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]:
```
import numpy
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
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
```

In [2]:
```
from sklearn.model_selection import train_test_split
```

Function Definition

In [3]:
```
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'], dict) else x

    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 [4]:
```
#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
               }
```

In [5]:
```
used_listed_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11855 entries, 0 to 11854
Data columns (total 35 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   itemId                          11855 non-null  int64  
 1   title                           11855 non-null  object 
 2   galleryURL                      11854 non-null  object 
 3   viewItemURL                     11855 non-null  object 
 4   autoPay                         11855 non-null  bool   
 5   postalCode                      11576 non-null  object 
 6   returnsAccepted                 11855 non-null  bool   
 7   condition                       11854 non-null  float64
 8   topRatedListing                 11855 non-null  bool   
 9   pictureURLLarge                 11208 non-null  object 
 10  pictureURLSuperSize             11159 non-null  object 
 11  shipping_cost                   11855 non-null  float64
 12  price_in_US                     11855 non-null  float64
 13  converted_price                 11855 non-null  float64
 14  brand                           11855 non-null  object 
 15  cost                            11855 non-null  float64
 16  profit                          11855 non-null  float64
 17  ROI                             11855 non-null  float64
 18  PictureURL                      11854 non-null  object 
 19  Location                        11854 non-null  object 
 20  Country                         11855 non-null  object 
 21  Blade Material                  6625 non-null   object 
 22  Model                           9373 non-null   object 
 23  Opening Mechanism               7100 non-null   object 
 24  Number of Blades                7865 non-null   object 
 25  Handle Material                 7529 non-null   object 
 26  Blade Type                      5336 non-null   object 
 27  Color                           8262 non-null   object 
 28  Type                            9147 non-null   object 
 29  Country/Region of Manufacture   6568 non-null   object 
 30  Lock Type                       5482 non-null   object 
 31  Blade Edge                      6155 non-null   object 
 32  Dexterity                       4383 non-null   object 
 33  Original/Reproduction           4879 non-null   object 
 34  Blade Range                     4403 non-null   object 
dtypes: bool(3), float64(7), int64(1), object(24)
memory usage: 2.9+ MB
```

Prepare Data

In [6]:
```
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 [7]:
```
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 [8]:
```
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()


df1 = pd.concat([sold_knives,used_listed_df]).copy()
df1['Image'].fillna(df1['pictureURLLarge'], inplace=True)

df = apply_iqr_filter(df1).copy()
df.reset_index(drop=True, inplace=True)
```

Neural network with "title" column as input

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

df['title'] = df['title'].str.replace("'", "")

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

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

In [ ]:
```
df_title['labels'].mean()
```

In [ ]:
```
from sklearn import preprocessing
```

In [ ]:
```
# Y/df_title['labels'].mean()
```

In [ ]:
```
y = df_title[["labels"]]
X = df_title.drop("labels", axis=1)
```

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

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

In [ ]:
```
scaler = preprocessing.MinMaxScaler()
```

In [ ]:
```
Ytrain_scaled = scaler.fit_transform(Ytrain)
Y_val_scaled = scaler.transform(Y_val)
Y_test_scaled = scaler.transform(Y_test)
```

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

In [ ]:
```
Y_val_scaled.shape
```

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

#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='linear')(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_scaled,
  epochs=5,
  validation_data=(data_val, Y_val_scaled)
)
```

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

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

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

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_MSE2.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_MAE2.png')
```

In [ ]:
```
y_test_hat = scaler.inverse_transform(pred)
Y_test = scaler.inverse_transform(Y_test_scaled)
```

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

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

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

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();
```

CNN using images as input

In [9]:
```
df_imgs = df.loc[:, ['Image', 'converted_price']]


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

In [10]:
```
df_imgs.dropna(subset=['data'], inplace=True)
```

In [11]:
```
df_imgs.reset_index(drop=True, inplace=True)
```

In [12]:
```
df_imgs['file_index'] = df_imgs.index.values
df_imgs['file_index'] = df_imgs['file_index'].astype(str)

df_imgs['filename'] = df_imgs['file_index'] + '.jpg'

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

In [13]:
```
from PIL import Image
```

In [14]:
```
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

In [15]:
```
def download(row):
    filename = row.filepath

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

    url = row.data
#     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 [16]:
```
# df_imgs.apply(download, axis=1)
```

In [17]:
```
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 [18]:
```
to_drop = df_imgs.loc[df_imgs['filename'].isin(removed_files)].index.to_list()
```

In [19]:
```
df_imgs.drop(to_drop, inplace=True)
```

In [20]:
```
 img_list = os.listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images/')
```

In [21]:
```
img_df = df_imgs.loc[df_imgs['filename'].isin(img_list)].copy()
```

In [22]:
```
img_df.reset_index(drop=True, inplace=True)
```

In [23]:
```
img_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75749 entries, 0 to 75748
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   data        75749 non-null  object 
 1   labels      75749 non-null  float64
 2   file_index  75749 non-null  object 
 3   filename    75749 non-null  object 
 4   filepath    75749 non-null  object 
dtypes: float64(1), object(4)
memory usage: 2.9+ MB
```

In [44]:
```
datagen=ImageDataGenerator(rescale=1./255.,validation_split=0.25)
```

In [45]:
```
df_train, df_test = train_test_split(img_df, test_size=0.20, random_state=114)
```

In [46]:
```
df_train['labels']
```

Out[46]:
```
42500    15.50
65066    16.00
24302    48.20
32105    49.00
51556    15.00
         ...
32255    49.95
47099    17.76
51794    42.00
5926     17.99
64275     9.00
Name: labels, Length: 60599, dtype: float64
```

In [47]:
```
from sklearn import preprocessing
```

In [48]:
```
scaler = preprocessing.MinMaxScaler()
```

In [49]:
```
df_train['labels'] = scaler.fit_transform(df_train[['labels']])
df_test['labels'] = scaler.transform(df_test[['labels']])
```

In [50]:
```
train_generator=datagen.flow_from_dataframe(
dataframe=df_train,
directory= None,
target_size=(500, 500),
color_mode='grayscale',
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,
target_size=(500, 500),
color_mode='grayscale',
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,
target_size=(500, 500),
color_mode='grayscale',
x_col="filepath",
y_col="labels",
batch_size=100,
seed=55,
shuffle=False,
class_mode="raw")
```

```
Found 45450 validated image filenames.
Found 15149 validated image filenames.
Found 15150 validated image filenames.
```

In [51]:
```
train_generator.image_shape
```

Out[51]:
```
(500, 500, 1)
```

In [54]:
```
model = models.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu',
                        input_shape=(500 ,500,  1)))
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'))
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(128, activation='relu'))
model.add(Dense(1, activation='linear'))

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

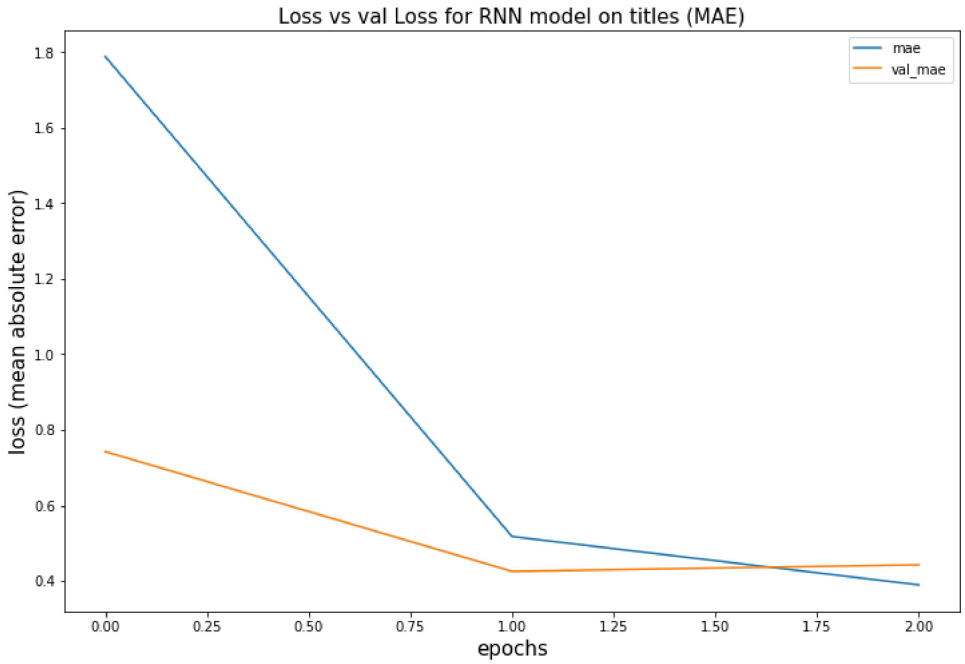
In [55]:
```
summary = model.fit(train_generator, epochs=3, validation_data=valid_generator)
```

```
Epoch 1/3
455/455 [==============================] - 4831s 11s/step - loss: 60.6494 - mae: 1.7879 - mse: 60.6494 - val_loss: 1.0904 - val
_mae: 0.7424 - val_mse: 1.0904
Epoch 2/3
455/455 [==============================] - 4859s 11s/step - loss: 0.4639 - mae: 0.5176 - mse: 0.4639 - val_loss: 0.2906 - val_m
ae: 0.4254 - val_mse: 0.2906
Epoch 3/3
455/455 [==============================] - 4920s 11s/step - loss: 0.2541 - mae: 0.3896 - mse: 0.2541 - val_loss: 0.3217 - val_m
ae: 0.4428 - val_mse: 0.3217
```

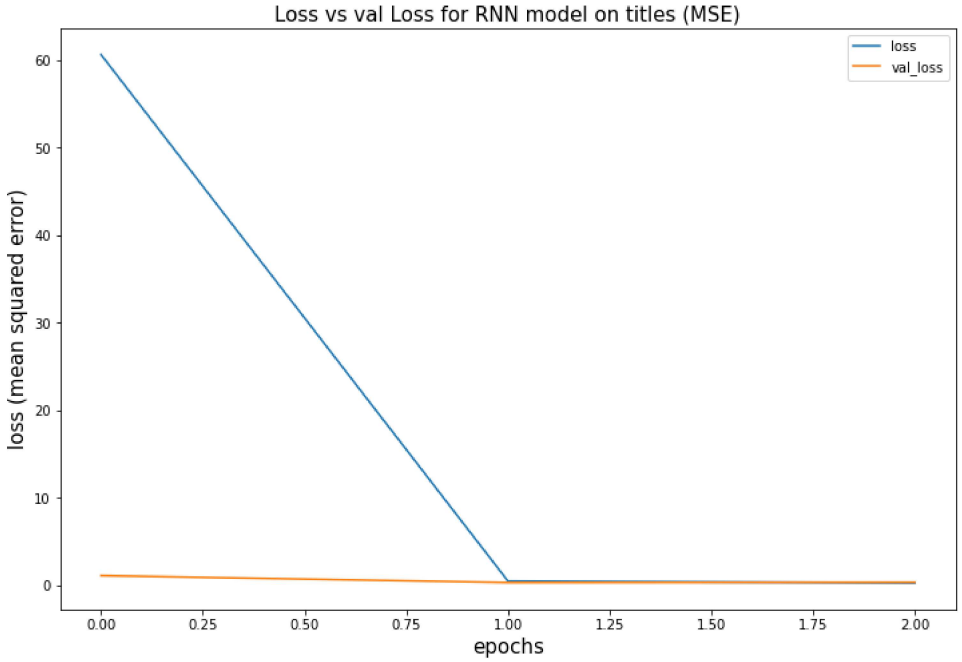
In [ ]:
```
# test_results = model.evaluate(test_generator)
```

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

In [59]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(summary.history['mae'], label='mae')
plt.plot(summary.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/CNN_images_MAE2.png')
```



In [60]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(summary.history['loss'], label='loss')
plt.plot(summary.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/CNN_images_MSE2.png')
```



In [84]:
```
model.save('cnn_grayscale_images2.h5')
```

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 [ ]:
```
```

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', inplace=False)
# 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"]*100, picker=T
# ax.set_title("Image Resolution")
# ax.set_xlabel("Width", size=14)
# ax.set_ylabel("Height", size=14);
```

In [ ]:
```
```

In [109]:
```
df_train, df_test = train_test_split(img_df, test_size=0.20, random_state=114)
```

In [110]:
```
df_train['labels']
```

Out[110]:
```
42500    15.50
65066    16.00
24302    48.20
32105    49.00
51556    15.00
         ...
32255    49.95
47099    17.76
51794    42.00
5926     17.99
64275     9.00
Name: labels, Length: 60599, dtype: float64
```

In [111]:
```
mean_price = df_train['labels'].mean()
```

In [113]:
```
df_train['labels'] = df_train['labels']/mean_price
df_test['labels'] = df_test['labels']/mean_price
```

In [116]:
```
train_generator=datagen.flow_from_dataframe(
dataframe=df_train,
directory= None,
target_size=(500, 500),
color_mode='grayscale',
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,
target_size=(500, 500),
color_mode='grayscale',
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,
target_size=(500, 500),
color_mode='grayscale',
x_col="filepath",
y_col="labels",
batch_size=100,
seed=55,
shuffle=False,
class_mode="raw")
```

```
Found 45450 validated image filenames.
Found 15149 validated image filenames.
Found 15150 validated image filenames.
```

In [97]:
```
train_generator.image_shape
```

Out[97]:
```
(500, 500, 1)
```

In [98]:
```
model = models.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu',
                        input_shape=(500 ,500,  1)))
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'))
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(128, activation='relu'))
model.add(Dense(1, activation='linear'))

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

In [99]:
```
summary = model.fit(train_generator, epochs=3, validation_data=valid_generator)
```

```
Epoch 1/3
455/455 [==============================] - 12586s 28s/step - loss: 176.6752 - mae: 4.6335 - mse: 176.6752 - val_loss: 1.9242 -
val_mae: 1.0726 - val_mse: 1.9242
Epoch 2/3
455/455 [==============================] - 12758s 28s/step - loss: 2.6480 - mae: 1.1848 - mse: 2.6480 - val_loss: 1.9837 - val_
mae: 1.0297 - val_mse: 1.9837
Epoch 3/3
455/455 [==============================] - 12886s 28s/step - loss: 1.2009 - mae: 0.7933 - mse: 1.2009 - val_loss: 0.6050 - val_
mae: 0.5674 - val_mse: 0.6050
```

In [118]:
```
y_test = test_generator.labels
y_test_rescaled = y_test*mean_price
```

In [121]:
```
test_results = model.evaluate(test_generator)
```

```
152/152 [==============================] - 678s 4s/step - loss: 0.5924 - mae: 0.5658 - mse: 0.5924
```

In [122]:
```
test_generator.reset()
pred=model.predict(test_generator,verbose=1)
```

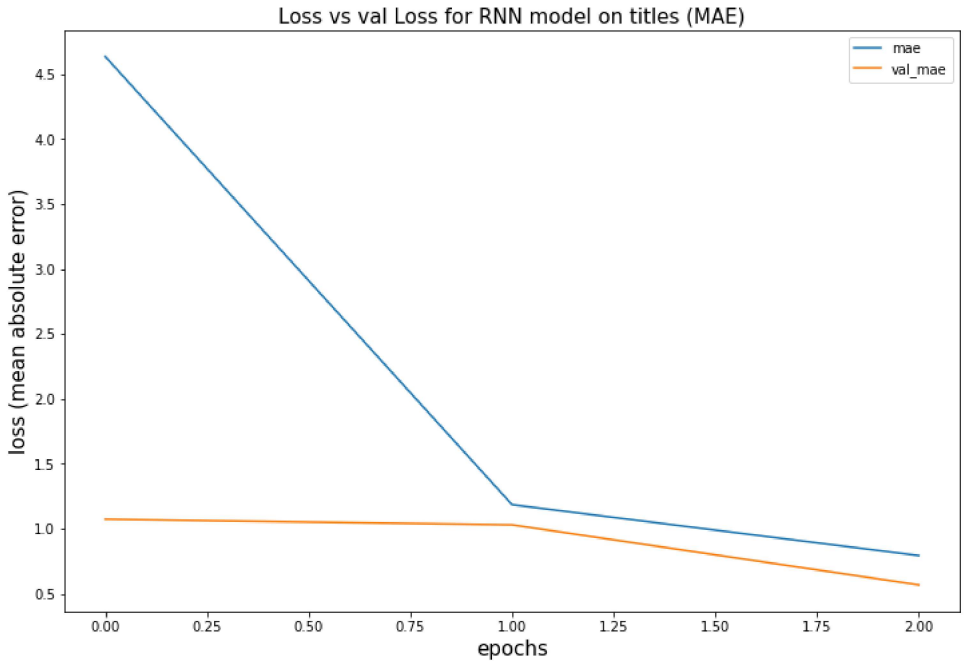
```
152/152 [==============================] - 670s 4s/step
```

In [123]:
```
y_hat_rescaled = pred*mean_price
```

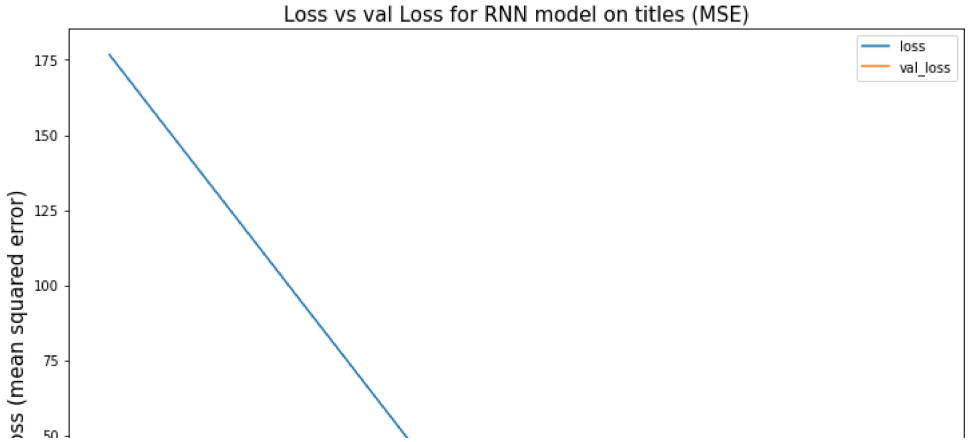
In [124]:
```
test_mae = mean_absolute_error(y_test_rescaled,y_hat_rescaled)
test_mae
```

Out[124]:
```
27.564040101055262
```

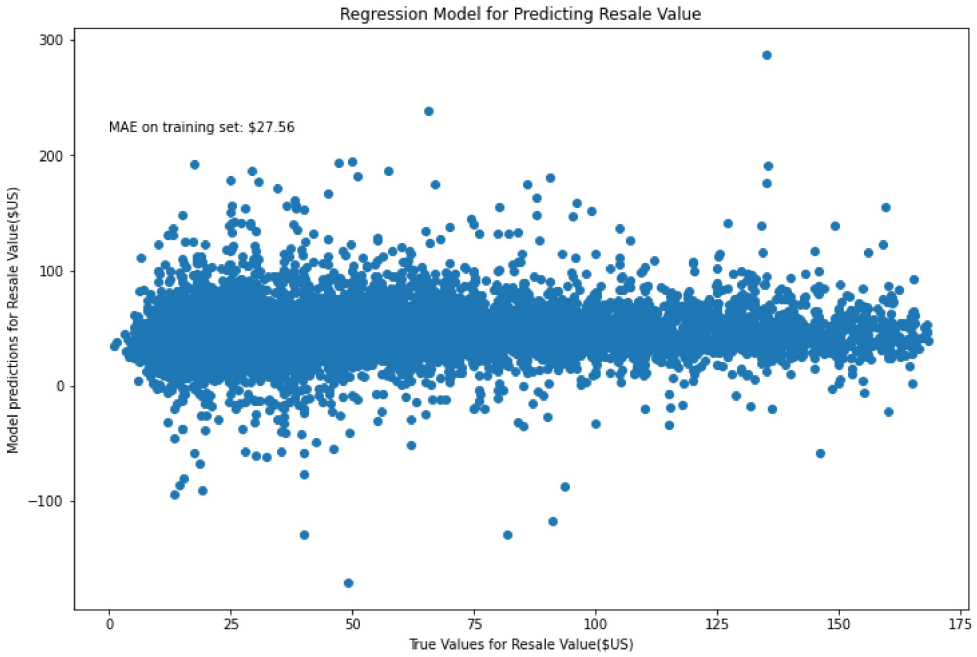
In [125]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(summary.history['mae'], label='mae')
plt.plot(summary.history['val_mae'], label='val_mae')
plt.title("Loss vs val Loss for RNN model on titles (MAE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean absolute error)", fontsize=15)
plt.legend()
plt.savefig('images/CNN_images_MAE3.png');
```



In [126]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(summary.history['loss'], label='loss')
plt.plot(summary.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/CNN_images_MSE3.png');
```



In [135]:
```
string_score = f'\nMAE on training set: ${test_mae:.2f}'
fig, ax = plt.subplots(figsize=(12, 8))
plt.scatter(y_test_rescaled,y_hat_rescaled)
# ax.plot([0, 1], [0, 1], transform=ax.transAxes, ls="--", c="red")
plt.text(0,220, string_score)
plt.title('Regression Model for Predicting Resale Value')
plt.ylabel('Model predictions for Resale Value($US)')
plt.xlabel('True Values for Resale Value($US)')
plt.savefig('images/CNN_images_regress_plot3.png')
plt.show();
```



In [84]:
```
model.save('cnn_grayscale_images3.h5')
```

In [ ]:
```
fig, ax = plt.subplots(figsize=(12, 8))
plt.scatter(y_t
```

In [137]:
```
df_train['labels'].describe()
```

Out[137]:
```
count    60599.000000
mean         1.000000
std          0.715170
min          0.020322
25%          0.450568
50%          0.779821
75%          1.340414
max          3.455312
Name: labels, dtype: float64
```

In [139]:
```
from sklearn.preprocessing import StandardScaler
```

In [146]:
```
datagen=ImageDataGenerator(rescale=1./255.,validation_split=0.25)
```

In [147]:
```
df_train, df_test = train_test_split(img_df, test_size=0.20, random_state=114)
```

In [149]:
```
train_generator=datagen.flow_from_dataframe(
dataframe=df_train,
directory= None,
target_size=(500, 500),
color_mode='grayscale',
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,
target_size=(500, 500),
color_mode='grayscale',
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,
target_size=(500, 500),
color_mode='grayscale',
x_col="filepath",
y_col="labels",
batch_size=100,
seed=55,
shuffle=False,
class_mode="raw")
```

```
Found 45450 validated image filenames.
Found 15149 validated image filenames.
Found 15150 validated image filenames.
```

In [150]:
```
train_generator.labels
```

Out[150]:
```
array([24.5 , 95.  , 37.25, ..., 42.  , 17.99,  9.  ])
```

In [151]:
```
model = models.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu',
                        input_shape=(500 ,500,  1)))
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'))
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(128, activation='relu'))
model.add(Dense(1, activation='relu'))

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

In [152]:
```
summary = model.fit(train_generator, epochs=3, validation_data=valid_generator)
```

```
Epoch 1/3
455/455 [==============================] - 4971s 11s/step - loss: 1389.6252 - mae: 27.8123 - mse: 1389.6252 - val_loss: 1396.26
94 - val_mae: 25.6290 - val_mse: 1396.2694
Epoch 2/3
455/455 [==============================] - 5027s 11s/step - loss: 1057.1609 - mae: 24.8306 - mse: 1057.1609 - val_loss: 1124.09
40 - val_mae: 25.8094 - val_mse: 1124.0940
Epoch 3/3
455/455 [==============================] - 5042s 11s/step - loss: 935.1313 - mae: 23.2094 - mse: 935.1313 - val_loss: 1139.5782
- val_mae: 25.9181 - val_mse: 1139.5782
```

In [153]:
```
test_results = model.evaluate(test_generator)
```

```
152/152 [==============================] - 339s 2s/step - loss: 1138.2579 - mae: 25.8766 - mse: 1138.2579
```

In [154]:
```
test_generator.reset()
pred=model.predict(test_generator,verbose=1)
```

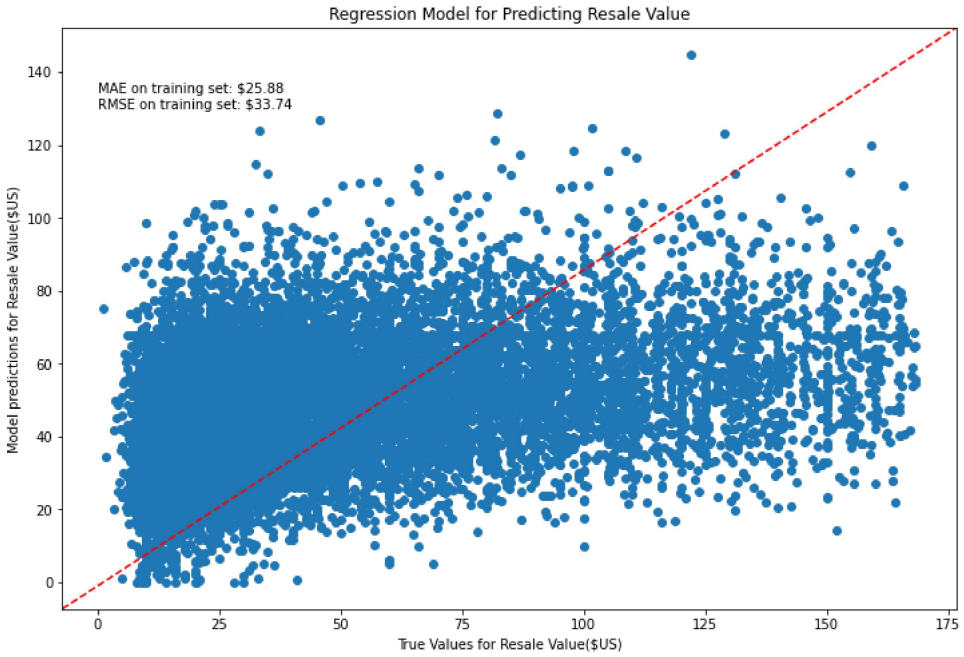
```
152/152 [==============================] - 330s 2s/step
```

In [155]:
```
test_mae = mean_absolute_error(test_generator.labels,pred)
test_mae
```

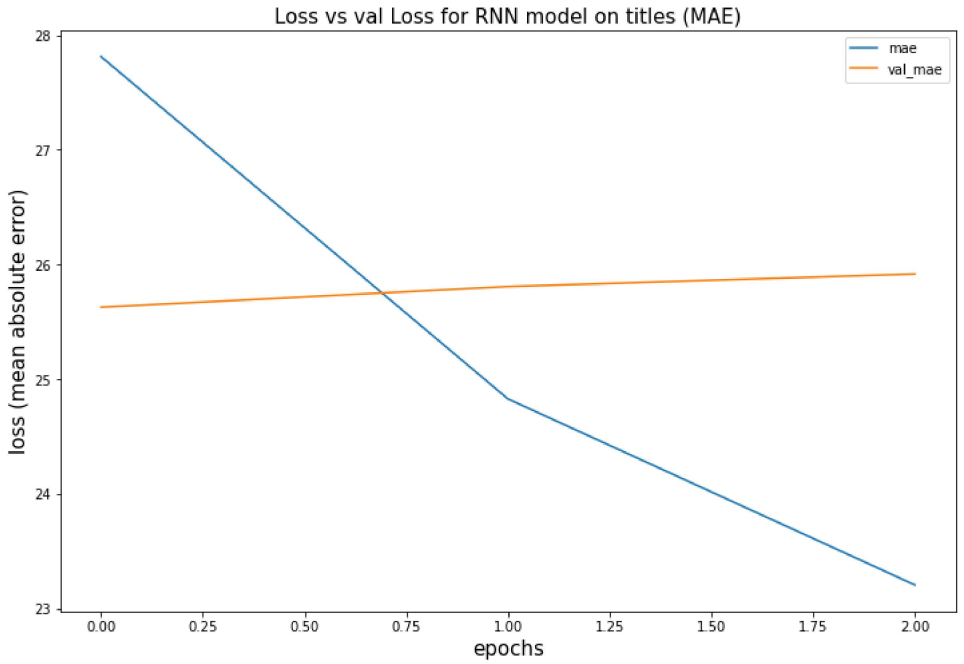
Out[155]:
```
25.876584786829934
```

In [165]:
```
RMSE = np.sqrt(1138.2579)
```

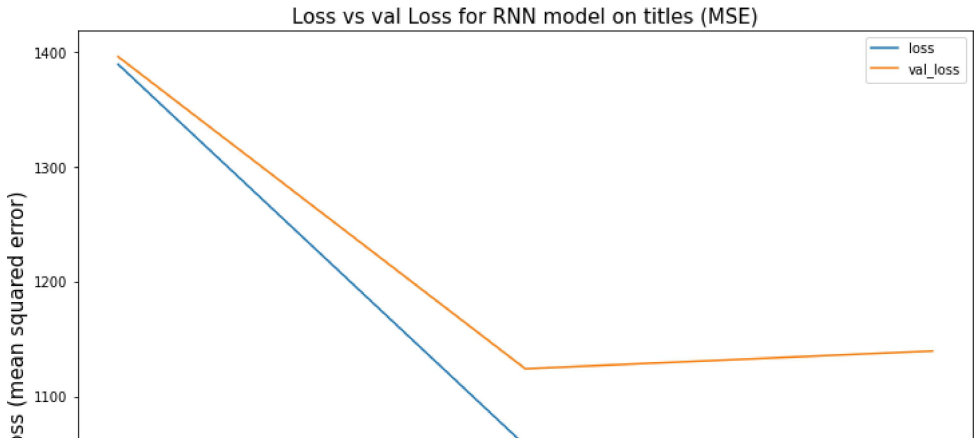
In [171]:
```
string_score = f'\nMAE on training set: ${test_mae:.2f}'
string_score += f'\nRMSE on training set: ${RMSE:.2f}'
fig, ax = plt.subplots(figsize=(12, 8))
plt.scatter(test_generator.labels,pred)
ax.plot([0, 1], [0, 1], transform=ax.transAxes, ls="--", c="red")
plt.text(0,130, string_score)
plt.title('Regression Model for Predicting Resale Value')
plt.ylabel('Model predictions for Resale Value($US)')
plt.xlabel('True Values for Resale Value($US)')
plt.savefig('images/Regression_CNN_relu1.png');
```



In [168]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(summary.history['mae'], label='mae')
plt.plot(summary.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/CNN_images_MAE4.png')
```



In [169]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(summary.history['loss'], label='loss')
plt.plot(summary.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/CNN_images_MSE4.png')
```



In [172]:
```
model.save("cnn_grayscale_relu1.h5")
```

In [ ]:
```
```