Capstone 2 - US Used Car Sales Predictive Model

A. Objective:

To develop a predictive model that automates the domestic used car sales market appraisal and estimation/evaluation process. The main driving force behind the development of this model is to minimize laborious efforts that go into researching used car sales data (e.g. what can I get for a 2011 toyota camry SE?) by deploying a model built using historical data with the ability to consume new data (leveraging Bayesian principles – ideal, future state model).

B. Assumptions and Exclusions:

B1. Key Assumptions/Comments →

- Prices are set by a free market (consumer demand versus available supply)
- Some potential key features were missing from this analysis: color, general condition of the car (ordinal scaling), damage/vehicle inspection history, etc.
- Accounted for missing values and erroneous/extreme data points after visualizing distributions of each variable

B2. Key Exclusions →

- Price data was missing from about 9.24% of the dataset; these observations were dropped as a result
- Some features in the original dataset were excluded due to them having an assumed low impact: 'id', 'vin', 'stock_no', 'street', 'seller_name', 'zip', 'city'
    - The vin and stock number are too granular for this analysis and would result in a ridiculously high dimensionality
    - I left the 'state' categorical variable to incorporate the spatial context; this helps to better understand the geographical impact on price
    - The 'seller_name' was not considered fundamental for this analysis, as I'm looking to find the true demand for a given car irrespective of what a specific entity sells said car for

---

C. Data Wrangling, EDA and Preprocessing:

C1. Data Collection and Wrangling →

- Imported the original US Car sales dataset (from Kaggle.com)
    - 1) the original dataset contained 7,104,304 used car sales transactions and incorporated 21 features (see fig 1 on the next page)
        - Excluded 7 features as mentioned in section B2
    - 2) manually added in the manufacturer origin (country)

```
id                object
vin               object
price            float64
miles            float64
stock_no          object
year             float64
make              object
model             object
trim              object
body_type         object
vehicle_type      object
drivetrain        object
transmission      object
fuel_type         object
engine_size      float64
engine_block      object
seller_name       object
street            object
city              object
state             object
zip               object
```

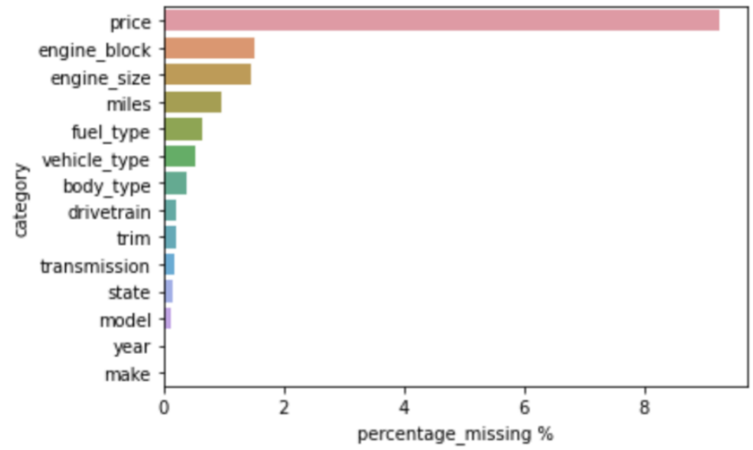
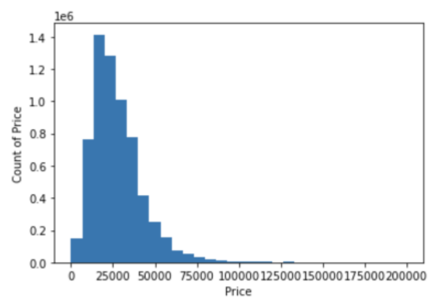


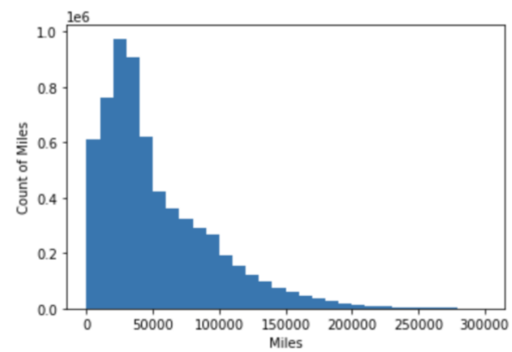
Fig. 1 – Original Dataset Features **Fig. 2** – Missing data

C2. Initial Inspection of numerical distributions →

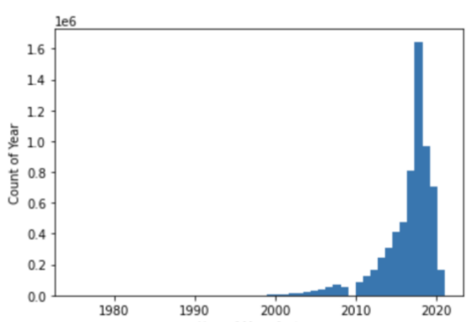



```
   25th Percentile  50th Percentile  75th Percentile  95th Percentile
0          16999.0          24500.0          34995.0          55799.0

   < 200k (%)  > 200k (%)
0   99.911392    0.088608
```



Figs. 3 and 4 – Price Distribution (left) and Mileage Distribution (right)



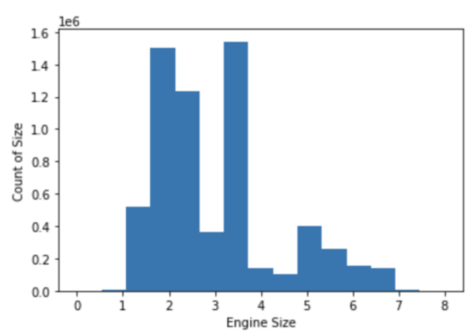


Fig. 5 – Year of Manufacturer Distribution **Fig. 6** – Engine Size Distribution

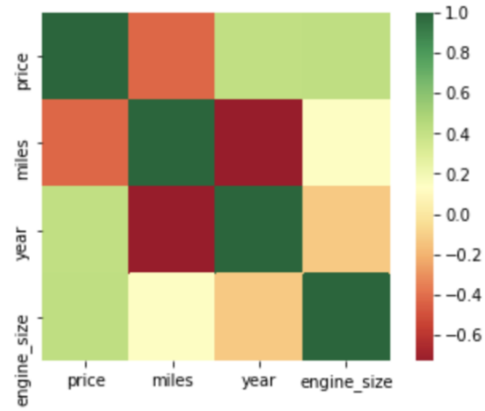


Fig. 6 – Numerical Feature Correlation

C3. Manual inspection of the values for categorical data and cleanup →

- Quite a bit of manual effort went into standardizing the naming conventions for the makes, models, fuel type and trim (example shown below)

```
# replacing some obvious errors; went down to 1176 unique values from 1204
df_us_n.replace({'model' : { '3-series' : '3 series', '3000gt' : '3000 gt', '5-series' : '5 series'
                           , '6-series' : '6 series', '7-series' : '7 series', 'cl' : 'cl-class'
                           , 'cl class' : 'cl-class', 'cla' : 'cla-class', 'clk' : 'clk-class', 'e-350' : 'e350'
                           , 'e-450sd' : 'e450', 'f150' : 'f-150', 'f350' : 'f-350', 'f750':'f-750','f-350sd':'f-350 sup
                           , 'f-450sd':'f-450 super duty', 'f-550sd':'f-550 super duty','legacy wagon' : 'legacy'
                           , 'ram pickup' : 'ram 1500 pickup', 'ram' : 'ram 1500 pickup', 'ram 1500' : 'ram 1500 pickup'
                           , 'ram 150' : 'ram 1500 pickup', 'ram 250' : 'ram 2500 pickup', 'ram pickup 2500' : 'ram 2500
                           , 'ram 3500 cab chassis' : 'ram 3500 chassis cab', 'ram 3500' : 'ram 3500 pickup', 'ram 4500
                           , 'xl-7':'xl7','transit crew van':'transit passenger van'}},inplace = True)

df_us_n['model'] = df_us_n.model.str.replace('[é]','e')
```

```
# length is now 1176 unique values
df_us_n['model'].nunique()
```

```
1176
```

Fig. 7 – Model cleanup

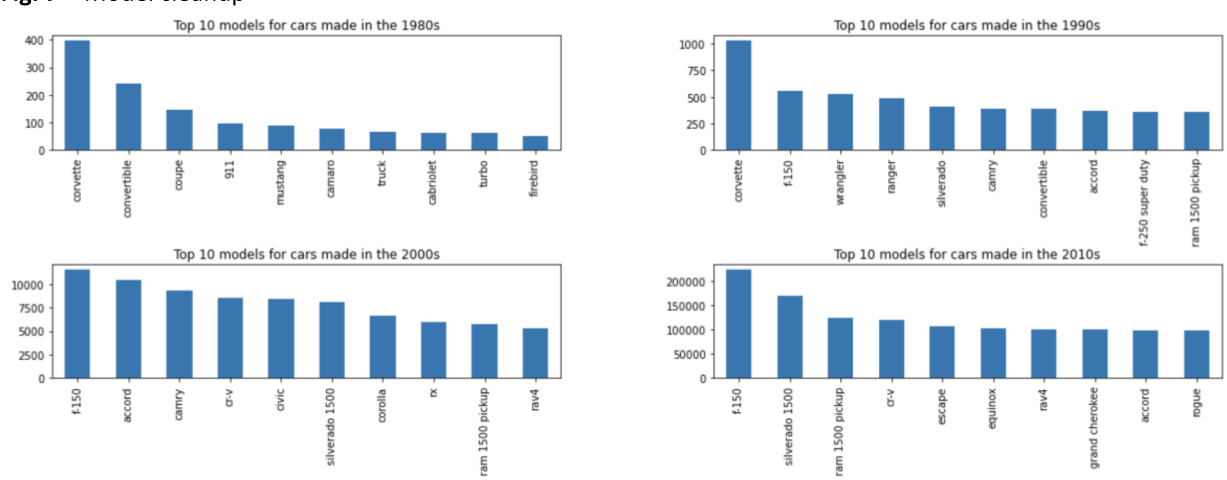


Fig. 8 – Top models sold by decade produced

```
# now it is time to inspect the trim feature
trim_cleanup = pd.DataFrame([Counter(df_us_n['trim'].sort_values())])
trim_cleanup = trim_cleanup.transpose()
# original length was 2342 prior to cleanup
# after inspection, it looks like there are quite a few issues: capitalization/inconsistencies regarding capitalizatio
# spacing, naming conventions, registered trademark symbol being applied by certain dealers, etc.
```

```
# applying the lower method to the trim column
df_us_n['trim'] = df_us_n['trim'].map(lambda x: x.lower() if isinstance(x,str) else x)
# went from 2342 to 2223.. not bad
```

```
df_us_n.replace({'trim' : { 'clk 320' : 'clk320', 'e 300' : 'e300', 'e 320' : 'e320', 'e 420' : 'e420'
                          , 'e 430' : 'e430', 'e320 4matic sedan' : 'e320 4matic', 'e430 sedan' : 'e430'
                          , 'ec' : 'eco', 'ed bauer' : 'eddie bauer', 'es4wd' : 'es awd'
                          , '+' : 'plus', 'ev +' : 'ev plus', 'ex w/leather' : 'ex leather'
                          , 'ex-l v-6' : 'ex-l v6', 'i l.l. bean' : 'i l.l. bean edition'
                          , 'inspiration' : 'inspiration series', 'iroc z coupe' : 'iroc z convertible'
                          , 'landmark' : 'landmark edition', 'lt entertainer' : 'lt entertainment'
                          , 'luxury edition' : 'luxury collection', 'lx 2-door sedan' : 'lx'
                          , 'm edition' : 'm', 'ml 320' : 'ml320', 'night shade' : 'nightshade'
                          , 'north' : 'north edition', 'off road' : 'off-road', 'premiere edition' : 'premiere'
                          , 'prerunner v6' : 'prerunner', 'rally' : 'rally edition', 'se 4wd' : 'se awd'
                          , 'se off road': 'se off-road', 'signature l': 'signature limited', 'sle-1':'sle1'
                          , 'sle-2':'sle2', 'srt-10':'srt10', 'srt-4':'srt4', 'srt-6':'srt6', 'srt-8':'srt8'
                          , 'st line':'st-line', 'touring-l':'touring l', 'touring-l plus':'touring l plus'
                          , 'trd off-road':'trd off road', 'type r':'type-r', 'type s':'type-s'
                          , 'uptown':'uptown edition', 'v-6':'v6', 'work series':'work', 'xl 2wd':'xl'
                          , 'z/28':'z28', 'zx2 hot':'zx2', 'amg s 63':'amg s63', 'amg s 65':'amg s65'
                          , '75th anniversary': '75th anniversary edition', '60 years edition': '60th anniversary'
                          , '50th anniversary': '50th anniversary special edition', '50th anniversary edition': '50th a
                          , '50th anniversary': '50th anniversary special edition'}}, inplace = True)

df_us_n['trim'] = df_us_n.trim.str.replace('[®]','')
```

Fig. 9 – Trim cleanup

C4. Additional EDA →

- Because there were quite a few categorical features, it was important to visualize their distribution/impact across the dataset in a variety of ways; mainly, how did the trend changed over time
- As one would have expected, the newer cars (>2015) yielded a higher sale price, but what was interesting is how the FWD was continuously the lowest while the RWD/4WD median price changed based on the manufactured year
- Notice how the lowest valued cars appear to be made from 2000-2005; why could this be? Is it purely a supply v demand (meaning, most cars older then 15 years are kept in pristine condition/low mileage and are coveted by car enthusiasts) or is there a theme related to how a car is manufactured (do people just generally like the look/feel of cars in the 80s/90s moreso than the early 2000s)?
  - I believe the prevailing phenomena is the former of what was outlined above. There are significantly less cars sold that were produced before 2000 (see **Fig 12** on the next page)
  - Hence, there were 1787x more cars sold that were produced from 2011-2020 when compared to ones produced from 1980-1990

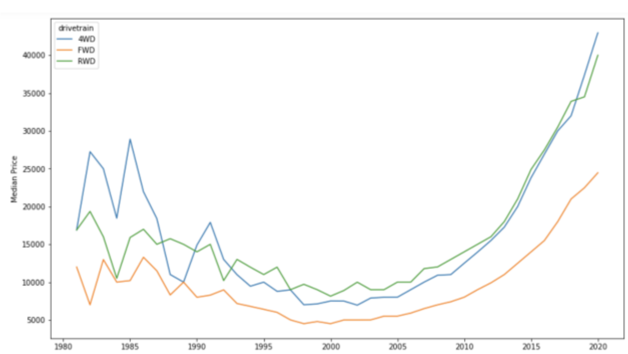


Fig. 10 – Median Price per Drivetrain Type by Year

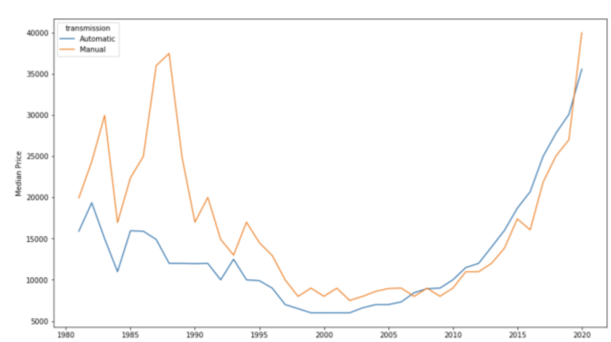
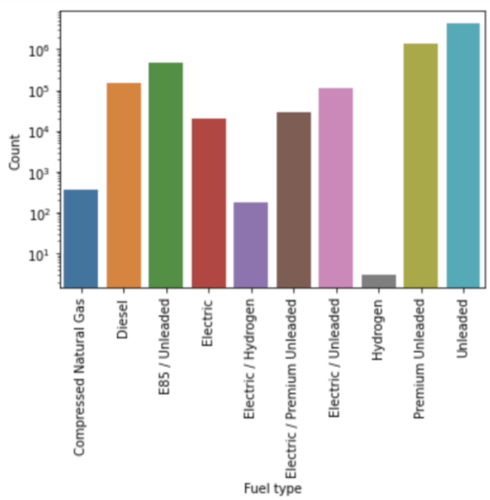


Fig. 11 – Median Price per Transmission Type by Year

```
Total number of cars sold that were produced from 1980 - 1990: 3331
Total number of cars sold that were produced from 1991 - 2000: 17585
Total number of cars sold that were produced from 2001 - 2010: 400676
Total number of cars sold that were produced from 2011 - 2020: 5955196
```

Fig. 12 - # of cars sold per manufactured decade

- For the next investigated step, I cleaned up the fuel types, and consolidated the naming conventions
- After the cleanup, the distribution of counts/observations of the various fuel types can be seen in **Fig. 13,** shown below
- Additionally, I wanted to visualize the relationship between price, fuel type and year of manufacture to get a better understanding of how the market has changed over time



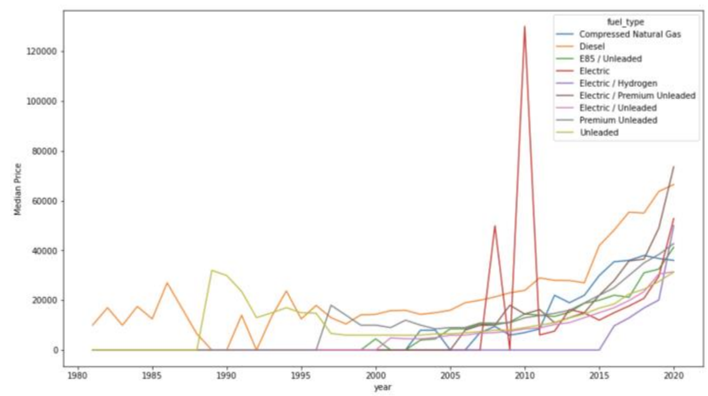


Fig. 13 – Fuel Type Counts

Fig. 14 – Price based on Fuel Type and Year

- As part of the analysis, I included the “country of origin” feature to better visualize how price is affected by the manufacturer’s country; as one would expect, Italy, Germany and England are consistently higher than the other nations
- Older foreign cars appear to be more valued/coveted by car enthusiasts

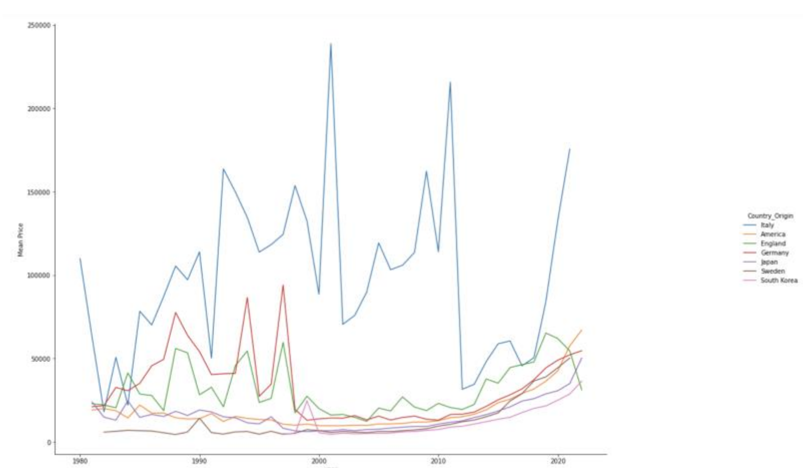


Fig. 15 – Country of Origin, Price and Year of Manufacture

Fig. 16 – Price v Miles by Make

- The scatter plot shown above is a bit busy, but it uses a log scale for the x axis (price)
- The emphasis on this plot is to identify some outliers, as mileage and price are correlated
  - Notice how some cars > $100k while sporting >150,000 miles; this is quite compelling and shows that some models have been deemed to warrant a heavy premium compared to the population as a whole
- The second scatter plot, shown below, brings the data up a few levels
  - Here you can see that Italian models are typically higher in price and lower in mileage when compared to the other countries of origin
  - Swedish cars are typically the cheapest and have quite a bit of mileage

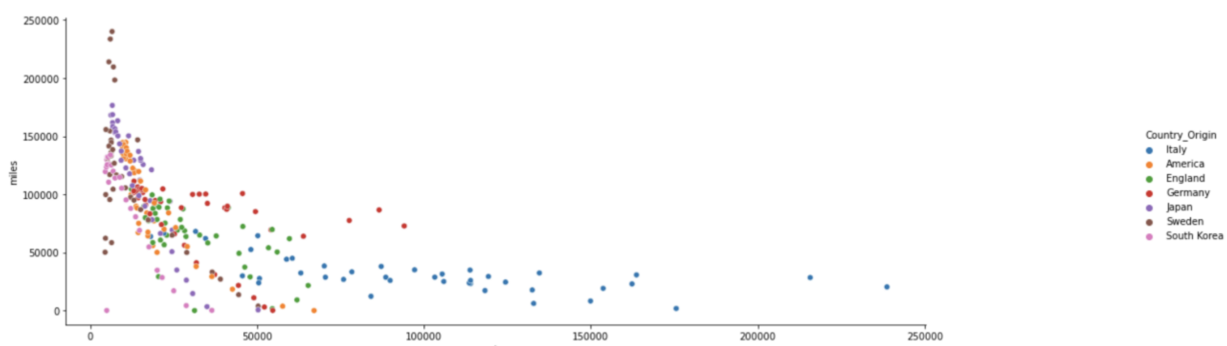


Fig. 17 – Price v Miles by Country of Origin

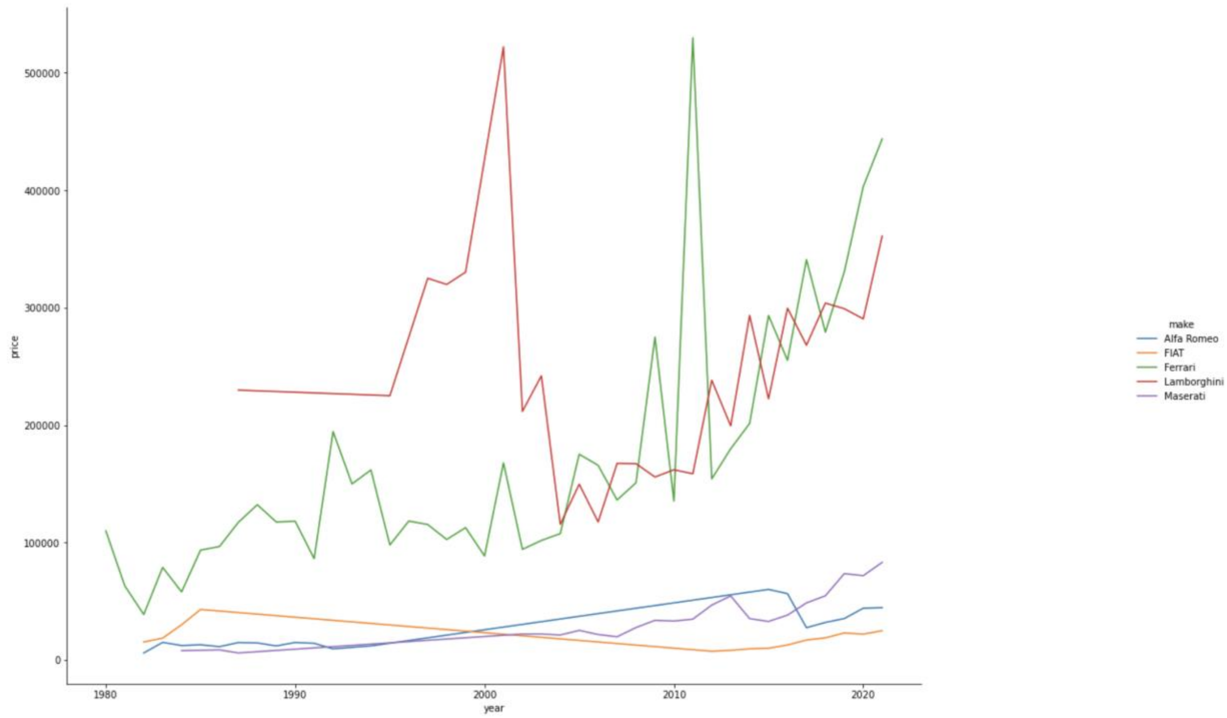


Fig. 18 – Closer look at Italian makes

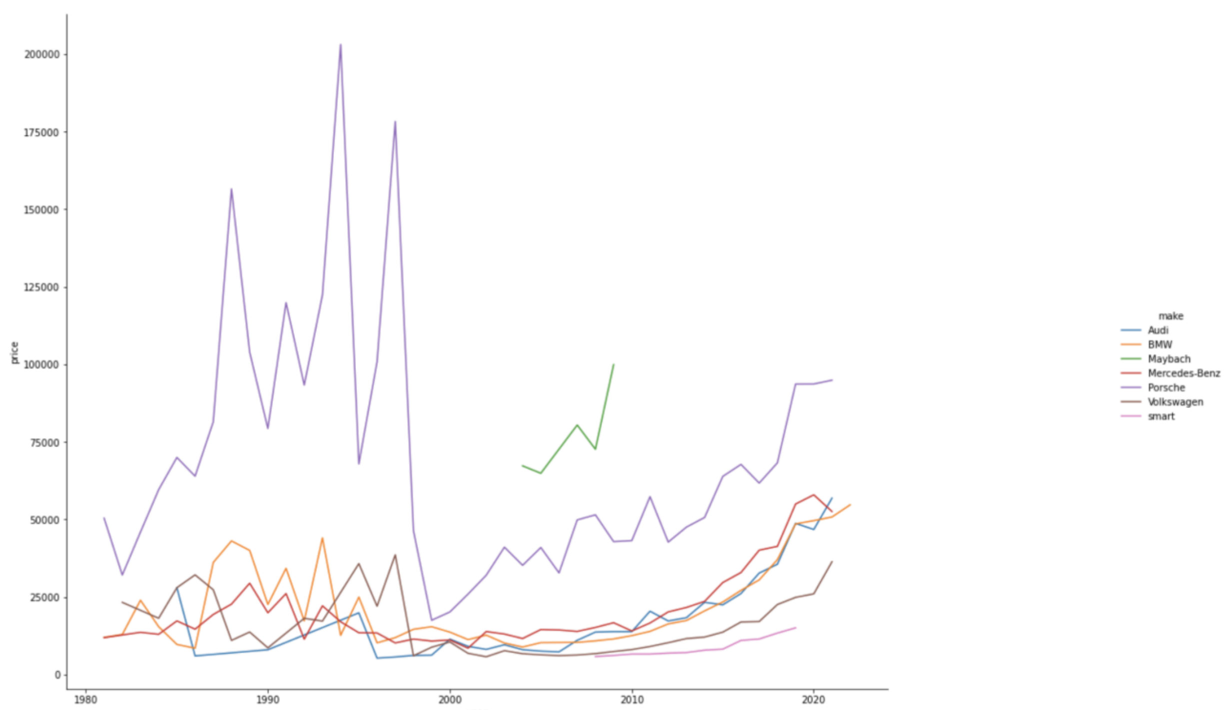


Fig. 19 – Closer look at German makes

C5. Preprocessing →

- After dropping the rows that had no associated price, I had quite a few variables with missing values (see **Fig. 20** below)
- Initially, I planned on using the missingforests imputation algorithm (or something a bit more intelligent), but it required too much time and was computationally expensive
    - Realistically, because the % of values missing is so small, a simple imputation approach is sufficient
- Because my intention was to use a tree-based regressor or a neural network, it made sense to just fill the missing categorical values with "missing" and the missing numerical values with something that would identify them as being different: miles = -10,000, year = 1900, engine_size = 1.0

	category	summation	percentage_missing %
0	engine_block	62749	0.97
1	engine_size	59884	0.93
2	miles	35264	0.55
3	fuel_type	32295	0.50
4	vehicle_type	28129	0.44
5	body_type	19804	0.31
6	state	11371	0.18
7	drivetrain	9613	0.15
8	trim	9473	0.15
9	transmission	8078	0.13
10	model	5526	0.09
11	year	77	0.00
12	Country_Origin	0	0.00
13	make	0	0.00
14	price	0	0.00

Fig. 20 – remaining missing values

```
# some simple imputation methods that can be applied to the entire dataset since I'm not using any mathematical techniq
# such as applying the mean/median; this will not affect the generalization ability of the training model(s)

X = X.fillna({'miles':-10000,'year':1900,'engine_size':1.0})
X[categorical] = X[categorical].fillna(value='missing')

# changing all object datatypes to "category"; this is a necessary step in order for the LGBMRegressor() to properly ha

for col in X.columns:
    if X[col].dtypes == 'object':
        X[col] = X[col].astype('category')
```

Fig. 21 – imputing the rest of the missing values

- After completing the imputation of all missing values, I went ahead and created a custom dummy transformer that would handle the high dimensionality encoding requirements (fig 22 on the next page); this was necessary for testing out some of the regressors (RandomForestRegressor, LinearRegressor, XGBRegressor)
- Once the categorical variables were successfully encoded, I was looking to scale the numerical variables
- The aformentioned steps were part of the pre_pipeline, that when combined with the model, would complete the full_pipeline as shown in Fig. 23 on the next page

```python
from sklearn.base import TransformerMixin, BaseEstimator, clone

class ToDummiesTransformer(BaseEstimator, TransformerMixin):
    """ A Dataframe transformer that provide dummy variable encoding
    """

    def transform(self, X, **transformparams):
        """ Returns a dummy variable encoded version of a DataFrame

        Parameters
        ----------
        X : pandas DataFrame

        Returns
        ----------
        trans : pandas DataFrame

        """

        trans = pd.get_dummies(X).copy()
        return trans

    def fit(self, X, y=None, **fitparams):
        """ Do nothing operation

        Returns
        ----------
        self : object
        """
        return self
```

Fig. 22 – Custom Dummy Encoder

```python
# initializing the XGBRegressor()
# initial model took 2.97 hours and had a MSE of 28,930,331.50 using default parameters/hyperparameters
m1 = XGBRegressor()

# initializing the LinearRegressor()
# initial model took .131 hours and had a MSE of 91,910,606.69 using default parameters/hyperparameters
m2 = LinearRegression(normalize=True)

# initializing the LGBMRegressor()
# initial model took .043 hours to run and had a MSE of 33,920,875.33 using default parameters/hyperparameters
# don't need to encode - it does integer encoding for me
# will have to change data types to "categorical"
# will handle the nulls; natively supports GPUs and parallelization; it has it's own feature importance
m3 = LGBMRegressor()

# initializing the RandomForestRegressor()
# initial model took .86 hours to run and had a MSE of 195,518,841.5 with a max depth of 3
# second model took 3.04 hours to run and had a MSE of 12,898,790.17 with a max depth of 20
m4 = RandomForestRegressor(max_depth=20, random_state=0, oob_score = True)

# initializing the RidgeRegressor() with the default alpha; code for looping through various alpha values is in the nex
m5 = Ridge(solver="sag", random_state=42)

# Defining the preprocessing pipeline and initial model
pre_pipeline = ColumnTransformer([
    ('numerical', MinMaxScaler(), numerical),
    ('categorical', ToDummiesTransformer(), categorical)])

full_pipeline = Pipeline([('preprocessing', pre_pipeline),('model', m5)])
```

Fig. 23 – Initial Pipeline

D. <u>Modelling:</u>

D1. Initial Discovery Phase →

- Based on preprocessing steps defined in section C5, I was able to test 4 different algorithms on the training data (80/20 split)
    - The caveat here is that because my categorical variables created a wide encoded dataset, I was forced to exclude the trim and model features to run some of the more complex models
    - This was not ideal and led me to focusing my efforts on utilizing the LGBMRegressor, as it has more capabilities for handling higher dimensions with ease
    - The RidgeRegressor failed due to a lack of memory available

- The MSE scores of the initial models were as follows:

Algorithm:	Timing to run model:	MSE:
XGBRegressor	2.97 hours	28,930,331.50
LinearRegressor	.131 hours	91,910,606
LGBMRegressor	.043 hours	33,920,875.33
RandomForestRegressor	.86 hours	195,518,841

- Based on the above table, it appears as though the XBGRegressor could be the winner; after considering the necessary computational requirements and model flexibility, the LGBMRegressor was a more applicable algorithm
  - The main reason for going with the LGBMRegressor over the XGBRegressor was the native capabilities of the LGBMRegressor to effortlessly handle categorical variables without having to encode; this allowed me to incorporate all pertinent features instead of having to drop the model and trim features because of resource limitations

D2. Baseline Model: LGBM Regressor (not optimized) →

```
Time to run the Model (hrs):  0.009779014653629728
Mean squared error:  17030493.10822307
Mean absolute error:  2181.0350781143466
sq_rt(Mean squared error):  4126.801801422388
```

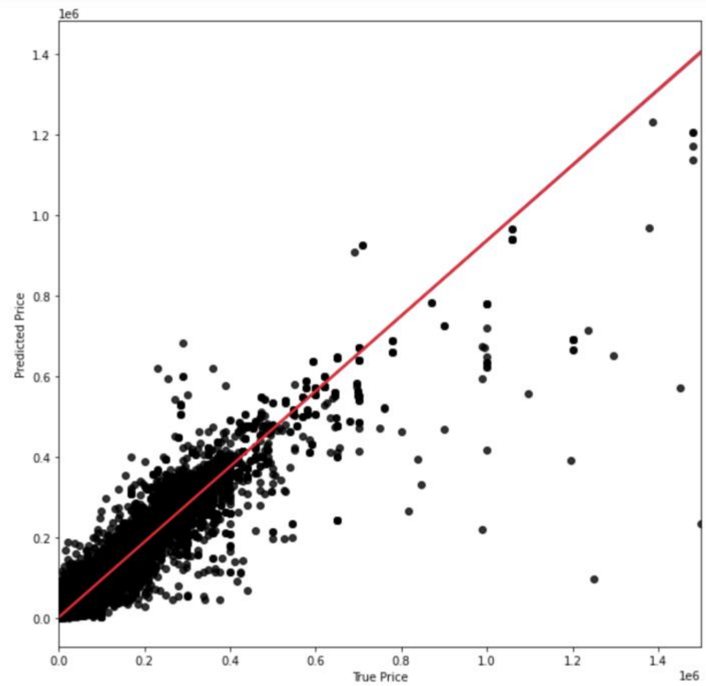


Fig. 24 – Training Data Performance

- Ideally, I would've like to use the RSMLE (log error) metric, but the y_pred forecasted some negative values and the RSMLE metric cannot handle negative values
  - Presumably because of the imputation method
  - Now, there are some simple solutions to get around the error, but the combination of MSE, MAE and sq_rt(MSE) was sufficient for tuning

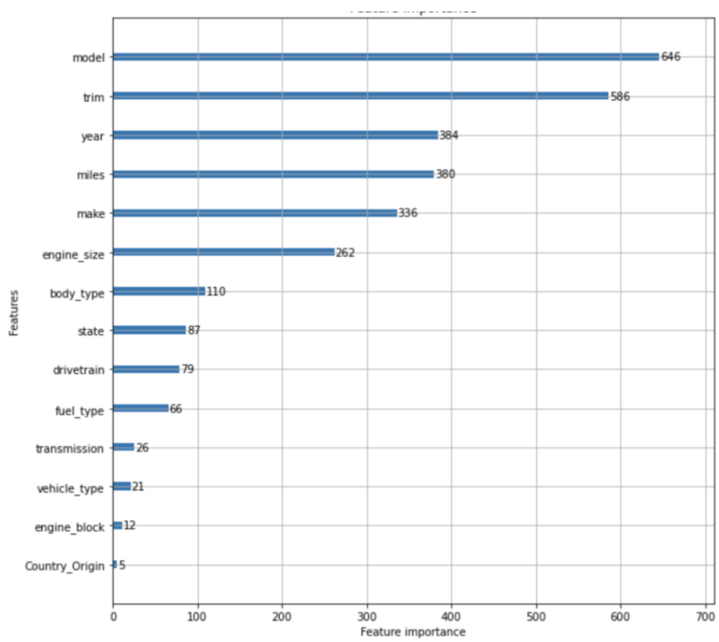


Fig. 25 – Training Model Feature Importance

- Good thing I kept the model and trim categories, as those appear to be the most important after fitting and evaluating the model on the training data; intuitively, this makes sense based on the EDA
  - Why? Because as I discussed in the EDA portion of the analysis, you have some high-mileage cars that yield incredibly high prices based on their “model” and type of model/ “trim”

D3. Optimized Model: LGBM Regressor (tuning the hyperparameters) →

- To tune the base model, I went ahead and leveraged the sklearn RandomizedSearchCV package
- The hyperparameters in focus are shown below in **Fig.** 26

```
# hyperparameter tuning using RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from scipy.stats import uniform, truncnorm, randint
start = time.time()

lgbm_model = LGBMRegressor()
# iter_ = 1000 for RandomizedSearchCV
lgbm_params = {}
lgbm_params['learning_rate'] = [0 + (float(i) / 100 ) for i in range(0, 101)]
lgbm_params['boosting_type'] = ['gbdt', 'dart', 'goss','rf']
lgbm_params['objective'] = ['regression']
lgbm_params['feature_fraction'] = [0 + (float(i) / 100 ) for i in range(0, 101)]
lgbm_params['tree_learner'] = ['serial','feature','data','voting']
lgbm_params['max_depth'] = [0 + (int(i)) for i in range(1, 301)]
lgbm_params['n_estimators'] = [0 + (int(i)) for i in range(1, 301)]

# executing the GridSearchCV based on the model hyperparameters; I attempted to use RandomizedSearchCV, but ran into
# fitting errors
lgbm_gs_model = RandomizedSearchCV(lgbm_model, param_distributions = lgbm_params, n_jobs = 8, scoring = 'neg_mean_squar
lgbm_gs_model.fit(X_train, y_train)
lgbm_best_parameters = lgbm_gs_model.best_params_
y_pred_best = lgbm_gs_model.predict(X_train)

mae_hyp = mean_absolute_error(y_train, y_pred_best)
rsme_hyp = np.sqrt(mean_squared_error(y_train, y_pred_best))

end = time.time()
print('Time to run the Model (hrs): ', (end - start)/60/60)
print(lgbm_gs_model.best_estimator_.get_params())
print('MAE: ', mae_hyp, ' \n')
print('SQRT MSE: ',rsme_hyp, ' \n')
```
Fig. 26 – RandomizedSearchCV Hyperparameter tuning

```
[LightGBM] [Warning] feature_fraction is set=0.64, colsample_bytree=1.0 will be ignored. Current value: feature_fract
ion=0.64
Time to run the Model (hrs):  4.85794770055347
{'boosting_type': 'dart', 'class_weight': None, 'colsample_bytree': 1.0, 'importance_type': 'split', 'learning_rate':
0.53, 'max_depth': 65, 'min_child_samples': 20, 'min_child_weight': 0.001, 'min_split_gain': 0.0, 'n_estimators': 27
0, 'n_jobs': -1, 'num_leaves': 31, 'objective': 'regression', 'random_state': None, 'reg_alpha': 0.0, 'reg_lambda':
0.0, 'silent': True, 'subsample': 1.0, 'subsample_for_bin': 200000, 'subsample_freq': 0, 'tree_learner': 'data', 'fea
ture_fraction': 0.64}
MAE:  1858.322610118308

SQRT MSE:  3298.037521361062
```

Fig. 27 – Timing, Best Parameters and Scoring of the Tuned Model

```
feature_importances = lgbm_gs_model.best_estimator_.feature_importances_
lgbm_cols = X_train.columns
top_params = pd.DataFrame({'feature names': lgbm_cols, 'feature importances'
sns.barplot(data = top_params, y = 'feature names', x = 'feature importances
plt.show()
```

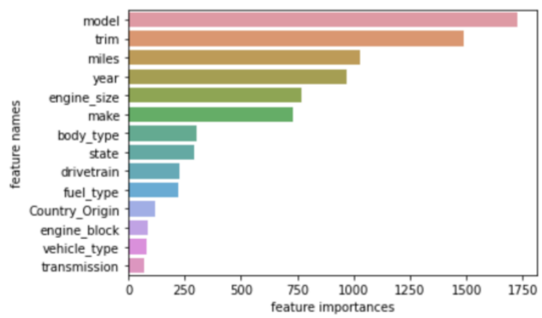


Fig. 28 – Updated Feature Importance

- The base model had a SQRT MSE of 5,824.16 while the tuned model was 3,298.03; that marks a significant improvement
- The only thing left to do is to apply the tuned model to the test dataset after scaling the test dataset

```
Time to run the Model (hrs):  0.11182119442356958

MSE:  14883891.341055792

MAE:  1876.6653906024278

SQRT MSE:  3857.964663012842
```

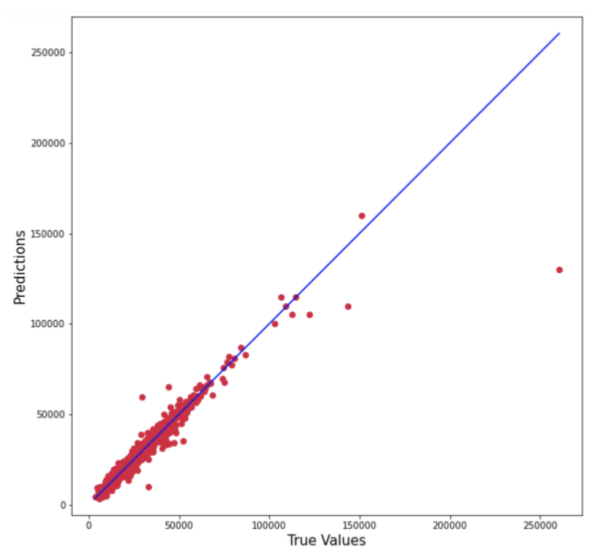


Fig. 29 – Test Data Scores and Visualizing 1000 Predictions

- The test dataset scored a little bit lower than the training data, but it is clear that the model is generic enough to be applied/utilized on unseen data

E. Lessons Learned and Comments:

- This project served as a great introduction into predictive modelling via leveraging regression techniques/algorithms
- Potential future improvements include the following:
  - This model can be improved upon by incorporating a bit more detail/features as described in the introductory section
  - With the addition of Bayesian principles, the model can serve as a foundation for creating a dynamic model that can learn and adapt as new information is digested
  - Eventually, one could fully automate the quoting process for buying used cars based on questionnaire and in-person validation of the questionnaire
    - Think of a program that requires user input (information about the car) and spits out how much a business should pay for the car in addition to how much they could get it they resold it (assuming X,Y and Z standard improvements based on % return on invested capital into renovations)
  - Create an additional model that aims at creating a more fruitful investment thesis; hence, do we go after smaller margin, higher moving models such as ford f150s or do we go after higher-end, slower-moving models that could take months to sell but lead to a higher margin per car sold
  - Potentially tie this in with Kelly blue book to further optimize a quoting program