Boeing Data Science Challenge

Hazen Eckert

December 7, 2023

I used Python 3 in a Jupyter notebook to do all my work. I chose these based on familiarity and ease of use for this small dataset. For data cleaning and exploration I utilized Pandas. Natural Language Toolkit was used to generate features from strings. Sci-Kit Learn was used for the machine learning models and model validation.

I first cleaned the data. I imputed the values for features that were null for very few entries and added a label for null entries for the other features. A few entries had unique labels for a specific trim. These were relabeled using the most common label for that trim package.

I then generated several features. I created a geographical region feature to more easily capture similarities across nearby states. I used the information from wikipedia to generate normalized engine and drivetrain labels. The engine is a particularly important feature for Jeep trim levels as certain engines are only available with a subset of trim packages. I generated a feature that extracts a trim label from the unstructured text. For the most of the features that were multi-token strings I also created bag-of-words and bag-of-bigram features. I dropped the words and bigrams that did not occur in more than at least 5 entries. The vehicle history was a standardized list so I just extracted the list of elements.

I then evaluated several different regressor and classifier models. A random forest performed very well for both the the trim classification task and the price prediction. The hyperparameters for these models were then chosen by choosing the simplest model that had a cross validation score no more than one standard error below the best model. Each model was trained on the full dataset minus the entries where the predicted feature was null. The predictions are contained in "predictions.csv"

1 Appendix 1: Minimal Code to Generate Predictions

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

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from nltk.metrics import edit_distance
import re, nltk
nltk.download('punkt')

train = pd.read_csv('Training_DataSet.csv')
test = pd.read_csv('Test_Dataset.csv')

# remove capitalization
for x in ["SellerCity","VehColorExt","VehColorInt","VehFeats", "VehSellerNotes"]:
train[x]=train[x].str.lower()
test[x]=test[x].str.lower()



train.at[1125,"SellerZip"]=23294
train.at[1125,"SellerListSrc"]="HomeNet Automotive" # used this source for future sales
train.at[1125,"VehFuel"]="Gasoline"

train.at[3855,"SellerZip"]=48124
train.at[3855,"SellerListSrc"]="Inventory Command Center"
```

```
train.at[3855,"VehFuel"]="Gasoline"


# Two records are missing VehMileage and VehListdays, we will just fill those in with the means.


train["VehMileage"].fillna(value=train["VehMileage"].mean(),inplace=True)
test["VehMileage"].fillna(value=test["VehMileage"].mean(),inplace=True)
train["VehListdays"].fillna(value=train["VehListdays"].mean(),inplace=True)


# For the null VehColorExt VehColorInt VehDriveTrain VehEngine VehPriceLabel VehTransmission add an "unknown" category.


for x in ["VehColorExt", "VehColorInt", "VehDriveTrain", "VehEngine", "VehPriceLabel", "VehTransmission"]:
train[x].fillna(value="unknown",inplace=True)
test[x].fillna(value="unknown",inplace=True)



# For many names the first token is most informative with the remaining tokens just specifying the location.
# To capture sellers that have multiple locations, add a feature for the first token in a name.


train["SellerShortName"]=train["SellerName"].str.lower().str.split(n=1,expand=True)[0]
test["SellerShortName"]=test["SellerName"].str.lower().str.split(n=1,expand=True)[0]


# There is just one data point for HI and it doesn't show up in testing. HI is likely an outlier so it is removed.



train = train[train["SellerState"]!= "HI"]


# We add a feature that consolidates states in their Census Bureau-designated divisions.


divisions = {"New England" : {"CT", "ME", "MA", "NH", "RI", "VT"},
"Middle Atlantic" : {"NJ","NY","PA"},
"South Atlantic" : {"MD","DC","DE","VA","WV","NC","SC","GA","FL"},
"East North Central" : {"IL","IN","OH","MI","WI"},
"East South Central" : {"KY","TN","MS","AL"},
"West North Central" : {"ND","SD","NE","KS","MO","IA","MN"},
"West South Central" : {"TX","LA","AR","OK"},
"Mountain" : {"MT","ID","WY","NV","UT","CO","AZ","NM"},
"Pacific" : {"AK","HI","WA","OR","CA"}}
state_to_division = dict([(state,division) for division in divisions.keys() for state in divisions[division]])

train["SellerDivision"] = train["SellerState"].apply(lambda state: state_to_division[state])
test["SellerDivision"] = test["SellerState"].apply(lambda state: state_to_division[state])


# Normalize the labels for 4WD and FWD drivetrains.

drivetrains = {
"4WD" : ["4WD", "AWD", "4X4", "Four Wheel Drive", "ALL-WHEEL DRIVE", "All Wheel Drive",
"4x4", "4x4/4-wheel drive", "4x4/4WD", "AWD or 4x4", "All-wheel Drive", "ALL WHEEL",
"AllWheelDrive", "ALL-WHEEL DRIVE WITH LOCKING AND LIMITED-SLIP DIFFERENTIAL","4WD/AWD"],
"FWD" : ["FWD","FRONT-WHEEL DRIVE", "Front Wheel Drive","Front-wheel Drive","2WD"],
"unknown" : ["unknown"]
}
clean_drivetrain = dict([(raw,clean) for clean in drivetrains.keys() for raw in drivetrains[clean]])
train["VehDriveTrainClean"] = train["VehDriveTrain"].apply(lambda raw: clean_drivetrain[raw])
test["VehDriveTrainClean"] = test["VehDriveTrain"].apply(lambda raw: clean_drivetrain[raw])

# Tokenize the engine strings
train["VehEngineTokens"] = train["VehEngine"].str.split()
test["VehEngineTokens"] = test["VehEngine"].str.split()

# Extract the list of features from VehFeats and generate words and bigrams
nonalphanum = re.compile("[^a-zA-Z0-9_]")
def words_and_bigrams(s):
words = list(nltk.word_tokenize(s))
bigrams = list(nltk.bigrams(words))
return [(w,) for w in words] + bigrams

def extract_feats(feats):
if pd.isna(feats):
return [("unknown",)]
else:
return [gram for feat in feats.strip("]['").split("', '") for gram in words_and_bigrams(nonalphanum.sub(' ', feat).strip())]
train["VehFeatTokens"] = train["VehFeats"].apply(extract_feats)
test["VehFeatTokens"] = test["VehFeats"].apply(extract_feats)

# Extract words and bigrams from Seller notes
def extract_notes(notes):
if pd.isna(notes):
return [("unknown",)]
else:
return [gram for gram in words_and_bigrams(nonalphanum.sub(' ', notes).strip())]
train["VehSellerNotesTokens"] = train["VehSellerNotes"].apply(extract_notes)
```

```
test["VehSellerNotesTokens"] = test["VehSellerNotes"].apply(extract_notes)

# As VehHistory is standardized we just extract the elements in the list
train["VehHistoryTokens"] = train["VehHistory"].fillna('unknown').str.split(', ')
test["VehHistoryTokens"] = test["VehHistory"].fillna('unknown').str.split(', ')

# Extract words and bigrams from seller names
def extract_names(name):
if pd.isna(name):
return [("unknown",)]
else:
return [gram for gram in words_and_bigrams(nonalphanum.sub(' ', name).strip())]
train["SellerNameTokens"] = train["SellerName"].apply(extract_names)
test["SellerNameTokens"] = test["SellerName"].apply(extract_names)

def extract_engine(row):
if (not pd.isna(row["VehFeats"]) and "3.6" in row["VehFeats"]) or (not pd.isna(row["VehSellerNotes"]) and "3.6" in row["VehSellerNotes"]):
return "3.6L V6"
elif (not pd.isna(row["VehFeats"]) and "5.7" in row["VehFeats"]) or (not pd.isna(row["VehSellerNotes"]) and "5.7" in row["VehSellerNotes"]):
return "5.7L V8"
elif (not pd.isna(row["VehFeats"]) and "6.2" in row["VehFeats"]) or (not pd.isna(row["VehSellerNotes"]) and "6.2" in row["VehSellerNotes"]):
return "6.2L V8"
elif (not pd.isna(row["VehFeats"]) and "6.4" in row["VehFeats"]) or (not pd.isna(row["VehSellerNotes"]) and "6.4" in row["VehSellerNotes"]):
return "6.4L V8"
else:
return row["VehEngine"]
def clean_train_engine(row):
if row["VehMake"] == "Cadillac":
return "3.6L V6"
elif row["VehFuel"]=="Diesel":
return "3.0L V6"
elif "3.6" in row["VehEngine"] or "V6" in row["VehEngine"] or row["VehEngine"] in ["6-cylinder","6","6 Cylinder","V-6 cyl"]  \
or row["Vehicle_Trim"] in ["Laredo", "Laredo E"]:
return "3.6L V6"
elif "6.4" in row["VehEngine"] or row["Vehicle_Trim"] in ["SRT","SRT8"]:
return "6.4L V8"
elif "6.2" in row["VehEngine"] or "Trackhawk" == row["Vehicle_Trim"]:
return "6.2L V8"
elif "5.7" in row["VehEngine"] or ("8" in row["VehEngine"] and not (row["Vehicle_Trim"] in ["SRT","SRT8","Trackhawk"])):
return "5.7L V8"
else:
return extract_engine(row)
def clean_test_engine(row):
if row["VehMake"] == "Cadillac":
return "3.6L V6"
elif row["VehFuel"]=="Diesel":
return "3.0L V6"
elif "3.6" in row["VehEngine"] or "V6" in row["VehEngine"] or row["VehEngine"] in ["6-cylinder","6","6 Cylinder","V-6 cyl"]:
return "3.6L V6"
elif "6.2" in row["VehEngine"]:
return "6.2L V8"
elif "6.4" in row["VehEngine"] or row["VehEngine"]=="8-cylinder": # the one "8-cylinder" has SRT in notes
return "6.4L V8"
elif "5.7" in row["VehEngine"] or row["VehEngine"]=="8 Cylinder Engine":
# the one "8 Cylinder Engine" has uninformative features, rather than label it "unknown" I label it the most common V8 engine type
return "5.7L V8"
elif row["VehEngine"]=="0":
return "unknown"
else:
return extract_engine(row)

train["VehEngineClean"] = train.apply(clean_train_engine,axis=1)
test["VehEngineClean"] = test.apply(clean_test_engine,axis=1)

# A simple attempt to extract a trim label from VehFeats or VehSellerNotes
def extract_trim(row):
if not pd.isnull(row["VehSellerNotes"]):
if row["VehMake"]=="Cadillac":
if "platinum" in row["VehSellerNotes"]:
return "Platinum"
elif "premium luxury" in row["VehSellerNotes"]:
return "Premium Luxury"
elif "luxury" in row["VehSellerNotes"]:
return "Luxury"
elif "base" in row["VehSellerNotes"]:
return "Base"
else:
if "laredo e" in row["VehSellerNotes"]:
return "Laredo E"
elif "laredo" in row["VehSellerNotes"]:
return "Laredo E"
elif "overland" in row["VehSellerNotes"]:
return "Overland"
elif "high altitude" in row["VehSellerNotes"]:
return "High Altitude"
elif "altitude" in row["VehSellerNotes"]:
return "Altitude"
elif "summit" in row["VehSellerNotes"]:
return "Summit"
elif "trailhawk" in row["VehSellerNotes"]:
return "Trailhawk"
elif "srt" in row["VehSellerNotes"]:
```

```
return "SRT"
elif "trackhawk" in row["VehSellerNotes"]:
return "Trackhawk"
elif "sterling edition" in row["VehSellerNotes"]:
return "Sterling Edition"
elif "upland" in row["VehSellerNotes"]:
return "Upland"
elif "limited" in row["VehSellerNotes"]:
return "Limited"
elif not pd.isnull(row["VehFeats"]):
if row["VehMake"]=="Cadillac":
if "platinum" in row["VehFeats"]:
return "Platinum"
elif "premium luxury" in row["VehFeats"]:
return "Premium Luxury"
elif "luxury" in row["VehFeats"]:
return "Luxury"
elif "base" in row["VehFeats"]:
return "Base"
else:
if "laredo e" in row["VehFeats"]:
return "Laredo E"
elif "laredo" in row["VehFeats"]:
return "Laredo E"
elif "overland" in row["VehFeats"]:
return "Overland"
elif "high altitude" in row["VehFeats"]:
return "High Altitude"
elif "altitude" in row["VehFeats"]:
return "Altitude"
elif "summit" in row["VehFeats"]:
return "Summit"
elif "trailhawk" in row["VehFeats"]:
return "Trailhawk"
elif "srt" in row["VehFeats"]:
return "SRT"
elif "trackhawk" in row["VehFeats"]:
return "Trackhawk"
elif "sterling edition" in row["VehFeats"]:
return "Sterling Edition"
elif "upland" in row["VehFeats"]:
return "Upland"
elif "limited" in row["VehFeats"]:
return "Limited"
return "Unknown"
train["Vehicle_Trim_Extraction"] = train.apply(extract_trim,axis=1)
test["Vehicle_Trim_Extraction"] = test.apply(extract_trim,axis=1)

# An agressively normalized version of trim labels
def clean_trim(row):
if pd.isnull(row["Vehicle_Trim"]):
return row["Vehicle_Trim"]
elif row["VehMake"]=="Cadillac":
if row["Vehicle_Trim"] in ["Premium Luxury","Premium Luxury AWD","Premium Luxury FWD"]:
return "Premium Luxury"
elif row["Vehicle_Trim"] in ["Luxury","Luxury AWD","Luxury FWD"]:
return "Luxury"
elif row["Vehicle_Trim"] in ["Platinum","Platinum AWD"]:
return "Platinum"
else:
return "Base"
else:
if row["Vehicle_Trim"] in ["Limited","75th Anniversary","Limited 75th Anniversary Edition","Limited 4x4","75th Anniversary Edition", \
"Limited X","Limited 75th Anniversary"]:
return "Limited"
elif row["Vehicle_Trim"] in ["SRT", "SRT Night"]:
return "SRT"
else:
return row["Vehicle_Trim"]
train["Vehicle_Trim_Clean"] = train.apply(clean_trim,axis=1)

# Replace trim levels that are used only once
train["Vehicle_Trim"].value_counts()
def clean_trim(row):
if pd.isnull(row["Vehicle_Trim"]):
return row["Vehicle_Trim"]
elif row["VehMake"]=="Cadillac":
return row["Vehicle_Trim"]
else:
if row["Vehicle_Trim"] in ["Limited X", "Limited 4x4"]:
return "Limited"
elif row["Vehicle_Trim"] in ["Limited 75th Anniversary","75th Anniversary Edition"]:
return "75th Anniversary"
else:
return row["Vehicle_Trim"]
train["Vehicle_Trim_Semiclean"] = train.apply(clean_trim,axis=1)

# normalize unknown color labeld
def clean_ext_color(color):
if color == "undetermined" or color == "unspecified":
return "unknown"
else:
```

```
return color
train["VehColorExt"] = train["VehColorExt"].apply(clean_ext_color)
test["VehColorExt"] = test["VehColorExt"].apply(clean_ext_color)

# Extract words and bigrams from colors
def extract_color(color):
if pd.isna(color):
return [("unknown",)]
else:
return [gram for gram in words_and_bigrams(nonalphanum.sub(' ', color).strip())]
train["VehColorExtTokens"] = train["VehColorExt"].apply(extract_color)
test["VehColorExtTokens"] = test["VehColorExt"].apply(extract_color)
train["VehColorIntTokens"] = train["VehColorInt"].apply(extract_color)
test["VehColorIntTokens"] = test["VehColorInt"].apply(extract_color)


# Encode features for model training
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder, MultiLabelBinarizer,StandardScaler
real_features = ["SellerRating","SellerRevCnt","VehListdays","VehMileage"]
already_encoded_features = ["SellerIsPriv","VehCertified"]
one_hot_encoded_features = ["SellerListSrc","SellerState","SellerDivision","VehDriveTrainClean","VehEngineClean","VehFuel",\
"VehMake","VehPriceLabel","VehYear","Vehicle_Trim_Extraction"]
mlb_encoded_features = ["SellerNameTokens","VehEngineTokens","VehFeatTokens","VehSellerNotesTokens","VehHistoryTokens",\
"VehColorExtTokens","VehColorIntTokens"]

# Scale real valued features
scaler = StandardScaler().fit(pd.concat([train[real_features],test[real_features]],ignore_index=True))
X_train = pd.DataFrame(scaler.transform(train[real_features]), columns = real_features,index=train.index)
X_predict = pd.DataFrame(scaler.transform(test[real_features]), columns = real_features,index=test.index)


# One hot encode categorical variables
def custom_combiner(feature, category):
return str(feature) + "_" + str(category)
ohe = OneHotEncoder(feature_name_combiner=custom_combiner,handle_unknown='ignore',sparse_output=False).fit(train[one_hot_encoded_features])
temp = pd.DataFrame(ohe.transform(train[one_hot_encoded_features]),index=train.index,columns=ohe.get_feature_names_out())
X_train = X_train.join(temp)

temp_predict = pd.DataFrame(ohe.transform(test[one_hot_encoded_features]),index=test.index,columns=ohe.get_feature_names_out())
X_predict = X_predict.join(temp_predict)

# Encode lists of features using a multi label binarizer
for feature in mlb_encoded_features:
mlb = MultiLabelBinarizer(sparse_output=True).fit(pd.concat([train[feature],test[feature]],ignore_index=True))
temp = pd.DataFrame.sparse.from_spmatrix(mlb.transform(train[feature]),columns=[feature+str(c) for c in mlb.classes_],index=train.index)
temp = temp.loc[:,temp.sum()>=5] # drop features that don't occur in at least 5 samples in training
X_train = X_train.join(temp)

temp_predict = pd.DataFrame.sparse.from_spmatrix(mlb.transform(test[feature]),columns=[feature+str(c) for c in mlb.classes_],index=test.index)
temp_predict = temp_predict.loc[:,temp.loc[:,temp.sum()>=5].columns] # drop features that don't occur in at least 5 samples in training
X_predict = X_predict.join(temp_predict)

Y_train = train["Vehicle_Trim"]
Y_train_clean = train["Vehicle_Trim_Clean"]
Y_train_semiclean = train["Vehicle_Trim_Semiclean"]

# Dataset for trim prediction
X_train_trim = X_train[~Y_train.isnull()]
Y_train_trim = Y_train[~Y_train.isnull()]
Y_train_clean_trim = Y_train_clean[~Y_train.isnull()]
Y_train_semiclean_trim = Y_train_semiclean[~Y_train.isnull()]

# Dataset for price prediction
X_train_price = X_train[~train["Dealer_Listing_Price"].isnull()]
feature_scaler = StandardScaler().fit(X_train)
X_train_price_normalized = pd.DataFrame(feature_scaler.transform(X_train_price), columns = X_train_price.columns,index=X_train_price.index)
X_predict_normalized = pd.DataFrame(feature_scaler.transform(X_predict), columns = X_predict.columns,index=X_predict.index)

Y_train_price = train[~train["Dealer_Listing_Price"].isnull()]["Dealer_Listing_Price"]
price_scaler = StandardScaler().fit(Y_train_price.to_numpy().reshape(-1, 1))
Y_train_price_normalized =pd.Series(price_scaler.transform(Y_train_price.to_numpy().reshape(-1, 1)).reshape(1,-1)[0], index=Y_train_price.index)


from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(max_depth=18)
rf_clf.fit(X_train_trim, Y_train_semiclean_trim)
predict = pd.DataFrame(rf_clf.predict(X_predict),columns=["Vehicle_Trim_Predicted"]).join(test["ListingID"])

from sklearn.ensemble import RandomForestRegressor
rf_reg = RandomForestRegressor(n_jobs=-1)
rf_reg.fit(X_train_price_normalized,Y_train_price_normalized)
predict =pd.DataFrame(price_scaler.inverse_transform(rf_reg.predict(X_predict_normalized).reshape(-1, 1)), columns=["Predicted_price"], \
index=X_predict_normalized.index).join(predict)

predict[["ListingID","Vehicle_Trim_Predicted","Predicted_price"]].to_csv("predictions.csv",index=False)
```

2 Appendix 2: Full Jupyter Notebook

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from nltk.metrics import edit_distance
import re, nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/hazeneckert/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

[1]:
```
True
```

[2]:
```
train = pd.read_csv('Training_DataSet.csv')
test = pd.read_csv('Test_Dataset.csv')
```

3 Info about the vehicles

Cadillac XT5 2017-2019 - Only offers a 3.6L V6 gasoline engine in the US - Two drivetrains AWD or FWD - One transmission 8-speed - Four Trim levels [Base, Luxury, Premium Luxury, Platinum]

Jeep Grand Cherokee 2015-2019 - 10 trim levels: [Laredo, Laredo E, Altitude, High Altitude, Upland, Trailhawk, Limited, Sterling Edition, Overland, Summit, SRT, SRT Trackhawk] - Offers 6 engine options: - 3.6L V6: Laredo, Laredo E, Altitude, Limited, Sterling Edition, Trailhawk, Overland, High Altitude, Summit - 5.7L V8: Limited, Limited X, Sterling Edition, Trailhawk, Overland, High Altitude, Summit - 6.4L V8: SRT - 6.2L V8: SRT Trackhawk - 3.0L Diesel: Limited, Overland, Summit - The transmission is a function of year and engine - No info about drivetrain

4 Exploratory Data Analysis

ListingID is a unique identifier.

SellerCity - has some misspelled cities: 'Morganton' and 'Morgantown', 'Milwaukee' and 'Milwaukie', - has different capitalizations: 'Green Bay' and 'Green bay', - has some unicode symbols: 'Coeur d'Alene' and 'O'Fallon', - has different labels for the same location: 'Charter Twp of Clinton', 'Clinton','Clinton Township' - is never null in training or testing - There are cities in the test data never observed in the training data

SellerIsPriv is extremely biased towards False, 6284/6298 and is never null in training or testing.

SellerListSrc takes 8 non-null values and is null for 2 entries. These 2 null entries are also the null entries for SellerZip. All values observed in the test set is in the training set. There are no null values in the training data.

SellerName is never null. There are sellers in the test data not in the training data. There are many entries that are very similar like 'CarMax Buffalo' and 'CarMax Buford'.

SellerRating is never null in training or testing. There are 5988 nonzero entries.

SellerRevCnt is never null in training or testing. There are 6128 nonzero entries.

SellerState has observations for each state in training. 6 states are not included in the testing data, ['HI', 'ME', 'MT', 'NM', 'OR', 'VT']. Never null in training or testing. Probably a good idea to remove HI

VehBodystyle is always 'SUV'

VehCertified is mostly false, 4879/6298. Never null in training or testing.

VehColorExt has null values in training and testing but most entries are in the top 20 or so most common colors.

VehColorInt has null values in training and testing but most entries are in the top 10 or so most common colors. Maybe extracting the leather option would be helpful.

VehDriveTrain has null values in training and testing. There are several different names for the same thing.

VehEngine has null values in training and testing. There are several different names for the same thing.

Skip VehFeats for now

VehFuel has null values in the training but not the test data. Other than 'Unknown' and null entries this column is clean.

VehHistory has null values in training and testing. It seems easy to extract a small number of features from this column.

VehListdays is null for the same two entries that SellerListSrc is null. It is never null in the test set.

VehMake is never null in training or testing. It is either 'Jeep' or 'Cadillac'

VehMileage is null in training and test but very rarely, twice and once, respectively.

VehModel is never null in training or testing. It is either 'Grand Cherokee' or 'XT5'

VehPriceLabel is null in training and test but otherwise takes 3 values.

Skip VehSellerNotes for now

VehType is always 'Used'

VehTransmission is null in training and test and has a bunch of redundant labels.

VehYear is never null and the training and testing share the same 5 years.

Vehicle_Trim is null for 405 entries in training.

Dealer_Listing_Price is null for 52 entries in training.

[3]:
```
train.describe(include='all')
```

[3]:
```
           ListingID SellerCity SellerIsPriv            SellerListSrc  \
count   6.298000e+03       6298         6298                     6296
unique           NaN       1318            2                        8
top              NaN    Chicago        False  Digital Motorworks (DMi)
freq             NaN        118         6284                     3086
mean    4.318130e+06        NaN          NaN                      NaN
std     2.486031e+06        NaN          NaN                      NaN
min     3.287000e+03        NaN          NaN                      NaN
25%     2.178112e+06        NaN          NaN                      NaN
50%     4.298122e+06        NaN          NaN                      NaN
75%     6.488249e+06        NaN          NaN                      NaN
max     8.620012e+06        NaN          NaN                      NaN

                                    SellerName  SellerRating  \
count                                     6298   6298.000000
unique                                    2452           NaN
top     Vroom (Online Dealer - Nationwide Delivery)           NaN
freq                                       381           NaN
mean                                       NaN      4.138346
std                                        NaN      1.188033
min                                        NaN      0.000000
25%                                        NaN      4.000000
50%                                        NaN      4.600000
75%                                        NaN      4.800000
max                                        NaN      5.000000

        SellerRevCnt SellerState     SellerZip VehBodystyle  ... VehMake  \
count    6298.000000        6298   6296.000000         6298  ...    6298
unique           NaN          50           NaN            1  ...       2
top              NaN          IL           NaN          SUV  ...    Jeep
freq             NaN         753           NaN         6298  ...    4199
mean      434.565576         NaN  45234.211722          NaN  ...     NaN
std      1274.257411         NaN  20380.478191          NaN  ...     NaN
min         0.000000         NaN   1105.000000          NaN  ...     NaN
25%        28.000000         NaN  28806.000000          NaN  ...     NaN
50%       126.000000         NaN  46410.000000          NaN  ...     NaN
75%       401.000000         NaN  60126.000000          NaN  ...     NaN
max     14635.000000         NaN  99654.000000          NaN  ...     NaN

          VehMileage        VehModel VehPriceLabel  \
count    6296.000000            6298          6013
unique           NaN               2             3
top              NaN  Grand Cherokee     Good Deal
freq             NaN            4199          4488
mean    26369.364358             NaN           NaN
std     13036.568712             NaN           NaN
min         0.000000             NaN           NaN
25%     16835.000000             NaN           NaN
50%     26181.000000             NaN           NaN
75%     36468.500000             NaN           NaN
max     83037.000000             NaN           NaN

                                      VehSellerNotes VehType  \
count                                           6055    6298
unique                                          4920       1
top     CARVANA CERTIFIED INCLUDES: 150-POINT INSPECTI...    Used
freq                                             218    6298
mean                                             NaN     NaN
std                                              NaN     NaN
min                                              NaN     NaN
25%                                              NaN     NaN
50%                                              NaN     NaN
75%                                              NaN     NaN
max                                              NaN     NaN

          VehTransmission      VehYear  Vehicle_Trim Dealer_Listing_Price
count                6101  6298.000000          5893          6246.000000
unique                 33          NaN            29                  NaN
top     8-Speed Automatic          NaN       Limited                  NaN
freq                 4395          NaN          1912                  NaN
mean                  NaN  2016.792633           NaN         32265.053314
std                   NaN     1.206566           NaN          7538.339005
min                   NaN  2015.000000           NaN         18289.000000
25%                   NaN  2015.000000           NaN         26900.000000
50%                   NaN  2017.000000           NaN         31455.500000
75%                   NaN  2018.000000           NaN         35991.000000
max                   NaN  2019.000000           NaN         89500.000000

[11 rows x 29 columns]
```

[4]:
```
train.isnull().sum(),test.isnull().sum()
```

[4]:
```
(ListingID                  0
 SellerCity                 0
 SellerIsPriv               0
 SellerListSrc              2
 SellerName                 0
 SellerRating               0
 SellerRevCnt               0
 SellerState                0
 SellerZip                  2
 VehBodystyle               0
 VehCertified               0
 VehColorExt               73
 VehColorInt              728
 VehDriveTrain            401
 VehEngine                361
 VehFeats                 275
 VehFuel                    2
 VehHistory               201
 VehListdays                2
 VehMake                    0
 VehMileage                 2
 VehModel                   0
 VehPriceLabel            285
 VehSellerNotes           243
 VehType                    0
 VehTransmission          197
 VehYear                    0
 Vehicle_Trim             405
 Dealer_Listing_Price      52
 dtype: int64,
 ListingID             0
 SellerCity            0
 SellerIsPriv          0
 SellerListSrc         0
 SellerName            0
 SellerRating          0
 SellerRevCnt          0
 SellerState           0
 SellerZip             0
 VehBodystyle          0
 VehCertified          0
 VehColorExt           7
 VehColorInt         108
 VehDriveTrain        64
 VehEngine            58
 VehFeats             37
 VehFuel               0
 VehHistory           27
 VehListdays           0
 VehMake               0
 VehMileage            1
 VehModel              0
 VehPriceLabel        38
 VehSellerNotes       41
 VehType               0
 VehTransmission      27
 VehYear               0
 dtype: int64)
```

[5]:
```
# remove capitalization
for x in ["SellerCity","VehColorExt","VehColorInt","VehFeats", "VehSellerNotes"]:
    train[x]=train[x].str.lower()
    test[x]=test[x].str.lower()
```

[6]:
```
train.iloc[3855]
```

[6]:
```
ListingID                             5306897
SellerCity                           dearborn
SellerIsPriv                            False
SellerListSrc                             NaN
SellerName                 Jack Demmer Lincoln
SellerRating                              4.8
SellerRevCnt                              261
SellerState                                MI
SellerZip                                 NaN
VehBodystyle                              SUV
VehCertified                            False
VehColorExt                               NaN
VehColorInt                               NaN
VehDriveTrain                             NaN
VehEngine                                 NaN
VehFeats                                  NaN
VehFuel                                   NaN
VehHistory                                NaN
VehListdays                               NaN
```

```
VehMake                                   Jeep
VehMileage                             36678.0
VehModel                        Grand Cherokee
VehPriceLabel                       Fair Price
VehSellerNotes                             NaN
VehType                                   Used
VehTransmission                            NaN
VehYear                                   2015
Vehicle_Trim                           Limited
Dealer_Listing_Price                   23500.0
Name: 3855, dtype: object
```

[7]:

```
# Find the entries with null SellerListSrc
train[train.SellerListSrc.isnull()]
```

[7]:

```
      ListingID SellerCity  SellerIsPriv SellerListSrc  \
1125    1562581   richmond         False           NaN
3855    5306897   dearborn         False           NaN

                     SellerName  SellerRating  SellerRevCnt SellerState  \
1125  Pearson Chrysler Jeep Dodge RAM          1.0             4          VA
3855             Jack Demmer Lincoln          4.8           261          MI

      SellerZip VehBodystyle  ...  VehMake VehMileage        VehModel  \
1125        NaN          SUV  ...     Jeep    38329.0  Grand Cherokee
3855        NaN          SUV  ...     Jeep    36678.0  Grand Cherokee

     VehPriceLabel VehSellerNotes VehType VehTransmission VehYear  \
1125     Good Deal            NaN    Used             NaN    2017
3855    Fair Price            NaN    Used             NaN    2015

      Vehicle_Trim Dealer_Listing_Price
1125       Limited              26333.0
3855       Limited              23500.0

[2 rows x 29 columns]
```

[8]:

```
train[train.SellerName.str.contains("Lincoln")]["SellerListSrc"].value_counts()
```

[8]:

```
SellerListSrc
Inventory Command Center    39
Digital Motorworks (DMi)    34
HomeNet Automotive          10
Name: count, dtype: int64
```

Only two records have null SellerListSrc, VehFuel and SellerZip, records 1125 and 3855. Rather than drop these record or create an 'unknown' category, we are going to fill these in the best we can. We can look up the zipcodes for these dealerships. We have other entries from the seller in record 1125, which we will use to complete the SellerListSrc. For record 3855 we fill SellerListSrc using the most common source for Lincoln dealerships. We chose to fill in the VehFuel entries with Gasoline since it is the most common.

[9]:

```
train.at[1125,"SellerZip"]=23294
train.at[1125,"SellerListSrc"]="HomeNet Automotive" # used this source for future sales
train.at[1125,"VehFuel"]="Gasoline"

train.at[3855,"SellerZip"]=48124
train.at[3855,"SellerListSrc"]="Inventory Command Center"
train.at[3855,"VehFuel"]="Gasoline"
```

Two records are missing VehMileage and VehListdays, we will just fill those in with the means.

[10]:

```
train["VehMileage"].fillna(value=train["VehMileage"].mean(),inplace=True)
test["VehMileage"].fillna(value=test["VehMileage"].mean(),inplace=True)
train["VehListdays"].fillna(value=train["VehListdays"].mean(),inplace=True)
```

For the null VehColorExt VehColorInt VehDriveTrain VehEngine VehPriceLabel VehTransmission add an "unknown" category.

[11]:

```
for x in ["VehColorExt", "VehColorInt", "VehDriveTrain", "VehEngine", "VehPriceLabel", "VehTransmission"]:
    train[x].fillna(value="unknown",inplace=True)
    test[x].fillna(value="unknown",inplace=True)
```

For many names the first token is most informative with the remaining tokens just specifying the location. To capture sellers that have multiple locations, add a feature for the first token in a name.

[12]:

```
train["SellerShortName"]=train["SellerName"].str.lower().str.split(n=1,expand=True)[0]
test["SellerShortName"]=test["SellerName"].str.lower().str.split(n=1,expand=True)[0]
```

There is just one data point for HI and it doesn't show up in testing. HI is likely an outlier so it is removed.

[13]:

```
train = train[train["SellerState"]!= "HI"]
```

We add a feature that consolidates states in their Census Bureau–designated divisions.

[14]:
```
divisions = {"New England" : {"CT", "ME", "MA", "NH", "RI", "VT"},
            "Middle Atlantic" : {"NJ","NY","PA"},
             "South Atlantic" : {"MD","DC","DE","VA","WV","NC","SC","GA","FL"},
            "East North Central" : {"IL","IN","OH","MI","WI"},
             "East South Central" : {"KY","TN","MS","AL"},
            "West North Central" : {"ND","SD","NE","KS","MO","IA","MN"},
            "West South Central" : {"TX","LA","AR","OK"},
            "Mountain" : {"MT","ID","WY","NV","UT","CO","AZ","NM"},
            "Pacific" : {"AK","HI","WA","OR","CA"}}
state_to_division = dict([(state,division) for division in divisions.keys() for state in divisions[division]])

train["SellerDivision"] = train["SellerState"].apply(lambda state: state_to_division[state])
test["SellerDivision"] = test["SellerState"].apply(lambda state: state_to_division[state])
```

Normalize the labels for 4WD and FWD drivetrains.

[15]:
```
drivetrains = {
    "4WD" : ["4WD", "AWD", "4X4", "Four Wheel Drive", "ALL-WHEEL DRIVE", "All Wheel Drive",
             "4x4", "4x4/4-wheel drive", "4x4/4WD", "AWD or 4x4", "All-wheel Drive", "ALL WHEEL",
             "AllWheelDrive", "ALL-WHEEL DRIVE WITH LOCKING AND LIMITED-SLIP DIFFERENTIAL","4WD/AWD"],
    "FWD" : ["FWD","FRONT-WHEEL DRIVE", "Front Wheel Drive","Front-wheel Drive","2WD"],
    "unknown" : ["unknown"]
}
clean_drivetrain = dict([(raw,clean) for clean in drivetrains.keys() for raw in drivetrains[clean]])
train["VehDriveTrainClean"] = train["VehDriveTrain"].apply(lambda raw: clean_drivetrain[raw])
test["VehDriveTrainClean"] = test["VehDriveTrain"].apply(lambda raw: clean_drivetrain[raw])
```

[16]:
```
# Tokenize the engine strings
train["VehEngineTokens"] = train["VehEngine"].str.split()
test["VehEngineTokens"] = test["VehEngine"].str.split()
```

[17]:
```
# Extract the list of features from VehFeats and generate words and bigrams
nonalphanum = re.compile("[^a-zA-Z0-9_]")
def words_and_bigrams(s):
    words = list(nltk.word_tokenize(s))
    bigrams = list(nltk.bigrams(words))
    return [(w,) for w in words] + bigrams

def extract_feats(feats):
    if pd.isna(feats):
        return [("unknown",)]
    else:
        return [gram for feat in feats.strip("]['").split("', '") for gram in words_and_bigrams(nonalphanum.sub(' ', feat).strip())]
train["VehFeatTokens"] = train["VehFeats"].apply(extract_feats)
test["VehFeatTokens"] = test["VehFeats"].apply(extract_feats)
```

[18]:
```
# Extract words and bigrams from Seller notes
def extract_notes(notes):
    if pd.isna(notes):
        return [("unknown",)]
    else:
        return [gram for gram in words_and_bigrams(nonalphanum.sub(' ', notes).strip())]
train["VehSellerNotesTokens"] = train["VehSellerNotes"].apply(extract_notes)
test["VehSellerNotesTokens"] = test["VehSellerNotes"].apply(extract_notes)
```

[19]:
```
# As VehHistory is standardized we just extract the elements in the list
train["VehHistoryTokens"] = train["VehHistory"].fillna('unknown').str.split(', ')
test["VehHistoryTokens"] = test["VehHistory"].fillna('unknown').str.split(', ')
```

[20]:
```
# Extract words and bigrams from seller names
def extract_names(name):
    if pd.isna(name):
        return [("unknown",)]
    else:
        return [gram for gram in words_and_bigrams(nonalphanum.sub(' ', name).strip())]
train["SellerNameTokens"] = train["SellerName"].apply(extract_names)
test["SellerNameTokens"] = test["SellerName"].apply(extract_names)
```

This next feature cleans the VehEngine feature and attempts to extract it from VehFeats or VehSellerNotes if it is unknown.

[21]:
```
def extract_engine(row):
    if (not pd.isna(row["VehFeats"]) and "3.6" in row["VehFeats"]) or (not pd.isna(row["VehSellerNotes"]) and "3.6" in row["VehSellerNotes"]):
        return "3.6L V6"
    elif (not pd.isna(row["VehFeats"]) and "5.7" in row["VehFeats"]) or (not pd.isna(row["VehSellerNotes"]) and "5.7" in row["VehSellerNotes"]):
        return "5.7L V8"
    elif (not pd.isna(row["VehFeats"]) and "6.2" in row["VehFeats"]) or (not pd.isna(row["VehSellerNotes"]) and "6.2" in row["VehSellerNotes"]):
        return "6.2L V8"
    elif (not pd.isna(row["VehFeats"]) and "6.4" in row["VehFeats"]) or (not pd.isna(row["VehSellerNotes"]) and "6.4" in row["VehSellerNotes"]):
        return "6.4L V8"
    else:
        return row["VehEngine"]
```

```python
def clean_train_engine(row):
    if row["VehMake"] == "Cadillac":
        return "3.6L V6"
    elif row["VehFuel"]=="Diesel":
        return "3.0L V6"
    elif "3.6" in row["VehEngine"] or "V6" in row["VehEngine"] or row["VehEngine"] in ["6-cylinder","6","6 Cylinder","V-6 cyl"] or␣
↪row["Vehicle_Trim"] in ["Laredo", "Laredo E"]:
        return "3.6L V6"
    elif "6.4" in row["VehEngine"] or row["Vehicle_Trim"] in ["SRT","SRT8"]:
        return "6.4L V8"
    elif "6.2" in row["VehEngine"] or "Trackhawk" == row["Vehicle_Trim"]:
        return "6.2L V8"
    elif "5.7" in row["VehEngine"] or ("8" in row["VehEngine"] and not (row["Vehicle_Trim"] in ["SRT","SRT8","Trackhawk"])):
        return "5.7L V8"
    else:
        return extract_engine(row)
def clean_test_engine(row):
    if row["VehMake"] == "Cadillac":
        return "3.6L V6"
    elif row["VehFuel"]=="Diesel":
        return "3.0L V6"
    elif "3.6" in row["VehEngine"] or "V6" in row["VehEngine"] or row["VehEngine"] in ["6-cylinder","6","6 Cylinder","V-6 cyl"]:
        return "3.6L V6"
    elif "6.2" in row["VehEngine"]:
        return "6.2L V8"
    elif "6.4" in row["VehEngine"] or row["VehEngine"]=="8-cylinder": # the one "8-cylinder" has SRT in notes
        return "6.4L V8"
    elif "5.7" in row["VehEngine"] or row["VehEngine"]=="8 Cylinder Engine": # the one "8 Cylinder Engine" has uninformative features, rather␣
↪than label it "unknown" I label it the most common V8 engine type
        return "5.7L V8"
    elif row["VehEngine"]=="0":
        return "unknown"
    else:
        return extract_engine(row)

train["VehEngineClean"] = train.apply(clean_train_engine,axis=1)
test["VehEngineClean"] = test.apply(clean_test_engine,axis=1)
```

[22]:
```python
# A simple attempt to extract a trim label from VehFeats or VehSellerNotes
def extract_trim(row):
    if not pd.isnull(row["VehSellerNotes"]):
        if row["VehMake"]=="Cadillac":
            if "platinum" in row["VehSellerNotes"]:
                return "Platinum"
            elif "premium luxury" in row["VehSellerNotes"]:
                return "Premium Luxury"
            elif "luxury" in row["VehSellerNotes"]:
                return "Luxury"
            elif "base" in row["VehSellerNotes"]:
                return "Base"
        else:
            if "laredo e" in row["VehSellerNotes"]:
                return "Laredo E"
            elif "laredo" in row["VehSellerNotes"]:
                return "Laredo E"
            elif "overland" in row["VehSellerNotes"]:
                return "Overland"
            elif "high altitude" in row["VehSellerNotes"]:
                return "High Altitude"
            elif "altitude" in row["VehSellerNotes"]:
                return "Altitude"
            elif "summit" in row["VehSellerNotes"]:
                return "Summit"
            elif "trailhawk" in row["VehSellerNotes"]:
                return "Trailhawk"
            elif "srt" in row["VehSellerNotes"]:
                return "SRT"
            elif "trackhawk" in row["VehSellerNotes"]:
                return "Trackhawk"
            elif "sterling edition" in row["VehSellerNotes"]:
                return "Sterling Edition"
            elif "upland" in row["VehSellerNotes"]:
                return "Upland"
            elif "limited" in row["VehSellerNotes"]:
                return "Limited"
    elif not pd.isnull(row["VehFeats"]):
        if row["VehMake"]=="Cadillac":
            if "platinum" in row["VehFeats"]:
                return "Platinum"
            elif "premium luxury" in row["VehFeats"]:
                return "Premium Luxury"
            elif "luxury" in row["VehFeats"]:
                return "Luxury"
            elif "base" in row["VehFeats"]:
                return "Base"
        else:
            if "laredo e" in row["VehFeats"]:
                return "Laredo E"
            elif "laredo" in row["VehFeats"]:
                return "Laredo E"
```

```
            elif "overland" in row["VehFeats"]:
                return "Overland"
            elif "high altitude" in row["VehFeats"]:
                return "High Altitude"
            elif "altitude" in row["VehFeats"]:
                return "Altitude"
            elif "summit" in row["VehFeats"]:
                return "Summit"
            elif "trailhawk" in row["VehFeats"]:
                return "Trailhawk"
            elif "srt" in row["VehFeats"]:
                return "SRT"
            elif "trackhawk" in row["VehFeats"]:
                return "Trackhawk"
            elif "sterling edition" in row["VehFeats"]:
                return "Sterling Edition"
            elif "upland" in row["VehFeats"]:
                return "Upland"
            elif "limited" in row["VehFeats"]:
                return "Limited"
    return "Unknown"
train["Vehicle_Trim_Extraction"] = train.apply(extract_trim,axis=1)
test["Vehicle_Trim_Extraction"] = test.apply(extract_trim,axis=1)
```

[23]:

```
# An agressively normalized version of trim labels
def clean_trim(row):
    if pd.isnull(row["Vehicle_Trim"]):
        return row["Vehicle_Trim"]
    elif row["VehMake"]=="Cadillac":
        if row["Vehicle_Trim"] in ["Premium Luxury","Premium Luxury AWD","Premium Luxury FWD"]:
            return "Premium Luxury"
        elif row["Vehicle_Trim"] in ["Luxury","Luxury AWD","Luxury FWD"]:
            return "Luxury"
        elif row["Vehicle_Trim"] in ["Platinum","Platinum AWD"]:
            return "Platinum"
        else:
            return "Base"
    else:
        if row["Vehicle_Trim"] in ["Limited","75th Anniversary","Limited 75th Anniversary Edition","Limited 4x4","75th Anniversary␣
↪Edition","Limited X","Limited 75th Anniversary"]:
            return "Limited"
        elif row["Vehicle_Trim"] in ["SRT", "SRT Night"]:
            return "SRT"
        else:
            return row["Vehicle_Trim"]
train["Vehicle_Trim_Clean"] = train.apply(clean_trim,axis=1)
```

[24]:

```
# Replace trim levels that are used only once
train["Vehicle_Trim"].value_counts()
def clean_trim(row):
    if pd.isnull(row["Vehicle_Trim"]):
        return row["Vehicle_Trim"]
    elif row["VehMake"]=="Cadillac":
        return row["Vehicle_Trim"]
    else:
        if row["Vehicle_Trim"] in ["Limited X", "Limited 4x4"]:
            return "Limited"
        elif row["Vehicle_Trim"] in ["Limited 75th Anniversary","75th Anniversary Edition"]:
            return "75th Anniversary"
        else:
            return row["Vehicle_Trim"]
train["Vehicle_Trim_Semiclean"] = train.apply(clean_trim,axis=1)
```

[25]:

```
# normalize unknown color labeld
def clean_ext_color(color):
    if color == "undetermined" or color == "unspecified":
        return "unknown"
    else:
        return color
train["VehColorExt"] = train["VehColorExt"].apply(clean_ext_color)
test["VehColorExt"] = test["VehColorExt"].apply(clean_ext_color)
```

[26]:

```
# Extract words and bigrams from colors
def extract_color(color):
    if pd.isna(color):
        return [("unknown",)]
    else:
        return [gram for gram in words_and_bigrams(nonalphanum.sub(' ', color).strip())]
train["VehColorExtTokens"] = train["VehColorExt"].apply(extract_color)
test["VehColorExtTokens"] = test["VehColorExt"].apply(extract_color)
train["VehColorIntTokens"] = train["VehColorInt"].apply(extract_color)
test["VehColorIntTokens"] = test["VehColorInt"].apply(extract_color)
```

[27]:

```
# Encode features for model training
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder, MultiLabelBinarizer,StandardScaler
```

```
real_features = ["SellerRating","SellerRevCnt","VehListdays","VehMileage"]
already_encoded_features = ["SellerIsPriv","VehCertified"]
one_hot_encoded_features =␣
↪["SellerListSrc","SellerState","SellerDivision","VehDriveTrainClean","VehEngineClean","VehFuel","VehMake","VehPriceLabel","VehYear","Vehicle_Trim_Extraction"]
mlb_encoded_features =␣
↪["SellerNameTokens","VehEngineTokens","VehFeatTokens","VehSellerNotesTokens","VehHistoryTokens","VehColorExtTokens","VehColorIntTokens"]

# Scale real valued features
scaler = StandardScaler().fit(pd.concat([train[real_features],test[real_features]],ignore_index=True))
X_train = pd.DataFrame(scaler.transform(train[real_features]), columns = real_features,index=train.index)
X_predict = pd.DataFrame(scaler.transform(test[real_features]), columns = real_features,index=test.index)


# One hot encode categorical variables
def custom_combiner(feature, category):
    return str(feature) + "_" + str(category)
ohe = OneHotEncoder(feature_name_combiner=custom_combiner,handle_unknown='ignore',sparse_output=False).fit(train[one_hot_encoded_features])
temp = pd.DataFrame(ohe.transform(train[one_hot_encoded_features]),index=train.index,columns=ohe.get_feature_names_out())
X_train = X_train.join(temp)

temp_predict = pd.DataFrame(ohe.transform(test[one_hot_encoded_features]),index=test.index,columns=ohe.get_feature_names_out())
X_predict = X_predict.join(temp_predict)

# Encode lists of features using a multi label binarizer
for feature in mlb_encoded_features:
    mlb = MultiLabelBinarizer(sparse_output=True).fit(pd.concat([train[feature],test[feature]],ignore_index=True))
    temp = pd.DataFrame.sparse.from_spmatrix(mlb.transform(train[feature]),columns=[feature+str(c) for c in mlb.classes_],index=train.index)
    temp = temp.loc[:,temp.sum()>=5] # drop features that don't occur in at least 5 samples in training
    X_train = X_train.join(temp)

    temp_predict = pd.DataFrame.sparse.from_spmatrix(mlb.transform(test[feature]),columns=[feature+str(c) for c in mlb.classes_],index=test.
↪index)
    temp_predict = temp_predict.loc[:,temp.loc[:,temp.sum()>=5].columns] # drop features that don't occur in at least 5 samples in training
    X_predict = X_predict.join(temp_predict)

Y_train = train["Vehicle_Trim"]
Y_train_clean = train["Vehicle_Trim_Clean"]
Y_train_semiclean = train["Vehicle_Trim_Semiclean"]
```

[28]:
```
# Dataset for trim prediction
X_train_trim = X_train[~Y_train.isnull()]
Y_train_trim = Y_train[~Y_train.isnull()]
Y_train_clean_trim = Y_train_clean[~Y_train.isnull()]
Y_train_semiclean_trim = Y_train_semiclean[~Y_train.isnull()]
```

[29]:
```
# Dataset for price prediction
X_train_price = X_train[~train["Dealer_Listing_Price"].isnull()]
feature_scaler = StandardScaler().fit(X_train)
X_train_price_normalized = pd.DataFrame(feature_scaler.transform(X_train_price), columns = X_train_price.columns,index=X_train_price.index)
X_predict_normalized = pd.DataFrame(feature_scaler.transform(X_predict), columns = X_predict.columns,index=X_predict.index)

Y_train_price = train[~train["Dealer_Listing_Price"].isnull()]["Dealer_Listing_Price"]
price_scaler = StandardScaler().fit(Y_train_price.to_numpy().reshape(-1, 1))
Y_train_price_normalized =pd.Series(price_scaler.transform(Y_train_price.to_numpy().reshape(-1, 1)).reshape(1,-1)[0], index=Y_train_price.index)
```

```
/Users/hazeneckert/miniconda3/envs/practice/lib/python3.11/site-
packages/sklearn/utils/validation.py:787: UserWarning: pandas.DataFrame with
sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(
/Users/hazeneckert/miniconda3/envs/practice/lib/python3.11/site-
packages/sklearn/utils/validation.py:787: UserWarning: pandas.DataFrame with
sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(
/Users/hazeneckert/miniconda3/envs/practice/lib/python3.11/site-
packages/sklearn/utils/validation.py:787: UserWarning: pandas.DataFrame with
sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(
```

5 Model Training and Prediciton

See the section on model selection to see why these models and parameters were chosen.

[31]:
```
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(max_depth=18)
rf_clf.fit(X_train_trim, Y_train_semiclean_trim)
predict = pd.DataFrame(rf_clf.predict(X_predict),columns=["Vehicle_Trim_Predicted"]).join(test["ListingID"])
```

```
/Users/hazeneckert/miniconda3/envs/practice/lib/python3.11/site-
packages/sklearn/utils/validation.py:787: UserWarning: pandas.DataFrame with
sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(
/Users/hazeneckert/miniconda3/envs/practice/lib/python3.11/site-
packages/sklearn/utils/validation.py:787: UserWarning: pandas.DataFrame with
sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(
```

[32]:
```
from sklearn.ensemble import RandomForestRegressor
rf_reg = RandomForestRegressor(n_jobs=-1)
rf_reg.fit(X_train_price_normalized,Y_train_price_normalized)
predict =pd.DataFrame(price_scaler.inverse_transform(rf_reg.predict(X_predict_normalized).reshape(-1, 1)), columns=["Predicted_price"],
↪index=X_predict_normalized.index).join(predict)
```

[33]:
```
predict[["ListingID","Vehicle_Trim_Predicted","Predicted_price"]].to_csv("predictions.csv",index=False)
```

6 Model Selection and Evaluation

I first quickly evaluate the preformance of a collection of models on the data. I select the most promising one for each task and find hyperparameters for the simplest model within a std dev of the optimal score.

[ ]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score
from sklearn.ensemble import AdaBoostRegressor, RandomForestRegressor, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
regressors = [AdaBoostRegressor(),
              RandomForestRegressor(n_jobs=-1),
              GradientBoostingRegressor(),
              KNeighborsRegressor(3,n_jobs=-1),
              Lasso(alpha=0.1)]

train_ind,test_ind= next(KFold(n_splits=5).split(X_train_price, Y_train_price_normalized))
for reg in regressors:
    reg.fit(X_train_price_normalized.iloc[train_ind], Y_train_price_normalized.iloc[train_ind])

    print("-"*50)
    print(reg.__class__.__name__)

    print('****Results****')
    r2 = r2_score(Y_train_price_normalized.iloc[test_ind],reg.predict(X_train_price_normalized.iloc[test_ind]))
    print("R2: {:.4%}".format(r2))
```

RandomForestRegressor performs the best.

[ ]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import RocCurveDisplay, roc_curve, roc_auc_score
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
classifiers = [
    KNeighborsClassifier(3),
    RandomForestClassifier(),
    AdaBoostClassifier(),
    GradientBoostingClassifier(),
    GaussianNB(),
    LinearDiscriminantAnalysis(),
    QuadraticDiscriminantAnalysis()]


train_ind,test_ind = next(StratifiedKFold(n_splits=3).split(X_train_trim, Y_train_clean_trim))


for clf in classifiers:
    clf.fit(X_train_trim.iloc[train_ind], Y_train_clean_trim.iloc[train_ind])

    print("-"*50)
    print(clf.__class__.__name__)

    print('****Results****')
    auc = roc_auc_score(Y_train_clean_trim.iloc[test_ind],clf.predict_proba(X_train_trim.iloc[test_ind]), multi_class='ovr')
    print("AUC: {:.4%}".format(auc))
```

The default RandomForestClassifier works extremely well and is very quick to train so I will used a tuned version of it for the trim classifier.

[ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
parameters = {'criterion' : ("gini", "entropy", "log_loss"),
             'max_depth' :[5, 10, 25, 50]
             }
rf = RandomForestClassifier()
clf = GridSearchCV(rf, parameters,cv=3,verbose=3, scoring='roc_auc_ovr',n_jobs=-1)
res = clf.fit(X_train_trim, Y_train_semiclean_trim)
res.cv_results_
```

The criterion doesn't affect performance much and the best depth is 10. I am going to select the gini criterion and explore depth more.

[ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
parameters = {'criterion' : ["gini"],
```

```
            'max_depth' :[8, 12, 15, 18, 20, 22]
            }
clf = GridSearchCV(rf, parameters,cv=3,verbose=3, scoring='roc_auc_ovr',n_jobs=-1)
res = clf.fit(X_train_trim, Y_train_semiclean_trim)
res.cv_results_
```

The score is maximized at 20 but 18 is within a std dev so it will be chosen.