**Car Trim and Price Prediction Project**

By Roger Qiu

Objective

Given a training dataset of vehicles sold at different dealerships, accurately predict a vehicles trim and price given numerous features in a test set using machine learning models.

EDA and Feature Transformation

Based on initial EDA, fields with only 1 unique value were removed. Listing ID was dropped from features, these fields do not provide useful info. Dataset is then split into features set and the 2 labels fields.

The dataset included a variety of feature types, each needed different preprocessing methods to convert the fields into numerical values for the models:

1. **Numerical Features**: scaled and imputed using normalization to ensure no nulls and that they contributed equally to the model.
2. **Categorical Features**: One-hot encoding was applied to convert categorical features into a numerical format.
3. **Boolean Features**: transformed into binary 0 and 1 values.
4. **List Features (VehFeats)**: converted into multi-hot encoded features where the presence of each feature was encoded as a separate binary feature.
5. **Text Features (VehSellerNotes)**: Used TF-IDF vectorization to convert text into tokens, remove unnecessary stop words and then into numerical features as a matrix.

Dimensionality Reduction with PCA and splitting

After the preprocessing steps above, the feature set had high dimensionality, with 4909 features. To fix this issue and increase efficiency, Principal Component Analysis was used to reduce the number of features to those that explained 95% of the variance, resulting in 788 new fields. This transformation ensured that the essential information was retained while greatly reducing the number of fields. Records are then split into 70/30 for training and validation of model’s performance.

Model Selection and Tuning

Two separate sets models were trained and validated: classification models for predicting trim and regression models for price. Best models (highest accuracy and F1, lowest RMSE and highest R^2) for each were then optimized with GridsearchCV hyperparameter tuning to find optimal parameters.

1. **Predicting Vehicle Trim**:
   * **Best Model**: Logistic Regression
   * **Best Hyperparameters**: C = 10, max\_iter = 1000, solver = ‘liblinear’
   * **Performance**: Achieved an accuracy and F1 score of 86%.
2. **Predicting Dealer Listing Price**:
   * **Model**: Linear (Ridge) Regression
   * **Best Hyperparameters**: alpha = 1, solver = ‘saga’
   * **Performance**: Achieved an RMSE of 2917 and an R² score of 86%.

Test data pre-processing, predictions and write-out

# Using the same pre-processing techniques as earlier, we apply them to the test set. After PCA, the test set should have the same number of fields: 788. The test set is then fed into optimized logistic regression model and the linear (ridge) regression. The predictions are created, for the trim, the prediction results are numerical so they must be then un-encoded to product the original text trim values. These results are appended back to the original dataset, only the 3 required fields are kept, and the results are written out. Finally, spot checks on the output and visualizations were performed to ensure the predictions seemed reasonable

# Appendix

# Car Trim and Price Prediction

### By Roger Qiu

Objective: Given a training dataset of vehicles sold at different dealerships, accurately predict a vehicles trim and price given numerous features in a test set.

### import libraries

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score, cross\_val\_predict  
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, LabelEncoder, StandardScaler, Normalizer  
from sklearn.metrics import \*  
from sklearn.linear\_model import \*  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.model\_selection import GridSearchCV  
import random  
from sklearn.impute import SimpleImputer  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.ensemble import \*  
from sklearn.naive\_bayes import \*  
from sklearn.decomposition import \*  
from sklearn.tree import \*  
from sklearn.svm import \*  
from sklearn.neighbors import \*  
  
# show all columns  
pd.set\_option('display.max\_columns', None)  
  
# ignore all warnings  
import warnings  
warnings.filterwarnings('ignore')  
  
# download stopwords from NLTK  
from wordcloud import WordCloud  
import matplotlib.pyplot as plt  
from nltk.corpus import stopwords  
import nltk  
  
# set random to 42 to repeat results  
random.seed(42)  
np.random.seed(42)

### EDA

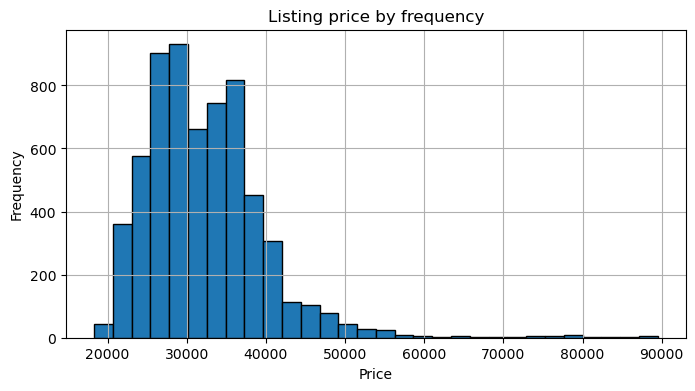
train\_data = pd.read\_csv('Training\_DataSet.csv')  
train\_data.head()

ListingID SellerCity SellerIsPriv SellerListSrc \  
0 3287 Warren False Inventory Command Center   
1 3920 Fargo False Cadillac Certified Program   
2 4777 Waukesha False Jeep Certified Program   
3 6242 Wentzville False Inventory Command Center   
4 7108 Fayetteville False HomeNet Automotive   
  
 SellerName SellerRating SellerRevCnt \  
0 Prime Motorz 5.0 32   
1 Gateway Chevrolet Cadillac 4.8 1456   
2 Wilde Chrysler Jeep Dodge Ram &amp; Subaru 4.8 1405   
3 Century Dodge Chrysler Jeep RAM 4.4 21   
4 Superior Buick GMC of Fayetteville 3.7 74   
  
 SellerState SellerZip VehBodystyle VehCertified \  
0 MI 48091.0 SUV False   
1 ND 58103.0 SUV True   
2 WI 53186.0 SUV True   
3 MO 63385.0 SUV False   
4 AR 72703.0 SUV False   
  
 VehColorExt VehColorInt VehDriveTrain \  
0 White Black 4X4   
1 Black NaN NaN   
2 Brilliant Black Crystal Pearlcoat Black 4x4/4WD   
3 Diamond Black Crystal Pearlcoat Black 4WD   
4 Radiant Silver Metallic Cirrus FWD   
  
 VehEngine \  
0 3.6L V6   
1 NaN   
2 Regular Unleaded V-6 3.6 L/220   
3 3.6L V6   
4 Gas V6 3.6L/222.6   
  
 VehFeats VehFuel \  
0 ['Adaptive Cruise Control', 'Antilock Brakes',... Gasoline   
1 NaN Gasoline   
2 ['18 WHEEL &amp; 8.4 RADIO GROUP-inc: Nav-Capa... Gasoline   
3 ['Android Auto', 'Antilock Brakes', 'Apple Car... Gasoline   
4 ['4-Wheel Disc Brakes', 'ABS', 'Adjustable Ste... Gasoline   
  
 VehHistory VehListdays VehMake \  
0 1 Owner, Non-Personal Use Reported, Buyback Pr... 8.600069 Jeep   
1 1 Owner, Buyback Protection Eligible 2.920127 Cadillac   
2 1 Owner, Buyback Protection Eligible 28.107014 Jeep   
3 1 Owner, Non-Personal Use Reported, Buyback Pr... 59.816875 Jeep   
4 1 Owner, Non-Personal Use Reported, Buyback Pr... 98.665301 Cadillac   
  
 VehMileage VehModel VehPriceLabel \  
0 39319.0 Grand Cherokee Fair Price   
1 30352.0 XT5 Good Deal   
2 38957.0 Grand Cherokee Good Deal   
3 20404.0 Grand Cherokee Good Deal   
4 19788.0 XT5 Good Deal   
  
 VehSellerNotes VehType \  
0 NaN Used   
1 Come take a look at our great pre-owned invent... Used   
2 Backed by a rigorous 125-point inspection by f... Used   
3 Drop by to see us and you will quickly see how... Used   
4 Luxury, Exterior Parking Camera Rear, Front Du... Used   
  
 VehTransmission VehYear Vehicle\_Trim Dealer\_Listing\_Price   
0 Automatic 8-Speed 2015 High Altitude 30990.0   
1 NaN 2017 NaN 34860.0   
2 8-Speed Automatic w/OD 2015 Laredo 23249.0   
3 Automatic 2018 Limited 31977.0   
4 8-Speed Automatic 2018 Luxury 33495.0

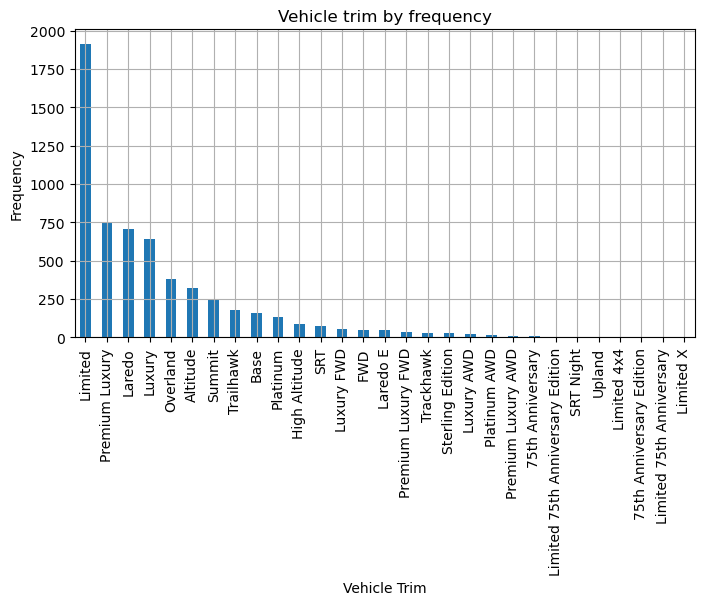
# 29 fields, 6298 records  
print(train\_data.shape)

(6298, 29)

# histogram of Dealer\_Listing\_Price  
plt.figure(figsize=(8, 4))  
plt.hist(train\_data['Dealer\_Listing\_Price'], bins=30, edgecolor='black')  
plt.title('Listing price by frequency')  
plt.xlabel('Price')  
plt.ylabel('Frequency')  
plt.grid(True)  
plt.show()



# bar chart of Vehicle\_Trim  
plt.figure(figsize=(8, 4))  
train\_data['Vehicle\_Trim'].value\_counts().plot(kind='bar')  
plt.title('Vehicle trim by frequency')  
plt.xlabel('Vehicle Trim')  
plt.ylabel('Frequency')  
plt.grid(True)  
plt.show()



# check nulls  
  
# count the null values in each field, print where more than 0 nulls  
null\_counts = train\_data.isnull().sum()  
print(null\_counts[null\_counts > 0])

SellerListSrc 2  
SellerZip 2  
VehColorExt 73  
VehColorInt 728  
VehDriveTrain 401  
VehEngine 361  
VehFeats 275  
VehFuel 2  
VehHistory 201  
VehListdays 2  
VehMileage 2  
VehPriceLabel 285  
VehSellerNotes 243  
VehTransmission 197  
Vehicle\_Trim 405  
Dealer\_Listing\_Price 52  
dtype: int64

### pre-processing

# field columns with only one unique value  
columns\_to\_remove = [col for col in train\_data.columns if train\_data[col].nunique() == 1]  
columns\_to\_remove

['VehBodystyle', 'VehType']

# remove these 2 field from the dataset, they provide no additional info  
train\_data = train\_data.drop(columns=columns\_to\_remove)  
train\_data.shape

(6298, 27)

# drop records with null Vehicle\_Trim or Dealer\_Listing\_Price, these are our labels which we need values in  
train\_data = train\_data.dropna(subset=['Vehicle\_Trim', 'Dealer\_Listing\_Price'])  
  
# count the null values in each field, print where more than 0 nulls  
null\_counts = train\_data.isnull().sum()  
print(null\_counts[null\_counts > 0])

SellerListSrc 2  
SellerZip 2  
VehColorExt 42  
VehColorInt 426  
VehDriveTrain 68  
VehEngine 28  
VehFeats 23  
VehFuel 2  
VehHistory 197  
VehListdays 2  
VehPriceLabel 233  
VehSellerNotes 74  
VehTransmission 32  
dtype: int64

# instead of 6298 records, now its 5841  
print(train\_data.shape)

(5841, 27)

# create features by dropping id field and labels  
X\_train = train\_data.drop(columns=['Vehicle\_Trim', 'Dealer\_Listing\_Price', 'ListingID'])  
print(X\_train.shape)

(5841, 24)

# create the 2 labels fields  
y\_trim\_train = train\_data['Vehicle\_Trim']  
y\_price\_train = train\_data['Dealer\_Listing\_Price']  
  
print(y\_trim\_train.head())  
print(y\_price\_train.head())

0 High Altitude  
2 Laredo  
3 Limited  
4 Luxury  
5 Limited  
Name: Vehicle\_Trim, dtype: object  
0 30990.0  
2 23249.0  
3 31977.0  
4 33495.0  
5 27900.0  
Name: Dealer\_Listing\_Price, dtype: float64

# check feature field types  
X\_train.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 5841 entries, 0 to 6297  
Data columns (total 24 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 SellerCity 5841 non-null object   
 1 SellerIsPriv 5841 non-null bool   
 2 SellerListSrc 5839 non-null object   
 3 SellerName 5841 non-null object   
 4 SellerRating 5841 non-null float64  
 5 SellerRevCnt 5841 non-null int64   
 6 SellerState 5841 non-null object   
 7 SellerZip 5839 non-null float64  
 8 VehCertified 5841 non-null bool   
 9 VehColorExt 5799 non-null object   
 10 VehColorInt 5415 non-null object   
 11 VehDriveTrain 5773 non-null object   
 12 VehEngine 5813 non-null object   
 13 VehFeats 5818 non-null object   
 14 VehFuel 5839 non-null object   
 15 VehHistory 5644 non-null object   
 16 VehListdays 5839 non-null float64  
 17 VehMake 5841 non-null object   
 18 VehMileage 5841 non-null float64  
 19 VehModel 5841 non-null object   
 20 VehPriceLabel 5608 non-null object   
 21 VehSellerNotes 5767 non-null object   
 22 VehTransmission 5809 non-null object   
 23 VehYear 5841 non-null int64   
dtypes: bool(2), float64(4), int64(2), object(16)  
memory usage: 1.0+ MB

# select all the groups of fields by type: numerical, categorical, boolean, text and list features  
num\_cols = X\_train.select\_dtypes(include=['int64', 'float64']).columns  
cat\_cols = X\_train.select\_dtypes(include=['object']).columns.difference(['VehSellerNotes','VehFeats'])  
bool\_cols = X\_train.select\_dtypes(include=['bool']).columns  
text\_col = 'VehSellerNotes'  
list\_col = 'VehFeats'  
  
print(num\_cols)  
print(cat\_cols)  
print(bool\_cols)  
print(text\_col)  
print(list\_col)  
  
# still 24 fields  
print(len(num\_cols) + len(cat\_cols) + len(bool\_cols) + 1 + 1)

Index(['SellerRating', 'SellerRevCnt', 'SellerZip', 'VehListdays',  
 'VehMileage', 'VehYear'],  
 dtype='object')  
Index(['SellerCity', 'SellerListSrc', 'SellerName', 'SellerState',  
 'VehColorExt', 'VehColorInt', 'VehDriveTrain', 'VehEngine', 'VehFuel',  
 'VehHistory', 'VehMake', 'VehModel', 'VehPriceLabel',  
 'VehTransmission'],  
 dtype='object')  
Index(['SellerIsPriv', 'VehCertified'], dtype='object')  
VehSellerNotes  
VehFeats  
24

### pre-process numerical features

# for numerical features, impute missing values and scale them  
num\_imputer = SimpleImputer(strategy='median')  
scaler = StandardScaler()  
  
# fit on numerical fields for training data  
X\_train\_num = num\_imputer.fit\_transform(X\_train[num\_cols])  
X\_train\_num = scaler.fit\_transform(X\_train\_num)  
  
print(X\_train\_num[:5])  
print(X\_train\_num.shape)

[[ 0.73661031 -0.32105247 0.14769339 -0.6983788 0.97497474 -1.45127235]  
 [ 0.56213936 0.72258538 0.40213716 -0.41007283 0.94719888 -1.45127235]  
 [ 0.21319747 -0.32941374 0.91147418 0.05858808 -0.47635222 1.01844168]  
 [-0.39745085 -0.28912764 1.37681414 0.63275454 -0.52361722 1.01844168]  
 [-1.00809917 -0.30661029 -1.51685333 -0.36779582 0.61665082 1.01844168]]  
(5841, 6)

### pre-process categorical features

# impute missing values with 'missing' and one hot encode categorical features  
cat\_imputer = SimpleImputer(strategy='constant', fill\_value='missing')  
encoder = OneHotEncoder(handle\_unknown='ignore', sparse=False)  
  
# fit on categorical training data  
X\_train\_cat = cat\_imputer.fit\_transform(X\_train[cat\_cols])  
X\_train\_cat = encoder.fit\_transform(X\_train\_cat)  
  
print(X\_train\_cat[:5])  
print(X\_train\_cat.shape)

[[0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]]  
(5841, 4004)

### pre-process boolean features

# preprocess boolean features, convert true or false 1 or 0  
X\_train\_bool = X\_train[bool\_cols].astype(int).values  
  
print(X\_train\_bool[:5])  
print(X\_train\_bool.shape)

[[0 0]  
 [0 1]  
 [0 0]  
 [0 0]  
 [0 0]]  
(5841, 2)

### pre-process seller notes text

# check values  
X\_train[text\_col].head()

0 NaN  
2 Backed by a rigorous 125-point inspection by f...  
3 Drop by to see us and you will quickly see how...  
4 Luxury, Exterior Parking Camera Rear, Front Du...  
5 Priced below KBB Fair Purchase Price! Clean CA...  
Name: VehSellerNotes, dtype: object

nltk.download('stopwords')  
  
# remove stopwords and concatenate all text  
stop\_words = set(stopwords.words('english'))  
print(list(stop\_words)[:5])

["needn't", 'between', 'in', 'can', 'when']

[nltk\_data] Downloading package stopwords to  
[nltk\_data] /Users/roger.qiu/nltk\_data...  
[nltk\_data] Package stopwords is already up-to-date!

# full nulls with empty strings  
X\_train[text\_col] = X\_train[text\_col].fillna('')  
X\_train[text\_col].head()

0   
2 Backed by a rigorous 125-point inspection by f...  
3 Drop by to see us and you will quickly see how...  
4 Luxury, Exterior Parking Camera Rear, Front Du...  
5 Priced below KBB Fair Purchase Price! Clean CA...  
Name: VehSellerNotes, dtype: object

# combine all text data in a single string  
text\_data = ' '.join(X\_train[text\_col])  
text\_data[:50]

' Backed by a rigorous 125-point inspection by fact'

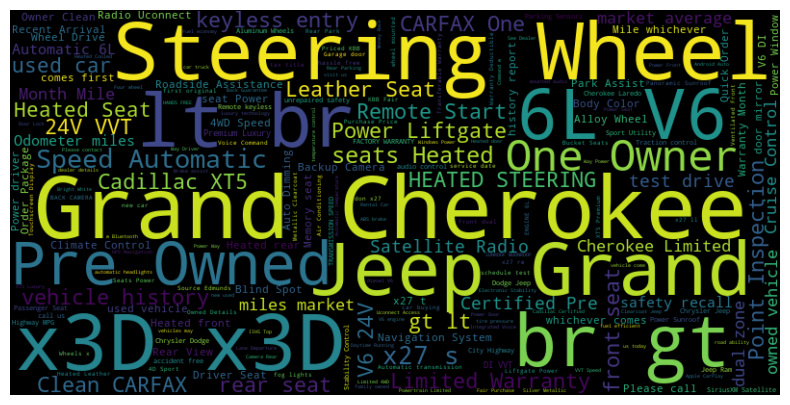
# remove stopwords  
filtered\_words = [word for word in text\_data.split() if word.lower() not in stop\_words]  
filtered\_words[:5]

['Backed', 'rigorous', '125-point', 'inspection', 'factory-trained']

# combine again into a single string  
filtered\_text = ' '.join(filtered\_words)  
filtered\_text[:50]

'Backed rigorous 125-point inspection factory-train'

# create and display the word cloud  
wordcloud = WordCloud(width=800, height=400, background\_color='black').generate(filtered\_text)  
plt.figure(figsize=(10, 5))  
plt.imshow(wordcloud, interpolation='bilinear')  
plt.axis('off')  
plt.show()



# create tfidf to convert text values into vectors  
# remove stopwords that dont provide info and use 100 of the top ranked terms across the corpus  
vectorizer = TfidfVectorizer(stop\_words='english', max\_features=100)  
  
# fit the vectorizer to the text column  
X\_train\_text = vectorizer.fit\_transform(X\_train[text\_col].fillna('')).toarray()  
  
print(X\_train\_text[2])  
print(X\_train\_text.shape)

[0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0.74619032 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0.66573269 0. 0. ]  
(5841, 100)

### pre-process vehicle features list

# check values  
X\_train[list\_col].head()

0 ['Adaptive Cruise Control', 'Antilock Brakes',...  
2 ['18 WHEEL &amp; 8.4 RADIO GROUP-inc: Nav-Capa...  
3 ['Android Auto', 'Antilock Brakes', 'Apple Car...  
4 ['4-Wheel Disc Brakes', 'ABS', 'Adjustable Ste...  
5 ['1st and 2nd row curtain head airbags', '4-wh...  
Name: VehFeats, dtype: object

# fill nulls, convert list to text  
X\_train[list\_col] = X\_train[list\_col].fillna('[]').apply(eval).apply(lambda x: ','.join(x))  
X\_train[list\_col].head()

0 Adaptive Cruise Control,Antilock Brakes,Audio ...  
2 18 WHEEL &amp; 8.4 RADIO GROUP-inc: Nav-Capabl...  
3 Android Auto,Antilock Brakes,Apple CarPlay,Aud...  
4 4-Wheel Disc Brakes,ABS,Adjustable Steering Wh...  
5 1st and 2nd row curtain head airbags,4-wheel A...  
Name: VehFeats, dtype: object

# use one hot encoder to convert comma seperated strings into a matrix with 1 or 0 for if the feature is present  
feats\_encoder = OneHotEncoder(handle\_unknown='ignore', sparse=False)  
X\_train\_feats = feats\_encoder.fit\_transform(X\_train[list\_col].values.reshape(-1, 1))  
  
print(X\_train\_feats[:5])  
print(X\_train\_feats.shape)

[[0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]]  
(5841, 797)

### combine features then use PCA to reduce dimentionality

# combine all preprocessed features together using hstack to horizontally concat the arrays  
X\_train\_preprocessed = np.hstack((X\_train\_num, X\_train\_cat, X\_train\_bool, X\_train\_text, X\_train\_feats))  
  
print(X\_train\_preprocessed[:1])  
print(X\_train\_preprocessed.shape)

[[ 0.73661031 -0.32105247 0.14769339 ... 0. 0.  
 0. ]]  
(5841, 4909)

# almost 5000 fields, so use principal component analysis to retain 95% of variance  
pca = PCA(n\_components=0.95, random\_state=42)  
X\_train\_pca = pca.fit\_transform(X\_train\_preprocessed)  
  
# only 788 fields now  
print(X\_train\_preprocessed.shape[1])  
print(X\_train\_pca.shape[1])

4909  
788

# check how the records look now  
print(X\_train\_preprocessed[0][:5])  
print(X\_train\_pca[0][:5])

[ 0.73661031 -0.32105247 0.14769339 -0.6983788 0.97497474]  
[-1.37042827 0.24928243 -0.69933417 0.31303048 -0.11801419]

### preprocess trim label

print(y\_trim\_train.head())  
print(y\_price\_train.head())

0 High Altitude  
2 Laredo  
3 Limited  
4 Luxury  
5 Limited  
Name: Vehicle\_Trim, dtype: object  
0 30990.0  
2 23249.0  
3 31977.0  
4 33495.0  
5 27900.0  
Name: Dealer\_Listing\_Price, dtype: float64

# encode categorical target variable y\_trim\_train  
label\_encoder = LabelEncoder()  
y\_trim\_train\_encoded = label\_encoder.fit\_transform(y\_trim\_train)  
  
print(y\_trim\_train\_encoded[:5])  
print(y\_trim\_train\_encoded.shape)

[ 5 6 8 13 8]  
(5841,)

# create a mapping from the original labels to the encoded numbers  
trim\_mapping = dict(zip(label\_encoder.classes\_, label\_encoder.transform(label\_encoder.classes\_)))  
  
for trim, number in trim\_mapping.items():  
 print(f"{trim}: {number}")

75th Anniversary: 0  
75th Anniversary Edition: 1  
Altitude: 2  
Base: 3  
FWD: 4  
High Altitude: 5  
Laredo: 6  
Laredo E: 7  
Limited: 8  
Limited 4x4: 9  
Limited 75th Anniversary: 10  
Limited 75th Anniversary Edition: 11  
Limited X: 12  
Luxury: 13  
Luxury AWD: 14  
Luxury FWD: 15  
Overland: 16  
Platinum: 17  
Platinum AWD: 18  
Premium Luxury: 19  
Premium Luxury AWD: 20  
Premium Luxury FWD: 21  
SRT: 22  
SRT Night: 23  
Sterling Edition: 24  
Summit: 25  
Trackhawk: 26  
Trailhawk: 27  
Upland: 28

### split training data for validation set

# split features, trim label and price label into 70/30 for train and validation sets  
X\_train\_final, X\_val, y\_trim\_train\_final, y\_trim\_val, y\_price\_train\_final, y\_price\_val = train\_test\_split(  
 X\_train\_pca, y\_trim\_train\_encoded, y\_price\_train, test\_size=0.3, random\_state=42)  
  
print(X\_train\_final.shape)  
print(X\_val.shape)  
  
print(y\_trim\_train\_final.shape)  
print(y\_trim\_val.shape)  
  
print(y\_price\_train\_final.shape)  
print(y\_price\_val.shape)

(4088, 788)  
(1753, 788)  
(4088,)  
(1753,)  
(4088,)  
(1753,)

X\_val[0][:5]

array([ 1.8161653 , -0.17188878, -1.66012565, -0.48802285, 0.29540282])

y\_trim\_val[0]

13

y\_price\_val[:1]

6147 35500.0  
Name: Dealer\_Listing\_Price, dtype: float64

### evaluate classification models on trim labels

# train and logistic regression model  
logistic\_reg = LogisticRegression(max\_iter=1000)  
logistic\_reg.fit(X\_train\_final, y\_trim\_train\_final)  
  
# create trim predictions given validation set  
y\_trim\_val\_pred = logistic\_reg.predict(X\_val)  
  
# check predictions vs actuals  
print(y\_trim\_val\_pred[:5])  
print(y\_trim\_val[:5])

[13 2 8 25 8]  
[13 2 2 25 8]

# check accuracy and f1 of the predictions compared to actuals  
print(accuracy\_score(y\_trim\_val, y\_trim\_val\_pred))  
print(f1\_score(y\_trim\_val, y\_trim\_val\_pred, average='weighted'))

0.8539646320593268  
0.8459025699506811

# now train and test on random forest  
random\_forest = RandomForestClassifier(random\_state=42)  
random\_forest.fit(X\_train\_final, y\_trim\_train\_final)  
  
y\_trim\_val\_pred = random\_forest.predict(X\_val)  
  
print(accuracy\_score(y\_trim\_val, y\_trim\_val\_pred))  
print(f1\_score(y\_trim\_val, y\_trim\_val\_pred, average='weighted'))

0.6948088990302339  
0.6440930059841969

# now train and test on k nearest neighbor  
knn\_clf = KNeighborsClassifier()  
knn\_clf.fit(X\_train\_final, y\_trim\_train\_final)  
  
y\_trim\_val\_pred = knn\_clf.predict(X\_val)  
  
print(accuracy\_score(y\_trim\_val, y\_trim\_val\_pred))  
print(f1\_score(y\_trim\_val, y\_trim\_val\_pred, average='weighted'))

0.6936679977181974  
0.6737595332560347

# now train and test on naive bayes  
naive\_bayes = GaussianNB()  
naive\_bayes.fit(X\_train\_final, y\_trim\_train\_final)  
  
y\_trim\_val\_pred = naive\_bayes.predict(X\_val)  
  
print(accuracy\_score(y\_trim\_val, y\_trim\_val\_pred))  
print(f1\_score(y\_trim\_val, y\_trim\_val\_pred, average='weighted'))

0.4015972618368511  
0.4207431829020433

### hyperparameter tuning to get best model with best parameters

# define the parameter grid for log reg  
# C changes regularization amount while solver chooses different optimization type  
param\_grid\_lr = {  
 'C': [0.01, 0.1, 1, 10],  
 'solver': ['liblinear', 'lbfgs']  
}  
  
# create the log reg model again  
logistic\_reg = LogisticRegression(max\_iter=1000)  
  
# perform grid search with cross-validation  
grid\_search\_lr = GridSearchCV(logistic\_reg, param\_grid\_lr, cv=5, scoring='accuracy')  
grid\_search\_lr.fit(X\_train\_final, y\_trim\_train\_final)  
  
# get the best parameters and the best model  
best\_params\_lr = grid\_search\_lr.best\_params\_  
best\_model\_lr = grid\_search\_lr.best\_estimator\_  
  
print(best\_params\_lr)  
print(best\_model\_lr)

{'C': 10, 'solver': 'liblinear'}  
LogisticRegression(C=10, max\_iter=1000, solver='liblinear')

# create predictions from validation features  
y\_trim\_val\_pred\_lr = best\_model\_lr.predict(X\_val)  
print(y\_trim\_val\_pred\_lr[:5])

[13 2 2 25 8]

print(accuracy\_score(y\_trim\_val, y\_trim\_val\_pred\_lr))  
print(f1\_score(y\_trim\_val, y\_trim\_val\_pred\_lr, average='weighted'))

0.8585282373074729  
0.8544483011681455

# Define the parameter grid for Random Forest  
param\_grid\_rf = {  
 'n\_estimators': [50, 100, 200],  
 'max\_depth': [5, 10, 20]  
}  
  
# Initialize the Random Forest model  
random\_forest = RandomForestClassifier(random\_state=42)  
  
# Perform Grid Search with cross-validation  
grid\_search\_rf = GridSearchCV(random\_forest, param\_grid\_rf, cv=5, scoring='accuracy')  
grid\_search\_rf.fit(X\_train\_final, y\_trim\_train\_final)  
  
# Get the best parameters and the best model  
best\_params\_rf = grid\_search\_rf.best\_params\_  
best\_model\_rf = grid\_search\_rf.best\_estimator\_  
  
print(best\_params\_rf)  
print(best\_model\_rf)

{'max\_depth': 20, 'n\_estimators': 100}  
RandomForestClassifier(max\_depth=20, random\_state=42)

# Predict on the validation set  
y\_trim\_val\_pred\_rf = best\_model\_rf.predict(X\_val)  
  
# Calculate metrics  
print(accuracy\_score(y\_trim\_val, y\_trim\_val\_pred\_rf))  
print(f1\_score(y\_trim\_val, y\_trim\_val\_pred\_rf, average='weighted'))

0.679406731317741  
0.6259801300356478

# best classification model is LogisticRegression(C=10, max\_iter=1000, solver='liblinear')  
# with accuracy and f1 of 86%

### evaluate regression models on price labels

# train Linear Regression  
linear\_reg = LinearRegression()  
linear\_reg.fit(X\_train\_final, y\_price\_train\_final)  
  
# predict prices  
y\_price\_val\_pred = linear\_reg.predict(X\_val)  
  
print(y\_price\_val\_pred[:5])  
print(y\_price\_val[:5])

[32079.35250155 33094.18363729 30619.49401673 37255.49237462  
 29743.81160114]  
6147 35500.0  
184 28637.0  
174 28695.0  
6021 38880.0  
2533 29790.0  
Name: Dealer\_Listing\_Price, dtype: float64

print(mean\_squared\_error(y\_price\_val, y\_price\_val\_pred, squared=False))  
print(r2\_score(y\_price\_val, y\_price\_val\_pred))

3007.7656746136336  
0.8548957604797974

# Train Decision Tree Regressor  
decision\_tree\_reg = DecisionTreeRegressor(random\_state=42)  
decision\_tree\_reg.fit(X\_train\_final, y\_price\_train\_final)  
  
# Predict and evaluate  
y\_price\_val\_pred = decision\_tree\_reg.predict(X\_val)  
  
print(mean\_squared\_error(y\_price\_val, y\_price\_val\_pred, squared=False))  
print(r2\_score(y\_price\_val, y\_price\_val\_pred))

6096.547938474299  
0.40384376542526457

# Train Random Forest Regressor  
random\_forest\_reg = RandomForestRegressor(random\_state=42)  
random\_forest\_reg.fit(X\_train\_final, y\_price\_train\_final)  
  
# Predict and evaluate  
y\_price\_val\_pred = random\_forest\_reg.predict(X\_val)  
  
print(mean\_squared\_error(y\_price\_val, y\_price\_val\_pred, squared=False))  
print(r2\_score(y\_price\_val, y\_price\_val\_pred))

4423.212391470294  
0.6861893408328988

# Train Support Vector Regressor  
svr = SVR()  
svr.fit(X\_train\_final, y\_price\_train\_final)  
  
# Predict and evaluate  
y\_price\_val\_pred = svr.predict(X\_val)  
  
print(mean\_squared\_error(y\_price\_val, y\_price\_val\_pred, squared=False))  
print(r2\_score(y\_price\_val, y\_price\_val\_pred))

7926.585226717974  
-0.007776834190083237

# Train K-Nearest Neighbors Regressor  
knn\_reg = KNeighborsRegressor()  
knn\_reg.fit(X\_train\_final, y\_price\_train\_final)  
  
# Predict and evaluate  
y\_price\_val\_pred = knn\_reg.predict(X\_val)  
  
print(mean\_squared\_error(y\_price\_val, y\_price\_val\_pred, squared=False))  
print(r2\_score(y\_price\_val, y\_price\_val\_pred))

4478.120847999408  
0.6783498762953561

### hyperparameter tuning on the best linear regression model

# parameter grid for best ridge regression  
# alpha controls regularization, solver is the optimizer  
param\_grid\_ridge = {  
 'alpha': [0.01, 0.1, 1, 10, 50],  
 'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sag', 'saga']  
}  
  
# initialize model  
ridge\_reg = Ridge()  
  
# perform grid search, scoring rmse  
grid\_search\_ridge = GridSearchCV(ridge\_reg, param\_grid\_ridge, cv=5, scoring='neg\_mean\_squared\_error')  
grid\_search\_ridge.fit(X\_train\_final, y\_price\_train\_final)  
  
# get best parameter and model  
best\_params\_ridge = grid\_search\_ridge.best\_params\_  
best\_model\_ridge = grid\_search\_ridge.best\_estimator\_  
  
# predict on validation set  
y\_price\_val\_pred\_ridge = best\_model\_ridge.predict(X\_val)  
  
y\_price\_val\_pred\_ridge[:5]

array([32433.81369174, 32258.65822153, 30668.6735075 , 37517.93789474,  
 29725.2143707 ])

best\_model\_ridge

Ridge(alpha=1, solver='saga')

print(mean\_squared\_error(y\_price\_val, y\_price\_val\_pred\_ridge, squared=False))  
print(r2\_score(y\_price\_val, y\_price\_val\_pred\_ridge))

2917.171100317708  
0.8635052621153299

# parameter grid for lasso regression  
# alpha controls regularization, max iter is iterations  
param\_grid\_lasso = {  
 'alpha': [0.01, 0.1, 1, 10, 50],  
 'max\_iter': [1000, 2000, 3000]  
}  
  
# initialize model  
lasso\_reg = Lasso()  
  
# perform grid search  
grid\_search\_lasso = GridSearchCV(lasso\_reg, param\_grid\_lasso, cv=5, scoring='neg\_mean\_squared\_error')  
grid\_search\_lasso.fit(X\_train\_final, y\_price\_train\_final)  
  
# get best param and model  
best\_params\_lasso = grid\_search\_lasso.best\_params\_  
best\_model\_lasso = grid\_search\_lasso.best\_estimator\_  
  
# predict on validation set  
y\_price\_val\_pred\_lasso = best\_model\_lasso.predict(X\_val)  
  
print(y\_price\_val\_pred\_lasso[:5])  
print(y\_price\_val[:5])

[32944.60322573 32262.39268183 30667.2036255 37623.06597038  
 29721.26259184]  
6147 35500.0  
184 28637.0  
174 28695.0  
6021 38880.0  
2533 29790.0  
Name: Dealer\_Listing\_Price, dtype: float64

# get rmse and r2  
print(mean\_squared\_error(y\_price\_val, y\_price\_val\_pred\_lasso, squared=False))  
print(r2\_score(y\_price\_val, y\_price\_val\_pred\_lasso))

2955.8838259114113  
0.8598584791657555

# best regression model for price  
best\_model\_ridge

Ridge(alpha=1, solver='saga')

# best classification model for trim  
best\_model\_lr

LogisticRegression(C=10, max\_iter=1000, solver='liblinear')

* Best model to predict trim: LogisticRegression(C=10, max\_iter=1000, solver='liblinear') with accuracy and f1 of 86%
* Best model to predict price: Ridge(alpha=1, solver='saga') with RMSE of 2917 and R2 of 86%

### use best models to predict price and trim on test set

test\_data = pd.read\_csv('Test\_DataSet.csv')  
test\_data.head()

ListingID SellerCity SellerIsPriv SellerListSrc \  
0 8622015 Seneca False HomeNet Automotive   
1 8625693 Bedford False Inventory Command Center   
2 8625750 Webster False Jeep Certified Program   
3 8626885 Louisville False Digital Motorworks (DMi)   
4 8627430 Palmyra False Digital Motorworks (DMi)   
  
 SellerName SellerRating SellerRevCnt \  
0 Lake Keowee Chrysler Dodge Jeep Ram 2.5 59   
1 North Coast Auto Mall 4.7 2116   
2 Marina Chrysler Dodge Jeep Mitsubishi RAM 3.9 46   
3 Oxmoor Ford Lincoln 4.5 1075   
4 F.C. Kerbeck &amp; Sons 4.6 162   
  
 SellerState SellerZip VehBodystyle VehCertified \  
0 SC 29678 SUV False   
1 OH 44146 SUV False   
2 NY 14580 SUV True   
3 KY 40222 SUV False   
4 NJ 8065 SUV False   
  
 VehColorExt VehColorInt VehDriveTrain \  
0 Stellar Black Metallic Cirrus FWD   
1 True Blue Pearlcoat Black Four Wheel Drive   
2 Brilliant Black Crystal Pearlcoat Black 4WD   
3 Brilliant Black Crystal Pearlcoat Black 4WD   
4 Harbor Blue Metallic Jet Black AWD   
  
 VehEngine \  
0 Gas V6 3.6L/222.6   
1 3.6L V6 CYLINDER   
2 3.6L V6 24V MPFI DOHC Flexible Fuel   
3 3.6L V6 24V MPFI DOHC   
4 3.6L V6 24V GDI DOHC   
  
 VehFeats VehFuel \  
0 ['4-Wheel Disc Brakes', 'ABS', 'Active Suspens... Gasoline   
1 ['12v Power Outlet', 'ABS Brakes', 'Air Condit... Gasoline   
2 ['1st and 2nd row curtain head airbags', '4-wh... E85 Flex Fuel   
3 ['1st and 2nd row curtain head airbags', '4-wh... Gasoline   
4 ['1st and 2nd row curtain head airbags', '4-wh... Gasoline   
  
 VehHistory VehListdays VehMake \  
0 1 Owner, Non-Personal Use Reported, Buyback Pr... 143.991262 Cadillac   
1 1 Owner, Accident(s) Reported, Non-Personal Us... 138.770486 Jeep   
2 1 Owner, Buyback Protection Eligible 31.951088 Jeep   
3 1 Owner, Buyback Protection Eligible 5.950127 Jeep   
4 1 Owner, Non-Personal Use Reported, Buyback Pr... 24.672986 Cadillac   
  
 VehMileage VehModel VehPriceLabel \  
0 13625.0 XT5 Good Deal   
1 42553.0 Grand Cherokee Good Deal   
2 48951.0 Grand Cherokee Good Deal   
3 44179.0 Grand Cherokee Good Deal   
4 22269.0 XT5 Good Deal   
  
 VehSellerNotes VehType \  
0 Thank you for visiting another one of Lake Keo... Used   
1 This 2017 Jeep Grand Cherokee 4dr Limited 4x4 ... Used   
2 Certified. Brilliant Black Crystal Pearlcoat 2... Used   
3 2015 Jeep Grand Cherokee \*\*\*THIS VEHICLE IS AT... Used   
4 AWD, CarFax One Owner! Navigation, Back-up Cam... Used   
  
 VehTransmission VehYear   
0 8-Speed Automatic 2018   
1 8-Speed Automatic 2017   
2 8-Speed Automatic 2015   
3 8-Speed Automatic 2015   
4 8-Speed Automatic 2018

# field columns with only one unique value  
columns\_to\_remove = [col for col in test\_data.columns if test\_data[col].nunique() == 1]  
columns\_to\_remove

['VehBodystyle', 'VehType']

# remove these 2 field from the dataset, they provide no additional info  
test\_data = test\_data.drop(columns=columns\_to\_remove)  
test\_data.shape

(1000, 25)

# create features by dropping id field  
X\_test = test\_data.drop(columns=['ListingID'])  
print(X\_test.shape)

(1000, 24)

# check feature field types  
X\_test.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1000 entries, 0 to 999  
Data columns (total 24 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 SellerCity 1000 non-null object   
 1 SellerIsPriv 1000 non-null bool   
 2 SellerListSrc 1000 non-null object   
 3 SellerName 1000 non-null object   
 4 SellerRating 1000 non-null float64  
 5 SellerRevCnt 1000 non-null int64   
 6 SellerState 1000 non-null object   
 7 SellerZip 1000 non-null int64   
 8 VehCertified 1000 non-null bool   
 9 VehColorExt 993 non-null object   
 10 VehColorInt 892 non-null object   
 11 VehDriveTrain 936 non-null object   
 12 VehEngine 942 non-null object   
 13 VehFeats 963 non-null object   
 14 VehFuel 1000 non-null object   
 15 VehHistory 973 non-null object   
 16 VehListdays 1000 non-null float64  
 17 VehMake 1000 non-null object   
 18 VehMileage 999 non-null float64  
 19 VehModel 1000 non-null object   
 20 VehPriceLabel 962 non-null object   
 21 VehSellerNotes 959 non-null object   
 22 VehTransmission 973 non-null object   
 23 VehYear 1000 non-null int64   
dtypes: bool(2), float64(3), int64(3), object(16)  
memory usage: 174.0+ KB

# select all the groups of fields by type: numerical, categorical, boolean, text and list features  
num\_cols = X\_test.select\_dtypes(include=['int64', 'float64']).columns  
cat\_cols = X\_test.select\_dtypes(include=['object']).columns.difference(['VehSellerNotes','VehFeats'])  
bool\_cols = X\_test.select\_dtypes(include=['bool']).columns  
text\_col = 'VehSellerNotes'  
list\_col = 'VehFeats'  
  
print(num\_cols)  
print(cat\_cols)  
print(bool\_cols)  
print(text\_col)  
print(list\_col)  
  
# still 24 fields  
print(len(num\_cols) + len(cat\_cols) + len(bool\_cols) + 1 + 1)

Index(['SellerRating', 'SellerRevCnt', 'SellerZip', 'VehListdays',  
 'VehMileage', 'VehYear'],  
 dtype='object')  
Index(['SellerCity', 'SellerListSrc', 'SellerName', 'SellerState',  
 'VehColorExt', 'VehColorInt', 'VehDriveTrain', 'VehEngine', 'VehFuel',  
 'VehHistory', 'VehMake', 'VehModel', 'VehPriceLabel',  
 'VehTransmission'],  
 dtype='object')  
Index(['SellerIsPriv', 'VehCertified'], dtype='object')  
VehSellerNotes  
VehFeats  
24

### pre-process numerical features

# for numerical features, impute missing values and scale them  
# fit on numerical fields for training data  
X\_test\_num = num\_imputer.transform(X\_test[num\_cols])  
X\_test\_num = scaler.transform(X\_test\_num)  
  
print(X\_test\_num[:5])  
print(X\_test\_num.shape)

[[-1.44427654 -0.30052937 -0.77184994 1.30265674 -0.99649737 1.01844168]  
 [ 0.47490389 1.26302713 -0.04931951 1.22549546 1.22311597 0.19520367]  
 [-0.22297991 -0.31041086 -1.52584252 -0.35325873 1.71402742 -1.45127235]  
 [ 0.30043294 0.47174744 -0.24528367 -0.73754404 1.34787715 -1.45127235]  
 [ 0.38766841 -0.22223752 -1.85120095 -0.46082658 -0.33325284 1.01844168]]  
(1000, 6)

### pre-process categorical features

# impute missing values with 'missing' and one hot encode categorical features  
# fit on categorical training data  
X\_test\_cat = cat\_imputer.transform(X\_test[cat\_cols])  
X\_test\_cat = encoder.transform(X\_test\_cat)  
  
print(X\_test\_cat[:5])  
print(X\_test\_cat.shape)

[[0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]]  
(1000, 4004)

### pre-process boolean features

# preprocess boolean features, convert true or false 1 or 0  
X\_test\_bool = X\_test[bool\_cols].astype(int).values  
  
print(X\_test\_bool[:5])  
print(X\_test\_bool.shape)

[[0 0]  
 [0 0]  
 [0 1]  
 [0 0]  
 [0 0]]  
(1000, 2)

### pre-process seller notes text

# check values  
X\_test[text\_col].head()

0 Thank you for visiting another one of Lake Keo...  
1 This 2017 Jeep Grand Cherokee 4dr Limited 4x4 ...  
2 Certified. Brilliant Black Crystal Pearlcoat 2...  
3 2015 Jeep Grand Cherokee \*\*\*THIS VEHICLE IS AT...  
4 AWD, CarFax One Owner! Navigation, Back-up Cam...  
Name: VehSellerNotes, dtype: object

# create tfidf to convert text values into vectors  
# remove stopwords that dont provide info and use 100 of the top ranked terms across the corpus  
# fit the vectorizer to the text column  
X\_test\_text = vectorizer.transform(X\_test[text\_col].fillna('')).toarray()  
  
print(X\_test\_text[2])  
print(X\_test\_text.shape)

[0.2438432 0.2444744 0.09450755 0. 0.08205055 0.06905422  
 0.06154613 0.15340564 0. 0.08194329 0. 0.05672536  
 0. 0.07859434 0. 0. 0. 0.  
 0.17426451 0. 0. 0.25816392 0.0621827 0.09179497  
 0. 0. 0. 0.17352975 0.06491281 0.08563113  
 0. 0.07734391 0. 0. 0. 0.  
 0.07625276 0.07695312 0. 0.08781169 0.0649596 0.06216842  
 0. 0. 0.1483974 0.08275679 0.16667804 0.07795112  
 0.08146509 0.05885015 0.07757607 0. 0. 0.13559938  
 0. 0.07432454 0. 0.23672637 0. 0.  
 0. 0.06709033 0.06474184 0.05772639 0. 0.  
 0.08096808 0.10976387 0.0687887 0. 0.06629873 0.08232031  
 0.07517658 0. 0.28666402 0. 0.06861293 0.  
 0. 0. 0. 0.06749372 0. 0.06299329  
 0. 0.12970106 0. 0. 0.08497588 0.  
 0.11662966 0.35989446 0. 0.07111084 0.38761589 0.05971019  
 0. 0. 0. 0. ]  
(1000, 100)

### pre-process vehicle features list

# check values  
X\_test[list\_col].head()

0 ['4-Wheel Disc Brakes', 'ABS', 'Active Suspens...  
1 ['12v Power Outlet', 'ABS Brakes', 'Air Condit...  
2 ['1st and 2nd row curtain head airbags', '4-wh...  
3 ['1st and 2nd row curtain head airbags', '4-wh...  
4 ['1st and 2nd row curtain head airbags', '4-wh...  
Name: VehFeats, dtype: object

# fill nulls, convert list to text  
X\_test[list\_col] = X\_test[list\_col].fillna('[]').apply(eval).apply(lambda x: ','.join(x))  
X\_test[list\_col].head()

0 4-Wheel Disc Brakes,ABS,Active Suspension,Adju...  
1 12v Power Outlet,ABS Brakes,Air Conditioning,A...  
2 1st and 2nd row curtain head airbags,4-wheel A...  
3 1st and 2nd row curtain head airbags,4-wheel A...  
4 1st and 2nd row curtain head airbags,4-wheel A...  
Name: VehFeats, dtype: object

# use one hot encoder to convert comma seperated strings into a matrix with 1 or 0 for if the feature is present  
X\_test\_feats = feats\_encoder.transform(X\_test[list\_col].values.reshape(-1, 1))  
  
print(X\_test\_feats[:5])  
print(X\_test\_feats.shape)

[[0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]  
 [0. 0. 0. ... 0. 0. 0.]]  
(1000, 797)

### combine features then use PCA to reduce dimentionality

# combine all preprocessed features together using hstack to horizontally concat the arrays  
X\_test\_preprocessed = np.hstack((X\_test\_num, X\_test\_cat, X\_test\_bool, X\_test\_text, X\_test\_feats))  
  
print(X\_test\_preprocessed[:1])  
print(X\_test\_preprocessed.shape)

[[-1.44427654 -0.30052937 -0.77184994 ... 0. 0.  
 0. ]]  
(1000, 4909)

# use principal component analysis from earlier to retain 95% of variance  
X\_test\_pca = pca.transform(X\_test\_preprocessed)  
  
# only 788 fields now  
print(X\_test\_preprocessed.shape[1])  
print(X\_test\_pca.shape[1])

4909  
788

# check how the records look now  
print(X\_test\_preprocessed[0][:5])  
print(X\_test\_pca[0][:5])

[-1.44427654 -0.30052937 -0.77184994 1.30265674 -0.99649737]  
[ 2.02366936 -1.58549429 0.60574677 0.11481991 1.33805313]

# predict prices on test set  
y\_price\_pred\_ridge = best\_model\_ridge.predict(X\_test\_pca)  
y\_price\_pred\_ridge[:5]

array([44726.30988604, 23094.37984588, 21849.13222156, 24999.6774848 ,  
 41616.42091966])

# predict trim on test set  
y\_trim\_pred\_lr = best\_model\_lr.predict(X\_test\_pca)  
y\_trim\_pred\_lr[:5]

array([19, 8, 6, 8, 19])

# conver trim from numerical back into categorical text  
y\_trim\_pred\_text = label\_encoder.inverse\_transform(y\_trim\_pred\_lr)  
y\_trim\_pred\_text[:5]

array(['Premium Luxury', 'Limited', 'Laredo', 'Limited', 'Premium Luxury'],  
 dtype=object)

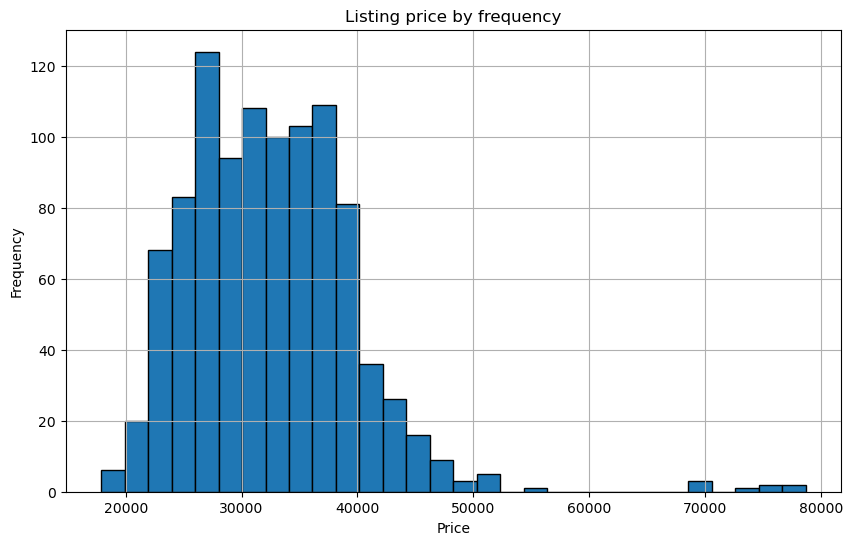
test\_data = pd.read\_csv('Test\_DataSet.csv')  
  
# add the predictions back to the test\_data  
test\_data['Vehicle\_Trim'] = y\_trim\_pred\_text  
test\_data['Dealer\_Listing\_Price'] = y\_price\_pred\_ridge  
  
test\_data.head()

ListingID SellerCity SellerIsPriv SellerListSrc \  
0 8622015 Seneca False HomeNet Automotive   
1 8625693 Bedford False Inventory Command Center   
2 8625750 Webster False Jeep Certified Program   
3 8626885 Louisville False Digital Motorworks (DMi)   
4 8627430 Palmyra False Digital Motorworks (DMi)   
  
 SellerName SellerRating SellerRevCnt \  
0 Lake Keowee Chrysler Dodge Jeep Ram 2.5 59   
1 North Coast Auto Mall 4.7 2116   
2 Marina Chrysler Dodge Jeep Mitsubishi RAM 3.9 46   
3 Oxmoor Ford Lincoln 4.5 1075   
4 F.C. Kerbeck &amp; Sons 4.6 162   
  
 SellerState SellerZip VehBodystyle VehCertified \  
0 SC 29678 SUV False   
1 OH 44146 SUV False   
2 NY 14580 SUV True   
3 KY 40222 SUV False   
4 NJ 8065 SUV False   
  
 VehColorExt VehColorInt VehDriveTrain \  
0 Stellar Black Metallic Cirrus FWD   
1 True Blue Pearlcoat Black Four Wheel Drive   
2 Brilliant Black Crystal Pearlcoat Black 4WD   
3 Brilliant Black Crystal Pearlcoat Black 4WD   
4 Harbor Blue Metallic Jet Black AWD   
  
 VehEngine \  
0 Gas V6 3.6L/222.6   
1 3.6L V6 CYLINDER   
2 3.6L V6 24V MPFI DOHC Flexible Fuel   
3 3.6L V6 24V MPFI DOHC   
4 3.6L V6 24V GDI DOHC   
  
 VehFeats VehFuel \  
0 ['4-Wheel Disc Brakes', 'ABS', 'Active Suspens... Gasoline   
1 ['12v Power Outlet', 'ABS Brakes', 'Air Condit... Gasoline   
2 ['1st and 2nd row curtain head airbags', '4-wh... E85 Flex Fuel   
3 ['1st and 2nd row curtain head airbags', '4-wh... Gasoline   
4 ['1st and 2nd row curtain head airbags', '4-wh... Gasoline   
  
 VehHistory VehListdays VehMake \  
0 1 Owner, Non-Personal Use Reported, Buyback Pr... 143.991262 Cadillac   
1 1 Owner, Accident(s) Reported, Non-Personal Us... 138.770486 Jeep   
2 1 Owner, Buyback Protection Eligible 31.951088 Jeep   
3 1 Owner, Buyback Protection Eligible 5.950127 Jeep   
4 1 Owner, Non-Personal Use Reported, Buyback Pr... 24.672986 Cadillac   
  
 VehMileage VehModel VehPriceLabel \  
0 13625.0 XT5 Good Deal   
1 42553.0 Grand Cherokee Good Deal   
2 48951.0 Grand Cherokee Good Deal   
3 44179.0 Grand Cherokee Good Deal   
4 22269.0 XT5 Good Deal   
  
 VehSellerNotes VehType \  
0 Thank you for visiting another one of Lake Keo... Used   
1 This 2017 Jeep Grand Cherokee 4dr Limited 4x4 ... Used   
2 Certified. Brilliant Black Crystal Pearlcoat 2... Used   
3 2015 Jeep Grand Cherokee \*\*\*THIS VEHICLE IS AT... Used   
4 AWD, CarFax One Owner! Navigation, Back-up Cam... Used   
  
 VehTransmission VehYear Vehicle\_Trim Dealer\_Listing\_Price   
0 8-Speed Automatic 2018 Premium Luxury 44726.309886   
1 8-Speed Automatic 2017 Limited 23094.379846   
2 8-Speed Automatic 2015 Laredo 21849.132222   
3 8-Speed Automatic 2015 Limited 24999.677485   
4 8-Speed Automatic 2018 Premium Luxury 41616.420920

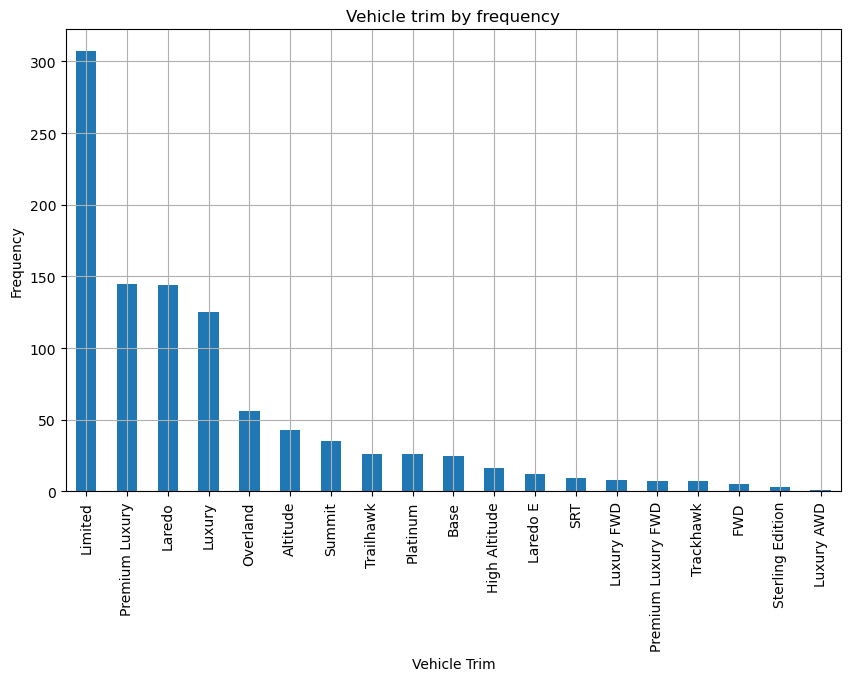
# select 3 required fields and write out  
output\_data = test\_data[['ListingID', 'Vehicle\_Trim', 'Dealer\_Listing\_Price']]  
output\_data.to\_csv('test\_data\_with\_predictions.csv', index=False)

### prediction visualizations

# gistogram of Dealer\_Listing\_Price  
plt.figure(figsize=(10, 6))  
plt.hist(output\_data['Dealer\_Listing\_Price'], bins=30, edgecolor='black')  
plt.title('Listing price by frequency')  
plt.xlabel('Price')  
plt.ylabel('Frequency')  
plt.grid(True)  
plt.show()



# bar Chart of Vehicle\_Trim  
plt.figure(figsize=(10, 6))  
output\_data['Vehicle\_Trim'].value\_counts().plot(kind='bar')  
plt.title('Vehicle trim by frequency')  
plt.xlabel('Vehicle Trim')  
plt.ylabel('Frequency')  
plt.grid(True)  
plt.show()



### Conclusion:

Based on our 2 models, we were able to correctly predict trims on validation data by an accuracy and F1 of 85%. The accuracy means that of 100 predictions for vehicle trims, 85 were correct.The F1 means the balance between correctly predicted trims and identifying all relevant trims is good.

On the price predictions, we were able to predict prices with an RMSE of 2917 and an R² score of 86%. The RMSE tells us that on average, the price predictions are off by $2917 and the R^2 tells us the model explains 86% of the variation in vehicle prices.