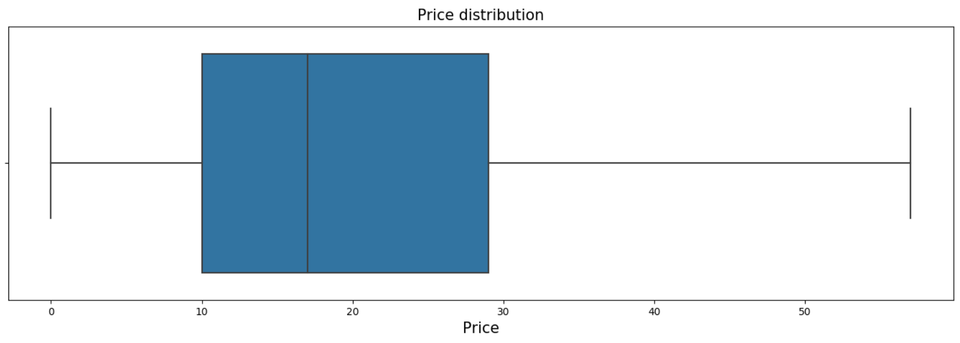


Fig 2.1: - Price Distribution (Box Plot)

Insight 1 from Price

- mean = 26, median = 17, min = 0, max = 2000
- 25% of the products are priced below 10 $
- 50% of products are priced below 17 $
- 75% of products are priced below 29 $Also, the maximum price that any product has is 2000 $

1.2 Distribution of the price variable

```
In [67]: plt.plot(figsize=(30,20))
         sns.distplot(data1['price'])
```

```
Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x2300020b4c8>
```

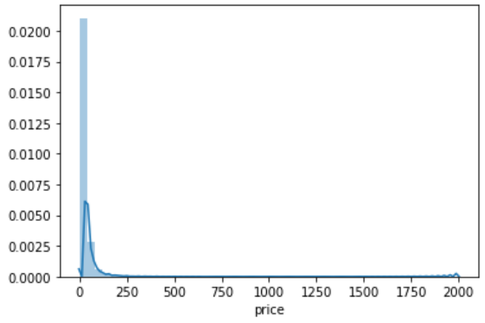


Fig 2.1.1(a): - Price Distribution (Distribution Plot)

- It can be concluded that the distribution of the price variable is heavily right-skewed.
- The price variable follows a skewed distribution and in order to make errors on low price product more relevant than for higher prices, we take the log transform
- log1p = log(p) + 1, we add 1 to avoid zero(log0=infinity) and negative values

1.3 Now let's compare both the graphs side by side

```
In [14]: plt.subplot(1, 2, 1)
         (data1['price']).plot.hist(bins=50, figsize=(12, 6), edgecolor = 'white', range = [0, 250])
         plt.xlabel('price', fontsize=12)
         plt.title('Price Distribution', fontsize=12)

         plt.subplot(1, 2, 2)
         np.log1p(data1['price']).plot.hist(bins=50, figsize=(12, 6), edgecolor='white')
         plt.xlabel('log(price)+1', fontsize=12)
         plt.title('Price Distribution', fontsize=12)
```

```
Out[14]: Text(0.5, 1.0, 'Price Distribution')
```

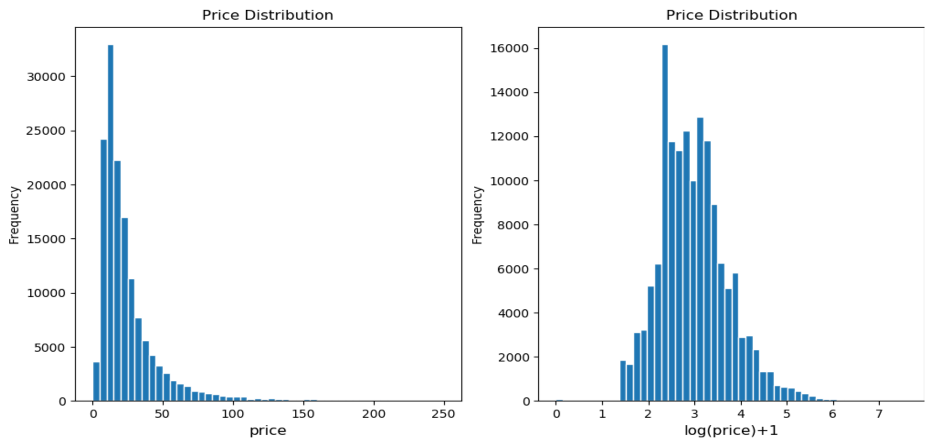


Fig 2.1.1(b): - Price Distribution (Histogram)

2. Shipping

These are the shipping fees paid by the buyer and seller at the time of shipping

1 means seller is paying and 0 means buyer is paying

```
In [15]: data1['shipping'].value_counts(normalize=True) # to show as a percentage of total values

Out[15]: 0    0.553587
         1    0.446413
         Name: shipping, dtype: float64
```

```
In [16]: index = ['Buyer','Seller']
         values =  data1['shipping'].value_counts()
         plt.figure(figsize=(7,4))
         plt.pie(values,startangle=90,autopct='%0.1f%%',explode=(0,0.1))
         plt.legend(title = "Shipping Paid by",loc = "upper right",labels= index,fontsize=10)
         plt.tight_layout()
         plt.title("Analysis on Shipping Paid",fontsize=20)

Out[16]: Text(0.5, 1, 'Analysis on Shipping Paid')
```

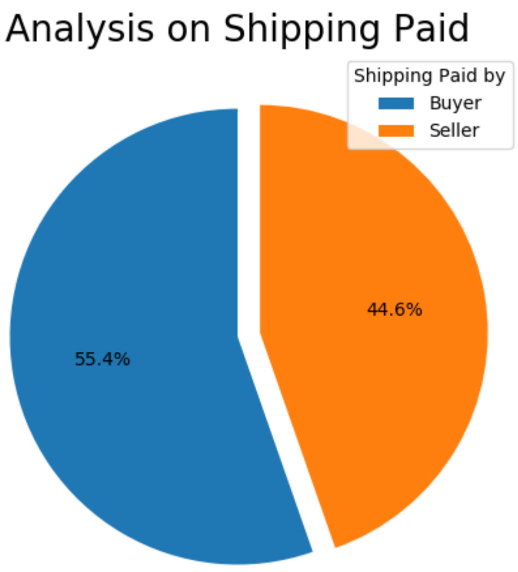


Fig 2.2: - Analysis on Shipping Paid (Pie Chart)

Insight 1 from shipping:

- 0 - 82071 times(buyer charged) 55.53%
- 1 - 66182 times(seller charged) 44.64%
- Over 55% of items' shipping fee were paid by the buyers

Suggestion to the seller:

- The above insight matches with our perception that the sellers need to keep a lower price to compensate for the additional shipping

2.1 Shipping vs Price:

To check how shipping is related to the price

```
In [17]: buyer_charged = data1.loc[data1['shipping'] == 0, 'price']
         seller_charged = data1.loc[data1['shipping'] == 1, 'price']


         fig, ax = plt.subplots(figsize=(14, 8))
         ax.hist(buyer_charged, bins=30, range=[0, 100], label='Price when buyer paid shipping', alpha=0.5, color='b')
         ax.hist(seller_charged, bins=30, range=[0, 100], label='Price when seller paid shipping', alpha=0.5, color='r')
         plt.title('Price Distribution by shipping type', fontsize = 20)
         plt.xlabel('Price', fontsize = 15)
         plt.ylabel('No. of Items', fontsize = 15)
         plt.legend(fontsize = 15)
         plt.show()
```

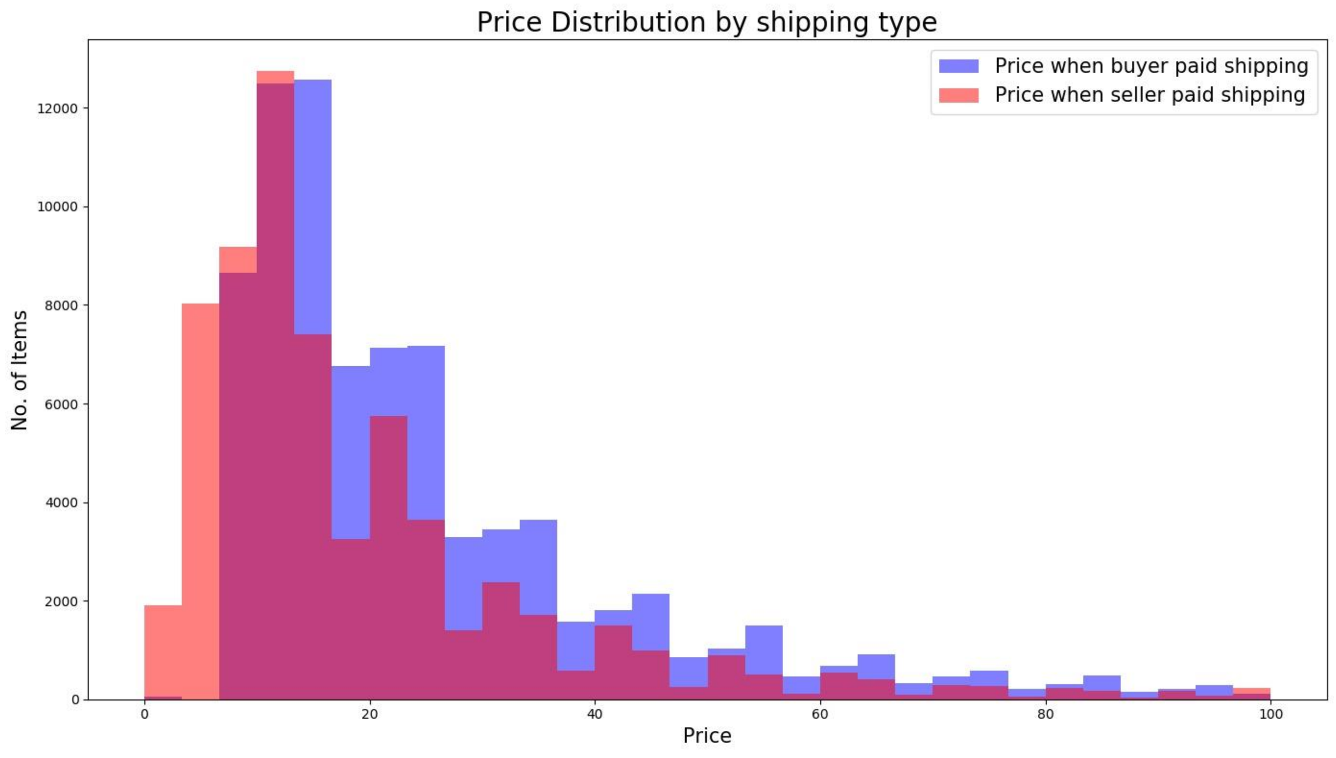


Fig 2.2.1: - Price distribution by shipping paid (Histogram)

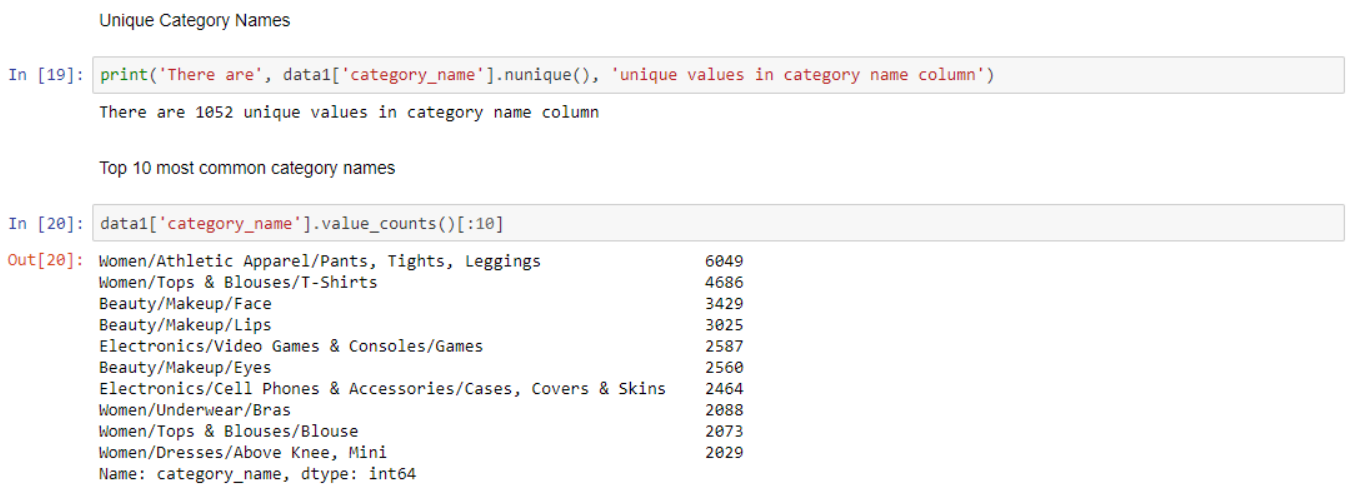
Insight 2 from shipping:

- The price when buyer pays is high compared to when seller pays

Checking the actual values:

```
In [18]: # average price for shipping for seller and buyer
         print('The average price is {}'.format(round(seller_charged.mean(), 2)), 'if seller pays shipping')
         print('The average price is {}'.format(round(buyer_charged.mean(), 2)), 'if buyer pays shipping')

         The average price is 22.82 if seller pays shipping
         The average price is 30.02 if buyer pays shipping
```

3. Category Name



Unique Category Names

```
In [19]: print('There are', data1['category_name'].nunique(), 'unique values in category name column')
```

```
There are 1052 unique values in category name column
```

Top 10 most common category names

```
In [20]: data1['category_name'].value_counts()[:10]
```

```
Out[20]: Women/Athletic Apparel/Pants, Tights, Leggings                 6049
         Women/Tops & Blouses/T-Shirts                                  4686
         Beauty/Makeup/Face                                             3429
         Beauty/Makeup/Lips                                             3025
         Electronics/Video Games & Consoles/Games                       2587
         Beauty/Makeup/Eyes                                             2560
         Electronics/Cell Phones & Accessories/Cases, Covers & Skins    2464
         Women/Underwear/Bras                                           2088
         Women/Tops & Blouses/Blouse                                    2073
         Women/Dresses/Above Knee, Mini                                 2029
         Name: category_name, dtype: int64
```

General Observations from the above insight:

- It is seen in this table that a subcategory in the category name is separated by a slash "/"
- For ex. Beauty/Makeup/Face and Beauty/Makeup/Lips
- This means **Beauty** category has a subcategory **MakeUp** and this sub category **MakeUp** is further divided into **Face** and **Lips**
- It is also observed that **Women** apparel has the maximum number of items followed by any other category.
- The category names are listed by '/' delimiter which tells about the main category, sub-category 1 and sub-category 2 of the products.
- Therefore, to get better idea of each product, we will do feature engineering here and split the category name into 3 different columns namely, '**Main_categ**', '**sub_categ1**' and '**sub_categ2**'.

Divide category names into Main category, Sub-category 1 and Sub-category 2

```
In [21]: temp = data1[data1['brand_name']!='Unknown'] # remove Unknown coz it represents missing values
         data1[['Main_categ','sub_categ1','sub_categ2']] = data1.category_name.str.split("/",expand = True,n= 2)
         for i in ['sub_categ1','sub_categ2']:
             data1[i].fillna(value = "Label not given", inplace=True)
```

After splitting the `category_name` column, the unique items that I have in each of the newly formed columns is listed below:

```
In [22]: print('There are', data1['Main_categ'].nunique(), 'unique values in Main category')
         print('There are', data1['sub_categ1'].nunique(), 'unique values in Sub-category 1')
         print('There are', data1['sub_categ2'].nunique(), 'unique values in Sub-category 2')
```

```
There are 10 unique values in Main category
There are 114 unique values in Sub-category 1
There are 741 unique values in Sub-category 2
```

3.1 (a) Main Category

Let’s find out which of the products rank the highest in terms of frequency of occurrence:

```
In [23]: maincat_count = data1.Main_categ.value_counts()

         plt.figure(figsize=(15, 10))
         sns.barplot(maincat_count.index[0:11], maincat_count[0:11])
         plt.title('Main category Analysis',fontsize = 20)
         plt.xlabel('Main category',fontsize = 15)
         plt.ylabel('Count',fontsize = 15)
         plt.xticks(rotation=15, fontsize=12)
         plt.show()
```

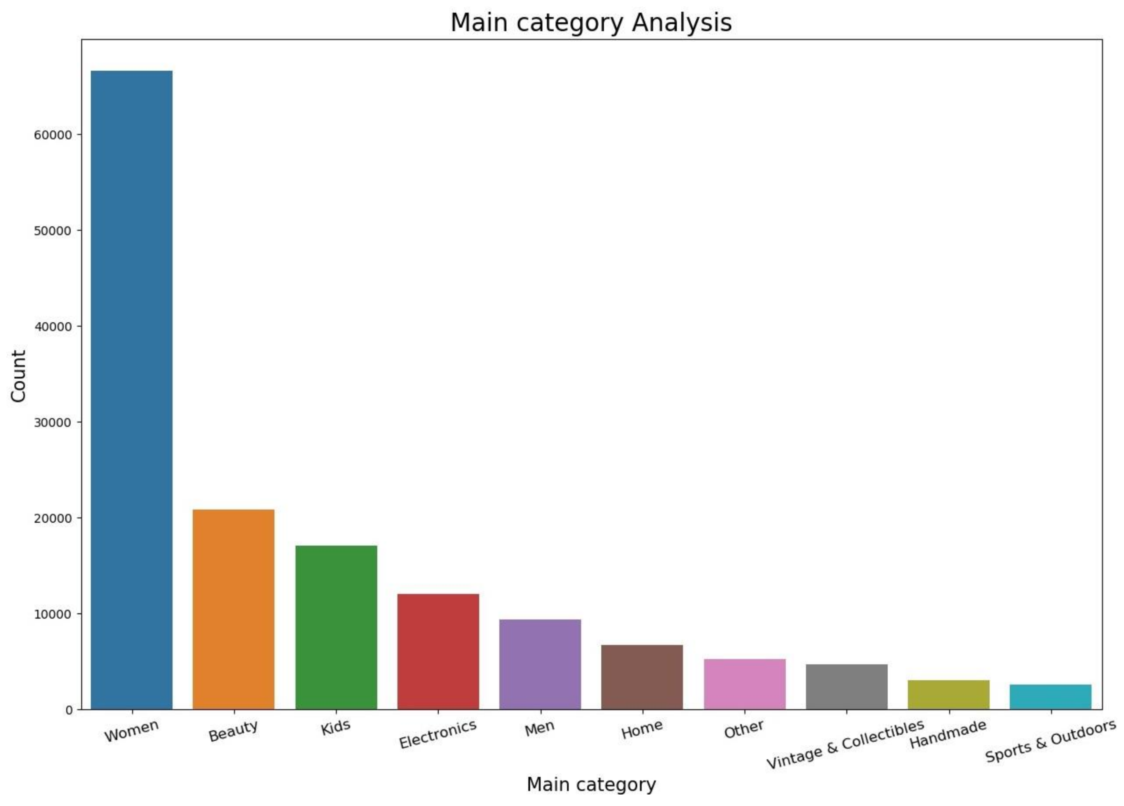


Fig 2.3.1(a): - Main Category Analysis (Bar Plot)

Insight 1 from Main Category:

- It can be said that **Women** products occur with the maximum frequency, followed by **Beauty** products. The 3rd largest general category is owned by **Kids** products

3.1 (b) Main Category vs Price

```
In [24]: temp = data1[data1['price'] < 80]
         plt.figure(figsize=(15, 10))
         sns.boxplot(temp['Main_categ'], temp['price'])
         plt.title('BoxPlot of Main category vs Price', fontsize = 20)
         plt.xlabel('Main Category',fontsize = 15)
         plt.ylabel('Price',fontsize = 15)
         plt.xticks(rotation = 15, fontsize=12)
         plt.show()
```

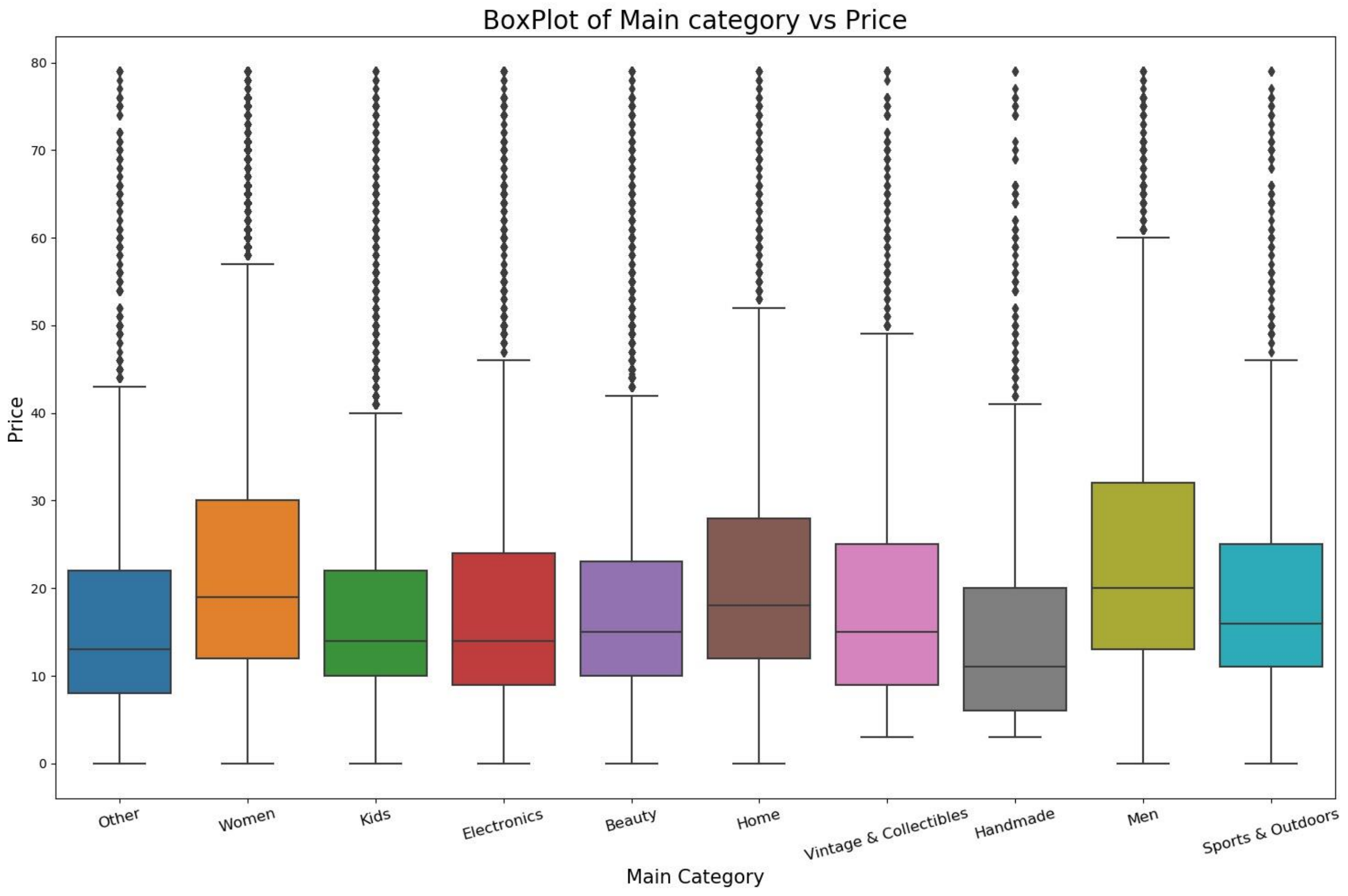


Fig 2.3.1(b): - Main Category vs Price (Box Plot)

```
In [25]: temp.groupby(['Main_categ'])['price'].agg('median')

Out[25]: Main_categ
         Beauty                    15.0
         Electronics               14.0
         Handmade                  11.0
         Home                      18.0
         Kids                      14.0
         Men                       20.0
         Other                     13.0
         Sports & Outdoors         16.0
         Vintage & Collectibles    15.0
         Women                     19.0
         Name: price, dtype: float64
```

Insight 2 from Main category:

- Although there are maximum products purchased in the **Women's** category, the price of **Men** category products(20$) is almost as expensive as the products in **Women** category(19$)

3.2 (a) Sub Category 1

```
In [26]: index = []
         [index.append(key) for key, value in Counter(data1['sub_categ1']).most_common()]
         top_10 = index[:10]
         temp = data1[data1['sub_categ1'].isin(top_10)]
         # the corresponding top 10 indexes get their corresponding names from "sub_categ1"
```

```
In [27]: plt.figure(figsize=(20,10))
         sns.countplot(temp['sub_categ1'])
         plt.title('Sub-Category 1 Analysis',fontsize = 20)
         plt.xticks(rotation = 12,wrap = True,fontsize = 15)
         plt.xlabel('Top 10 Sub-Category1',fontsize = 15)
         plt.ylabel('Frequency',fontsize = 15)

Out[27]: Text(0, 0.5, 'Frequency')
```

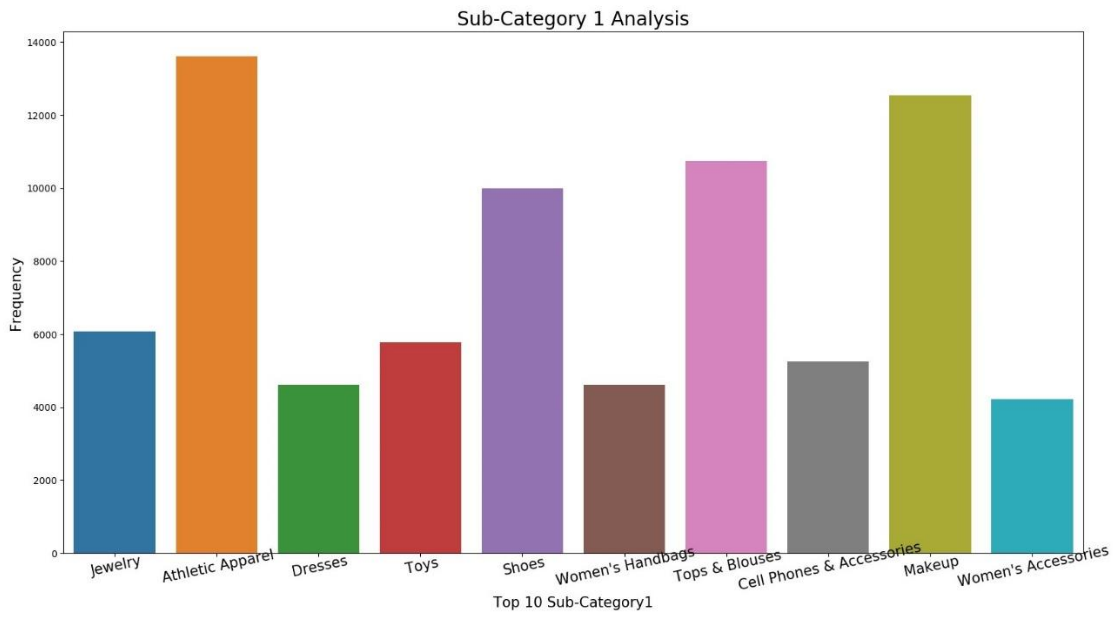


Fig 2.3.2(a): - Top 10 Sub-Category 1 Analysis (Bar Plot)

Insight 1 from Sub-category 1:

- Most of the products from Sub-category 1 are bought from **Athletic Apparel** and **Makeup** category

3.2 (b) Sub Category 1 vs Price

```
In [28]: temp2 = temp[temp['price']<80] # box plot
         plt.figure(figsize=(20,10))
         sns.boxplot(temp2['sub_categ1'],temp2['price'])
         plt.title('Sub-Category 1 vs Price',fontsize = 20)
         plt.xticks(rotation = 10,wrap = True,fontsize = 15)
         plt.xlabel('Top 10 Sub-Category 1',fontsize = 15)
         plt.ylabel('Price',fontsize = 15)
Out[28]: Text(0, 0.5, 'Price')
```

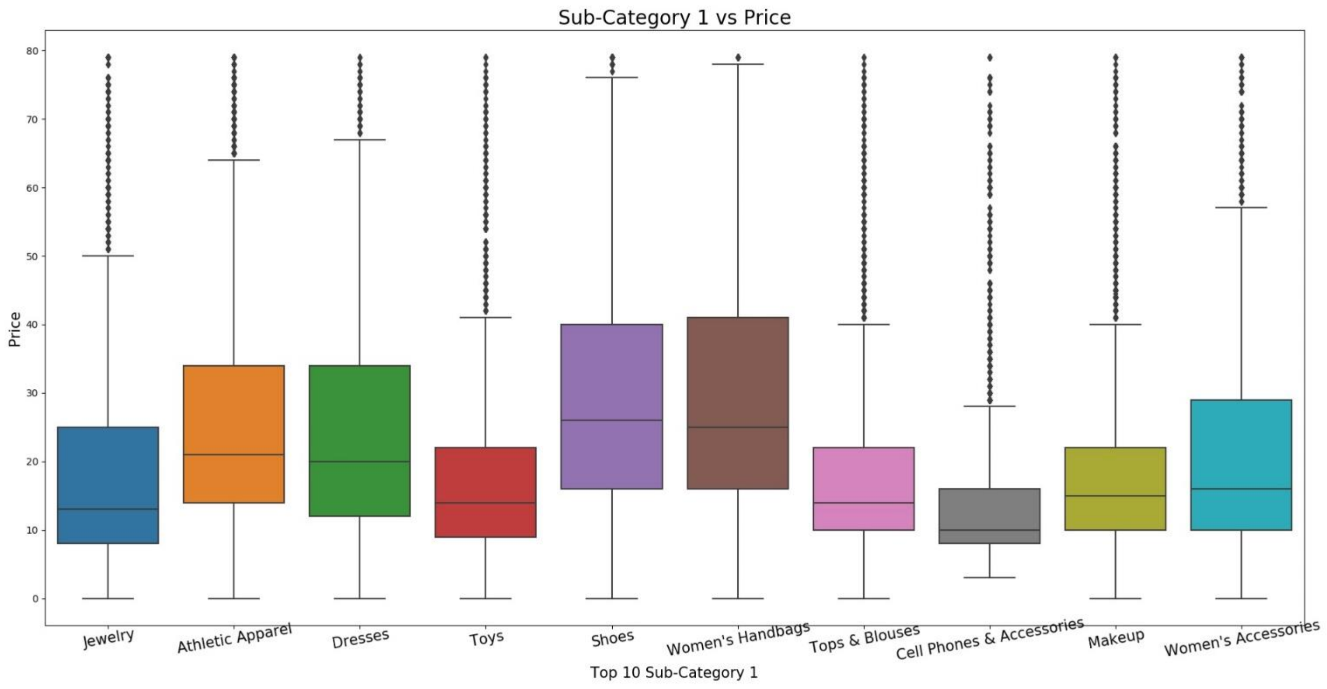


Fig 2.3.2(b): - Top 10 Sub-Category 1 vs Price (Box Plot)

```
In [29]: temp2.groupby(['sub_categ1'])['price'].agg('median')
Out[29]: sub_categ1
         Athletic Apparel             21.0
         Cell Phones & Accessories    10.0
         Dresses                      20.0
         Jewelry                      13.0
         Makeup                       15.0
         Shoes                        26.0
         Tops & Blouses               14.0
         Toys                         14.0
         Women's Accessories          16.0
         Women's Handbags             25.0
         Name: price, dtype: float64
```

Insight 2 from Sub-category 1:

- Products in **Shoes** category are more expensive than **Women's Handbag** by 1 $
- Products in **Women's Handbag** category is more expensive thus has low frequency
- Hence its satisfies the logic that Expensive products are bought less number of times

3.3 (a) Sub Category 2

```
In [30]: index = []
         [index.append(key) for key, value in Counter(data1['sub_categ2']).most_common()]
         top_10 = index[:10]
         temp = data1[data1['sub_categ2'].isin(top_10)]

         plt.figure(figsize=(20,10))
         sns.countplot(temp['sub_categ2'])
         plt.title('Sub-Category 2 Analysis',fontsize = 20)
         plt.xticks(rotation = 7,wrap = True,fontsize = 14)
         plt.xlabel('Top 10 Sub-Category 2',fontsize = 15)
         plt.ylabel('Frequency',fontsize = 15)

Out[30]: Text(0, 0.5, 'Frequency')
```

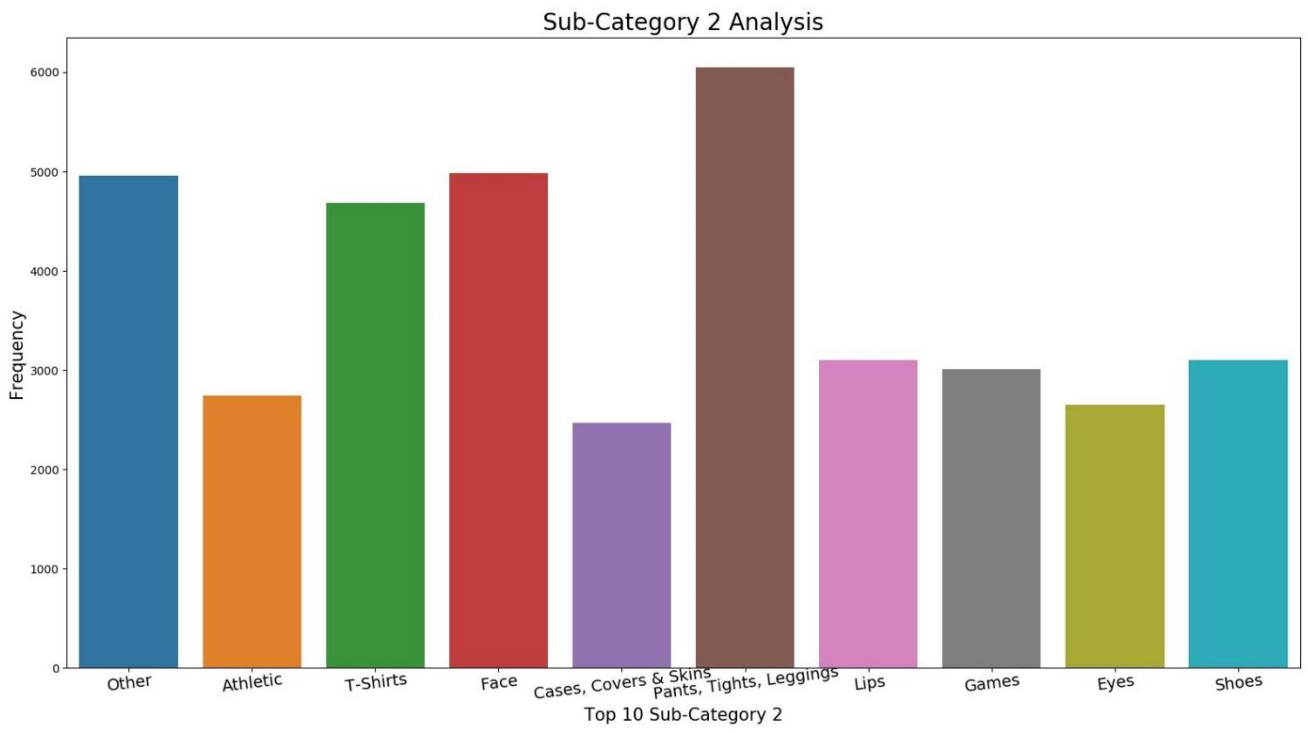


Fig 2.3.2(a): - Top 10 Sub-Category 2 Analysis (Bar Plot)

Insight 1 from Sub-category 2:

- Products under category **Pants, Tights, Leggings** are bought the highest number of times

3.3 (b) Sub Category 2 vs Price

```
In [31]: temp2 = temp[temp['price']<80]
         plt.figure(figsize=(20,10))
         sns.boxplot(temp2['sub_categ2'],temp2['price'])
         plt.title('Sub-Category 2 vs Price', fontsize = 20)
         plt.xticks(rotation = 10,wrap = True,fontsize = 14)
         plt.xlabel('Top 10 Sub-Category 2', fontsize = 15)
         plt.ylabel('Price', fontsize = 15)

Out[31]: Text(0, 0.5, 'Price')
```

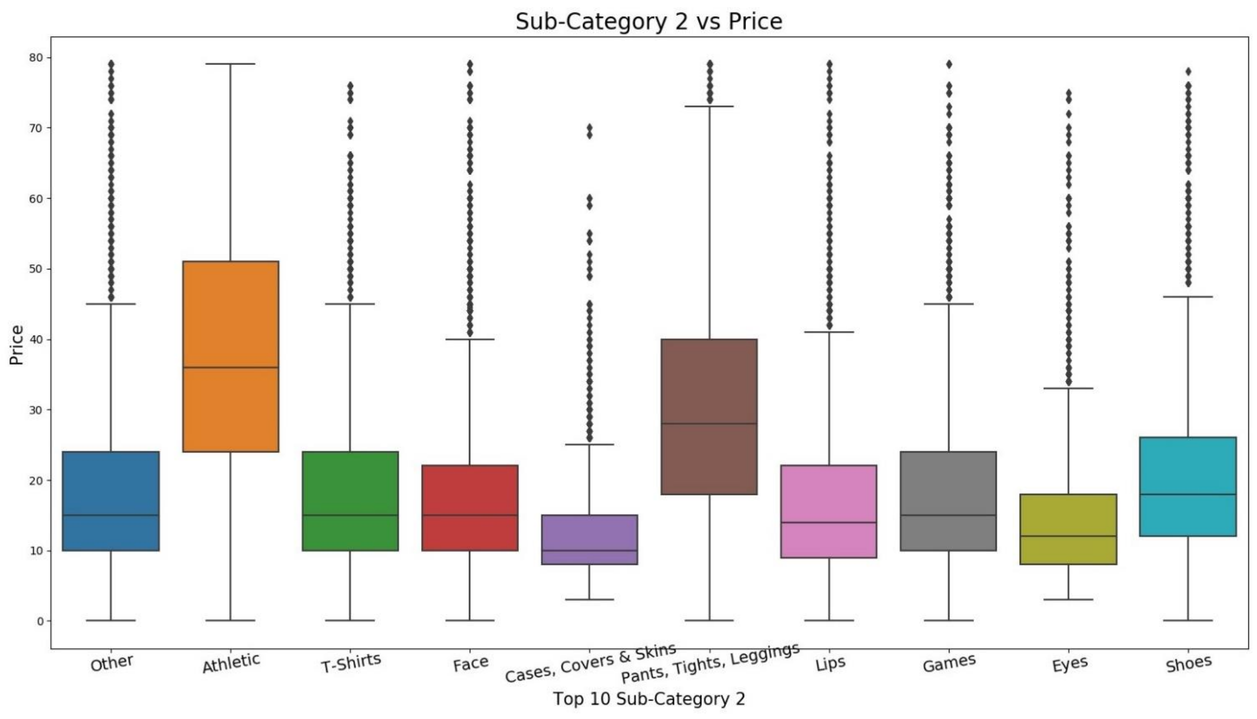


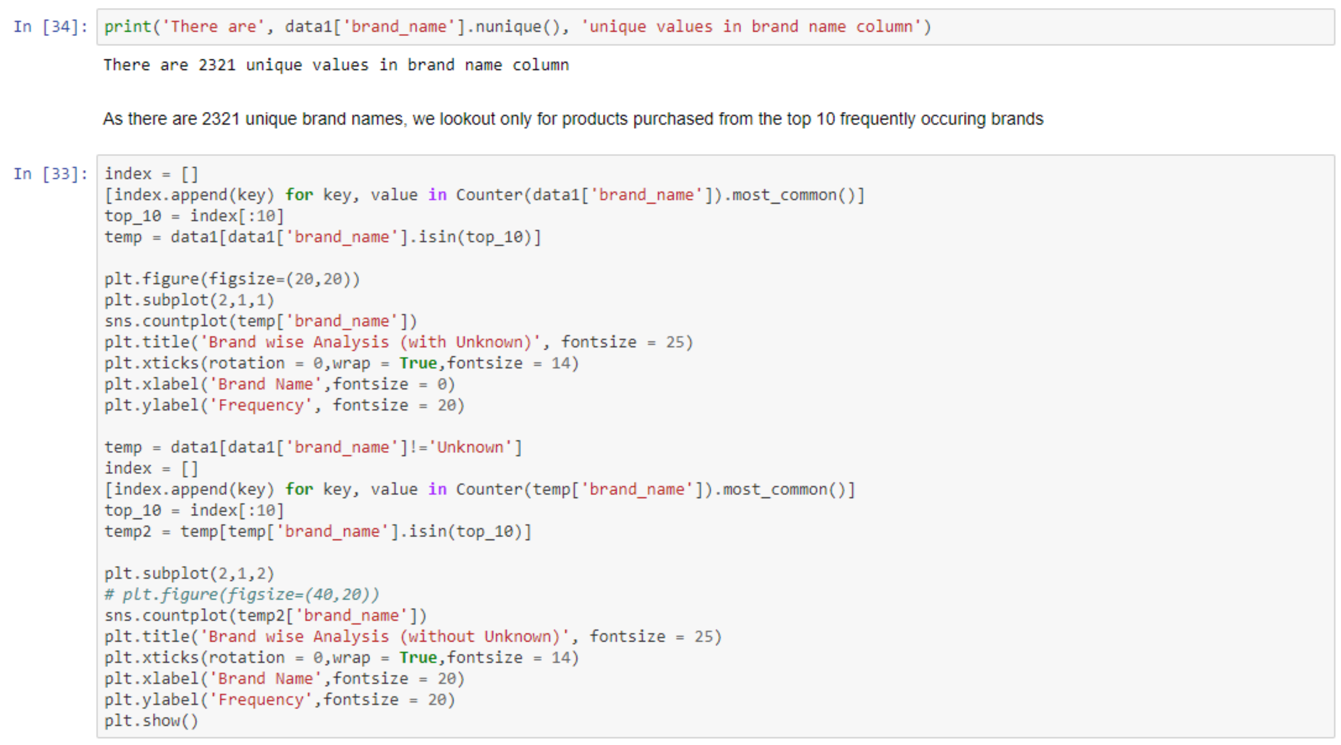
Fig 2.3.2(b): - Top 10 Sub-Category 2 vs Price (Box Plot)

```
In [32]: temp2.groupby(['sub_categ2'])['price'].agg('median')

Out[32]: sub_categ2
         Athletic                   36.0
         Cases, Covers & Skins      10.0
         Eyes                       12.0
         Face                       15.0
         Games                      15.0
         Lips                       14.0
         Other                      15.0
         Pants, Tights, Leggings    28.0
         Shoes                      18.0
         T-Shirts                   15.0
         Name: price, dtype: float64
```

Insight 2 from Sub-Category 2:

- Yet again products in the **Athletic** category are bought moderately as they are expensive
- Products in **Pants**, **Tights**, **Leggings** category are bought more as moderately expensive

4. Brand Name



```
In [34]: print('There are', data1['brand_name'].nunique(), 'unique values in brand name column')

         There are 2321 unique values in brand name column
```

As there are 2321 unique brand names, we lookout only for products purchased from the top 10 frequently occuring brands

```
In [33]: index = []
         [index.append(key) for key, value in Counter(data1['brand_name']).most_common()]
         top_10 = index[:10]
         temp = data1[data1['brand_name'].isin(top_10)]

         plt.figure(figsize=(20,20))
         plt.subplot(2,1,1)
         sns.countplot(temp['brand_name'])
         plt.title('Brand wise Analysis (with Unknown)', fontsize = 25)
         plt.xticks(rotation = 0,wrap = True,fontsize = 14)
         plt.xlabel('Brand Name',fontsize = 0)
         plt.ylabel('Frequency', fontsize = 20)

         temp = data1[data1['brand_name']!='Unknown']
         index = []
         [index.append(key) for key, value in Counter(temp['brand_name']).most_common()]
         top_10 = index[:10]
         temp2 = temp[temp['brand_name'].isin(top_10)]

         plt.subplot(2,1,2)
         # plt.figure(figsize=(40,20))
         sns.countplot(temp2['brand_name'])
         plt.title('Brand wise Analysis (without Unknown)', fontsize = 25)
         plt.xticks(rotation = 0,wrap = True,fontsize = 14)
         plt.xlabel('Brand Name',fontsize = 20)
         plt.ylabel('Frequency',fontsize = 20)
         plt.show()
```

Insight 1 from Brand name:

- As seen in the graph below we ignore the Brand name marked as Unknown as it represents our missing values
- Hence we make the graph again without considering the Unknown
- We observe that Pink, Nike, and Victoria's Secret are top 3 most purchased brands
- Also it is obvious that Pink and Victoria's Secret brand products are mostly bought from Women main category more frequently

4.1 Brand Wise Analysis

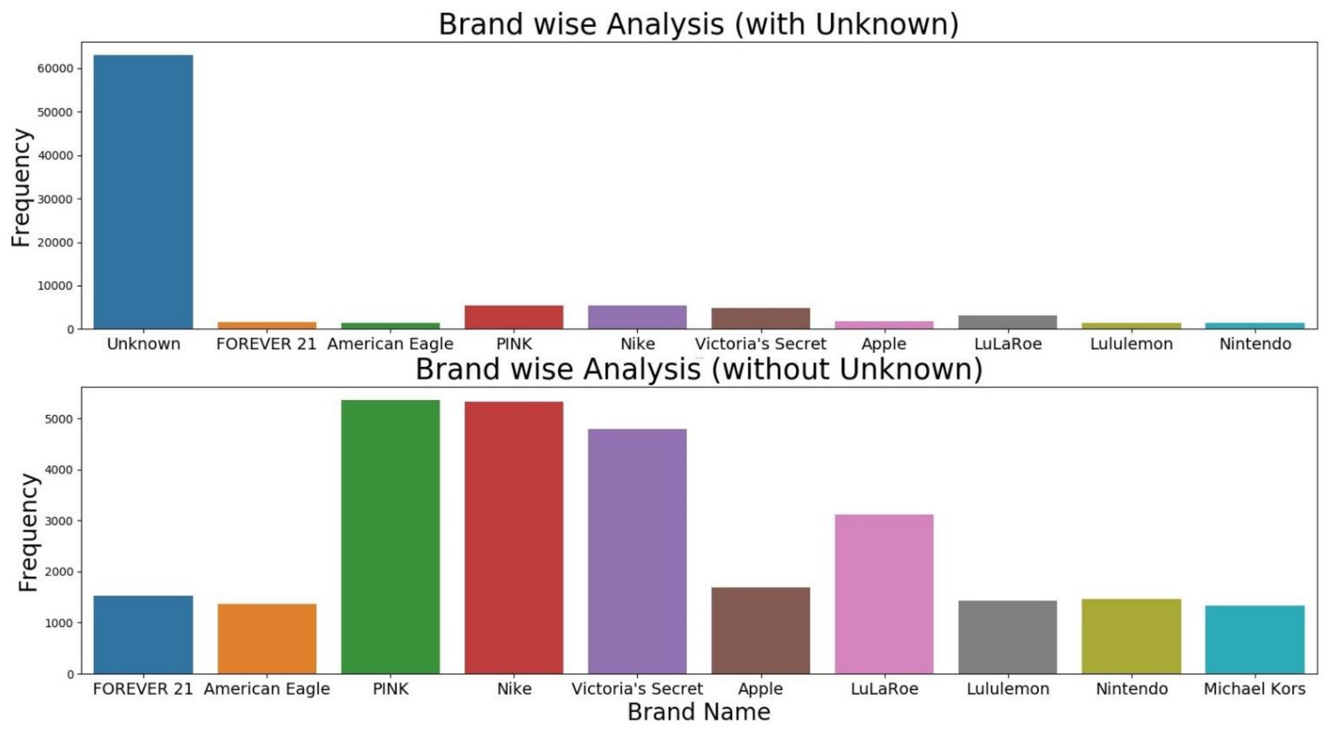


Fig 2.4: -Brand Wise Analysis (Bar Plot)

4.2 Brand Name vs Price

```
In [34]: temp3 = temp2[temp2['price']<80]
         plt.figure(figsize=(20,10))
         sns.boxplot(temp3['brand_name'],temp3['price'])
         plt.title("Top 10 brand wise Analysis (According to price)",fontsize=20)
         plt.xlabel("Brand Name",fontsize=15)
         plt.ylabel("Price",fontsize=15)
         plt.xticks(rotation = 0,wrap = True,fontsize = 15)

Out[34]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), <a list of 10 Text xticklabel objects>)
```

```
In [35]: temp3.groupby(['brand_name'])['price'].agg('median')

Out[35]: brand_name
         American Eagle       14.0
         Apple                15.0
         FOREVER 21           12.0
         LuLaRoe              29.0
         Lululemon            35.0
         Michael Kors         36.0
         Nike                 21.0
         Nintendo             19.0
         PINK                 20.0
         Victoria's Secret    19.0
         Name: price, dtype: float64
```

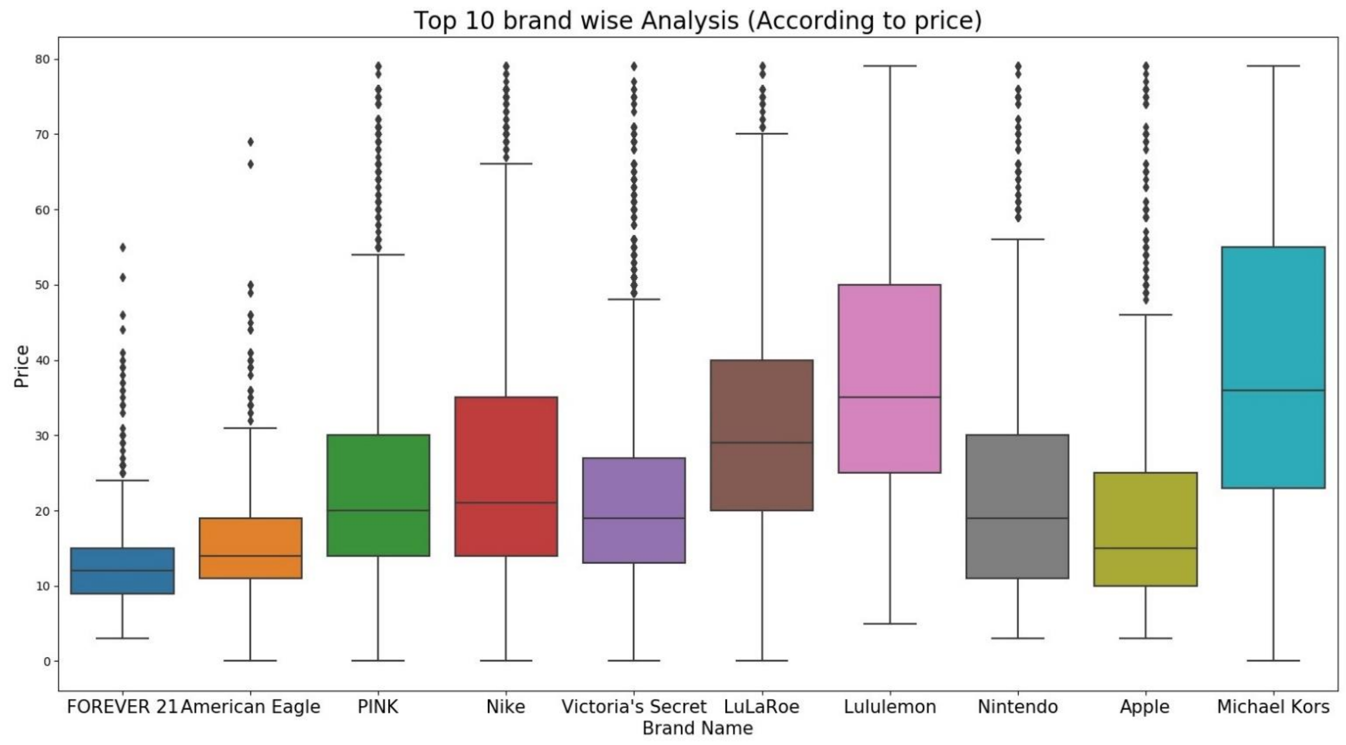


Fig 2.4.1: - Brand Name vs Price (Box Plot)

Insight 2 from Brand name:

- **Michael Kors** products cost high hence less bought
- **Lululemon** has the 2nd highest median price
- **Pink**, **Nike** and **Victoria's** Secret products have moderate pricing hence are more frequently bought

5. Item Description:

First we create a function to return us a count of words from each Description

```
In [36]: def length(description):
             count = 0
             for i in description.split():
                 count+=1
             return count
```

Then we create a series which has the description and the length of description in words

```
In [37]: lol=[]
         for i in data1['item_description']:
             temp=[]
             temp.append(i)
             temp.append(length(str(i)))
             lol.append(temp)
```

Making a new dataframe of the series we created - `lol`

```
In [39]: mydf = pd.DataFrame(lol, columns=['Description', 'Description_length'])
         print(mydf.head(2))

                                   Description  Description_length
0  For sale a brand new Hero 77 fountain pen, doe...                  14
1  14k black Onyx earrings Good condition Final sale                   8
```

Adding only 2nd coloumn `Description_length` from mydf to data1

```
In [40]: data1['Description_length'] = mydf['Description_length']
```

```
In [41]: plt.figure(figsize=(10,5))
         sns.scatterplot(x=data1.Description_length, y=data1.Description_length.value_counts())
         plt.title('Scatter-plot of description length',fontsize=20)
         plt.xlabel("Description Length in words",fontsize=15)
         plt.ylabel("Frequency",fontsize=15)

Out[41]: Text(0, 0.5, 'Frequency')
```

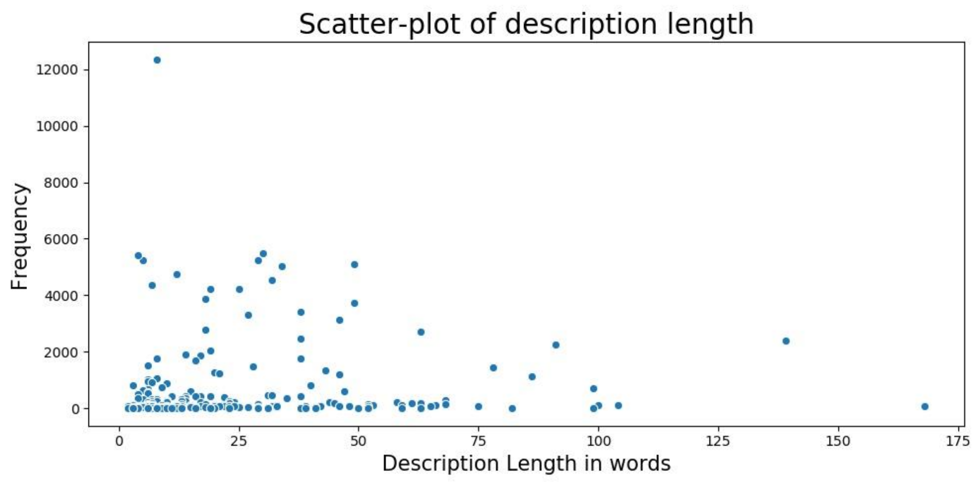


Fig 2.5: - Scatter Plot of description length

Insight 1 from Item Description:

- Most of the item descriptions are written in the range of 3-50 words

5.1 Item Description vs Price:

```
In [42]:  temp4=data1[data1['Description_length']<500]
          plt.figure(figsize=(15,5))
          sns.scatterplot(x=temp4['Description_length'], y=data1.price)
          plt.title('Description length vs Price',fontsize=20)
          plt.xlabel("Description Length in words",fontsize=15)
          plt.ylabel("Price",fontsize=15)

Out[42]:  Text(0, 0.5, 'Price')
```

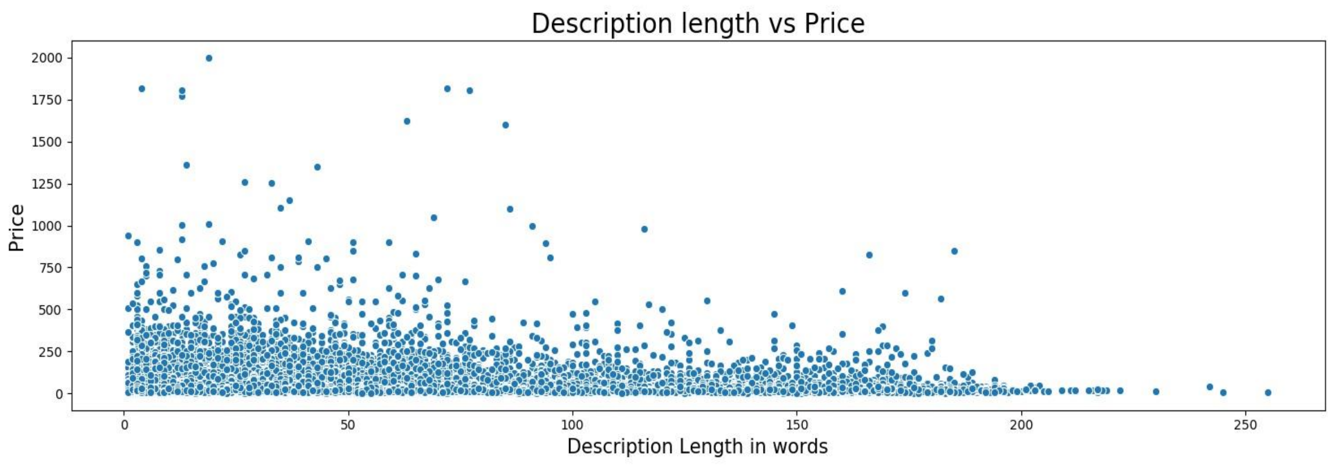


Fig 2.5.1: - Description Length vs Price (Scatter Plot)

Insight 2 from Item Description:

- As the item description gets longer the price starts decreasing

5.2 Let us check what length of item description occurs the most

```
In [43]:
desc_len_count = data1.Description_length.value_counts()
print(desc_len_count[0:3])
```

```
3    12346
6     5503
7     5423
Name: Description_length, dtype: int64
```

```
In [44]:
plt.figure(figsize=(20, 10))
sns.barplot(desc_len_count.index[0:11], desc_len_count[0:11])
plt.title('Item Description having most frequent number of words',fontsize=20)
plt.xticks(rotation = 0,wrap = True,fontsize = 15)
plt.xlabel('Number of words',fontsize=15)
plt.ylabel('Frequency',fontsize=15)
plt.show()
```

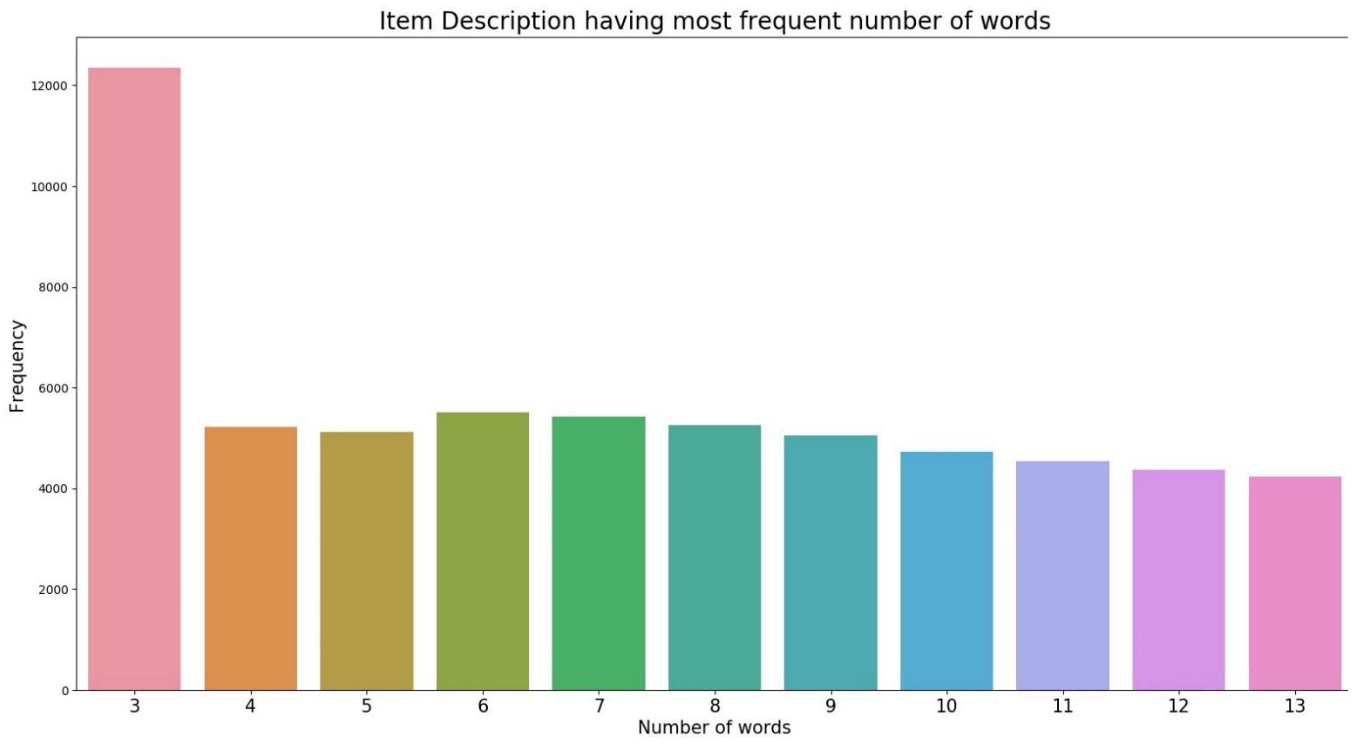


Fig 2.5.2: - Item Description having most frequent number of words

Insight 3 from Item Description:

- This tells us that there are a lot of item descriptions having 3 words in it
- The item descriptions with shorter length are significantly higher than longer description lengths

Text Processing:

Most of the time, the first steps of an NLP project is to "tokenize" your documents, which main purpose is to normalize our texts. The three fundamental stages will usually include:

- break the descriptions into sentences and then break the sentences into tokens
- remove punctuation and stop words
- lowercase the tokens

Word Cloud:

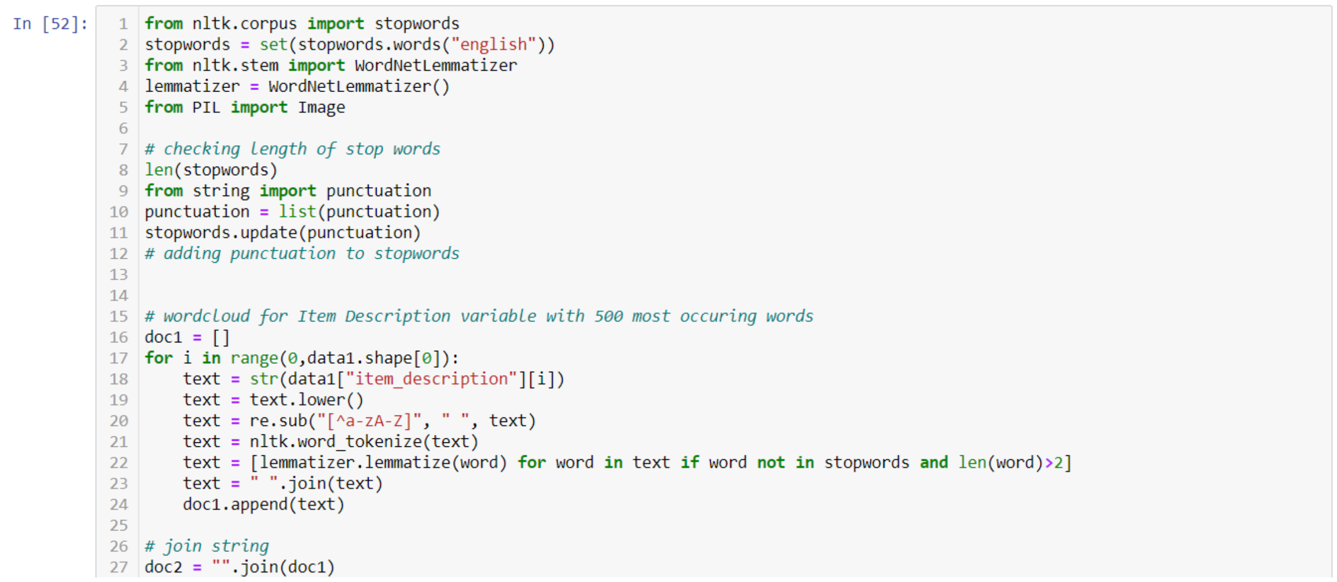
A word cloud is useful to visualize uncoded text responses and questions with too many categories to conveniently show in a bar chart or a table. The area taken by each word/category is proportional to the number of respondents giving that answer. A word cloud will only show the top 100 words/categories

The Word Cloud is mostly created using the 100 most frequent words, excluding uninteresting words, such as I, who and that. You can choose to exclude addition words by dragging them to the **Ignore** region. You can include words that have been excluded by dragging them back into the word cloud

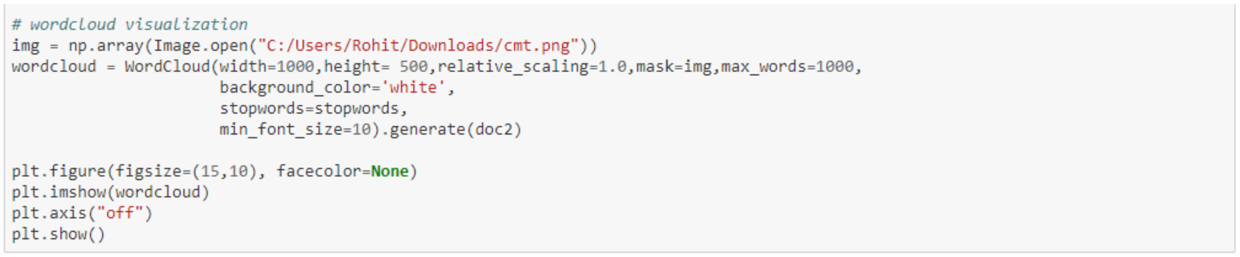
We can create custom Word Cloud by using an image's layout as the shape of our Word Cloud. You can see a plenty of examples in this project

5.3 Word Cloud for Item Description:

To check frequently occuring most **important words** in the description



```
In [52]:
from nltk.corpus import stopwords
stopwords = set(stopwords.words("english"))
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
from PIL import Image

# checking length of stop words
len(stopwords)
from string import punctuation
punctuation = list(punctuation)
stopwords.update(punctuation)
# adding punctuation to stopwords


# wordcloud for Item Description variable with 500 most occuring words
doc1 = []
for i in range(0,data1.shape[0]):
    text = str(data1["item_description"][i])
    text = text.lower()
    text = re.sub("[^a-zA-Z]", " ", text)
    text = nltk.word_tokenize(text)
    text = [lemmatizer.lemmatize(word) for word in text if word not in stopwords and len(word)>2]
    text = " ".join(text)
    doc1.append(text)

# join string
doc2 = "".join(doc1)
```



```
# wordcloud visualization
img = np.array(Image.open("C:/Users/Rohit/Downloads/cmt.png"))
wordcloud = WordCloud(width=1000,height= 500,relative_scaling=1.0,mask=img,max_words=1000,
                      background_color='white',
                      stopwords=stopwords,
                      min_font_size=10).generate(doc2)

plt.figure(figsize=(15,10), facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

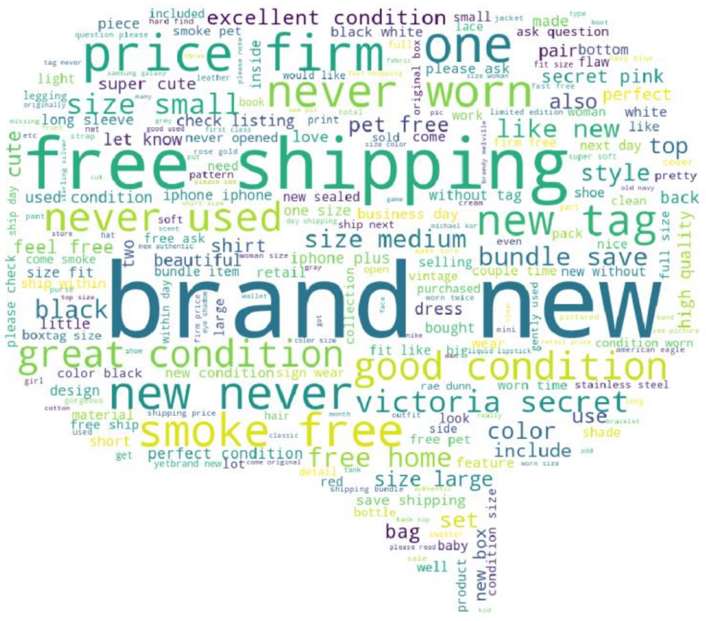
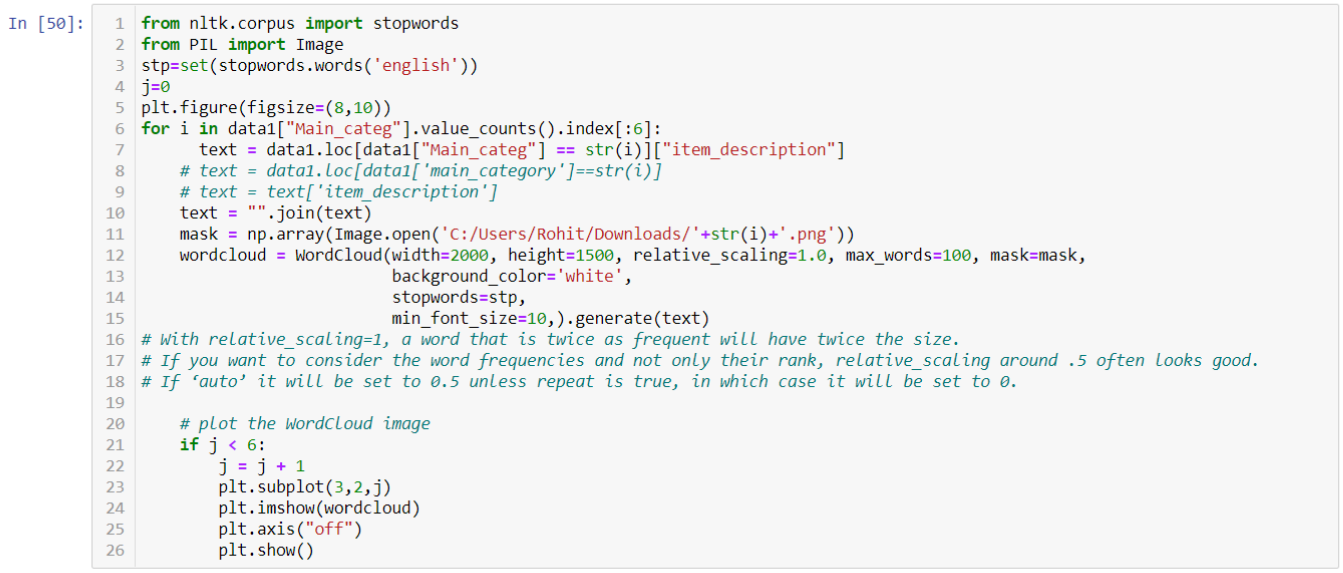


Fig 2.5.3: - Text Processing (Word Cloud) for Item Description

5.4 Word Cloud for Item Description of Top 6 Categories:



```
In [50]:
from nltk.corpus import stopwords
from PIL import Image
stp=set(stopwords.words('english'))
j=0
plt.figure(figsize=(8,10))
for i in data1["Main_categ"].value_counts().index[:6]:
      text = data1.loc[data1["Main_categ"] == str(i)]["item_description"]
    # text = data1.loc[data1['main_category']==str(i)]
    # text = text['item_description']
    text = "".join(text)
    mask = np.array(Image.open('C:/Users/Rohit/Downloads/'+str(i)+'.png'))
    wordcloud = WordCloud(width=2000, height=1500, relative_scaling=1.0, max_words=100, mask=mask,
                          background_color='white',
                          stopwords=stp,
                          min_font_size=10,).generate(text)
# With relative_scaling=1, a word that is twice as frequent will have twice the size.
# If you want to consider the word frequencies and not only their rank, relative_scaling around .5 often looks good.
# If 'auto' it will be set to 0.5 unless repeat is true, in which case it will be set to 0.

    # plot the WordCloud image
    if j < 6:
        j = j + 1
        plt.subplot(3,2,j)
        plt.imshow(wordcloud)
        plt.axis("off")
        plt.show()
```

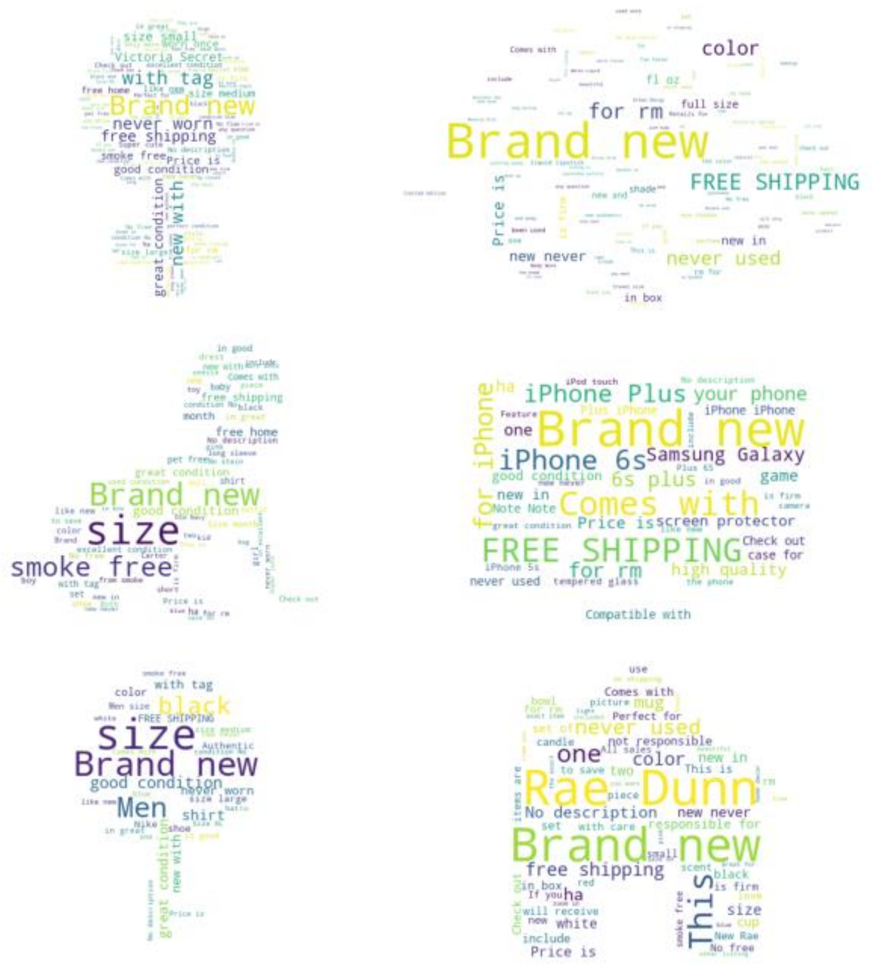


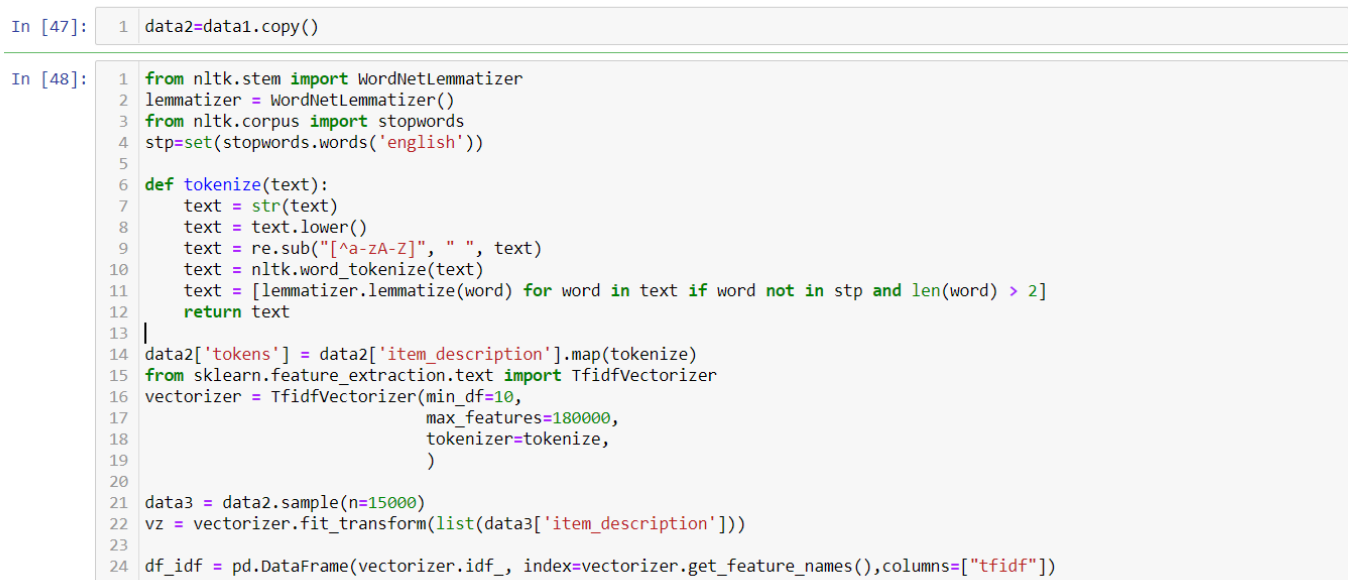
Fig 2.5.4: - Text Processing for Top 6 Category (Word Cloud)

5.5 Interactive Clustering graph with bokeh library:

Step 1- Pre-Processing: tf-idf

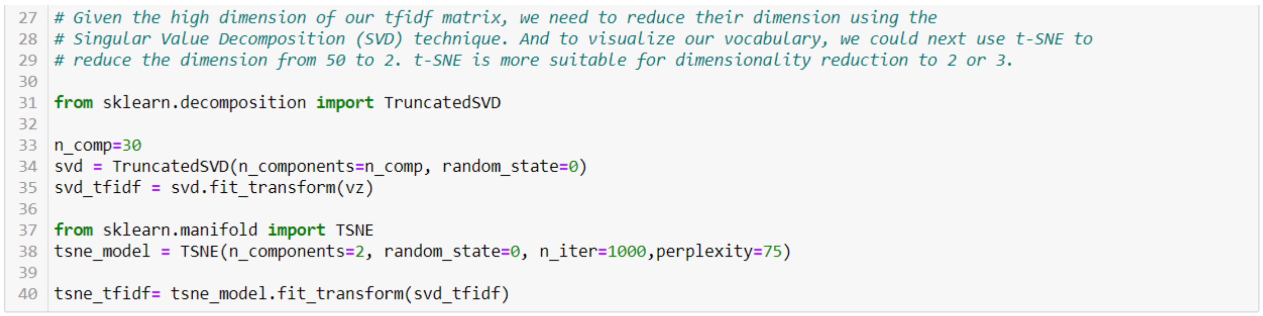
tf-idf is the acronym for Term Frequency–inverse Document Frequency. It quantifies the importance of a particular word in relative to the vocabulary of a collection of documents or corpus. The metric depends on two factors:

Term Frequency: the occurrences of a word in a given document (i.e. bag of words) Inverse Document Frequency: the reciprocal number of times a word occurs in a corpus of documents Think about of it this way: If the word is used extensively in all documents, its existence within a specific document will not be able to provide us much specific information about the document itself. So the second term could be seen as a penalty term that penalizes common words such as "a", "the", "and", etc. tf-idf can therefore, be seen as a weighting scheme for words relevancy in a specific document.



```
In [47]:  data2=data1.copy()
```

```
In [48]:  from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
from nltk.corpus import stopwords
stp=set(stopwords.words('english'))

def tokenize(text):
    text = str(text)
    text = text.lower()
    text = re.sub("[^a-zA-Z]", " ", text)
    text = nltk.word_tokenize(text)
    text = [lemmatizer.lemmatize(word) for word in text if word not in stp and len(word) > 2]
    return text

data2['tokens'] = data2['item_description'].map(tokenize)
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,
                             max_features=180000,
                             tokenizer=tokenize,
                             )

data3 = data2.sample(n=15000)
vz = vectorizer.fit_transform(list(data3['item_description']))

df_idf = pd.DataFrame(vectorizer.idf_, index=vectorizer.get_feature_names(),columns=["tfidf"])
```



```
# Given the high dimension of our tfidf matrix, we need to reduce their dimension using the
# Singular Value Decomposition (SVD) technique. And to visualize our vocabulary, we could next use t-SNE to
# reduce the dimension from 50 to 2. t-SNE is more suitable for dimensionality reduction to 2 or 3.

from sklearn.decomposition import TruncatedSVD

n_comp=30
svd = TruncatedSVD(n_components=n_comp, random_state=0)
svd_tfidf = svd.fit_transform(vz)

from sklearn.manifold import TSNE
tsne_model = TSNE(n_components=2, random_state=0, n_iter=1000,perplexity=75)

tsne_tfidf= tsne_model.fit_transform(svd_tfidf)
```

Step 2- Singular Vector Decomposition:

- Given the high dimension of our tfidf matrix, we need to reduce their dimension using the Singular Value Decomposition (SVD) technique
- And to visualize our vocabulary, we could next use t-SNE to reduce the dimension from 50 to 2
- t-SNE is more suitable for dimensionality reduction to 2 or 3
- In this case 2 coz we need to plot x-y axis

In [53]:
```
from sklearn.decomposition import TruncatedSVD

n_comp=30
svd = TruncatedSVD(n_components=n_comp, random_state=0)
svd_tfidf = svd.fit_transform(vz)
```

Step 3 - t Distributed Stochastic Neighbor Embedding (t-SNE)

- t-SNE is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets
- The goal is to take a set of points in a high-dimensional space and find a representation of those points in a lower-dimensional space, typically the 2D plane
- It is based on probability distributions with random walk on neighborhood graphs to find the structure within the data. But since t-SNE complexity is significantly high, usually we'd use other high-dimension reduction techniques before applying t-SNE

In [55]:
```
from sklearn.manifold import TSNE
tsne_model = TSNE(n_components=2, random_state=0, n_iter=1000,perplexity=75)

tsne_tfidf= tsne_model.fit_transform(svd_tfidf)
```

Step 4- K means Clustering:

- K-means clustering objective is to minimize the average squared Euclidean distance of the document / description from their cluster centroids

In [49]:
```
from sklearn.cluster import KMeans
tfidf_df = pd.DataFrame(tsne_tfidf, columns=['x', 'y'])

temp = tfidf_df.copy()
model_kmeans = KMeans(n_clusters=10,random_state=10).fit(tfidf_df)
temp['description'] = data2['item_description']
temp['tokens'] = data2['tokens']
temp['category'] = data2['Main_categ']
temp['cluster'] = model_kmeans.predict(tfidf_df)
```

```
In [51]: from bokeh.plotting import figure,show
         from bokeh.io import output_file, output_notebook
         from bokeh.transform import linear_cmap
         from bokeh.palettes import Category10
         from bokeh.resources import INLINE
         import bokeh.io
         bokeh.io.output_notebook(INLINE)

         temp.to_csv("C:/Users/Rohit/Downloads/tfidf_tsne.csv")
         sample = pd.read_csv("C:/Users/Rohit/Downloads/tfidf_tsne.csv")
         tooltip = [("Description","@description"),
                      ("Tokens","@tokens"),
                     ("Category","@category"),
                    ("Cluster","@cluster")
                   ]
         mapper = linear_cmap(field_name='cluster', palette=Category10[10],low=0,high=9)
         #output_notebook()
         p= figure(plot_width=900, plot_height=900,
                   title="K-Means clustering of the item description",
                   tools="pan,wheel_zoom,box_zoom,reset,hover",
                   tooltips = tooltip)

         p.scatter("x","y",source = sample, alpha=0.7,color = mapper)
         output_file("output_file_name.html")
         output_notebook()

         # from IPython.display import IFrame
         # IFrame(src='output_file_name.html', width=900, height=900)
         show(p)
```

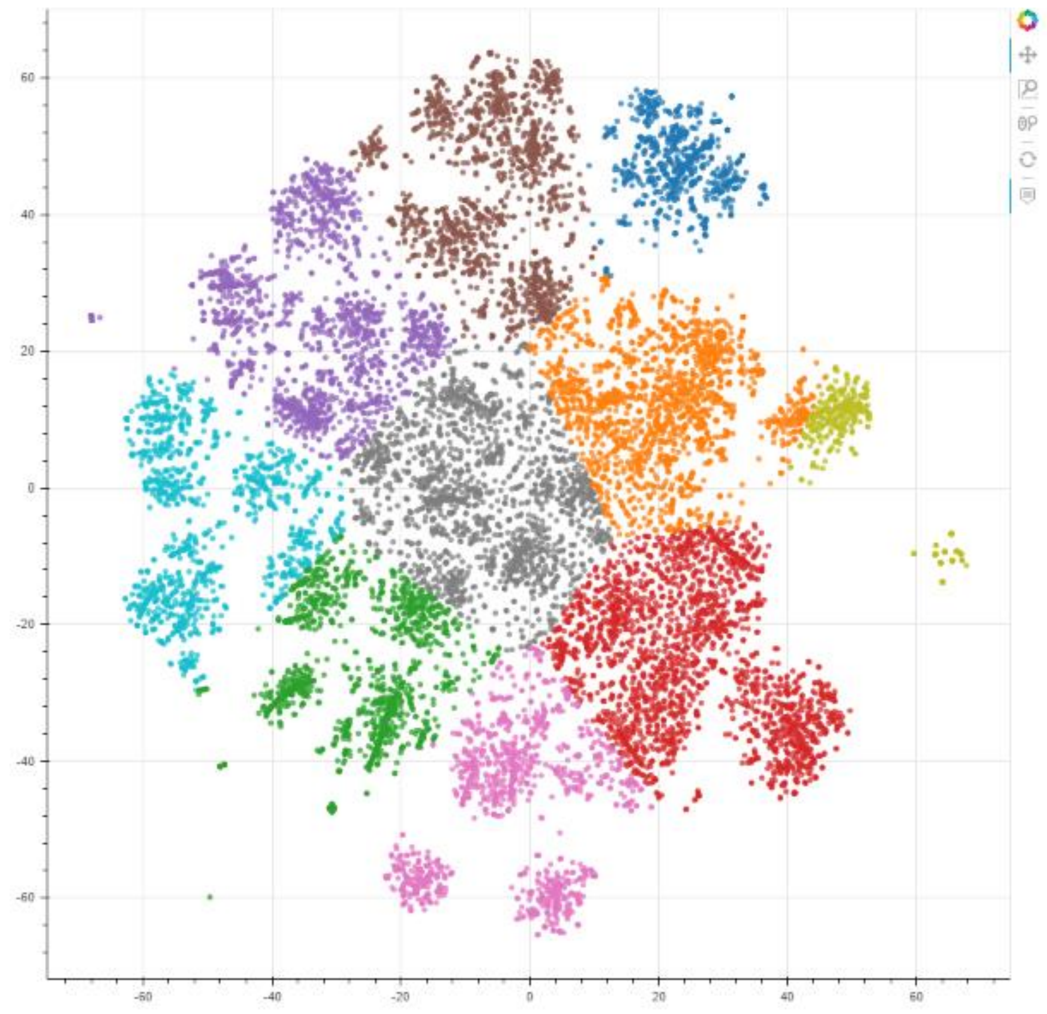


Fig 2.5.5: - K-Means Clustering of Item Description

6. Item Condition

```
In [45]: plt.figure(figsize=(15, 10))
         sns.boxplot(x=data1['item_condition_id'], y=np.log1p(data1.price))
         plt.title('Box Plot of item condition id VS Product price',fontsize=20)
         plt.xlabel('Item Condition ID',fontsize=15)
         plt.ylabel('log(price+1)',fontsize=15)
         plt.show()
```

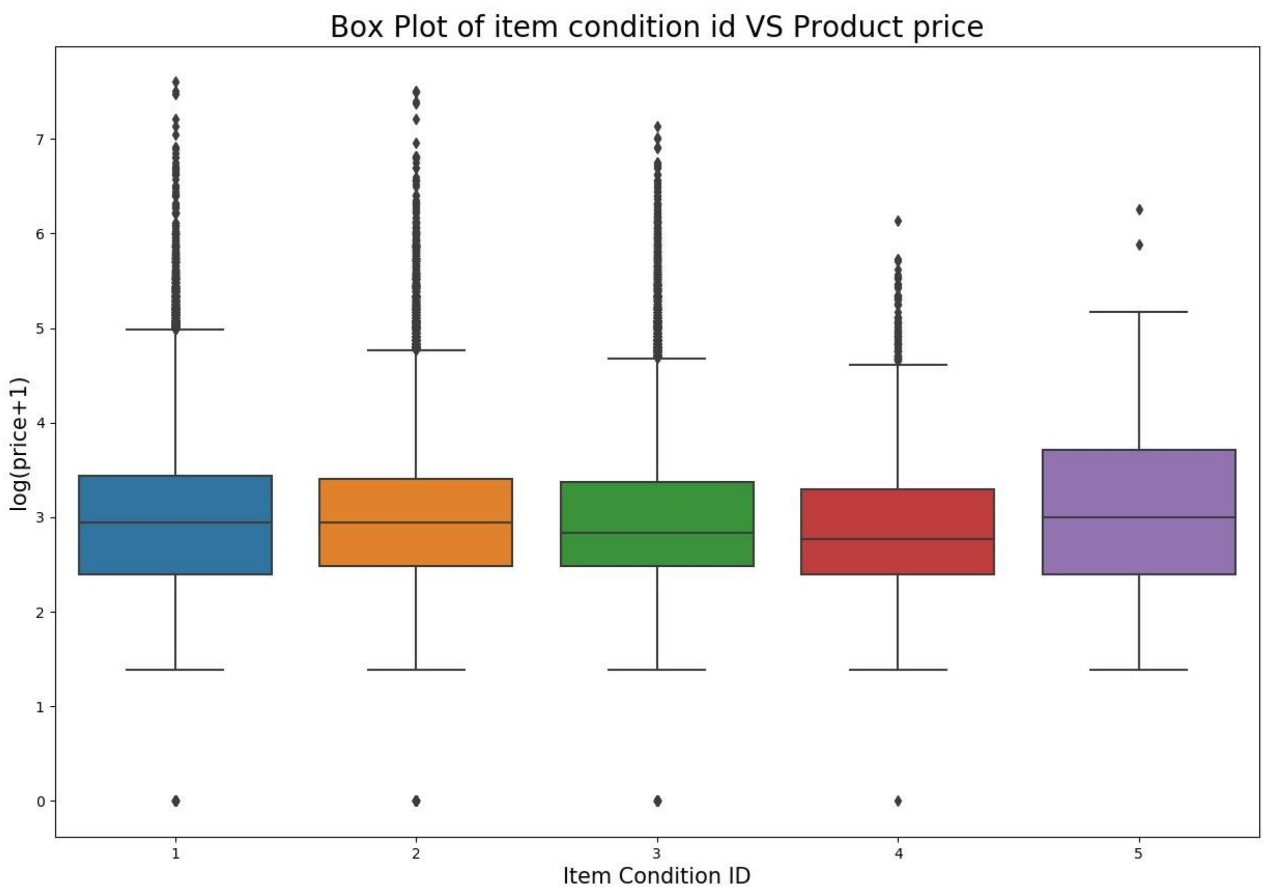


Fig 2.6: - Item Condition vs Price (Box Plot)

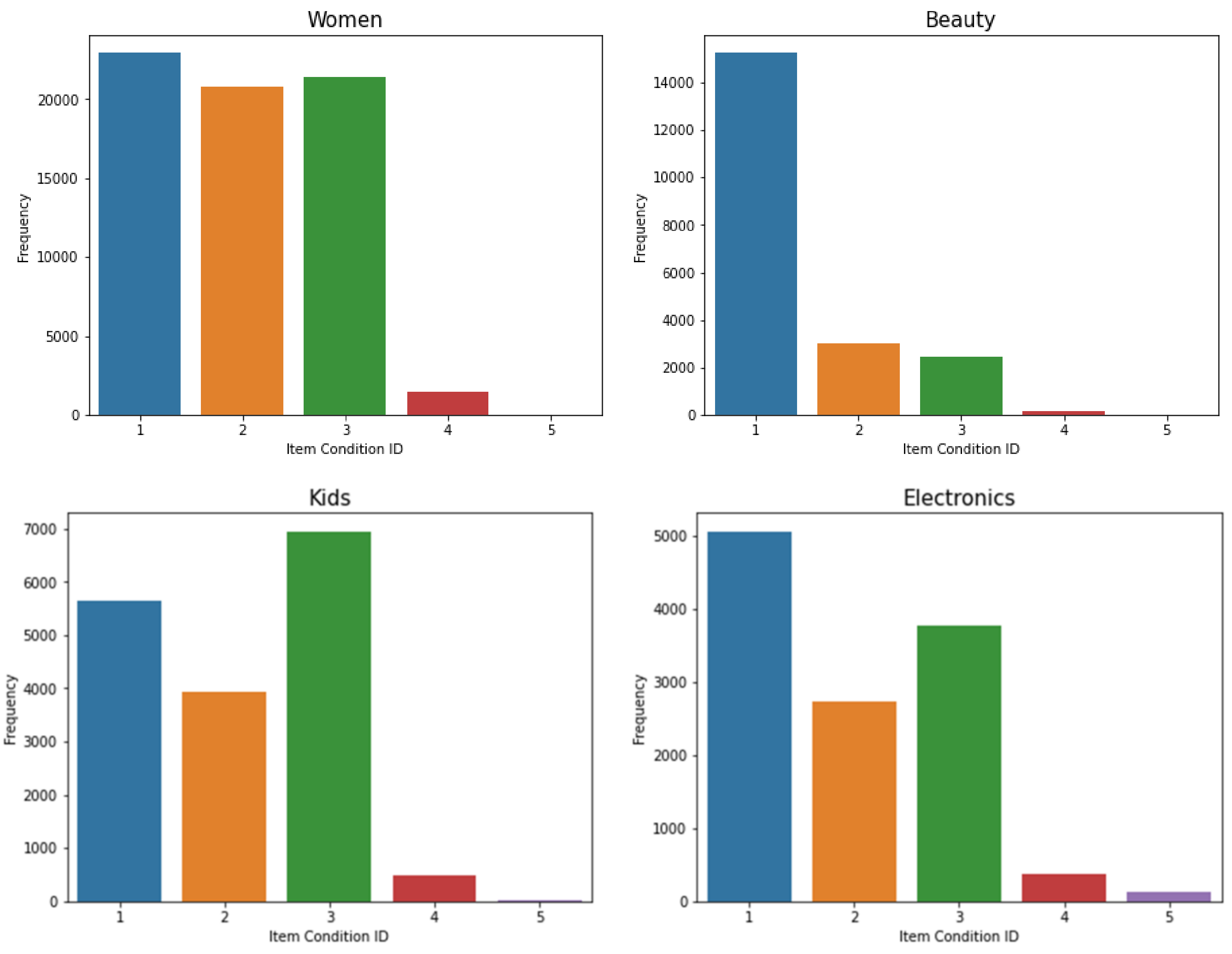
```
In [46]: data1.groupby(['item_condition_id'])['price'].agg('median')

Out[46]: item_condition_id
         1    18.0
         2    18.0
         3    16.0
         4    15.0
         5    19.0
         Name: price, dtype: float64
```

Insight 1 from Item Condition:

- The median of Item condition as 5 is higher than the others which states that items with item condition 5 were more expensive than others

6.1 Item Condition for every Main Category

```
In [47]: from collections import Counter
         index = []
         [index.append(key) for key, value in Counter(data1['Main_categ']).most_common()]


         f,axes = plt.subplots(5, 2, figsize=(15, 30))

         for i in range(data1['Main_categ'].nunique()):
             sns.countplot(data1[data1['Main_categ'] == index[i]]['item_condition_id'], ax = axes[int(i/2)][i%2])
             axes[int(i / 2)][i % 2].set_title(index[i],fontsize=15)
             axes[int(i / 2)][i % 2].set_xlabel('Item Condition ID')
             axes[int(i / 2)][i % 2].set_ylabel('Frequency')
```



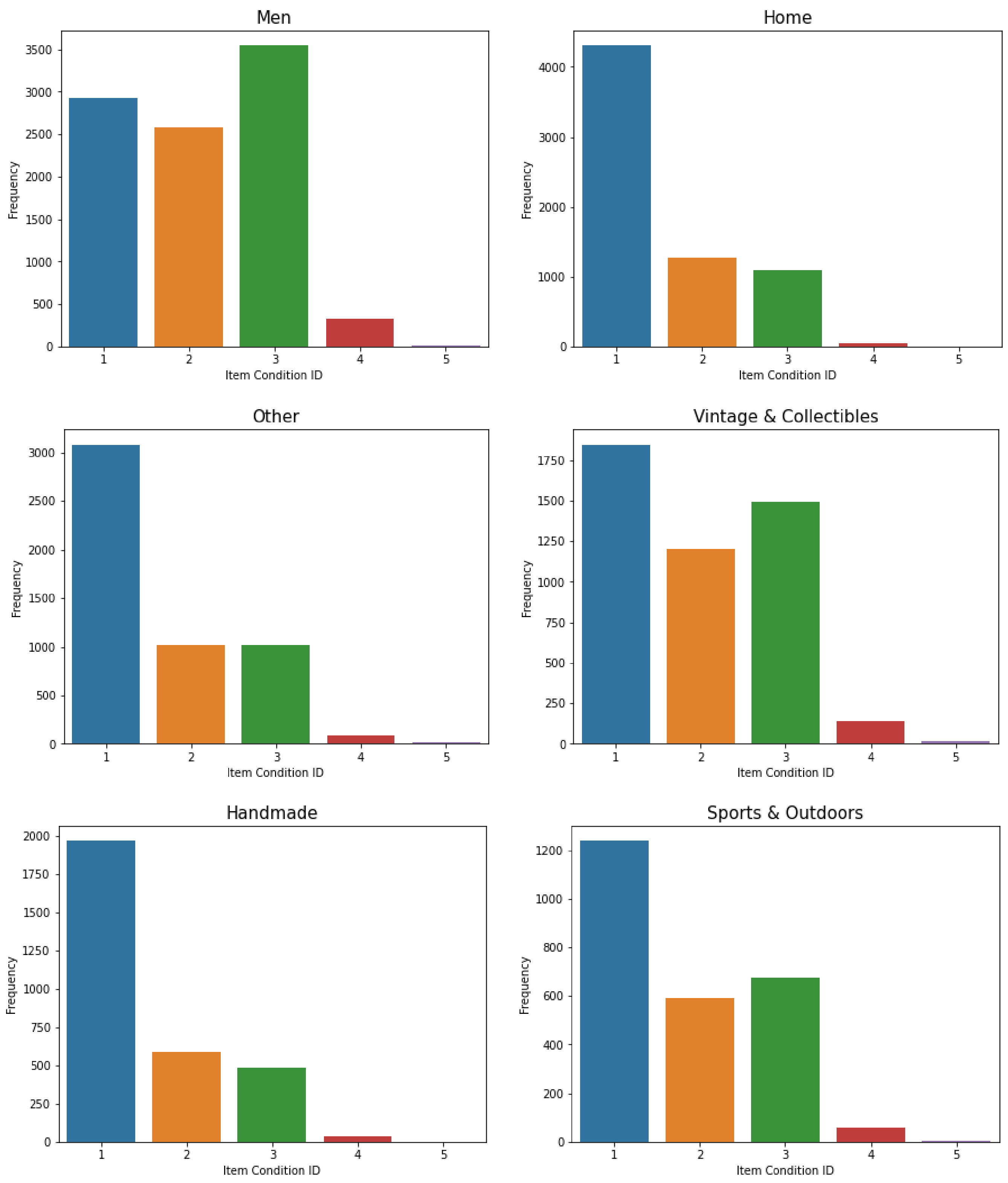


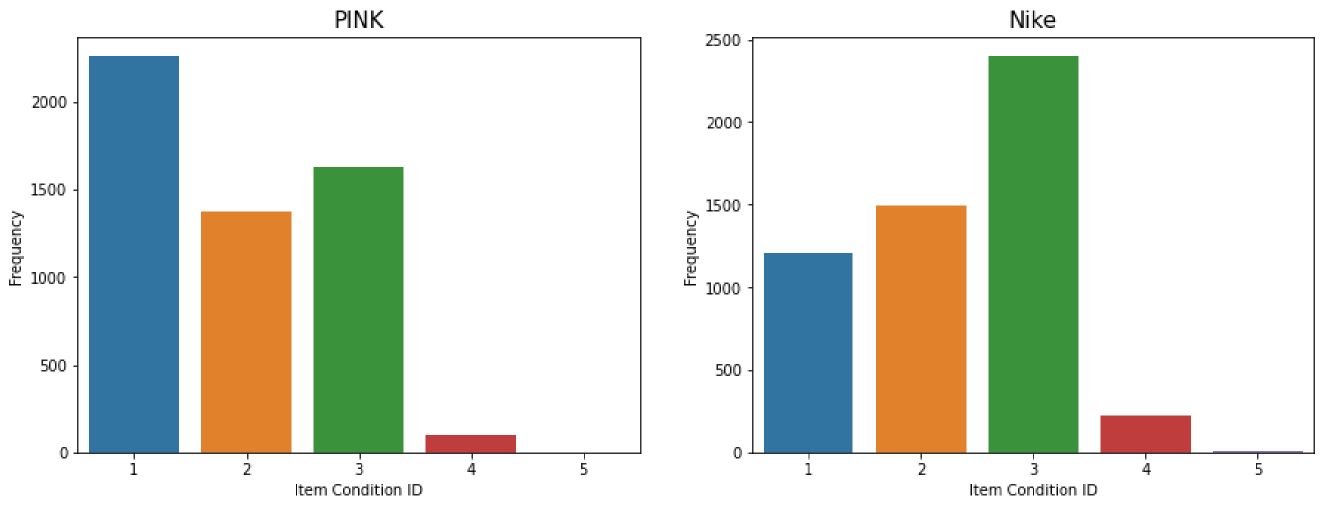
Fig 2.6.1: - Analysis on Item condition of each Main Category (Count Plot)

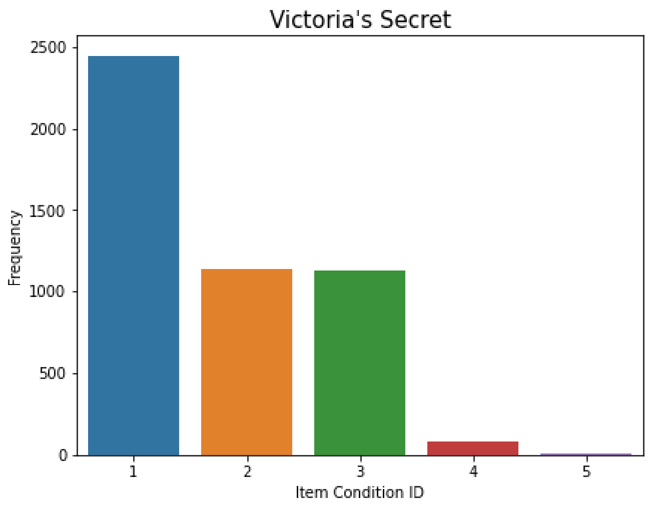
Insight 2 for Item Condition:

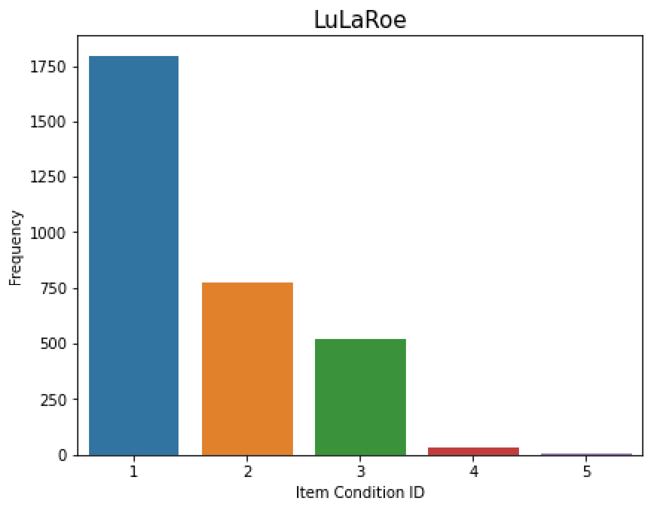
- Item condition marked as 1, 2 and 3 are most frequent in **Women**, **Kids**, **Men** and **Vintage** category
- Item condition marked as 4 and 5 are bought significantly less amount of times
- **Beauty** and **Handmade** categories have too many Item condition id as 1 as compared to other Item Condition ID
- The most number of poor items which marked as 5 are found in the **Electronics** category

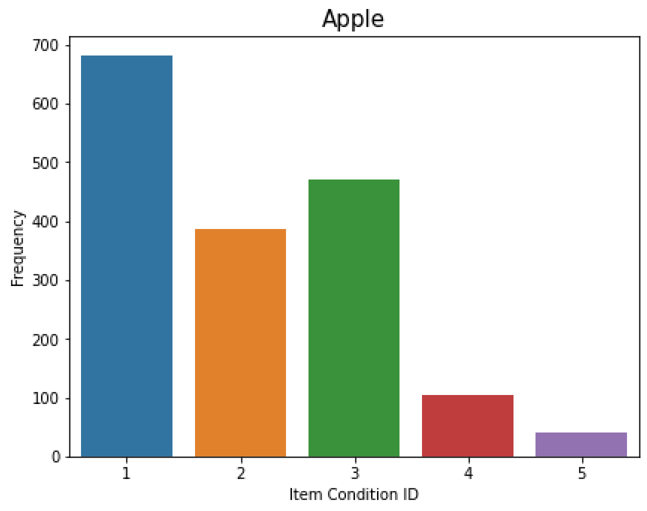
6.2 Item Condition for Top 10 Brands

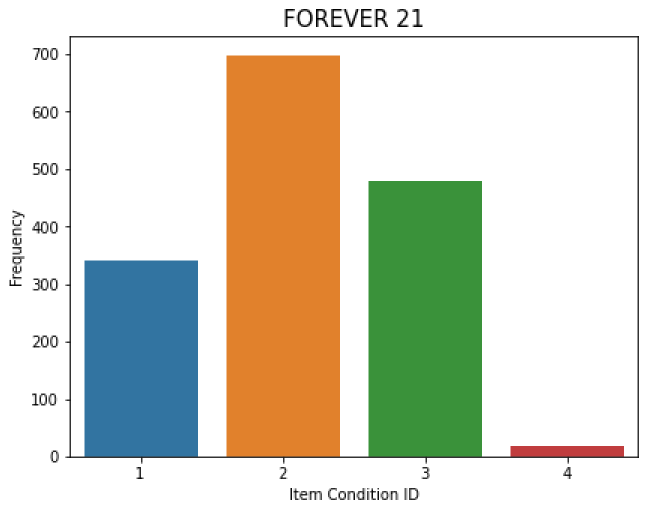
```
In [48]: from collections import Counter
         temp=data1[data1['brand_name']!='Unknown']
         index = []
         [index.append(key) for key, value in Counter(temp['brand_name']).most_common()]
         top_10 = index[:10]
         temp2 = temp[temp['brand_name'].isin(top_10)]

         f,axes = plt.subplots(5, 2, figsize=(15, 30))

         for i in range(temp2['brand_name'].nunique()):
             sns.countplot(temp2[temp2['brand_name'] == index[i]]['item_condition_id'], ax = axes[int(i/2)][i%2])
             axes[int(i / 2)][i % 2].set_title(index[i],fontsize=15)
             axes[int(i / 2)][i % 2].set_xlabel('Item Condition ID')
             axes[int(i / 2)][i % 2].set_ylabel('Frequency')
```

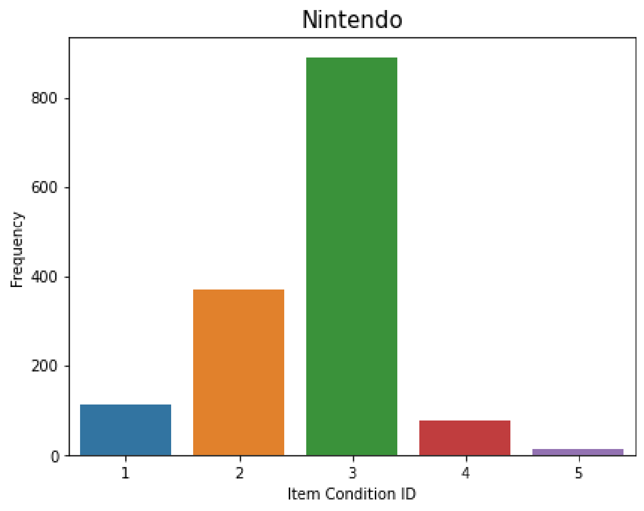


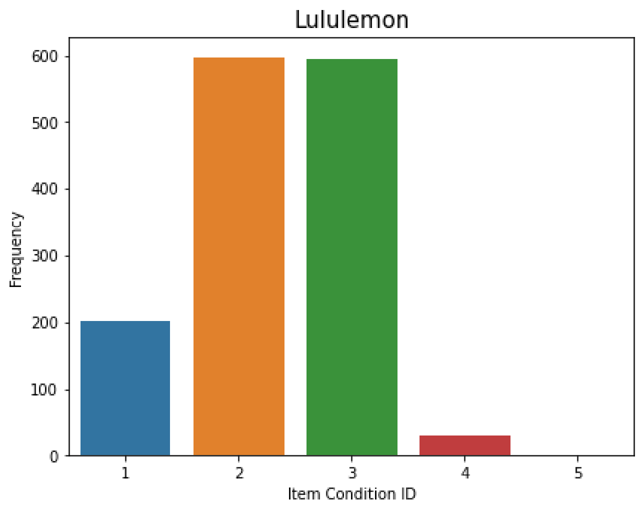

Victoria's Secret
Frequency
0
500
1000
1500
2000
2500
1
2
3
4
5
Item Condition ID


LuLaRoe
Frequency
0
250
500
750
1000
1250
1500
1750
1
2
3
4
5
Item Condition ID


Apple
Frequency
0
100
200
300
400
500
600
700
1
2
3
4
5
Item Condition ID


FOREVER 21
Frequency
0
100
200
300
400
500
600
700
1
2
3
4
Item Condition ID


Nintendo
Frequency
0
200
400
600
800
1
2
3
4
5
Item Condition ID


Lululemon
Frequency
0
100
200
300
400
500
600
1
2
3
4
5
Item Condition ID

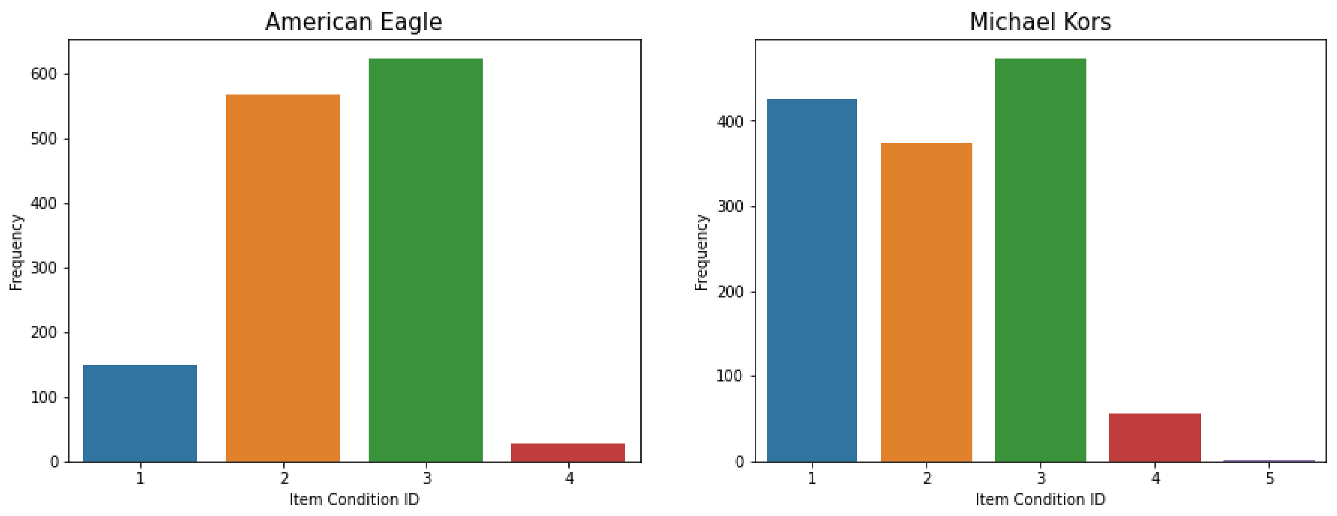


Fig 2.6.2: - Analysis on Item condition of Top 10 Brands (Count Plot)

Insight 3 from Item Condition ID:

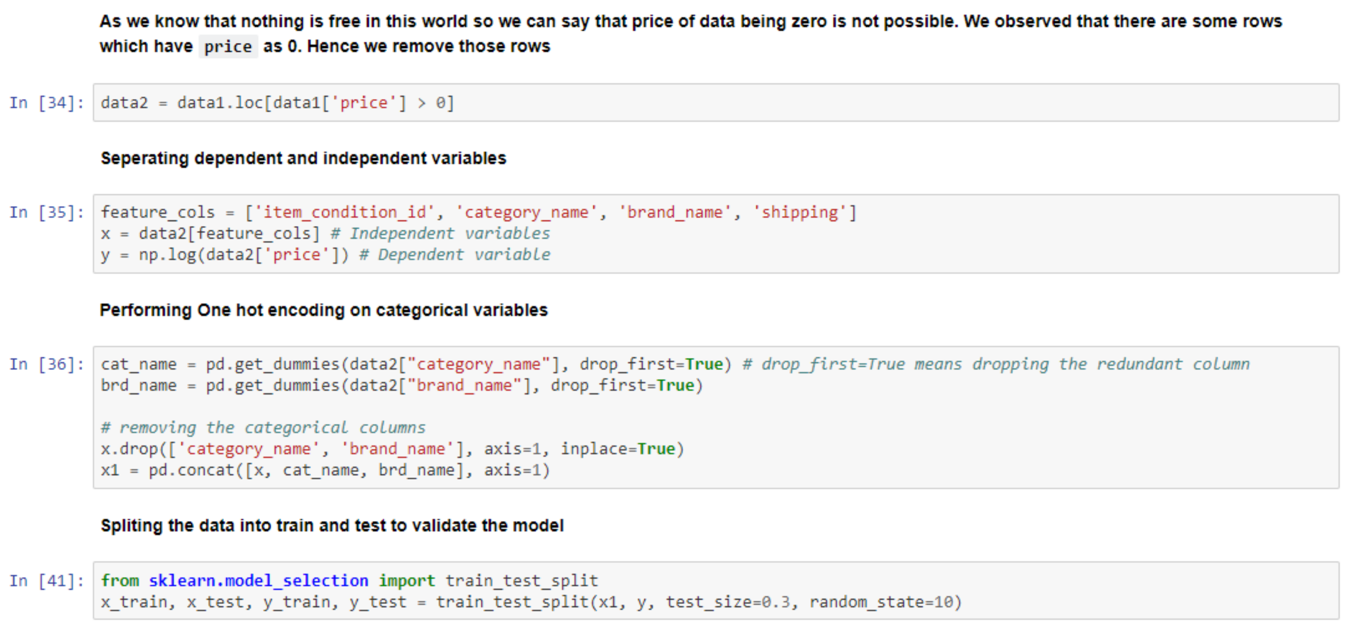
- Brands like Pink, Nike and Michael Kors have more number of New(1) to Good(3) condition items as compared to other brands
- American Eagle doesn't have a single product of poor quality(5)
- Apple has the most number of poor quality(5) products

CHAPTER 3

MACHINE LEARNING

Applying Machine Learning Models:

Coming to our actual problem, we will be now applying machine learning models to give us a price that we should suggest to the seller. As discussed earlier, this is a regression problem, so we will be applying some regressions models based on our data and other factors that we will face during solving.
In python we have to do a bit of preprocessing on categorical data while performing regression algorithms. Methods like One hot encoding and Label Encoding are used for the same.
For validation purposes, there is a need to split the data into training and testing and also to avoid problems like overfitting and bias/variance tradeoff.


As we know that nothing is free in this world so we can say that price of data being zero is not possible. We observed that there are some rows which have `price` as 0. Hence we remove those rows

```
In [34]: data2 = data1.loc[data1['price'] > 0]
```

Seperating dependent and independent variables

```
In [35]: feature_cols = ['item_condition_id', 'category_name', 'brand_name', 'shipping']
         x = data2[feature_cols] # Independent variables
         y = np.log(data2['price']) # Dependent variable
```

Performing One hot encoding on categorical variables

```
In [36]: cat_name = pd.get_dummies(data2["category_name"], drop_first=True) # drop_first=True means dropping the redundant column
         brd_name = pd.get_dummies(data2["brand_name"], drop_first=True)

         # removing the categorical columns
         x.drop(['category_name', 'brand_name'], axis=1, inplace=True)
         x1 = pd.concat([x, cat_name, brd_name], axis=1)
```

Spliting the data into train and test to validate the model

```
In [41]: from sklearn.model_selection import train_test_split
         x_train, x_test, y_train, y_test = train_test_split(x1, y, test_size=0.3, random_state=10)
```

1. OLS (Ordinary Least Squares Regression) Model:

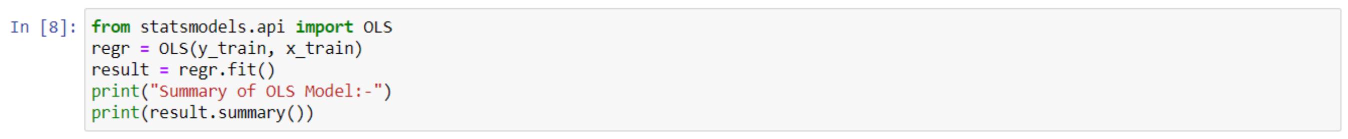
Regression analysis predicts the value of a dependent variable based on the known value of the independent variable. Also defined as a relationship between one outcome and several explanatory variables in terms of an equation. It tells about the magtd. Of the relation or numerical relation.

It is a statistical method that allows to summarize and study relationships between 2 continuous variables. One is predictor or independent or explanatory (x) and other is dependent or outcome or response (y)

The main ideas behind linear regression are:

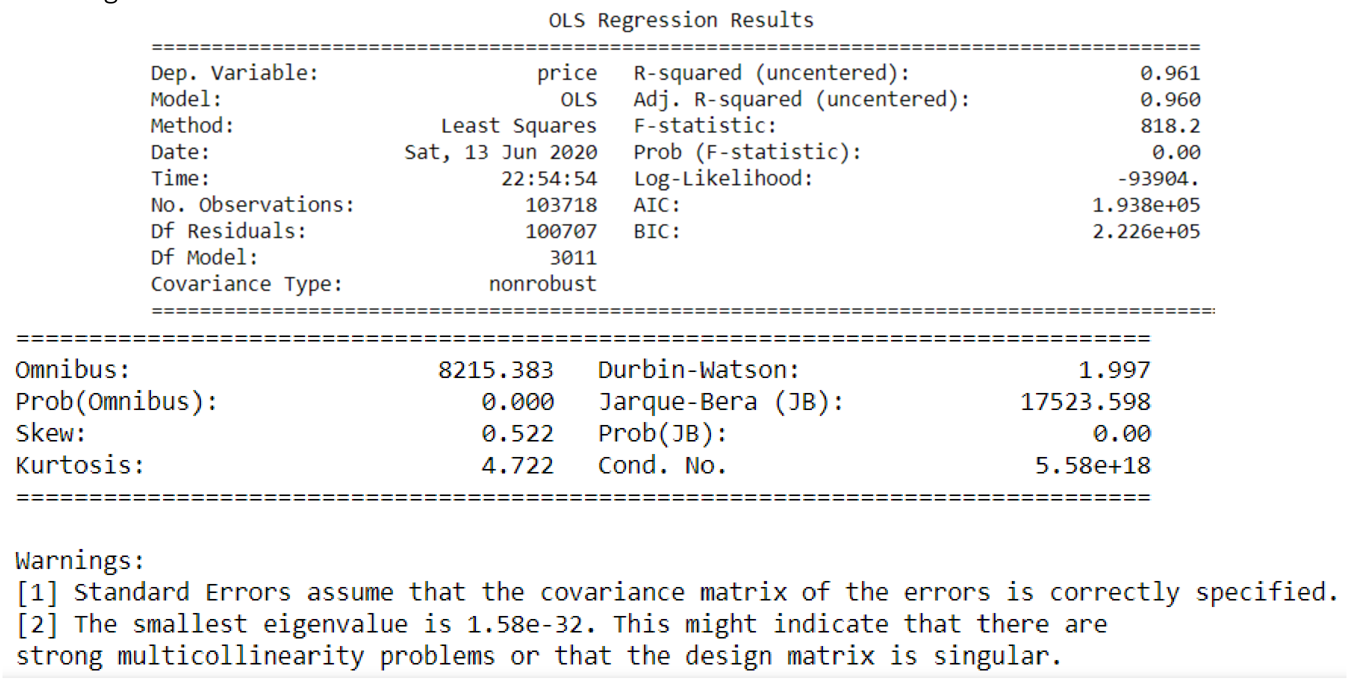
1. Use a least squares line to fit a line to the data
2. Calculate R^2
3. Calculate a P-value for R^2

Also there is another important measure called RMSE which signifies how closely our predicted values are to the actual data
Formula - $\sqrt{}$ (Actual value – Predicted value) ^{2}



```
In [8]: from statsmodels.api import OLS
        regr = OLS(y_train, x_train)
        result = regr.fit()
        print("Summary of OLS Model:-")
        print(result.summary())
```

Checking the results



```
                              OLS Regression Results
=======================================================================================
Dep. Variable:                  price   R-squared (uncentered):                   0.961
Model:                            OLS   Adj. R-squared (uncentered):              0.960
Method:                 Least Squares   F-statistic:                              818.2
Date:                Sat, 13 Jun 2020   Prob (F-statistic):                        0.00
Time:                        22:54:54   Log-Likelihood:                         -93904.
No. Observations:              103718   AIC:                                  1.938e+05
Df Residuals:                  100707   BIC:                                  2.226e+05
Df Model:                        3011
Covariance Type:            nonrobust
=======================================================================================
==============================================================================
Omnibus:                     8215.383   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            17523.598
Skew:                           0.522   Prob(JB):                         0.00
Kurtosis:                       4.722   Cond. No.                     5.58e+18
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.58e-32. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

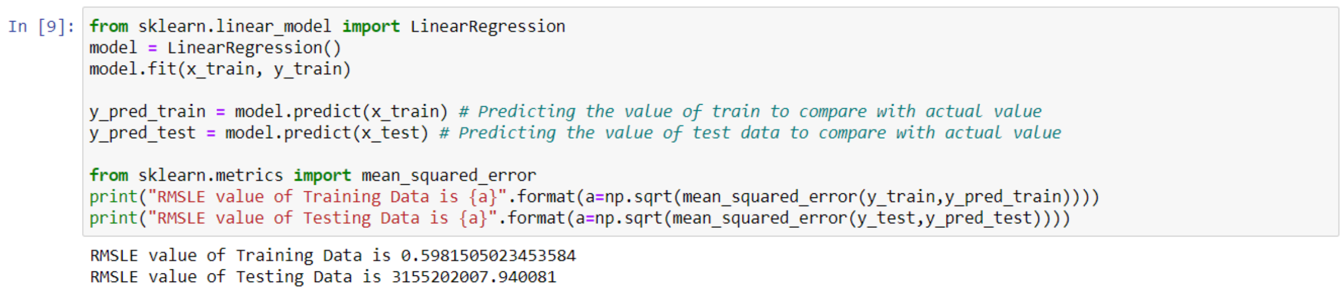
Important Note:

1. Not always **R-squared** is the best parameter to judge our model as sometimes no matter what we do, we can never get a proper R-squared value because the way in which the data is built. Hence we look after other deciding parameter called RMSLE
2. We are calling **RMSLE** (Root Mean Squared log Error) instead of RMSE because we have taken the log of our dependent variable as it wasn't normally distributed

Insights from OLS model:

- As we can see that the R-squared is almost close to 1 which means the predicted value of the training data is very similar to that of the test data
- The 2nd warning tells us that there are strong multi-collinearity problems or that the design matrix is singular because of the one hot encoding and many unique values in our data columns
- Hence this is one of the major drawbacks of this model in our case
- Other factors like Skewness (0.5), Kurtosis (4.7) i.e. almost close to Mesokurtic curve (3) and Auto-correlation (1.99) tell us that model is optimal, however we still can't use this model to know the RMSLE as well as to predict the values
- For predicting the values for testing data we use linear regression and see if we can overcome the OLS model's drawbacks in **statsmodel** library

2. Linear Regression Model:



```
In [9]: from sklearn.linear_model import LinearRegression
        model = LinearRegression()
        model.fit(x_train, y_train)

        y_pred_train = model.predict(x_train) # Predicting the value of train to compare with actual value
        y_pred_test = model.predict(x_test) # Predicting the value of test data to compare with actual value

        from sklearn.metrics import mean_squared_error
        print("RMSLE value of Training Data is {a}".format(a=np.sqrt(mean_squared_error(y_train,y_pred_train))))
        print("RMSLE value of Testing Data is {a}".format(a=np.sqrt(mean_squared_error(y_test,y_pred_test))))

        RMSLE value of Training Data is 0.5981505023453584
        RMSLE value of Testing Data is 3155202007.940081
```

Insights from Linear Model:

- We can observe that training data has very less rmsle but the testing data the rmsle is extremely high.

Problem:

- The data is containing multi-collinearity which can be seen in the RMSLE of testing data

Solution:

- Almost 99% of our columns are the dummies representing the categorical data and we cannot remove them and it would be futile to check the VIF of over 3500 columns
- Hence, we have to use algorithms which we can fine tune to optimize our model
- This leads us to a conclusion that there is a need to build models which can help us apply techniques like Dimension reduction, Regularization, Boosting or Bagging. Let's check them out one by one
- In ML, the ideal model is the one which has low bias i.e it can capture true relationship and low variability by producing consistent predictions across different datasets
- The methods used to do this are –

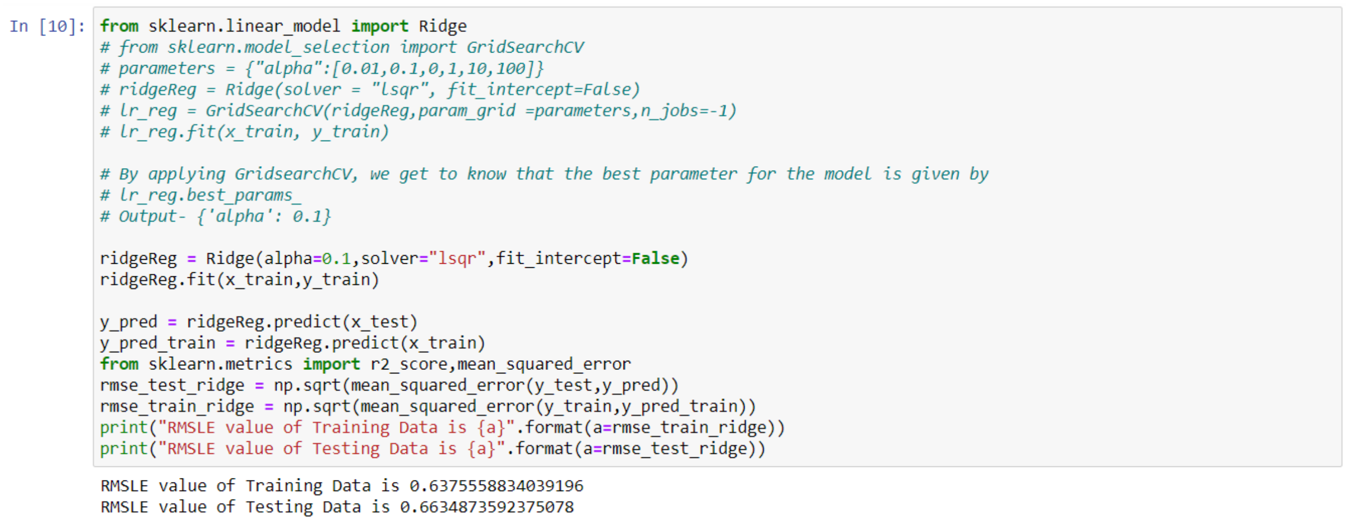
  • Regularization • Boosting • Bagging

3. Ridge Regression:

Overview:

- Ridge Regression comes under the Regularization method
- It provides a way to create a simple model with great explanatory power when number of predictor variables in a dataset are very high and not very significant or when a dataset has multi-collinearity
- It uses a type of Shrinkage Parameter called ridge estimator which belongs to a class of L2 regularization (λ)
- L2 regularization adds a penalty which equals the square of the magnitude of co-efficient
- All co-effs are shrunk by the same factor
- When λ = 0, ridge regression equals least squares regression and when λ = infinity, all co-effs are shrunk to 0
- Our goal here is to find the perfect λ value because as λ increases, error due to bias increases and as λ decreases, error due to variance increases

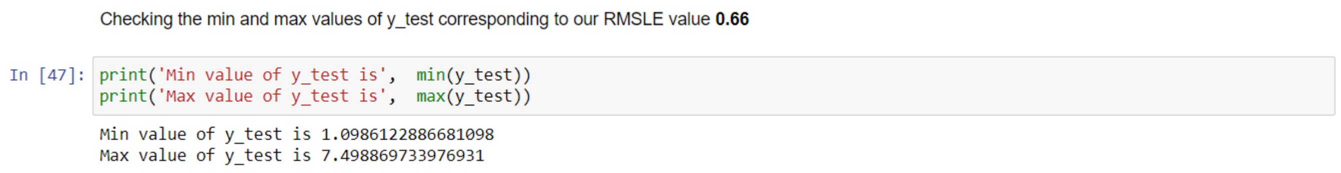
Note: - In python, while modelling the **λ** is considered as **alpha**



```
In [10]: from sklearn.linear_model import Ridge
         # from sklearn.model_selection import GridSearchCV
         # parameters = {"alpha":[0.01,0.1,0,1,10,100]}
         # ridgeReg = Ridge(solver = "lsqr", fit_intercept=False)
         # lr_reg = GridSearchCV(ridgeReg,param_grid =parameters,n_jobs=-1)
         # lr_reg.fit(x_train, y_train)

         # By applying GridsearchCV, we get to know that the best parameter for the model is given by
         # lr_reg.best_params_
         # Output- {'alpha': 0.1}

         ridgeReg = Ridge(alpha=0.1,solver="lsqr",fit_intercept=False)
         ridgeReg.fit(x_train,y_train)

         y_pred = ridgeReg.predict(x_test)
         y_pred_train = ridgeReg.predict(x_train)
         from sklearn.metrics import r2_score,mean_squared_error
         rmse_test_ridge = np.sqrt(mean_squared_error(y_test,y_pred))
         rmse_train_ridge = np.sqrt(mean_squared_error(y_train,y_pred_train))
         print("RMSLE value of Training Data is {a}".format(a=rmse_train_ridge))
         print("RMSLE value of Testing Data is {a}".format(a=rmse_test_ridge))

         RMSLE value of Training Data is 0.6375558834039196
         RMSLE value of Testing Data is 0.6634873592375078
```

Interpretation of RMSLE value:

- It is a way of figuring out how much a model disagrees with the actual data
- The lower this number is, the better the fit of the model
- It is preferred that the value of RMSLE should be close to the minimum value of the response variable in test data



Checking the min and max values of y_test corresponding to our RMSLE value **0.66**

```
In [47]: print('Min value of y_test is',  min(y_test))
         print('Max value of y_test is',  max(y_test))

         Min value of y_test is 1.0986122886681098
         Max value of y_test is 7.498869733976931
```

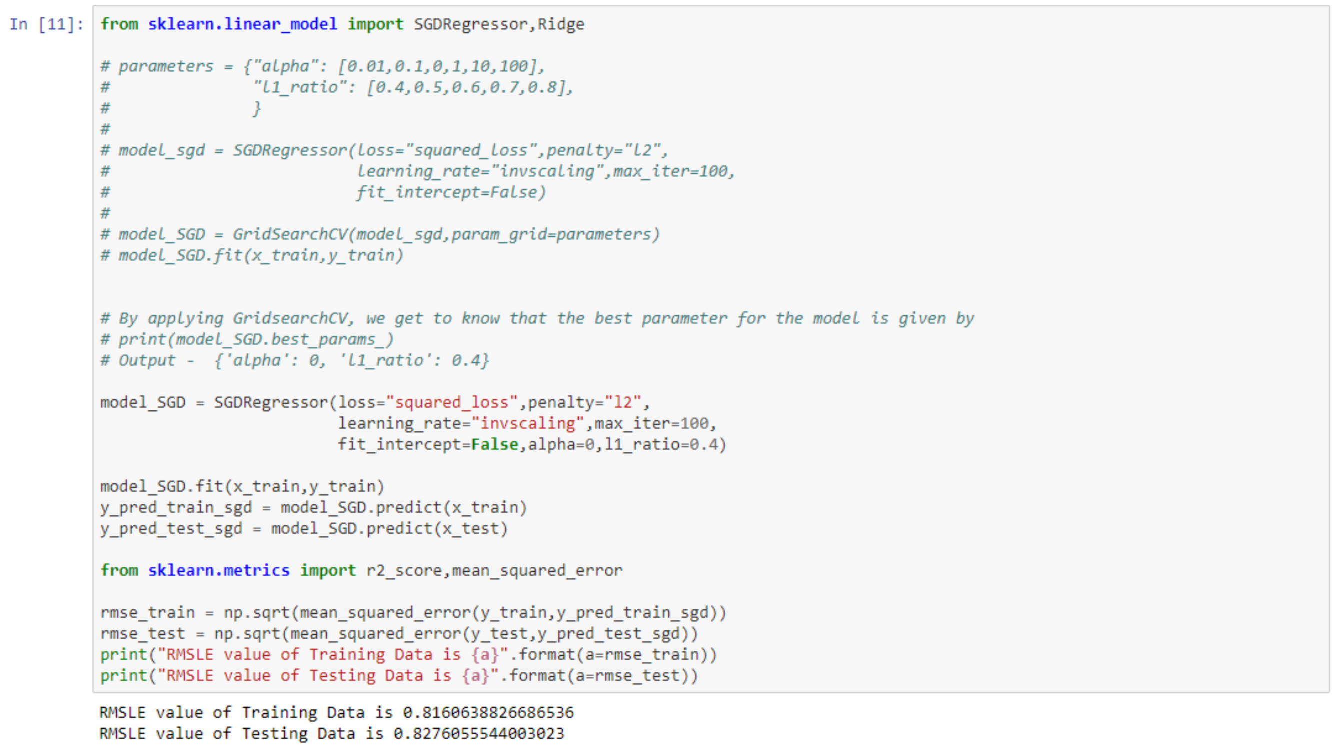
Insight from Ridge Model:

As we can see that 0.66 is *close* to 1.09, we can conclude that our model is able to predict the value close to the actual value

4. SGD (Stochastic Gradient Descent) Model:

Overview:

- In statistics, machine learning and data science fields, we optimize a lot of stuff
- When we fit a line with linear regression, we optimize the slope and intercept
- In logistic regression, we optimize the squiggly line
- In a similar way Gradient descent also helps optimize such things and much more
- In this case we have almost half billion data points and hence computations can take a long time
- This method uses a randomly selected subset of data at every step rather than full dataset
- This reduces the time spent calculating derivatives of loss function (Sum of squared Residuals)



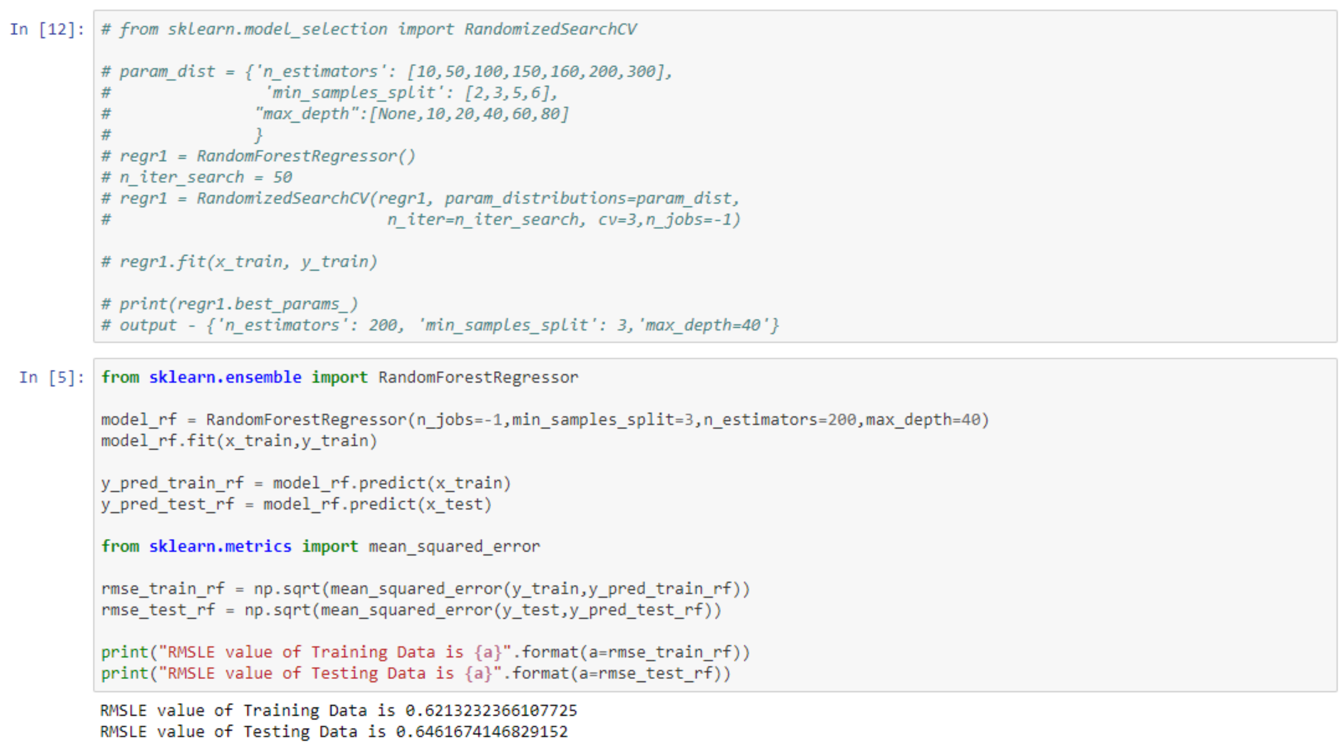
```
In [11]: from sklearn.linear_model import SGDRegressor,Ridge

         # parameters = {"alpha": [0.01,0.1,0,1,10,100],
         #               "l1_ratio": [0.4,0.5,0.6,0.7,0.8],
         #               }
         #
         # model_sgd = SGDRegressor(loss="squared_loss",penalty="l2",
         #                          learning_rate="invscaling",max_iter=100,
         #                          fit_intercept=False)
         #
         # model_SGD = GridSearchCV(model_sgd,param_grid=parameters)
         # model_SGD.fit(x_train,y_train)


         # By applying GridsearchCV, we get to know that the best parameter for the model is given by
         # print(model_SGD.best_params_)
         # Output -  {'alpha': 0, 'l1_ratio': 0.4}

         model_SGD = SGDRegressor(loss="squared_loss",penalty="l2",
                                  learning_rate="invscaling",max_iter=100,
                                  fit_intercept=False,alpha=0,l1_ratio=0.4)

         model_SGD.fit(x_train,y_train)
         y_pred_train_sgd = model_SGD.predict(x_train)
         y_pred_test_sgd = model_SGD.predict(x_test)

         from sklearn.metrics import r2_score,mean_squared_error

         rmse_train = np.sqrt(mean_squared_error(y_train,y_pred_train_sgd))
         rmse_test = np.sqrt(mean_squared_error(y_test,y_pred_test_sgd))
         print("RMSLE value of Training Data is {a}".format(a=rmse_train))
         print("RMSLE value of Testing Data is {a}".format(a=rmse_test))

         RMSLE value of Training Data is 0.8160638826686536
         RMSLE value of Testing Data is 0.8276055544003023
```

Insight from SGD model:

- We can see that although the RMSLE is **0.82** which is very less yet not as good as the RMSLE in the Ridge Model

5. Random Forest Regression Model:

- Decision trees have the possibility of overfitting and have inaccuracy for predictive learning as not all the variables give less impurity at each split
- Also they are not flexible with new data
- This problem is easily overcome by random forests as it combines the simplicity of the decision trees with flexibility resulting in a vast improvement in accuracy
- Bagging method is used to solve the problem we are facing
- Variance decreases because of the sub samples created and 1/3rd of the observations are left for validation purpose



```
In [12]: # from sklearn.model_selection import RandomizedSearchCV

         # param_dist = {'n_estimators': [10,50,100,150,160,200,300],
         #                'min_samples_split': [2,3,5,6],
         #               "max_depth":[None,10,20,40,60,80]
         #               }
         # regr1 = RandomForestRegressor()
         # n_iter_search = 50
         # regr1 = RandomizedSearchCV(regr1, param_distributions=param_dist,
         #                            n_iter=n_iter_search, cv=3,n_jobs=-1)

         # regr1.fit(x_train, y_train)

         # print(regr1.best_params_)
         # output - {'n_estimators': 200, 'min_samples_split': 3,'max_depth=40'}
```

```
In [5]:  from sklearn.ensemble import RandomForestRegressor

         model_rf = RandomForestRegressor(n_jobs=-1,min_samples_split=3,n_estimators=200,max_depth=40)
         model_rf.fit(x_train,y_train)

         y_pred_train_rf = model_rf.predict(x_train)
         y_pred_test_rf = model_rf.predict(x_test)

         from sklearn.metrics import mean_squared_error

         rmse_train_rf = np.sqrt(mean_squared_error(y_train,y_pred_train_rf))
         rmse_test_rf = np.sqrt(mean_squared_error(y_test,y_pred_test_rf))

         print("RMSLE value of Training Data is {a}".format(a=rmse_train_rf))
         print("RMSLE value of Testing Data is {a}".format(a=rmse_test_rf))

         RMSLE value of Training Data is 0.6213232366107725
         RMSLE value of Testing Data is 0.6461674146829152
```

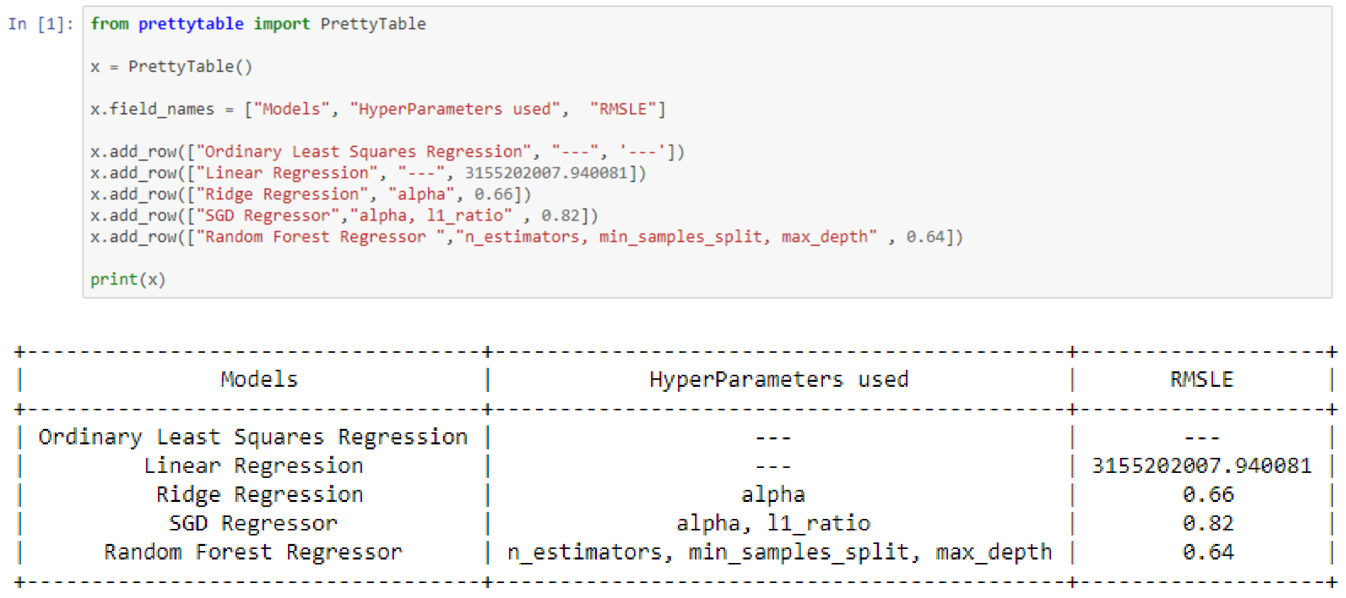
Insight from Random Forest Model:

- RMSLE value we obtained here **0.64** is a tiny bit, yet better than the RMSLE we got in Ridge model, thus the best model so far

CHAPTER 4

CONCLUSION

Conclusion:



```
In [1]: from prettytable import PrettyTable

        x = PrettyTable()

        x.field_names = ["Models", "HyperParameters used",  "RMSLE"]

        x.add_row(["Ordinary Least Squares Regression", "---", '---'])
        x.add_row(["Linear Regression", "---", 3155202007.940081])
        x.add_row(["Ridge Regression", "alpha", 0.66])
        x.add_row(["SGD Regressor","alpha, l1_ratio" , 0.82])
        x.add_row(["Random Forest Regressor ","n_estimators, min_samples_split, max_depth" , 0.64])

        print(x)
```

```
+-----------------------------------+-------------------------------------------+-------------------+
|               Models              |            HyperParameters used           |       RMSLE       |
+-----------------------------------+-------------------------------------------+-------------------+
| Ordinary Least Squares Regression |                    ---                    |        ---        |
|         Linear Regression         |                    ---                    | 3155202007.940081 |
|          Ridge Regression         |                   alpha                   |        0.66       |
|           SGD Regressor           |              alpha, l1_ratio              |        0.82       |
|      Random Forest Regressor      | n_estimators, min_samples_split, max_depth |        0.64       |
+-----------------------------------+-------------------------------------------+-------------------+
```

- We couldn't find RMSLE in OLS model and the Linear model had a very huge value
- **Multi-collinearity** in the data lead us to build models where we could fine tune it according to the best parameters provided by **grid search mechanism**
- Out of all the models we applied, **Random Forest Regressor** gave us the best and the lowest RMSLE value **0.64**
- Hence we can deploy this model at the clients machine so that sellers can get an estimate of the price they should set for their products
- Another solution would be making an app wherein the seller will enter the parameters one by one and after submitting he will get the result as the **Suggested Price** for the inputs provided by him/her

Summary:

- The Dataset is imported from Kaggle and basic data analysis is done for all the features to extract useful information
- We did **feature engineering** and incorporated 4 **additional features** 'Length of item description' and 'Main Category', 'Sub Category 1', ' Sub Category 2' that influence price of an item fairly
- We did necessary **preprocessing** on the text features, **one-hot encoding** for the categorical features and **tf-idf vectorization** for **text** features
- This is a **regression** problem and we have used **'OLS Regression'**, **'Linear Regression'**, **'Ridge Regression'**, **'SGD regressor'** and **'Random Forest regressor'** for our models
- "**RMSLE**" was the error metric required, so all our algorithms calculated RMSLE as the error metric
- Hyper parameters were **tuned** for each of the models and the best parameters were used to predict prices on the test dataset
- After using **ensembling** modelling, we brought down the RMSLE score and it was reported to be **0.64**
- So, ensemble modelling i.e. **Random Forest Regressor** gives the **best** performance for our data

References

1. Investopedia
2. Wikipedia
3. YouTube (Theoretical Info.)
4. Kaggle (Mercari Data set)
5. Stack Overflow
6. Python Documentations