Predictive Analytics and Classification:

Auction risk of “kicks” – Vehicles unable to be resold to customers

Data Driven Car Buying Predictor for Carvana

CARVANA

Roadmap

Cleaning and EDA

- Cull columns
- Determine significant features
- Dimensionality Reduction

Model and Pipeline Building

- Build transformers and preprocessor
- Choose logistic regression classifier
- Split into train and test

Model Fit

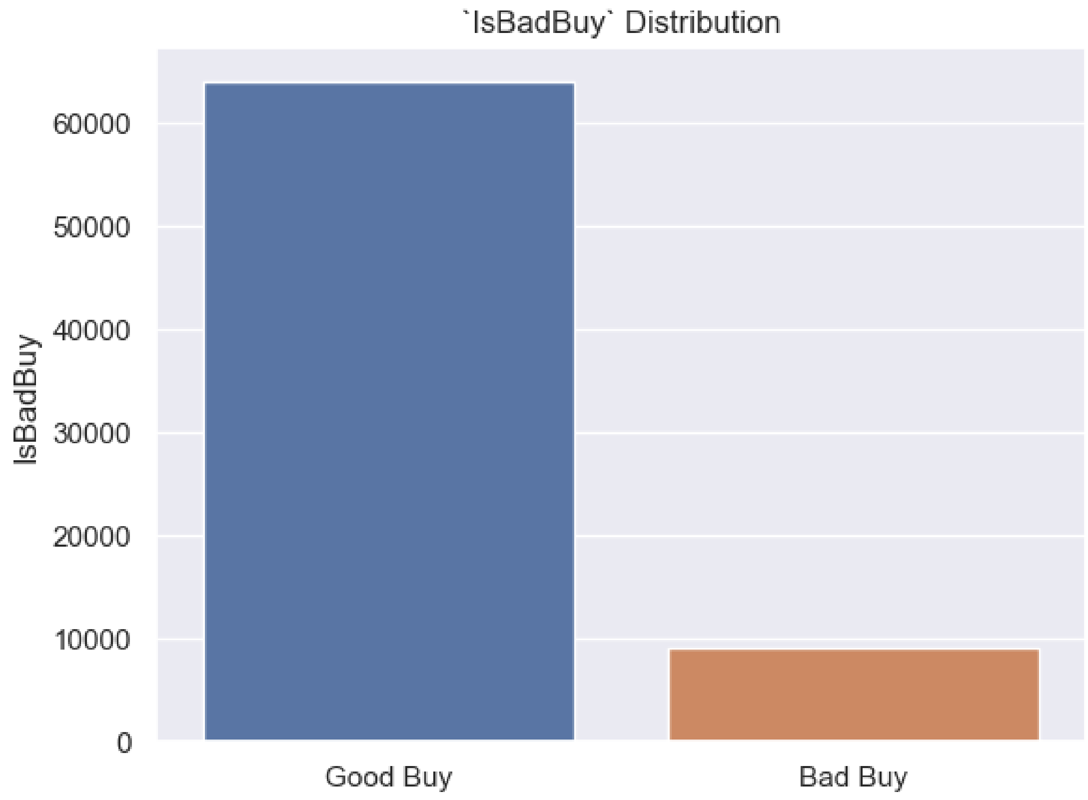
- Fit model to training
- Apply model to test
- Evaluate performance

Test Output

- Retrain model with full training dataset
- Apply model to holdout test data
- Submit to Kaggle

EDA – Target Variable: IsBadBuy

- Quite unbalanced
- Dumb model (which predicts 0 for all) will perform quite well
- **87.70% accuracy** => should be our minimum accuracy for our model



Is Bad Buy	
0	64,007
1	8,976

Cleaning – Column Culling

We want to drop for the following reasons:

- Unique IDs
  - These columns do not impact target variable
- Redundancy
  - Keeping in redundant variables will introduce collinearity
- High Cardinality / Many NULLs
  - Categorical features which have many different values or NULLs will negatively impact performance

Cleaning – Column Culling

```
dropUID = ['RefId', 'BYRNO', 'VNZIP1', 'VNST','WheelTypeID']
dropRedundancy = ['PurchDate', 'VehYear','Nationality']
dropHighCardinality = ['Model','Trim', 'SubModel']
dropInsignificant = ['Transmission']

dropList = dropHighCardinality + dropUID + dropInsignificant
+ dropRedundancy + ['IsBadBuy']
```

Drop UID:

- Unique IDs are not needed

Redundancy:

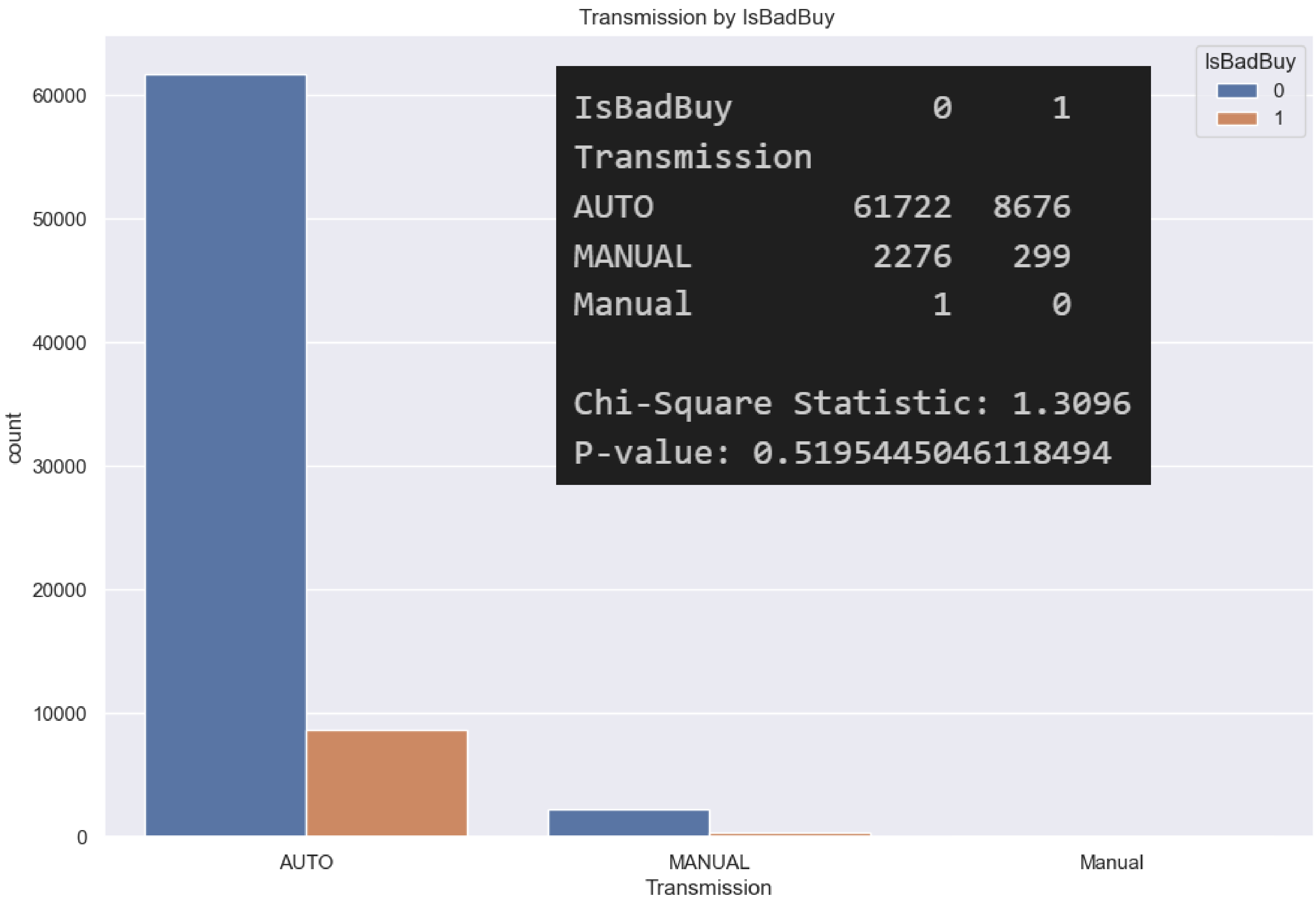
- `Purchase Date` and `Vehicle Year` are captured by `Vehicle Age`
- Nationality is captured by Make

High Cardinality:

- Many options and nulls are contained in `Model`, `Trim`, and `Submodel`

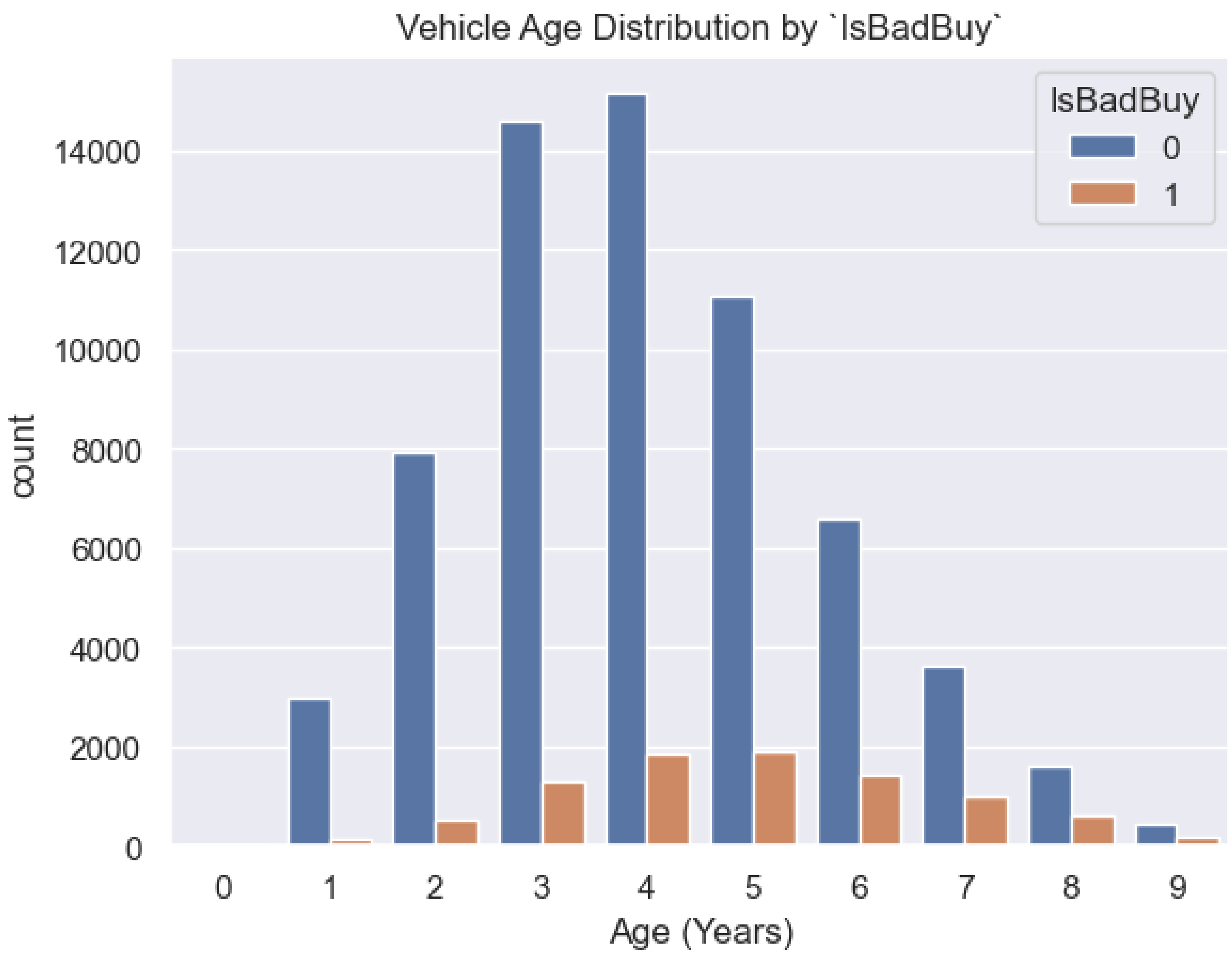
Cleaning – Column Culling

- Performed Chi squared test against transmission feature
- P-Value = 0.5195
- Fail to reject the null hypothesis, transmission does not have a significant effect on target
    - Removing this feature resulted in slightly better model performance



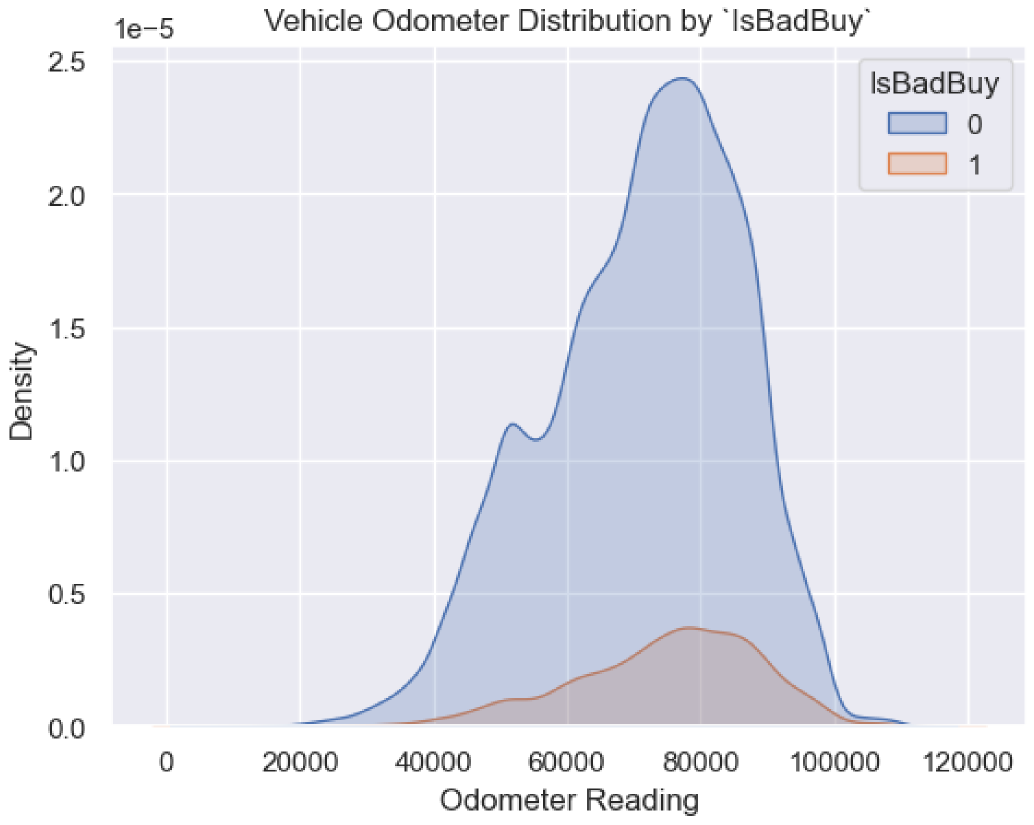
EDA – Feature: VehicleAge

- Distributions clearly have different shapes
  - This will likely be a strong feature for predicting target



EDA – Feature: VehOdo

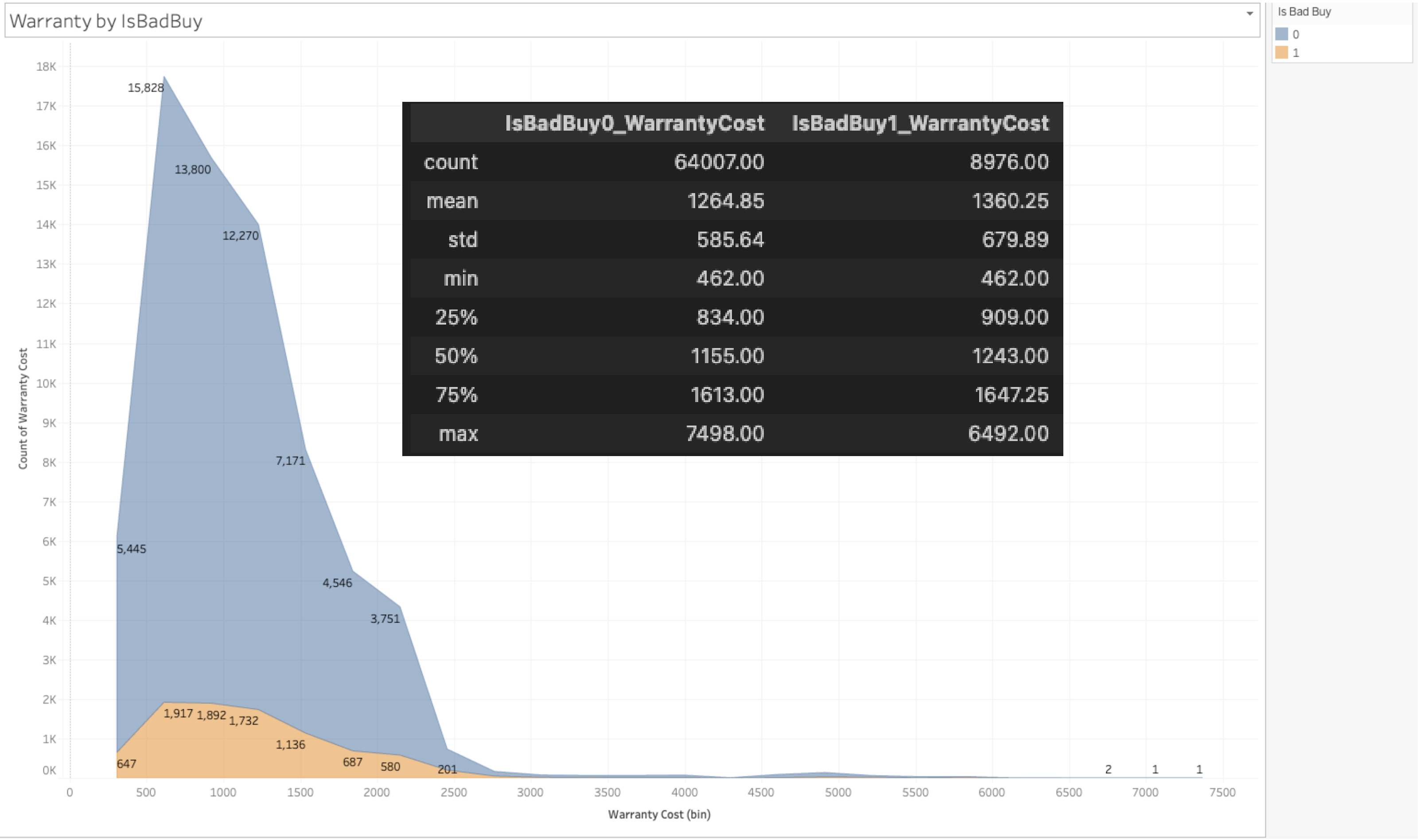
- It also appears that the odometer feature will also be an important numerical feature for the model



	IsBadBuy0_VehOdo	IsBadBuy1_VehOdo
count	64007.00	8976.00
mean	71049.26	74714.15
std	14581.50	14150.97
min	5368.00	4825.00
25%	61302.50	65978.00
50%	72880.00	76545.50
75%	82010.50	84942.00
max	113617.00	115717.00

EDA – Feature: WarrantyCost

WarrantyCost contains different distribution when spliced by IsBadBuy, therefore this will also be a relevant feature for the target



	IsBadBuy0_WarrantyCost	IsBadBuy1_WarrantyCost
count	64007.00	8976.00
mean	1264.85	1360.25
std	585.64	679.89
min	462.00	462.00
25%	834.00	909.00
50%	1155.00	1243.00
75%	1613.00	1647.25
max	7498.00	6492.00

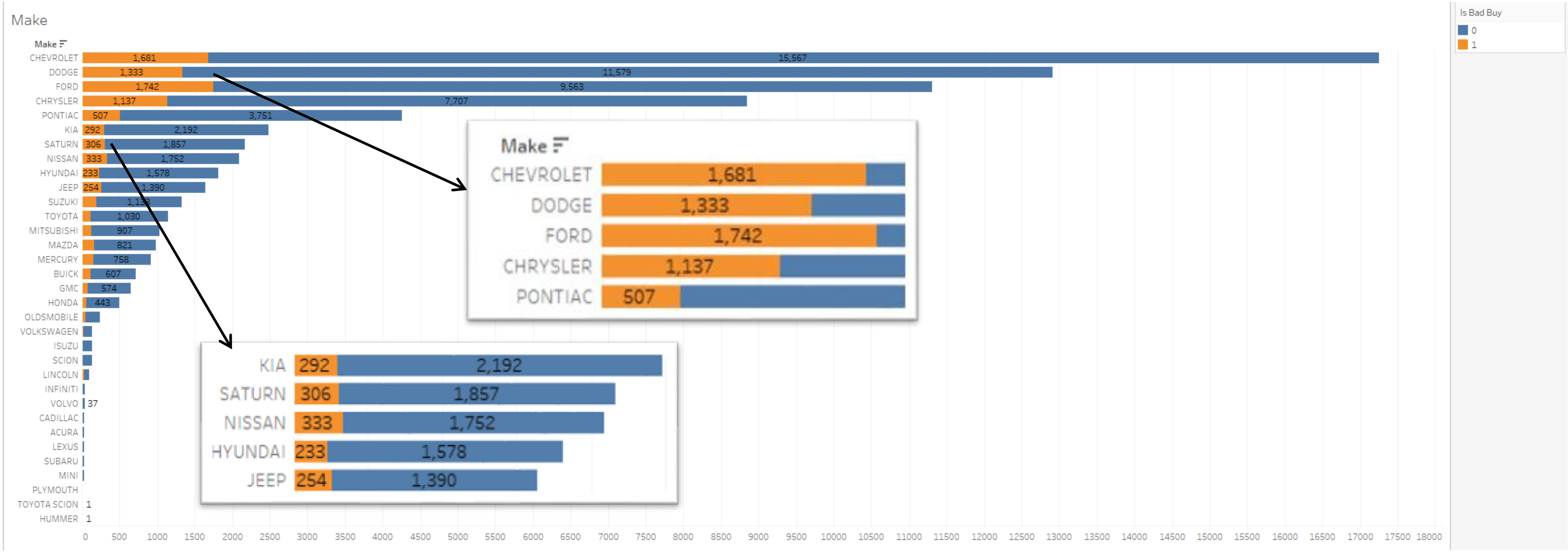
EDA – Plan for Categorical Features

We will profile categorical variables by doing the following:

- Splicing the variables by ` IsBadBuy`
    - Generating visuals of the above
- Getting all dummies and running chi squared tests using scipy.stats => chi2_contingency

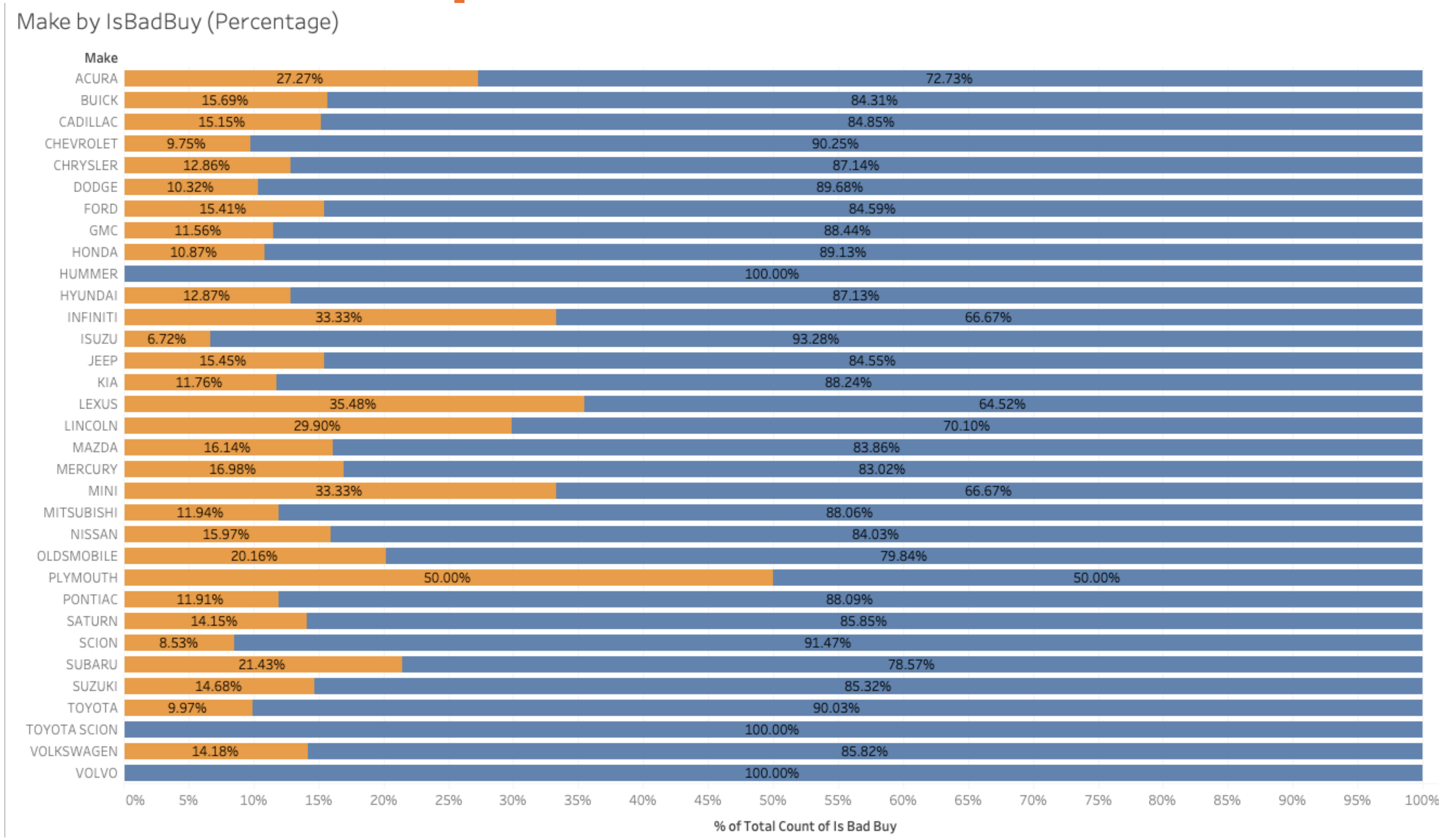
EDA – Feature: Make

- Chevy, Dodge, Ford, Chrysler, Pontaic have high amounts of bad buys
- Next are Kia, Saturn, Nissan, Hyundai, Jeep



EDA – Feature: Make (Percentages)

- When looking at the percentages of IsBadBuy within each Make, we can see that many makes have percentages around 15%



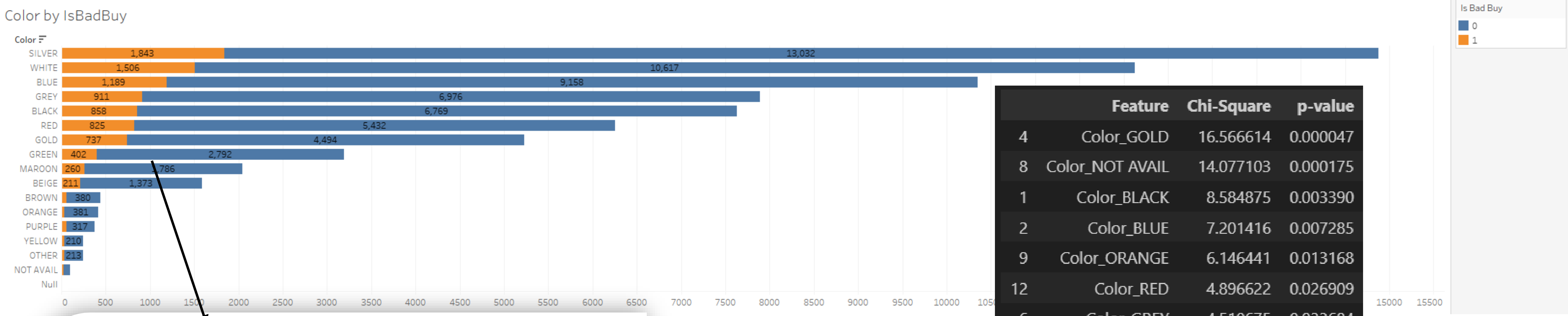
EDA – Feature: Make

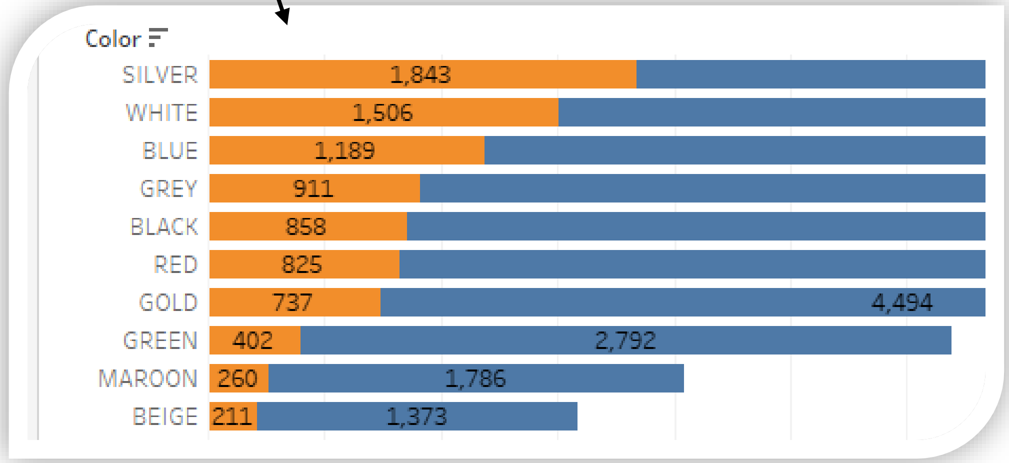
Chi squared test results

	Feature	Chi-Square	p-value
3	Make_CHEVROLET	136.137303	1.861993e-31
6	Make_FORD	119.640842	7.581665e-28
5	Make_DODGE	56.509807	5.591879e-14
21	Make_NISSAN	26.488213	2.651510e-07
16	Make_LINCOLN	26.278156	2.956133e-07
18	Make_MERCURY	18.323525	1.863912e-05
11	Make_INFINITI	15.342471	8.967757e-05
13	Make_JEEP	15.188127	9.731336e-05
15	Make_LEXUS	13.380388	2.542691e-04
22	Make_OLDSMOBILE	13.263432	2.706338e-04
17	Make_MAZDA	13.208417	2.786946e-04
19	Make_MINI	7.993973	4.693330e-03
1	Make_BUICK	7.458988	6.312047e-03
28	Make_SUZUKI	6.909627	8.573285e-03
25	Make_SATURN	6.884166	8.696263e-03
29	Make_TOYOTA	5.650626	1.744914e-02
0	Make_ACURA	5.544438	1.853951e-02
32	Make_VOLVO	4.113176	4.255039e-02
12	Make_ISUZU	3.377369	6.609751e-02
4	Make_CHRYSLER	2.840473	9.191716e-02
27	Make_SUBARU	1.400668	2.366117e-01
26	Make_SCION	1.372014	2.414662e-01
8	Make_HONDA	0.824311	3.639227e-01
14	Make_KIA	0.653097	4.190075e-01
24	Make_PONTIAC	0.605401	4.365246e-01
10	Make_HYUNDAI	0.501045	4.790413e-01
23	Make_PLYMOUTH	0.299135	5.844250e-01
31	Make_VOLKSWAGEN	0.282740	5.949111e-01
7	Make_GMC	0.268854	6.041012e-01
20	Make_MITSUBISHI	0.092162	7.614469e-01
2	Make_CADILLAC	0.054765	8.149703e-01
9	Make_HUMMER	0.000000	1.000000e+00
30	Make_TOYOTA SCION	0.000000	1.000000e+00

EDA – Feature: Color

- The color feature has some good variance amongst the groups; will be good to keep

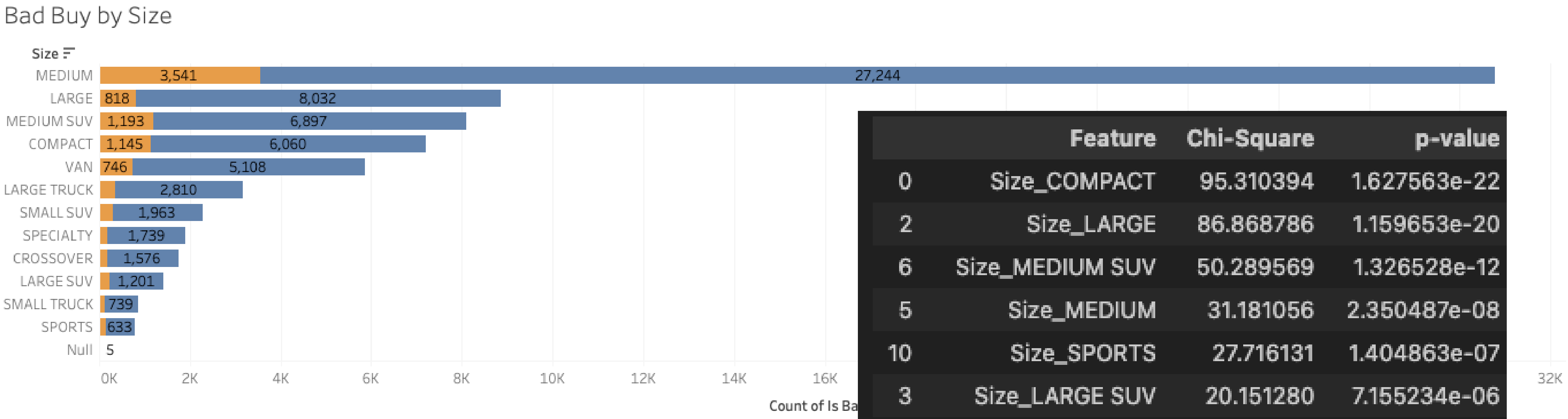




	Feature	Chi-Square	p-value
4	Color_GOLD	16.566614	0.000047
8	Color_NOT AVAIL	14.077103	0.000175
1	Color_BLACK	8.584875	0.003390
2	Color_BLUE	7.201416	0.007285
9	Color_ORANGE	6.146441	0.013168
12	Color_RED	4.896622	0.026909
6	Color_GREY	4.510675	0.033684
11	Color_PURPLE	2.314778	0.128150
0	Color_BEIGE	1.472406	0.224966
15	Color_YELLOW	0.464630	0.495468
7	Color_MAROON	0.288568	0.591140
5	Color_GREEN	0.228586	0.632574
14	Color_WHITE	0.193404	0.660099
13	Color_SILVER	0.133526	0.714804
3	Color_BROWN	0.075401	0.783630
10	Color_OTHER	0.002658	0.958879

EDA – Feature: Size

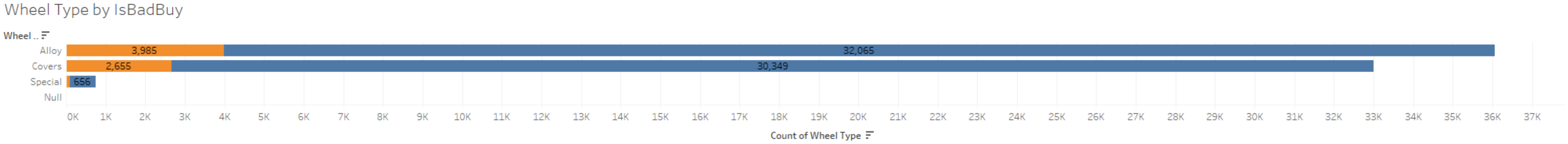
- Size will also be kept as many p-values are below 0.05



	Feature	Chi-Square	p-value
0	Size_COMPACT	95.310394	1.627563e-22
2	Size_LARGE	86.868786	1.159653e-20
6	Size_MEDIUM SUV	50.289569	1.326528e-12
5	Size_MEDIUM	31.181056	2.350487e-08
10	Size_SPORTS	27.716131	1.404863e-07
3	Size_LARGE SUV	20.151280	7.155234e-06
9	Size_SPECIALTY	17.319177	3.159818e-05
1	Size_CROSSOVER	5.822897	1.581887e-02
7	Size_SMALL SUV	4.463048	3.463564e-02
8	Size_SMALL TRUCK	3.612285	5.735425e-02
4	Size_LARGE TRUCK	2.637439	1.043717e-01
11	Size_VAN	1.122340	2.894151e-01

EDA – Feature: Wheel Type

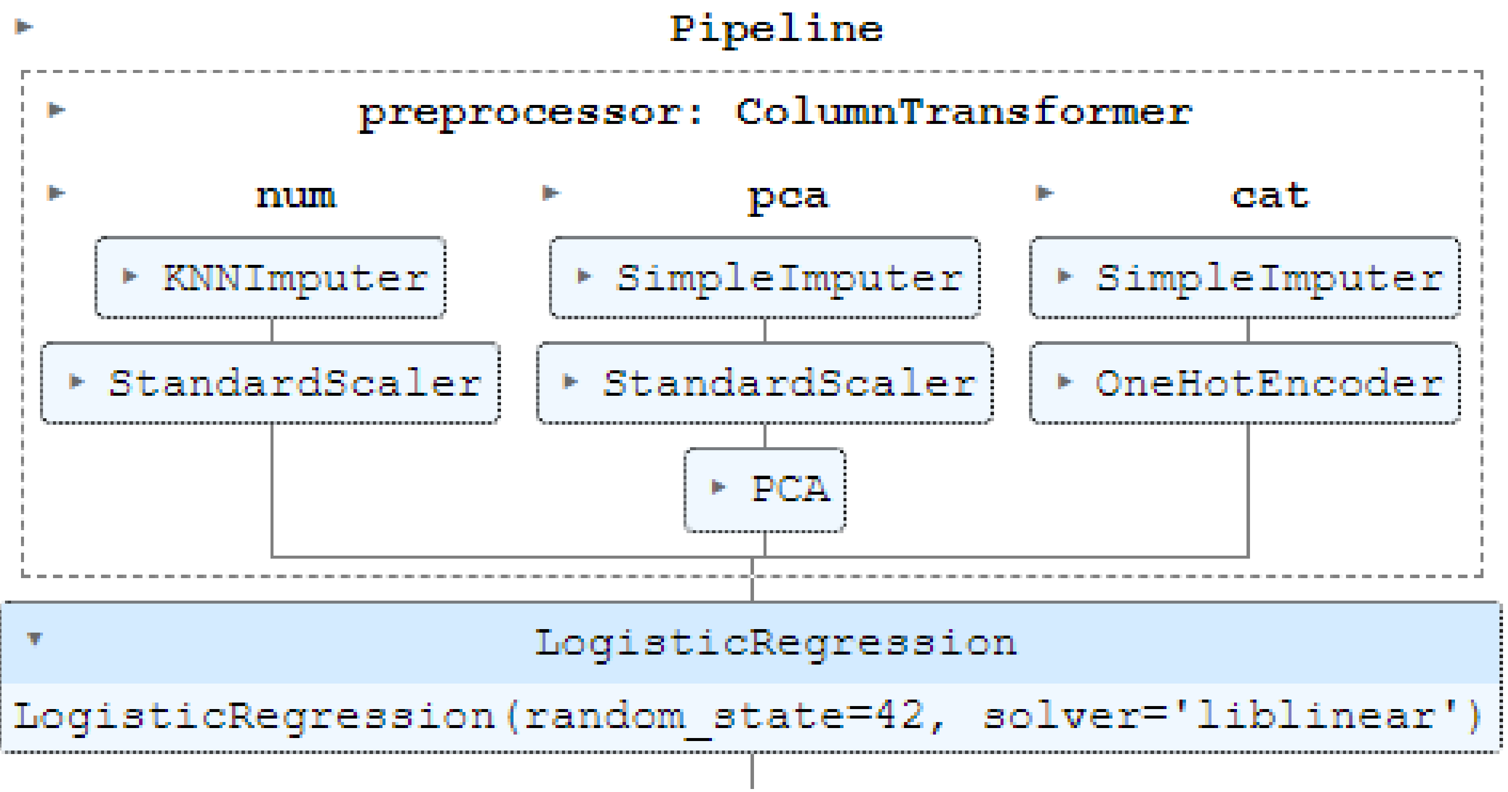
- Wheel type will also be a strong variable, based on the chi squared test results



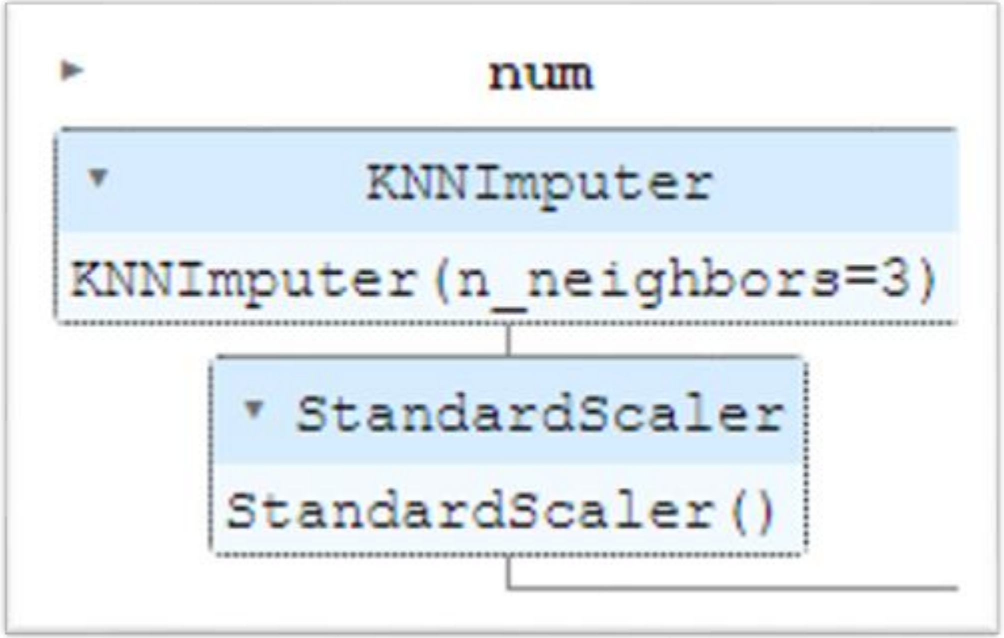
	Feature	Chi-Square	p-value
1	WheelType_Covers	1010.255525	1.059487e-221
0	WheelType_Alloy	102.089389	5.307208e-24
2	WheelType_Special	0.395311	5.295211e-01

Model Building

We will go through each part (column transformer) of the pipeline design



Model Building – **Numerical Processor**



- KNN to fill null values
  - Should give more accurate values to fill rather than mean imputation
- Standardization: removing the mean and scaling to unit variance.
  - Important for regularization and coefficient interpretability

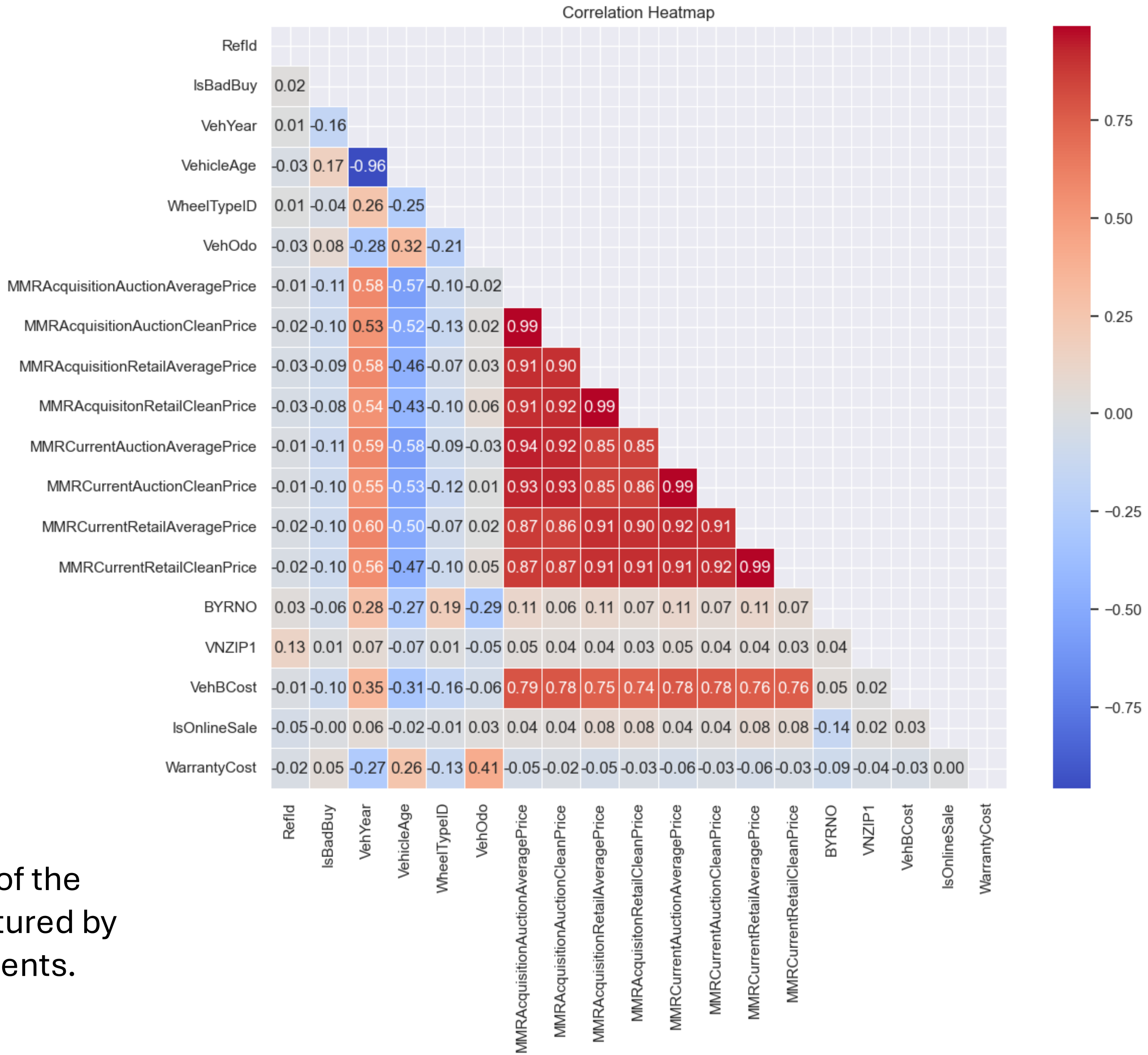
Model Building – **Dimensionality Reduction**

- All MMR numerical features are highly correlated
- To avoid collinearity and still capture variability by these features, we use PCA

```
pca.explained_variance_ratio_.round(3)

>>> array([0.922, 0.037, 0.032, 0.005, 0.004, 0.001, 0. , 0. ])
```

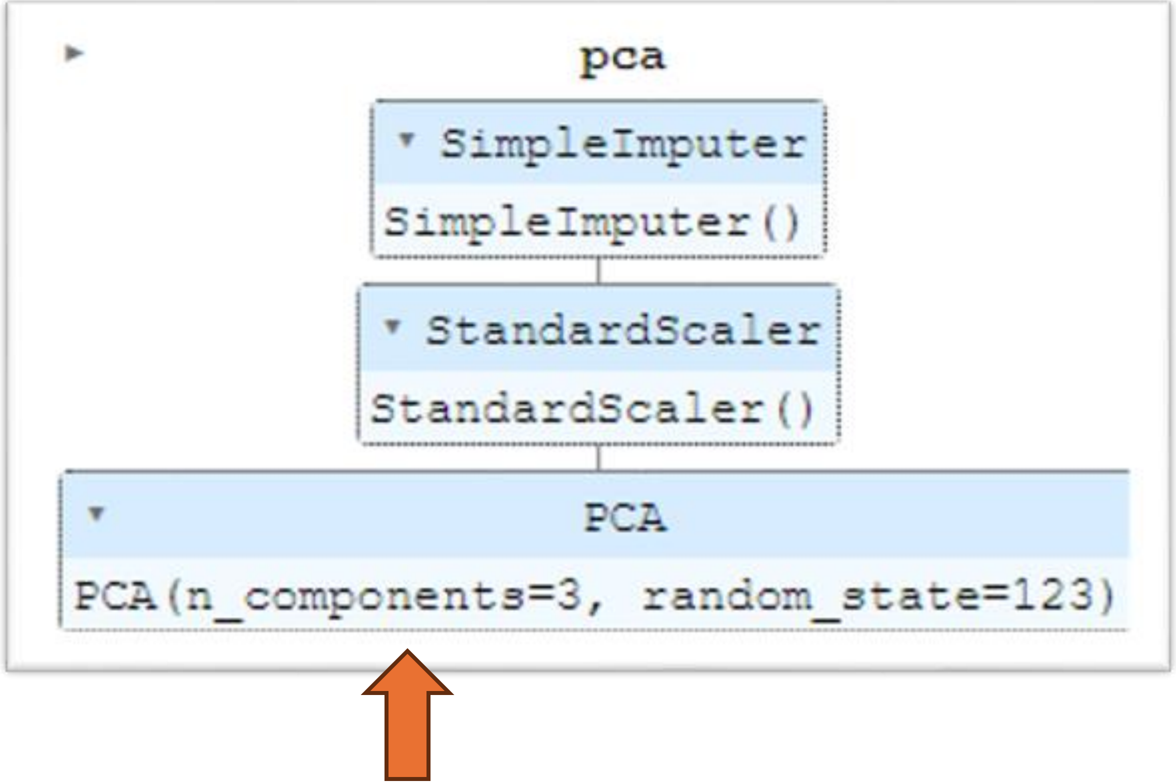
pca1, pca2, pca3

A large percent of the variance is captured by using 3 components.



Model Building –
Dimensionality Reduction

```python
# Define the columns for PCA
pca_columns = ['MMRAcquisitionAuctionAveragePrice',
               'MMRAcquisitionAuctionCleanPrice',
               'MMRAcquisitionRetailAveragePrice',
               'MMRAcquisitonRetailCleanPrice',
               'MMRCurrentAuctionAveragePrice',
               'MMRCurrentAuctionCleanPrice',
               'MMRCurrentRetailAveragePrice',
               'MMRCurrentRetailCleanPrice']
```

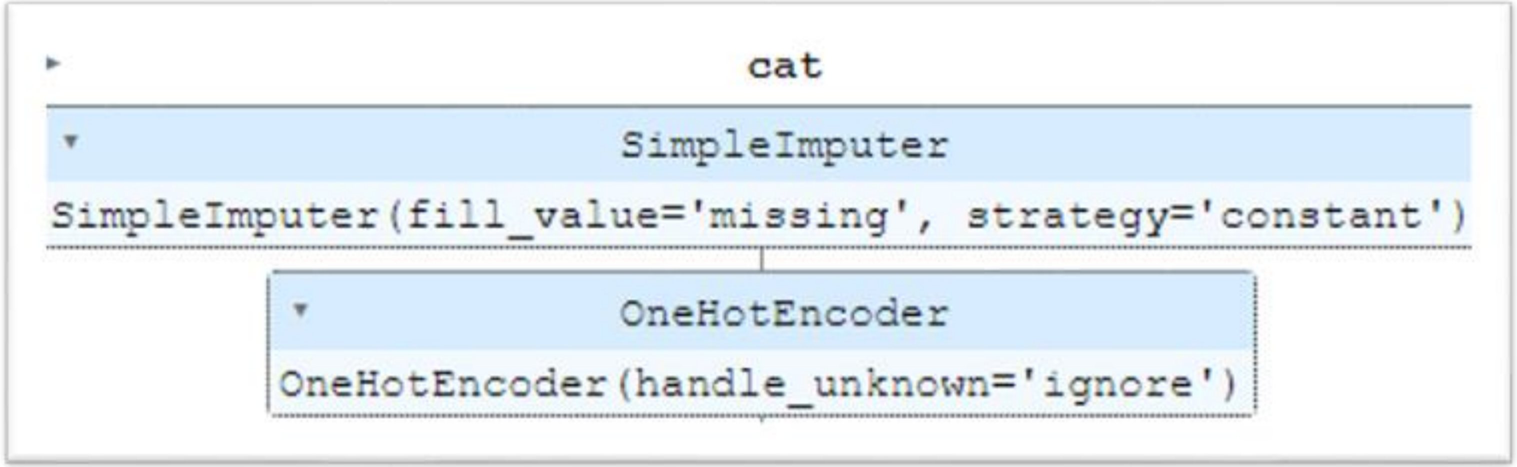


pca1, pca2, pca3

A large percent of the variance is captured by using 3 components.

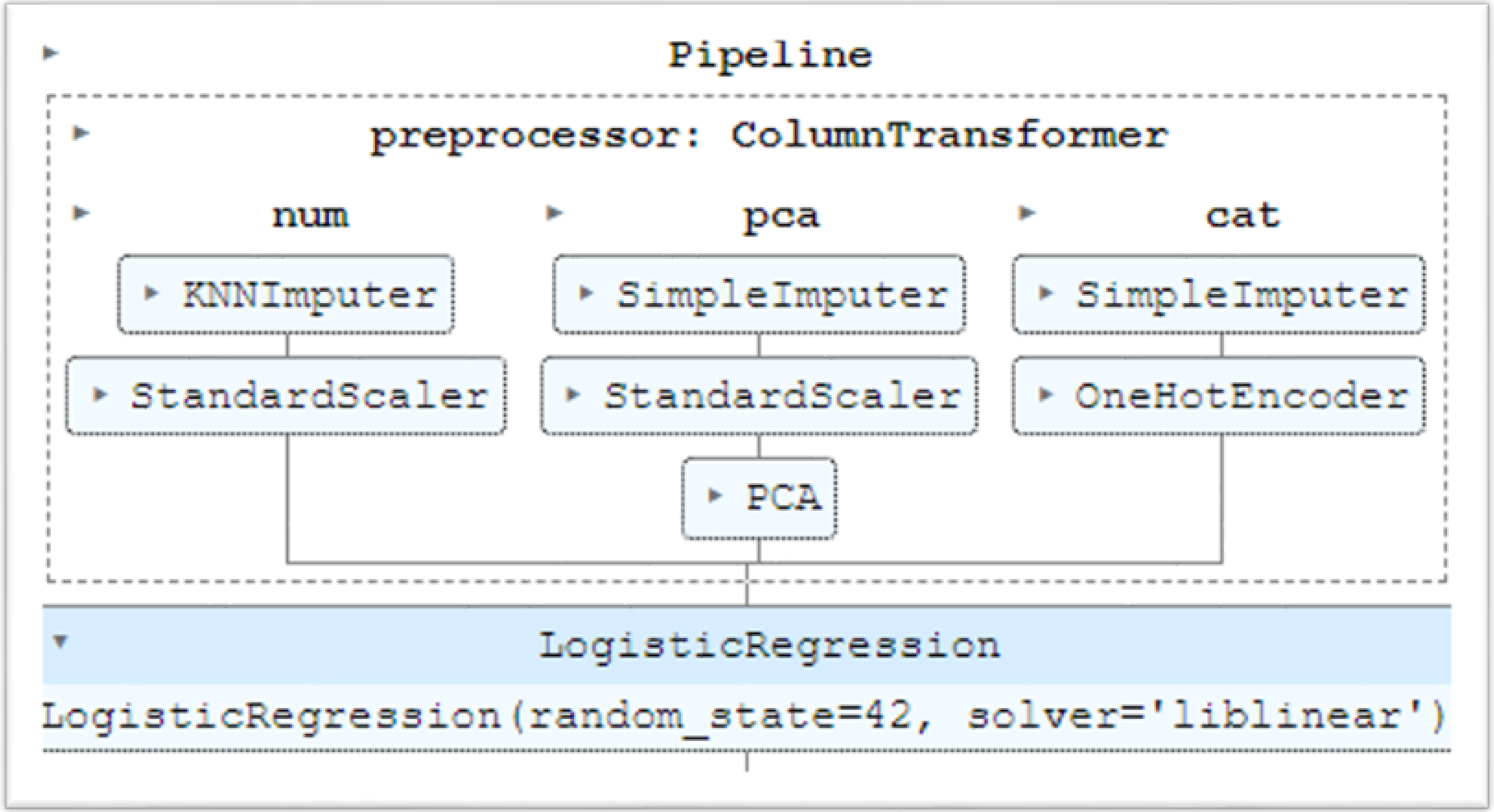
Model Building – **One Hot Encoder (Categorical)**

- Handle null values by filling with string:` missing`
  - Logit model requires data filled in all cells; `null` will be interpreted as `NaN` so we do this for readability
- Apply encoding of categorical values



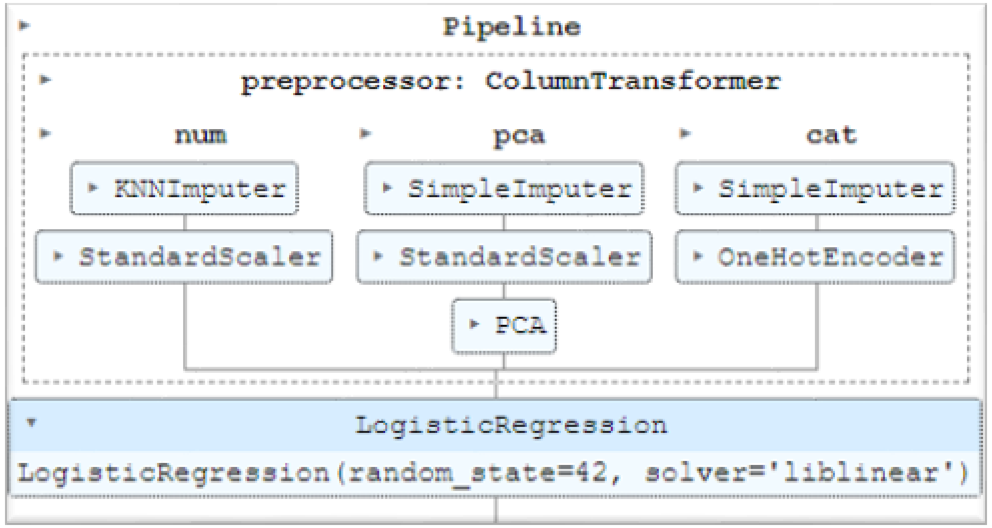
Model Building - **Classifier**

- Finally, choose the logistic regression classifier
- Building a 'pipeline' allows us to easily configure:
  - Testing of different features; less refactoring required
  - Application of different imputation and dimensionality reduction methods



Model Fit – **Train Test Split**

```
# Split the data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# Define the model

model = Pipeline(steps=[

    ('preprocessor', preprocessor),

    ('classifier', LogisticRegression(solver='liblinear',
random_state=42))

])

# Fit the model

model.fit(X_train, y_train)
```



- Reserve 0.2 (20%) of training data for model evaluation
- Create model
- Fit model using training data

Model Prediction – **Training Data**

```
# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
conf_matrix = confusion_matrix(y_test, y_pred)
classification_report_str =
classification_report(y_test, y_pred)

print(classification_report_str, '\nConfusion
Matrix:\n', conf_matrix)
```

Metric Report Output

```
              precision    recall  f1-score   support

           0       0.91      0.99      0.94     12850
           1       0.70      0.24      0.36      1747

    accuracy                           0.90     14597
   macro avg       0.80      0.61      0.65     14597
weighted avg       0.88      0.90      0.87     14597

Confusion Matrix:
 [[12669   181]
 [ 1327   420]]
```

Model Prediction – **Evaluation**

- Model performs well in terms of TP and FP
- Model performs somewhat inadequately for TN and FN
- Recall that we stated the following at the beginning: **87.70% accuracy => should be our <u>minimum</u> for the model**
- We obtained 90% accuracy, so our model performs better than the dumb model

Metric Report Output

```
              precision    recall  f1-score   support

           0       0.91      0.99      0.94     12850
           1       0.70      0.24      0.36      1747

    accuracy                           0.90     14597
   macro avg       0.80      0.61      0.65     14597
weighted avg       0.88      0.90      0.87     14597

Confusion Matrix:
 [[12669   181]
 [ 1327   420]]
```

Model Evaluation – **Pseudo R^2 $(McFadden)$**

The McFadden pseudo R^2 compares the log likelihood of the full model (L) to the log likelihood of the model with just the intercept (L_0, the null model).

$$R^2 = 1.0 - \frac{\ln(L)}{\ln(L_0)}.$$

Sci-Kit Learn doesn't have a function for this, so we shall use the above definition to evaluate it manually

Model Evaluation – **Pseudo R^2 $(McFadden)$**

```python
# Preprocess the test set
X_test_preprocessed = model['preprocessor'].transform(X_test)

# Fit the null model
null_model = LogisticRegression(solver='liblinear', random_state=42)
null_model.fit(np.ones((X_train.shape[0], 1)), y_train)

# Get the log likelihood for the null model
null_prob = null_model.predict_proba(np.ones((X_test.shape[0], 1)))[:, 1]
log_likelihood_null_model = np.sum(y_test * np.log(null_prob) + (1 - y_test) * np.log(1
- null_prob))

# Get the log likelihood for the full model
full_prob = model.predict_proba(X_test)[:, 1]
log_likelihood_full_model = np.sum(y_test * np.log(full_prob) + (1 - y_test) * np.log(1
- full_prob))

# Calculate McFadden's pseudo R-squared
pseudo_r_squared = 1 - (log_likelihood_full_model / log_likelihood_null_model)
print(f"McFadden's pseudo R-squared: {pseudo_r_squared}")
```

- Create instance of transformed X_test
- Create instance of null model (simply predicts majority class)
- Definition of likelihood

$$\ell = \sum_{i=1}^{n} [y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

- Definition of McFadden

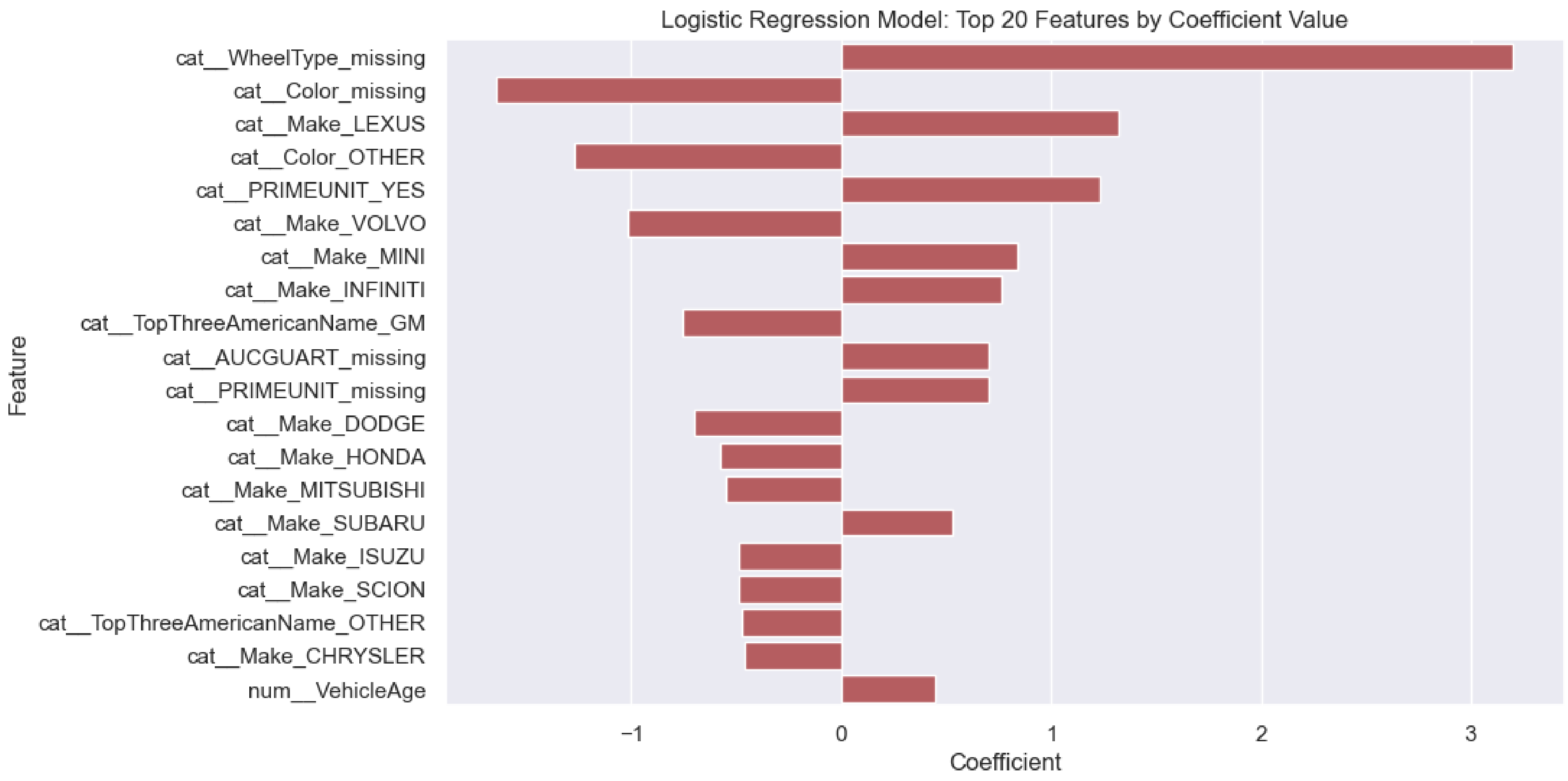
Model Evaluation – **Pseudo R^2 $(McFadden)$**

```
# Preprocess the test set
X_test_preprocessed = model['preprocessor'].transform(X_test)

# Fit the null model
null_model = LogisticRegression(solver='liblinear', random_state=42)
null_model.fit(np.ones((X_train.shape[0], 1)), y_train)

# Get the log likelihood for the null model
null_prob = null_model.predict_proba(np.ones((X_test.shape[0], 1)))[:,
1]
log_likelihood_null_model = np.sum(y_test * np.log(null_prob) + (1 -
y_test) * np.log(1 - null_prob))

# Get the log likelihood for the full model
full_prob = model.predict_proba(X_test)[:, 1]
log_likelihood_full_model = np.sum(y_test * np.log(full_prob) + (1 -
y_test) * np.log(1 - full_prob))

# Calculate McFadden's pseudo R-squared
pseudo_r_squared = 1 - (log_likelihood_full_model /
log_likelihood_null_model)
print(f"McFadden's pseudo R-squared: {pseudo_r_squared}")
```

```
>>>
McFadden's pseudo-R-squared:
0.1644701988577303
```

Model Evaluation - **Coefficients**

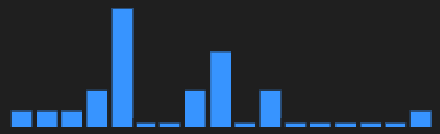
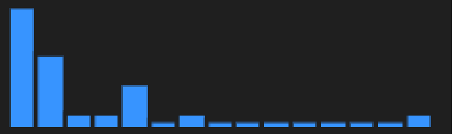
Top 20 coefficients by absolute value



- Categorical variables are the most important features, specifically
  - Missing WheelType, Color Missing, and various manufacturers
- VehicleAge was the most important numerical feature

Model Evaluation - **Coefficients**

- Positive => increase log odds
  - (pulls towards predicting 1)
- Negative => decrease log odds
  - (pulls towards predicting 0)

Feature	Coefficient	abs_coefficient
Missing: 0 (0%) Distinct: 20 (100%) 20 Distinct values	Missing: 0 (0%) Distinct: 19 (95%)  Min -1.643187402... Max 3.2018261745...	Missing: 0 (0%) Distinct: 19 (95%)  Min 0.4453449119... Max 3.201826174...
cat__WheelType_missing	3.2018261746	3.2018261746
cat__Color_missing	-1.6431874029	1.6431874029
cat__Make_LEXUS	1.3235120591	1.3235120591
cat__Color_OTHER	-1.2702212596	1.2702212596
cat__PRIMEUNIT_YES	1.2306793437	1.2306793437
cat__Make_VOLVO	-1.0194766177	1.0194766177
cat__Make_MINI	0.8391517538	0.8391517538
cat__Make_INFINITI	0.7632711626	0.7632711626
cat__TopThreeAmericanName_GM	-0.7540623822	0.7540623822
cat__AUCGUART_missing	0.6999365859	0.6999365859
cat__PRIMEUNIT_missing	0.6999365859	0.6999365859
cat__Make_DODGE	-0.6997347633	0.6997347633
cat__Make_HONDA	-0.5758981396	0.5758981396
cat__Make_MITSUBISHI	-0.5461449453	0.5461449453
cat__Make_SUBARU	0.5288889602	0.5288889602
cat__Make_ISUZU	-0.4899470639	0.4899470639
cat__Make_SCION	-0.4839919915	0.4839919915
cat__TopThreeAmericanName_OTHE	-0.4740121248	0.4740121248
cat__Make_CHRYSLER	-0.4595486878	0.4595486878
num__VehicleAge	0.4453449119	0.4453449119

Model Evaluation – **All Coefficients (Page 1)**

There are a lot of features; we will review this decision in the discussion section

(Predictive Power vs. Statistical Significance)

index	Feature	Coefficient	Std Err	p-value
	Missing: 0 (0%) Distinct: 82 (100%)	Missing: 0 (0%) Distinct: 82 (100%)	Missing: 5 (6%) Distinct: 76 (93%)	Missing: 5 (6%) Distinct: 76 (93%)
	82 Distinct values	Min -5.07816099... Max 3.219374640...	Min 0.0118130176... Max 58430542452...	Min 0 Max 1
const	const	-2.6421053971	21.6420554358	0.9028340777
x1	num__VehicleAge	0.4444057316	0.0250252196	1.486299929e-70
x2	num__VehOdo	0.1142236079	0.0175785149	1e-10
x3	num__VehBCost	-0.2805326975	0.032486373	5.852391797e-18
x4	num__IsOnlineSale	-0.0264683074	0.0149203388	0.0760672931
x5	num__WarrantyCost	0.058189485	0.0206645148	0.004863877
x6	pca__pca0	0.0532743747	0.0118130177	0.0000064888
x7	pca__pca1	-0.087530012	0.0322085611	0.0065757118
x8	pca__pca2	0.0202401885	0.0317203913	0.5234207592
x9	cat__Auction_MANHEIM	0.0144927814	0.0359804277	0.6870981191
x10	cat__Auction_OTHER	-0.1317028136	0.0434303719	0.002425332
x11	cat__Make_BUICK	-0.2775832843	17.357626631	0.9872407685
x12	cat__Make_CADILLAC	-0.7869752214	17.3982552811	0.9639215937
x13	cat__Make_CHEVROLET	-0.2643472172	17.3448441985	0.9878401676
x14	cat__Make_CHRYSLER	-0.7403642013	*Missing value*	*Missing value*
x15	cat__Make_DODGE	-0.9803704014	21.741009192	0.9640330682
x16	cat__Make_FORD	-0.5063053462	*Missing value*	*Missing value*
x17	cat__Make_GMC	-0.2554694214	18.1240864997	0.9887537307
x18	cat__Make_HONDA	-0.2595414412	0.602284918	0.6665207162
x19	cat__Make_HUMMER	-0.4317773378	17.9785879015	0.9808396878
x20	cat__Make_HYUNDAI	0.0305171946	0.5825408306	0.9582208411
x21	cat__Make_INFINITI	1.2229368858	0.6852247937	0.074306155
x22	cat__Make_ISUZU	-0.2158681942	0.6836200679	0.7521754411
x23	cat__Make_JEEP	-0.681230378	21.7662127285	0.9750321986
x24	cat__Make_KIA	0.2771006125	0.7376125862	0.7071605376
x25	cat__Make_LEXUS	1.9313705876	0.7368403393	0.008763316
x26	cat__Make_LINCOLN	0.1828237607	*Missing value*	*Missing value*
x27	cat__Make_MAZDA	0.3978391237	0.8669559698	0.646311753
x28	cat__Make_MERCURY	-0.3516185369	*Missing value*	*Missing value*
x29	cat__Make_MINI	1.4194871566	0.7389805675	0.0547478336
x30	cat__Make_MITSUBISHI	-0.2238062425	0.5858435171	0.7024436152
x31	cat__Make_NISSAN	0.3854081856	0.5777518711	0.5047194121
x32	cat__Make_OLDSMOBILE	0.0595854707	17.2944885883	0.9972510183
x33	cat__Make_PLYMOUTH	2.7949193499	22.2566257099	0.9000668035
x34	cat__Make_PONTIAC	-0.0586163071	17.2913409194	0.9972952383
x35	cat__Make_SATURN	-0.0392602799	17.2266793344	0.9981815913
x36	cat__Make_SCION	-0.2309724475	0.6884289182	0.7372429649
x37	cat__Make_SUBARU	1.0990002697	0.8205654456	0.1804662996
x38	cat__Make_SUZUKI	0.6160887404	0.5798683913	0.2880255232
x39	cat__Make_TOYOTA	-0.0076781598	0.7399425344	0.9917207419

Model Evaluation – **All Coefficients (Page 2)**

There are a lot of features; we will review this decision in the discussion section

(Predictive Power vs. Statistical Significance)

Export as file Refresh data Report an issue

index	Feature	Coefficient	Std Err	p-value
	Missing: 0 (0%) Distinct: 82 (100%) 82 Distinct values	Missing: 0 (0%) Distinct: 82 (100%) Min -5.07816099... Max 3.219374640...	Missing: 5 (6%) Distinct: 76 (93%) Min 0.0118130176... Max 58430542452...	Missing: 5 (6%) Distinct: 76 (93%) Min 0 Max 1
x41	cat__Make_VOLKSWAGEN	0.1454523819	0.6322250938	0.8180418567
x42	cat__Make_VOLVO	-5.0781609955	8.7548410848	0.5618874147
x43	cat__Color_BLACK	0.0802082965	0.1046104754	0.4432403194
x44	cat__Color_BLUE	0.0107248314	0.1029135014	0.9170010372
x45	cat__Color_BROWN	0.2754065457	0.1927802268	0.1531181695
x46	cat__Color_GOLD	0.1088988739	0.1069164377	0.3084204887
x47	cat__Color_GREEN	-0.0631414311	0.1155192986	0.5846620014
x48	cat__Color_GREY	0.0497144735	0.1050274457	0.6359653151
x49	cat__Color_MAROON	0.0937805254	0.124194698	0.4501836261
x50	cat__Color_NOT AVAIL	-0.1383593743	0.3650158065	0.7046504644
x51	cat__Color_ORANGE	-0.049217326	0.2385198643	0.8365216112
x52	cat__Color_OTHER	-1.3583383135	0.2779770219	0.0000010264
x53	cat__Color_PURPLE	0.1105685433	0.214664178	0.6064993363
x54	cat__Color_RED	0.1433957506	0.1054320803	0.173805549
x55	cat__Color_SILVER	0.0773943079	0.0990922538	0.4347831593
x56	cat__Color_WHITE	0.0675890716	0.1008127774	0.5025763749
x57	cat__Color_YELLOW	-0.2681208053	0.2469680007	0.277633878
x58	cat__Color_missing	-3.0897521984	1.1161351352	0.0056356484
x59	cat__WheelType_Covers	-0.081772727	0.0346331867	0.0182203769
x60	cat__WheelType_Special	0.162819824	0.1246714103	0.1915553886
x61	cat__WheelType_missing	3.2193746402	0.052936991	0
x62	cat__Size_CROSSOVER	-0.3563172059	0.1218725947	0.0034591343
x63	cat__Size_LARGE	-0.314728469	0.0719470227	0.000012174
x64	cat__Size_LARGE SUV	0.0686049195	0.1315129319	0.6019076968
x65	cat__Size_LARGE TRUCK	-0.2161667917	0.1011572625	0.03260304
x66	cat__Size_MEDIUM	-0.205939678	0.0461100092	0.0000079596
x67	cat__Size_MEDIUM SUV	0.0237915578	0.0860099178	0.782076421
x68	cat__Size_SMALL SUV	-0.3592296765	0.1046645727	0.0005987073
x69	cat__Size_SMALL TRUCK	-0.2842970692	0.1287905702	0.0272835442
x70	cat__Size_SPECIALTY	-0.0972946437	0.131670835	0.4599535922
x71	cat__Size_SPORTS	0.1901513431	0.1280422105	0.1375259077
x72	cat__Size_VAN	-0.3443073272	0.0772414073	0.0000082902
x73	cat__Size_missing	-1.6572649119	5843054245000000000	1
x74	cat__TopThreeAmericanName_FORD	-0.3836271801	Missing value	Missing value
x75	cat__TopThreeAmericanName_GM	-0.9601326885	18.8742173876	0.9594290583
x76	cat__TopThreeAmericanName_OTHE	-1.0855909414	21.6971966546	0.9600955364
x77	cat__TopThreeAmericanName_missi	-1.6572649119	5843054245000000000	1
x78	cat__PRIMEUNIT_YES	1.6156038086	0.4908342692	0.0009963815
x79	cat__PRIMEUNIT_missing	0.7300437671	2585027.175395142	0.9999997747
x80	cat__AUCGUART_RED	0.5030982919	0.4214414302	0.2325733758
x81	cat__AUCGUART_missing	0.7300437671	2592061.269607757	0.9999997753

Retrain Model and Create Kaggle Submission

```
# Train model against full training set and apply model to test holdout (Kaggle)
testFile = r'C:\Users\joshu\OneDrive\Desktop Files\Textbooks and Syllabi\CSUN Semester 6\MRKT 656\Case2\case2\test.csv'
testdf = pd.read_csv(testFile)

X1 = testdf.drop(columns=['RefId',
                     'PurchDate', 'VehYear',
                     'BYRNO', 'VNST',
                     'VNZIP1',  'WheelTypeID',
                     'Nationality', 'Transmission'])

model.fit(X,y)
y1Pred = model.predict(X1)

y1Preddf = pd.DataFrame(y1Pred)
testOutputdf = pd.concat([testdf['RefId'], y1Preddf], axis=1)

folder = r'C:\Users\joshu\OneDrive\Desktop Files\Textbooks and Syllabi\CSUN Semester 6\MRKT 656\Case2\case2\entryn.csv'
testOutputdf.to_csv(folder)
```

Drop same columns as performed in training

Retrain model using fill training dataset

Kaggle Results

Don't Get Kicked!

Late Submission

Overview Data Code Models Discussion Leaderboard Rules Team Submissions

All Successful Selected Errors

Recent

Submission and Description	Private Score	Public Score	Selected
entry10.csv Complete (after deadline) · 3d ago	**0.1117**	**0.10054**	

Discussion – **Agenda**

- Previous Model Iterations
- Predictive Power vs. Statistical Significance
- Important Features we did not expect

Discussion – **Previous Model Iterations**

- The different entries are the different models that had been tested during building
- The lower scoring models occurred when grouping insignificant categorical values into a base case (will be discussed in next slide)
- Entry3 is when we realized we should have trained the model against the entire dataset. We had originally left the 80/20 train test split and did not retrain the data
- The slight improvement from .1004 to .10054 was after implementing PCA

Don't Get Kicked!

Late Submission

Overview Data Code Models Discussion Leaderboard Rules Team Submissions

Submission and Description	Private Score	Public Score	Selected
entry11.csv Complete (after deadline) · 2d ago	0.1117	0.10054	☐
entry10.csv Complete (after deadline) · 5d ago	0.1117	0.10054	☐
entry9.csv Complete (after deadline) · 5d ago	0.11149	0.09929	☐
entry8.csv Complete (after deadline) · 5d ago	0.10906	0.09688	☐
entry7.csv Complete (after deadline) · 6d ago	0.10906	0.09688	☐
entry6.csv Complete (after deadline) · 6d ago	0.1117	0.10054	☐
entry5.csv Complete (after deadline) · 6d ago	0.11278	0.09971	☐
entry4.csv Complete (after deadline) · 6d ago	0.1117	0.10054	☐
entry3.csv Complete (after deadline) · 6d ago	0.11162	0.1004	☐
entry2.csv Complete (after deadline) · 6d ago	0.11276	0.09971	☐
entry2.csv Error (after deadline) · 6d ago			
example_entry0.csv Complete (after deadline) · 14d ago	-0.01126	-0.0208	☐
entry1.csv Complete (after deadline) · 14d ago	0.1132	0.10023	☐

Discussion – **Predictive Power vs. Statistical Significance**

- Grouping insignificant categorical features (*determined by Chi squared test and by looking at the p-values of the coefficients*) led to worse model performance and lower Kaggle score against holdout testing data
  - Make, Color, VNST (State), were the features attempted to be grouped.
- Because of this, we decided to keep the original cardinality of the categorical variables
- We believe that this may be because significant variance is captured by the different categorical values, and this is lost when grouping into base case
- Balancing predictive power and avoiding overfitting is quite difficult, in this case

Discussion – **Importance and Remarks**

- We initially thought the numerical features would have higher coefficients in our model, but this was not the case
- The manufacturer columns made up 10 of the top 20 features by absolute value
  - We determined that many manufacturers tend to be strongly associated with good/bad buys