Group Name: Three DataMiners Student1: Vinay Huchanahalli Nagaraju(N10180893) Student2:Kalpana Menthem(N10155694) Student3:Anudeep Gottigundala(N10155678)

In [33]:

```
#Importing required libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV
import pydot
from io import StringIO
from sklearn.tree import export_graphviz
#from dm_tools import data_prep
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from collections import Counter
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.feature_extraction.text import CountVectorizer
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import ClusterCentroids
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
```

In [4]:

```
# Preprocessing
#listing the nominal and numercial values
nominal_cols = ['Auction', 'Make','TopThreeAmericanName','Color', 'Transmission', 'Nationality', 'S
ize', 'VNST','WheelType']

num_cols = ['VehYear', 'VehBCost','VehOdo','IsOnlineSale',
'WarrantyCost','MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice',
                        'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitonRetailCleanPrice',
                        'MMRCurrentAuctionAveragePrice', 'MMRCurrentAuctionCleanPrice',
                        'MMRCurrentRetailAveragePrice', 'MMRCurrentRetailCleanPrice']
```

In [5]:

```
## Defining missing values
def fill_missing_values(df):

    for col in num_cols:
        df[col] = df[col].fillna(df[col].median())

    for col in nominal_cols:
        mode = df[col].mode()[0]
        df[col] = df[col].fillna(mode)

    return df
```

In [6]:

```
#deleting the unwanted features
def feature_engineering(df):

    del df['WheelTypeID'] #Wheeltype is used. WheeltypeID is dervided from WheelType
    del df['PurchaseID']  # Just a serial number.
    del df['ForSale']     #Just inclined to yes excluding 6 records-No contribution to the
prediction
```

```python
    del df['PurchaseDate'] #We are using Vehicle year  to measure the time series and hence,
deleting it.
    del df['MMRCurrentRetailRatio'] #Derived from MMRCurrentRetailAveragePrice and
MMRCurrentRetailCleanPrice
    del df['PRIMEUNIT'] # more than 80% of the values are undefined(?)
    del df['AUCGUART'] # more than 80% of the values are undefined(?)
    del df['PurchaseTimestamp'] # We are using year as a way of time measure, derived from
PurchaseDate. We are deleting both of them as we have year to measure.

    return df
```

In [7]:

```python
def data_type_change(df):

    #Assigning nominal values to either binary or numerical to support the datatype change
    Transmission_map = {"AUTO":0, "MANUAL": 1,'Manual':1}
    df['Transmission'] = df['Transmission'].map(Transmission_map)
    df['Transmission'].fillna(df['Transmission'].mode(),inplace=True)

    WheelType_map = {"Alloy":1, "Covers": 2, "Special": 3}
    df['WheelType'] = df['WheelType'].map(WheelType_map)

    Auction_map={'ADESA':0,'MANHEIM':1,'OTHER':2}
    df['Auction']=df['Auction'].map(Auction_map)


    Make_map={'ACURA':0,'BUICK':1,'CADILLAC':3,'CHEVROLET':4,'CHRYSLER':5,'DODGE':6,'FORD':7,'GMC':
8,'HONDA':9,'HYUNDAI':10,'INFINITI':11,'ISUZU':12,'JEEP':13,'KIA':14,'LEXUS':15,'LINCOLN':16,'MAZDA
':17,'MERCURY':18,'MINI':19,'MITSUBISHI':20,'NISSAN':21,'OLDSMOBILE':22,'PONTIAC':23,'SATURN':24,'S
CION':25,'SUBARU':26,'SUZUKI':27,'TOYOTA':2,'VOLKSWAGEN':28,'VOLVO':29}
    df['Make']=df['Make'].map(Make_map)
    #df['Make'].fillna(df['Make'].mode(),inplace=True)


    american_name_map={'CHRYSLER':0,'FORD':1,'GM':2,'OTHER':3}
    df['TopThreeAmericanName']=df['TopThreeAmericanName'].map(american_name_map)

    Color_map={'BEIGE':0,'BLACK':1,'BLUE':2,'BROWN':3,'GOLD':4,'GREEN':5,'GREY':6,'MAROON':7,'NOT A
VAIL':8,'ORANGE':9,'OTHER':10,'PURPLE':11,'RED':12,'SILVER':13,'WHITE':14,'YELLOW':15}
    df['Color']=df['Color'].map(Color_map)

    Nationality_map={'AMERICAN':0,'OTHER':1,'OTHER ASIAN':2,'TOP LINE ASIAN':3,'USA':4}
    df['Nationality']=df['Nationality'].map(Nationality_map)

    Size_map={'COMPACT':0,'CROSSOVER':1,'LARGE':2,'LARGE SUV':3,'LARGE TRUCK':4,'MEDIUM':5,'MEDIUM
SUV':6,'SMALL SUV':7,'SMALL TRUCK':8,'SPECIALTY':9,'SPORTS':10,'VAN':11}
    df['Size']=df['Size'].map(Size_map)

    vnst_map = {'TX':0, 'FL':1,'CO':2,'NC':3,'AZ':4,'CA':5,'OK':6,'SC':7,'TN':8,'GA':9,'VA':10,'MO'
:11,'PA':12,'NV':13,'IN':14,'MS':15,'LA':16,'NJ':17,'NM':18,'KY':19,'AL':20,'IL':21,'UT':22,'WV':2
3,'WA':24,'OR':25,'NH':26,'NE':27,'OH':28,'ID':29,'NY':30}
    df['VNST'] = df['VNST'].map(vnst_map)

    # changing datatypes as required
    df['Transmission'] = df['Transmission'].astype(float)
    df['Auction'] = df['Auction'].astype(float)
    df['Make'] = df['Make'].astype(float)
    df['TopThreeAmericanName'] = df['TopThreeAmericanName'].astype(float)
    df['Nationality'] = df['Nationality'].astype(float)
    df['Size'] = df['Size'].astype(float)
    df['VNST'] = df['VNST'].astype(float)
    df['VehBCost'] = df['VehBCost'].astype(float)
    df['WheelType'] = df['WheelType'].astype(int)
    df['IsOnlineSale'] = df['IsOnlineSale'].astype(float)
    df['MMRAcquisitionAuctionAveragePrice'] = df['MMRAcquisitionAuctionAveragePrice'].astype(float)
    df['MMRAcquisitionAuctionCleanPrice'] = df['MMRAcquisitionAuctionCleanPrice'].astype(float)
    df['MMRAcquisitionRetailAveragePrice'] = df['MMRAcquisitionRetailAveragePrice'].astype(float)
    df['MMRAcquisitonRetailCleanPrice'] = df['MMRAcquisitonRetailCleanPrice'].astype(float)
    df['MMRCurrentAuctionAveragePrice'] = df['MMRCurrentAuctionAveragePrice'].astype(float)
    df['MMRCurrentAuctionCleanPrice'] = df['MMRCurrentAuctionCleanPrice'].astype(float)
    df['MMRCurrentRetailAveragePrice'] = df['MMRCurrentRetailAveragePrice'].astype(float)
    df['MMRCurrentRetailCleanPrice'] = df['MMRCurrentRetailCleanPrice'].astype(float)
```

```
    return df
```

In [8]:

```
# IsOnlineSale is a binary varibale which accepts wither zero or one. So we are replacing other va
lues wiht nan, which will be rpelaced by the median
def error_replacing_For_IsOnlineSale(df):
    mask = df['IsOnlineSale'] == -1
    df.loc[mask, 'IsOnlineSale'] = np.nan
    mask = df['IsOnlineSale'] == 2
    df.loc[mask, 'IsOnlineSale'] = np.nan
    mask = df['IsOnlineSale'] == 4
    df.loc[mask, 'IsOnlineSale'] = np.nan

    return df
```

In [9]:

```
# Converting nominal cols to one-hot vectors
def convert_nominal_cols(df):
    global nominal_cols
    df_with_dummies = pd.get_dummies(df, columns = nominal_cols)

    return df_with_dummies
```

In [10]:

```
def analyse_feature_importance(dm_model, feature_names, n_to_display=20):
    # grab feature importances from the model
    importances = dm_model.feature_importances_

    # sort them out in descending order
    indices = np.argsort(importances)
    indices = np.flip(indices, axis=0)

    # limit to 20 features, you can leave this out to print out everything
    indices = indices[:n_to_display]

    for i in indices:
       print(feature_names[i], ':', importances[i])

def visualize_decision_tree(dm_model, feature_names, save_name):
    #Visualise the model using three parameters
    import pydot
    from io import StringIO
    from sklearn.tree import export_graphviz

    dotfile = StringIO()
    export_graphviz(dm_model, out_file=dotfile, feature_names=feature_names)
    graph = pydot.graph_from_dot_data(dotfile.getvalue())
    graph[0].write_png(save_name) # saved in the following file
```

In [13]:

```
def preprocessing():

    df = pd.read_csv('CaseStudyData.csv')
    #droping the columns with continuosly 10 null values. We have 44 records with blank values for
26 variables(or columns)
    new1_df=df.dropna(axis=0,thresh=10)
    #Rerplacing ? with null values
    new_df = new1_df.replace(['?'], np.nan, inplace=False)
    error_replacing_For_IsOnlineSale(new_df)
    fill_missing_values(new_df)
    feature_engineering(new_df)
    convert_nominal_cols(new_df)
    return new_df
```

In [14]:

```
df = pd.read_csv('CaseStudyData.csv')
df = preprocessing()
```

```
df2 = data_type_change(df)

print(df2.info())
Y = df2['IsBadBuy']
X = df2.drop(['IsBadBuy'], axis=1)
rs=10
#Split the data based on training and testing with 70 and 30%
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, stratify=Y, random_state=r
s)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3020: DtypeWarning:
Columns (27) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3185: DtypeWarning:
Columns (27) have mixed types. Specify dtype option on import or set low_memory=False.
  if (yield from self.run_code(code, result)):
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41432 entries, 0 to 41475
Data columns (total 23 columns):
Auction                              41432 non-null float64
VehYear                              41432 non-null float64
Make                                 41432 non-null float64
Color                                41432 non-null int64
Transmission                         41432 non-null float64
WheelType                            41432 non-null int32
VehOdo                               41432 non-null float64
Nationality                          41432 non-null float64
Size                                 41432 non-null float64
TopThreeAmericanName                 41432 non-null float64
MMRAcquisitionAuctionAveragePrice    41432 non-null float64
MMRAcquisitionAuctionCleanPrice      41432 non-null float64
MMRAcquisitionRetailAveragePrice     41432 non-null float64
MMRAcquisitonRetailCleanPrice        41432 non-null float64
MMRCurrentAuctionAveragePrice        41432 non-null float64
MMRCurrentAuctionCleanPrice          41432 non-null float64
MMRCurrentRetailAveragePrice         41432 non-null float64
MMRCurrentRetailCleanPrice           41432 non-null float64
VNST                                 41432 non-null float64
VehBCost                             41432 non-null float64
IsOnlineSale                         41432 non-null float64
WarrantyCost                         41432 non-null float64
IsBadBuy                             41432 non-null int64
dtypes: float64(20), int32(1), int64(2)
memory usage: 7.4 MB
None
```

Task 1. Data Selection and Distribution.

In [15]:

```
print("Proportion of cars who can be classified as a kick : ",df.IsBadBuy.value_counts()[1]/(df.Is
BadBuy.value_counts()[0]+df.IsBadBuy.value_counts()[1]))
```

```
Proportion of cars who can be classified as a kick :  0.12948928364549142
```

1. What is the proportion of cars who can be classified as a “kick”? Answer: Nearly 13% of the cars can be classified as Kick by the given data of IsBadBuy. IsBadBuy=1 suggests that it is a Kick.
2. Did you have to fix any data quality problems? Detail them Answer: We have missing values, noise and errorneous values and incorrect format for some of the features in the given data.These are the data quality problems we encountered. Detailed description is given below.

```
 i) We have 44 records with blank values for 26 features continuosly, which of no use to
predict a car is a Kick or not..
 ii) We have derived features from other fetures(Eg.WheeltypeID is dervided from WheelTy
pe,MMRCurrentRetailRatio    derived from MMRCurrentRetailAveragePrice and
MMRCurrentRetailCleanPrice and PurchaseTimestamp is derived from PurchaseDate).
 iii) PurchaseID is just a serial number which is not contributing to predict the target
.
```

```
 iv) PRIMEUNIT and AUCGUART are having undefined values as '?' for more than 80% of the
records.
 v) ForSale has just 6 records with value "No", remainig all 'Yes' amongst 41476 records
, which is again not useful to predict the target.
 vi) PurchaseDate is no where needed as we have VehYear, which is a measure of time to p
redict the target.
 vii) PurchaseTimestamp is derived from PurchaseDate, and we have VehYear to measure the
time. So no need of PurchaseTimestamp to predict a car is Kick or not.
 Viii) Most of the features have skewness as shown below using boxplot and Histograms fo
r all the variables.
 ix) Some of the records of fetures have undefined value which is a '?'(errorneous value
s) and also have blank values.
```

3. Can you identify any clear patterns by initial exploration of the data using histogram or box plot?

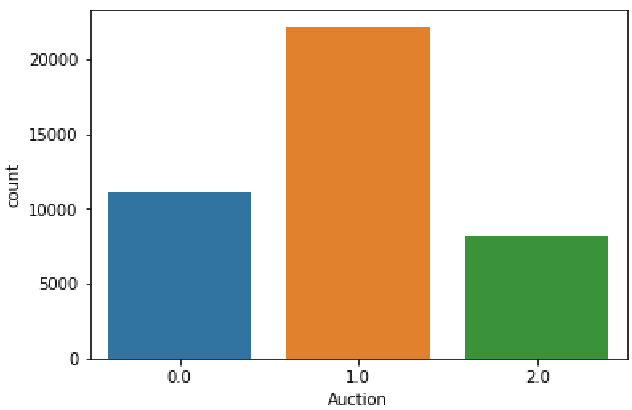
   We have plot the hostograms using distplot for numerical/categorical values and countplot for nominal values.

   Auction "MANHEIM" has more distribution compared to the other auction company names i.e. There are more chances of a kick cars of the from the MANHEIM company.CHEVROLET cars,GM cars, Silver colours cars,Auto tranmission cars,American nationality cars, Alloy wheel type cars have the highest count, which are kick cars from the histograms. We have skeness for the price variables which means the data is distibuted for particular range of price.

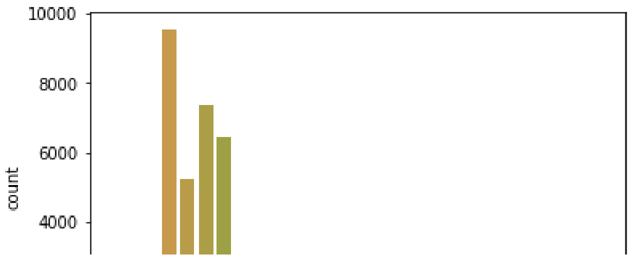
In [16]:

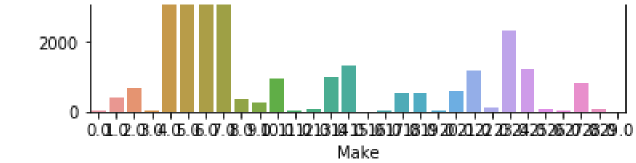
```
# Histogram  (Univariate Analysis)
print('Histogram plot for Nominal columns')
for col in nominal_cols:
    dg = sns.countplot(df[col])
    plt.show()
    print('-'*70)

print('+='*100)

print('Histogram plot for numerical columns')

print('-'*50)
for col in num_cols:
    dg = sns.distplot(df[col])
    plt.show()
    print('-'*70)
```

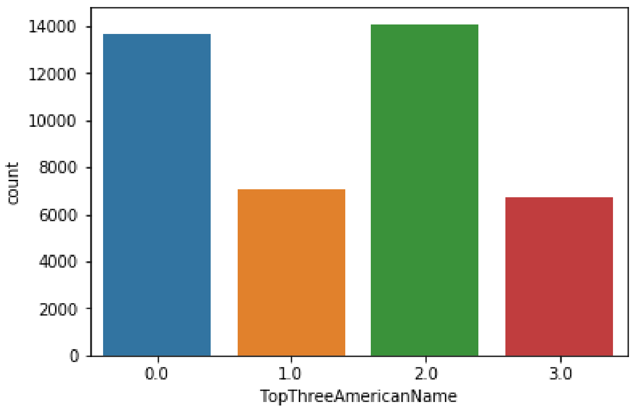
```
Histogram plot for Nominal columns
```

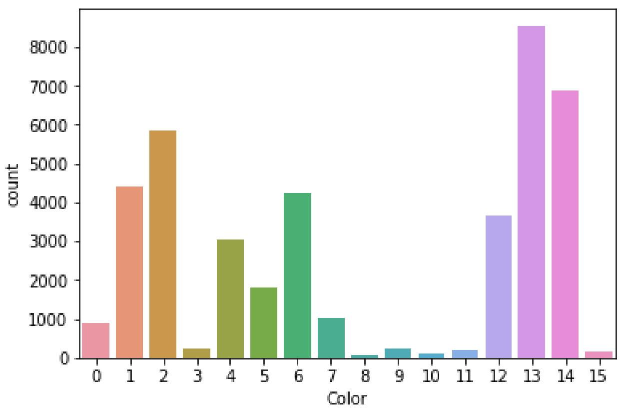


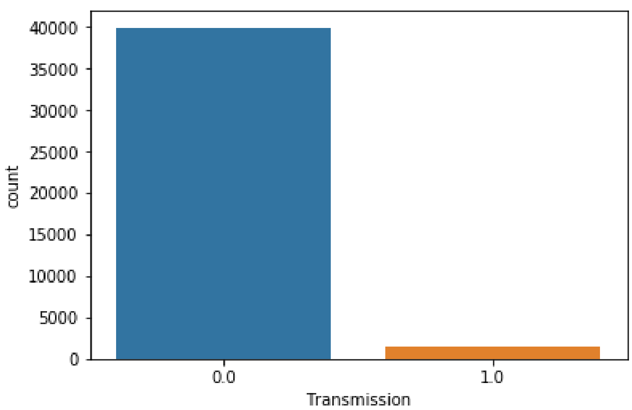

```
----------------------------------------------------------------------
```



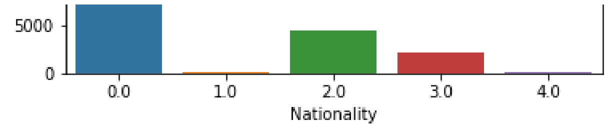

2000
0
Make


14000
12000
10000
8000
6000
4000
2000
0
count
0.0
1.0
2.0
3.0
TopThreeAmericanName

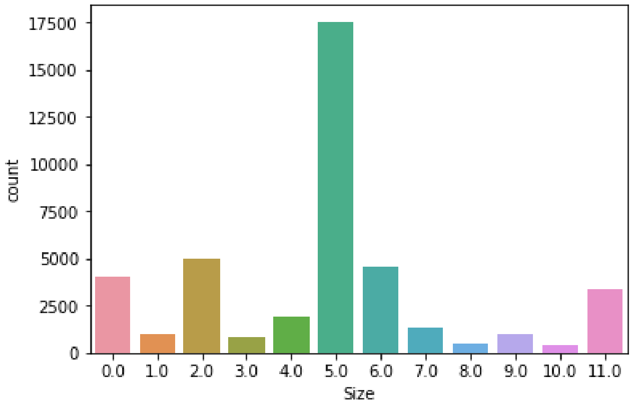

8000
7000
6000
5000
4000
3000
2000
1000
0
count
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Color


40000
35000
30000
25000
20000
15000
10000
5000
0
count
0.0
1.0
Transmission

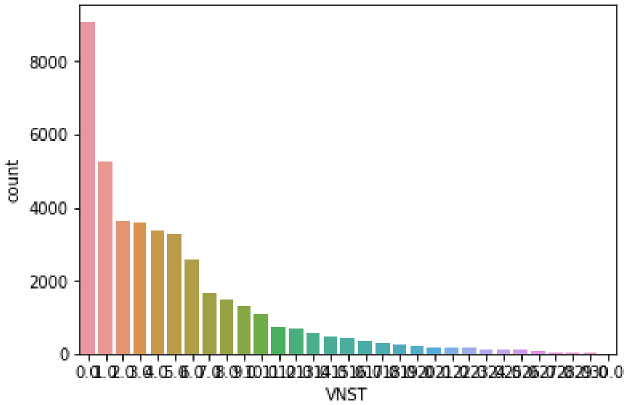

35000
30000
25000
20000
15000
10000
count



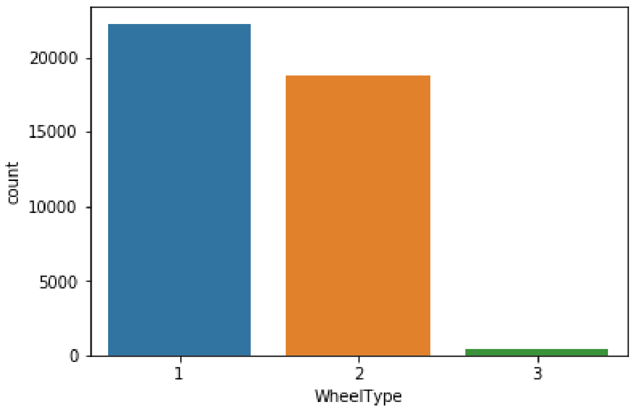

```
----------------------------------------------------------------------
```




```
----------------------------------------------------------------------
```

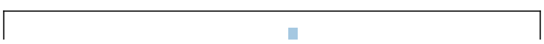


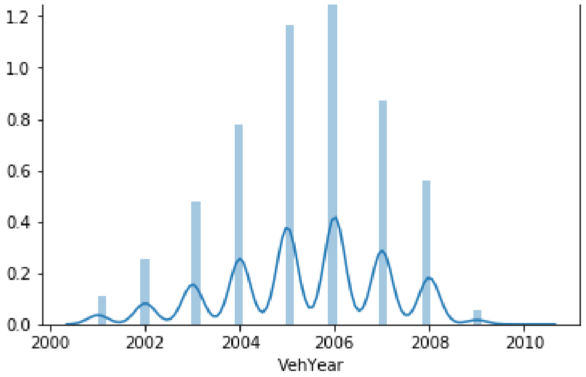

```
----------------------------------------------------------------------
```

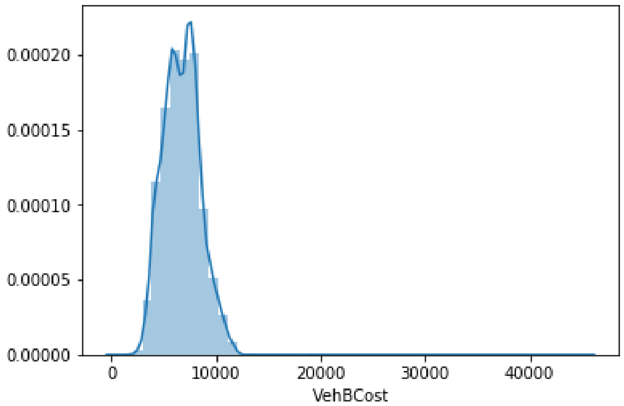


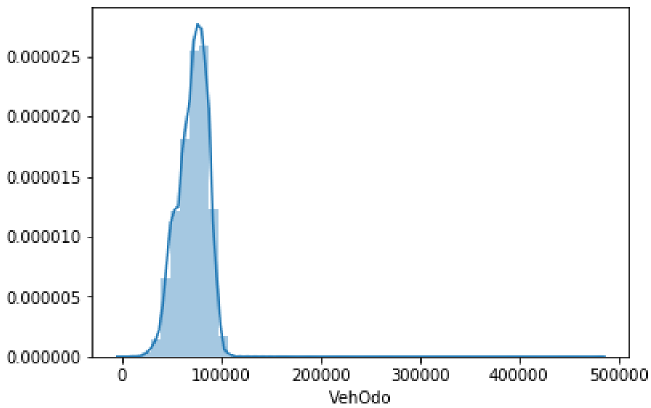

```
----------------------------------------------------------------------
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
Histogram plot for numerical columns
--------------------------------------------------
```

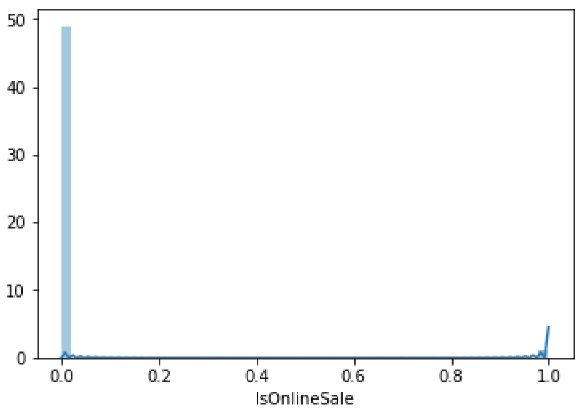
```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-t
uple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[s
eq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will r
esult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



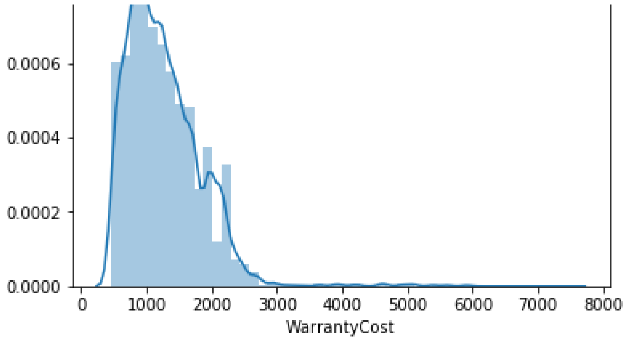

1.2
1.0
0.8
0.6
0.4
0.2
0.0
2000
2002
2004
2006
2008
2010
VehYear

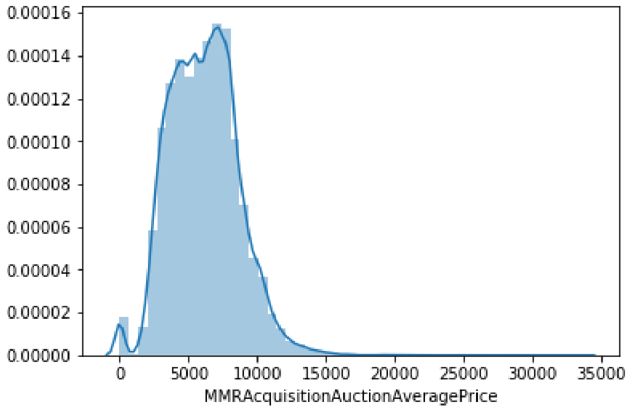

0.00020
0.00015
0.00010
0.00005
0.00000
0
10000
20000
30000
40000
VehBCost

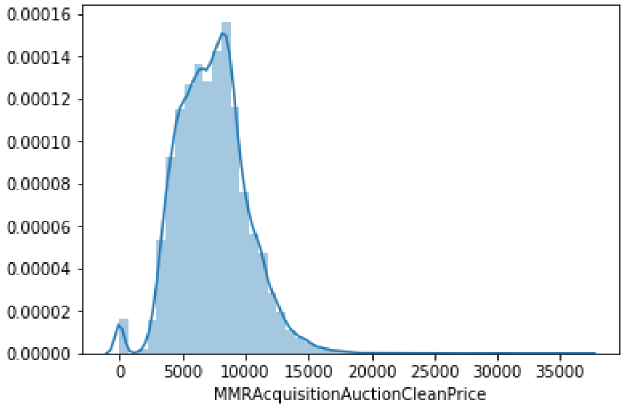

0.000025
0.000020
0.000015
0.000010
0.000005
0.000000
0
100000
200000
300000
400000
500000
VehOdo

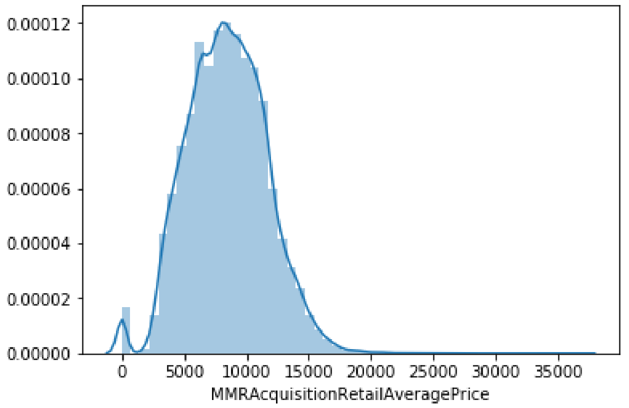

50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
IsOnlineSale

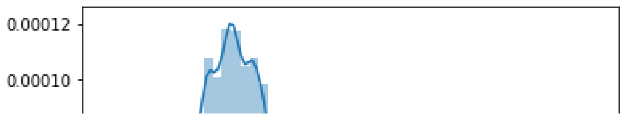

0.0008

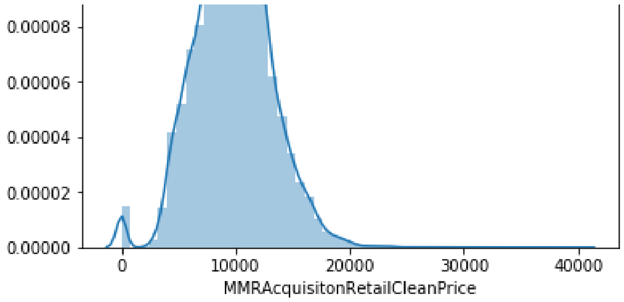

WarrantyCost

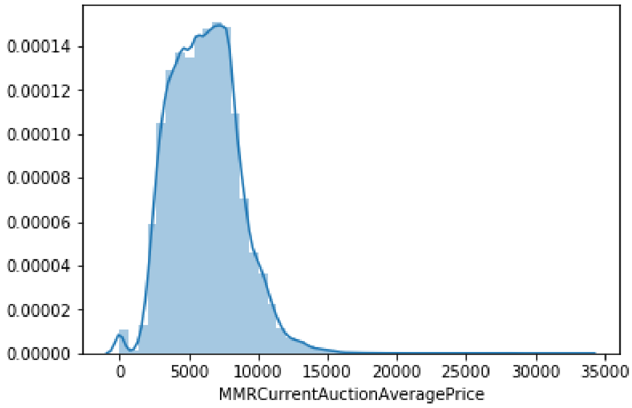

MMRAcquisitionAuctionAveragePrice

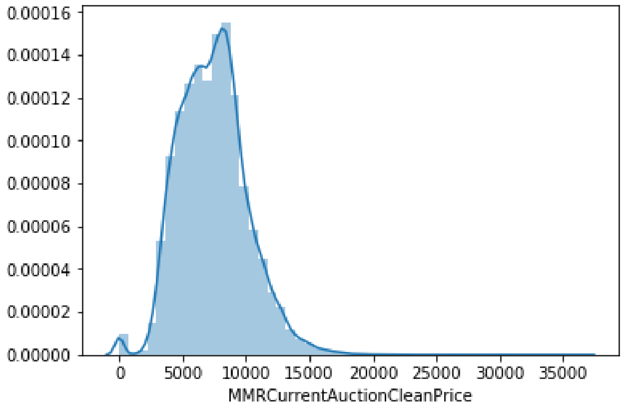

MMRAcquisitionAuctionCleanPrice

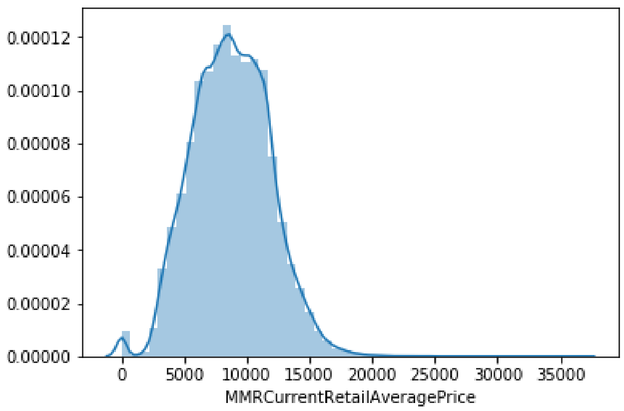

MMRAcquisitionRetailAveragePrice

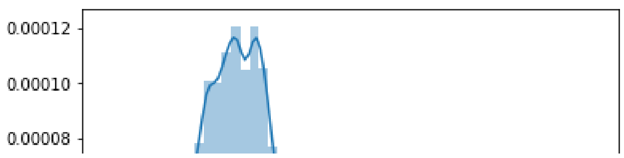


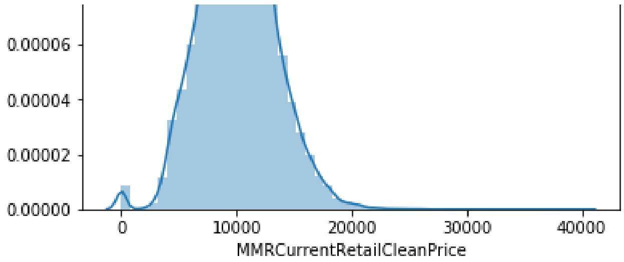

MMRAcquisitonRetailCleanPrice


MMRCurrentAuctionAveragePrice


MMRCurrentAuctionCleanPrice


MMRCurrentRetailAveragePrice

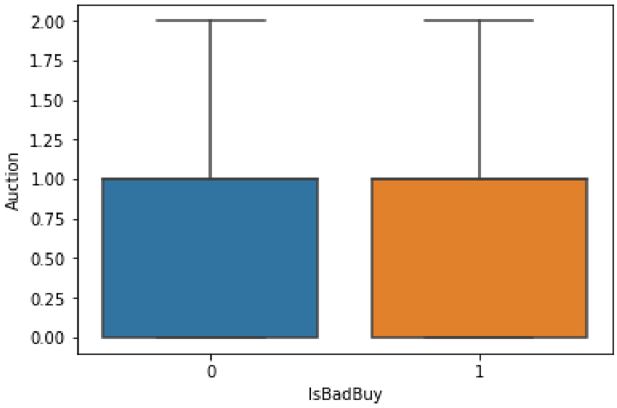


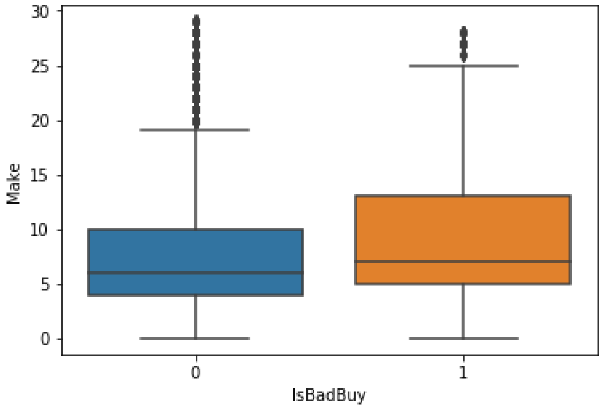


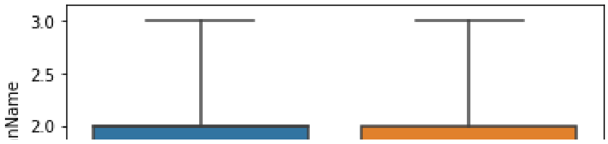
In [19]:

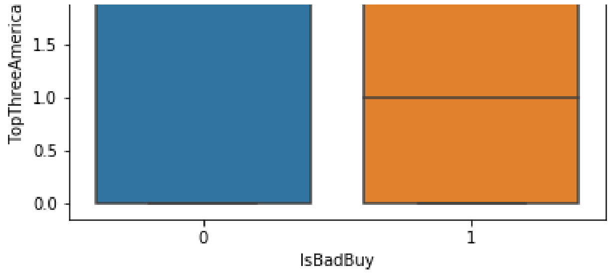
```
# BoxPlot  (Univariate Analysis)

print('Box Plot for nominal columns')
for col in nominal_cols:
    ax = sns.boxplot(x="IsBadBuy", y=col, data=df)
    plt.show()
    print('--------------------------------')

print('BoxPlot for numerical columns')

print('-------------------------------------')
for col in num_cols:
    ax = sns.boxplot(x="IsBadBuy", y=col, data=df)
    print('-'*70)
```

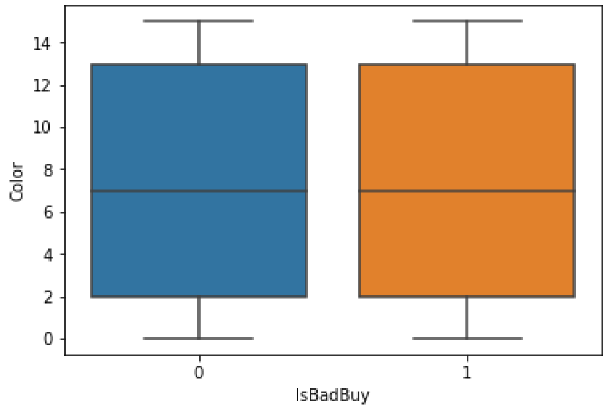
Box Plot for nominal columns

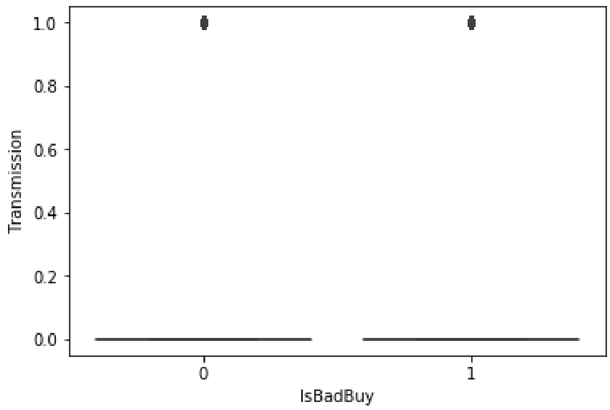


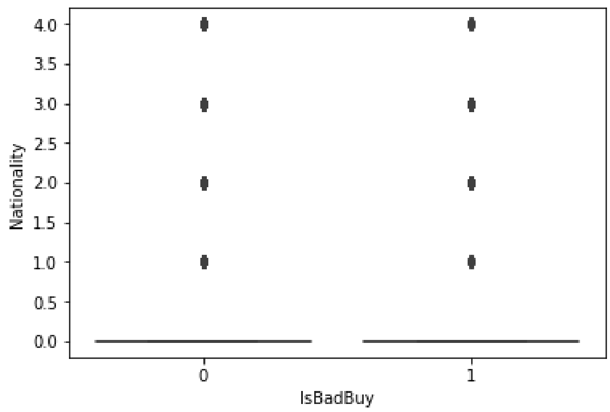


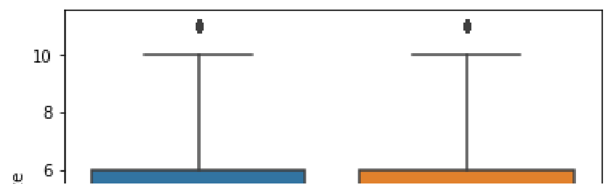


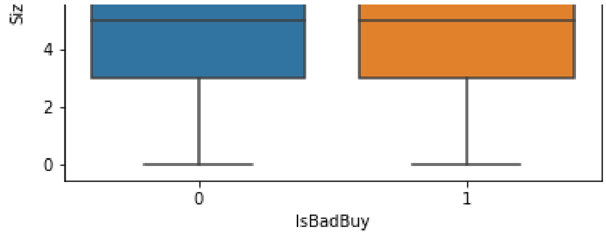

TopThreeAmerica
1.5
1.0
0.5
0.0
0
1
IsBadBuy


Color
14
12
10
8
6
4
2
0
0
1
IsBadBuy

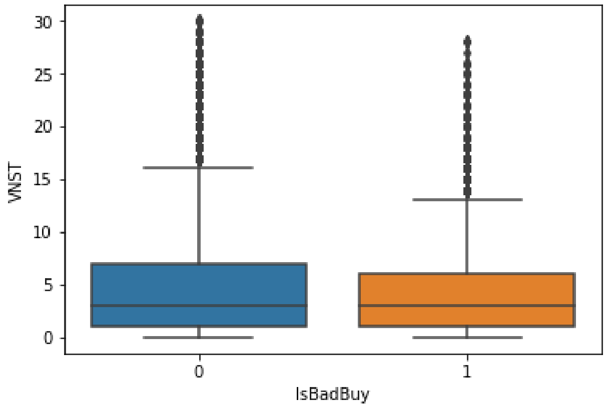

Transmission
1.0
0.8
0.6
0.4
0.2
0.0
0
1
IsBadBuy


Nationality
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
IsBadBuy

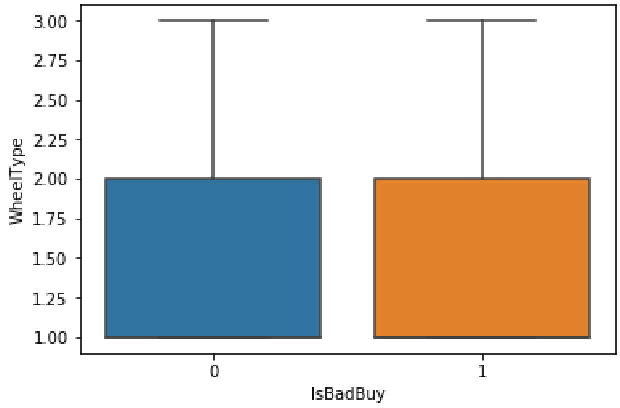

10
8
6



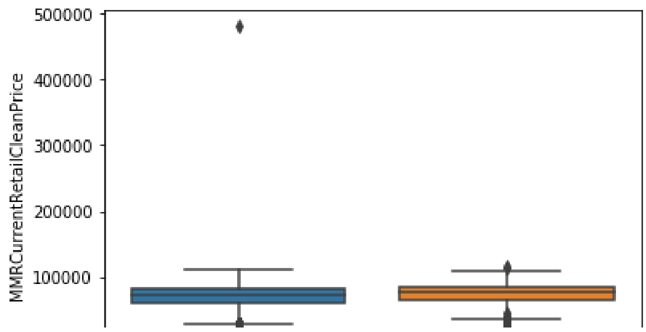

```
--------------------------------
```




```
--------------------------------
```




```
--------------------------------
BoxPlot for numerical columns
-------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
```



1. What variables did you include in the analysis and what were their roles and measurement level set? Justify your choice

   We are including all the variables except WheelTypeID,PurchaseID,ForSale,PurchaseDate,MMRCurrentRetailRatio,PRIMEUNIT,AUCGUART and PurchaseTimestamp.

   Role: Auction will give the company name which can give kick cars and non-kick cars. From vehicle year, we can predict which year making car is a kick or not. Vehicle make will give the which type of make gives the kick cars and non-kick cars. From Color, we can predict the kick cars based on the color of the car. Transmission gives either Auto cars or manual cars are kicks. WheelType, based on the wheeltype of a car , can distinguish the kick cars from non-kick cars. VehOdo will differntiate the cars based on the kilometers it has run already. Natioanlity also gives the some prediction for the kick cars. Size can also predict the kick cars based on the size of the vehicle.TopThreeAmericanName, can distinguish the kick cars from non-kick cars as it will give the trust. Based on the prices also we can predict the kick cars, as the prices are trustable or not and which price is suitable based on the car. Geograpihc region can affect the kick cars.VehBCost, can predict the kick cars based on the price of the car at the puchase time. IsOnlineSale, says weather the car is available online or not. Warranty cost also affects the kick car.

   Measurement Level Set: Below command gives the measurement level for the variables which we included. For example, Transmission=0 means Auto and Transmission=1 means Manual. Auto cars slightly are the kick cars based on the measurement level given below.

1. What distribution scheme did you use? What data partitioning allocation did you set? Explain your selection. Answer: We are dividing the data into training and test sets. First we will train the model using train data and then we will test it using the test data. This is the distribution scheme which has been used here. We have used the 70/30 partition allocation here. 70% of the data has been used for training and 30% has been used for testing. Our data is not a big data so no need of validation data set. That is why we are just using trainig and testing data sets in our distribution scheme. We have taken 70/30 as partitioning allocation as it is a common criteria. We are also using stratify while performing the split to ensure the same ratio of positive and negative targets in both trainig and testing datasets. We can see the command for this below.

In [20]:

```
# Measurement Level Set  (Univariate Analysis)

print('Measurement level for nominal columns')
for col in nominal_cols:
    print(df.groupby(['IsBadBuy'])[col].value_counts(normalize=True))
    print('--------------------------------')

print('Measurement level for numerical columns')

print('-------------------------------------')
for col in num_cols:
    print(df.groupby(['IsBadBuy'])[col].value_counts(normalize=True))
    print('-'*70)
```

```
Measurement level for nominal columns
IsBadBuy  Auction
0         1.0        0.541825
          0.0        0.258657
          2.0        0.199518
1         1.0        0.489469
          0.0        0.327493
          2.0        0.183038
Name: Auction, dtype: float64
--------------------------------
IsBadBuy  Make
0         4.0     0.238085
          6.0     0.182078
          7.0     0.149361
          5.0     0.126459
          23.0    0.057476
          14.0    0.032329
          24.0    0.029390
          21.0    0.027782
          13.0    0.022874
          10.0    0.022763
          27.0    0.019907
          2.0     0.016414
```

```
          20.0    0.013835
          17.0    0.012505
          18.0    0.011728
          1.0     0.009732
          8.0     0.008595
          9.0     0.006294
          22.0    0.003189
          12.0    0.002079
          25.0    0.001941
          28.0    0.001691
          16.0    0.001026
          11.0    0.000471
          3.0     0.000388
          26.0    0.000388
          0.0     0.000333
          29.0    0.000333
          19.0    0.000305
          15.0    0.000250
1         7.0     0.199627
          4.0     0.179124
          6.0     0.152470
          5.0     0.130103
          23.0    0.052563
          24.0    0.034483
          21.0    0.034296
          14.0    0.031873
          13.0    0.029823
          10.0    0.025349
          27.0    0.023113
          18.0    0.019385
          17.0    0.015098
          2.0     0.013420
          20.0    0.013048
          1.0     0.011556
          8.0     0.007642
          9.0     0.006710
          22.0    0.005778
          16.0    0.003169
          28.0    0.002237
          11.0    0.001864
          19.0    0.001491
          0.0     0.001305
          12.0    0.001305
          25.0    0.001305
          15.0    0.000746
          3.0     0.000559
          26.0    0.000559
Name: Make, dtype: float64
--------------------------------
IsBadBuy  TopThreeAmericanName
0         2.0                     0.346938
          0.0                     0.331356
          1.0                     0.162115
          3.0                     0.159592
1         0.0                     0.312395
          2.0                     0.291705
          1.0                     0.222181
          3.0                     0.173719
Name: TopThreeAmericanName, dtype: float64
--------------------------------
IsBadBuy  Color
0         13       0.205617
          14       0.166607
          2        0.142318
          1        0.107827
          6        0.103446
          12       0.087864
          4        0.072144
          5        0.042865
          7        0.025148
          0        0.021405
          9        0.006377
          3        0.005822
          11       0.004686
          10       0.003327
          15       0.003299
          8        0.001248
1         13       0.210811
```

```
1          13       0.210811
           14       0.164212
           2        0.134576
           6        0.096365
           1        0.093756
           12       0.091705
           4        0.085182
           5        0.046598
           7        0.024604
           0        0.022740
           3        0.007269
           11       0.006710
           9        0.004660
           15       0.004101
           8        0.003728
           10       0.002982
Name: Color, dtype: float64
--------------------------------
IsBadBuy  Transmission
0          0.0             0.963512
           1.0             0.036488
1          0.0             0.966449
           1.0             0.033551
Name: Transmission, dtype: float64
--------------------------------
IsBadBuy  Nationality
0          0.0            0.837469
           2.0            0.106857
           3.0            0.050406
           4.0            0.002939
           1.0            0.002329
1          0.0            0.822740
           2.0            0.115564
           3.0            0.054427
           1.0            0.003728
           4.0            0.003541
Name: Nationality, dtype: float64
--------------------------------
IsBadBuy  Size
0          5.0     0.427233
           2.0     0.125405
           6.0     0.106552
           0.0     0.093132
           11.0    0.081044
           4.0     0.046081
           7.0     0.031636
           9.0     0.024870
           1.0     0.023845
           3.0     0.018965
           8.0     0.011617
           10.0    0.009621
1          5.0     0.397763
           6.0     0.135322
           0.0     0.126002
           2.0     0.082945
           11.0    0.082759
           4.0     0.043802
           7.0     0.035601
           3.0     0.027213
           1.0     0.021249
           9.0     0.018826
           10.0    0.014539
           8.0     0.013979
Name: Size, dtype: float64
--------------------------------
IsBadBuy  VNST
0          0.0     0.216819
           1.0     0.127956
           2.0     0.088197
           3.0     0.086644
           4.0     0.080572
           5.0     0.076219
           6.0     0.064713
           7.0     0.041090
           8.0     0.035878
           9.0     0.031441
           10.0    0.026090
```

```
           11.0    0.018743
           12.0    0.015887
           13.0    0.012837
           14.0    0.011590
           15.0    0.010259
           16.0    0.008235
           17.0    0.007680
           19.0    0.005961
           18.0    0.005822
           20.0    0.004464
           22.0    0.004270
           21.0    0.004020
           24.0    0.003549
           25.0    0.003438
           23.0    0.003355
           26.0    0.002440
           27.0    0.000693
           28.0    0.000582
           29.0    0.000388
           30.0    0.000166
1          0.0     0.234110
           1.0     0.118360
           5.0     0.096738
           4.0     0.088910
           3.0     0.087418
           2.0     0.082386
           6.0     0.048649
           7.0     0.033551
           8.0     0.032992
           9.0     0.028518
           10.0    0.028332
           12.0    0.023672
           13.0    0.016775
           11.0    0.015284
           14.0    0.012675
           16.0    0.009692
           15.0    0.007829
           17.0    0.007456
           18.0    0.005405
           21.0    0.003728
           20.0    0.003355
           23.0    0.002982
           19.0    0.002796
           25.0    0.002237
           22.0    0.002050
           26.0    0.001678
           24.0    0.001491
           28.0    0.000746
           27.0    0.000186
Name: VNST, dtype: float64
-------------------------------
IsBadBuy  WheelType
0         1            0.512518
          2            0.477001
          3            0.010480
1         1            0.698975
          2            0.290214
          3            0.010811
Name: WheelType, dtype: float64
-------------------------------
Measurement level for numerical columns
------------------------------------
IsBadBuy  VehYear
0         2006.0     0.240136
          2005.0     0.207447
          2007.0     0.166662
          2004.0     0.133696
          2008.0     0.108964
          2003.0     0.078049
          2002.0     0.038900
          2001.0     0.015915
          2009.0     0.010203
          2010.0     0.000028
1         2005.0     0.223672
          2004.0     0.180801
          2006.0     0.180615
          2003.0     0.137745
```

```
          2007.0     0.093756
          2002.0     0.088723
          2008.0     0.046039
          2001.0     0.045107
          2009.0     0.003541
Name: VehYear, dtype: float64
----------------------------------------------------------------------
IsBadBuy  VehBCost
0         7500.0      0.011811
          6500.0      0.007209
          7800.0      0.006654
          7000.0      0.006488
          7200.0      0.006294
          6000.0      0.006183
          8000.0      0.006155
          7100.0      0.005878
          6300.0      0.005795
          7400.0      0.005268
          6400.0      0.005018
          4200.0      0.004963
          6100.0      0.004824
          7700.0      0.004381
          7300.0      0.004298
          8200.0      0.004131
          5500.0      0.004103
          5000.0      0.003882
          6700.0      0.003715
          6600.0      0.003660
          6800.0      0.003660
          6200.0      0.003632
          7600.0      0.003604
          8100.0      0.003577
          5800.0      0.003410
          7900.0      0.003355
          5700.0      0.003327
          6900.0      0.003133
          4175.0      0.003078
          5900.0      0.002828
                        ...
1         11385.0     0.000186
          11425.0     0.000186
          11430.0     0.000186
          11445.0     0.000186
          11495.0     0.000186
          11500.0     0.000186
          11505.0     0.000186
          11527.0     0.000186
          11595.0     0.000186
          11600.0     0.000186
          11620.0     0.000186
          11645.0     0.000186
          11705.0     0.000186
          11760.0     0.000186
          11785.0     0.000186
          11845.0     0.000186
          11900.0     0.000186
          12025.0     0.000186
          12090.0     0.000186
          12330.0     0.000186
          12590.0     0.000186
          13535.0     0.000186
          18245.0     0.000186
          19000.0     0.000186
          20100.0     0.000186
          28180.0     0.000186
          29795.0     0.000186
          32300.0     0.000186
          38785.0     0.000186
          45469.0     0.000186
Name: VehBCost, Length: 3237, dtype: float64
----------------------------------------------------------------------
IsBadBuy  VehOdo
0         50902.0     0.000166
          67756.0     0.000166
          67860.0     0.000166
          71225.0     0.000166
          74671.0     0.000166
```

```
          76267.0     0.000166
          79600.0     0.000166
          84675.0     0.000166
          59355.0     0.000139
          62143.0     0.000139
          62277.0     0.000139
          63053.0     0.000139
          67138.0     0.000139
          67622.0     0.000139
          67625.0     0.000139
          67953.0     0.000139
          69089.0     0.000139
          69413.0     0.000139
          70269.0     0.000139
          70334.0     0.000139
          71005.0     0.000139
          71783.0     0.000139
          72101.0     0.000139
          73154.0     0.000139
          73232.0     0.000139
          74538.0     0.000139
          74783.0     0.000139
          75007.0     0.000139
          75064.0     0.000139
          75309.0     0.000139
                        ...
1         103531.0    0.000186
          103575.0    0.000186
          103675.0    0.000186
          103806.0    0.000186
          103834.0    0.000186
          103929.0    0.000186
          104125.0    0.000186
          104716.0    0.000186
          104957.0    0.000186
          105313.0    0.000186
          105536.0    0.000186
          105776.0    0.000186
          105989.0    0.000186
          106225.0    0.000186
          106774.0    0.000186
          106885.0    0.000186
          107091.0    0.000186
          107383.0    0.000186
          107741.0    0.000186
          107860.0    0.000186
          108275.0    0.000186
          108486.0    0.000186
          108825.0    0.000186
          109260.0    0.000186
          109348.0    0.000186
          109549.0    0.000186
          109728.0    0.000186
          109848.0    0.000186
          114184.0    0.000186
          115717.0    0.000186
Name: VehOdo, Length: 31051, dtype: float64
-----------------------------------------------------------------------
IsBadBuy  IsOnlineSale
0         0.0             0.978041
          1.0             0.021959
1         0.0             0.982293
          1.0             0.017707
Name: IsOnlineSale, dtype: float64
-----------------------------------------------------------------------
IsBadBuy  WarrantyCost
0         920.0           0.041257
          1974.0          0.034408
          2152.0          0.030804
          1215.0          0.029279
          1389.0          0.028807
          1155.0          0.025647
          728.0           0.023373
          803.0           0.022514
          1086.0          0.021377
          1503.0          0.020850
          1703.0          0.020323
```

```
          1243.0          0.020240
          569.0           0.019907
          1020.0          0.018882
          983.0           0.018188
          834.0           0.017578
          1272.0          0.017301
          533.0           0.016913
          1623.0          0.016802
          754.0           0.015887
          853.0           0.015721
          1763.0          0.015582
          671.0           0.015222
          505.0           0.015111
          825.0           0.015055
          1373.0          0.014917
          693.0           0.014279
          1506.0          0.014279
          975.0           0.014113
          1633.0          0.013974
                            ...
1         1137.0          0.000186
          1181.0          0.000186
          1275.0          0.000186
          1301.0          0.000186
          1418.0          0.000186
          1487.0          0.000186
          1557.0          0.000186
          1571.0          0.000186
          1590.0          0.000186
          1610.0          0.000186
          1634.0          0.000186
          1931.0          0.000186
          1944.0          0.000186
          2090.0          0.000186
          2101.0          0.000186
          2141.0          0.000186
          2251.0          0.000186
          2441.0          0.000186
          2499.0          0.000186
          2700.0          0.000186
          2711.0          0.000186
          2799.0          0.000186
          2838.0          0.000186
          2891.0          0.000186
          2976.0          0.000186
          3115.0          0.000186
          3222.0          0.000186
          3298.0          0.000186
          3667.0          0.000186
          6208.0          0.000186
Name: WarrantyCost, Length: 503, dtype: float64
----------------------------------------------------------------------
IsBadBuy  MMRAcquisitionAuctionAveragePrice
0         0.0                                   0.012144
          5480.0                                0.005545
          6311.0                                0.002551
          5569.0                                0.001941
          7644.0                                0.001941
          7991.0                                0.001941
          7811.0                                0.001858
          4573.0                                0.001636
          7245.0                                0.001636
          6858.0                                0.001469
          8196.0                                0.001469
          6892.0                                0.001414
          7048.0                                0.001331
          7960.0                                0.001331
          7293.0                                0.001275
          5427.0                                0.001248
          7513.0                                0.001248
          3688.0                                0.001192
          6820.0                                0.001164
          7541.0                                0.001137
          7314.0                                0.001109
          7533.0                                0.001109
          8268.0                                0.001109
          8194.0                                0.001081
```

```
          6733.0                                     0.001054
          5500.0                                     0.001026
          6867.0                                     0.000998
          8012.0                                     0.000998
          6948.0                                     0.000943
          7171.0                                     0.000943
                                                       ...
1         14804.0                                    0.000186
          15091.0                                    0.000186
          15313.0                                    0.000186
          15314.0                                    0.000186
          15585.0                                    0.000186
          15613.0                                    0.000186
          15852.0                                    0.000186
          16412.0                                    0.000186
          16536.0                                    0.000186
          16982.0                                    0.000186
          17229.0                                    0.000186
          17246.0                                    0.000186
          18181.0                                    0.000186
          18843.0                                    0.000186
          18900.0                                    0.000186
          19190.0                                    0.000186
          19250.0                                    0.000186
          19480.0                                    0.000186
          19546.0                                    0.000186
          19810.0                                    0.000186
          20635.0                                    0.000186
          21611.0                                    0.000186
          21870.0                                    0.000186
          23031.0                                    0.000186
          25033.0                                    0.000186
          27680.0                                    0.000186
          28077.0                                    0.000186
          28354.0                                    0.000186
          32250.0                                    0.000186
          33543.0                                    0.000186
Name: MMRAcquisitionAuctionAveragePrice, Length: 12518, dtype: float64
----------------------------------------------------------------------
IsBadBuy  MMRAcquisitionAuctionCleanPrice
0         0.0                                      0.009954
          6461.0                                   0.005601
          7450.0                                   0.002634
          1.0                                      0.002190
          8892.0                                   0.001969
          8258.0                                   0.001913
          6584.0                                   0.001830
          8449.0                                   0.001664
          7837.0                                   0.001553
          9044.0                                   0.001553
          8107.0                                   0.001469
          8469.0                                   0.001442
          5967.0                                   0.001414
          8466.0                                   0.001331
          7614.0                                   0.001248
          8151.0                                   0.001164
          6235.0                                   0.001137
          6508.0                                   0.001137
          9045.0                                   0.001137
          4783.0                                   0.001109
          7934.0                                   0.001109
          7195.0                                   0.001081
          8006.0                                   0.001081
          9772.0                                   0.001081
          7771.0                                   0.001054
          8187.0                                   0.001054
          8287.0                                   0.001054
          9027.0                                   0.001054
          6920.0                                   0.001026
          7280.0                                   0.001026
                                                     ...
1         17369.0                                  0.000186
          17383.0                                  0.000186
          17530.0                                  0.000186
          17625.0                                  0.000186
          17699.0                                  0.000186
          17734.0                                  0.000186
```

```
           18034.0                            0.000186
           18305.0                            0.000186
           18384.0                            0.000186
           18427.0                            0.000186
           19745.0                            0.000186
           20042.0                            0.000186
           20795.0                            0.000186
           20809.0                            0.000186
           21049.0                            0.000186
           21221.0                            0.000186
           21234.0                            0.000186
           21338.0                            0.000186
           21597.0                            0.000186
           21605.0                            0.000186
           23021.0                            0.000186
           23751.0                            0.000186
           23969.0                            0.000186
           25681.0                            0.000186
           28053.0                            0.000186
           29498.0                            0.000186
           30114.0                            0.000186
           30408.0                            0.000186
           35215.0                            0.000186
           36701.0                            0.000186
Name: MMRAcquisitionAuctionCleanPrice, Length: 13313, dtype: float64
----------------------------------------------------------------------
IsBadBuy  MMRAcquisitionRetailAveragePrice
0          0.0                                 0.012144
           6418.0                              0.005573
           7316.0                              0.002551
           8756.0                              0.002052
           11114.0                             0.001885
           6515.0                              0.001858
           11882.0                             0.001580
           7907.0                              0.001525
           8325.0                              0.001497
           5439.0                              0.001442
           9352.0                              0.001442
           9097.0                              0.001331
           11006.0                             0.001303
           7943.0                              0.001220
           6361.0                              0.001192
           4483.0                              0.001164
           8600.0                              0.001164
           8614.0                              0.001164
           9429.0                              0.001137
           7866.0                              0.001081
           10574.0                             0.001081
           7916.0                              0.001054
           9350.0                              0.001054
           6378.0                              0.001026
           7772.0                              0.001026
           10856.0                             0.001026
           10875.0                             0.001026
           11396.0                             0.001026
           9091.0                              0.000998
           6440.0                              0.000970
                                                 ...
1          18619.0                             0.000186
           18841.0                             0.000186
           18940.0                             0.000186
           19107.0                             0.000186
           19126.0                             0.000186
           19159.0                             0.000186
           20008.0                             0.000186
           20135.0                             0.000186
           20736.0                             0.000186
           20737.0                             0.000186
           20850.0                             0.000186
           20912.0                             0.000186
           20976.0                             0.000186
           21225.0                             0.000186
           21290.0                             0.000186
           21336.0                             0.000186
           21538.0                             0.000186
           21895.0                             0.000186
           22786.0                             0.000186
```

```
         23361.0                            0.000186
         23456.0                            0.000186
         23840.0                            0.000186
         24120.0                            0.000186
         27295.0                            0.000186
         30048.0                            0.000186
         30196.0                            0.000186
         31599.0                            0.000186
         33872.0                            0.000186
         35330.0                            0.000186
         36726.0                            0.000186
Name: MMRAcquisitionRetailAveragePrice, Length: 14426, dtype: float64
----------------------------------------------------------------------
IsBadBuy  MMRAcquisitonRetailCleanPrice
0         0.0                            0.012116
          7478.0                         0.005601
          9722.0                         0.002717
          8546.0                         0.002495
          11562.0                        0.001885
          10103.0                        0.001858
          7611.0                         0.001830
          9643.0                         0.001525
          10268.0                        0.001525
          8964.0                         0.001442
          9256.0                         0.001442
          6944.0                         0.001414
          12239.0                        0.001414
          11599.0                        0.001220
          11513.0                        0.001192
          8271.0                         0.001164
          11443.0                        0.001164
          11447.0                        0.001164
          7529.0                         0.001137
          5666.0                         0.001081
          9303.0                         0.001081
          11054.0                        0.001081
          10269.0                        0.001054
          12565.0                        0.001054
          9613.0                         0.001026
          10748.0                        0.001026
          7234.0                         0.000998
          7974.0                         0.000998
          9342.0                         0.000998
          9624.0                         0.000998
                                           ...
1         20732.0                        0.000186
          20868.0                        0.000186
          21171.0                        0.000186
          21370.0                        0.000186
          21662.0                        0.000186
          21825.0                        0.000186
          22145.0                        0.000186
          22888.0                        0.000186
          23011.0                        0.000186
          23123.0                        0.000186
          23233.0                        0.000186
          23419.0                        0.000186
          23433.0                        0.000186
          23545.0                        0.000186
          23738.0                        0.000186
          23825.0                        0.000186
          23833.0                        0.000186
          24870.0                        0.000186
          25363.0                        0.000186
          25640.0                        0.000186
          25799.0                        0.000186
          26151.0                        0.000186
          26387.0                        0.000186
          29981.0                        0.000186
          32383.0                        0.000186
          32760.0                        0.000186
          33736.0                        0.000186
          36096.0                        0.000186
          38532.0                        0.000186
          40137.0                        0.000186
Name: MMRAcquisitonRetailCleanPrice, Length: 14948, dtype: float64
----------------------------------------------------------------------
```

```
IsBadBuy  MMRCurrentAuctionAveragePrice
0         0.0                              0.006959
          6074.0                           0.004991
          5480.0                           0.004381
          6311.0                           0.002079
          7269.0                           0.002079
          8186.0                           0.001941
          8033.0                           0.001774
          7644.0                           0.001747
          5569.0                           0.001608
          6858.0                           0.001359
          6966.0                           0.001331
          8196.0                           0.001331
          6814.0                           0.001220
          7524.0                           0.001164
          6967.0                           0.001081
          7608.0                           0.001026
          7612.0                           0.001026
          7901.0                           0.001026
          5033.0                           0.000998
          7495.0                           0.000998
          8568.0                           0.000970
          6892.0                           0.000943
          8018.0                           0.000943
          8140.0                           0.000943
          8268.0                           0.000943
          4573.0                           0.000915
          7457.0                           0.000915
          7661.0                           0.000915
          5760.0                           0.000860
          7927.0                           0.000860
                                             ...
1         14827.0                          0.000186
          14982.0                          0.000186
          15038.0                          0.000186
          15161.0                          0.000186
          15292.0                          0.000186
          15368.0                          0.000186
          15581.0                          0.000186
          15605.0                          0.000186
          15852.0                          0.000186
          16091.0                          0.000186
          16412.0                          0.000186
          16645.0                          0.000186
          16721.0                          0.000186
          16988.0                          0.000186
          17240.0                          0.000186
          17343.0                          0.000186
          17779.0                          0.000186
          17844.0                          0.000186
          18416.0                          0.000186
          18546.0                          0.000186
          19359.0                          0.000186
          19963.0                          0.000186
          20817.0                          0.000186
          21940.0                          0.000186
          23015.0                          0.000186
          27543.0                          0.000186
          27795.0                          0.000186
          28099.0                          0.000186
          32250.0                          0.000186
          33369.0                          0.000186
Name: MMRCurrentAuctionAveragePrice, Length: 12440, dtype: float64
----------------------------------------------------------------------
IsBadBuy  MMRCurrentAuctionCleanPrice
0         7324.0                         0.005601
          0.0                            0.004935
          6461.0                         0.004464
          7450.0                         0.002079
          1.0                            0.002024
          8892.0                         0.001747
          7898.0                         0.001580
          8107.0                         0.001525
          6584.0                         0.001469
          9279.0                         0.001442
          9044.0                         0.001275
          8484.0                         0.001192
```

```
           7500.0                        0.001164
           7560.0                        0.001137
           8282.0                        0.001137
           9237.0                        0.001137
           5967.0                        0.001109
           8277.0                        0.001054
           9325.0                        0.001054
           8438.0                        0.000998
           8513.0                        0.000998
           8811.0                        0.000970
           9129.0                        0.000970
           8639.0                        0.000943
           8669.0                        0.000943
           9209.0                        0.000943
           7783.0                        0.000915
           7885.0                        0.000915
           8132.0                        0.000915
           8168.0                        0.000915
                                           ...
1          17267.0                       0.000186
           17366.0                       0.000186
           17432.0                       0.000186
           17515.0                       0.000186
           17621.0                       0.000186
           17625.0                       0.000186
           17699.0                       0.000186
           17751.0                       0.000186
           17767.0                       0.000186
           18427.0                       0.000186
           18762.0                       0.000186
           18942.0                       0.000186
           19025.0                       0.000186
           19080.0                       0.000186
           19760.0                       0.000186
           19769.0                       0.000186
           20133.0                       0.000186
           20422.0                       0.000186
           20790.0                       0.000186
           20881.0                       0.000186
           21514.0                       0.000186
           21601.0                       0.000186
           21870.0                       0.000186
           24293.0                       0.000186
           26168.0                       0.000186
           29042.0                       0.000186
           29811.0                       0.000186
           30136.0                       0.000186
           35215.0                       0.000186
           36478.0                       0.000186
Name: MMRCurrentAuctionCleanPrice, Length: 13194, dtype: float64
----------------------------------------------------------------------
IsBadBuy  MMRCurrentRetailAveragePrice
0         0.0                             0.006959
          8704.0                          0.005213
          6418.0                          0.004298
          7316.0                          0.002052
          8756.0                          0.001747
          10834.0                         0.001580
          11674.0                         0.001525
          6515.0                          0.001442
          7907.0                          0.001331
          9352.0                          0.001303
          11237.0                         0.001164
          10921.0                         0.001137
          9753.0                          0.001081
          10564.0                         0.000970
          11640.0                         0.000970
          11710.0                         0.000970
          5439.0                          0.000943
          7943.0                          0.000943
          9429.0                          0.000943
          10481.0                         0.000887
          11598.0                         0.000887
          8885.0                          0.000860
          11913.0                         0.000860
          9332.0                          0.000832
          11713.0                         0.000832
```

```
           6721.0                          0.000804
           8554.0                          0.000804
           9005.0                          0.000804
           9085.0                          0.000804
           9128.0                          0.000804
                                             ...
1          18073.0                         0.000186
           18225.0                         0.000186
           18559.0                         0.000186
           19092.0                         0.000186
           19127.0                         0.000186
           19160.0                         0.000186
           19299.0                         0.000186
           19643.0                         0.000186
           20090.0                         0.000186
           20291.0                         0.000186
           20379.0                         0.000186
           20912.0                         0.000186
           20930.0                         0.000186
           21181.0                         0.000186
           21431.0                         0.000186
           21688.0                         0.000186
           22417.0                         0.000186
           22581.0                         0.000186
           22850.0                         0.000186
           23327.0                         0.000186
           24286.0                         0.000186
           24349.0                         0.000186
           24501.0                         0.000186
           27269.0                         0.000186
           28050.0                         0.000186
           29921.0                         0.000186
           31128.0                         0.000186
           32928.0                         0.000186
           35330.0                         0.000186
           36539.0                         0.000186
Name: MMRCurrentRetailAveragePrice, Length: 14229, dtype: float64
--------------------------------------------------------------------
IsBadBuy  MMRCurrentRetailCleanPrice
0          0.0                           0.006959
           10090.0                       0.005601
           7478.0                        0.004381
           8546.0                        0.002024
           10103.0                       0.001747
           12387.0                       0.001580
           7611.0                        0.001497
           11413.0                       0.001497
           12864.0                       0.001331
           10268.0                       0.001303
           9256.0                        0.001275
           11706.0                       0.001275
           11739.0                       0.001109
           11542.0                       0.001081
           12701.0                       0.001081
           10571.0                       0.001054
           12308.0                       0.001054
           6944.0                        0.001026
           9830.0                        0.001026
           11431.0                       0.001026
           11944.0                       0.000998
           12309.0                       0.000998
           11054.0                       0.000970
           12272.0                       0.000943
           12687.0                       0.000943
           9613.0                        0.000887
           10599.0                       0.000887
           12252.0                       0.000860
           7826.0                        0.000832
           7234.0                        0.000804
                                           ...
1          20673.0                       0.000186
           20806.0                       0.000186
           21047.0                       0.000186
           21102.0                       0.000186
           21370.0                       0.000186
           21910.0                       0.000186
           22053.0                       0.000186
```

```
        22079.0                        0.000186
        22120.0                        0.000186
        22202.0                        0.000186
        22888.0                        0.000186
        23285.0                        0.000186
        23673.0                        0.000186
        23922.0                        0.000186
        24113.0                        0.000186
        24396.0                        0.000186
        24653.0                        0.000186
        25060.0                        0.000186
        25342.0                        0.000186
        25518.0                        0.000186
        25970.0                        0.000186
        26143.0                        0.000186
        26164.0                        0.000186
        30194.0                        0.000186
        31317.0                        0.000186
        31744.0                        0.000186
        33014.0                        0.000186
        35366.0                        0.000186
        38532.0                        0.000186
        39896.0                        0.000186
Name: MMRCurrentRetailCleanPrice, Length: 14699, dtype: float64
----------------------------------------------------------------------
```

Decision tree

Task 2. Predictive Modeling Using Decision Trees

1. Python: Build a decision tree using the default setting.

Answer the followings: a. What is the classification accuracy on training and test datasets?

As the data we have is imbalanced(we don't have equal proportion of zeros and ones for target variable IsBadBuy), we are doing oversampling and undersampling to balance the dataset. Below are the test and training accuracies for normal data given, oversampled data and undersampled data respectively in the tabular format.

```
               Normal dataset  Oversampling dataset    Under Sampling Dataset
```

Train Accuracy 1.0 1.0 1.0 Test Accuracy 0.7851166532582462 0.7839098954143202 0.22751407884151248

b. What is the size of tree (i.e. number of nodes)?

```
               Normal dataset  Oversampling dataset    Under Sampling Dataset
```

NumberOfNodes 7373 7973 923

c. How many leaves are in the tree that is selected based on the validation data set?

```
               Normal dataset  Oversampling dataset    Under Sampling Dataset
```

NumberOfNodes 3687 3987 462

d. Which variable is used for the first split? What are the competing splits for this first split? WheelType is the variable used for the first split. Auction and VehYear are the competing splits.

e. What are the 5 important variables in building the tree?

WheelType,Auction,VehYear,Make,TopThreeAmericanName are the five important features here. We can see this from feature importance cell.

f. Report if you see any evidence of model overfitting.

From the graph, we can observe the testdata is perfrming better than training data from maxdepth 2.5 to 8. After maxdepth=8, test data is not performing better than the training data. So overfitting is there.

g. Did changing the default setting (i.e., only focus on changing the setting of the number of splits to create a node) help improving the model? Answer the above questions on the best performing tree.

Answer: Yes. We changed the number of splits by changing the max-depth to 3. Then accuracy has been improved a little bit.

```
                   Normal dataset   Oversampling dataset    Under Sampling Dataset   Max depth i
s 3
```

Train Accuracy 1.0 1.0 1.0 0.7325820889 Test Accuracy 0.7851166532582462 0.7839098954143202 0.22751407884151248 0.7746580852

In [21]:

```
# simple decision tree training
model = DecisionTreeClassifier(random_state=rs)
model.fit(X_train, Y_train)
#print Y_train.value_counts()
print ("*********************Simple decisoin tree*********************")
print("Train accuracy:", model.score(X_train, Y_train))
print("Test accuracy:", model.score(X_test, Y_test))
print("Number of nodes: ",model.tree_.node_count)
Y_pred = model.predict(X_test)
print(classification_report(Y_test, Y_pred))

def get_num_leaves(model):
    n_nodes = model.tree_.node_count
    ll = model.tree_.children_left
    rl = model.tree_.children_right
    count = 0
    for i in range(0,n_nodes):
        if (ll[i] & rl[i]) == -1:
            count = count + 1
    return count

print("Number of leaves in the tree",get_num_leaves(model))

print("Number of leaves present in the leftside",model.tree_.children_left)
print("Number of leaves present in the rightside",model.tree_.children_right)
```

```
*********************Simple decisoin tree*********************
Train accuracy: 1.0
Test accuracy: 0.7851166532582462
Number of nodes:  7373
              precision    recall  f1-score   support

           0       0.88      0.87      0.88     10820
           1       0.21      0.23      0.22      1610

   micro avg       0.79      0.79      0.79     12430
   macro avg       0.55      0.55      0.55     12430
weighted avg       0.80      0.79      0.79     12430

Number of leaves in the tree 3687
Number of leaves present in the leftside [   1    2    3 ... 7371   -1   -1]
Number of leaves present in the rightside [4520  555  138 ... 7372   -1   -1]
```

In [22]:

```
# simple decision tree training for Oversampled data

sm = SMOTE(random_state=42)
x_res, y_res = sm.fit_sample(X_train, Y_train)
model.fit(x_res, y_res)
print ("*********************Prediction for test data*********************")
print("Train accuracy:", model.score(x_res, y_res))
print("Test accuracy:", model.score(X_test, Y_test))
print("Number of nodes: ",model.tree_.node_count)
y_pred = model.predict(X_test)
print(classification_report(Y_test, y_pred))

def get_num_leaves(model):
    n_nodes = model.tree_.node_count
    ll = model.tree_.children_left
    rl = model.tree_.children_right
    count = 0
```

```
    for i in range(0,n_nodes):
        if (ll[i] & rl[i]) == -1:
            count = count + 1
    return count

print("Number of leaves in the tree",get_num_leaves(model))

print("Number of leaves present in the leftside",model.tree_.children_left)
print("Number of leaves present in the rightside",model.tree_.children_right)
```

```
********************Prediction for test data*********************
Train accuracy: 1.0
Test accuracy: 0.7839098954143202
Number of nodes:  7973
              precision    recall  f1-score   support

           0       0.89      0.86      0.87     10820
           1       0.22      0.26      0.24      1610

   micro avg       0.78      0.78      0.78     12430
   macro avg       0.55      0.56      0.56     12430
weighted avg       0.80      0.78      0.79     12430

Number of leaves in the tree 3987
Number of leaves present in the leftside [ 1  2  3 ... -1 -1 -1]
Number of leaves present in the rightside [5028 5027 1690 ...   -1   -1   -1]
```

In [23]:

```
# simple decision tree training for undersampled data
cc = ClusterCentroids(random_state=0)
X_under, Y_under = cc.fit_resample(X_train, Y_train)
model = DecisionTreeClassifier(random_state=rs)
model.fit(X_under, Y_under)
print ("********************Decisoin tree with underfitting of the train
data*********************")
print("Train accuracy:", model.score(X_under, Y_under))
print("Test accuracy:", model.score(X_test, Y_test))
print("Number of nodes: ",model.tree_.node_count)
print ("********************Prediction for test data*********************")
y_pred = model.predict(X_test)
print(classification_report(Y_test, y_pred))

def get_num_leaves(model):
    n_nodes = model.tree_.node_count
    ll = model.tree_.children_left
    rl = model.tree_.children_right
    count = 0
    for i in range(0,n_nodes):
        if (ll[i] & rl[i]) == -1:
            count = count + 1
    return count

print("Number of leaves in the tree",get_num_leaves(model))
```

```
********************Decisoin tree with underfitting of the train data********************
Train accuracy: 1.0
Test accuracy: 0.22751407884151248
Number of nodes:  923
********************Prediction for test data*********************
              precision    recall  f1-score   support

           0       0.89      0.13      0.22     10820
           1       0.13      0.90      0.23      1610

   micro avg       0.23      0.23      0.23     12430
   macro avg       0.51      0.51      0.23     12430
weighted avg       0.79      0.23      0.22     12430

Number of leaves in the tree 462
```

In [24]:

```
# grab feature importances from the model and feature name from the original X
```

```
importances = model.feature_importances_
feature_names = X.columns

# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)

# limit to 20 features, you can leave this out to print out everything
indices = indices[:20]

for i in indices:
    print(feature_names[i], ':', importances[i])
```

```
WheelType : 0.5193201112555421
MMRCurrentRetailCleanPrice : 0.08695262600861428
Color : 0.05021582558505198
VehOdo : 0.040198242192698845
VehBCost : 0.03152407246499129
WarrantyCost : 0.030431006723877545
MMRAcquisitionAuctionCleanPrice : 0.025506270772206595
MMRCurrentAuctionCleanPrice : 0.024304890248503425
MMRAcquisitionRetailAveragePrice : 0.023707795484547933
Auction : 0.022296079407274006
Size : 0.020683431645761308
MMRCurrentRetailAveragePrice : 0.018971382791140013
MMRAcquisitonRetailCleanPrice : 0.01883585753469289
MMRAcquisitionAuctionAveragePrice : 0.015889166999467193
VNST : 0.014541274111661815
VehYear : 0.013764994799570036
MMRCurrentAuctionAveragePrice : 0.013409476585198204
TopThreeAmericanName : 0.011878920097363115
Make : 0.010779762872448508
Nationality : 0.0025594011596332213
```

In [25]:

```
#retrain with a small max_depth limit

model = DecisionTreeClassifier(max_depth=3, random_state=rs)
model.fit(x_res, y_res)

print("Train accuracy:", model.score(x_res, y_res))
print("Test accuracy:", model.score(X_test, Y_test))

y_pred = model.predict(X_test)
print(classification_report(Y_test, y_pred))
```

```
Train accuracy: 0.7325820889610647
Test accuracy: 0.7746580852775543
              precision    recall  f1-score   support

           0       0.88      0.85      0.87     10820
           1       0.20      0.25      0.22      1610

   micro avg       0.77      0.77      0.77     12430
   macro avg       0.54      0.55      0.54     12430
weighted avg       0.80      0.77      0.78     12430
```

In [26]:

```
# grab feature importance from the model and feature name from the original X
importances = model.feature_importances_
feature_names = X.columns

# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)

# limit to 20 features, you can leave this out to print out everything
indices = indices[:20]

for i in indices:
    print(feature_names[i], ':', importances[i])
```

```
    print(feature_names[i], ':', importances[i])

# visualize
print("Number of nodes: ",model.tree_.node_count)
dotfile = StringIO()
export_graphviz(model, out_file=dotfile, feature_names=X.columns)
graph = pydot.graph_from_dot_data(dotfile.getvalue())
graph[0].write_png("week3_dt_viz.png") # saved in the following file
```

```
WheelType : 0.8196587486514747
Auction : 0.0877895719305143
VehYear : 0.08725508168771123
Nationality : 0.005296597730299794
IsOnlineSale : 0.0
Make : 0.0
Color : 0.0
Transmission : 0.0
VehOdo : 0.0
Size : 0.0
TopThreeAmericanName : 0.0
WarrantyCost : 0.0
MMRAcquisitionAuctionCleanPrice : 0.0
MMRAcquisitionRetailAveragePrice : 0.0
MMRAcquisitonRetailCleanPrice : 0.0
MMRCurrentAuctionAveragePrice : 0.0
MMRCurrentAuctionCleanPrice : 0.0
MMRCurrentRetailAveragePrice : 0.0
MMRCurrentRetailCleanPrice : 0.0
VNST : 0.0
Number of nodes:  13
```

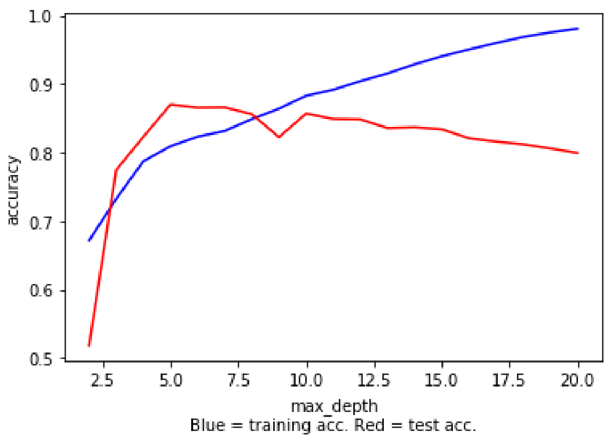
In [27]:

```
test_score = []
train_score = []

# check the model performance for max depth from 2-20
for max_depth in range(2, 21):
    model = DecisionTreeClassifier(max_depth=max_depth, random_state=rs)
    model.fit(x_res, y_res)
    test_score.append(model.score(X_test, Y_test))
    train_score.append(model.score(x_res, y_res))
```

In [28]:

```
# plot max depth hyperparameter values vs training and test accuracy score
print("Number of nodes: ",model.tree_.node_count)
plt.plot(range(2, 21), train_score, 'b', range(2,21), test_score, 'r')
plt.xlabel('max_depth\nBlue = training acc. Red = test acc.')
plt.ylabel('accuracy')
plt.show()
```

```
Number of nodes:  6075
```



In [37]:

```
# visualize
dotfile = StringIO()


analyse_feature_importance(model, X.columns, 20)
visualize_decision_tree(model, X.columns, "optimal_tree.png")
img = Image.open('optimal_tree.png')
new_width  = 70000
new_height =6000
img = img.resize((new_width, new_height), Image.ANTIALIAS)
img.save('optimal_tree.png')
img.show()
```

```
WheelType : 0.27254852450362715
Auction : 0.1799555474226916
VehYear : 0.1292813916062509
Make : 0.06268755690544962
TopThreeAmericanName : 0.05327562753257296
VehOdo : 0.03808119317710718
VehBCost : 0.03538142037570167
MMRCurrentAuctionCleanPrice : 0.022310905812623107
MMRCurrentRetailAveragePrice : 0.021712119423501562
MMRAcquisitonRetailCleanPrice : 0.020259026348441542
MMRCurrentRetailCleanPrice : 0.019790010254355634
MMRAcquisitionRetailAveragePrice : 0.01965782567957907
WarrantyCost : 0.01902925885020823
MMRCurrentAuctionAveragePrice : 0.018443183697614184
MMRAcquisitionAuctionAveragePrice : 0.01806431330321963
MMRAcquisitionAuctionCleanPrice : 0.016178665179954733
Color : 0.016035190805012162
VNST : 0.015816419285077397
Size : 0.011886114811957038
Nationality : 0.006236801846077701
```

Finding optimal hyperparameters with GridSearchCV

1. Python: Build another decision tree tuned with GridSearchCV

a. What is the classification accuracy on training and test datasets? We are considering the oversampleddata as it is balanicng the the target values. Below is the accuracy for training data and testing data for oversampleddata with gridsearchCV.

Train accuracy: 0.8312670812373747 Test accuracy: 0.8662107803700724

b. What is the size of tree (i.e. number of nodes)? Is the size different from the maximal tree or the tree in the previous step? Why?

We have number of leaves 707. The size is different when compared to the maximal tree in the previous step becuase we are using gridsearchCV with different parameters like criterion,max_depth and min_samples_leaf. It will get optimal tree by the models from n possible cases of training set and test sets.

c. How many leaves are in the tree that is selected based on the validation dataset?

{'criterion': 'gini', 'max_depth': 11, 'min_samples_leaf': 10} Number of leaves in the tree 354

d. Which variable is used for the first split? What are the competing splits for this first split?

WheelType has been used for the first split. Competing split is based on the Auction.

e. What are the 5 important variables in building the tree? Five important features are

WheelType Auction VehYear Make TopThreeAmericanName

f. Report if you see any evidence of model overfitting. There is no sign of Overfitting. Because the training and testing accuracy is performing well. The Depth of the tree and number of leaves is optimal. Testing accuracy is also fine.

g. What are the parameters used? Explain your choices. Criterion, max_depth and min_samples_leaf are the paramters userd. We are trying to get the optimal tree by choosing the best criterion among gini and entropy and proper max_depth and minimum samples leaf.Criterion is the function to measure the quality of a split.

In [41]:

```
# grid search CV
params = {'criterion': ['gini', 'entropy'],
```

```python
          'max_depth': range(2, 12),
          'min_samples_leaf': range(10,40, 5)}

cv = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(random_state=rs), cv=10)
cv.fit(x_res, y_res)

print("Train accuracy:", cv.score(x_res, y_res))
print("Test accuracy:", cv.score(X_test, Y_test))

# test the best model
y_pred = cv.predict(X_test)
print(classification_report(Y_test, Y_pred))

# print parameters of the best model
print(cv.best_params_)

def get_num_leaves(model):
    n_nodes = model.tree_.node_count
    ll = model.tree_.children_left
    rl = model.tree_.children_right
    count = 0
    for i in range(0,n_nodes):
        if (ll[i] & rl[i]) == -1:
            count = count + 1
    return count

print("Number of nodes: ",cv. best_estimator_   .tree_.node_count)
print('-'*50)
print("Number of leaves in the tree",get_num_leaves(cv. best_estimator_))
print('-'*50)
```

```
Train accuracy: 0.8881253218204143
Test accuracy: 0.8462590506838295
              precision    recall  f1-score   support

           0       0.88      0.87      0.88     10820
           1       0.21      0.23      0.22      1610

   micro avg       0.79      0.79      0.79     12430
   macro avg       0.55      0.55      0.55     12430
weighted avg       0.80      0.79      0.79     12430

{'criterion': 'gini', 'max_depth': 11, 'min_samples_leaf': 10}
Number of nodes:  707
--------------------------------------------------
Number of leaves in the tree 354
--------------------------------------------------
```

In [42]:

```python
# grid search CV
params = {'criterion': ['gini', 'entropy'],
          'max_depth': range(10, 12),
          'min_samples_leaf': range(8, 15, 2)}

cv = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(random_state=rs), cv=10)
cv.fit(x_res, y_res)

print("Train accuracy:", cv.score(x_res, y_res))
print("Test accuracy:", cv.score(X_test, Y_test))

# test the best model
y_pred = cv.predict(X_test)
print(classification_report(Y_test, Y_pred))

# print parameters of the best model
print(cv.best_params_)


print("Number of nodes: ",cv. best_estimator_   .tree_.node_count)
print('-'*50)
print("Number of leaves in the tree",get_num_leaves(cv. best_estimator_))
print('-'*50)
```

```
Train accuracy: 0.8881253218204143
Test accuracy: 0.8462590506838295
              precision    recall  f1-score   support

           0       0.88      0.87      0.88     10820
           1       0.21      0.23      0.22      1610

   micro avg       0.79      0.79      0.79     12430
   macro avg       0.55      0.55      0.55     12430
weighted avg       0.80      0.79      0.79     12430

{'criterion': 'gini', 'max_depth': 11, 'min_samples_leaf': 10}
Number of nodes:  707
--------------------------------------------------
Number of leaves in the tree 354
--------------------------------------------------
```

In [43]:

```
import numpy as np
import pydot
from io import StringIO
from sklearn.tree import export_graphviz

def analyse_feature_importance(dm_model, feature_names, n_to_display=20):
    # grab feature importances from the model
    importances = dm_model.feature_importances_

    # sort them out in descending order
    indices = np.argsort(importances)
    indices = np.flip(indices, axis=0)

    # limit to 20 features, you can leave this out to print out everything
    indices = indices[:n_to_display]

    for i in indices:
        print(feature_names[i], ':', importances[i])

def visualize_decision_tree(dm_model, feature_names, save_name):
    dotfile = StringIO()
    export_graphviz(dm_model, out_file=dotfile, feature_names=feature_names)
    graph = pydot.graph_from_dot_data(dotfile.getvalue())
    graph[0].write_png(save_name) # saved in the following file


analyse_feature_importance(cv.best_estimator_, X.columns, 20)
visualize_decision_tree(cv.best_estimator_, X.columns, "optimal_tree.png")
img = Image.open('optimal_tree.png')
new_width  = 70000
new_height =6000
img = img.resize((new_width, new_height), Image.ANTIALIAS)
img.save('optimal_tree.png')
img.show()
```

```
WheelType : 0.37947729011943926
Auction : 0.24738203591539104
VehYear : 0.15653290048457935
Make : 0.06890074370292999
TopThreeAmericanName : 0.05672496040350978
VehBCost : 0.01503433953486562
VehOdo : 0.011850695926835592
MMRAcquisitionRetailAveragePrice : 0.007114144425115366
MMRCurrentAuctionCleanPrice : 0.006725079060459914
MMRCurrentRetailCleanPrice : 0.006560750300985196
MMRCurrentRetailAveragePrice : 0.0064748932077347485
WarrantyCost : 0.00595886682981698
MMRAcquisitionAuctionAveragePrice : 0.00542409748998343
MMRAcquisitonRetailCleanPrice : 0.004719512839399656
Size : 0.00460513659508568
Nationality : 0.0042393168642713384
MMRAcquisitionAuctionCleanPrice : 0.0030628930478118353
VNST : 0.002931243056199737
Color : 0.002578750625361529
MMRCurrentAuctionAveragePrice : 0.001990652939404491
```

1. What is the significant difference do you see between these two decision tree models (steps 2.1 & 2.2)? How do they compare performance-wise? Explain why those changes may have happened.

Answer: Step 2.1 gave us maximal tree whereas step 2.2 gave us the optimal tree with the gini criterion. Using grid searchCV, we are taking n samples of data. Among them we are taking n-1 samples as training data nth sample as test data with different hypermeters like criterion,max_depth and minimum number of sample leaves. We will do the same thing by chaning the test sample until all n samples becomes the test data. Then it will give us the optimal tree with the best hyperperameters with best accuracy.

1. From the better model, can you identify which cars could potential be “kicks”? Can you provide some descriptive summary of those cars? Based on the feature importance the top 5 features plays important role in determing the kicked cars. WheelType Auction VehYear Make TopThreeAmericanName