Karun Mokha
Varun Dinesh
Bret Dubois

Cogs 109 Final Project: Used Car Price Prediction

Intro:

The used car market has been revolutionized by the internet over the last few decades, and car owners, buyers, and sellers are paying attention. According to a survey by Capital One, "More than half (55%) of car buyers and 68% of dealers said that inaccurate financing estimates provided by third-party digital tools don't match final rates and payments" (Capital One Car Buying Outlook, 2023). Before the internet, one would go to their local dealership, and have access to very little information about how a used car stacks up in performance compared to hundreds or thousands of used cars of that same model across the country. It was much harder to research the vehicle history and specification for cars, let alone be able to read thousands of tutorials and threads online about car sale recommendations, used vehicles, fixing cars, and so forth. It was a whole different process compared to now, where we can access massive online datasets of used car information and run hypothetical models to derive meaning.

Nowadays car sellers and buyers can complete this entire process free from a car salesman or dealership. Sellers can leverage online data to find the maximum price their car can be resold for, whereas buyers can leverage the data to find the appropriate price they should be paying as well. Suppose at your local dealership they offered you $18k for your car, but from running regression on the public access dataset of your vehicle makes history, you find that cars with similar characteristics are actually selling for $22k, or vice versa when selling your car and being faced with a lowball offer. For our project we ran regression models to predict a used car's selling price from the public access (CC0 kaggle) Used Car Auction Prices dataset. We used 100,000 observations and included year, transmission, state, make_model, condition, odometer as our predictors, and we transformed the categorical predictors into all numerical predictors; we will discuss in more detail in the methods section.

Some key characteristics about our dataset is that our dataset is open access public domain on kaggle, and contains historical car sales prices, scraped from online in 2015 by Bojan Tunguz. Our dataset has 558,832 rows and 16 columns. Our 16 columns are given by the following variables: {Date, Make, Model, Trim, Body, Transmission, VIN, State, Condition, Odometer, Color, Interior, Seller, mmr, Selling Price, Saledate}. Though not all of these columns are relevant towards our research question. We downloaded the file from kaggle and uploaded it to JupyterHub in Python. Condition and odometer are of float64 type, and year, mmr, and selling price are of int64 type, and the rest are of object types. When we explore the null values of our dataset we see that year, state, seller, mmr, selling price, and selling date have no null values, and make, model, trim, body, condition, color, and interior have around 1000 null values, while transmission has the most at 6500, and odometer with only about 100 null values.

In our code you will notice we remove the vin number, and combine make and model into one column. We then decide that we could use our data to effectively replace our null values based on our data. We then replace null transmission values with the transmission mode of the cars make model if the make model is not null across the dataset, and the rest of the transmission nulls with the mode of all the transmission values. We also replaced the condition nulls based on cars of that same year's average. We replaced the color's nulls with the mode of colors across the dataset, and odometer nulls with the average odometer of the make model. We finally removed rows where trim, body, interior, and make/model were null. You will finally see we turn categorical columns into numerical depending on the type of categorical variable.

Methods:

Due to the large size of the dataset and computational power needed when trying to use it entirely, we started by downsampling the complete dataset down into a smaller subset with a sample size of 100,000 entries. To evaluate the effectiveness of our models, we used K-Fold cross-validation with k=5 to split the new dataset into 5 subsets. Then the model is trained on 4 of the subsets while testing on the 5th subset. The process is repeated five times where each subset is used exactly one time as the testing set. Finally, the performance of the model is averaged over the five runs so we could get a better estimate of how it performs. Ultimately this was helpful for reducing the risks of overfitting and to help provide a more generalized measurement of how effective the model is at making predictions.

As for our data analysis approach, we chose to focus on prediction using a set of the key features (year, transmission, state, make_model, condition, odometer) that we found to be most influential when determining a used car's ideal selling price. The categorical features include transmission, year, state, and make_model. The numerical features include the condition and odometer. Since we are using models that require features with numerical values, we needed to use a data preprocessing pipeline to transform the categorical features into numerical values. We decided to one-hot encode the transmission, state, and make_model features and perform an ordinal encoder on the ordinal categorical feature, that being year. For the numerical features, we created a pipeline to normalize or standardize the values in the features.

After engineering the data to fit the specifications of the model, we were able to use a total of three regression models to predict the selling price of used cars, including linear regression, polynomial regression, and lasso regression. We chose to use the linear regression model as a baseline due to its simplicity and interpretability. It assumes that there is a linear relationship between the predictors and the selling price, whereas the polynomial model helped to capture more complex relationships between the predictors and selling price by incorporating higher-order terms. We also created the lasso regression model to include a penalty term that would help with feature selection and to prevent the model from overfitting by not considering the less relevant features.

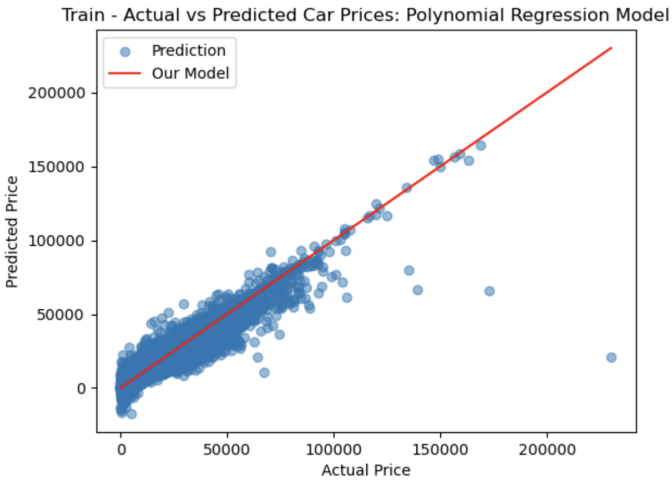
Results - Model Selection:

To find the best model for our prediction, we compared the three models using their R-squared scores and Mean Squared Error (MSE) from our cross-validation. MSE representing our average squared errors. The R-squared score indicate the variance proportion found in the dependent variable that is predictable from the independent variable with a score ranging from 0 to 1, where a value of 1 indicates that the model can perfectly predict the selling price and a score of 0 means that the model does no better than just finding the mean of the selling price.

In one execution of a random sample, the linear regression model gave a mean CV test R-squared score of about 0.813 and a mean MSE of 17522683.991 across each of the 5-folds of CV. This implies that the model can explain about 81.3% of the variation of the selling price. On average, the predicted selling price is different from the actual selling price by about the square root of the MSE. In comparison, the second degree polynomial regression model had stronger performance than the linear regression model since it had a mean CV test R-squared score of about 0.901 and a mean MSE of 9269615.688, meaning the model can explain about 90.1% of selling price variability; additionally, since in comparison the MSE is lower than the linear regression model, then the polynomial regression model has a higher accuracy of making predictions for the actual selling price. The lasso regression model resulted in similar performance to the linear regression model with a mean CV test R-squared score of about 0.812 and a mean CV MSE of 17621479.788 which tells us that the lasso penalty doesn't offer significant improvements to the model's predictive capabilities using this data. From the results of each model we determined that the polynomial regression model is most ideal due to its superior performance in comparison to the linear and lasso regression models.

Results - Model Estimation:

Based on the performance in terms of both R-squared and MSE, the final model chosen was the polynomial regression model with degree 2. Due to the high-dimensional nature of the model, along with there being interaction terms, it is not easy to interpret the specific parameter estimates. Instead, we calculated the R-squared score using the fitted model on the smaller subset of our complete dataset.



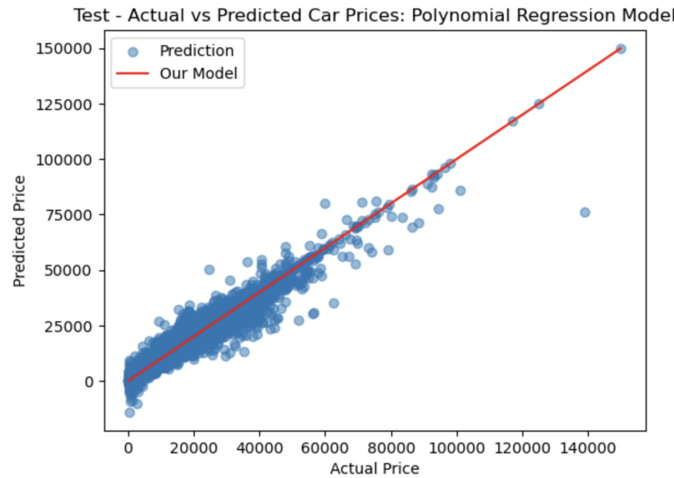


Table.1

Model	Average R-squared Score on Training Sets	Average R-squared Score on Testing Sets	Average MSE on Testing Sets
Linear Regression	0.8271	0.8137	17522683.9917
Polynomial Regression (degree = 2)	0.9311	0.90173	9269615.68814
Lasso Regression	0.82716	0.81238	17621479.7889

Conclusion/Discussion:

Based on the features chosen (year, transmission, state, make_model, condition, odometer), we can conclude that our models performed decently well in predicting the used car selling price. Overall the polynomial regression model performed the best, with the average of the cross validation test R^squared scores resulting in 0.9017 and the average of the cross validation on the MSE being 9269615.68814 which is the lowest for both out of all the models. We also showed the averages of the training R^squared scores through cross validation for each of the models to see whether our models were overfitting to the training sets. We observed that the average training accuracy R^squared score was only slightly higher than that of the average testing accuracy R^squared score for the polynomial regression model, which indicates that there isn’t much overfitting happening.

There were a few other implementations we tried in order to improve the models performance that did not work out because of the time it took to run. For example, we tried using GridSearchCV for our polynomial regression model to provide us with the best performing degree out of a list of predefined degree values, but it took too long to run for degree values greater than 2. This is because our feature matrix already included too many columns after performing one-hot encoding and ordinal encoding. Since the polynomial regression model from Sklearn creates new features created by interactions between all the different combinations of features we have in the feature matrix, this would increase the dimensions of our feature matrix by a lot thus taking too long. A higher degree Polynomial model could have potentially performed better than the ones we had, but also runs the risk of overfitting to the training data. We also tried implementing GridSearchCV for Lasso Regression in order to see which Regularization value out of a predefined list would result in the best average R^squared scores on the test sets after performing cross validation. This took too long to run as well, so we scrapped it.

Going forward, some potential implications for researchers also interested in this topic is to consider the following, though having a large dataset (550k+ columns) was fascinating to work with, it made processing time slow and a hassle to deal with, additionally more research should be done from the sellers point of view building models to fit the predicted selling price based on their car's specs. In larger datasets having the processing power to apply more complex machine learning models would be fascinating as well, considering how hundreds of millions of cars there are in the world, there is inevitably going to be more car data than we can count, but being able to leverage accurate datasets is continuing to change the game of used car selling and buying.

References:

Tunguz, Bojan. "Used Car Auction Prices." *Kaggle*, 18 May 2021, www.kaggle.com/datasets/tunguz/used-car-auction-prices?select=car_prices.csv.

"2023 Car Buying Outlook Examines State of Car Buying I Capital One." *Capital One*, 27 Jan. 2023, www.capitalone.com/about/newsroom/car-buying-outlook-report-2023/?PFFSRCID=S-F1-12345678901-SOC-1007&external_id=COAF_V1_SOC_NAT_USAT_P_KET_216503_Z_CHP_XD_Z_STRY_Z_20230131.

FinalUsedCarPrediction (1)

June 15, 2023

[1]:
```
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
```

[1]:
```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
#import plotly
from scipy import stats
import warnings
import statsmodels.api as sm
warnings.filterwarnings("ignore")

data = pd.read_csv('car_prices.csv', on_bad_lines='skip')

#Get rid of the irrelevent columns
data = data.drop(['vin'], axis=1)



data.head(10)
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:7:
FutureWarning: pandas.Int64Index is deprecated and will be removed from pandas
in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:7:
```

```
FutureWarning: pandas.Float64Index is deprecated and will be removed from pandas
in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
```

[1]:

```
   year       make                model                       trim  \
0  2015        Kia              Sorento                         LX
1  2015        Kia              Sorento                         LX
2  2014        BMW             3 Series                 328i SULEV
3  2015      Volvo                  S60                         T5
4  2014        BMW  6 Series Gran Coupe                       650i
5  2015     Nissan               Altima                      2.5 S
6  2014        BMW                   M5                       Base
7  2014  Chevrolet                Cruze                        1LT
8  2014       Audi                   A4  2.0T Premium Plus quattro
9  2014  Chevrolet               Camaro                         LT

          body transmission state  condition  odometer  color interior  \
0          SUV    automatic    ca        5.0   16639.0  white    black
1          SUV    automatic    ca        5.0    9393.0  white    beige
2        Sedan    automatic    ca        4.5    1331.0   gray    black
3        Sedan    automatic    ca        4.1   14282.0  white    black
4        Sedan    automatic    ca        4.3    2641.0   gray    black
5        Sedan    automatic    ca        1.0    5554.0   gray    black
6        Sedan    automatic    ca        3.4   14943.0  black    black
7        Sedan    automatic    ca        2.0   28617.0  black    black
8        Sedan    automatic    ca        4.2    9557.0  white    black
9  Convertible    automatic    ca        3.0    4809.0    red    black

                                               seller    mmr  sellingprice  \
0                          kia motors america, inc  20500         21500
1                          kia motors america, inc  20800         21500
2              financial services remarketing (lease)  31900         30000
3                          volvo na rep/world omni  27500         27750
4              financial services remarketing (lease)  66000         67000
5  enterprise vehicle exchange / tra / rental / t…  15350         10900
6                            the hertz corporation  69000         65000
7  enterprise vehicle exchange / tra / rental / t…  11900          9800
8                               audi mission viejo  32100         32250
9                               d/m auto sales inc  26300         17500

                                  saledate
0  Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
1  Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2  Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
3  Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
4  Thu Dec 18 2014 12:30:00 GMT-0800 (PST)
5  Tue Dec 30 2014 12:00:00 GMT-0800 (PST)
```

```
6   Wed Dec 17 2014 12:30:00 GMT-0800 (PST)
7   Tue Dec 16 2014 13:00:00 GMT-0800 (PST)
8   Thu Dec 18 2014 12:00:00 GMT-0800 (PST)
9   Tue Jan 20 2015 04:00:00 GMT-0800 (PST)
```

Data cleaning

[3]:
```
#print the data object types
print(data.dtypes)
```

```
year              int64
make             object
model            object
trim             object
body             object
transmission     object
state            object
condition       float64
odometer        float64
color            object
interior         object
seller           object
mmr               int64
sellingprice      int64
saledate         object
dtype: object
```

[4]:
```
print(data.isnull().sum())
```

```
year                0
make            10301
model           10399
trim            10651
body            13195
transmission    65353
state               0
condition       11794
odometer           94
color             749
interior          749
seller              0
mmr                 0
sellingprice        0
saledate            0
dtype: int64
```

[5]:
```python
#combine make and model columns into one
data['make_model'] = data['make'] + ' ' + data['model']
data = data.drop(columns=['make','model'], axis = 1)
```

[6]:
```python
#transmission nulls: could be replaced with transmission mode of cars make␣
↪model if make model is not null
```

[7]:
```python
mapping_model_transm = data.groupby('make_model')['transmission'].apply(lambda␣
↪x: x.value_counts().idxmax() if not x.isnull().all() else None).to_dict()
```

[8]:
```python
data['transmission'] = data.apply(lambda row:␣
↪mapping_model_transm[row['make_model']] if pd.isna(row['transmission']) and␣
↪not pd.isna(row['make_model']) else row['transmission'],axis=1)
```

[9]:
```python
# replace rest of transmission nulls with the mode of all the transmission␣
↪values
```

[10]:
```python
data['transmission'] = data['transmission'].fillna(data['transmission'].mode().
↪loc[0])
```

[11]:
```python
# condition nulls: replaced based on years average
```

[12]:
```python
mapping = data.groupby('year')['condition'].mean().round(1).to_dict()
data['condition'] = data.apply(lambda row: mapping[row['year']]
                                  if pd.isna(row['condition'])
                                  else row['condition'],axis=1)
```

[13]:
```python
# color nulls: filling in with mode of colors
data['color'] = data['color'].fillna(data['color'].mode().loc[0])
```

[14]:
```python
# removing rows where trim,body,interior,make_model is null
```

[15]:
```python
data = data.dropna(subset = ['trim','body','interior','make_model'])
```

[16]:
```python
# odometer nulls replacing with average of make model
```

[17]:
```python
mapping_odometer = data.groupby('make_model')['odometer'].mean().round(1).
↪to_dict()
data['odometer'] = data.apply(lambda row: mapping_odometer[row['make_model']]
                                  if pd.isna(row['odometer'])
                                  else row['odometer'],axis=1)
```

[18]:
```python
#remove irrelevant columns
```

[19]:
```python
data = data.drop(columns = ['seller','saledate'])
```

[20]:
```python
#just in case the data gets messed up
```

[21]:
```
data_sellingpr = data
```

[22]:
```
# testing with small subset of data
```

[23]:
```
subset = data.sample(n=100000, random_state=42)
```

[24]:
```
X = subset.drop(columns = ['sellingprice', 'mmr'])
X.reset_index(drop=True, inplace=True)
```

[25]:
```
Y = np.array(subset[['sellingprice']])
```

Data exploration

[26]:
```
data.info()
```

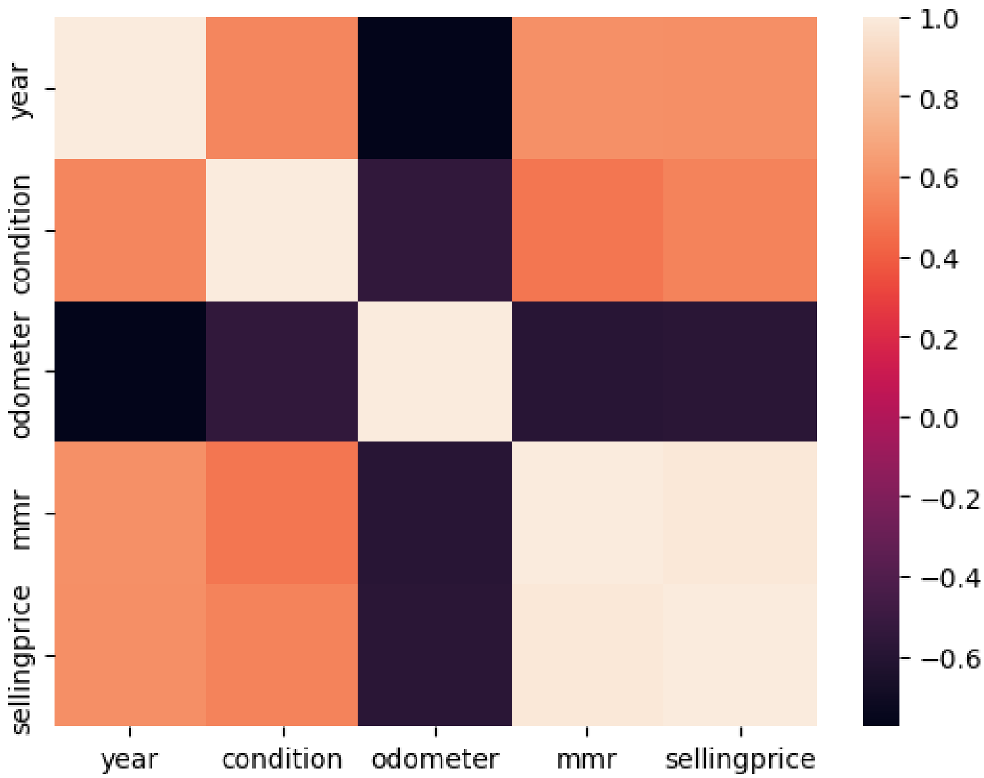
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 544795 entries, 0 to 558810
Data columns (total 12 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   year          544795 non-null  int64
 1   trim          544795 non-null  object
 2   body          544795 non-null  object
 3   transmission  544795 non-null  object
 4   state         544795 non-null  object
 5   condition     544795 non-null  float64
 6   odometer      544795 non-null  float64
 7   color         544795 non-null  object
 8   interior      544795 non-null  object
 9   mmr           544795 non-null  int64
 10  sellingprice  544795 non-null  int64
 11  make_model    544795 non-null  object
dtypes: float64(2), int64(3), object(7)
memory usage: 54.0+ MB
```

[27]:
```
corr = data.corr()
corr.sort_values(["sellingprice"], ascending = False, inplace = True)
print(corr.sellingprice)
```

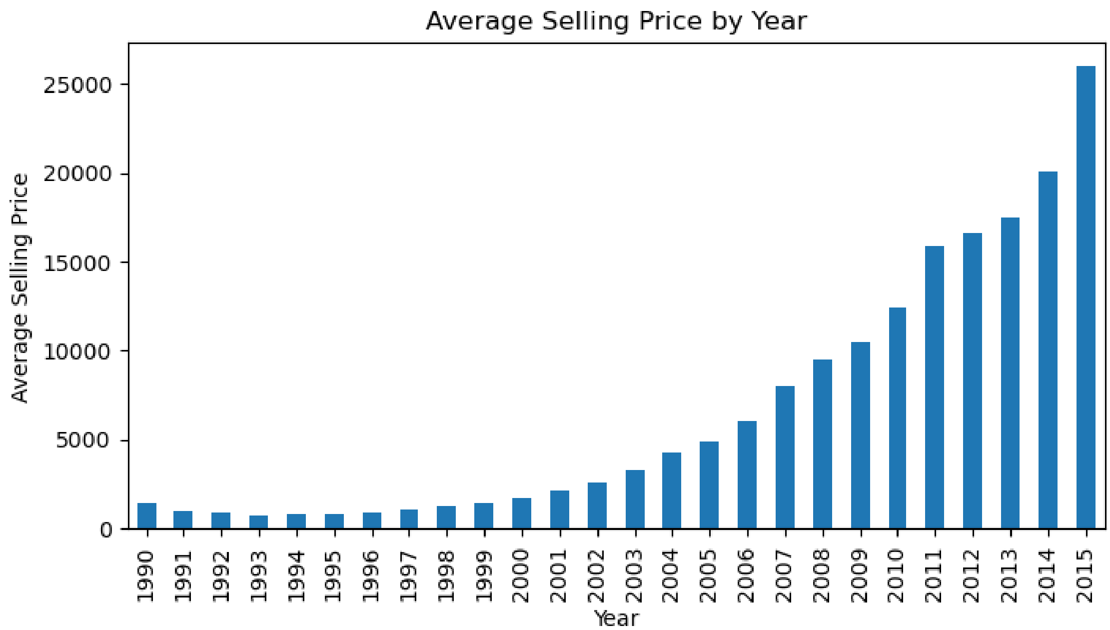
```
sellingprice    1.000000
mmr             0.983511
year            0.585672
condition       0.540597
odometer       -0.580802
Name: sellingprice, dtype: float64
```

[28]:
```
corr_matrix = data.corr()
sns.heatmap(corr_matrix)
```

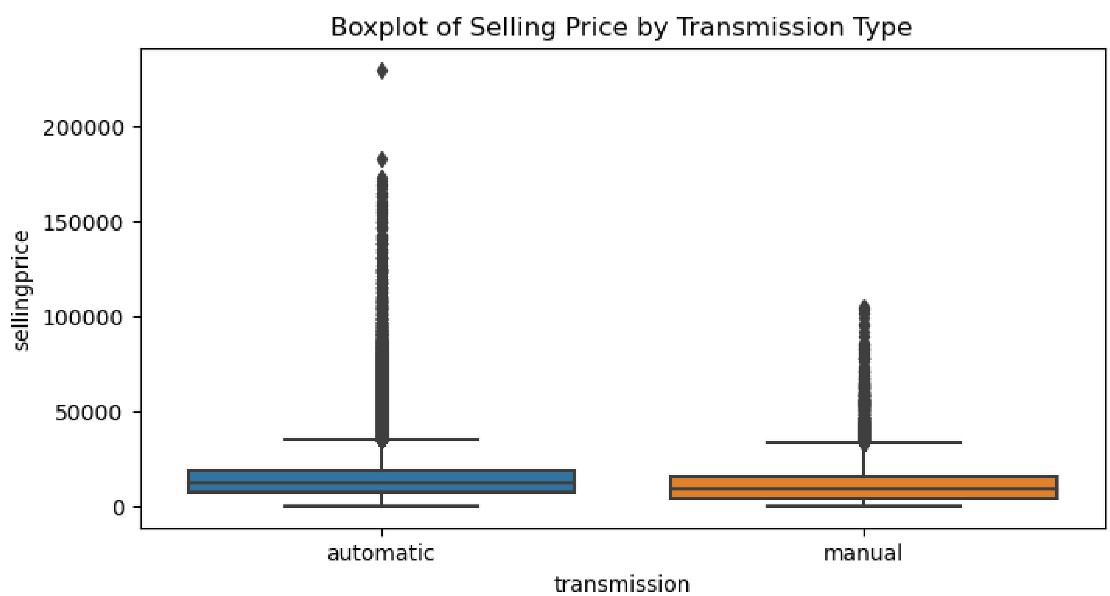
[28]: <Axes: >



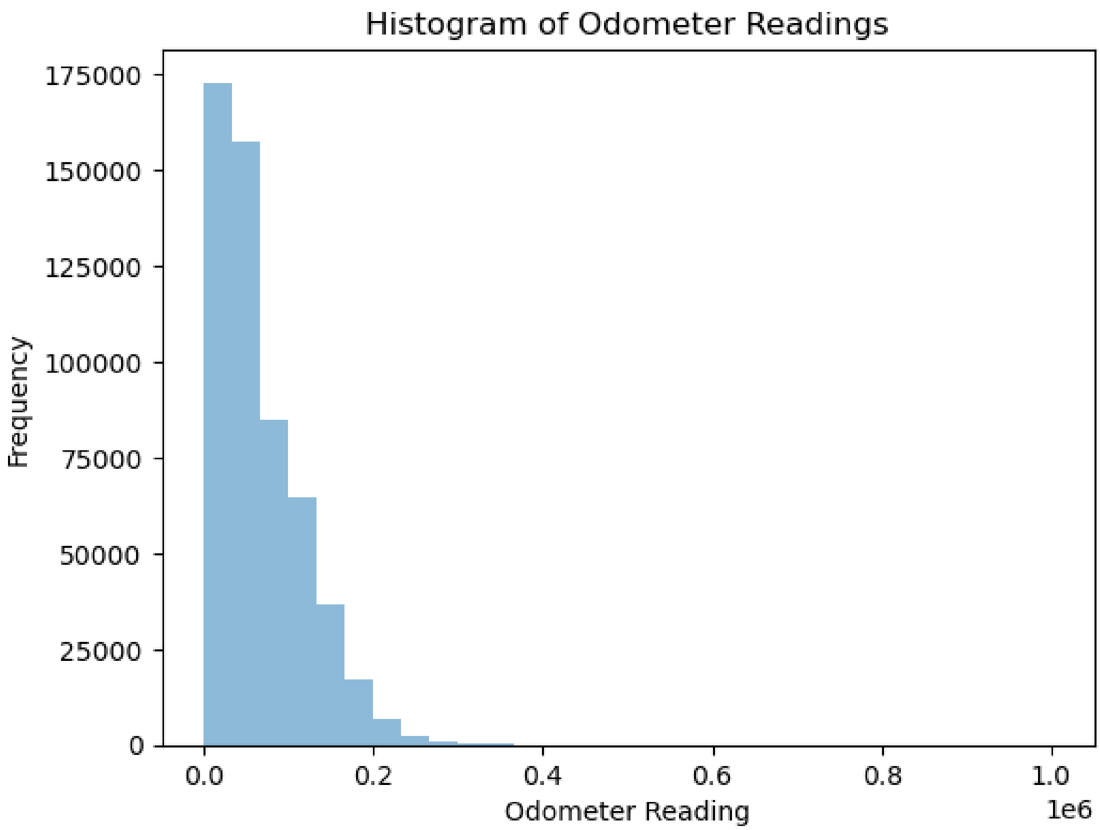
[65]:
```
# Bar plot of average selling price by year
plt.figure(figsize=(8,4))
data.groupby('year')['sellingprice'].mean().plot(kind='bar')
plt.xlabel('Year')
plt.ylabel('Average Selling Price')
plt.title('Average Selling Price by Year')
plt.show()
```



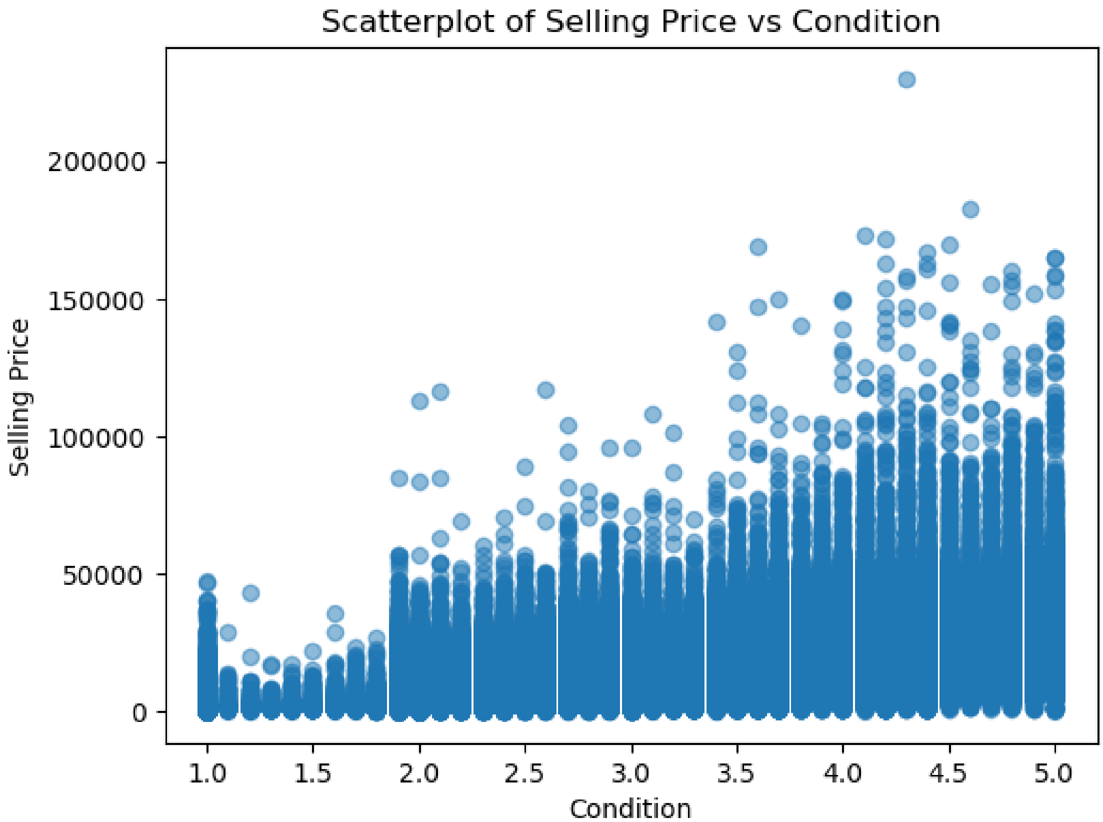
[66]:
```
# Box plot of selling price by transmission
plt.figure(figsize=(8,4))
sns.boxplot(x='transmission', y='sellingprice', data=data)
plt.title('Boxplot of Selling Price by Transmission Type')
plt.show()
```



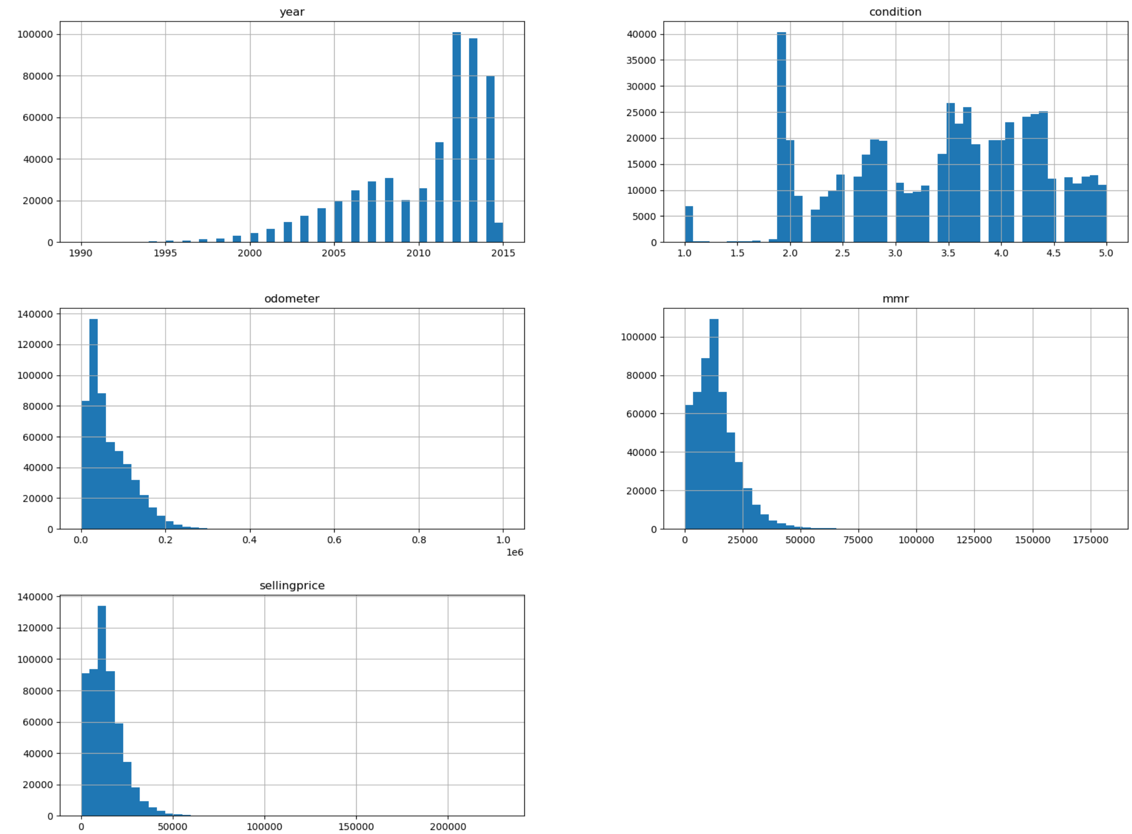
[31]:
```
# Histogram of Odometer Readings
plt.hist(data['odometer'], bins=30, alpha=0.5)
plt.xlabel('Odometer Reading')
plt.ylabel('Frequency')
plt.title('Histogram of Odometer Readings')
plt.show()
```



[32]:
```
# Scatterplot of Selling Price vs Condition
plt.scatter(data['condition'], data['sellingprice'], alpha=0.5)
plt.xlabel('Condition')
plt.ylabel('Selling Price')
plt.title('Scatterplot of Selling Price vs Condition')
plt.show()
```



[33]:
```
data.hist(bins=50, figsize=(20,15))
plt.show()
```



Data processing

[34]:
```
# turning categorical columns to numerical depending on the type of categorical variable
ordinal_features = ['year']
categorical_features = ['transmission','state','make_model']
numerical_features = ['condition','odometer']
```

[35]:
```
num_pipeline = Pipeline([
            ('scaler', StandardScaler())
])
```

[36]:
```
ordinal_pipeline = Pipeline([
            ('ordinal',OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1))
])
```

[37]:
```
categorical_pipeline = Pipeline([
            ('categorical',OneHotEncoder(handle_unknown='ignore'))
```

```
])
```

[38]:
```
preprocessor = ColumnTransformer(
            transformers = [('num',num_pipeline,numerical_features),
                ('ord',ordinal_pipeline, ordinal_features),
                ('categ',categorical_pipeline,categorical_features)
            ]
)
```

Data modeling

[39]:
```
# Linear Regression Model
```

[40]:
```
linear_reg_pipeline = Pipeline([
            ('preproc', preprocessor),
            ('reg', LinearRegression())
])
```

[41]:
```
k = 5

indices = subset.index.values

# Initialize the cross-validation splitter
kf = KFold(n_splits=k, shuffle=True)

# Initialize arrays to store the test and train scores
cv_test_scores = np.zeros(k)
cv_train_scores = np.zeros(k)
cv_MSE = np.zeros(k)

# Perform cross-validation
for i, (train_index, test_index) in enumerate(kf.split(indices)):
    X_train, X_test = X.loc[list(train_index)], X.loc[list(test_index)]
    y_train, y_test = Y[train_index], Y[test_index]

    # Fit the pipeline on the training data
    linear_reg_pipeline.fit(X_train, y_train)

    # Evaluate using R^2 score on test data
    cv_test_scores[i] = linear_reg_pipeline.score(X_test, y_test)

    # Evaluate using R^2 score on train data
    cv_train_scores[i] = linear_reg_pipeline.score(X_train, y_train)

    # Evaluate using MSE
    predictions = linear_reg_pipeline.predict(X_test)
```

```
    cv_MSE[i] = mean_squared_error(y_test, predictions)

# Print the cross-validation scores
print("Cross-validation test scores:", cv_test_scores)
print("Mean CV test R^2 score:", cv_test_scores.mean())
print("Cross-validation train scores:", cv_train_scores)
print("Mean CV train R^2 score:", cv_train_scores.mean())
print("Mean CV MSE:", cv_MSE.mean())
```

```
Cross-validation test scores: [0.80775288 0.82026531 0.81023046 0.80639941
0.8229214 ]
Mean CV test R^2 score: 0.813513891757966
Cross-validation train scores: [0.82732146 0.8252911  0.82589449 0.83050484
0.82686002]
Mean CV train R^2 score: 0.8271743814918462
Mean CV MSE: 17522683.99178185
```

---

[42]:
```
# Polynomial Regression with Degree 2
desired_degree = [2]
poly_reg_pipeline = Pipeline([
    ('preproc', preprocessor),
    ('poly', PolynomialFeatures(desired_degree)),
    ('reg', LinearRegression())
])
```

[43]:
```
cv_scores_mean = []
cv_train_scores_mean = []

for deg in desired_degree:
    poly_reg_pipeline.named_steps['poly'].degree = deg
    k = 5

    indices = subset.index.values

    # Initialize the cross-validation splitter
    kf = KFold(n_splits=k, shuffle=True)

    # Initialize an array to store the scores
    cv_test_scores = np.zeros(k)
    cv_train_scores = np.zeros(k)
    cv_MSE = np.zeros(k)

    # Perform cross-validation
    for i, (train_index, test_index) in enumerate(kf.split(indices)):
        X_train, X_test = X.loc[list(train_index)], X.loc[list(test_index)]
        y_train, y_test = Y[train_index], Y[test_index]
```

```
        # Fit the pipeline on the training data
        poly_reg_pipeline.fit(X_train, y_train)

        # Evaluate using R^2 Scoring on the test data
        cv_test_scores[i] = poly_reg_pipeline.score(X_test, y_test)

        # Evaluate using R^2 Scoring on the training data
        cv_train_scores[i] = poly_reg_pipeline.score(X_train, y_train)

        # Evaluate using MSE
        predictions = poly_reg_pipeline.predict(X_test)
        cv_MSE[i] = mean_squared_error(y_test, predictions)


    # Print the cross-validation scores
    print(f"Cross-validation test scores Polynomial Degree {deg}:",␣
  ↪cv_test_scores)
    print("Mean CV test R^2 score:", cv_test_scores.mean())
    print(f"Cross-validation train scores Polynomial Degree {deg}:",␣
  ↪cv_train_scores)
    print("Mean CV train R^2 score:", cv_train_scores.mean())
    print("Mean CV MSE:", cv_MSE.mean())
    cv_scores_mean.append(cv_test_scores.mean())
    cv_train_scores_mean.append(cv_train_scores.mean())
```

```
Cross-validation test scores Polynomial Degree 2: [0.87336313 0.90821349
0.90755686 0.91431875 0.90520735]
Mean CV test R^2 score: 0.9017319143834508
Cross-validation train scores Polynomial Degree 2: [0.93421824 0.92941467
0.93039654 0.92956963 0.93226543]
Mean CV train R^2 score: 0.93117290129245
Mean CV MSE: 9269615.68814611
```

---

[44]:
```
#Lasso Regression
```

[45]:
```
lasso = Pipeline([
            ('preproc', preprocessor),
            ('lasso', Lasso(alpha = 0.001))
])
```

[46]:
```
k = 5

indices = subset.index.values

# Initialize the cross-validation splitter
```

```
kf = KFold(n_splits=k, shuffle=True)

# Initialize an array to store the scores
cv_test_scores = np.zeros(k)
cv_train_scores = np.zeros(k)
cv_MSE = np.zeros(k)

for i, (train_index, test_index) in enumerate(kf.split(indices)):
    X_train, X_test = X.loc[list(train_index)], X.loc[list(test_index)]
    y_train, y_test = Y[train_index], Y[test_index]

    # Fit the pipeline on the training data
    lasso.fit(X_train, y_train)

    # Evaluate the pipeline on the test data
    cv_test_scores[i] = lasso.score(X_test, y_test)

    # Evaluate the pipeline on the training data
    cv_train_scores[i] = lasso.score(X_train, y_train)

     # Evaluate using MSE
    predictions = lasso.predict(X_test)
    cv_MSE[i] = mean_squared_error(y_test, predictions)

# Print the cross-validation scores
print("Cross-validation test scores Lasso Regression:", cv_test_scores)
print("Mean CV test R^2 score:", cv_test_scores.mean())
print("Cross-validation train scores Lasso Regression:", cv_train_scores)
print("Mean CV train R^2 score:", cv_train_scores.mean())
print("Mean CV MSE:", cv_MSE.mean())
```

```
Cross-validation test scores Lasso Regression: [0.8165692  0.8122635  0.82851745
0.78823466 0.81633291]
Mean CV test R^2 score: 0.8123835436098231
Cross-validation train scores Lasso Regression: [0.8242872  0.82541508 0.8251388
0.83406257 0.8269062 ]
Mean CV train R^2 score: 0.8271619709907352
Mean CV MSE: 17621479.788961075
```

---

[47]:
```
# Model estimation
```

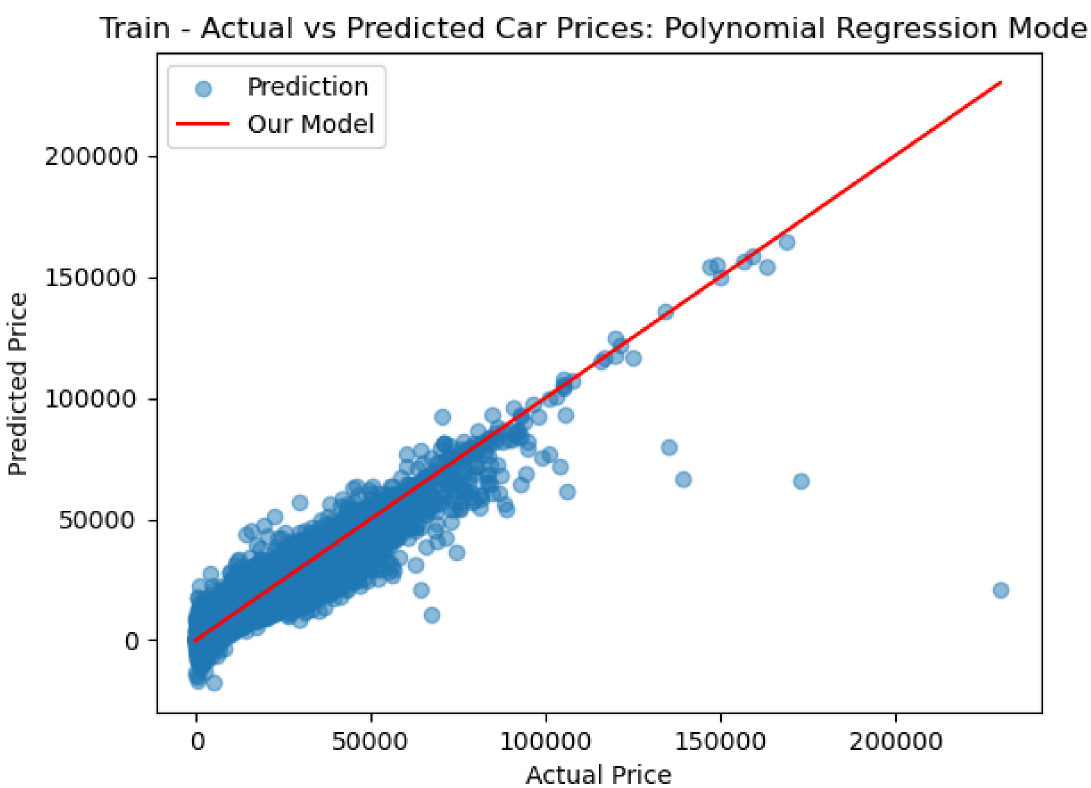
[64]:
```
# What are the final parameter estimates?

# Extract the LinearRegression model from the pipeline
linear_reg_model = poly_reg_pipeline.named_steps['reg']
```

```
# Get the coefficients
coefficients = linear_reg_model.coef_
coefficients
```

[64]:
```
array([[ 2.65354472e-09,  3.43916632e+02, -2.33617921e+02, …,
        -1.28112232e+03,  0.00000000e+00,  2.97281966e+03]])
```

[60]:
```
poly_reg_pipeline.fit(X, Y)
final_accuracy = poly_reg_pipeline.score(X, Y)
predicted_prices = poly_reg_pipeline.predict(X)
```

[61]:
```
plt.scatter(Y, predicted_prices, alpha=0.5)
plt.plot([Y.min(), Y.max()], [Y.min(), Y.max()], color='red')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Train - Actual vs Predicted Car Prices: Polynomial Regression Model')
plt.legend(['Prediction', 'Our Model'])
plt.show()
```



[50]:
```
poly_reg_pipeline.fit(X_test, y_test)
final_accuracy = poly_reg_pipeline.score(X_test, y_test)
```

```
predicted_prices = poly_reg_pipeline.predict(X_test)
```

[54]:
```
plt.scatter(y_test, predicted_prices, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
 ↪color='red')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Test - Actual vs Predicted Car Prices: Polynomial Regression Model')
plt.legend(['Prediction', 'Our Model'])
plt.show()
```