PEACHES AND LEMONS

Predicting whether a vehicle will be 'kicked back' to the auction.

Will Morgan

Nov 2, 2018

MOTIVATION & BACKGROUND

(In American slang, a lemon is a car that is found to be defective only after it has been bought.)

The Lemons Problem

In Economics, the Lemons Problem refers to issues that arise regarding the value of a product due to asymmetric information between the buyer and the seller.

This information asymmetry leads a degradation in the quality of products in a given market.

The subject even has laws named after it.

PROPOSED FLOW

- Overview of the Problem
- Objective
- The Data
- Feature Engineering
- Modeling & Feature Selection
- Tuning and Evaluating the Model
- Conclusion

Overview of the Problem

From kaggle:

"One of the biggest challenges of an auto dealership purchasing a used car at an auto auction is the risk of that the vehicle might have serious issues that prevent it from being sold to customers. The auto community calls these unfortunate purchases "kicks".

Asymmetric information

Kicked cars often result when there are tampered odometers, mechanical issues the dealer is not able to address, issues with getting the vehicle title from the seller, or some other unforeseen problem. Kick cars can be very costly to dealers after transportation cost, throw-away repair work, and market losses in reselling the vehicle.

Preventing quality degredation

Modelers who can figure out which cars have a higher risk of being kick can provide real value to dealerships trying to provide the best inventory selection possible to their customers.

The challenge of this competition is to predict if the car purchased at the Auction is a Kick (bad buy)."

TL;DR:

Vehicles that are returned to an auction are called kicks (lemons), and they can become a huge cost for dealers.

OBJECTIVE

The objective of the Kaggle competition is to predict which cars will be lemons.

In addition to predicting lemons, the data can also be used in an attempt to maximize the expected profit margins of the inventory.

Click Me!

THE DATA:

The data comes from Carvana, contains a list of vehicles purchased across two auctions (and other sources)

I used the training data and split it to train my model.

```
In [32]:
df.shape
Out[32]:
(72983, 34)
```

The test data does not contain the outcome attribute.

```
Class Balance:
Number of Lemons: 8976
Number of Peaches: 64007
Occurance Rate: 12.3%
```

The data is imbalanced.

THE DATA

Original Attributes

```
In [57]: df.dtypes

Out[57]: RefId                                   int64
         IsBadBuy                                int64
         PurchDate                              object
         Auction                                object
         VehYear                                 int64
         VehicleAge                              int64
         Make                                   object
         Model                                  object
         Trim                                   object
         SubModel                               object
         Color                                  object
         Transmission                           object
         WheelTypeID                           float64
         WheelType                              object
         VehOdo                                  int64
         Nationality                            object
         Size                                   object
         TopThreeAmericanName                   object
         MMRAcquisitionAuctionAveragePrice     float64
         MMRAcquisitionAuctionCleanPrice       float64
         MMRAcquisitionRetailAveragePrice      float64
         MMRAcquisitonRetailCleanPrice         float64
         MMRCurrentAuctionAveragePrice         float64
         MMRCurrentAuctionCleanPrice           float64
         MMRCurrentRetailAveragePrice          float64
         MMRCurrentRetailCleanPrice            float64
         PRIMEUNIT                              object
         AUCGUART                               object
         BYRNO                                   int64
         VNZIP1                                  int64
         VNST                                   object
         VehBCost                              float64
         IsOnlineSale                            int64
         WarrantyCost                            int64
         dtype: object
```

Attributes Dropped (a prior i)

['RefId', 'BYRNO', 'AUCGUART', 'PRIMEUNIT','VNZIP','WheelType', 'VehYear']

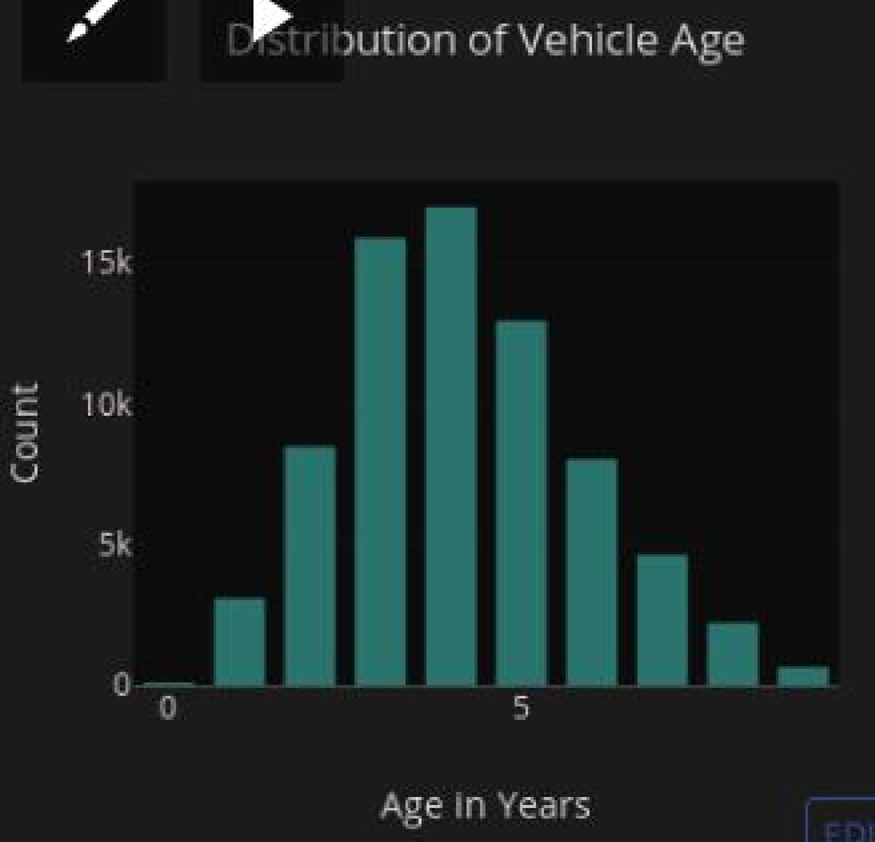
Continuous Attributes

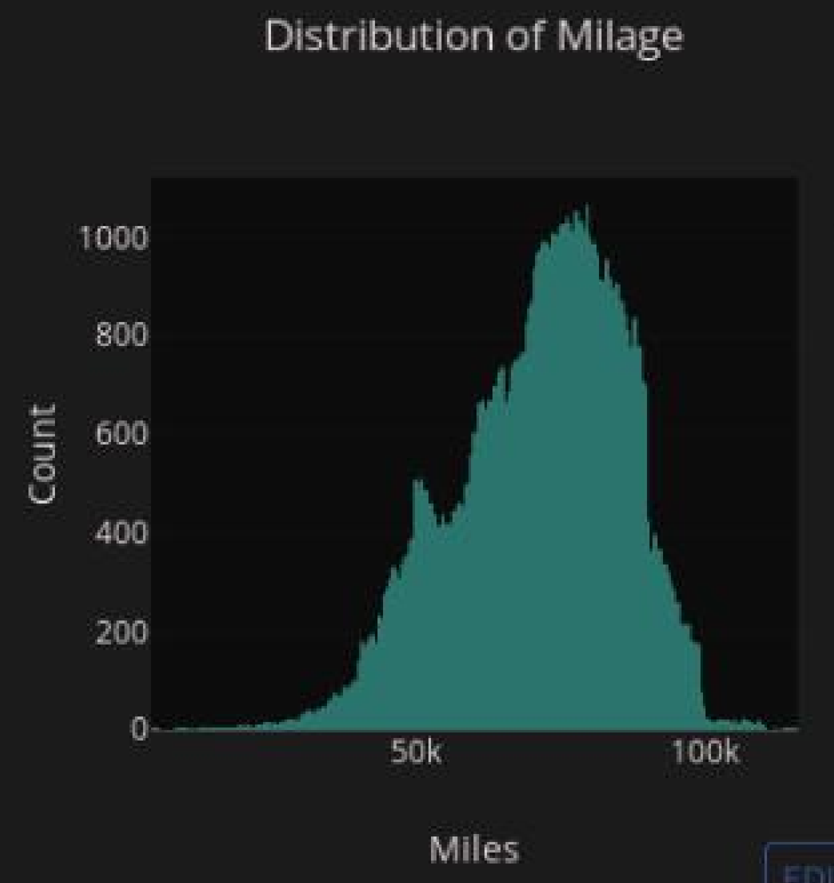
```
Index(['VehYear', 'VehicleAge',  'VehOdo', 'VehBCost',
       'MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice',
       'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitonRetailCleanPrice',
       'MMRCurrentAuctionAveragePrice', 'MMRCurrentAuctionCleanPrice',
       'MMRCurrentRetailAveragePrice', 'MMRCurrentRetailCleanPrice',
        'WarrantyCost']
```

Categorical/Binary Attributes

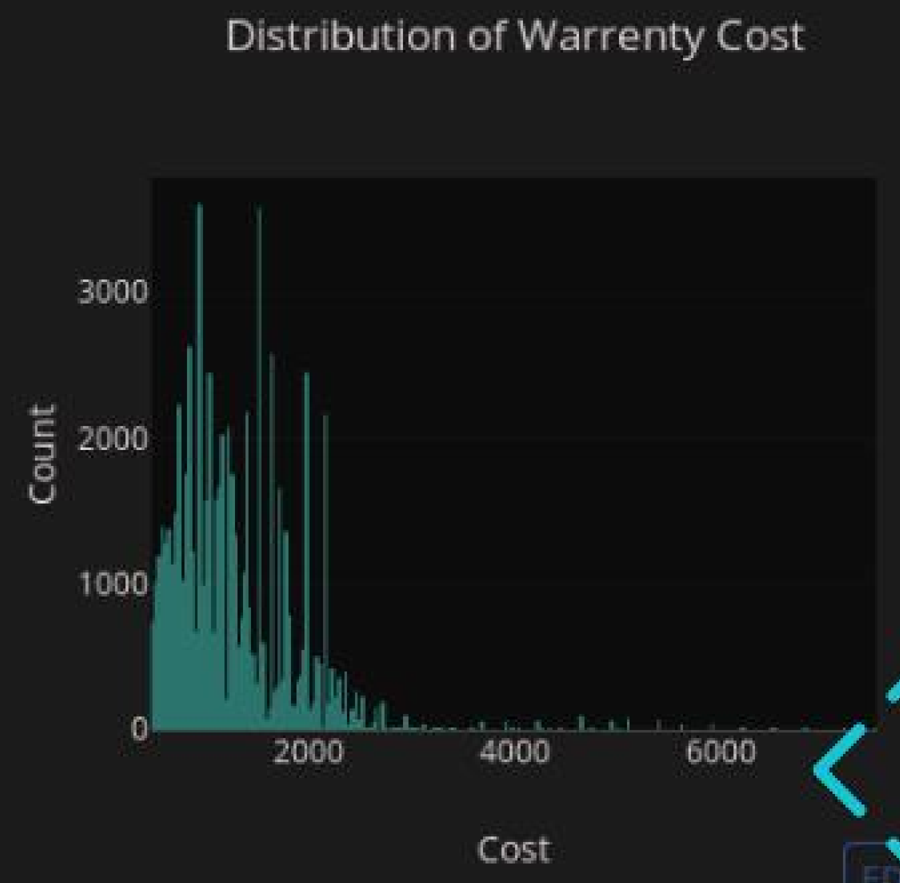
```
Index(['Auction','PurchDate 'Make', 'Model', 'Trim', 'SubModel', 'Color',
       'Transmission','VNST', 'WheelTypeID', 'Nationality','Size',
       'TopThreeAmericanName', 'IsOnlineSale']
```

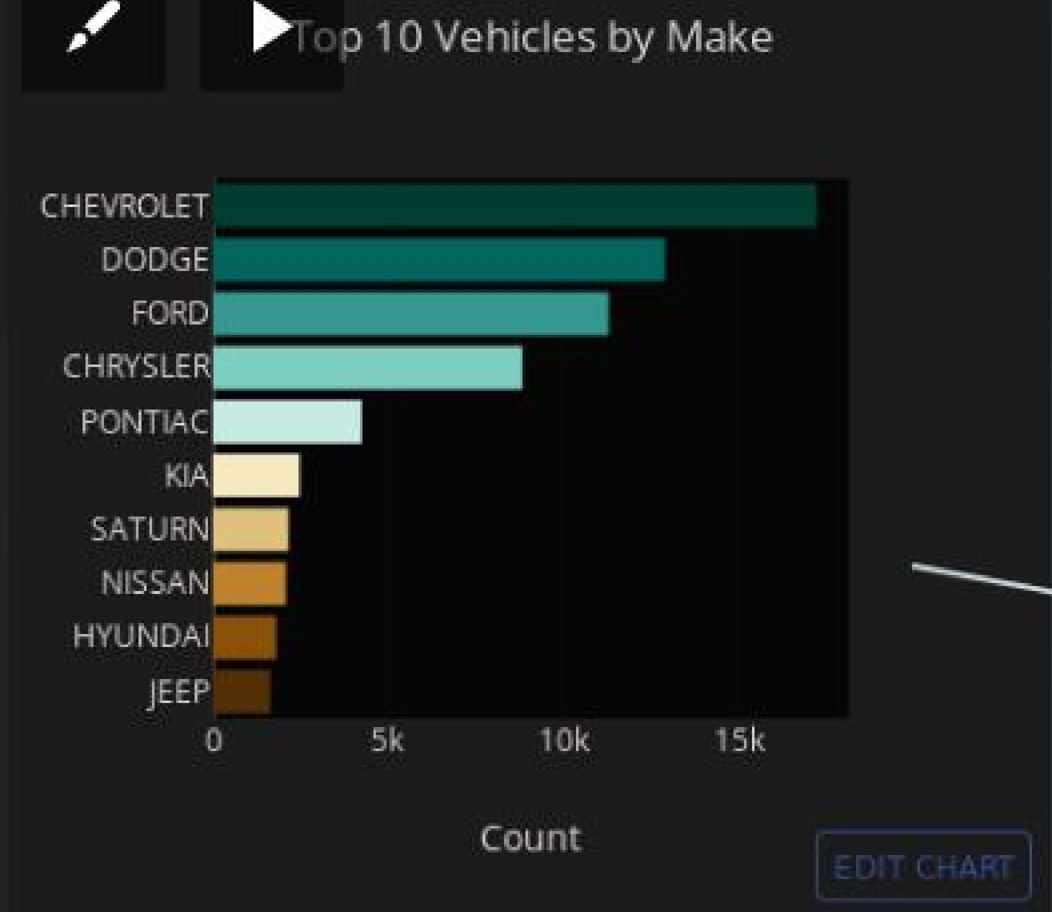
THE DATA: Continuous Attributes

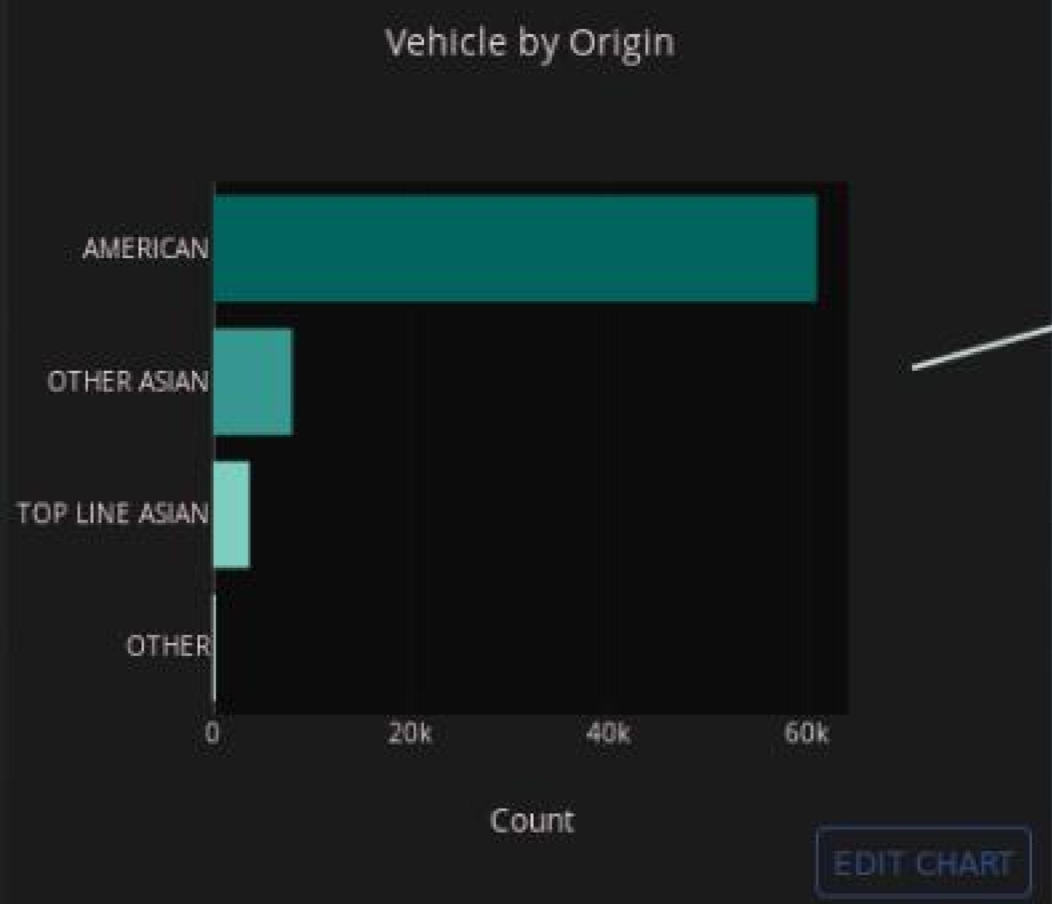




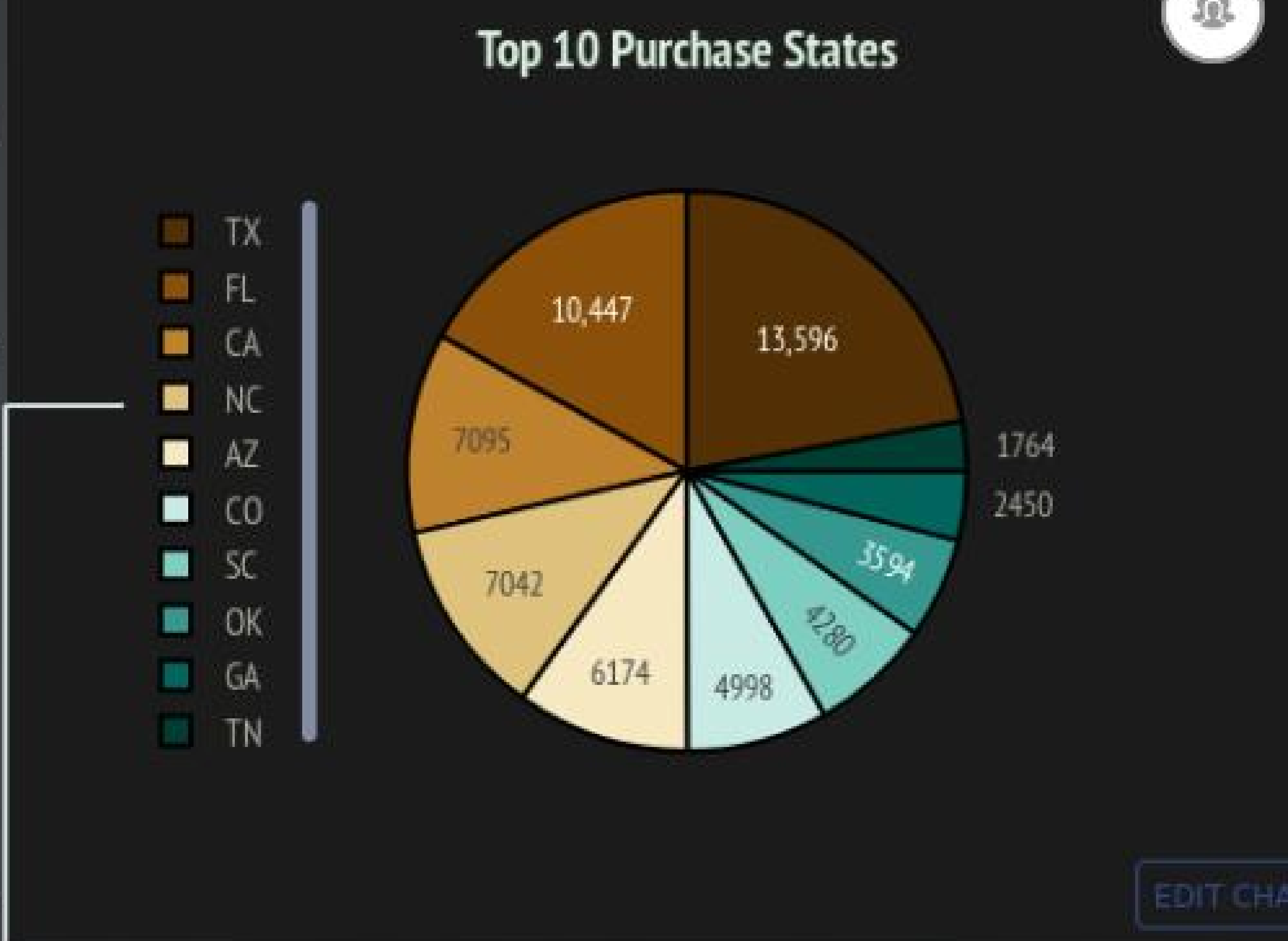

```
'VehicleAge' - Age of Vehicle in Years
'VehOdo' - Vehicle Odometer Reading
'VehBCost - Price paid at the time of acquisition
'MMRAcquisitionAuctionAveragePrice' - Acquisition Average Auction Price
'MMRAcquisitionAuctionCleanPrice' - Acquisition Good Condition Auction Price
'MMRAcquisitionRetailAveragePrice' - Acquisition Average Retail Price
'MMRAcquisitonRetailCleanPrice' - Acquisition Good Condition Auction Price
'MMRCurrentAuctionAveragePrice' - Current Average Auction Price
'MMRCurrentAuctionCleanPrice' - Current Good Condition Auction Price
'MMRCurrentRetailAveragePrice' - Current Average Retail Price
'MMRCurrentRetailCleanPrice' - Current Retail Good Condition Price
'WarrantyCost' - Cost of the warranty (36k mi, 36 mo)
```







THE DATA: Categorical Attributes



'**Auction**' - Vehicle Source

'**PurchDate**' - Purchase Date

'**Make**' - Make

'**Model**'* - Model Type

'**Trim**'* - Style of the Vehicle

'**SubModel**'* - Additional specifications

'**Color**' - Color

'**Transmission**' - Transmission Type

'**WheelTypeID**' - Wheel Type

'**Nationality**' - Manufacturing Nation

'**Size**' - Size

'**TopThreeAmericanName**' - GM, Ford, Chrysler, other

'**VNST**' - State where Vehivle was Purchased

'**IsOnlineSale**' - The vehicle was purchased online (binary)

'IsBadBuy' (outcome) - The vehicle is a lemon (binary)

```
for col in cats:
    print(col,len(df[col].unique()))
```

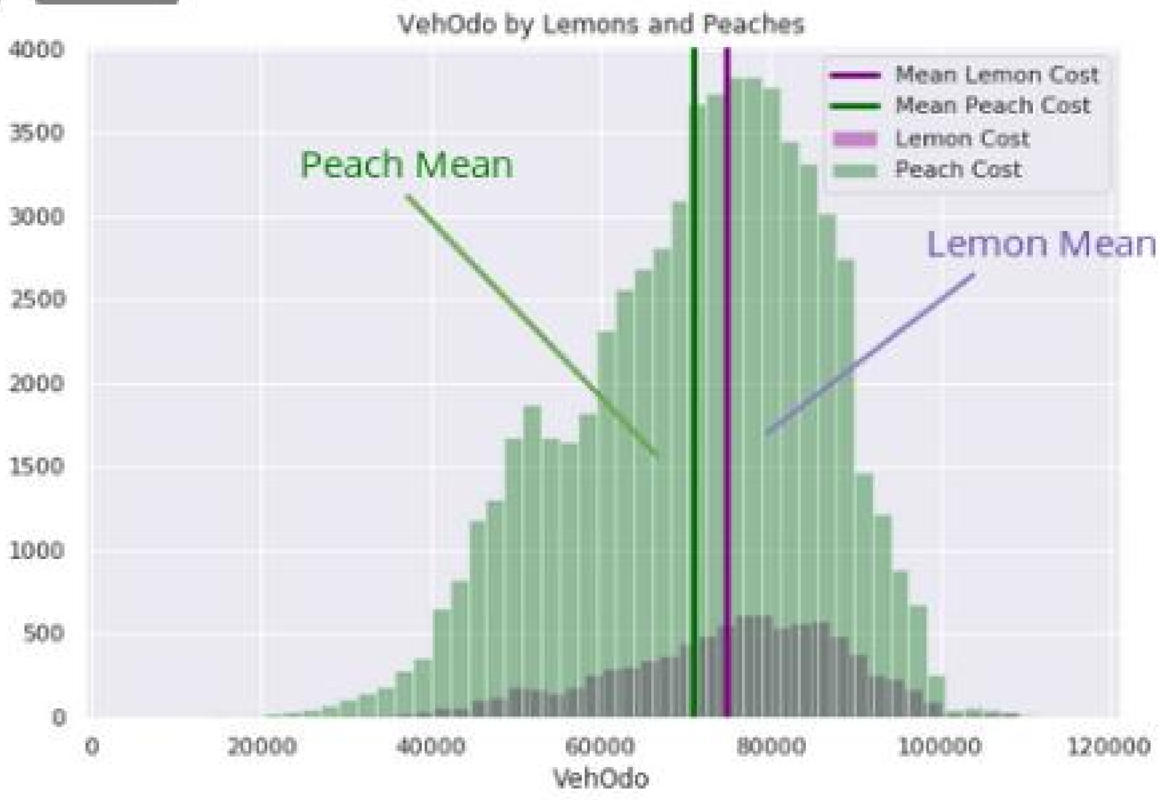
```
Auction 3
Make 33
Model 1063
Trim 135
SubModel 864
Color 17
Transmission 4
WheelTypeID 5
Nationality 5
Size 13
TopThreeAmericanName 5
VNST 37
IsOnlineSale 2
IsBadBuy 2
```

```
df.Color.unique()
```

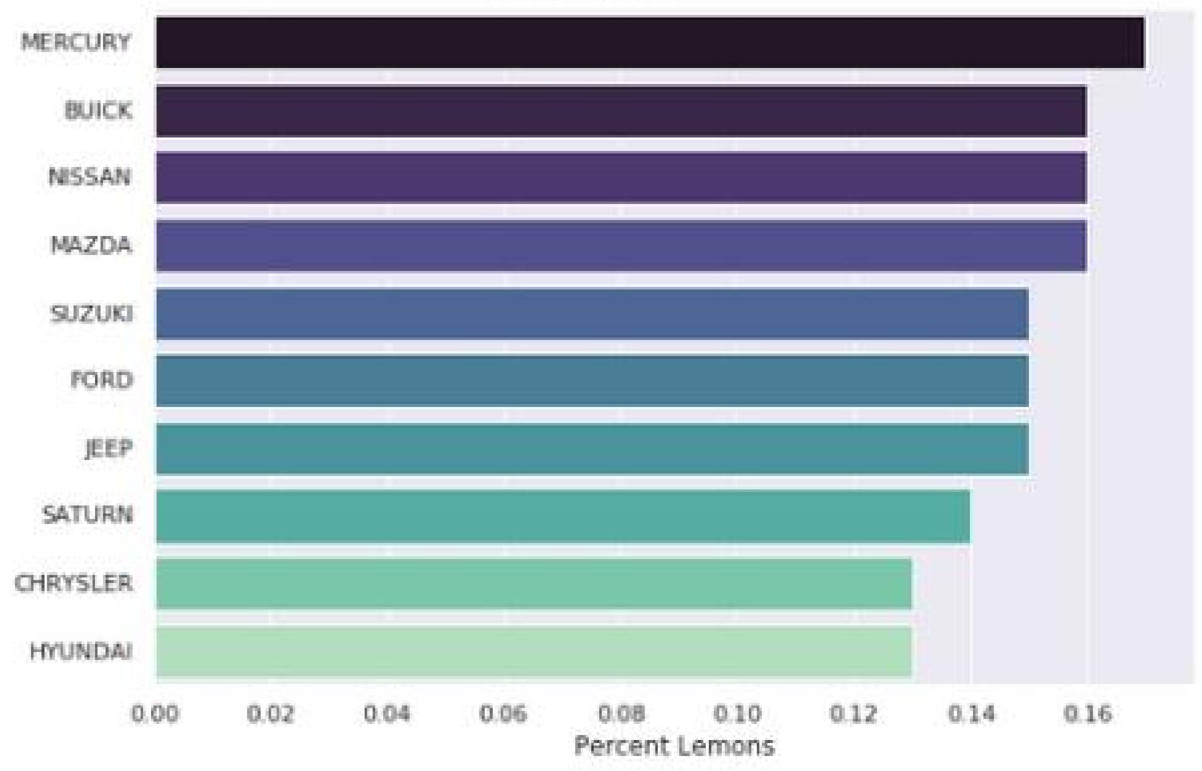
```
array(['RED', 'WHITE', 'MAROON', 'SILVER', 'BLACK', 'GOLD', 'GREY',
       'BLUE', 'BEIGE', 'PURPLE', 'ORANGE', 'GREEN', 'BROWN', 'YELLOW',
       'NOT AVAIL', 'OTHER'], dtype=object)
```

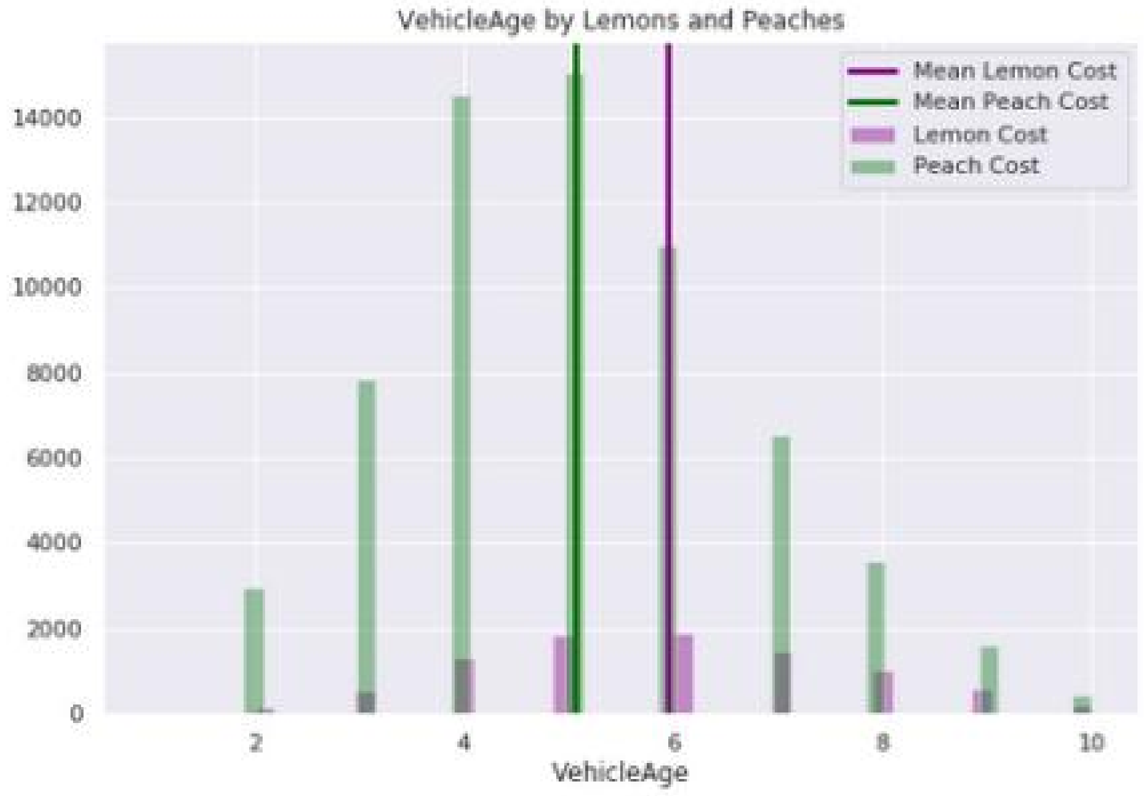
```
IsOnlineSale:
Number of Vehicles Purcahsed Online: 1845
Number of Vehicles Purchased not Online: 71138
Occurance Rate: 2.53%
```

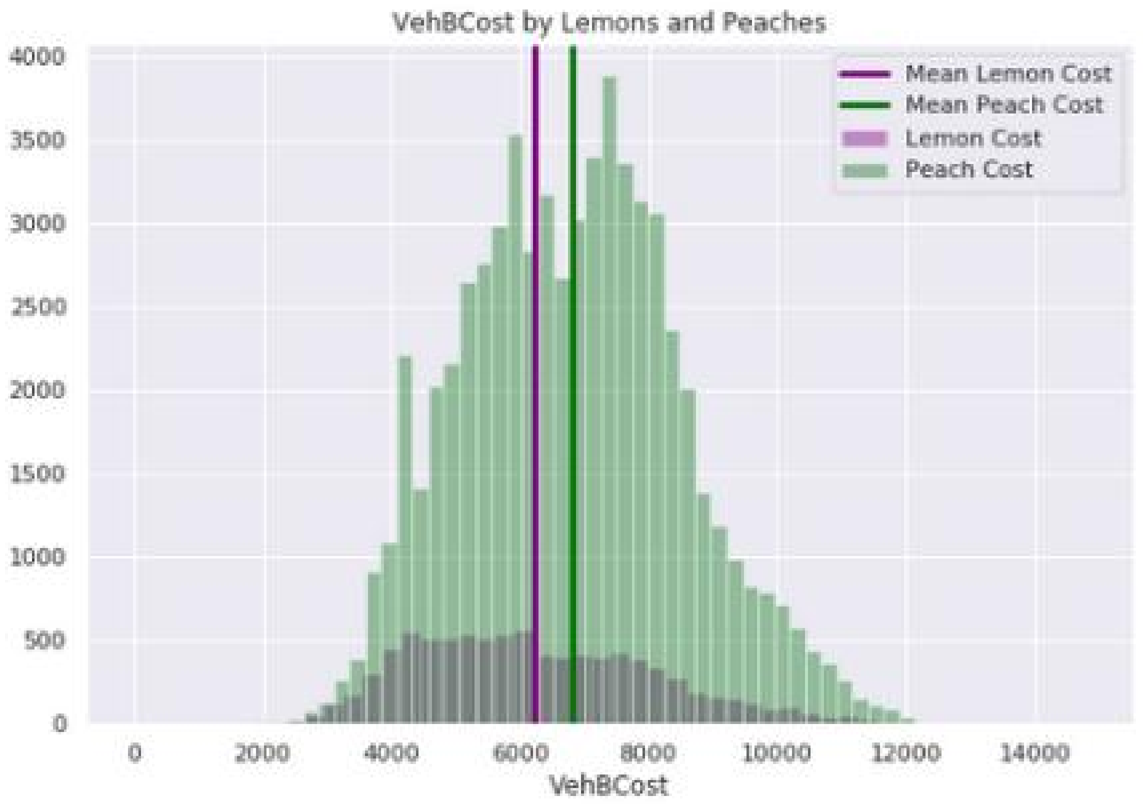
*Category is Problematic


VehOdo by Lemons and Peaches
Mean Lemon Cost
Mean Peach Cost
Lemon Cost
Peach Cost
Peach Mean
Lemon Mean
VehOdo

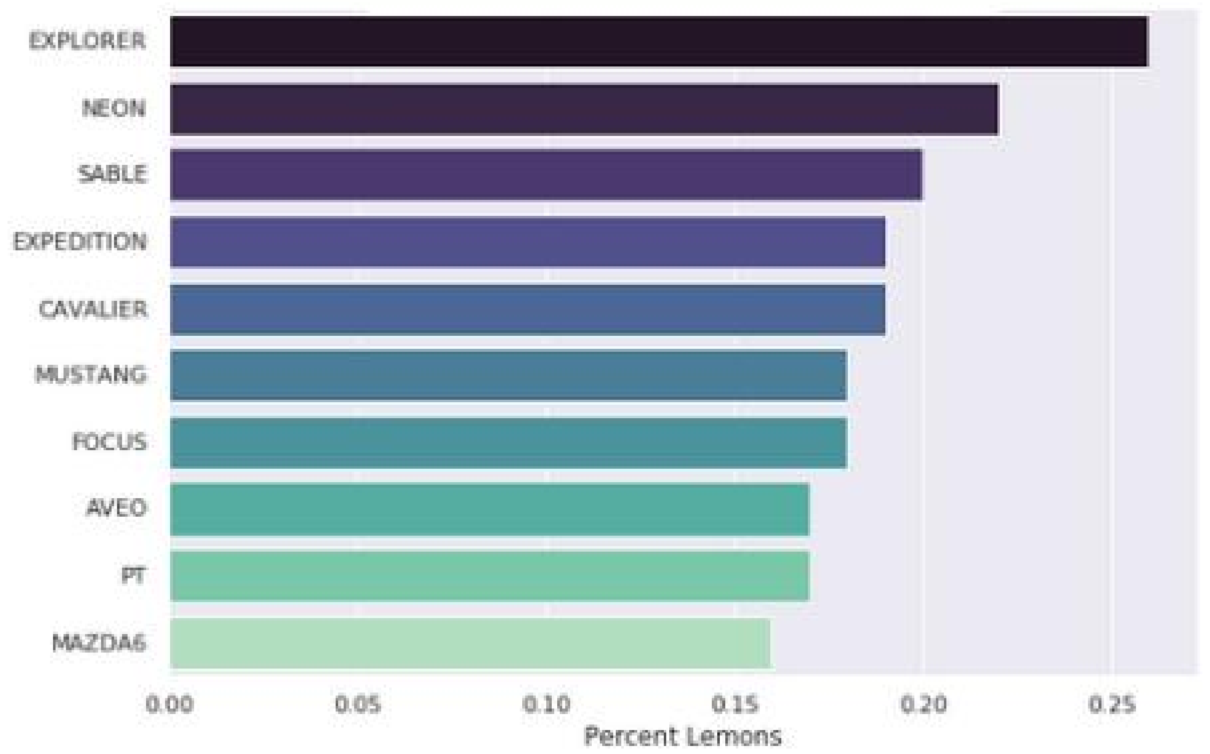
MAKE

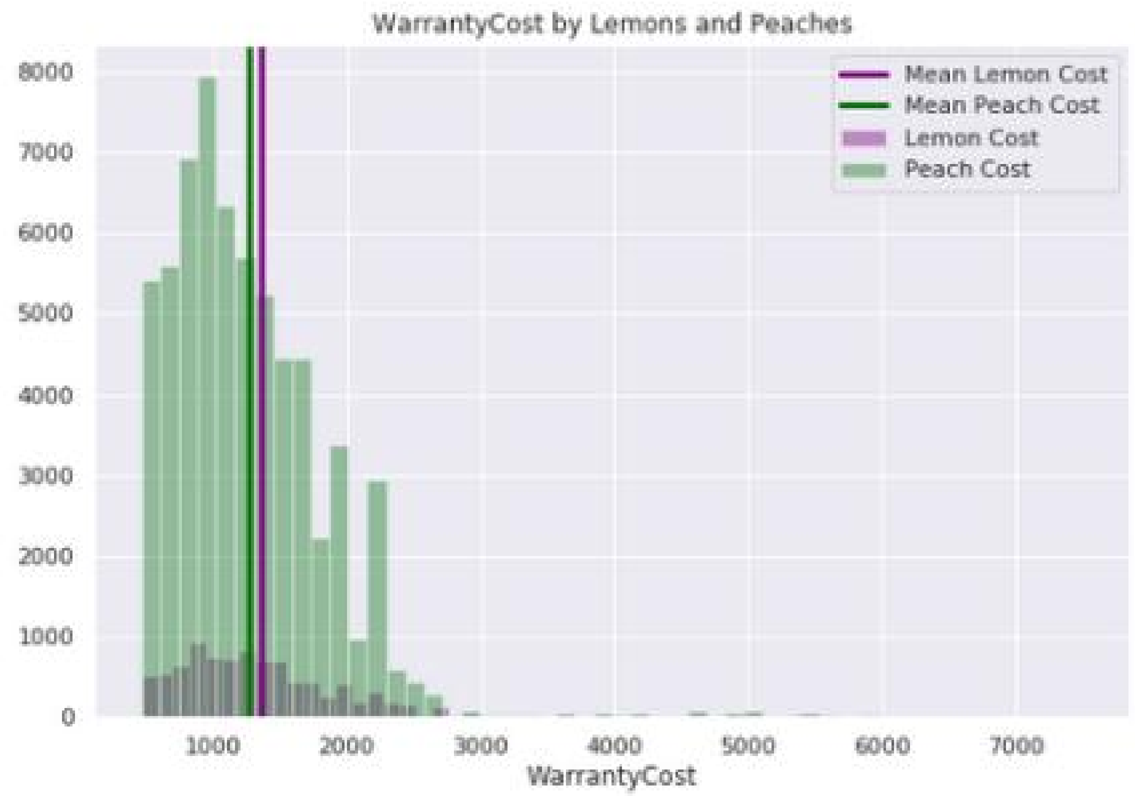

MERCURY
BUICK
NISSAN
MAZDA
SUZUKI
FORD
JEEP
SATURN
CHRYSLER
HYUNDAI
Percent Lemons


VehicleAge by Lemons and Peaches
Mean Lemon Cost
Mean Peach Cost
Lemon Cost
Peach Cost
VehicleAge


VehBCost by Lemons and Peaches
Mean Lemon Cost
Mean Peach Cost
Lemon Cost
Peach Cost
VehBCost

MODEL


EXPLORER
NEON
SABLE
EXPEDITION
CAVALIER
MUSTANG
FOCUS
AVEO
PT
MAZDA6
Percent Lemons


WarrantyCost by Lemons and Peaches
Mean Lemon Cost
Mean Peach Cost
Lemon Cost
Peach Cost
WarrantyCost

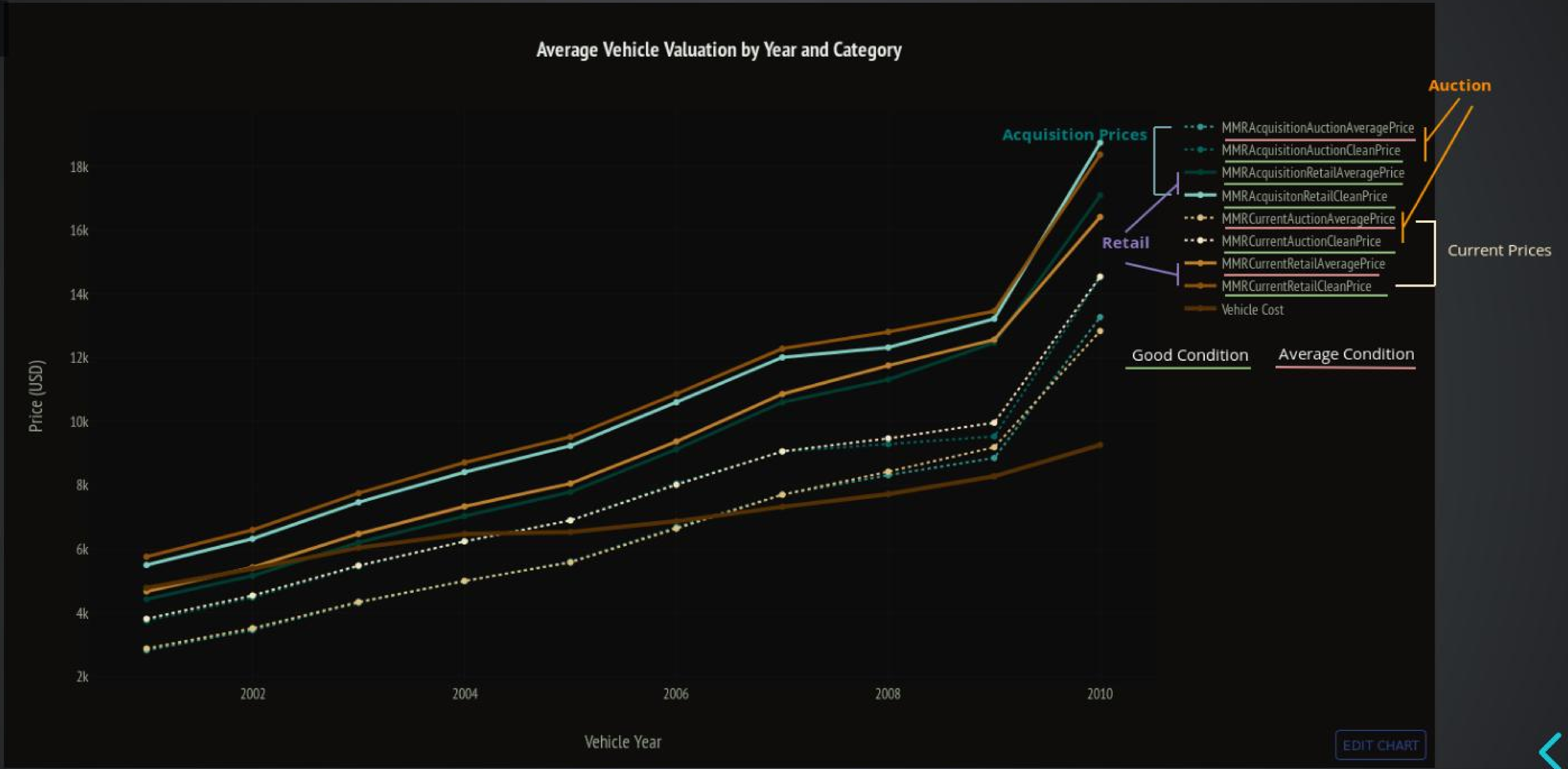
Market Rate Attributes

CurrentRetailCleanPrice - Current Retail Good Condition
AcquisitonRetailCleanPrice - Acquisition Good Condition

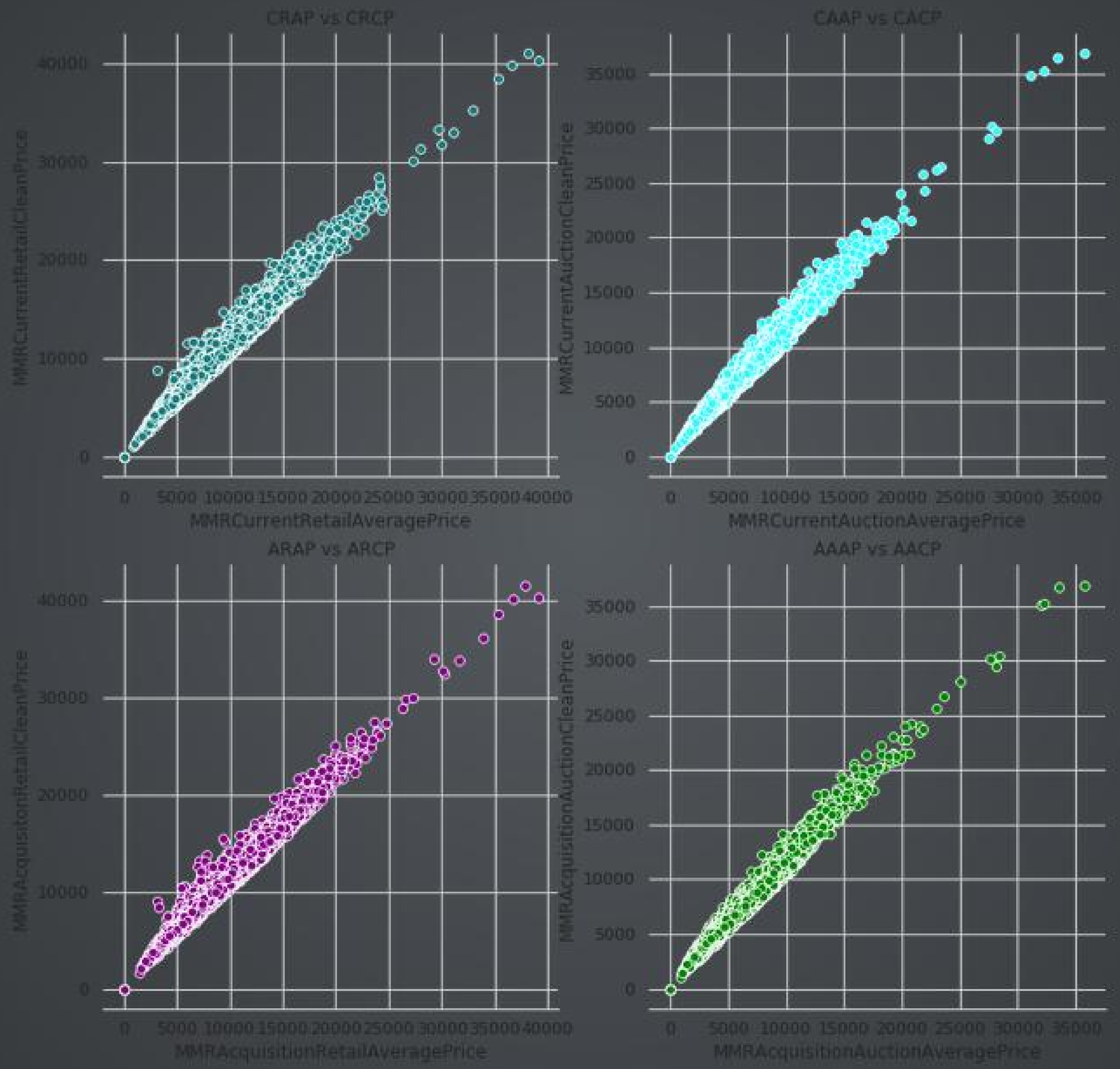
CurrentRetailAveragePrice - Current Average Retail
AcquisitionRetailAveragePrice - Acquisition Average Retail

CurrentAuctionCleanPrice - Current Good Condition Auction
AcquisitionAuctionCleanPrice - Acquisition Good Condition Auction

CurrentAuctionAveragePrice - Current Average Auction Price
AcquisitionAuctionAveragePrice - Acquisition Average Auction


Average Vehicle Valuation by Year and Category
Auction
Acquisition Prices
MMRAcquisitionAuctionAveragePrice
MMRAcquisitionAuctionCleanPrice
MMRAcquisitionRetailAveragePrice
MMRAcquisitonRetailCleanPrice
MMRCurrentAuctionAveragePrice
MMRCurrentAuctionCleanPrice
MMRCurrentRetailAveragePrice
MMRCurrentRetailCleanPrice
Vehicle Cost
Retail
Current Prices
Good Condition
Average Condition
Price (USD)
18k
16k
14k
12k
10k
8k
6k
4k
2k
2002
2004
2006
2008
2010
Vehicle Year
EDIT CHART

Correlation Between Prices



Expected Margin

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

'Average Price'

Average Price - Vehicle Cost = Expected Profit Margin


Expected Model Profitability by Make
Expected Margin
2500
2000
1500
1000
500
0
−500
TSX
TL
MDX
RSX
1500
AVALANCHE
1500HD
2500
RONDO
AMANTI
RIO
OPTIMA
AVALON
4
CAMRY
COROLLA
Acura
Chrysler
Kia
Toyota
Top 4 Models by Make
EDIT CHART

FEATURE ENGINEERING

Dealing with the categorical features

```
In [13]: characteristics = df[['Model','SubModel','Trim']]

In [14]: characteristics.head(10)
Out[14]:
```

	Model	SubModel	Trim
0	MAZDA3	4D SEDAN I	i
1	1500 RAM PICKUP 2WD	QUAD CAB 4.7L SLT	ST
2	STRATUS V6	4D SEDAN SXT FFV	SXT
3	NEON	4D SEDAN	SXT
4	FOCUS	2D COUPE ZX3	ZX3
5	GALANT 4C	4D SEDAN ES	ES
6	SPECTRA	4D SEDAN EX	EX
7	TAURUS	4D SEDAN SE	SE
8	SPECTRA	4D SEDAN EX	EX
9	FIVE HUNDRED	4D SEDAN SEL	SEL

```
df.PurchDate = pd.to_datetime(df.PurchDate)
```

pyear	pmonth	pday
2009	12	7
2009	12	7
2009	12	7
2009	12	7
2009	12	7

```
for col in cats:
    print(col,len(df[col].unique()))
```

```
Auction 3
Make 33
Model 1063
Trim 135
SubModel 864
Color 17
Transmission 4
WheelTypeID 5
Nationality 5
Size 13
TopThreeAmericanName 5
VNST 37
IsOnlineSale 2
IsBadBuy 2
```

FEATURE ENGINEERING

Dealing with the categorical features

'Model'

Before

```
In [16]: characteristics.Model.head(25)
Out[16]:
0                MAZDA3
1      1500 RAM PICKUP 2WD
2             STRATUS V6
3                   NEON
4                  FOCUS
5              GALANT 4C
6                SPECTRA
7                 TAURUS
8                SPECTRA
9           FIVE HUNDRED
10    1500 SIERRA PICKUP 2
11     F150 PICKUP 2WD V6
12    CARAVAN GRAND FWD V6
13                ALTIMA
14    CARAVAN GRAND FWD V6
15           CAVALIER 4C
16     TRAILBLAZER 2WD 6C
17            VUE 2WD 4C
18                IMPALA
19        ENVOY XL 2WD 6C
20       VOYAGER FWD V6
21           MONTE CARLO
22       VENTURE FWD V6
23                   HHR
24                   HHR
Name: Model, dtype: object
```

After

```
In [18]: characteristics.Model.str.split().str[0].str.strip().head(25)
Out[18]:
0          MAZDA3
1            1500
2         STRATUS
3            NEON
4           FOCUS
5          GALANT
6         SPECTRA
7          TAURUS
8         SPECTRA
9            FIVE
10           1500
11           F150
12        CARAVAN
13         ALTIMA
14        CARAVAN
15       CAVALIER
16    TRAILBLAZER
17            VUE
18         IMPALA
19          ENVOY
20        VOYAGER
21          MONTE
22        VENTURE
23            HHR
24            HHR
Name: Model, dtype: object
```

This is a quick way to remove the additional information after the model name. However, some mix ups will occur. Such as makers having similar models like the Chevy and Dodge.The Sierra and Ram are both identified as 1500 and 'Monte Carlo' is truncated.

'Model' and 'SubModel'

Identify distinguishing features

```
In [13]: characteristics = df[['Model','SubModel','Trim']]

In [14]: characteristics.head(10)
Out[14]:
                 Model             SubModel Trim
0              MAZDA3              4D SEDAN    i
1   1500 RAM PICKUP 2WD  QUAD CAB 4.7L SLT   ST
2          STRATUS V6      4D SEDAN SXT FFV  SXT
3                NEON             4D SEDAN  SXT
4               FOCUS        2D COUPE ZX3  ZX3
5           GALANT 4C          4D SEDAN ES   ES
6             SPECTRA          4D SEDAN EX   EX
7              TAURUS          4D SEDAN SE   SE
8             SPECTRA          4D SEDAN EX   EX
9        FIVE HUNDRED         4D SEDAN SEL  SEL
```

```
mod_chars = ['2WD','V6','4C','PICKUP','6C','FWD',
             '4WD','AWD','SFI','EFI','DOHC','I4','MPI']
```

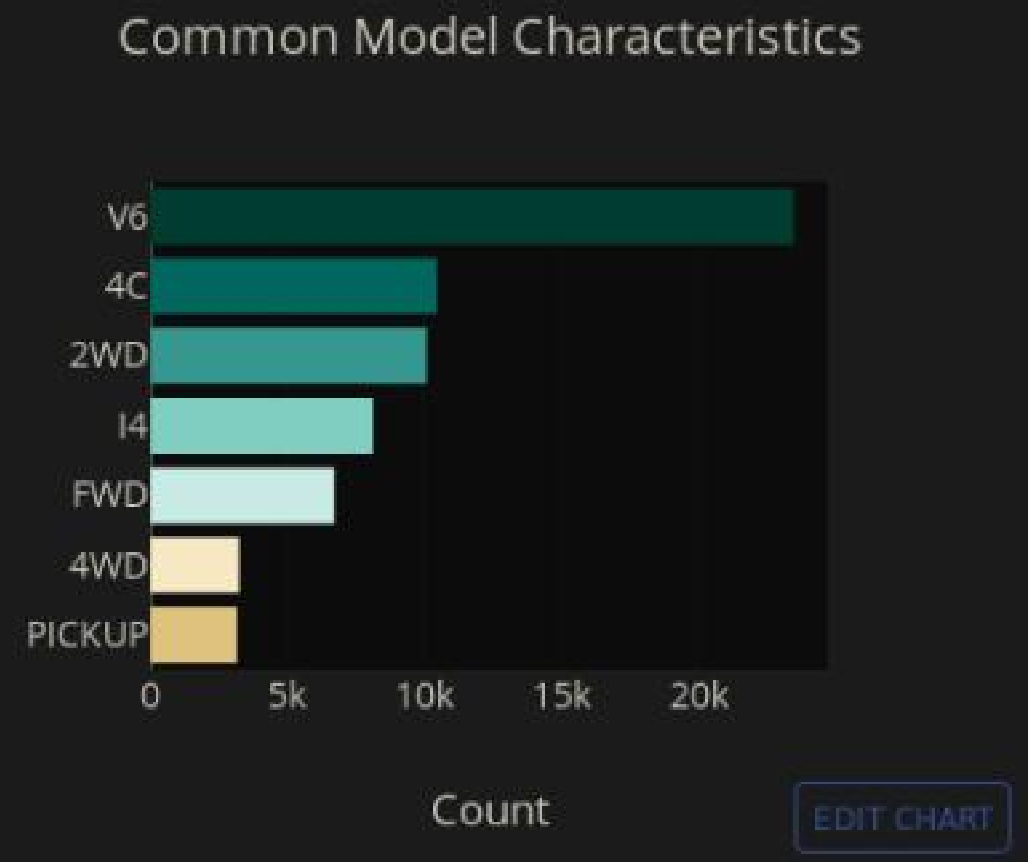
```
submod_chars = ['SEDAN','4D','EXT','CAB','2D','CAB','WAGON',
                'REG','FFV','PASSENGER','SUV','SPORT',
                'UTILITY','QUAD','COUPE','MINIVAN','CUV']
```

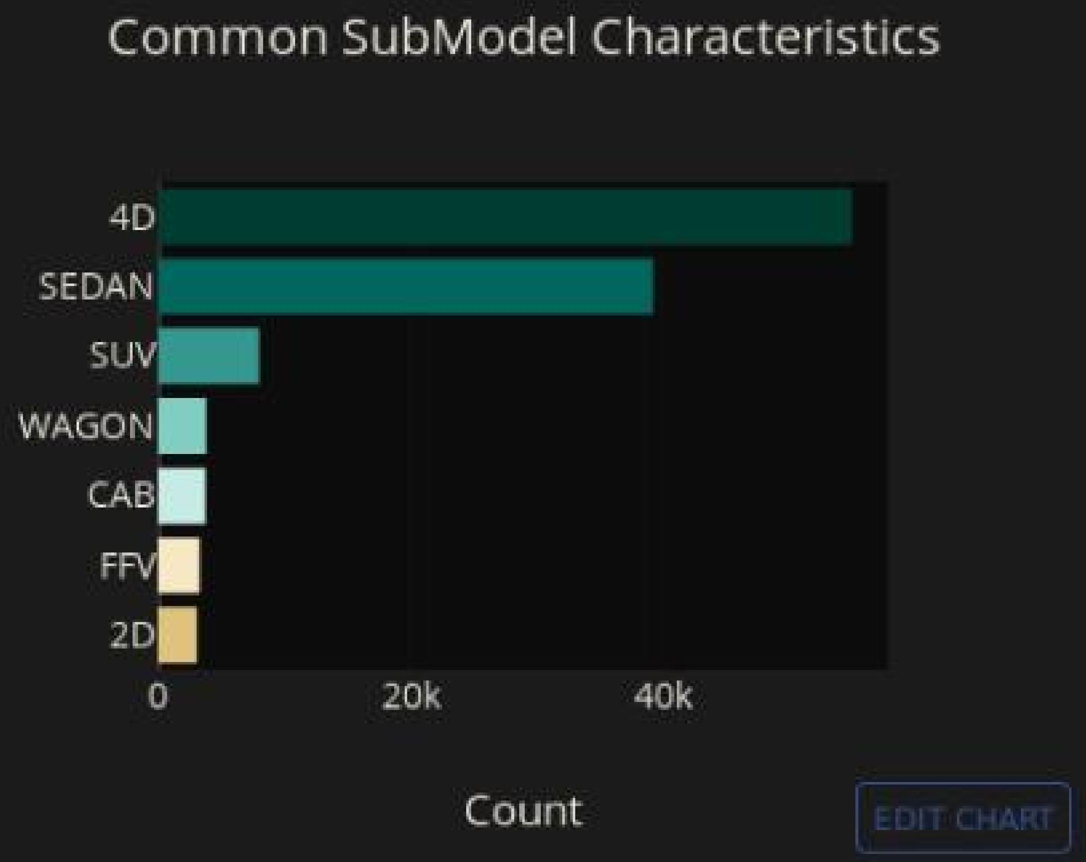
```
In [29]: mod_val_sets.head(10)
Out[29]:
   2WD  V6  4C  PICKUP  6C  FWD  4WD
0    0   0   0       0   0    0    0
1    1   0   0       1   0    0    0
2    0   1   0       0   0    0    0
3    0   0   0       0   0    0    0
4    0   0   0       0   0    0    0
5    0   0   1       0   0    0    0
6    0   0   0       0   0    0    0
7    0   0   0       0   0    0    0
8    0   0   0       0   0    0    0
9    0   0   0       0   0    0    0
```

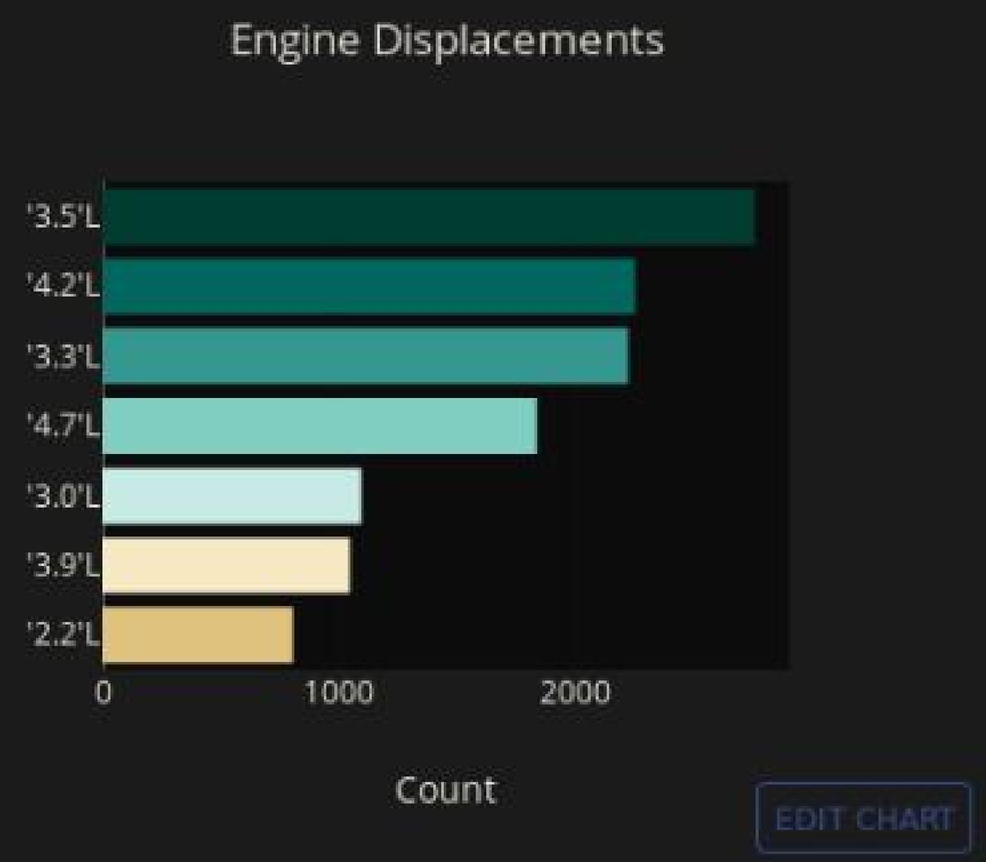
```
In [27]: sub_val_sets.head(10)
Out[27]:
   SEDAN  4D  EXT  CAB  2D  WAGON
0      1   1    0    0   0      0
1      0   0    0    1   0      0
2      1   1    0    0   0      0
3      1   1    0    0   0      0
4      0   0    0    0   1      0
5      1   1    0    0   0      0
6      1   1    0    0   0      0
7      1   1    0    0   0      0
8      1   1    0    0   0      0
9      1   1    0    0   0      0
```

```
In [42]: displacements[:10]
Out[42]: ['4.7', '4.3', '4.2', '3.3', '3.8', '4.2', '4.2', '3.3', '3.4', '2.2']
```

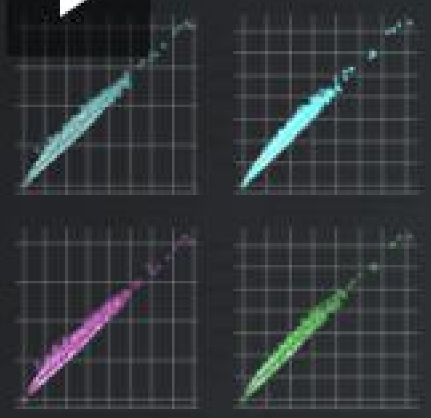
```
Total Number of Model Characteristic matches: 75219
Total Number of SubModel Charatistic matches: 134395
Total number of Displacement matches: 20247
```







FEATURE ENGINEERING



```
plodf= df[['MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice',
 'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitonRetailCleanPrice',
 'MMRCurrentAuctionAveragePrice', 'MMRCurrentAuctionCleanPrice',
 'MMRCurrentRetailAveragePrice', 'MMRCurrentRetailCleanPrice',
 'VehicleAge','VehOdo','WarrantyCost','IsBadBuy']].copy()
plodf.dropna(inplace=True)
```

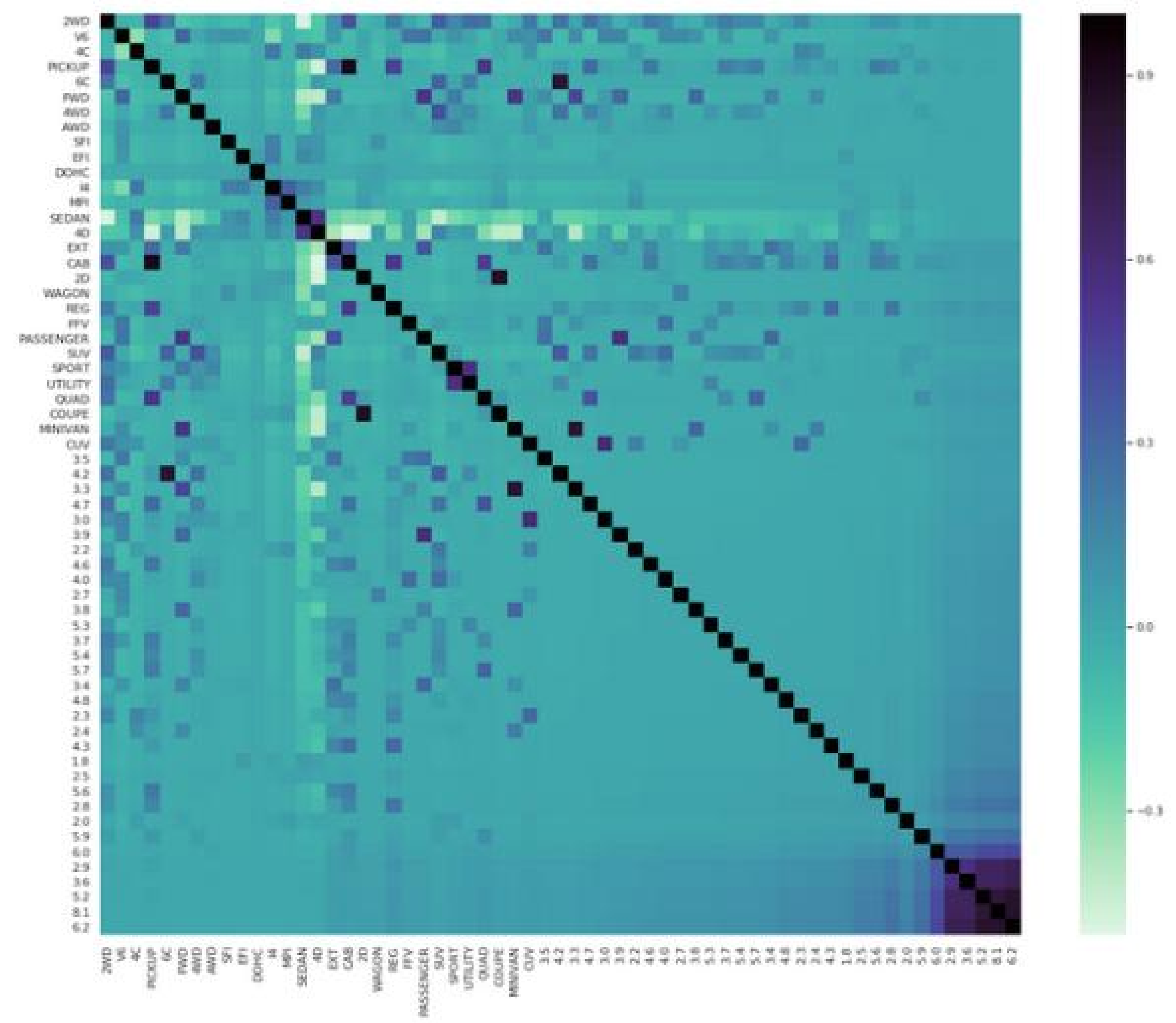

```
In [62]: list(plodf.columns[:20])
Out[62]:
['MMRAcquisitionAuctionAveragePrice',
 'MMRAcquisitionAuctionCleanPrice',
 'MMRAcquisitionRetailAveragePrice',
 'MMRAcquisitonRetailCleanPrice',
 'MMRCurrentAuctionAveragePrice',
 'MMRCurrentAuctionCleanPrice',
 'MMRCurrentRetailAveragePrice',
 'MMRCurrentRetailCleanPrice',
 'VehicleAge',
 'VehOdo',
 'WarrantyCost',
 'IsBadBuy',
 'crcp_cacp',
 'arcp_aacp',
 'crap_caap',
 'arap_aaap',
 'caap_aaap',
 'cacp_aacp',
 'crap_arap',
 'crcp_arcp']
```

```
In [63]: plodf.shape
Out[63]: (67275, 46)
```

```
In [70]: price_pvals.sort_values(by='pvalues')
Out[70]:
```

	price	pvalues
8	VehicleAge	0.000000e+00
11	IsBadBuy	0.000000e+00
4	MMRCurrentAuctionAveragePrice	3.613714e-177
0	MMRAcquisitionAuctionAveragePrice	8.336208e-177
5	MMRCurrentAuctionCleanPrice	2.236708e-157
1	MMRAcquisitionAuctionCleanPrice	9.032533e-155
6	MMRCurrentRetailAveragePrice	2.946974e-146
7	MMRCurrentRetailCleanPrice	5.401022e-134
2	MMRAcquisitionRetailAveragePrice	2.777969e-114
9	VehOdo	1.225459e-111
3	MMRAcquisitonRetailCleanPrice	5.123667e-104
10	WarrantyCost	9.440349e-41
39	crap_caap_arap_aaap	3.615944e-19
24	crcp_caap	1.769231e-17
38	crcp_cacp_arcp_aacp	1.293532e-16
40	crcp_caap_arcp_aaap	1.706332e-16
44	crcp_aacp_arcp_cacp	1.706332e-16
41	crap_cacp_arap_aacp	2.844209e-16

Very low p-values on t-tests for prices between lemons and non lemons



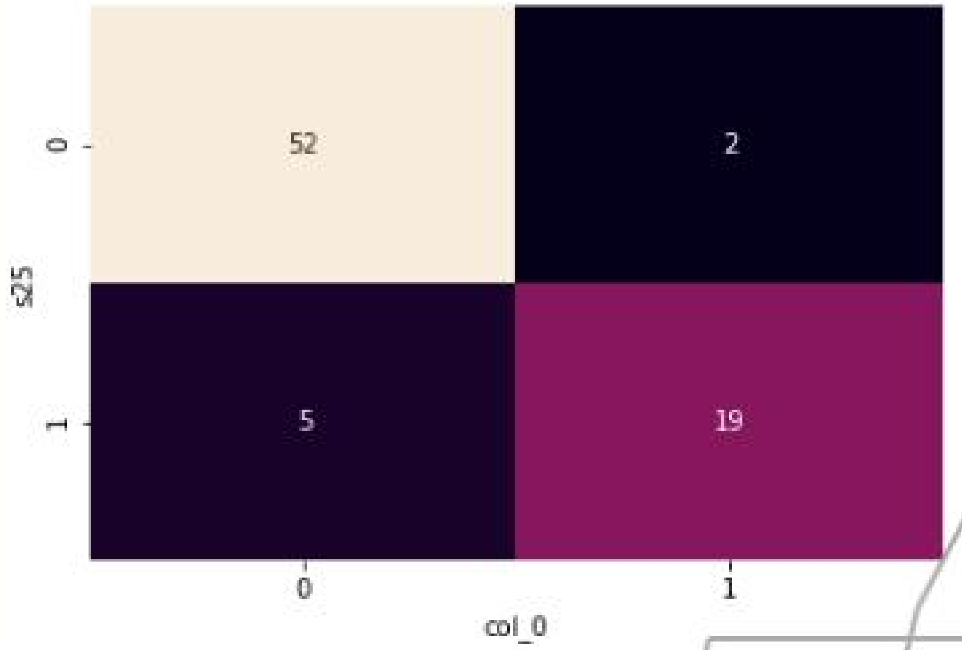
	level_0	level_1	0
199	PICKUP	CAB	0.911014
979	CAB	PICKUP	0.911014
1063	2D	COUPE	0.874932
1603	COUPE	2D	0.874932
3598	8.1	6.2	0.843254
3718	6.2	8.1	0.843254
3659	5.2	6.2	0.843254
3719	6.2	5.2	0.843254
274	6C	4.2	0.828402
1834	4.2	6C	0.828402
1678	MINIVAN	3.3	0.826839
1918	3.3	MINIVAN	0.826839
3597	8.1	5.2	0.799973
3657	5.2	8.1	0.799973

I dropped a few of the newly created features based on their correlation after reviewing how frequently each feature appeared in the data. e.g. 'CAB' has 3,916 occurrences while 'PICKUP' has 3,280.

Metrics for Evaluating the Model

Confusion Matrix

```
ROC_AUC score: 0.8773148148148148
 0,0 : Correctly Classified Negatives
 0,1 : False Positives (Type I Error)
 1,0 : False Negatives (Type II Error)
 1,1 : Correctly Classified Positives
```




```
Classification Report :
             precision    recall  f1-score   support

          0       0.91      0.96      0.94        54
          1       0.90      0.79      0.84        24

avg / total       0.91      0.91      0.91        78
```

What percent of your predictions were correct?

You answer: the "accuracy" was (52+19) out of 78 = 91%

What percent of the positive cases did you catch?

You answer: the "recall" was 19 out of 24 = 79%

What percent of positive predictions were correct?

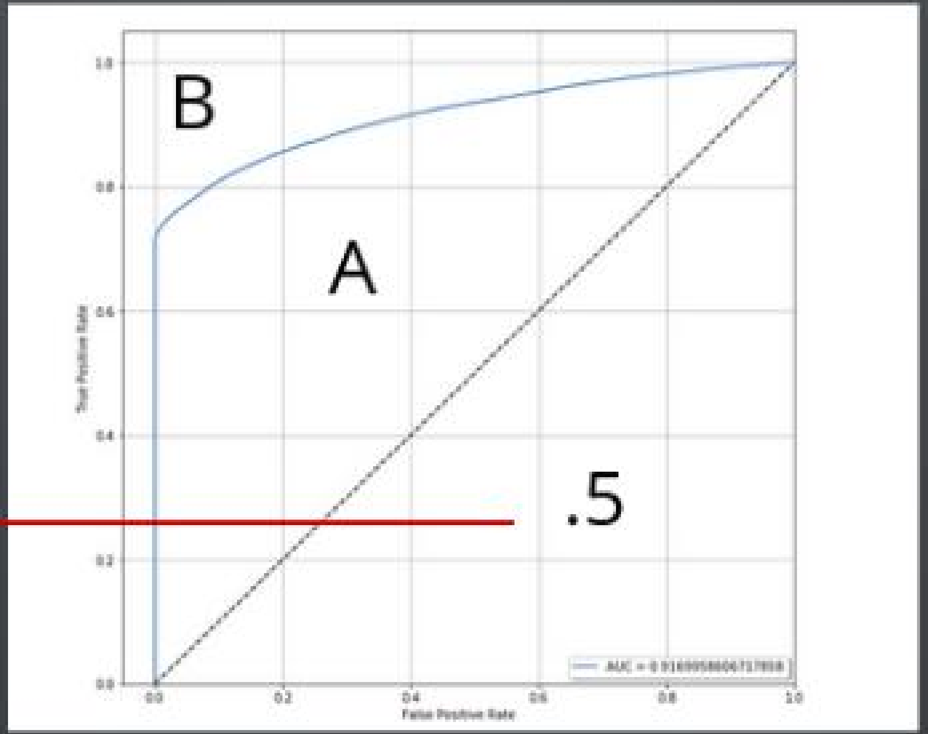
You answer: the "precision" was 19 out of 21 = 90%

$$A = \boxed{.87} - \boxed{.5} = .37$$

$$B = .5 - A = .13$$

$$G = .37/.5 \sim \mathbf{.74}$$

ROC & AUC



Area under the curve (AUC)

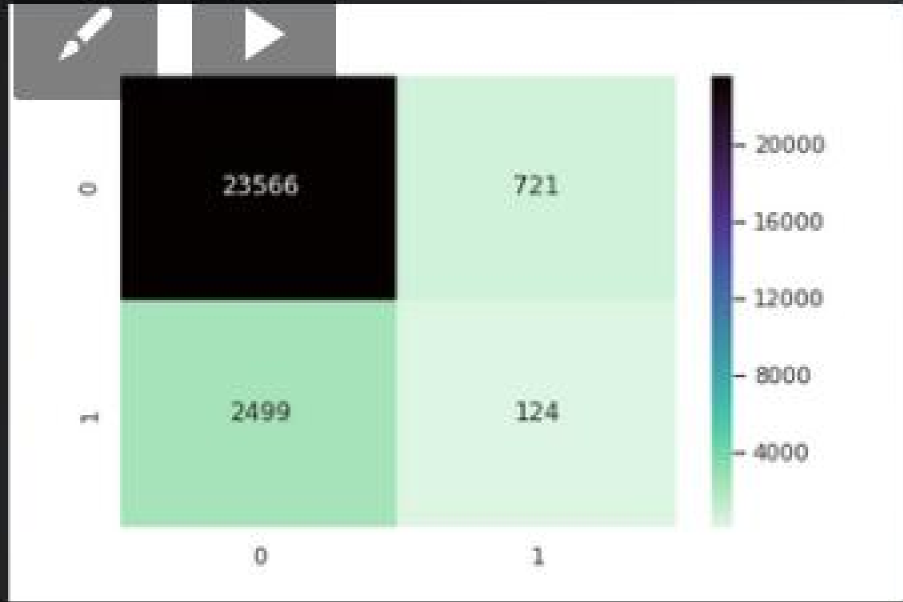
$.5 + A$

Gini Index

$$\frac{A}{A + B} = 2*A$$

PRELIMINARY MODEL

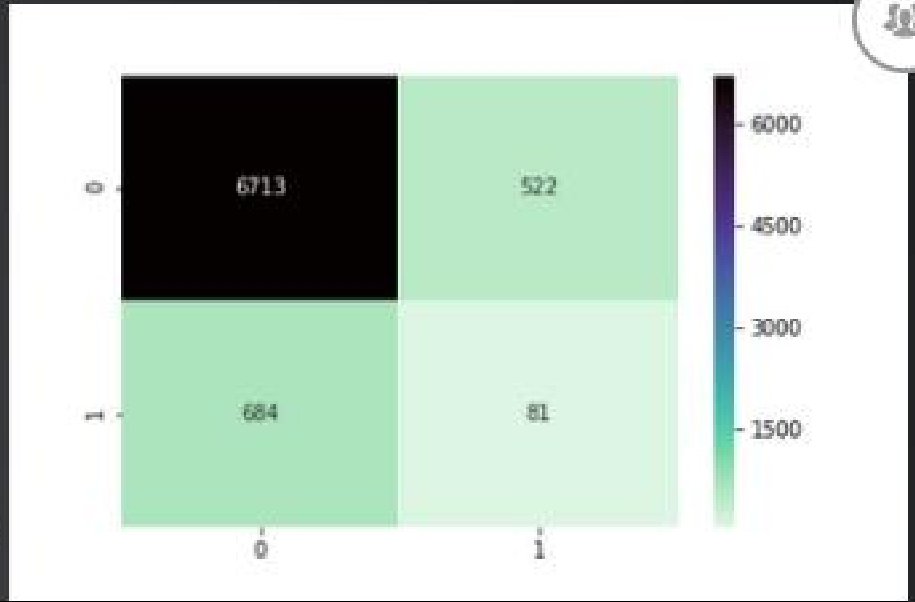
KNN Classifier




```
Classification Report: KNN k=4

              precision    recall  f1-score   support

           0       0.91      0.97      0.94     24305
           1       0.17      0.06      0.09      2605

   micro avg       0.88      0.88      0.88     26910
   macro avg       0.54      0.51      0.51     26910
weighted avg       0.83      0.88      0.85     26910

ROC_AUC SCORE: 0.51
```

```
In [21]: mod_val_sets.shape,sub_val_sets.shape,disp_keys.shape,cont_df.shape
Out[21]: ((67275, 13), (67275, 16), (67275, 31), (67275, 3))

In [44]: model_df = pd.concat([df,add_feats],axis=1)

In [45]: model_df.drop(['SubModel'],1,inplace=True)
    ...: model_df.Model = model_df.Model.str.split().str[0].str.strip()
    ...:
    ...:

In [46]: model_df = pd.get_dummies(model_df)

In [47]: model_df.shape
Out[47]: (67275, 534)
```

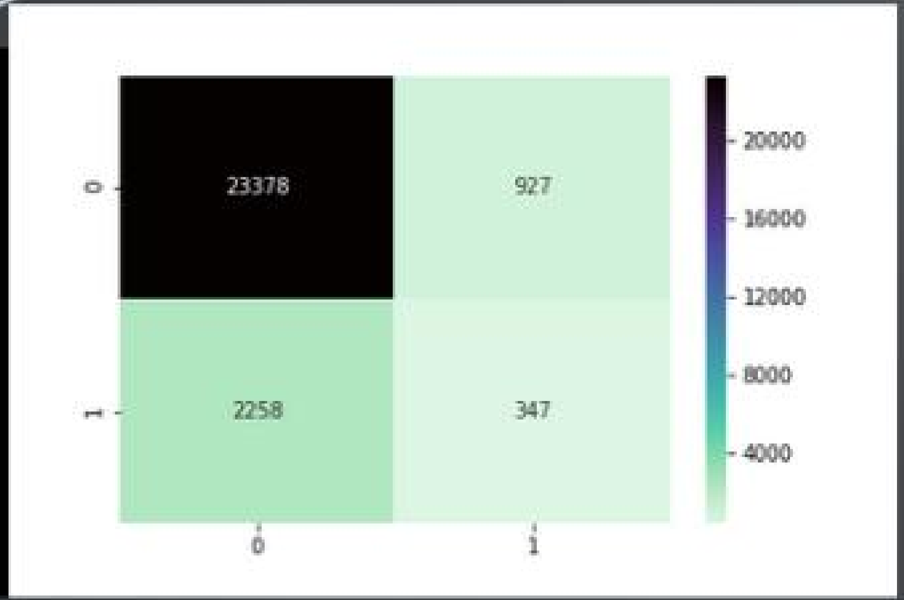
SVC




```
Classification Report: SVC (rbf)

              precision    recall  f1-score   support

           0       0.91      0.93      0.92      7235
           1       0.13      0.11      0.12       765

   micro avg       0.85      0.85      0.85      8000
   macro avg       0.52      0.52      0.52      8000
weighted avg       0.83      0.85      0.84      8000

ROC_AUC SCORE: 0.52
```

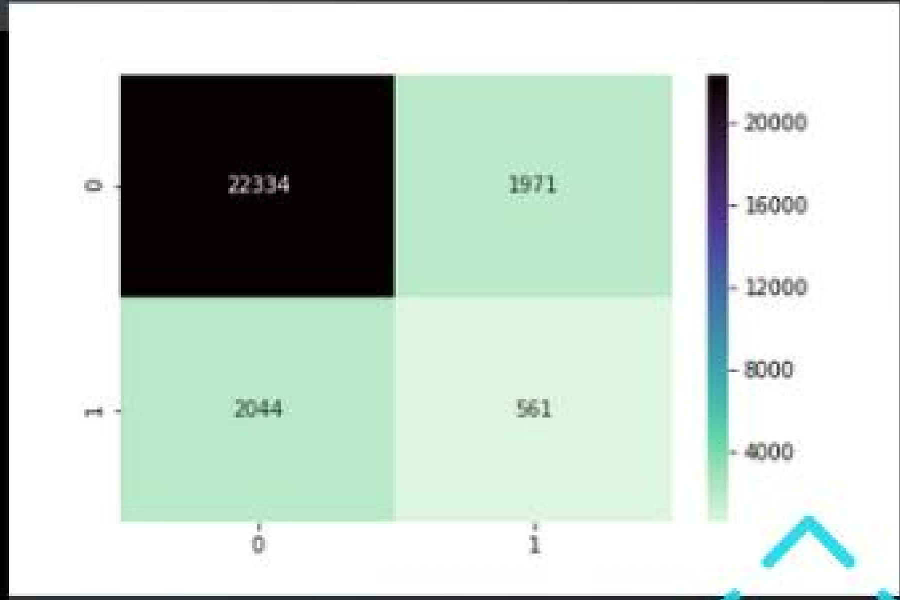
Logistic Regression (lasso)

```
Classification Report: LRC (l1)

              precision    recall  f1-score   support

           0       0.91      0.96      0.94     24305
           1       0.27      0.13      0.18      2605

   micro avg       0.88      0.88      0.88     26910
   macro avg       0.59      0.55      0.56     26910
weighted avg       0.85      0.88      0.86     26910

ROC_AUC SCORE: 0.55
```



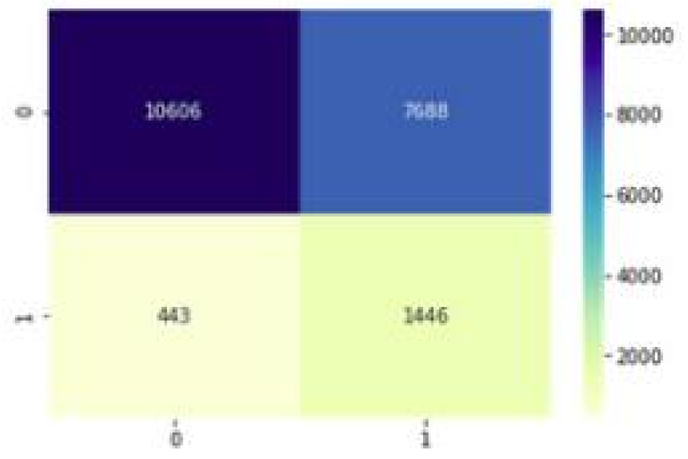
Random Forest

```
Classification Report: Random Forest

              precision    recall  f1-score   support

           0       0.92      0.91      0.91     24305
           1       0.22      0.23      0.22      2605

   micro avg       0.85      0.85      0.85     26910
   macro avg       0.57      0.57      0.57     26910
weighted avg       0.85      0.85      0.85     26910

ROC_AUC SCORE: 0.57
```



FEATURE SELECTION & SMOTE

```
Classification Report: RFC with Tuned Parameters
              precision    recall  f1-score   support

           0       0.96      0.58      0.72     18294
           1       0.16      0.77      0.26      1889

   micro avg       0.60      0.60      0.60     20183
   macro avg       0.56      0.67      0.49     20183
weighted avg       0.88      0.60      0.68     20183

ROC_AUC SCORE: 0.67
```




```
sig_feats = feature_sigs[feature_sigs[0]>0][1]
len(sig_feats)
```

245

	Feature	Significance
0	VehicleAge	0.0442
1	TopThreeAmericanName_GM	0.0428
2	Auction_MANHEIM	0.0361
3	pyear	0.0352
4	WheelTypeID	0.0352
5	TopThreeAmericanName_CHRYSLER	0.0349
6	Make_CHEVROLET	0.0334
7	V6	0.0278
8	SEDAN	0.0264
9	Size_MEDIUM	0.0241

```
new_model = pd.get_dummies(model_df)
new_model = new_model[sig_feats]
```

Keep significant features from Random Forest Classifier.

Use SMOTE to mitigate the class imbalance in the outcome variable

Oversampling and Undersampling

Lemons Before

```
print(model_df.IsBadBuy.describe())

count    67275.000000
mean         0.095637
std          0.294095
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: IsBadBuy, dtype: float64
```

Lemons After

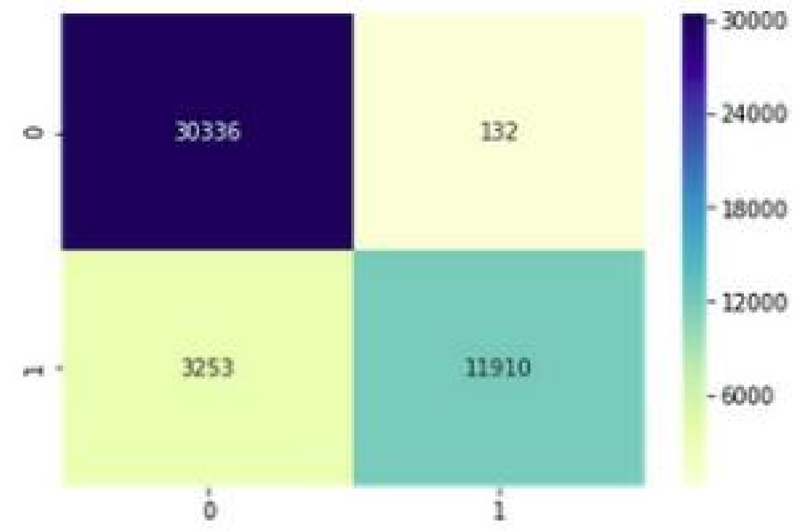
```
print(resampled_y.describe())

count    103429.000000
mean          0.411761
std           0.492155
min           0.000000
25%           0.000000
50%           0.000000
75%           1.000000
max           1.000000
dtype: float64
```

#1. Gradient Bootsting

Classification Report: Gradient Boosting

	precision	recall	f1-score	support
0	0.90	1.00	0.95	30468
1	0.99	0.79	0.88	15163
micro avg	0.93	0.93	0.93	45631
macro avg	0.95	0.89	0.91	45631
weighted avg	0.93	0.93	0.92	45631

ROC_AUC SCORE: 0.89



After eliminating some features and performing SMOTE on the data, the results improved dramatically.

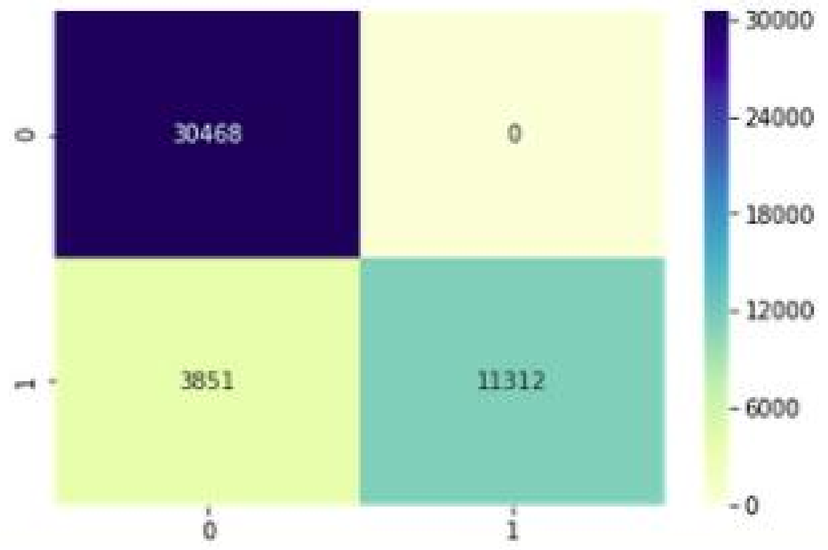
But how would this model do if it was given new, imbalanced information?

2. Random Forest

Classification Report: Optimized Random Forest

	precision	recall	f1-score	support
0	0.89	1.00	0.94	30468
1	1.00	0.75	0.85	15163
micro avg	0.92	0.92	0.92	45631
macro avg	0.94	0.87	0.90	45631
weighted avg	0.93	0.92	0.91	45631

ROC_AUC SCORE: 0.87

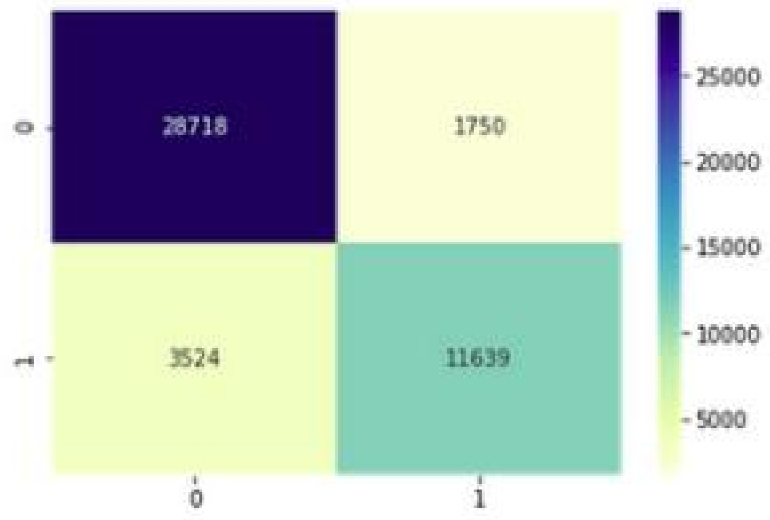


#3. Logistic Regression (lasso)

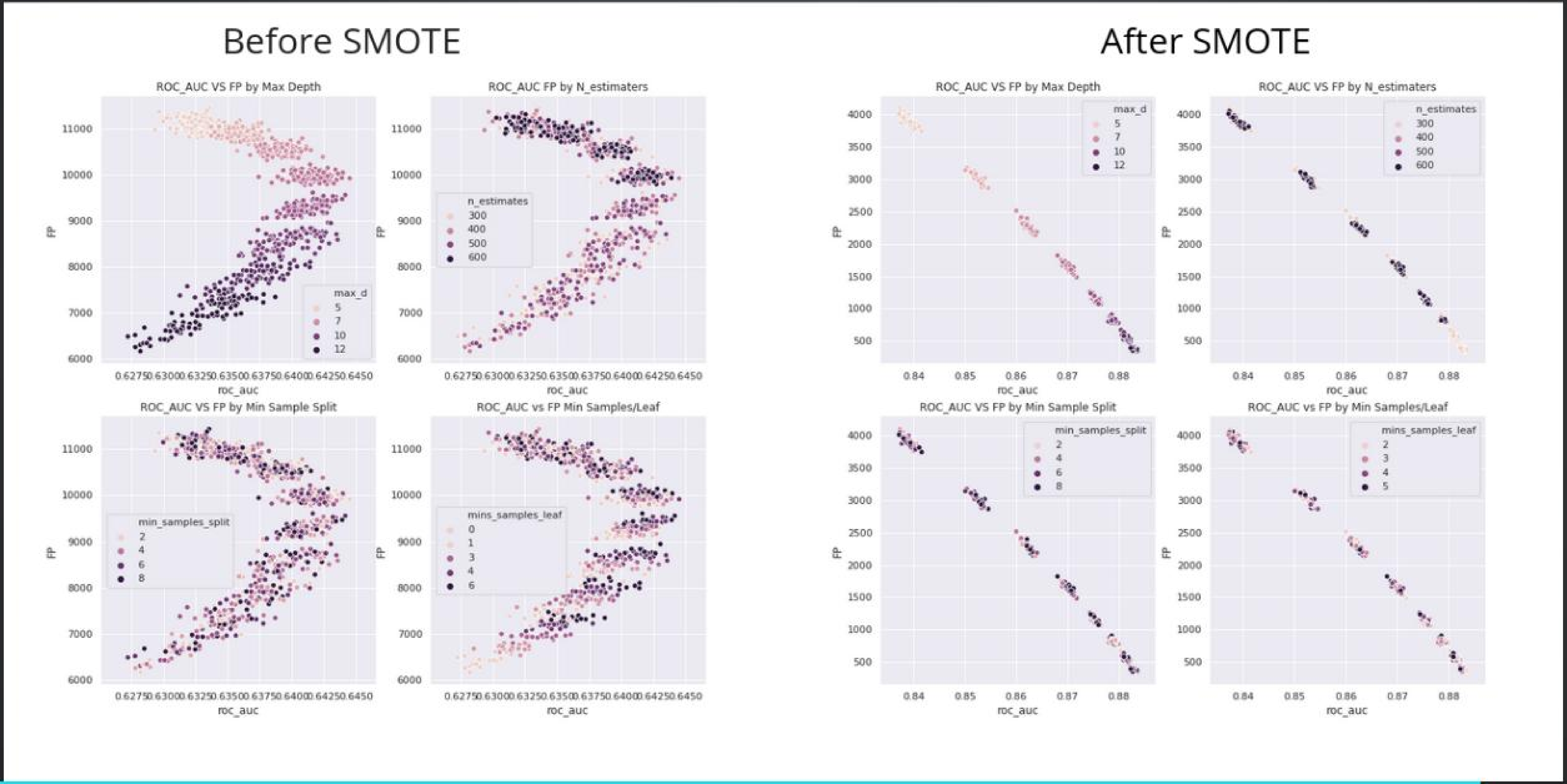
Classification Report: LRC (l1)

	precision	recall	f1-score	support
0	0.89	0.94	0.92	30468
1	0.87	0.77	0.82	15163
micro avg	0.88	0.88	0.88	45631
macro avg	0.88	0.86	0.87	45631
weighted avg	0.88	0.88	0.88	45631

ROC_AUC SCORE: 0.86



SMOTE &HYPERPARAMETERS



SMOTE Data Here

Original Data

Train Model Here

Training Set

SMOTE Training Set

SMOTE Testing Set

Test Set

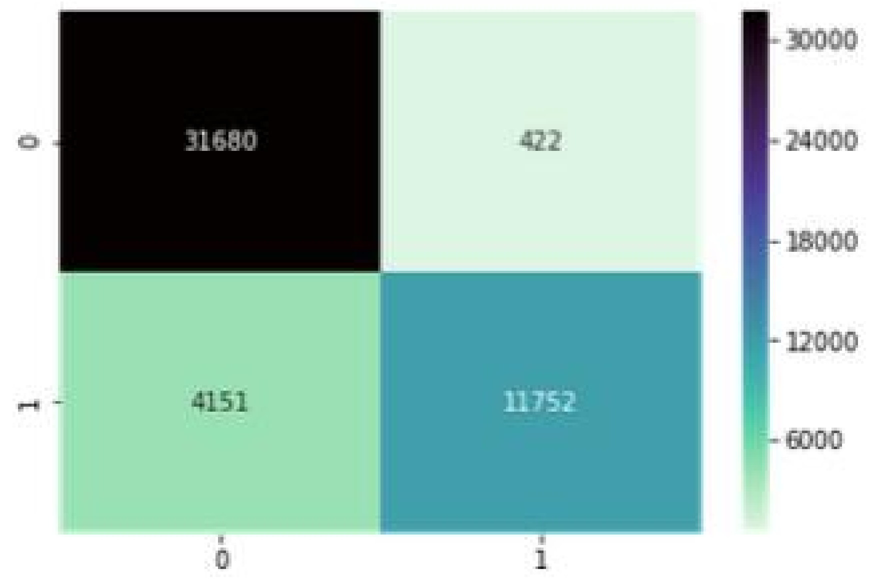
Top Kaggle Entries

Team Members	Score	Entries	Last
	0.27038	210	7y
	0.26929	119	7y
	0.26905	120	7y
	0.26884	57	7y

Classification Report: RFC

	precision	recall	f1-score	support
0	0.88	0.99	0.93	32102
1	0.97	0.74	0.84	15903
micro avg	0.90	0.90	0.90	48005
macro avg	0.92	0.86	0.88	48005
weighted avg	0.91	0.90	0.90	48005

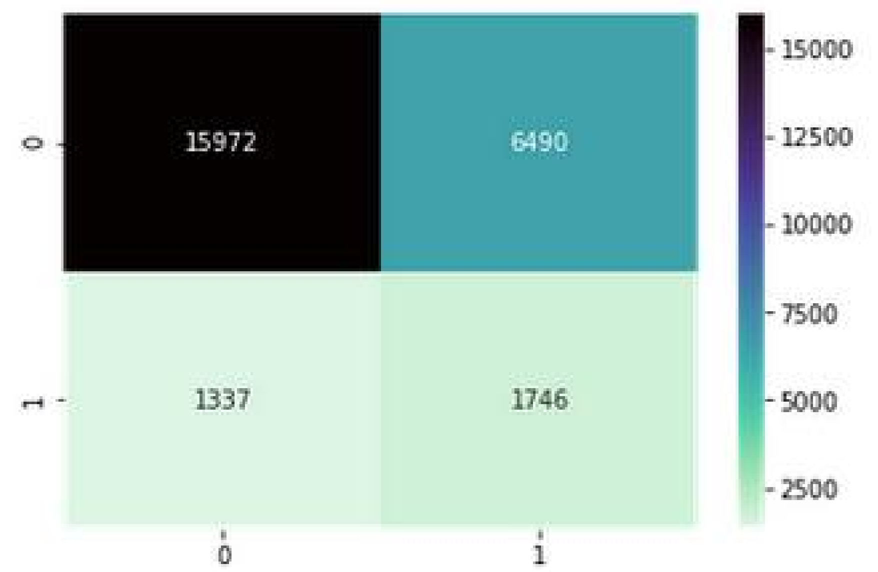
ROC_AUC SCORE: 0.86
Gini Index:0.7239



Classification Report: RFC

	precision	recall	f1-score	support
0	0.92	0.71	0.80	22462
1	0.21	0.57	0.31	3083
micro avg	0.69	0.69	0.69	25545
macro avg	0.57	0.64	0.56	25545
weighted avg	0.84	0.69	0.74	25545

ROC_AUC SCORE: 0.64
Gini Index:0.2802



Conclusion

The model performed extremely well after performing SMOTE on the data.

However, the complexity of the model became apparent after an average score of 76% in a 5-CV cross validation. (recall-macro)

The idea of the model being too complex is further supported by the scores from the leader board on Kaggle.

Next Steps

1. Continue to reduce attributes in order to reduce the complexity of the model.
2. Increase Gini index by further parameter tuning.
3. Subscribe to carfax and use the incident reports to add information to dataset.

Industry Application

While the gap in information symmetry may never be bridged, this model shows that it is capable of helping car dealers avoid lemons.

Through exploratory analysis it is possible to attempt to maximize the profitability of the inventory.

Sources

Lemons - Definition

The Market for Lemons

The Data

kdnuggets

Lemon laws

This publication is for non-commercial educational purposes.

Code:
https://github.com/ExtraLime/lemons

Host
https://slides.com/will-m/lemons/

Will Morgan Nov 2, 2018
willdox7@live.com