In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```
import os
os.chdir("/content/drive/My Drive/Classroom/projects/Mercari")
!ls -l
```

```
total 6189307
-rw------- 1 root root        151 Nov 19 17:35 akarshan.1711@gmail.com_CS1.gdoc
-rw------- 1 root root        151 Dec 16 13:22 EDA+FE.gdoc
-rw------- 1 root root    2441752 Dec 20 16:29 EDA.ipynb
-rw------- 1 root root      14393 Dec 27 21:06 FE+prep+modelling.ipynb
-rw------- 1 root root      30163 Dec 29 18:34 HptBrnandImpute.v1.0.ipynb
-rw------- 1 root root      40352 Dec 30 20:02 HptTfidf.v1.0.ipynb
-rw------- 1 root root     927353 Dec 28 15:17 mercari_mainV2.ipynb
-rw------- 1 root root    7996136 Dec 30 19:54 price_log2.pickle
-rw------- 1 root root   20384538 Dec 29 15:57 price_log_BrandImput.pickle
-rw------- 1 root root  308669128 Dec 10  2019 test_stg2.tsv.zip
-rw------- 1 root root 1407107658 Dec 30 19:54 tfidf2.pickle
-rw------- 1 root root  610480534 Dec 29 15:57 tfidf_BrandImpute.pickle
-rw------- 1 root root 3641944242 Dec 29 17:10 tfidf.pickle
-rw------- 1 root root  337809843 Nov 11  2017 train.tsv
```

In [ ]:

```
#importing modules/libraries
import pandas as pd
import numpy as np
import scipy
import seaborn as sns
import matplotlib.pyplot as plt
import gc
import sys
import os
import psutil
# from scipy.stats import randint as sp_randint
# from scipy.stats import uniform as sp_uniform


from tqdm.notebook import tqdm
# from collections import Counter
# from collections import defaultdict
import re
import random
# from random import sample
# from bs4 import BeautifulSoup
import pickle
import inspect
import time

import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelBinarizer
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
import lightgbm as lgb
from sklearn.linear_model import Lasso,Ridge


import string
# import emoji
```

```python
# from wordcloud import WordCloud
import nltk
nltk.download("stopwords")
# nltk.download("brown")
# nltk.download("names")
# nltk.download('punkt')
nltk.download('wordnet')
# nltk.download('averaged_perceptron_tagger')
# nltk.download('universal_tagset')
# from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
# from nltk.stem.porter import PorterStemmer





import warnings
warnings.filterwarnings("ignore")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

In [ ]:

```python
# function to load train as well as Test data(3 times larger than train data)
def Data(clock,n_rows):
# using n_rows to get a subset of data to debug
    if int(n_rows) == -1:
        tr = pd.read_csv('train.tsv', sep='\t')
        ts = pd.read_csv('test_stg2.tsv.zip',sep ='\t')
    else:
        tr = pd.read_csv('train.tsv', sep='\t',nrows = n_rows)
        ts = pd.read_csv('test_stg2.tsv.zip',sep ='\t',nrows = n_rows)



    tr = tr.drop(['train_id'], axis =1)
    ts = ts.drop(['test_id'], axis =1)

# dropping rows with invalid price
    tr = tr.drop(tr[tr.price < 1.0].index)
    tr.reset_index(inplace=True)
# changing price to Normal distribution by log transformation so that linear
# models don't give negative prediction
    y = np.log1p(tr.price)
    df=pd.concat([tr,ts],axis=0, ignore_index=True)
    df['item_condition_id'] = df['item_condition_id'].astype('category')
    df['shipping'] = df['shipping'].astype('category')


    gc.collect()
   # print fucntion completion time and with function name
    print(f'[{round((time.time() - clock),2)}] {inspect.stack()[0][3]} completed')
    return df, y,round(tr.shape[0]*0.8),tr.shape[0]
```

In [ ]:

```python
def Impute(df, tr_len,clock):
  # imputing with a 'abs' or absent
    df['category_name'].fillna(value='abs', inplace=True)
    df['name'].fillna(value='abs', inplace=True)
    df['item_description'].fillna(value='No Description Yet', inplace=True)

    # using brands from train data only
    tr = df.iloc[:tr_len,:]
    test = df.iloc[tr_len:,:]
```

```
    # imputing with a 'abs' or absent for train data and using  brand
    # names form train data only as target and freqency encoding done
    # in later sections ofnotebook so avoiding data leakage
      tr['brand_name'].fillna(value='abs',inplace=True)
      brand_name =tr.brand_name.unique()
      test.loc[~test['brand_name'].isin(brand_name),'brand_name'] = 'abs'

     # print fucntion completion time and with function name
      print(f'[{round((time.time() - clock),2)}] {inspect.stack()[0][3]} completed')
      del brand_name
      del df
      gc.collect()
      return pd.concat([tr,test],axis=0, ignore_index=True)
```

In [ ]:

```
# a category column contains Nan or 3 or more sub category in it upto 5
# as rows with more than 3 or less than 3 categories are less than 0.1 percent,
# we make only 3 new cols with segregated category names
def sub_cat(row):
      try:
        split = row.split('/')
        if len(split) >= 3:
          return split[0],split[1],split[2]
        if len(split) == 2:
          return split[0], split[1], 'abs'
        elif len(split) == 1:
          return split[0], 'abs', 'abs'
        else:
          return 'abs', 'abs', 'abs'
      except Exception:
          return  'abs', 'abs', 'abs'

# extracting extra features from data
def Extract_features(df,tr_len,clock):
  # regex used later in this section to count number of them in a text column
    RE_PUNCTUATION = '|'.join([re.escape(x) for x in string.punctuation])
    non_alphanumpunct = re.compile(u'[^A-Za-z0-9\.?!,; \(\)\[\]\'\"\$]+')

  # extracting sub categories
    print(f'[{round(time.time()-clock)}]Extracting Subcat')
    df['sc1'], df['sc2'],df['sc3'] =  zip(*df['category_name'].apply(sub_cat))
    df.drop(columns='category_name',inplace = True)
    df['sc1'] = df['sc1'].astype('category')
    df['sc2'] = df['sc2'].astype('category')
    df['sc3'] = df['sc3'].astype('category')

  # has description or not/ missing value added as a feature
    print(f'[{round(time.time()-clock)}]Extracting HasDescription ')
    df['HasDescription'] = 1
    df.loc[df['item_description']=='No description yet', 'HasDescription'] = 0
    df['HasDescription'] =df['HasDescription'].astype('category')

  # has price or not/ [rm] values in textual columns are indicative of presence
  # price in the data which has been cleaned as suggested by the compition itself
    print(f'[{round(time.time()-clock)}]Extracting HasPrice ')
    df['HasPrice'] = 0
    df.loc[df['item_description'].str.contains('[rm]', regex=False), 'HasPrice'] = 1
    df.loc[df['name'].str.contains('[rm]', regex=False), 'HasPrice'] = 1
    df['HasPrice'] =df['HasPrice'].astype('category')

    gc.collect()
  # counting number of tokens in textual columns
    print(f'[{round(time.time()-clock)}]Extracting Token Count ')
    df['NameTokenCount'] = df['name'].str.split().apply(len)
    df['DescTokenCount'] = df['item_description'].str.split().apply(len)
    df['NameTokenCount'] = df['NameTokenCount'].astype('uint32')
    df['DescTokenCount'] = df['DescTokenCount'].astype('uint32')

  # ratio of token token counts in name and description columns(2 textual cols)
```

```
    print(f'[{round(time.time()-clock)}]Extracting Name to Desc token Ratio ')
    df['NameDescTokenRatio'] = df['NameTokenCount']/df['DescTokenCount']
    df['NameDescTokenRatio'] =df['NameDescTokenRatio'].astype('float32')

  # adding missing value as a feature for brand
    print(f'[{round(time.time()-clock)}]Extracting HasBrand ')
    df['HasBrand'] =1
    df.loc[df['brand_name'] == 'abs', 'HasBrand'] = 0
    df['HasBrand'] =df['HasBrand'].astype('category')

  # counting uppper and lower count of characters as EDA suggested phoney/
  # counterfiet items when listed uses too many bold and Caps charactes with emojis
    print(f'[{round(time.time()-clock)}]Extracting Lower count ')
    df['NameLowerCount'] = df.name.str.count('[a-z]')
    df['DescriptionLowerCount'] = df.item_description.str.count('[a-z]')
    df['NameLowerCount'] =df['NameLowerCount'].astype('uint32')
    df['DescriptionLowerCount'] =df['DescriptionLowerCount'].astype('uint32')


    print(f'[{round(time.time()-clock)}]Extracting Upper count ')
    df['NameUpperCount'] = df.name.str.count('[A-Z]')
    df['DescriptionUpperCount'] = df.item_description.str.count('[A-Z]')
    df['NameUpperCount'] =df['NameUpperCount'].astype('uint32')
    df['DescriptionUpperCount'] =df['DescriptionUpperCount'].astype('uint32')

  # punctuation count
    print(f'[{round(time.time()-clock)}]Extracting Punctuation Count ')
    df['NamePunctCount'] = df.name.str.count(RE_PUNCTUATION)
    df['DescriptionPunctCount'] = df.item_description.str.count(RE_PUNCTUATION)
    df['NamePunctCount'] =df['NamePunctCount'].astype('uint32')
    df['DescriptionPunctCount'] =df['DescriptionPunctCount'].astype('uint32')

  # punct count ratio
    print(f'[{round(time.time()-clock)}]Extracting Punctuation Ratio ')
    df['NamePunctCountRatio'] = df['NamePunctCount'] / df['NameTokenCount']
    df['DescriptionPunctCountRatio'] = df['DescriptionPunctCount'] / df['DescTokenCount'
]
    df['NamePunctCountRatio'] =df['NamePunctCountRatio'].astype('float32')
    df['DescriptionPunctCountRatio'] =df['DescriptionPunctCountRatio'].astype('float32')

  # digit count( if model can get a sense of bundled items)
    print(f'[{round(time.time()-clock)}]Extracting Digit count ')
    df['NameDigitCount'] = df.name.str.count('[0-9]')
    df['DescriptionDigitCount'] = df.item_description.str.count('[0-9]')
    df['NameDigitCount'] =df['NameDigitCount'].astype('uint32')
    df['DescriptionDigitCount'] =df['DescriptionDigitCount'].astype('uint32')

  # emoji and/or other nonalphanum count
    print(f'[{round(time.time()-clock)}]Extracting NonAlphaNum count ')
    df['NonAlphaDescCount'] = df['item_description'].str.count(non_alphanumpunct)
    df['NonAlphaNameCount'] = df['name'].str.count(non_alphanumpunct)
    df['NonAlphaDescCount'] =df['NonAlphaDescCount'].astype('uint32')
    df['NonAlphaNameCount'] =df['NonAlphaNameCount'].astype('uint32')


    cols = set(df.columns.values)
    non_num_col = {'name', 'item_condition_id', 'brand_name',
                   'shipping', 'item_description', 'sc1',
                   'sc2', 'sc3','HasDescription','HasPrice','HasBrand',
                    'price','index'
                   }

    cols_to_normalize = cols - non_num_col

  # normalizing all the counts and ratios
    print(f'[{round(time.time()-clock)}]Normalizing')
    df_to_normalize = df[list(cols_to_normalize)]
    df_to_normalize = (df_to_normalize - df_to_normalize.min()) / (df_to_normalize.max()
- df_to_normalize.min())


    df = df[list(non_num_col)]
```

```
    df = pd.concat([df, df_to_normalize],axis=1)

    df.drop(columns='index',inplace=True)

    del(df_to_normalize)
    gc.collect()


 '''  extracting mean categorical and brand price with minding data leakage with addition 
of
  random noise so making data more robust. An idea taken form a some youtube video of a 
  kaggle grandmaster . This noise addition clearly has impacted the performace of model v
ery positively. '''

    print(f'[{round(time.time()-clock)}]Extracting Mean price Categories')
    mean_dc = {}
    tr = df.iloc[:tr_len,:]
    ts = df.iloc[tr_len:,:]
    lst = ['sc1','sc2','sc3', 'brand_name']

  #imputing values for nan with mean
    def boundary_case(hmap,key):
      try:
        return float(hmap[key])*np.random.normal(1,0.1)
      except:
      '''  when cases in test data are not
        present in train data mean_dict[feat] returns a nan to tackle that this part
        has been added( tho with normal usage it does not occur as this has been
        taken care of in  the imputation part itself , i had an experiment run which
        produced those cases so made this part as permanent only) '''
        hmap.mean()*np.random.normal(1,0.1)


    for feat in lst:
      ''' for every categorical column in the list above  finding the mean price of
      every category in it and adding that price in a column with a noise added to it
      *np.randon.normal(1,0.1)'''
        mean_dc[feat] = tr.groupby(feat)['price'].mean().astype(np.float32)
        mean_dc[feat] /= np.max(mean_dc[feat])#normalising dict
        tr['MeanPrice_'+feat] = tr[feat].apply(lambda x : boundary_case(mean_dc[feat],x)
).astype(np.float32)
        tr['MeanPrice_'+feat].fillna( mean_dc[feat].mean(), inplace=True  )


        ts['MeanPrice_'+feat] = ts[feat].apply(lambda x : boundary_case(mean_dc[feat],x)
).astype(np.float32)
        ts['MeanPrice_'+feat].fillna( mean_dc[feat].mean(), inplace=True  )


    tr.drop(columns='price',inplace = True)
    ts.drop(columns='price',inplace = True)

    print(f'[{round((time.time() - clock),2)}] {inspect.stack()[0][3]} completed')

    del df,mean_dc
    gc.collect()
    return pd.concat([tr,ts],axis=0)
```

In [ ]:

```
def Make_text_column(df,clock):
    '''As we saw in EDA that brands with NAN values can be imputed with names and item_de
scription
    columns as there are brand names prevelent with more than 40 to 45 percent of chance.

    So instead of imputing so many brands (43 percent), just creating a new column by mer
ging
    brands_name, name and item_description and making a text column and letting tfidf tak
ing care of it.
    '''
```

```
    df['text'] = df['name'].astype(str)+' '+df['brand_name'].str.strip().astype(str)+' '
+df['item_description'].str.strip().astype(str)
    df = df.drop(columns=['item_description'])


    def decontracted(text):
    # tried many kinds of regex to clean the data but final result wasnt effected much wi
ht
    # this part so only doing necessary onces
        try:
            text = re.sub(u"won't", "will not", text)
            text = re.sub(u"can\'t", "can not", text)
            text = re.sub(u"n\'t", " not", text)
            text = re.sub(u"\'t", " not", text)
            # separating digits for a sense of count if bundled items sold
            text = u" ".join(re.split('(\d+)',text) )

        except:
            print('error')
        return text

    def clean(df,col):
        non_alphanums = re.compile(u'[^A-Za-z0-9 ]+')
        wl = WordNetLemmatizer()

        preprocessed_text = []
        for _,sentance in tqdm(df[col].iteritems(),total=df.shape[0]):

            sentance = decontracted(sentance)

            # nonalphanumeric character removal
            sentance = non_alphanums.sub(u' ', sentance)

           ''' did not lemmatize cause takes a lot of time and has negligible to no
            effect on performance
            did not convert to lower case as this takes a lot of time and can be done
            interensicly within TFIDF and Count vectorization along with text standardiz
ation'''
            # lemmetizing
            # sentance = ' '.join(wl.lemmatize(word.strip()) for word in sentance.split(
))
            sentance = ' '.join(word.strip() for word in sentance.split())

            preprocessed_text.append(sentance)
        df[col] = pd.Series(preprocessed_text).values
        del preprocessed_text
        return df


    print('Cleaning text')
    df= clean(df,'text')
    print(f'Done')

    print(f'[{round((time.time() - clock),2)}] {inspect.stack()[0][3]} completed')


    gc.collect()
    return df
```

In [ ]:

```
''' As brand has close to 5000 categories, converting it to numbers
 by Frequency encoding brand with minding data leakage, Tho  lgbm handles high order
 categorical columns efficiently this encoding boosted performance instead of just
 using OHE, also it is with addition of random noise so making data more robust and
 enhancing performance I had chosen to do the same with sub categorical columns but the
 did not perform that well and took extra memory space too'''
def high_categorical(df,ts,col='brand_name'):
        dictionary_frq = df[col].value_counts().to_dict()
        dict_replace = {k:(v/max(dictionary_frq.values()) * np.random.normal(1,0.01)) fo
r k,v in dictionary_frq.items()}
```

```
        col_cat: pd.Series.astype('float16') = df[col].map(dict_replace)
        col_cat_ts: pd.Series.astype('float16') = ts[col].map(dict_replace)
        del dictionary_frq
        del dict_replace
        gc.collect()
        return col_cat.values.reshape(-1,1),col_cat_ts.values.reshape(-1,1)

'''Converting all the data till yet to numeric form if not yet done'''
def Convert_to_predictor(df, tr_len,clock,stopwords=stopwords,high_categorical=high_categ
orical):
    try:
      df.drop(columns='index',inplace = True)
    except:
      pass

    df_dummies = scipy.sparse.csc_matrix(pd.get_dummies(df[['item_condition_id', 'shippi
ng',\
                                                          'HasDescription', 'HasPrice
','HasBrand',\
                                                          'sc1','sc2','sc3']], sparse
=True).values)

    df.drop(columns=['item_condition_id', 'shipping','HasDescription', 'HasPrice','HasBr
and'],inplace=True)
    df.drop(columns=['sc1','sc2','sc3'],inplace=True)

    print(f'[{round((time.time() - clock),2)}]Transform categories data completed.')


    cols = ['NamePunctCount', 'NameDigitCount', 'DescriptionDigitCount',\
            'NameUpperCount', 'DescriptionPunctCount', 'DescriptionPunctCountRatio', \
            'DescTokenCount', 'DescriptionUpperCount', 'NonAlphaDescCount', \
            'NonAlphaNameCount', 'NameTokenCount', 'NameLowerCount', \
            'NameDescTokenRatio', 'DescriptionLowerCount', 'NamePunctCountRatio',\
            'MeanPrice_sc1','MeanPrice_sc2','MeanPrice_sc3','MeanPrice_brand_name']

    df_num = scipy.sparse.csc_matrix(df[cols].values)

    df.drop(columns=cols,inplace=True)

    print(f'[{round((time.time() - clock),2)}]Transform numeric  data completed.')

    gc.collect()

    tr = df.iloc[:tr_len,:]
    test = df.iloc[tr_len:,:]
    gc.collect()
    del df
    vect = CountVectorizer(ngram_range=(1,3),min_df=5, max_df=0.85,
                           lowercase=True, max_features=50000,
                          analyzer='word', strip_accents = 'ascii',
                          stop_words= set(stopwords.words("english")))

    tr_name = scipy.sparse.csr_matrix(vect.fit_transform(tr.name))
    ts_name = scipy.sparse.csr_matrix(vect.transform(test.name))
    df_name = scipy.sparse.vstack((tr_name,ts_name),format='csc')

    tr.drop(columns=['name'],inplace=True)
    test.drop(columns=['name'],inplace=True)

    del vect,ts_name,tr_name

    print(f'[{round((time.time() - clock),2)}]Transform name data completed.')



    vect = TfidfVectorizer(ngram_range=(1,3),min_df=5, max_df=0.85,
                           lowercase=True, max_features=100000,
                          analyzer='word', strip_accents = 'ascii', smooth_idf=True,stop_w
ords= set(stopwords.words("english")))
```

```
    tr_text = scipy.sparse.csr_matrix(vect.fit_transform(tr.text))
    ts_text = scipy.sparse.csr_matrix(vect.transform(test.text))
    df_text = scipy.sparse.vstack((tr_text,ts_text),format='csc')

    tr.drop(columns=['text'],inplace=True)
    test.drop(columns=['text'],inplace=True)

    del vect,ts_text,tr_text

    print(f'[{round((time.time() - clock),2)}]Transform text data completed.')




    # frequency encoding brands
    tr_brand,ts_brand = high_categorical(tr,test)
    tr_brand,ts_brand = scipy.sparse.csr_matrix(tr_brand),scipy.sparse.csr_matrix(ts_bra
nd)
    df_brand = scipy.sparse.vstack((tr_brand,ts_brand),format='csc')

    tr.drop(columns=['brand_name'],inplace=True)
    test.drop(columns=['brand_name'],inplace=True)
    del tr_brand,ts_brand,high_categorical

    print(f'[{round((time.time() - clock),2)}]Transform brand data completed.')



    df_merge = scipy.sparse.hstack((df_brand,df_dummies, df_num,df_name, df_text ))
    print('Merge all data completed.')


    del df_brand,df_dummies,df_num,df_text,df_name
    print(f'[{round((time.time() - clock),2)}] {inspect.stack()[0][3]} complete')

    gc.collect()
    return df_merge
```

In [ ]:

```
# dummy function to work with just Train data
def Data_tronly(clock,n_rows):
  if int(n_rows) == -1:
    df = pd.read_csv('train.tsv', sep='\t')
  else:
    df = pd.read_csv('train.tsv', sep='\t',nrows =n_rows)

  df = df.drop(['train_id'], axis =1)
  df = df.drop(df[df.price <= 1.0].index)
  df.reset_index(inplace=True)
  df['item_condition_id'] = df['item_condition_id'].astype('category')
  # df =df[df['brand_name'].notnull()]
  y = np.log1p(df.price)
  gc.collect()
  print(f'[{round((time.time() - clock),2)}] {inspect.stack()[0][3]} completed')
  return df, y,round(df.shape[0]*0.8),df.shape[0]
```

In [ ]:

```
clock =time.time()
df,y, tr_len,whole_tr= Data(clock,n_rows = -1)
gc.collect()
df = Impute(df,tr_len,clock)
gc.collect()
df = Extract_features(df,tr_len,clock)
gc.collect()
df = Make_text_column(df,clock)
gc.collect()
df = Convert_to_predictor(df,tr_len,clock)
gc.collect()
```

```
[30.18] Data completed
```

```
[33.36] Impute completed
[35]Extracting Subcat
[56]Extracting HasDescription
[57]Extracting HasPrice
[62]Extracting Token Count
[114]Extracting Name to Desc token Ratio
[114]Extracting HasBrand
[114]Extracting Lower count
[180]Extracting Upper count
[203]Extracting Punctuation Count
[221]Extracting Punctuation Ratio
[221]Extracting Digit count
[239]Extracting NonAlphaNum count
[265]Normalizing
[269]Extracting Mean price Categories
[309.73] Extract_features completed
Cleaning text


Done
[445.71] Make_text_column completed
[521.32]Transform categories data completed.
[524.88]Transform numeric  data completed.
[620.49]Transform name data completed.
[1119.04]Transform text data completed.
[1120.62]Transform brand data completed.
Merge all data completed.
[1121.35] Convert_to_predictor complete
```

Out[ ]:

```
0
```

In [ ]:

```
with open('tfidf.pickle','wb') as f:
  pickle.dump(df,f)
```

In [ ]:

```
with open('price_log.pickle','wb') as f:
  pickle.dump(y,f)
```

In [ ]:

```
df.shape
```

Out[ ]:

```
(4942386, 151063)
```

In [ ]:

```
np.isnan(df.data).sum()
```

Out[ ]:

```
0
```

i have tried numerous variations of this notebook presenting the one which worked best yet.