Data Science Challenge Problem — 2021 — Boeing Global Services

David Tersegno
dtersegno@gmail.com
Github
315 515 9983

**Data preparation.** Numeric data was largely left as-is. Some that was not considered useful, like zip codes, or Vehicle Make (redundant with the models) was removed. Textual or list-like data describing car attributes was converted to over 200 dummy features.

**Predicting dealer list price.** The data converted in the previous step was fed into a linear model. This model was then applied to the test set. The result was a fit with $R^2 \approx 0.8$.

**Predicting vehicle trim.** The 29 listed trims were encoded as a 29-d vector. Data was duplicated to balance trim classes, and fed into a keras sequential model with a hidden layer. This model only achieved about 12% accuracy with the balanced classes, but this is better than random choices of $1/29 \approx 0.03$. This is worse than simply assuming all models are Limited, which would give about 30% accuracy assuming the frequency of Limited vehicles is about the same as in the data.

**source code.** The sources are included here. However, they are in the form of ipython notebooks, so is inconvenient. Please check my Github for the boeingds project repository to view more easily.

```
#!/usr/bin/env python
# coding: utf-8

# EDA
# ---
#
# This notebook passes through each features in the used car data.
#

# In[457]:


#import libraries
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
#display options
pd.options.display.max_columns = 40
get_ipython().run_line_magic('matplotlib', 'inline')
plt.style.use('dark_background')


# In[458]:


#import data
data_path = '../data/'
train_data_filename = 'Training_DataSet.csv'
test_data_filename = 'Test_Dataset.csv'

traindf = pd.read_csv(data_path + train_data_filename)
traindf.info()


# In[459]:


#look at top of the list
traindf.head()


# In[18]:


#look at basic statistics of numeric data
traindf.describe()


# ---
# # ListingID
# ---

# In[26]:


# 'ListingID' monotonically increases, approxmately linearly, with index.
testdf.ListingID.plot()


# In[21]:
```

```
#Increase of subsequent IDs is always positive but variable.
testdf.ListingID.diff().plot()


# In[24]:


#The distribution of the increase is exponentially decaying with larger skips.
testdf.ListingID.diff().hist(bins = 30, log = True)


# # SellerCity
# These all appear to be USA cities. Top represented cities largely not from the west

# In[408]:


#plot number of instances of sales from the most represented cities
scvaluecounts = traindf.SellerCity.value_counts();
plot_limit = 20
plt.barh(scvaluecounts.index[:plot_limit], scvaluecounts[:plot_limit]);


# In[400]:


#how many cities are represented?
len(traindf.SellerCity.unique())


# In[409]:


#how many cities have more than one sale?
scvaluecounts[scvaluecounts > 1]


# In[410]:


#how many have more than 2 sales?
scvaluecounts[scvaluecounts > 2]
```

4

```
# In[424]:


#look at value counts of value counts.
scvcvc = scvaluecounts.value_counts()

#plot how many cities (y) have x sales
fig, ax = plt.subplots(figsize = (12,6))
ax.set_yscale('log')
plt.scatter(scvcvc.index, scvcvc)


# In[425]:


# plot how many sales come from a city with X sales (product of representation above
# the characteristic log shape for higher X comes from the values with only one city
fig, ax = plt.subplots(figsize = (12,6))
ax.set_yscale('log')
plt.scatter(scvcvc.index, scvcvc*scvcvc.index)


# In[431]:


#the above might be a bit deceptive. look at a histogram to account for the different

#number of cit
fig, ax = plt.subplots(figsize = (12,6))
plt.hist(scvcvc*scvcvc.index, bins = 50);


# In[443]:


# is there anything special about being sold from a high-sales city?
hi_sale_cutoff = 10
hi_sales_cities = traindf['SellerCity'].value_counts()[traindf['SellerCity'].value_co


# In[444]:


hi_sales_cities
```

```
# In[448]:


#get sales statistics only from high-sales (>10 records) cities
traindf[traindf['SellerCity'].apply(lambda city: city in hi_sales_cities.to_list())].


# In[447]:


#compare to statistics from low-sales cities.
traindf[traindf['SellerCity'].apply(lambda city: city not in hi_sales_cities.to_list(


# In[449]:


#with this (admittedly arbitrary) cutoff of cities with >10 sales in the set, cutting
# and std of sale price is about the same.


# In[451]:


#look at mean dealer sales for each city. look at high sales cities only since the me
unique_cities = traindf['SellerCity'].unique()
city_means = []
for city in unique_cities:
city_means.append(
traindf.loc[traindf['SellerCity'] == city, 'Dealer_Listing_Price'].mean()
)
print(len(city_means), 'different city means')


# In[455]:


#plot city sale USD mean with number of sales
plt.scatter(traindf['SellerCity'].value_counts(), city_means)
plt.xlabel('Number of sale records in city')
plt.ylabel('city Dealer listing price mean')


# In[456]:
```

```
#it looks like number of sales and the sale mean aren't related. Although higher sale
#in the middle, this is likely an artifact of averaging more sales.


# ---
# # SellerIsPriv
#
# If private seller. (Is a dealership a non-private seller?)

# In[62]:


#Only 14 of the listings are listed as private. This may cause overfitting.
priv


# In[70]:


# What are typical prices of these?
privateprices = traindf.loc[traindf['SellerIsPriv'] == 1, 'Dealer_Listing_Price' ]
privateprices.describe()


# In[94]:


privateprices.hist(bins = 8)
plt.axvline(privateprices.mean(), color = 'red')
plt.axvline(privateprices.describe()['50%'], color = 'green')


# In[95]:


#Compare this to the overall prices for the set later.


# # SellerListSrc

# In[98]:
```

```
#only a few different "seller listing source identifiers".
traindf.SellerListSrc.value_counts()


# In[99]:


#two listings are NaN
# These also have SellerZip, VehSellerNotes, and VehTransmission as NaN.
traindf.loc[traindf['SellerListSrc'].isna()]


# ---
# # SellerName

# In[104]:


#about 60% of the sellers only show once
len(traindf.SellerName.unique()), len(traindf.SellerName.unique())/len(traindf)


# In[130]:


#Look at number of sales of instances from each seller type.
sellernamecounts = traindf.SellerName.value_counts()[:40]
plt.figure(figsize = (12,12))
plt.barh(sellernamecounts.index, sellernamecounts, log = True)
plt.xticks(np.logspace(1,3,3));


# ---
# # SellerRating

# In[131]:


traindf.SellerRating.describe()


# In[139]:


#look at rating distribution
```

```
traindf.SellerRating.hist(bins = 20)


# In[143]:


#Seller rating with avg sales price isn't directly correlated
plt.scatter(
traindf.SellerRating,
traindf['Dealer_Listing_Price']
)
plt.xlabel('seller rating')
plt.ylabel("sale price")


# ---
# # SellerRevCnt

# In[150]:


#look at distribution of review counts. these include repeat-counted values from the
traindf.SellerRevCnt.hist(log = True)
plt.xlabel('review count')
plt.ylabel('number of sales with this count');


# In[155]:


#price with review count
plt.scatter(
traindf['SellerRevCnt'],
traindf['Dealer_Listing_Price']
)
plt.xlabel('Number of reviews')
plt.ylabel('sale price (USD)');


# ---
# # SellerState

# In[176]:
```

```
## this is redone with state names and regions following

#look at all unique seller states and rates
# statevcs = traindf['SellerState'].value_counts()
# plt.figure(figsize = (6,12))
# plt.barh(statevcs.index, statevcs, log = True)


# In[177]:


#import region table to look at regional representation
region_filepath = '../us-census-regions-divisions.csv'
regions = pd.read_csv(region_filepath)
regions.set_index('State Code', inplace=True)
#take a look to see it worked
regions.head()


# In[188]:


#join regions onto data to see representation of states and view
regiondata = traindf.join(regions, on = 'SellerState')[['SellerState', 'State', 'Regi
statesvcs = regiondata['State'].value_counts()
plt.figure(figsize=(6,12))
plt.barh(statesvcs.index, statesvcs, log = True)
plt.title("State representation (log)");
plt.grid(b = True, axis ='x')


# In[191]:


# look at region representations
regvcs = regiondata['Region'].value_counts()
plt.figure(figsize = (6,6))
plt.barh(regvcs.index, regvcs, log = False)
plt.title('Region representation')
plt.grid(b = True, axis = 'x')


# In[200]:
```

```
divvcs = regiondata['Division'].value_counts()
plt.figure(figsize = (6,6))
plt.barh(divvcs.index, divvcs, log = True)
plt.title('division representation (log)')
plt.grid(b = True, axis = 'x')
plt.xlim(1,3000);


# ---
# # SellerZip
#
# Most of the listings do not have a zip --- this may be influenced by online sales.
#
# Zip is highly related to seller and other location values.

# In[206]:


#look at common zips
traindf['SellerZip'].value_counts().head(20)


# In[202]:


traindf['SellerName'].value_counts()


# ---
# # VehBodystyle
#
# Every listing is SUV body style. This is a useless column.

# In[211]:


traindf['VehBodystyle'].value_counts()


# ---
# # VehCertified

# In[218]:
```

```
certvcs = traindf['VehCertified'].value_counts()
plt.bar(['Not Certified', 'Certified'], certvcs)
plt.title("vehicle certified");


# ---
# # VehColorExt
#
# There are a lot of exterior colors. It will be good to reduce these to subpropertie
#
# - general color
# - descriptive words like metallic, clearcoat, crystal, pearlcoat or Pearl Coat/Coat
# - texture words like Cashmere, Velvet, Frost, Pearl, ivory
# - extra sexy words like diamond, maximum, sangria (?), Stellar, Radiant, Passion, *
#     - find out which extra descriptive words are true descriptors or simply marketi
#
# It also appears that roof camera information is included in some of these.

# In[237]:


def value_count_barplot(series, h = True, title = "plot title", log = False, figsize
plt.figure(figsize = figsize)
this_value_counts = series.value_counts()
if h:
plot = plt.barh
else:
plot = plt.bar
plot(this_value_counts.index, this_value_counts, log = log)
plt.title(title)


# In[238]:


colorevcs = traindf['VehColorExt'].value_counts()
value_count_barplot(traindf['VehColorExt'], title = "exterior color representation",


# ---
# # VehColorInt
#
# Separate these into color and texture as well. Many are overlapping, for instance t

# In[239]:
```

```
value_count_barplot(traindf['VehColorInt'], title = 'Interior color representation',


# ---
# # VehDriveTrain
#
# some of these are repeats, such as many different ways of writing All Wheel Drive.
# Some are overlapping or ambiguous, such as 2WD. Is that FWD or RWD?

# In[240]:


value_count_barplot(traindf['VehDriveTrain'], title = 'Drive Train', log = True)


# ---
# # VehEngine
#
# Engine can be split into a few features:
#
# - volume
# - number of cylinders
# - horsepower
# - special (HEMI, supercharged, MDS == Multi displacement, vvt variable valve timing
#
# These are tough because most cars may have DOHC, certainly they all have "horsepowe

# In[242]:


value_count_barplot(traindf['VehEngine'], title = 'Engine represenation', log = True,


# In[ ]:


volumes = ['3.0L', '3.6L', '5.7L', '6.2L', '']


# ---
# # VehFeats
#
# Lot of good info here. It appears to be well-written and organized --- information
# In[465]:


#turn these objects into lists
#
traindf['VehFeats'] = traindf['VehFeats'].dropna().apply(lambda entry: eval(entry))


# In[471]:


traindf['VehFeats'].dropna()


# In[466]:


traindf['VehFeats']


# In[467]:


# create a flat list of all feats
feat_list = [
feat for feats in traindf['VehFeats'].dropna() for feat in feats
]
#make that list a series to easily count values
featvcs = pd.Series(feat_list).value_counts()
#look at those value counts
featvcs.head()


# In[485]:


featvcs.head(80)


# In[468]:


traindf['VehFeats']
```

```
# ---
# # VehFuel
#
# There is an unknown entry that is not used often, this can be mixed with NaN.

# In[297]:


traindf.loc[traindf['VehFuel'].isna(), 'VehFuel']


# In[298]:


traindf['VehFuel'].value_counts()


# ---
# # VehHistory
#
# string split this over number of previous owners, as well as other features to trac

# In[305]:


traindf['VehHistory'].head(10)


# ---
# # VehListdays

# In[318]:


#look at distribution of listed days
traindf['VehListdays'].hist(bins = 50, log = False)
plt.title('List days histogram ');
plt.xlabel('days listed')
plt.ylabel('count');
plt.show()

traindf['VehListdays'].hist(bins = 50, log = True)
plt.title('List days histogram (log)');
plt.xlabel('days listed')
plt.ylabel('count');
plt.show()


# In[322]:


#look at listed days with price. high price ones may go faster.
plt.scatter(traindf['VehListdays'], traindf['Dealer_Listing_Price'])
plt.xlabel('listed days')
plt.ylabel('listing price')


# ---
# # VehMake
#
# Only Jeeps and Cadillacs. None missing. About twice as many Jeeps.

# In[326]:


makevcs = traindf['VehMake'].value_counts()
makevcs


# ---
# # VehMileage
#
# What happens after 50,000 miles? Something special? There is a big dropoff in listi
#
# The prices for the long-milage ones do not sink as low as the lowest pre-50k ones.
#

# In[332]:


traindf['VehMileage'].hist(bins = 30)
plt.title('Mileage histogram')
plt.xlabel("miles driven")
plt.ylabel('count')


# In[333]:
```

```
#mileage with price
plt.scatter(traindf['VehMileage'], traindf['Dealer_Listing_Price'])


# In[339]:


#cars with milage >50k are all cadillacs.
traindf.loc[traindf['VehMileage'] > 50000].head(40)


# ---
# # VehModel
#
# All cars are either Jeep Grand Cherokee or Cadillac XT5s.

# In[340]:


traindf['VehModel'].value_counts()


# ---
# # VehPriceLabel
#
# Many of these are missing. Missing ones may all be the same, probably a "bad deal".

# In[344]:


traindf['VehPriceLabel'].value_counts()


# In[361]:


traindf['Dealer_Listing_Price'].hist(density = True)
traindf[traindf['VehPriceLabel'].isna()]['Dealer_Listing_Price'].hist(density = True)


# In[351]:
```

```
# ---
# # VehSellerNotes
#
# This will be tough to parse, as it's seller specific and manually entered. Much of
# It may be valuable to look through some to see if there is any unusual words to gra

# In[368]:


traindf['VehSellerNotes']


# In[367]:


traindf['VehSellerNotes'][4]


# ---
# # VehType
#
# Useless. They're all used cars.

# In[370]:


traindf['VehType'].value_counts()


# ---
# # VehTransmission
#
# It looks like most of these are like drivetrain --- manually entered, with a lot of

# In[371]:


traindf['VehTransmission'].value_counts()


# ---
# # VehYear
#
# Straightforward model year. This could be turned into dummies (each model is differ
# In[372]:


traindf['VehYear'].value_counts()


# ---
# # Vehicle_Trim
#
# there are a lot of vehicle trims. Some of these seem to overlap. For the sake of th

# In[374]:


traindf['Vehicle_Trim'].value_counts()


# ---
# # Dealer_Listing_Price
#
# the target. people don't like selling around \$30k?

# In[392]:


traindf['Dealer_Listing_Price'].hist(bins = np.linspace(15000,100000,86));
plt.title("dealer listing price histogram")
plt.xlabel("listing price (USD)")
plt.ylabel('count');


# ---

# In[398]:


correlation_features = ['SellerIsPriv', 'SellerRating', 'SellerRevCnt','VehCertified'
corr = traindf[correlation_features].corr()
plt.figure(figsize = (12,12))
sns.heatmap(corr, annot=True)

#!/usr/bin/env python
# coding: utf-8
# Processing
# ---
#
# This notebook prepares the data for modeling. Null values in numeric data are imput

# In[1]:


#import libraries
import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
# import sklearn

#display options
pd.options.display.max_columns = 40
get_ipython().run_line_magic('matplotlib', 'inline')
plt.style.use('dark_background')


# In[2]:


#import data
data_path = '../data/'
train_data_filename = 'Training_DataSet.csv'
test_data_filename = 'Test_Dataset.csv'

traindf = pd.read_csv(data_path + train_data_filename)


#copy training data to a new dataframe to use for modeling
traindf_proc = traindf.copy()
traindf_proc.info()


# ---
# # Processing feature by feature

# # ListingID
#
# drop it.

# In[3]:
```

```
#keep track to drop later.
columns_to_drop = ['ListingID']


#
#
# ## SellerCity
# Although SellerCity did not appear to influence the average dealer listing price, i

# In[4]:


#make dummies for Seller Cities with the most sales in the training set.
#perhaps a bit arbitrary, but let's cut it off at cities above 30 sales.
#that's the first 20 common cities.
cities = traindf['SellerCity'].value_counts(ascending= False)[:20]
cities = cities.index
cities


# In[5]:


# takes a dataframe, a column to expand upon, and a list of values for that column to
# any value in the column that is not in the dummy_list will be ignored

#the purpose of this is to make the same ordered columns in the training set as in an
#the dummy columns may notinclude the same cities, if there is a different frequency
#or if a certain city isn't represented.
def make_specific_dummies(df, column, dummy_list):
#make a copy so we don't change the original
df2 = df.copy()
#remove any entries in column that aren't in the dummy list
for dummy_value in dummy_list:
df2[column + '_' + str(dummy_value)] = df2[column].apply(lambda entry: 1 if entry ==
return df2.drop(columns = column)



# In[6]:


traindf = make_specific_dummies(traindf, 'SellerCity', cities)
```

```
traindf.head()


# # SellerListSrc
#
# Only two are nulls --- this will leave those as the only 0 for dummies. (no dropfir
# Again, ‘pd.get_dummies‘ could be a problem if there are a different order. Make dum

# In[7]:


sources = traindf['SellerListSrc'].dropna().unique()
sources


# In[8]:


traindf = make_specific_dummies(traindf, 'SellerListSrc', sources)


# ## SellerName
# Treat this like cities, with the most popular names marked. Again, arbitrarily for
# Some sellers might be tied with low prices for certain models.

# In[9]:


#not taking unique because there are so many. order by value count first, then take t
sellers = traindf['SellerName'].value_counts(ascending = False)[:20]
sellers = sellers.index
sellers


# In[10]:


traindf = make_specific_dummies(traindf, 'SellerName', sellers)


# ## SellerState
# All 50 states are represented. The limit of US states means that we can drop one. L

# In[11]:
```

```
states = traindf['SellerState'].value_counts(ascending = False).index[:-1]
states


# In[12]:


traindf = make_specific_dummies(traindf,'SellerState', states)


# # SellerZip, VehBodystyle
# drop.

# In[13]:


columns_to_drop.append('SellerZip')
columns_to_drop.append('VehBodystyle')


# # VehColorExt
#
# colors to consolidate:
#
#    - gray, platinum, silver, steel, granite, billet, billiet, Gy, sil, rhino
#    - black, Midnight Sky, Shadow, charcoal
#    - white, ivory
#    - blue
#    - brown, brownstone, mocha
#    - beige, beigh, cashmere, bronze, tan
#    - gold
#    - purple, amethyst, velvet
#    - deep red, dark red, deep cherry red, burgundy, sangria, deep auburn, maroon,
#    - red, red horizon
#    - (black forest) green
#    - pink
#
# textures / coats:
#
#     - metallic, me
#     - pearl, pearlcoat, pearl-coat
#     - crystal
#     - clear, clearcoat
#     - 3-coat, tricoat, tri-coat
#     - tintcoat
#
# nonvalues:
#
#     - nan
#     - unspecified
#     - Not Specified

# In[14]:


colors = ['silver', 'black','white', 'blue', 'brown', 'tan', 'gold', 'purple', 'deep

textures = ['metallic', 'pearl', 'crystal', 'diamond', 'clearcoat', 'tintcoat', 'tric

silvers = ['gray', 'platinum', 'silver', 'steel', 'granite', 'billet', 'billiet', 'Gy
blacks = ['black', 'midnight','shadow','charcoal']
whites = ['white','ivory']
browns = ['brown','brownstone','mocha']
tans = ['beige','beigh','cashmere','brown','tan']
purples = ['purple','amethyst','velvet']
deepreds = ['deep red','dark red','deep cherry red','burgundy','sangria','deep auburn


# In[15]:


#fill missing ext color values with blank string
traindf['VehColorExt'].fillna('', inplace = True)


# In[16]:


#create new dummy columns for basic colors. some will overlap (red, deep red, black f
color_synonyms = zip(colors, [silvers, blacks, whites, ['blue'], browns, tans, ['gold
color_synonyms = list(color_synonyms)
for color in color_synonyms:
this_color = color[0]
these_synonyms = color[1]
traindf['ext_' + this_color] = 0
for synonym in these_synonyms:
traindf['ext_' + this_color] = traindf[['VehColorExt','ext_' + this_color]].apply(lam
# In[17]:


#create dummy columns for textures
texture_synonyms = zip(textures,
[
['metal', 'me'],
['pearl'],
['crystal'],
['diamond'],
['clear'],
['tint'],
['3-coat', 'tricoat']
])
texture_synonyms = list(texture_synonyms)
for texture in texture_synonyms:
this_texture = texture[0]
these_synonyms = texture[1]
traindf['ext_' + this_texture] = 0
for synonym in these_synonyms:
#make the dummy value 1 if the synonym is in the description or if it's already 1
traindf['ext_' + this_texture] = traindf[['VehColorExt', 'ext_' + this_texture]].appl


# In[18]:


columns_to_drop.append('VehColorExt')


# # VehColorInt
#
# Do the same for interior colors.
#
# colors:
#
#     - white
#     - beige, cream, cirrus
#     - black, carbon, graphite, ebony
#     - gray, pewter, aluminum, sterling
#     - tan
#     - ruby red
#     - sugar maple
#     - bronze
#     - blue, indigo, plum
#     - brown, sepia
#     - red
#     - jet
#
#
#
# style:
#
#     - sport
#     - accent/ accents
#     - mini-perf, mini-perfo
#
# material:
#
#     - leather
#     - suede
#     - titanium
#     - cloth
#     - sapelle, sapele
#     - aluminum

# In[19]:


traindf['VehColorInt'].fillna('', inplace = True)


# In[20]:


int_colors = ['beige',
'black',
'jet',
'gray',
'red',
'maple',
'blue',
'brown',
]

styles = ['sport',
'accent',
'perf']

materials = ['leather',
'suede',
'titanium',
'cloth',
'sapele',
'aluminum']


# In[21]:


#create dummy columns for interior colors
int_color_synonyms = zip(int_colors,
[
['beige','cream','cirrus'],
['black','carbon','graphite','ebony'],
['jet'],
['gray','pewter','aluminum','sterling','steel'],
['red'],
['maple'],
['blue','indigo','plum'],
['brown','sepia']
])
int_color_synonyms = list(int_color_synonyms)
for color in int_color_synonyms:
this_color = color[0]
these_synonyms = color[1]
#create a column of zeros for this feature
traindf['int_' + this_color] = 0
#fill the new column if the scanned column has any of the synonyms
for synonym in these_synonyms:
#make the dummy value 1 if the synonym is in the description or if it's already 1
traindf['int_' + this_color] = traindf[['VehColorInt', 'int_' + this_color]].apply(la




# In[22]:




# scans through df[column] for anything in a synonym list, makes new columns with syn
# synonyms looks like
# [
#   [ value1, [synonym1, synonym2...],
#   [ value2, [synonym1, synonym2...],
#   ...
# ]
def make_synonym_dummies(df, column, new_column_prefix, synonyms):
#not memory efficient, but safe.
df2 = df.copy()
for value in synonyms:
this_value = value[0]
these_synonyms = value[1]
#create a column of zeros for this feature
df2[new_column_prefix + '_' + this_value] = 0
#fill the new column if the scanned column has any of the synonyms
for synonym in these_synonyms:
#make the dummy value 1 if the synonym is in the description or if it's already 1
df2[new_column_prefix + '_' + this_value] = df2[[column, new_column_prefix + '_' + th
return df2

def make_inclusive_dummies(df, column, new_column_prefix, values):
df2 = df.copy()
for value in values:
df2[new_column_prefix + '_' + value] = df2[column].apply(lambda entry: value in entry
return df2



# In[23]:


#style dummies
traindf = make_inclusive_dummies(traindf, 'VehColorInt', 'int', styles)


# In[24]:


#internal material dummies
traindf = make_inclusive_dummies(traindf, 'VehColorInt', 'int', materials)
traindf.columns[-9:]


# In[25]:


columns_to_drop.append('VehColorInt')
```

```
# ---
# # VehDriveTrain
#
# Split these into two, 4x4/AWD and FWD/2WD. There are fine distinctions in cars in r
#
# - AWD
#     - 4WD/AWD
#     - AllWheelDrive
#     - ALL WHEEL
#     - All-wheel drive
#     - AWD or 4x4
#     - 4x4/4WD
#     - 4x4
#     - All Wheel Drive
#     - ALL-WHEEL
#     - four wheel
#     - 4WD
# - 2WD
#     - front-wheel Drive
#     - 2WD
#     - FWD

# In[26]:


traindf['VehDriveTrain'].fillna('', inplace = True)


# In[27]:


drivetrains = ['awd', 'fwd']

awds = [
'4wd',
'awd',
'4x4',
'all-wheel',
'all wheel',
'four wheel',
'four-wheel'
]

fwds = [
'fwd',
'front-wheel',
'front wheel',
'2wd'
]

drivetrain_synonyms = zip(drivetrains,
[
awds,
fwds
])
drivetrain_synonyms = list(drivetrain_synonyms)


# In[28]:


drivetrain_synonyms


# In[29]:


#assign drivetrain dummies
traindf = make_synonym_dummies(traindf, 'VehDriveTrain', 'drivetrain', drivetrain_syn


# In[30]:


columns_to_drop.append('VehDriveTrain')


# # VehEngine
# A lot of these engine values are included in `VehFeats`. Instead of regexing a bunc

# In[31]:


columns_to_drop.append('VehEngine')


# # VehFeats

# In[32]:
```

```
#turn these objects into lists
#
traindf['VehFeats'] = traindf['VehFeats'].dropna().apply(lambda entry: eval(entry))


# In[33]:


# create a flat list of all feats
feat_list = [
feat for feats in traindf['VehFeats'].dropna() for feat in feats
]
#make that list a series to easily count values
featvcs = pd.Series(feat_list).value_counts(ascending = False)
#look at those value counts
featvcs.head()


# In[34]:


#grab the top feats to make a dummy list.
feats = featvcs.index[:80]


# In[35]:


traindf['VehFeats']


# In[36]:


for feat in feats:
traindf['feat_' + feat] = traindf['VehFeats'].dropna().apply(lambda featlist: feat in


# In[37]:


columns_to_drop.append('VehFeats')
```

```
# # VehFuel

# In[38]:


fuels = traindf['VehFuel'].unique()[:3]
fuels


# In[39]:


traindf = make_specific_dummies(traindf, 'VehFuel', fuels )


# # VehHistory

# In[40]:


traindf['VehHistory']


# In[41]:


#turn vehhistory into lists of strings
traindf['VehHistory'].fillna('', inplace = True)
traindf['VehHistory'] = traindf['VehHistory'].apply(lambda this_history: this_history


# In[42]:


# create a flat list of all hist
hist_list = [
hist for hists in traindf['VehHistory'].dropna() for hist in hists
]
#make that list a series to easily count values
histvcs = pd.Series(hist_list).value_counts(ascending = False)
#look at those value counts
histvcs
```

```
# In[43]:


#make hist dummies
hists = histvcs.index
for hist in hists:
traindf['hist_' + hist] = traindf['VehHistory'].dropna().apply(lambda histlist: hist


# In[44]:


columns_to_drop.append('VehHistory')


# # VehListdays
#
# Good to leave alone besides nas. Fill those with median.

# In[45]:


traindf['VehListdays'].fillna(traindf['VehListdays'].median(), inplace = True)


# # VehMake
#
# only two makes.

# In[46]:


traindf = make_specific_dummies(traindf, 'VehMake', ['Jeep'])


# # VehMileage
#
# numeric. Fine to leave alone. Two NaNs fill with median.

# In[47]:


traindf['VehMileage'].fillna(traindf['VehMileage'].median(), inplace = True)
```

```
# # VehModel
#
# These are 1:1 with Make since there are only two models and two makes.

# In[48]:


columns_to_drop.append('VehModel')


# # VehPriceLabel
#
# I'm guessing missing values are "bad". These will be the missing dummy.

# In[49]:


price_labels = traindf['VehPriceLabel'].unique()[:3]
price_labels


# In[50]:


traindf = make_specific_dummies(traindf,'VehPriceLabel', price_labels)


# # VehSellerNotes
#
# I'd love to go through this to mark off important notes. But that will take time th

# In[51]:



columns_to_drop.append('VehSellerNotes')


# # VehType
# all the same.

# In[52]:


columns_to_drop.append('VehType')
```

```
# # VehTransmission
#
# These are overwhelmingly 8-speed automatic. I don't think I will be able to extract

# In[53]:


columns_to_drop.append('VehTransmission')


# # VehYear
#
# make dummies from these. A certain year may be favorable.

# In[54]:


years = traindf['VehYear'].unique()
years


# In[55]:


traindf = make_specific_dummies(traindf, 'VehYear', years)


# # VehTrim
#
# the classification target.

# In[56]:


#save as a separate series
trim_data = traindf['Vehicle_Trim']
trim_data.to_csv('../data/train_trim_data.csv')

#drop
columns_to_drop.append('Vehicle_Trim')


# # finish up and save
# In[57]:


#drop columns
traindf.drop(columns = columns_to_drop, inplace = True)


# In[58]:


#turn all columns into floats for saving
traindf = traindf.astype(float)
traindf.info()


# In[59]:


#many rows are missing dealer list price, unfortunately. Drop these for now --- that’
traindf = traindf.dropna()


# In[60]:


#save processed training data
processed_train_path = ’../data/train_processed.csv’
traindf.to_csv(processed_train_path, index=None)


# # Repeat entire process for testing data.
#
# Ideally a lot of this would be made with a single python script and some nice custo

# In[61]:


testdf = pd.read_csv(data_path + test_data_filename)
testdf.info()


# In[62]:
```

```
testdf = make_specific_dummies(testdf, 'SellerCity', cities)
testdf = make_specific_dummies(testdf, 'SellerListSrc', sources)
testdf = make_specific_dummies(testdf, 'SellerName', sellers)
testdf = make_specific_dummies(testdf,'SellerState', states)
testdf['VehColorExt'].fillna('', inplace = True)
for color in color_synonyms:
this_color = color[0]
these_synonyms = color[1]
testdf['ext_' + this_color] = 0
for synonym in these_synonyms:
testdf['ext_' + this_color] = testdf[['VehColorExt','ext_' + this_color]].apply(lambd
for texture in texture_synonyms:
this_texture = texture[0]
these_synonyms = texture[1]
testdf['ext_' + this_texture] = 0
for synonym in these_synonyms:
#make the dummy value 1 if the synonym is in the description or if it's already 1
testdf['ext_' + this_texture] = testdf[['VehColorExt', 'ext_' + this_texture]].apply(
testdf['VehColorInt'].fillna('', inplace = True)
for color in int_color_synonyms:
this_color = color[0]
these_synonyms = color[1]
#create a column of zeros for this feature
testdf['int_' + this_color] = 0
#fill the new column if the scanned column has any of the synonyms
for synonym in these_synonyms:
#make the dummy value 1 if the synonym is in the description or if it's already 1
testdf['int_' + this_color] = testdf[['VehColorInt', 'int_' + this_color]].apply(lamb
testdf = make_inclusive_dummies(testdf, 'VehColorInt', 'int', styles)
testdf = make_inclusive_dummies(testdf, 'VehColorInt', 'int', materials)
testdf['VehDriveTrain'].fillna('', inplace = True)
testdf = make_synonym_dummies(testdf, 'VehDriveTrain', 'drivetrain', drivetrain_synon
testdf['VehFeats'] = testdf['VehFeats'].dropna().apply(lambda entry: eval(entry))
for feat in feats:
testdf['feat_' + feat] = 0
testdf['feat_' + feat] = testdf['VehFeats'].dropna().apply(lambda featlist: feat in f
testdf = make_specific_dummies(testdf, 'VehFuel', fuels )

testdf['VehHistory'].fillna('', inplace = True)
testdf['VehHistory'] = testdf['VehHistory'].apply(lambda this_history: this_history.s
for hist in hists:
testdf['hist_' + hist] = testdf['VehHistory'].dropna().apply(lambda histlist: hist in
testdf['VehListdays'].fillna(testdf['VehListdays'].median(), inplace = True)
testdf = make_specific_dummies(testdf, 'VehMake', ['Jeep'])
testdf['VehMileage'].fillna(testdf['VehMileage'].median(), inplace = True)
```

```
testdf = make_specific_dummies(testdf,'VehPriceLabel', price_labels)
testdf = make_specific_dummies(testdf, 'VehYear', years)


# In[63]:


testdf['VehFeats']


# In[64]:


# drop columns.
# it looks like vehicle trim isn't in the test data anyway.
columns_to_drop.remove('Vehicle_Trim')


# In[65]:



testdf.drop(columns = columns_to_drop, inplace = True)


# In[66]:


traindf.columns


# In[67]:


#The testing data has one fewer column --- that's the dealer price in training data!
testdf.columns


# In[68]:


# the feats introduced a lot of nulls --- fill these with 0
testdf.fillna(0., inplace = True)
```

```
# In[69]:


#turn bools into floats
testdf = testdf.astype(float)

#save the test data.
processed_test_path = '../data/test_processed.csv'
testdf.to_csv(processed_test_path, index=None)


# In[70]:


#from here, go to notebook 3 to train models.




#!/usr/bin/env python
# coding: utf-8

# model
# ---
#
# This notebook creates a linear regression model from the training data and applies

# In[46]:


#import libraries
import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score#, train_test_split


# In[47]:


train_data_path = '../data/train_processed.csv'
test_data_path = '../data/test_processed.csv'

traindf = pd.read_csv(train_data_path)
```

```
testdf = pd.read_csv(test_data_path)
traindf.info()


# In[61]:


testdf.info()


# In[48]:


#just to double-check, the training and testing data columns are the same.

traincols = traindf.columns.to_list()
traincols.remove('Dealer_Listing_Price')

traincols == testdf.columns.to_list()


# # predicting dealer listing price

# In[49]:


###### train-test split the training data to check
X = traindf.drop(columns = 'Dealer_Listing_Price')
y = traindf['Dealer_Listing_Price']

# actually, no need to tts. Just use cross_val_score. The linear fit is the same.
# Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state = 1916)


# In[50]:


#instantiate and fit a linear model to the training data using 5 cv folds

lr = LinearRegression()
cross_val_score(lr, X, y, cv = 5)


# In[51]:
```

```
# the fit is O.K. Use the linear regression on the test set and save.


# In[52]:


lr.fit(X, y)
preds = lr.predict(testdf)


# In[56]:


#the mean and std of the predicted values for the test set are similar to the dealer
preds.mean(), preds.std(), y.mean(), y.std()


# In[58]:


# look at some predictions
testdf['Dealer_Listing_Price'] = preds
testdf['Dealer_Listing_Price']


# In[59]:


testdf


# # predicting trim

# In[98]:


#imports
from keras.layers import Dense
from keras.models import Sequential


# In[66]:
```

```
#load the trim data
trimdata = pd.read_csv('../data/train_trim_data.csv', index_col='Unnamed: 0')
trimdata.head()


# In[67]:


#slap the trim back on
traindf = traindf.join(trimdata)


# In[71]:


#remove nulls. A lot of the trims are missing.
traindf.dropna(inplace=True)


# In[78]:


# #get unique trims
# trims = list(traindf['Vehicle_Trim'].unique())


# In[133]:


# trims.index('Limited')


# In[135]:


# #turn trim strings into a vector and vice versa.

# def encode_trim(some_trim_string, unique_trim_list):
#     encoded_trim_vector = np.full(len(unique_trim_list), 0)
#     trim_index = list(unique_trim_list).index(some_trim_string)
#     encoded_trim_vector[trim_index] = 1
#     return encoded_trim_vector

# def decode_trim(trim_vector, unique_trim_list):
#     trim_index = list(trim_vector).index(1)
#     decoded_trim_string = unique_trim_list[trim_index]
#     return decoded_trim_string


# In[134]:


# #test them out
# test_decode = np.full(len(trims), 0)
# test_decode[5] = 1

# test_encode = decode_trim(test_decode, trims)
# test_encode


# In[194]:


#encode_trim(test_encode, trims)


# In[137]:


#turn all training trims into vectors

train_trims = pd.get_dummies(traindf['Vehicle_Trim'])


# In[150]:


trims = train_trims.columns


# In[208]:


traindf[traindf['Vehicle_Trim'] == 'Limited']


# In[211]:


get_ipython().run_cell_magic('time', '', "\n#duplicate data with underrepresented cla
# In[213]:


len(newdf)


# In[250]:


num_inputs = len(newdf.columns)
num_outputs = len(trims)

#create a model
model = Sequential()
model.add(Dense(num_inputs, activation='relu'))
model.add(Dense(num_outputs*2, activation='relu'))
model.add(Dense(num_outputs, activation = 'softmax'))

model.compile(optimizer = 'adam', loss='categorical_crossentropy', metrics = ['accura


# In[251]:


#train it!

X = newdf.drop(columns = ['Dealer_Listing_Price','Vehicle_Trim'])
y = pd.DataFrame(newdf['Vehicle_Trim'].apply(lambda this_trim: encode_trim(this_trim,


# In[252]:


y


# In[255]:


get_ipython().run_cell_magic('time', '', 'epochs = 5120\nbatch_size = 256\nhistory =


# In[256]:
```

```
acc_hist, val_acc_hist = history.history['accuracy'], history.history['val_accuracy']

num_epochs_passed = len(acc_hist)
xes = np.linspace(0,num_epochs_passed,num_epochs_passed)
plt.plot(xes, acc_hist)
plt.plot(xes,val_acc_hist)


# In[257]:


#this is not good at all. Just train it on the unbalanced data.


# In[258]:


num_inputs = len(traindf.columns) - 2
num_outputs = len(trims)

#create a model
model = Sequential()
model.add(Dense(num_inputs, activation='relu'))
model.add(Dense(num_outputs*2, activation='relu'))
model.add(Dense(num_outputs, activation = 'softmax'))

model.compile(optimizer = 'adam', loss='categorical_crossentropy', metrics = ['accura


# In[259]:


#train it!

X = traindf.drop(columns = ['Dealer_Listing_Price','Vehicle_Trim'])
y = pd.DataFrame(traindf['Vehicle_Trim'].apply(lambda this_trim: encode_trim(this_tri


# In[261]:


get_ipython().run_cell_magic('time', '', 'epochs = 512\nbatch_size = 256\nhistory = m
# In[266]:


acc_hist, val_acc_hist = history.history['accuracy'], history.history['val_accuracy']

num_epochs_passed = len(acc_hist)
xes = np.linspace(0,num_epochs_passed,num_epochs_passed)
plt.plot(xes, acc_hist)
plt.plot(xes,val_acc_hist)


# In[272]:


model_preds = pd.Series([
trims[np.argmax(prediction)]
for prediction in model.predict(testdf.drop(columns = 'Dealer_Listing_Price'))
])


# In[273]:


model_preds.value_counts()


# In[280]:


predictions = pd.DataFrame(zip(preds, model_preds), columns = ['Dealer_List_Price', '
predictions.head()


# In[281]:


#save predictions
predictions.to_csv('../predictions.csv')


# In[ ]:
```