Predicting Resale Value of Knives from a Texas Government Surplus Store

Using Machine Learning to Support an Ebay Store's Financial Success

Data Exploration and Modeling

Author: Dylan Dey

Model

In [1]:

```
from sklearn.model_selection import train_test_split
import os
from collections import Counter

import pandas as pd
import  json
import requests
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import ast
import re

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Input, GlobalMaxPooling1D
from tensorflow.keras.layers import LSTM, Embedding, Flatten, GRU
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalMaxPooling2D
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, BatchNormalization
from tensorflow.keras.models import Model
from keras import models
from keras import layers
import tensorflow as tf

from keras_preprocessing.image import ImageDataGenerator
```

Function Definition

In [2]:
```
def apply_iqr_filter(df):

    price_Q1 = df['converted_price'].quantile(0.25)
    price_Q3 = df['converted_price'].quantile(0.75)
    price_iqr = price_Q3 - price_Q1

    profit_Q1 = df['profit'].quantile(0.25)
    profit_Q3 = df['profit'].quantile(0.75)
    profit_iqr = profit_Q3 - profit_Q1

    ROI_Q1 = df['ROI'].quantile(0.25)
    ROI_Q3 = df['ROI'].quantile(0.75)
    ROI_iqr = ROI_Q3 - ROI_Q1

    price_upper_limit = price_Q3 + (1.5 * price_iqr)
    price_lower_limit = price_Q1 - (1.5 * price_iqr)

    profit_upper_limit = profit_Q3 + (1.5 * profit_iqr)
    profit_lower_limit = profit_Q1 - (1.5 * profit_iqr)

    ROI_upper_limit = ROI_Q3 + (1.5 * ROI_iqr)
    ROI_lower_limit = ROI_Q1 - (1.5 * ROI_iqr)

#     print(f'Brand: {df.brand[0]}')
#     print(f'price upper limit: ${np.round(price_upper_limit,2)}')
#     print(f'price lower limit: ${np.round(price_lower_limit,2)}')
#     print('-----------------------------------')
#     print(f'profit upper limit: ${np.round(profit_upper_limit,2)}')
#     print(f'profit lower limit: ${np.round(profit_lower_limit,2)}')
#     print('-----------------------------------')
#     print(f'ROI upper limit: {np.round(ROI_upper_limit,2)}%')
#     print(f'ROI lower limit: {np.round(ROI_lower_limit,2)}%')
#     print('-----------------------------------')


    new_df = df[(df['converted_price'] <= price_upper_limit) &
                (df['converted_price'] >= price_lower_limit) &
                (df['profit'] <= profit_upper_limit) &
                (df['ROI'] <= ROI_upper_limit) &
                (df['profit'] <= profit_upper_limit) &
                (df['ROI'] >= ROI_lower_limit)]

    return new_df
#download jpg urls from dataFrame
def download(row):
    filename = os.path.join(root_folder, str(row.name) + im_extension)

# create folder if it doesn't exist
    os.makedirs(os.path.dirname(filename), exist_ok=True)

    url = row.Image
#     print(f"Downloading {url} to {filename}")

    try:
        r = requests.get(url, allow_redirects=True)
        with open(filename, 'wb') as f:
            f.write(r.content)
    except:
        print(f'{filename} error')


def cardinality_threshold(column,threshold=0.75,return_categories_list=True):
    #calculate the threshold value using
    #the frequency of instances in column
    threshold_value=int(threshold*len(column))
    #initialize a new list for lower cardinality column
    categories_list=[]
    #initialize a variable to calculate sum of frequencies
    s=0
    #Create a dictionary (unique_category: frequency)
    counts=Counter(column)

    #Iterate through category names and corresponding frequencies after sorting the categories
    #by descending order of frequency
    for i,j in counts.most_common():
        #Add the frequency to the total sum
        s += dict(counts)[i]
        #append the category name to the categories list
        categories_list.append(i)
        #Check if the global sum has reached the threshold value, if so break the loop
        if s >= threshold_value:
            break
      #append the new 'Other' category to list
    categories_list.append('Other')

    #Take all instances not in categories below threshold
```

```
    #that were kept and lump them into the
    #new 'Other' category.
    new_column = column.apply(lambda x: x if x in categories_list else 'Other')

    #Return the transformed column and
    #unique categories if return_categories = True
    if(return_categories_list):
        return new_column,categories_list
    #Return only the transformed column if return_categories=False
    else:
        return new_column

def fix(col):
    dd = dict()
    for d in col:
        values = list(d.values())
        if len(values) == 2:
            dd[values[0]] = values[1]
    return dd

#function for extracted item Specifics from Shopping API data
def transform_item_specifics(df, perc=90.0):

    df.dropna(subset=['ItemSpecifics'], inplace=True)
    df['ItemSpecifics'] = df['ItemSpecifics'].apply(lambda x: ast.literal_eval(x))
    df['item_list'] = df['ItemSpecifics'].apply(lambda x: x['NameValueList'])

    df['ItemSpecifics'] = df['ItemSpecifics'].apply(lambda x: [x['NameValueList']] if isinstance(x['NameValueList'

    df['ItemSpecifics'] = df['ItemSpecifics'].apply(fix)

    df = pd.json_normalize(df['ItemSpecifics'])

    min_count =  int(((100-perc)/100)*df.shape[0] + 1)
    mod_df = df.dropna(axis=1,
                       thresh=min_count)

    return mod_df

# This function removes noisy data
#lots/sets/groups of knives can
#confuse the model from predicting
#the appropriate value of individual knives
def data_cleaner(df):
    lot = re.compile('(?<!-\S)lot(?![^\s.,:?!])')
    group = re.compile('(group)')
    is_set = re.compile('(?<!-\S)set(?![^\s.,?!])')
    df['title'] = df['title'].str.lower()
    trim_list = [lot,group,is_set]
    for item in trim_list:
        df.loc[df['title'].apply(lambda x: re.search(item, x)).notnull(), 'trim'] = 1
    to_drop = df.loc[df['trim'] == 1].index
    df.drop(to_drop, inplace=True)
    df.drop('trim', axis=1, inplace=True)

    return df



def prepare_listed(listed_data_df, Ids_df):
    listed_data_df.drop('galleryPlusPictureURL', axis=1, inplace=True)

    Ids_df.rename({'Title': 'title',
                  'ItemID': 'itemId'},
                   axis=1,inplace=True)

    Ids_df.drop(['ConditionID','ConvertedCurrentPrice'],
                        axis=1, inplace=True)
    Ids_df['title'] = Ids_df['title'].str.lower()

    df_merged = listed_data_df.merge(Ids_df)

    df_spec = transform_item_specifics(df_merged, perc=65.0)

    df_spec.drop('Brand', axis=1, inplace=True)

    tot_listed_df = df_merged.join(df_spec)

    listed_knives = data_cleaner(tot_listed_df).copy()
    listed_knives.drop(['sellingStatus', 'shippingInfo',
                        'GalleryURL', 'ItemSpecifics',
                        'item_list', 'listingInfo'],
                        axis=1, inplace=True)
    listed_used_knives = listed_knives.loc[listed_knives['condition'] != 1000.0]
    listed_used_knives.reset_index(drop=True, inplace=True)

```

```
    return listed_used_knives


def prepare_tera_df(df, x, overhead_cost=3):
    df['price_in_US'] = df['price_in_US'].str.replace("$", "")
    df['price_in_US'] = df['price_in_US'].str.replace(",", "")
    df['price_in_US'] = df['price_in_US'].apply(float)

    df['shipping_cost'] = df['shipping_cost'].str.replace("$", "")
    df['shipping_cost'] = df['shipping_cost'].str.replace(",", "")
    df['shipping_cost'] = df['shipping_cost'].apply(float)

    df['converted_price'] = (df['price_in_US'] + df['shipping_cost'])

    df['profit'] = ((df['converted_price']*.87) - list(bucket_dict.values())[x] - overhead_cost)
    df['ROI'] = (df['profit']/(list(bucket_dict.values())[x]))*100.0

    df['brand'] = list(bucket_dict.keys())[x]
    df['cost'] = list(bucket_dict.values())[x]


    return df

```

Load Data

In [3]:
```
#load Finding API data
df_bench = pd.read_csv("listed_data/df_bench.csv")
df_buck = pd.read_csv("listed_data/df_buck.csv")
df_case = pd.read_csv("listed_data/df_case.csv")
df_caseXX = pd.read_csv("listed_data/df_CaseXX.csv")
df_crkt = pd.read_csv("listed_data/df_crkt.csv")
df_kersh = pd.read_csv("listed_data/df_kershaw.csv")
df_sog = pd.read_csv("listed_data/df_sog.csv")
df_spyd = pd.read_csv("listed_data/df_spyderco.csv")
df_vict = pd.read_csv("listed_data/df_victorinox.csv")

#load Shopping API data
bench = pd.read_csv("listed_data/benchIds.csv")
buck = pd.read_csv("listed_data/buckIds.csv")
case = pd.read_csv("listed_data/caseIds.csv")
caseXX = pd.read_csv("listed_data/caseXXIds.csv")
crkt = pd.read_csv("listed_data/crktIds.csv")
kershaw = pd.read_csv("listed_data/kershawIds.csv")
sog = pd.read_csv("listed_data/sogIds.csv")
spyd = pd.read_csv("listed_data/spydIds.csv")
vict = pd.read_csv("listed_data/victIds.csv")

#Load scraped terapeak sold data
sold_bench = pd.read_csv("terapeak_data/bench_scraped2.csv")
sold_buck1 = pd.read_csv("terapeak_data/buck_scraped2.csv")
sold_buck2 = pd.read_csv("terapeak_data/buck_scraped2_reversed.csv")
sold_case = pd.read_csv("terapeak_data/case_scraped2.csv")
sold_caseXX1 = pd.read_csv("terapeak_data/caseXX_scraped2.csv")
sold_caseXX2 = pd.read_csv("terapeak_data/caseXX2_reversed.csv")
sold_crkt = pd.read_csv("terapeak_data/crkt_scraped.csv")
sold_kershaw1 = pd.read_csv("terapeak_data/kershaw_scraped2.csv")
sold_kershaw2 = pd.read_csv("terapeak_data/kershaw_scraped2_reversed.csv")
sold_sog = pd.read_csv("terapeak_data/SOG_scraped2.csv")
sold_spyd = pd.read_csv("terapeak_data/spyd_scraped2.csv")
sold_vict1 = pd.read_csv("terapeak_data/vict_scraped.csv")
sold_vict2 = pd.read_csv("terapeak_data/vict_reversed.csv")

sold_list = [sold_bench,sold_buck1,
             sold_buck2,sold_case,
             sold_caseXX1,sold_caseXX2,
             sold_crkt,sold_kershaw1,
             sold_kershaw2,sold_sog,
             sold_spyd, sold_vict1,
             sold_vict2]


listed_df = pd.concat([df_bench,df_buck,
                       df_case,df_caseXX,
                       df_crkt,df_kersh,
                       df_sog,df_spyd,
                       df_vict])


Ids_df = pd.concat([bench,buck,
                   case,caseXX,
                   crkt,kershaw,
                   sog,spyd,vict])

used_listed_df = prepare_listed(listed_df, Ids_df)



bucket_dict = {'benchmade': 45.0,
               'buck': 20.0,
               'case': 20.0,
               'crkt': 15.0,
               'kershaw': 15.0,
               'sog': 15.0,
               'spyderco': 30.0,
               'victorinox': 20.0
               }
```

Prepare Data

In [4]:
```
for dataframe in sold_list:
    dataframe.rename({'Text': 'title',
                      'shipping_': 'shipping_cost'},
                     axis=1, inplace=True)

    dataframe['date_sold'] = pd.to_datetime(dataframe['date_sold'])

sold_buck = pd.concat([sold_buck1,sold_buck2])
sold_caseXX = pd.concat([sold_caseXX1,sold_caseXX2])
sold_kershaw = pd.concat([sold_kershaw1,sold_kershaw2])
sold_vict = pd.concat([sold_vict1,sold_vict2])

sold_bench = prepare_tera_df(sold_bench, 0)
sold_buck = prepare_tera_df(sold_buck, 1)
sold_case = prepare_tera_df(sold_case, 2)
sold_caseXX = prepare_tera_df(sold_caseXX, 2)
sold_crkt = prepare_tera_df(sold_crkt, 3)
sold_kershaw = prepare_tera_df(sold_kershaw, 4)
sold_sog = prepare_tera_df(sold_sog, 5)
sold_spyd = prepare_tera_df(sold_spyd, 6)
sold_vict = prepare_tera_df(sold_vict, 7)
```

In [5]:
```
for dataframe in sold_list:
    dataframe['title'] = dataframe['title'].str.lower()
    dataframe['title'] = dataframe['title'].str.strip()
    dataframe.drop_duplicates(
        subset = ['date_sold','price_in_US',
                  'shipping_cost'],
        keep = 'last', inplace=True)
```

In [6]:
```
sold_df = pd.concat([sold_bench, sold_buck,
                     sold_case, sold_caseXX,
                     sold_crkt, sold_kershaw,
                     sold_sog, sold_spyd,
                     sold_vict])

sold_knives = data_cleaner(sold_df).copy()


df = pd.concat([sold_knives,used_listed_df]).copy()
df['Image'].fillna(df['pictureURLLarge'], inplace=True)

df = apply_iqr_filter(df).copy()
df.reset_index(drop=True, inplace=True)
```

In [7]:
```
df['title'] = df['title'].str.replace("'", "")
```

In [8]:
```
def clean_text(x):
    pattern = r'[^a-zA-z0-9\s]'
    text = re.sub(pattern, '', x)
    return x
```

In [9]:
```
df['title'] = df['title'].apply(clean_text)
```

In [12]:
```
df['title'].sample(20).apply(print)
```

```
case red stag tiny toothpick r511096 knifecase red stag tiny toothpick r511096 knife
kershaw crown liner lock knife 3.25" 3160 micarta scales
victorinox classic sd swiss army knife - silver aloxvictorinox classic sd swiss army knife - silver alox
buck sekiden dual linerlock two blade knife made in japan free shipping
case xx-usa--6220rsc  ss--bail peanut--magenta jigged bone--folding knifecase xx-usa--6220rsc  ss--bail peanut--magen
ta jigged bone--folding knife
case 63032 cv folding knife case 63032 cv folding knife
kershaw 1310wm spring assist folding pocket knife- fp128kershaw 1310wm spring assist folding pocket knife- fp128
case xx trapper knife 3207 ss smooth yellow delrin made/usacase xx trapper knife 3207 ss smooth yellow delrin made/us
a
1660swblk kershaw leek pocket knife plain blade usa  made black scales a14611660swblk kershaw leek pocket knife plain
blade usa  made black scales a1461
 kershaw brawler 1990 assisted open pocket knife liner lock plain edge blade  kershaw brawler 1990 assisted open pock
et knife liner lock plain edge blade
spyderco tenacious folding knife 3-3/8" satin plain blade, black g10 handlesspyderco tenacious folding knife 3-3/8" s
atin plain blade, black g10 handles
buck knives bos 5160  folding hunter lock-back knife gently usedbuck knives bos 5160  folding hunter lock-back knife
gently used
sog zoom zm1011 drop point spring-assisted knife satin 3.625" bladesog zoom zm1011 drop point spring-assisted knife s
atin 3.625" blade
crkt sting 3b fixed blade boot knife crkt sting 3b fixed blade boot knife
benchmade - mini griptilian 556 used manual open folding knife made in usabenchmade - mini griptilian 556 used manual
open folding knife made in usa
buck usa  482 lock blade folding pocket knifebuck usa  482 lock blade folding pocket knife
kershaw barstow assisted opening 8cr13mov spear point blade folding pocket knifekershaw barstow assisted opening 8cr1
3mov spear point blade folding pocket knife
benchmade usa mel pardue 154cm 530 folding pocket knifebenchmade usa mel pardue 154cm 530 folding pocket knife
victorinox rover swiss army knifevictorinox rover swiss army knife
case wildlife series grizzly pocket knife & wooden casecase wildlife series grizzly pocket knife & wooden case
```

Out[12]:
```
19296    None
71996    None
59074    None
66654    None
18202    None
18294    None
41454    None
25617    None
39208    None
42150    None
55176    None
14949    None
50479    None
33300    None
114      None
12896    None
40977    None
2365     None
60323    None
21510    None
Name: title, dtype: object
```

In [13]:
```
df['title_len'] = df['title'].apply(lambda x: len(x))
df['word_count'] = df['title'].apply(lambda x: len(x.split()))
```

In [15]:
```
def avg_word_len(x):
    words = x.split()
    word_len = 0
    for word in words:
        word_len += len(word)

    return word_len / len(words)
```

In [16]:
```
df['avg_word_len'] = df['title'].apply(lambda x: avg_word_len(x))
```

In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76400 entries, 0 to 76399
Data columns (total 41 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Image                          75859 non-null  object
 1   url                            13764 non-null  object
 2   date_sold                      66405 non-null  datetime64[ns]
 3   price_in_US                    76400 non-null  float64
 4   shipping_cost                  76400 non-null  float64
 5   title                          76400 non-null  object
 6   converted_price                76400 non-null  float64
 7   profit                         76400 non-null  float64
 8   ROI                            76400 non-null  float64
 9   brand                          76400 non-null  object
 10  cost                           76400 non-null  float64
 11  itemId                         9995 non-null   float64
 12  galleryURL                     9994 non-null   object
 13  viewItemURL                    9995 non-null   object
 14  autoPay                        9995 non-null   object
 15  postalCode                     9884 non-null   object
 16  returnsAccepted                9995 non-null   object
 17  condition                      9994 non-null   float64
 18  topRatedListing                9995 non-null   object
 19  pictureURLLarge                9454 non-null   object
 20  pictureURLSuperSize            9431 non-null   object
 21  PictureURL                     9994 non-null   object
 22  Location                       9994 non-null   object
 23  Country                        9995 non-null   object
 24  Blade Material                 5509 non-null   object
 25  Model                          7812 non-null   object
 26  Opening Mechanism              5951 non-null   object
 27  Number of Blades               6561 non-null   object
 28  Handle Material                6240 non-null   object
 29  Blade Type                     4429 non-null   object
 30  Color                          6896 non-null   object
 31  Type                           7693 non-null   object
 32  Country/Region of Manufacture  5433 non-null   object
 33  Lock Type                      4547 non-null   object
 34  Blade Edge                     5138 non-null   object
 35  Dexterity                      3616 non-null   object
 36  Original/Reproduction          4068 non-null   object
 37  Blade Range                    3629 non-null   object
 38  title_len                      76400 non-null  int64
 39  word_count                     76400 non-null  int64
 40  avg_word_len                   76400 non-null  float64
dtypes: datetime64[ns](1), float64(9), int64(2), object(29)
memory usage: 23.9+ MB
```

In [19]:
```
pd.options.plotting.backend = "plotly"
```

In [23]:
```
df['word_count'].plot(kind = 'hist', title = 'Word Count Distribution')
```

In [24]:
```
df['avg_word_len'].plot(kind='hist', bins = 50, title = 'Avg_Word_len Distribution')
```

In [29]:
```
df['title_len'].plot(kind='hist', bins= 100,title = 'Title Length Distribution')
```

In [ ]:
```

```

Neural network with "title" column as input

In [113]:
```
df_title = df.loc[:, ['title', 'converted_price']]


df_title.rename({'title': 'data',
                 'converted_price': 'labels'},
                axis=1, inplace=True)
```

In [114]:
```
df_title.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76400 entries, 0 to 76399
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   data    76400 non-null  object 
 1   labels  76400 non-null  float64
dtypes: float64(1), object(1)
memory usage: 1.2+ MB
```

In [115]:
```
df_title['data'].sample(10).apply(print)
```

```
2008 case xx medium stockman knife 63032 cv carbon vanadium& amber jig bone usa 2008 case xx medium stockman knife 63
032 cv carbon vanadium& amber jig bone usa
victorinox swiss army pocket knife - red super tinker multi tool - great 273victorinox swiss army pocket knife - red
super tinker multi tool - great 273
kershaw 1730ss zing stainless speedsafe assisted open flipper knife usedkershaw 1730ss zing stainless speedsafe assis
ted open flipper knife used
kershaw leek 1660 knife - silver (tip is missing)kershaw leek 1660 knife - silver (tip is missing)
case classic christmas tree whittler 73043  1/2 knifecase classic christmas tree whittler 73043  1/2 knife
case xx usa 3254 trapper knife tuquoise 4x 2 blade blue pocketcase xx usa 3254 trapper knife tuquoise 4x 2 blade blue
pocket
case xx usa 1999 6254 ss dark orange trapper knife
benchmade 273fe-2 mini adamas® tactical folding knife cpm-cruwearbenchmade 273fe-2 mini adamas® tactical folding knif
e cpm-cruwear
buck usa 305 x 2 blade stockman black handles knifebuck usa 305 x 2 blade stockman black handles knife
spyderco tenacious 8cr13mov folding pocket knife china black
```

Out[115]:
```
27593    None
62362    None
40037    None
48483    None
19128    None
30401    None
69637    None
1173     None
15768    None
73631    None
Name: data, dtype: object
```

In [116]:
```
# df_title['labels'] = (df_title['labels']/mean_price)
Y = df_title['labels'].values
```

In [117]:
```
df_train, df_test, Ytrain, Ytest = train_test_split(df_title['data'], Y, test_size=0.3)
```

In [118]:
```
X_val, X_test, Y_val, Y_test = train_test_split(df_test, Ytest, test_size=0.5)
```

In [130]:
```
voc_size = 25000
max_len = 30
embedding_features = 25
tokenizer = Tokenizer(num_words=voc_size, oov_token = '<OOV>')
tokenizer.fit_on_texts(df_train)
sequences_train = tokenizer.texts_to_sequences(df_train)
sequences_val = tokenizer.texts_to_sequences(X_val)
sequences_test = tokenizer.texts_to_sequences(X_test)
```

In [131]:
```
data_train = pad_sequences(sequences_train, maxlen=max_len, padding= 'post', truncating = 'post')
data_val = pad_sequences(sequences_val, maxlen=max_len, padding= 'post', truncating = 'post')
data_test = pad_sequences(sequences_test, maxlen=max_len, padding= 'post', truncating = 'post')
```

In [132]:
```
model = models.Sequential()
model.add(Embedding(voc_size, embedding_features, input_length = max_len))
# model.add(Dropout(0.3))
model.add(GRU(100))
model.add(Dense(62, activation = 'relu'))
# model.add(Dropout(0.3))
model.add(Dense(32, activation = 'relu'))
# model.add(Dropout(0.3))
model.add(Dense(1, activation = 'relu'))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 30, 25)            625000
_________________________________________________________________
gru_4 (GRU)                  (None, 100)               38100
_________________________________________________________________
dense_12 (Dense)             (None, 62)                6262
_________________________________________________________________
dense_13 (Dense)             (None, 32)                2016
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 33
=================================================================
Total params: 671,411
Trainable params: 671,411
Non-trainable params: 0
_________________________________________________________________
```

In [133]:

```
# Compile and fit
model.compile(
  loss='MSE',
  optimizer='adam',
  metrics=['mae']
)


print('Training model...')
r = model.fit(
  data_train,
  Ytrain,
  epochs=5,
  validation_data=(data_val, Y_val)
)
```

```
Training model...
Epoch 1/5
1672/1672 [==============================] - 24s 14ms/step - loss: 839.5955 - mae: 21.0601 - val_loss: 479.4761 - val
_mae: 15.7006
Epoch 2/5
1672/1672 [==============================] - 23s 14ms/step - loss: 424.7324 - mae: 14.5346 - val_loss: 439.2792 - val
_mae: 15.6913
Epoch 3/5
1672/1672 [==============================] - 23s 14ms/step - loss: 347.1382 - mae: 12.9299 - val_loss: 422.3552 - val
_mae: 14.5125
Epoch 4/5
1672/1672 [==============================] - 23s 14ms/step - loss: 306.3951 - mae: 12.1047 - val_loss: 392.9435 - val
_mae: 13.6199
Epoch 5/5
1672/1672 [==============================] - 23s 14ms/step - loss: 266.5142 - mae: 11.1126 - val_loss: 396.1306 - val
_mae: 13.6661
```

In [134]:

```
pred=model.predict(data_test)
```

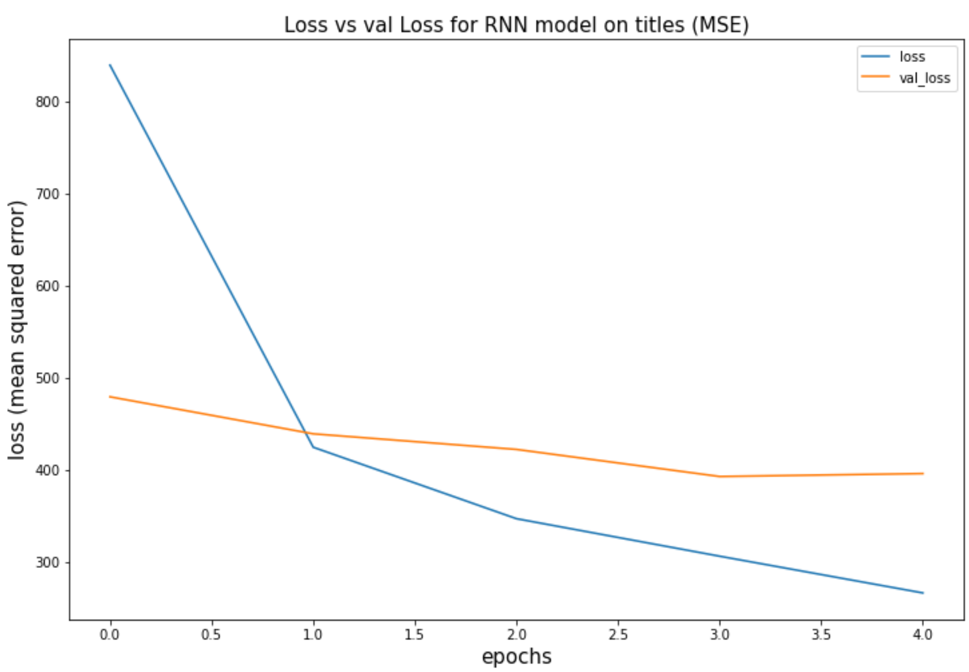
In [135]:

```
test_results = model.evaluate(data_test, Y_test)
```

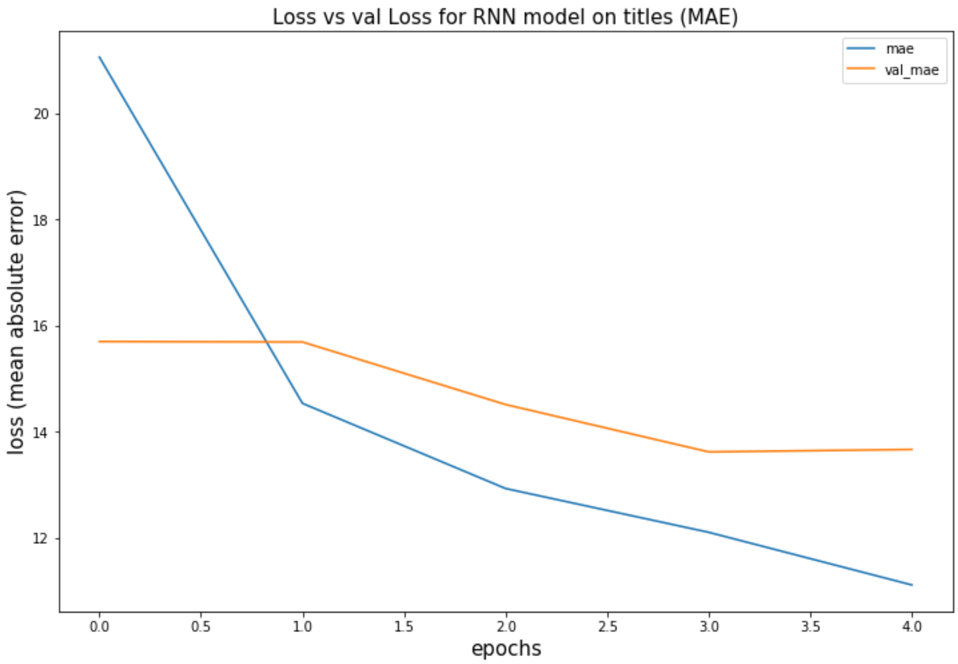
```
359/359 [==============================] - 1s 3ms/step - loss: 386.1020 - mae: 13.4815
```

In [136]:

```
fig = plt.subplots(figsize=(12,8))
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.title("Loss vs val Loss for RNN model on titles (MSE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean squared error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_GRU_MSE1.png')
```



In [137]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(r.history['mae'], label='mae')
plt.plot(r.history['val_mae'], label='val_mae')
plt.title("Loss vs val Loss for RNN model on titles (MAE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean absolute error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_GRU_MAE1.png')
```

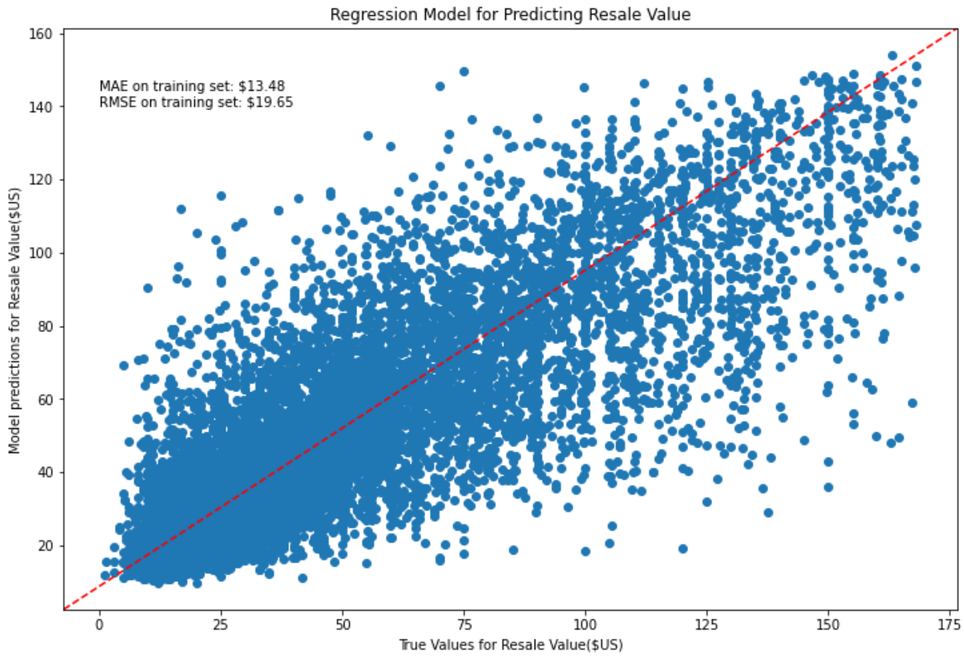


In [138]:
```
from sklearn.metrics import mean_absolute_error
```

In [139]:
```
test_mae = mean_absolute_error(Y_test, pred)
```

In [150]:
```
RMSE = np.sqrt(test_results[0])
```

In [151]:
```
string_score = f'\nMAE on training set: ${test_mae:.2f}'
string_score += f'\nRMSE on training set: ${RMSE:.2f}'
fig, ax = plt.subplots(figsize=(12, 8))
plt.scatter(Y_test, pred)
ax.plot([0, 1], [0, 1], transform=ax.transAxes, ls="--", c="red")
plt.text(0, 140, string_score)
plt.title('Regression Model for Predicting Resale Value')
plt.ylabel('Model predictions for Resale Value($US)')
plt.xlabel('True Values for Resale Value($US)')
plt.savefig('images/regression_GRU_relu1.png');
```



In [145]:
```
df_title['labels'].describe()
```

Out[145]:
```
count    76400.000000
mean        48.743928
std         34.865987
min          0.990000
25%         21.960000
50%         38.000000
75%         65.482500
max        168.330000
Name: labels, dtype: float64
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# Convert sentences to sequences
MAX_VOCAB_SIZE = 30000
tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE)
tokenizer.fit_on_texts(df_train)
sequences_train = tokenizer.texts_to_sequences(df_train)
sequences_val = tokenizer.texts_to_sequences(X_val)
sequences_test = tokenizer.texts_to_sequences(X_test)
```

In [152]:
```
# Convert sentences to sequences
MAX_VOCAB_SIZE = 25000
tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE)
tokenizer.fit_on_texts(df_train)
sequences_train = tokenizer.texts_to_sequences(df_train)
sequences_val = tokenizer.texts_to_sequences(X_val)
sequences_test = tokenizer.texts_to_sequences(X_test)
```

In [153]:
```
# get word -> integer mapping
word2idx = tokenizer.word_index
V = len(word2idx)
print('Found %s unique tokens.' % V)
```

```
Found 27823 unique tokens.
```

In [154]:
```
# pad sequences so that we get a N x T matrix
data_train = pad_sequences(sequences_train)
print('Shape of data train tensor:', data_train.shape)

# get sequence length
T = data_train.shape[1]
```

```
Shape of data train tensor: (53480, 42)
```

In [155]:
```
data_val = pad_sequences(sequences_val, maxlen=T)
print('Shape of data test tensor:', X_val.shape)
```

```
Shape of data test tensor: (11460,)
```

In [156]:
```
data_test = pad_sequences(sequences_test, maxlen=T)
print('Shape of data test tensor:', X_test.shape)
```

```
Shape of data test tensor: (11460,)
```

In [ ]:
```
model.add(Embedding(voc_size, embedding_features, input_length = max_len))
# model.add(Dropout(0.3))
model.add(GRU(100))
model.add(Dense(62, activation = 'relu'))
# model.add(Dropout(0.3))
model.add(Dense(32, activation = 'relu'))
# model.add(Dropout(0.3))
model.add(Dense(1, activation = 'relu'))
model.summary()
```

In [161]:
```
# Create the RNN model

# We get to choose embedding dimensionality
D = 30

# Hidden state dimensionality
M = 25


i = Input(shape=(T,))
x = Embedding(V + 1, D)(i)
x = LSTM(M, return_sequences=True)(x)
x = GlobalMaxPooling1D()(x)
x = Dense(62, activation='relu')(x)
x = Dense(32, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(1, activation='relu')(x)

model = Model(i, x)
```

In [162]:
```
# Compile and fit
model.compile(
  loss='MSE',
  optimizer='adam',
  metrics=['mae']
)


print('Training model...')
r = model.fit(
  data_train,
  Ytrain,
  epochs=5,
  validation_data=(data_val, Y_val)
)
```

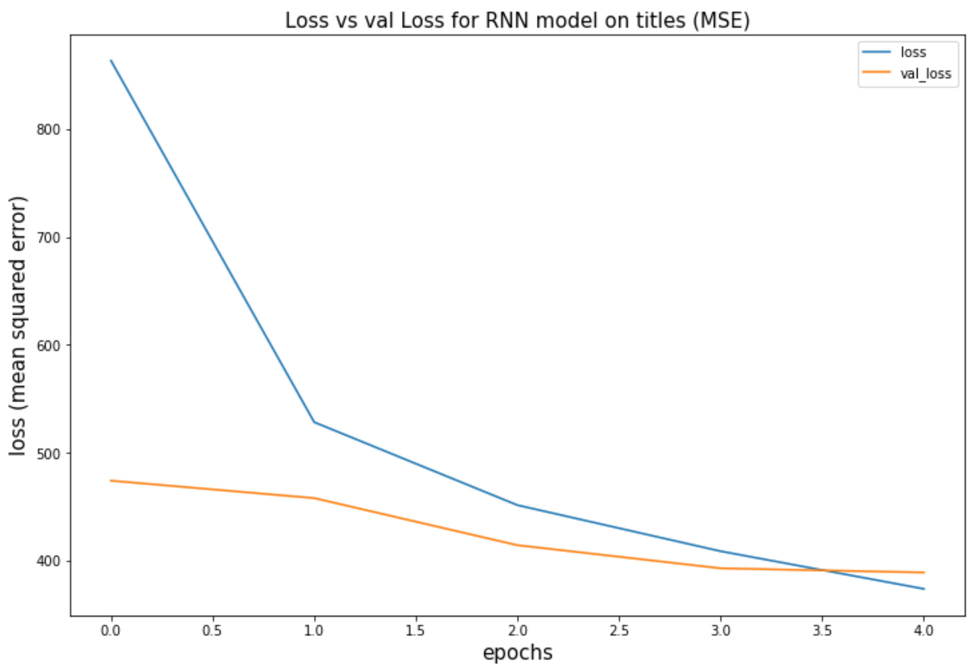
```
Training model...
Epoch 1/5
1672/1672 [==============================] - 24s 14ms/step - loss: 863.3234 - mae: 20.6806 - val_loss: 474.2351 - val
_mae: 14.9007
Epoch 2/5
1672/1672 [==============================] - 23s 14ms/step - loss: 528.4490 - mae: 15.9809 - val_loss: 458.1084 - val
_mae: 14.2202
Epoch 3/5
1672/1672 [==============================] - 24s 14ms/step - loss: 451.5854 - mae: 14.6465 - val_loss: 414.4811 - val
_mae: 13.6043
Epoch 4/5
1672/1672 [==============================] - 24s 14ms/step - loss: 408.9058 - mae: 13.8346 - val_loss: 393.0957 - val
_mae: 13.2682
Epoch 5/5
1672/1672 [==============================] - 24s 14ms/step - loss: 374.0296 - mae: 13.1777 - val_loss: 389.2262 - val
_mae: 13.1884
```

In [163]:
```
pred=model.predict(data_test)
```

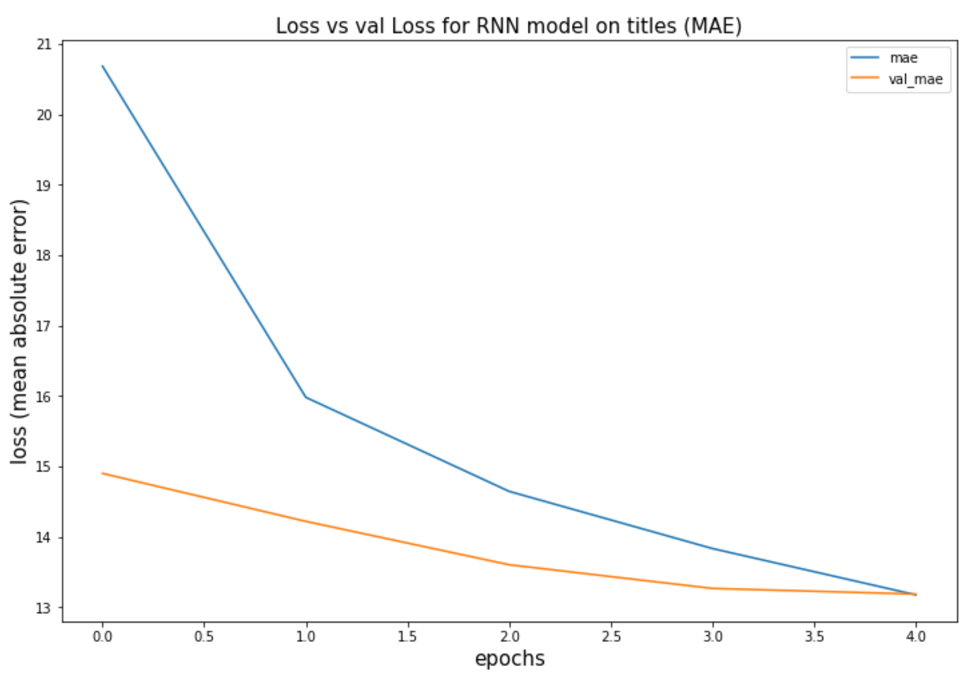
In [164]:
```
test_results = model.evaluate(data_test, Y_test)
```

```
359/359 [==============================] - 1s 2ms/step - loss: 376.6573 - mae: 13.0322
```

In [165]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.title("Loss vs val Loss for RNN model on titles (MSE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean squared error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_LSTM_MSE5.png')
```



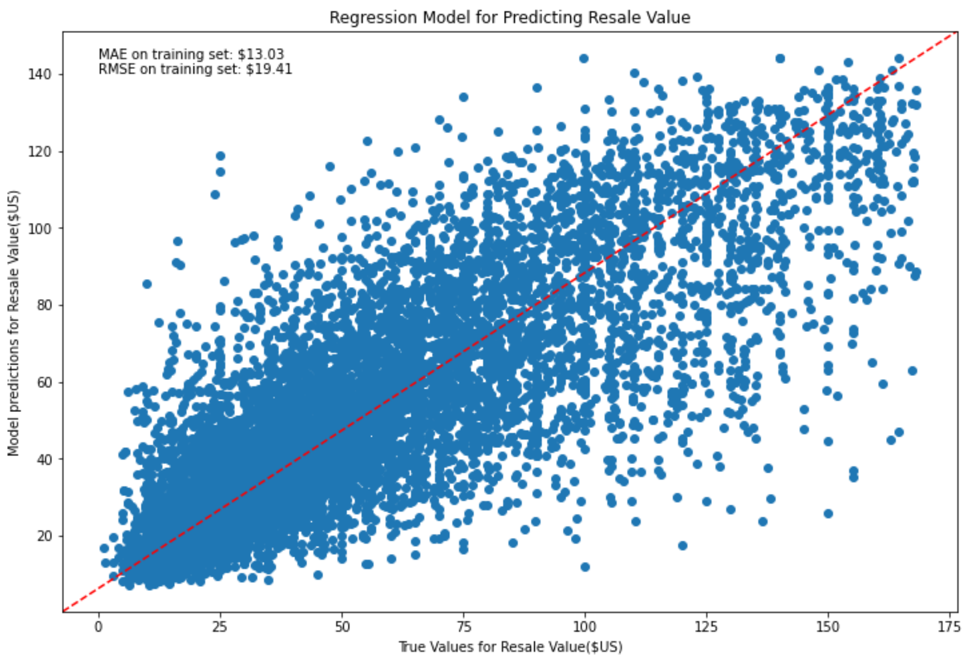
In [166]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(r.history['mae'], label='mae')
plt.plot(r.history['val_mae'], label='val_mae')
plt.title("Loss vs val Loss for RNN model on titles (MAE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean absolute error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_LSTM_MAE5.png')
```



In [167]:
```
test_mae = mean_absolute_error(Y_test, pred)
```

In [168]:
```
RMSE = np.sqrt(test_results[0])
```

In [169]:
```
string_score = f'\nMAE on training set: ${test_mae:.2f}'
string_score += f'\nRMSE on training set: ${RMSE:.2f}'
fig, ax = plt.subplots(figsize=(12, 8))
plt.scatter(Y_test, pred)
ax.plot([0, 1], [0, 1], transform=ax.transAxes, ls="--", c="red")
plt.text(0, 140, string_score)
plt.title('Regression Model for Predicting Resale Value')
plt.ylabel('Model predictions for Resale Value($US)')
plt.xlabel('True Values for Resale Value($US)')
plt.savefig('images/regression_LSTM_relu5.png');
```



In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# get word -> integer mapping
word2idx = tokenizer.word_index
V = len(word2idx)
print('Found %s unique tokens.' % V)
```

In [ ]:
```
# pad sequences so that we get a N x T matrix
data_train = pad_sequences(sequences_train)
print('Shape of data train tensor:', data_train.shape)

# get sequence length
T = data_train.shape[1]
```

In [ ]:
```
data_val = pad_sequences(sequences_val, maxlen=T)
print('Shape of data test tensor:', X_val.shape)
```

In [ ]:
```
data_test = pad_sequences(sequences_test, maxlen=T)
print('Shape of data test tensor:', X_test.shape)
```

In [ ]:
```
# Create the RNN model

# We get to choose embedding dimensionality
D = 20

# Hidden state dimensionality
M = 15


i = Input(shape=(T,))
x = Embedding(V + 1, D)(i)
x = LSTM(M, return_sequences=True)(x)
x = GlobalMaxPooling1D()(x)
x = Dense(1, activation='relu')(x)

model = Model(i, x)
```

In [ ]:
```
# Compile and fit
model.compile(
  loss='MSE',
  optimizer='adam',
  metrics=['mae']
)


print('Training model...')
r = model.fit(
  data_train,
  Ytrain,
  epochs=5,
  validation_data=(data_val, Y_val)
)
```

In [ ]:
```
model.summary()
```

In [ ]:
```
pred=model.predict(data_test)
```

In [ ]:
```
test_results = model.evaluate(data_test, Y_test)
```

In [ ]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.title("Loss vs val Loss for RNN model on titles (MSE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean squared error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_titles_MSE1.png')
```

In [ ]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(r.history['mae'], label='mae')
plt.plot(r.history['val_mae'], label='val_mae')
plt.title("Loss vs val Loss for RNN model on titles (MAE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean absolute error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_titles_MAE1.png')
```

In [ ]:
```
test_mae = mean_absolute_error(Y_test, pred)
```

In [ ]:
```

```

In [ ]:
```
string_score = f'\nMAE on training set: ${test_mae:.2f}'

fig, ax = plt.subplots(figsize=(12, 8))
plt.scatter(Y_test, pred)
ax.plot([0, 1], [0, 1], transform=ax.transAxes, ls="--", c="red")
plt.text(0, 1, string_score)
plt.title('Regression Model for Predicting Resale Value')
plt.ylabel('Model predictions for Resale Value($US)')
plt.xlabel('True Values for Resale Value($US)')
plt.show();
```

In [ ]:
```
# Create the RNN model

# We get to choose embedding dimensionality
D = 25

# Hidden state dimensionality
M = 20


i = Input(shape=(T,))
x = Embedding(V + 1, D)(i)
x = LSTM(M, return_sequences=True)(x)
x = GlobalMaxPooling1D()(x)
x = Dense(1, activation='relu')(x)

model = Model(i, x)
```

In [ ]:
```
# Compile and fit
model.compile(
  loss='MSE',
  optimizer='adam',
  metrics=['mae']
)


print('Training model...')
r = model.fit(
  data_train,
  Ytrain,
  epochs=5,
  validation_data=(data_val, Y_val)
)
```

In [ ]:
```
model.summary()
```

In [ ]:
```
pred=model.predict(data_test)
```

In [ ]:
```
test_results = model.evaluate(data_test, Y_test)
```

In [ ]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.title("Loss vs val Loss for RNN model on titles (MSE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean squared error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_titles_MSE3.png')
```

In [ ]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(r.history['mae'], label='mae')
plt.plot(r.history['val_mae'], label='val_mae')
plt.title("Loss vs val Loss for RNN model on titles (MAE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean absolute error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_titles_MAE3.png')
```

In [ ]:
```
test_mae = mean_absolute_error(Y_test, pred)
```

In [ ]:
```

```

In [ ]:
```
string_score = f'\nMAE on training set: ${test_mae:.2f}'

fig, ax = plt.subplots(figsize=(12, 8))
plt.scatter(Y_test, pred)
ax.plot([0, 1], [0, 1], transform=ax.transAxes, ls="--", c="red")
plt.text(0, 105, string_score)
plt.title('Regression Model for Predicting Resale Value')
plt.ylabel('Model predictions for Resale Value($US)')
plt.xlabel('True Values for Resale Value($US)')
plt.show();
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
# Create the CNN model

# We get to choose embedding dimensionality
D = 20



i = Input(shape=(T,))
x = Embedding(V + 1, D)(i)
x = Conv1D(32, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)
x = Conv1D(64, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)
x = Conv1D(128, 3, activation='relu')(x)
x = GlobalMaxPooling1D()(x)
x = Dense(1, activation='linear')(x)

model = Model(i, x)
```

In [ ]:
```
# Compile and fit
model.compile(
  loss='MSE',
  optimizer='adam',
  metrics=['MSE']
)


print('Training model...')
r = model.fit(
  data_train,
  Ytrain,
  epochs=5,
  validation_data=(data_test, Ytest)
)
```

In [ ]:
```
# Plot loss per iteration
import matplotlib.pyplot as plt
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.legend();
```

In [ ]:
```
# Plot accuracy per iteration
plt.plot(r.history['MSE'], label='MSE')
plt.plot(r.history['val_MSE'], label='val_MSE')
plt.legend();
```

In [ ]:

In [ ]:

CNN using images as input

In [ ]:

In [ ]:
```
df_imgs = df.drop(['title', 'url',
                   'date_sold', 'profit',
                   'ROI', 'brand', 'cost',
                   'pictureURLLarge'],
                     axis=1).copy()
```

In [ ]:
```
df_imgs.dropna(subset=['Image'], inplace=True)
```

In [ ]:
```
df_imgs.reset_index(drop=True, inplace=True)
```

In [ ]:
```
df_imgs['file_index'] = df_imgs.index.values
df_imgs['file_index'] = df_imgs['file_index'].astype(str)
```

In [ ]:
```
df_imgs['filename'] = df_imgs['file_index'] + '.jpg'
```

In [ ]:
```
# Identify Image Resolutions

# # Import Packages
# import pandas as pd
# import matplotlib.pyplot  as plt
# from PIL import Image
# from pathlib import Path
# import imagesize
# import numpy as np

# # Get the Image Resolutions
# imgs = [img.name for img in Path(root).iterdir() if img.suffix == ".jpg"]
# img_meta = {}
# for f in imgs: img_meta[str(f)] = imagesize.get(root+f)

# # Convert it to Dataframe and compute aspect ratio
# img_meta_df = pd.DataFrame.from_dict([img_meta]).T.reset_index().set_axis(['FileName', 'Size'], axis='columns', i
# img_meta_df[["Width", "Height"]] = pd.DataFrame(img_meta_df["Size"].tolist(), index=img_meta_df.index)
# img_meta_df["Aspect Ratio"] = round(img_meta_df["Width"] / img_meta_df["Height"], 2)

# print(f'Total Nr of Images in the dataset: {len(img_meta_df)}')
# img_meta_df.head()



# # Visualize Image Resolutions

# fig = plt.figure(figsize=(8, 8))
# ax = fig.add_subplot(111)
# points = ax.scatter(img_meta_df.Width, img_meta_df.Height, color='blue', alpha=0.5, s=img_meta_df["Aspect Ratio"]
# ax.set_title("Image Resolution")
# ax.set_xlabel("Width", size=14)
# ax.set_ylabel("Height", size=14);
```

In [ ]:
```
def download(row):
    filename = row.filepath

# create folder if it doesn't exist
#     os.makedirs(os.path.dirname(filename), exist_ok=True)

    url = row.Image
#     print(f"Downloading {url} to {filename}")

    try:
        r = requests.get(url, allow_redirects=True)
        with open(filename, 'wb') as f:
            f.write(r.content)
    except:
        print(f'{filename} error')
```

In [ ]:
```
root_folder = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images/'
df_imgs['filepath'] = root_folder + df_imgs['filename']
```

In [ ]:
```
df_imgs['filepath'].sample(2).apply(print)
```

In [ ]:
```
df_imgs.apply(download, axis=1)
```

In [ ]:
```
removed_files = []
pathway = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images/'
for filename in os.listdir(pathway):
    if filename.endswith('.jpg'):
        try:
            img = Image.open(pathway + filename)  # open the image file
            img.verify()  # verify that it is, in fact an image
        except (IOError, SyntaxError) as e:
            print(filename)
            removed_files.append(filename)
            os.remove(pathway + filename)
```

In [ ]:
```
to_drop = df_imgs.loc[df_imgs['filename'].isin(removed_files)].index.to_list()
```

In [ ]:
```
df_imgs.drop(to_drop, inplace=True)
```

In [ ]:
```
 img_list = os.listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images/'
```

In [ ]:
```
img_df = df_imgs.loc[df_imgs['filename'].isin(img_list)].copy()
```

In [ ]:
```
img_df.reset_index(drop=True, inplace=True)
```

In [ ]:
```
img_df.info()
```

In [ ]:
```
img_df.rename({'Image': 'data',
               'converted_price': 'labels'},
                axis=1, inplace=True)
```

In [ ]:
```
median_price = img_df['labels'].median()
median_price
```

In [ ]:
```
img_df['labels'] = (img_df['labels']/median_price)
```

In [ ]:
```
Y = img_df['labels'].values
```

In [ ]:
```
df_train, df_test, Ytrain, Ytest = train_test_split(img_df, Y, test_size=0.20)
```

In [ ]:
```
datagen=ImageDataGenerator(rescale=1./255.,validation_split=0.20)
```

In [ ]:
```
train_generator=datagen.flow_from_dataframe(
dataframe=df_train,
directory= None,
x_col="filepath",
y_col="labels",
subset="training",
batch_size=100,
seed=55,
shuffle=True,
class_mode="raw")

valid_generator=datagen.flow_from_dataframe(
dataframe=df_train,
directory=None,
x_col="filepath",
y_col="labels",
subset="validation",
batch_size=100,
seed=55,
shuffle=True,
class_mode="raw")

test_datagen=ImageDataGenerator(rescale=1./255.)
test_generator=test_datagen.flow_from_dataframe(
dataframe=df_test,
directory=None,
x_col="filepath",
y_col="labels",
batch_size=100,
seed=55,
shuffle=False,
class_mode="raw")
```

In [ ]:
```
model = models.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu',
                        input_shape=(256 ,256,  3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(16, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu',
                        input_shape=(256 ,256,  3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='linear'))

model.compile(loss='MSE',
              optimizer='Adam',
               metrics=['mae', 'mse'])
```

In [ ]:
```
summary = model.fit(train_generator, epochs=3, validation_data=valid_generator)
```

In [ ]:
```
model.evaluate(valid_generator)
```

In [ ]:
```
test_generator.reset()
pred=model.predict(test_generator,verbose=1)
```

In [ ]:
```
test_results = model.evaluate(test_generator)
```

In [ ]:
```
fig = plt.figure(figsize=(12,8))
plt.plot(summary.history['loss'])
plt.plot(summary.history['val_loss'])
plt.plot
plt.title('model loss')
plt.ylabel('loss(mean absolute error)')
plt.xlabel('epoch')
plt.legend(['train_loss', 'val_loss'], loc='upper right')
plt.show();
```

In [ ]:

```
# # define two sets of inputs
# inputA = Input(shape=(32,))
# inputB = Input(shape=(128,))
# # the first branch operates on the first input
# x = Dense(8, activation="relu")(inputA)
# x = Dense(4, activation="relu")(x)
# x = Model(inputs=inputA, outputs=x)
# # the second branch opreates on the second input
# y = Dense(64, activation="relu")(inputB)
# y = Dense(32, activation="relu")(y)
# y = Dense(4, activation="relu")(y)
# y = Model(inputs=inputB, outputs=y)
# # combine the output of the two branches
# combined = concatenate([x.output, y.output])
# # apply a FC layer and then a regression prediction on the
# # combined outputs
# z = Dense(2, activation="relu")(combined)
# z = Dense(1, activation="linear")(z)
# # our model will accept the inputs of the two branches and
# # then output a single value
# model = Model(inputs=[x.input, y.input], outputs=z)
```

In [ ]:

```

```