Mercari Price Suggestion Challenge

I. Definition

Project Overview

Can you automatically suggest product prices to online sellers?

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.

Mercari, Japan's biggest community-powered shopping app, knows this problem deeply. They'd like to offer pricing suggestions to sellers, but this is tough because their sellers are enabled to put just about anything, or any bundle of things, on Mercari's marketplace.

In this Kaggle competition, Mercari's challenging us to build an algorithm that automatically suggests the right product prices. You'll be provided user-inputted text descriptions of their products, including details like product category name, brand name, and item condition.

I chose this data set because I can explore various machine learning approaches to solve this problem and also use Natural Language processing concepts since it involves text data as well.

Source of the data challenge: https://www.kaggle.com/c/mercari-price-suggestion-challenge

Problem Statement

- **Define the objective in business terms :** The objective is to come up with the right pricing algorithm that can we can use as a pricing recommendation to the users.
- **How will your solution be used? :** Allowing the users to see a suggest price before purchasing or selling will hopefully allow more transaction within Mercari's business.
- **How should you frame this problem? :** This problem can be solved using a supervised learning approach, and possible some unsupervised learning methods as well for clustering analysis.

Metrics

- **How should performance be measured? :** Since its a regression problem, the evaluation metric that should be used is RMSE (Root Mean Squared Error). But in this case for the competition, we'll be using the RMSLE; which puts less penalty on large errors and focuses more on the smaller errors (since our main distribution in price is centered at around $10)

- **Are there any other data sets that you could use? :** To get a more accurate understanding and prediction for this problem, a potential dataset that we can gather would be more about the user. Features such as user location, user gender, and seasonality.

II. Analysis

Data Exploration

The dataset is provided by the competition organizer in the form of two tab

separated files, train.tsv amd test.tsv.

The data set can be downloaded from : https://www.kaggle.com/c/mercari-price-suggestion-challenge/data

Dataset Features areas follows:

- ID: the id of the listing
- Name: the title of the listing
- Item Condition: the condition of the items provided by the seller
- Category Name: category of the listing
- Brand Name: brand of the listing
- Shipping: whether or not shipping cost was provided
- Item Description: the full description of the item
- Price: the price that the item was sold for. This is the target variable that you will predict. The unit is USD.

- The training data set has 1482535 rows and 8 feature columns.
- The test data has 693359 rows and 7 columns. The 'Price' column is the the target variable here.
- For my project I have only taken the training data since it's a supervised learning regression problem and we have to predict the prices of the items listed.

Exploratory Visualization

The first thing to do is to perform an exploratory data analysis to understand different features, their distribution, get summary statistics, how different features varies with the target variable 'Price'.

This will allow us to clean the data by appropriately filling in missing values, getting rid of unexpected values, normalize the scale of different features as

necessary and create new features from existing ones to help us better predict the prices.

Descriptive Statistics :

```
In [9]: train_data.astype('object').describe().T
```

Out[9]:

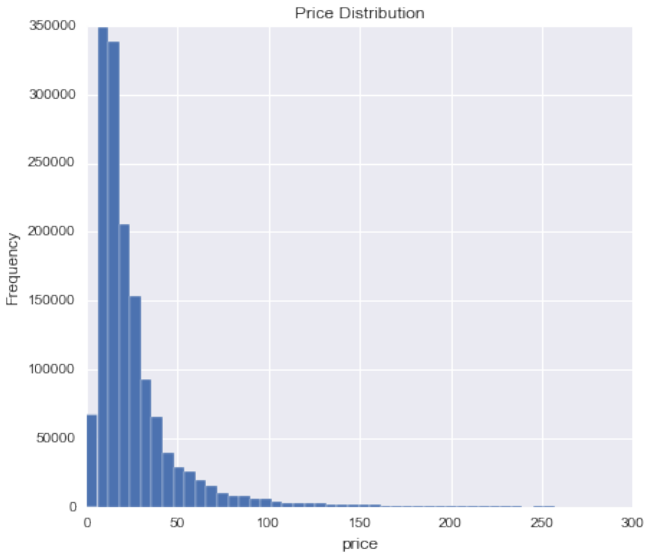
	count	unique	top	freq
train_id	1482535	1482535	1482534	1
name	1482535	1225273	Bundle	2232
item_condition_id	1482535	5	1	640549
category_name	1476208	1287	Women/Athletic Apparel/Pants, Tights, Leggings	60177
brand_name	849853	4809	PINK	54088
price	1.48254e+06	828	10	99416
shipping	1482535	2	0	819435
item_description	1482531	1281426	No description yet	82489

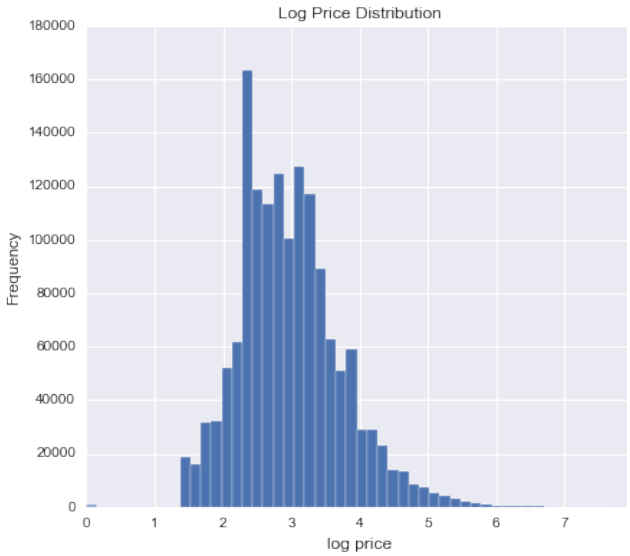
- The most common title for a listing is **Bundle**.
- The most common brand name is **PINK**.
- Most common price point is **10 dollars**.
- About 82k listings do not have descriptions.
- The most common item category is **Women apparel**.
- About 82k listings do not have shipping cost provided.

Looking at the Price distribution

```
In [13]: # let's look at Price distribution
         pd.options.display.float_format = "{:.3f}".format
         train_data.price.describe()
```

```
Out[13]: count   1482535.000
         mean         26.738
         std          38.586
         min           0.000
         25%          10.000
         50%          17.000
         75%          29.000
         max        2009.000
         Name: price, dtype: float64
```





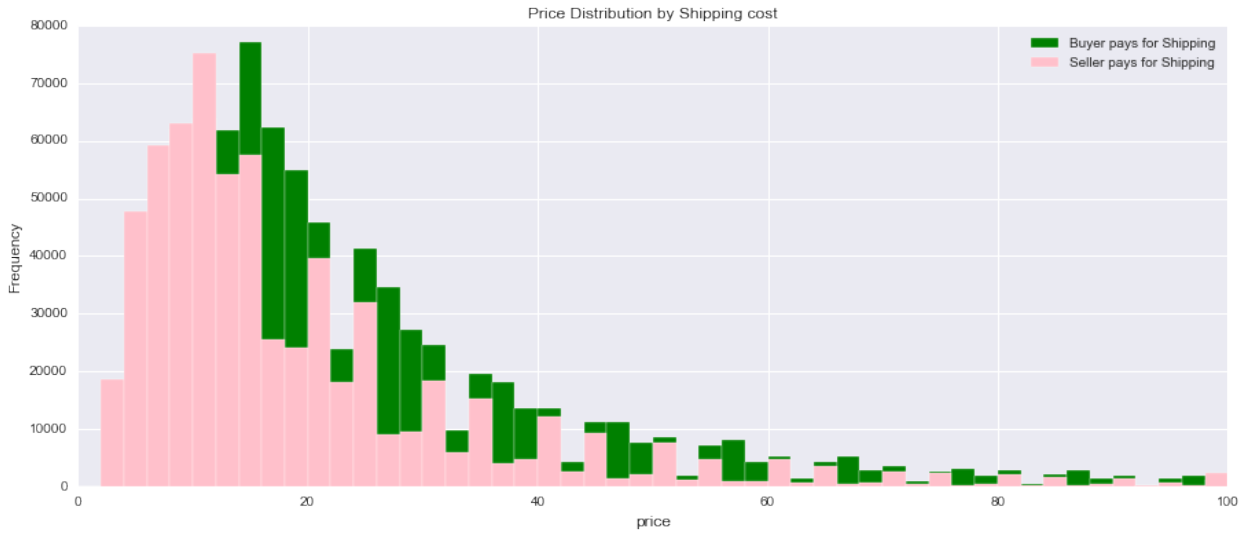
- Average Price of the items is **$26**.
- Median price is **$17**.
- Maximum amount is **$2000**.
- Most of the items are sold at **$10**.
- There are 874 listings that are priced at $0, which seems like data entry error. We can safely remove them from our analysis.

Why we are taking log(price) ? If we look at the price distribution, we see that most of the items have been sold for close to 10 dollars. The price distribution is heavily skewed on left side for lower values. Therefore, we can take can transform it using log transformation.

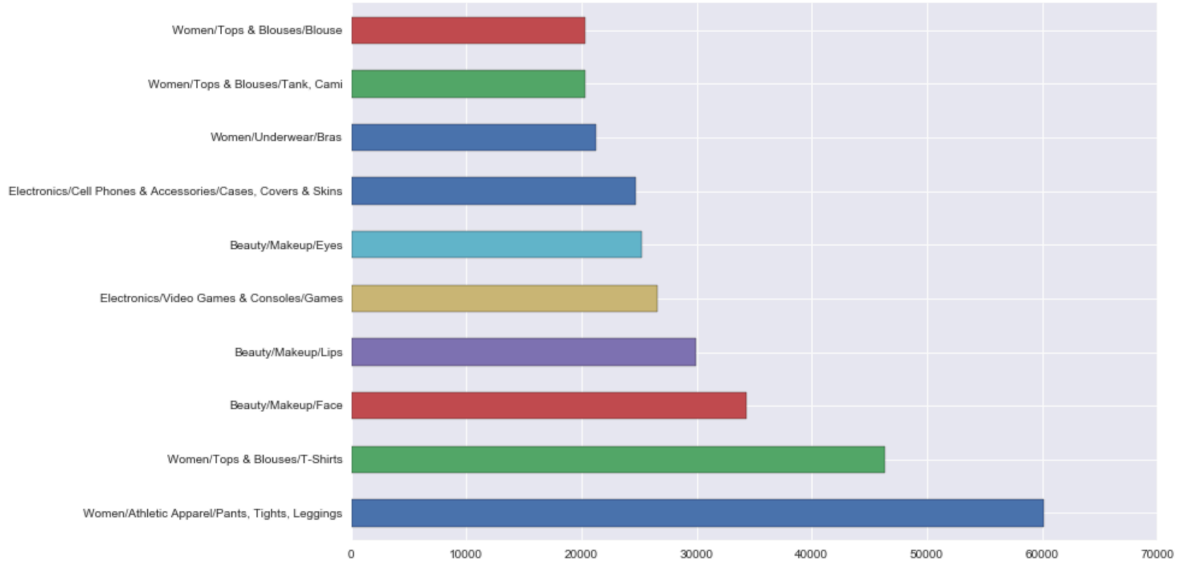
This is a regression problem, and the error can be measured using RMSE(Root Mean Square Error), but since price has a long right tail distribution, we should use RMSLE(Root Mean Square Log Error). We are doing this so that the errors for lower priced items are penalized more than higher priced items. So, we do the log transformation of the price (our target variable). The evaluation metric is RMSLE. We then convert the log of price to actual price using exponential transformation.

Looking at the Shipping distribution

We want to see how prices for a listing varies if the shipping cost is included in the price or not. The below visualization show the price distribution when buyer pays for the shipping versus when seller pays for the shipping cost. Clearly, we can see from the below graph that, prices tend to be higher when shipping cost is not provided and buyer pays for the shipping.



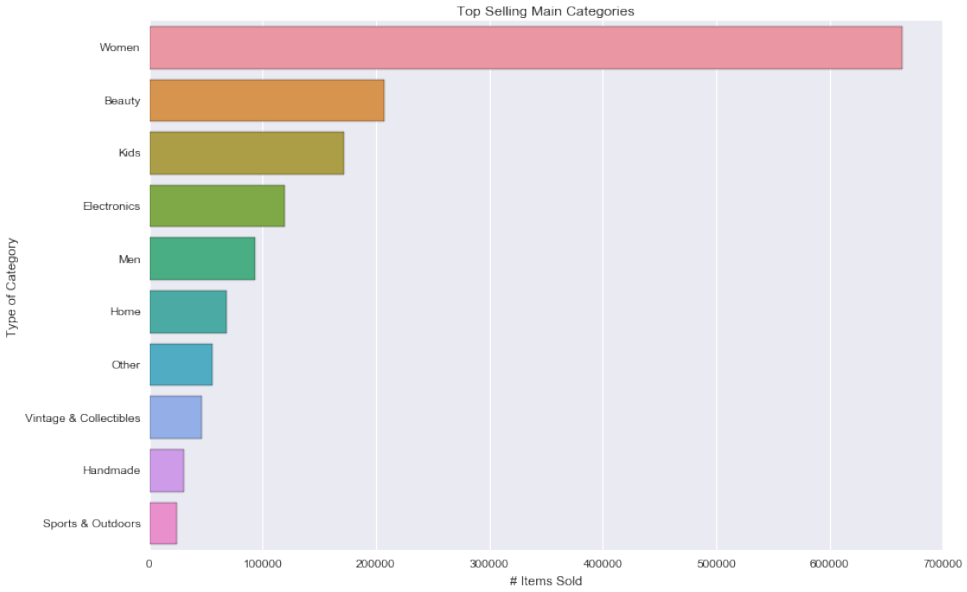
Looking at Top Selling Categories



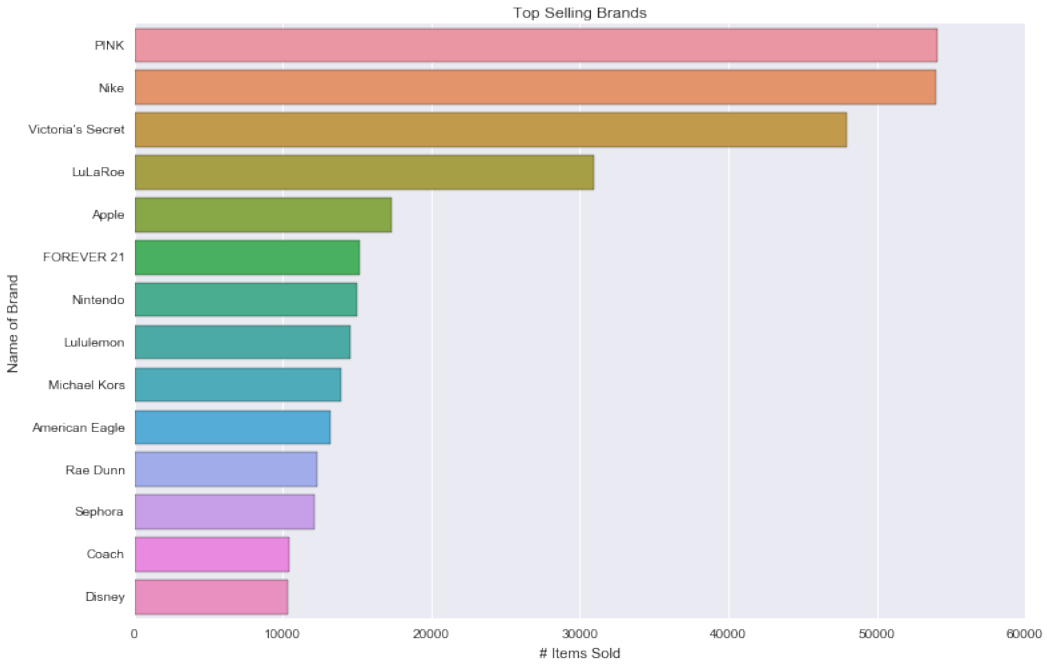
- Women apparel is sold the most.
- It looks like the category feature has a hierarchy like *main category/category/subcategory*, so should split them into main category and two sub categories to create new features. This is feature engineering.

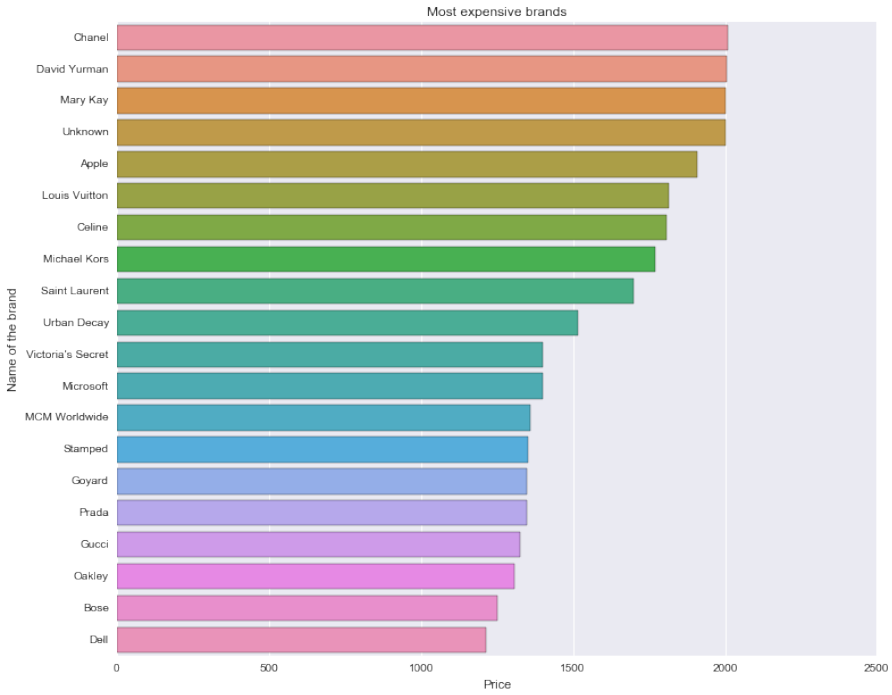
	category_main	category_sub1	category_sub2
0	Men	Tops	T-shirts
1	Electronics	Computers & Tablets	Components & Parts
2	Women	Tops & Blouses	Blouse
3	Home	Home Décor	Home Décor Accents
4	Women	Jewelry	Necklaces

- After splitting the hierarchy, let's look at the top selling main categories. We find that Women categories sells the most followed by Beauty and Kids.



Looking do some Brand Analysis by looking at top selling and most expensive brands in all the listings provided.





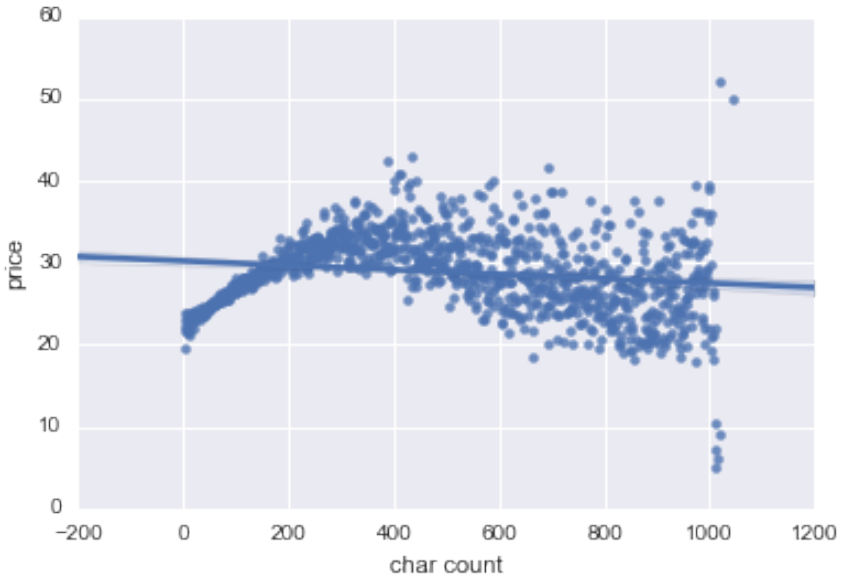
- Pink, Nike and Victoria Secret are the most selling brands.
- Most expensive brands are Chanel, David Yurman, Mary Kay, Apple and Louis Vitton.

- We have explored all of the categorical variables, now its time to use natural language processing technique to explore text features.
- It would be good to see if the description of any particular item affects the price point of the item.

Effect of item description length to the selling price.

Now we will explore text features.

It might be possible that a better described have more selling price vs item description with a few words. To understand relationship between selling price and the length of item description, we can create additional column, character count and create a scatter plot between price and character count.



- By looking at above scatter plot, it seems there is an increase in price point when the item description length goes from 0 to 400 characters. But after that it does not matter much.

Topic Modelling – Latent Dirichlet Allocation

One of the best ways to make sense of larger text data is through topic modelling. Topic models allows us to cluster words into topics. **Latent Dirichlet Allocation** or LDA is one of the most widely used algorithm for topic modelling. I have used LDA to explore and visualize most important topics in the feature 'item description'.

In order for us to use LDA, we have to convert the text data into numeric vector representation. There are different feature extract methods such **CountVectorizer**, **TDF-IDFVectorizer** etc can help us to do that. I am using CountVectorizer, which convers words into count vectors with resulting array of each word and word count within the text. Below we can see results of the LDA.

n [53]:
```
# Get the topic words
topic_word = lda_model.components_
# Get the vocabulary from the text features
vocab = cvectorizer.get_feature_names()


# Define variables
n_top_words = 10
topic_summaries = []


# Display the Topic Models
for i, topic_dist in enumerate(topic_word):
    topic_words = np.array(vocab)[np.argsort(topic_dist)][:-(n_top_words+1):-1]
    topic_summaries.append(' '.join(topic_words))
    print('Topic {}: {}'.format(i, ' | '.join(topic_words)))
```

```
Topic 0: size | new | worn | brand | tags | like | small | medium | cute | fit
Topic 1: new | brand | description | box | used | firm | price | oz | authentic | color
Topic 2: pink | size | free | white | black | blue | smoke | home | secret | victoria
Topic 3: comes | gift | box | includes | quality | body | included | pretty | scratches | works
Topic 4: rm | shipping | free | bundle | ship | price | items | new | save | ask
Topic 5: bag | gold | silver | black | inside | leather | used | pocket | time | beautiful
Topic 6: item | look | clean | left | face | wash | order | add | picture | beauty
Topic 7: iphone | case | use | phone | inches | plus | lot | hair | skin | screen
Topic 8: condition | great | good | used | size | times | perfect | men | shape | boots
Topic 9: set | bought | condit | great | piece | short | picture | game | necklace | come
```

Here we see a list of 10 topics, with 10 top words in each of the topics. Actually looking at all the 10 words for each of the topics is not providing us with overall topic we can assign it to. Running LDA took a really long time, I wanted to play around with it more, but had resource constraints.

Algorithms and Techniques

- Since this is regression problem, I will start by using Linear Regression. I will also make sure feature scaling is done correctly.
- After that, I will optimize my results by using Ridge, Lasso and XG Boost Regression Algorithms.
- Ridge and Lasso allows for adding penalty parameters into order to avoid overfitting.
- XG Boost is a very power algorithm from the family of tree based ensemble algorithm. It can handle nonlinear and highly dimensional data very well. I think I will get the best results using XB Boost Regressor.

Benchmark

- Since it's a regression problem, where we have to predict the prices for the listed item. The baseline or benchmark model that automatically registers our mind is using 'Multiple Linear regression'.
- The way linear regression works is, it tries to find relationship between our predictor variables and target variables. The model is a linear equation for a straight line, and the coefficients are learned during the training the model. The values of coefficients provide is an idea of which feature is more important than others.
- To evaluate our benchmark model using linear regression, we are going to use 'log of root mean square error' as our evaluation criterion. We will keep the same criterion to evaluate other models as well, this will allow us to compare benchmark model with other refined and better models we create later.
- One of the drawbacks of linear regression is that, it is influenced by outliers in the data set, so it is important to either normalize the data by scaling it or using some kind of penalty to constrain the values of parameters learned during training.
- Once we achieve a baseline model using linear regression, we can use

other algorithms like Ridge, Lasso, Random Forest etc to create more sophisticated models that are more robust to outliers and can handle non linearity in the data as well.

```
# Create Linear Regression with 5-Fold CV

kf = KFold(X_train.shape[0],n_folds=5, shuffle=True, random_state=42)
for train_ids, valid_ids in kf:
    # Define Linear Regression Model
    model_linear = LinearRegression(fit_intercept = True, normalize = True)

    # Fit Ridge Regression Model
    model_linear.fit(X_train[train_ids], y_train[train_ids])

    # Predict & Evaluate Training Score
    y_pred_train = model_linear.predict(X_train[train_ids])
    rmsle_train = get_rmsle(y_pred_train, y_train[train_ids])

    # Predict & Evaluate Validation Score
    y_pred_valid = model_linear.predict(X_train[valid_ids])
    rmsle_valid = get_rmsle(y_pred_valid, y_train[valid_ids])


    print('Linear Regression Training RMSLE:', rmsle_train)
    print('Linear Regression Validation RMSLE', rmsle_valid)
```

```
Linear Regression Training RMSLE: 0.458452346556
Linear Regression Validation RMSLE 0.488405025765
Linear Regression Training RMSLE: 0.458636207633
Linear Regression Validation RMSLE 0.487230225134
Linear Regression Training RMSLE: 0.458547477402
Linear Regression Validation RMSLE 0.487153298977
Linear Regression Training RMSLE: 0.45850227322
Linear Regression Validation RMSLE 0.487596780558
Linear Regression Training RMSLE: 0.458025040819
Linear Regression Validation RMSLE 0.489425893927
```

III. Methodology

1. Data Pre-processing

- In real world data is always messy and most of the time is spent on cleaning and preparing the data to be used by machine learning algorithms.

- In our exploratory data analysis, we found that many if the input features have missing values. It is very important to handle missing values appropriately, else they can bias our results or can lead to wrong interpretation of the results.
- We see a lot of listing with brand names missing. We can impute the missing values by ‘Unknown’.
- There 6327 category names missing can be replaced with ‘Other’.
- And the missing item description can be replaced with ‘No Description’.

```
# Let's look at the missing values
train_data.isnull().sum()
```

```
train_id                0
name                    0
item_condition_id        0
category_name         6327
brand_name          632682
price                    0
shipping                 0
item_description         4
dtype: int64
```

```
# filling missing values
train_data['category_name'].fillna(value = 'Other', inplace = True)
train_data['brand_name'].fillna(value = 'Unknown', inplace = True)
train_data['item_description'].fillna(value = 'No Description', inplace = True)
```

- We see that ‘the category name’ feature has a hierarchy like *main category/category/subcategory*, so should split them into main category and two sub categories to create new features. This is feature engineering. Therefore, I created a function that will take the ‘category name’ and split into three parts, the first will be the main category and other two are the subcategories.

```
def create_categories(category_name):
    try:
        main, sub1, sub2= category_name.split('/')
        return main, sub1, sub2
    except:
        return 'Other','Other','Other'
```

```
train_data['category_main'], train_data['category_sub1'], train_data['category_sub2'] = zip(*train_data['category_name'
```

- There are 874 listings that are priced at $0, which seems like data entry error. We can safely remove them from our analysis.

```
train_data[train_data.price == 0].shape
```

```
(874, 8)
```

```
train_data = train_data[train_data.price !=0]
```

```
train_data.shape
```

```
(1481661, 8)
```

Text Processing

In order for us to use text features, such as item description, category names, brand names, we need to first normalize the text and then convert it to numeric vector representation.

We will use NLTK (Natural Language Processing Toolkit) library to work on text features. To normalize we have to:

- **Removing Punctuations**: it is important to get rid of all the punctuations in the text.
- **Making lower case**: making all the text into lower case ensures that we are not treating same word differently in different cases.
- **Removing stop words** (most common words in english language) : In any language there are words which are very common and do not add much value when we are trying to understand the text. It is important to remove them, else they will dominate the analysis and model prediction. We can use ‘stopwords’ module from nltk to get rid of predefined common words. We can also concatenate this with our own list of common words based on specific text we are dealing with.
- **Stemming the words :** Stemming the words to their root word allows to treact related words same. We can use ‘porter’ stemmer to do this, it is available in nltk.

Let's look at the 'item_description' and 'name' after normalizing and stemming.

```
In [46]: # let's see how item description looks like after normalization and stemming
         train_data['item_description'][100:105]

Out[46]: 100    what goes better summer tacos tequila chill fr...
         101                  any questions please ask price firm
         102                           size medium perfect condit
         103    size kids brand new paid rm firm on the price ...
         104    everything perfect condition tags heaven famil...
         Name: item_description, dtype: object
```

```
In [47]: train_data['name'][5:10]

Out[47]: 5              bundled items requested ruie
         6        acacia pacific tides santorini top
         7               girls cheer tumbling bundle
         8                    girls nike pro shorts
         9    porcelain clown doll checker pants vtg
         Name: name, dtype: object
```

Change the data types to work with the algorithm

- Changing the data types for 'item condition', 'brand name', 'category main' to categorical variables.

```
train_data["category_main"] = train_data["category_main"].astype("category")
```

```
train_data["item_condition_id"] = train_data["item_condition_id"].astype("category")
```

```
train_data["brand_name"] = train_data["brand_name"].astype("category")
```

- Creating numerical vectors using CountVectorizer() and TFIDFVectorizer() for 'name' and 'item description', 'category main'.

- CountVectorizer counts the words and provides the frequencies. With the TFIDFVectorizer the value increases proportionally to count, but is offset by the frequency of the word in the corpus. - This is the IDF (inverse document frequency part). This helps to adjust for the fact that some words appear more frequently.

- The hyperparameters min_df and max_features are used to remove the infrequent words or limit the amount of features (vocabulary) the vectorizer will learn.

```
count = CountVectorizer(min_df=10)
X_name = count.fit_transform(train_data["name"])
```

```
count_category = CountVectorizer()
X_category = count_category.fit_transform(train_data["category_main"])
```

```
count_descp = TfidfVectorizer(max_features = 50000,
                              ngram_range = (1,3),
                              stop_words = "english")
X_descp = count_descp.fit_transform(train_data["item_description"])
```

- For feature ‘brand names’, we are using ‘Label Binarizer()’ to encode them to numeric values. We can also use One hot encoding as well.

```
from sklearn.preprocessing import LabelBinarizer
vect_brand = LabelBinarizer(sparse_output=True)
X_brand = vect_brand.fit_transform(train_data["brand_name"])
```

- Similarly, we have converted ‘shipping’ and ‘item condition id’ which are also categorical variables into numeric representation.

2. Implementation

Combining all the transformed features into Sparse matrix.

Now, we are ready to create to use out training data and features for prediction. I have used ‘hstack’ from scipy library, which creates horizontal stacking of the data. After that, we have to convert the features matrix to sparse matrix because there are are lot of zeros in the transformed numeric vectors and we need to create sparse matric for better efficiencies.

```
In [128]: X_train = scipy.sparse.hstack((X_dummies,
                                         X_descp,
                                         X_brand,
                                         X_category,
                                         X_name)).tocsr()
```

Defining evaluation method.

Since out project in hand is a regression problem where we have to predict the prices of the listed items on mercari, we will be using 'root mean square error' to see how our prediction algorithm is performing. As we observed earlier that our target variable 'price' is heavily skewed toward lower values and therefore we have to change our evaluation criterion to logarithm of root mean square error.

```
def get_rmsle(y, pred): return np.sqrt(mean_squared_error(y, pred))
```

Benchmark

- Since it's a regression problem, where we have to predict the prices for the listed item. The baseline or benchmark model that automatically registers our mind is using 'Multiple Linear regression'.
- The way linear regression works is, it tries to find relationship between our predictor variables and target variables. The model is a linear equation for a straight line, and the coefficients are learned during the training the model. The values of coefficients provide is an idea of which feature is more important than others.
- To evaluate our benchmark model using linear regression, we are going to use 'log of root mean square error' as our evaluation criterion. We will keep the same criterion to evaluate other models as well, this will allow us to compare benchmark model with other refined and better models we create later.
- One of the drawbacks of linear regression is that, it is influenced by

outliers in the data set, so it is important to either normalize the data by scaling it or using some kind of penalty to constrain the values of parameters learned during training.

- Once we achieve a baseline model using linear regression, we can use other algorithms like Ridge, Lasso, Random Forest etc to create more sophisticated models that are more robust to outliers and can handle non linearity in the data as well.

```
# Create Linear Regression with 5-Fold CV

kf = KFold(X_train.shape[0],n_folds=5, shuffle=True, random_state=42)
for train_ids, valid_ids in kf:
    # Define Linear Regression Model
    model_linear = LinearRegression(fit_intercept = True, normalize = True)

    # Fit Ridge Regression Model
    model_linear.fit(X_train[train_ids], y_train[train_ids])

    # Predict & Evaluate Training Score
    y_pred_train = model_linear.predict(X_train[train_ids])
    rmsle_train = get_rmsle(y_pred_train, y_train[train_ids])

    # Predict & Evaluate Validation Score
    y_pred_valid = model_linear.predict(X_train[valid_ids])
    rmsle_valid = get_rmsle(y_pred_valid, y_train[valid_ids])


    print('Linear Regression Training RMSLE:', rmsle_train)
    print('Linear Regression Validation RMSLE', rmsle_valid)
```

```
Linear Regression Training RMSLE: 0.458452346556
Linear Regression Validation RMSLE 0.488405025765
Linear Regression Training RMSLE: 0.458636207633
Linear Regression Validation RMSLE 0.487230225134
Linear Regression Training RMSLE: 0.458547477402
Linear Regression Validation RMSLE 0.487153298977
Linear Regression Training RMSLE: 0.45850227322
Linear Regression Validation RMSLE 0.487596780558
Linear Regression Training RMSLE: 0.458025040819
Linear Regression Validation RMSLE 0.489425893927
```

- In order to train our model, we split the data into train and validation. During training the linear regression model will learn the parameters which minimizes the error function or perform well on the evaluation criterion.
- I have used K-fold cross validation to observe how well the model is going, is it

overfitting or not and will it generalize well on any new data sets.

- Once the model is trained, we fit the model parameters to training data. After this, we make predictions on the test data. We calculate difference between true value for prices of items listed and predicted prices, take square and then log of that value. This is our evaluation criterion.
- As we can see from the above results, Linear Regression gives a training error of 45% and test error close to 49%.
- We will use these results to compare it to other models.

3. Refinement

Next steps in the process is to try different algorithms, to see if they perform any better than our baseline model. This process will also involve doing hyperparameter tuning.

Predicting prices using Ridge regression with K-fold cross validation

- As mentioned earlier, one of the drawbacks of using linear regression is that it is influenced by outliers and can suffer from problem of multi-collinearity, which happens when the predictor features are highly correlated with each other. This leads to model instability and can impact the performance of the model adversely. One way to avoid this issue is to introduce penalty to error function which will limit the values of coefficients attached to predictor variables. We can use Ridge regression. Ridge regression is basically our linear regression, but good part is it handles issue of multi-collinearity very well by penalizing coefficient with higher values.

```
# Create 5-Fold CV

kf = KFold(X_train.shape[0],n_folds=5, shuffle=True, random_state=42)
for train_ids, valid_ids in kf:
    # Define Ridge Regression Model
    model_ridge = Ridge(solver = "lsqr", fit_intercept=True, random_state=42)

    # Fit Ridge Regression Model
    model_ridge.fit(X_train[train_ids], y_train[train_ids])

    # Predict & Evaluate Training Score
    y_pred_train = model_ridge.predict(X_train[train_ids])
    rmsle_train = get_rmsle(y_pred_train, y_train[train_ids])

    # Predict & Evaluate Validation Score
    y_pred_valid = model_ridge.predict(X_train[valid_ids])
    rmsle_valid = get_rmsle(y_pred_valid, y_train[valid_ids])


    print('Ridge Training RMSLE:', rmsle_train)
    print('Ridge Validation RMSLE', rmsle_valid)
```

```
Ridge Training RMSLE: 0.4606959424053349
Ridge Validation RMSLE 0.48341287071286004
Ridge Training RMSLE: 0.4608009696669849
Ridge Validation RMSLE 0.48283888517701923
Ridge Training RMSLE: 0.4607232751956621
Ridge Validation RMSLE 0.4831722174710801
Ridge Training RMSLE: 0.46070241594949424
Ridge Validation RMSLE 0.48300317929756875
Ridge Training RMSLE: 0.4602185183702532
Ridge Validation RMSLE 0.4851761017020206
```

- We observe from above model that our error is around 46% on training data and 48% on validation set/test set.
- I am also using k-fold cross validation to see how the model is doing on different folds of test data. This helps understand whether our model is overfitting or not. This way we will have an idea if the model will be able to generalize on new data or not.
- If we compare Ridge regression model, compared to our baseline model of just Linear Regression, we can see that Ridge is performing more or less same as Linear regression. So intuitively we should prefer linear regression over ridge this case of this problem. But one thing that I observed was training our baseline linear regression took longer than training ridge regression model. So, if speed is a concern, sometimes we might have to prefer a model with relatively lower performance but is faster to output the results. This is a tradeoff we can do by talking to the business stakeholders.

Predicting prices using Lasso with K-fold cross validation

```
kf = KFold(X_train.shape[0],n_folds=5, shuffle=True, random_state=42)
for train_ids, valid_ids in kf:
    # Define Lasso Regression Model

    model_LASSO = Lasso(fit_intercept=True, random_state=42)

    # Fit Lasso Regression Model
    model_LASSO.fit(X_train[train_ids], y_train[train_ids])

    # Predict & Evaluate Training Score
    y_pred_train = model_LASSO.predict(X_train[train_ids])
    rmsle_train = get_rmsle(y_pred_train, y_train[train_ids])

    # Predict & Evaluate Validation Score
    y_pred_valid = model_LASSO.predict(X_train[valid_ids])
    rmsle_valid = get_rmsle(y_pred_valid, y_train[valid_ids])


    print('Lasso Training RMSLE:', rmsle_train)
    print('Lasso Validation RMSLE', rmsle_valid)
```

```
Lasso Training RMSLE: 0.7460417348909648
Lasso Validation RMSLE 0.7454686912702255
Lasso Training RMSLE: 0.7461269264231346
Lasso Validation RMSLE 0.7451274390682916
Lasso Training RMSLE: 0.746014784538195
Lasso Validation RMSLE 0.7455765035642409
Lasso Training RMSLE: 0.7462221944811667
Lasso Validation RMSLE 0.7447457339837585
Lasso Training RMSLE: 0.7452295715333028
Lasso Validation RMSLE 0.7487108954676894
```

- We observe from above model that our error is around 74% on training data and 75% on validation set/test set.
- This shows that Lasso is doing worse than Ridge regression.
- Lasso also belongs to family of multiple linear regression just like ridge. But in Lasso the penalty for overfitting is defined differently. In Lasso we take absolute values of model coefficients and add it to our error function, this can lead to some of the coefficients reducing to zero values and therefore helps in feature selection.

Predicting prices using XGBoost Regressor with K-fold cross validation

```
from xgboost import XGBRegressor
```

```
kf = KFold(X_train.shape[0],n_folds=5, shuffle=True, random_state=42)
for train_ids, valid_ids in kf:
    # Define light gbm Model

    model_xgb = XGBRegressor()

    # Fit light gbm Model
    model_xgb.fit(X_train[train_ids], y_train[train_ids])

    # Predict & Evaluate Training Score
    y_pred_train = model_xgb.predict(X_train[train_ids])
    rmsle_train = get_rmsle(y_pred_train, y_train[train_ids])

    # Predict & Evaluate Validation Score
    y_pred_valid = model_xgb.predict(X_train[valid_ids])
    rmsle_valid = get_rmsle(y_pred_valid, y_train[valid_ids])


    print('XGBoost Training RMSLE:', rmsle_train)
    print('XGBoost Validation RMSLE', rmsle_valid)
```

```
XGBoost Training RMSLE: 0.6266541617215364
XGBoost Validation RMSLE 0.6264454136074914
XGBoost Training RMSLE: 0.626680262607618
XGBoost Validation RMSLE 0.6255996499890663
XGBoost Training RMSLE: 0.6263736221007529
XGBoost Validation RMSLE 0.6271790771000194
XGBoost Training RMSLE: 0.6277236868396517
XGBoost Validation RMSLE 0.6259352542944666
XGBoost Training RMSLE: 0.6264606935710202
XGBoost Validation RMSLE 0.630414085366093
```

- We see that XG Boost regressor model gives error around 62% on training data and around same on validation set/test set.
- XG Boosting algorithm comes from the family of ensemble boosting trees and have been performing really well on many machine learning problems, so I decided to use it for this problem as well. But it is actually performing poorly than ridge.

IV. Results

Model Evaluation and Validation

- Ridge regression with k-fold is doing best overall compared to other two models. So I will keep that as my final model.

- The resulting model has many good qualities like it is simple to implement, without much complexity involved. There are no many hyperparameters to tune, so it can provide immediate value to business.
- The final parameters of the model are:

```
model_ridge.get_params
```

```
<bound method BaseEstimator.get_params of Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=42, solver='lsqr', tol=0.001)>
```

alpha = 0.1. alpha is regularization strength. It is tuned to reduce overfitting by reducing variance.

fit_intercept = True, is used to calculate the intercept value for the model.

Normalize = True, all the input features are normalized before fitting the model. I have kept if to False.

Solver = 'lsqr', uses the dedicated regularized least-squares routine scipy.sparse.linalg.lsqr. It is the fastest and uses an iterative procedure.

- The final model is robust and is not sensitive to outliers. The data is normalized and scaled before fitting. In addition, ridge regression adds penalty parameters which is square of coefficients to the loss function. This makes sure that model does not over fit and also resulting model is stable.
- The values of evaluation criterion, log of root mean square error on each of the 5 fold cross validation is close to each other. This shows that model is not overfitting and will generalize well on new data set. If the values of errro hops between wide range across different folds, then it means model is not robust and will not lead to same results every time.

Justification

- In fact, some of the teams in the kaggle leaderboard all wen with some or other variation of ridge regression. In addition, using K-fold cross validation allows us to see validate how robust is our model, and we can see from the results that it is able to generalize well.
- We can also modify the results to suit other price prediction problems. But we

have done feature engineering with respect to the mercari data, we have change it to suit other problem, however it definitely will give an approach and a head start.

V. Conclusion

Free Form Visualization

ELI5: which features are important for price prediction. Or, explain like I'm 5, how does a linear ridge predict prices?

EIL5 is a library that can help us with that, let's see it in action. It has support for many models, including XGBoost and LightGBM, but we'll be using it to analyze our final Ridge mode.

Let's check some predictions from the validation set. We get a summary of various vectorizer's contribution at the top, and then below you can see features highlighed in text.

```
eli5.show_prediction(model, doc=train.values[100], vec=vectorizer)
```

y (score **2.551**) top features

Contribution[?]	Feature
+2.972	<BIAS>
+0.230	item_description: Highlighted in text (sum)
-0.068	brand_name: Highlighted in text (sum)
-0.075	item_condition_id: Highlighted in text (sum)
-0.079	category_name: Highlighted in text (sum)
-0.178	name: Highlighted in text (sum)
-0.251	shipping: Highlighted in text (sum)

name: muscle t-shirt
category_name: women/tops & blouses/t-shirts
brand_name: missing
shipping: 0
item_condition_id: 3
item_description: what goes better with summer than tacos & tequila? chill out with friends sporting this great beachwear cover or wear as a stand alon with that oh so sexy bralette or bikini top! don't forget your cool shades! great condition! worn once, no stains, holes, rips or treats.

Improvement

- There is still room for improvement in the project with respect to feature engineering. Such as creating features like ‘unique words in item description’, ‘most popular brand’ etc. This will be for future work.
- In addition, dealing with text data is very challenging and requires lot processing time and compute resources. I used my local machine to do all the analysis and it took around forty minutes to just run LDA. One way to improve processing time is to use store the data in AWS instance and use PySpark to parallelize the operations. This can be done for future cases. As the data increases, we need to use better computing resources.
- One other challenge was that text data is messy and requires a lot of clean up and normalization before using it. Also, when we convert the text into numeric vector representation it creates very sparse matrix. Sparse matrices are matrices with lot of zeros, so when we have these matrices our compute time increases significantly. One thing I can do to improve is to use Selectbest() feature selection and run the model only top features.
- Another way to make an improvement is to to use deep learning algorithms such as Recurrent Neural Nets to make price predictions. RNNs have been very successful where sequences have been involved, just like text data. This will definitely require use of GPU resources.

Reflection

- Overall, I truly believe it was a good project, that not only required me to apply machine learning techniques learned in the nanodegree, but also allowed me to learn and apply concepts in natural language processing.
- The interesting thing is that the results of project can be used in lot of use cases. Using my learnings from nanodegree and applying it to a real world problem was definitely a fulfilling experience.

References

- “Airbnb's Pricing Algorithm and Aerosolve, Its Open-Source Machine Learning Tool.” *Tnooz*,
- www.tnooz.com/article/airbnbs-pricing-algorithm-and-aerosolve-its-open-source-machine- learning-tool/
- Demner-Fushman, D., Chapman, W. W., & Mcdonald, C. J. (2009). What can natural language processing do for clinical decision support? *Journal of Biomedical Informatics, 42*(5), 760-772. doi:10.1016/j.jbi.2009.08.007
- https://www.kaggle.com/thykhuely/mercari-interactive-eda-topic-modelling