Auctioned Car Preditions - Machine Learning

Author: Marzena Porebski
Email: mmazur5@yahoo.com

Objective: To predict the quality of a car purchased at the Auction, using car related and purchase related data.

Target variable: "IsBadBuy"
Definition: Post-purchase classification for a kicked or non-kicked car.
Rationale: Most relevant dependant variable that enables us identify key features that predict likelihood of a kicked car.

In [442]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,precision_score,
recall_score,precision_recall_curve,roc_auc_score,confusion_matrix,roc_curve
```

In [439]:
```
#kernel.json
```

In [440]:
```
#import sys
#!{sys.executable} -m pip install xgboost
```

In [425]:
```
import xgboost as xgb
import lightgbm as lgbm
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-425-45a63c4f3d31> in <module>
----> 1 import xgboost as xgb
      2 import lightgbm as lgbm

ModuleNotFoundError: No module named 'xgboost'
```

In [423]:
```
#conda install -c anaconda py-xgboost
```

In [424]:
```
#!pip install xgboost
```

Load the data

In [456]:
```
cars = pd.read_csv('U:/Documents/Gunn Notes/Data Analyst Training/CELL/FINAL PROJECT/training.csv', sep=",", low_memory=False)
```

Examine the Data

In [457]:
```
#display 5000 columns in a dataframe
pd.set_option('display.max_columns', 5000)
```

In [458]:
```
cars.head()
```

Out[458]:

	RefId	IsBadBuy	PurchDate	Auction	VehYear	VehicleAge	Make	Model	Trim	SubModel	Color
0	1	0	12/7/2009	ADESA	2006	3	MAZDA	MAZDA3	i	4D SEDAN I	RED
1	2	0	12/7/2009	ADESA	2004	5	DODGE	1500 RAM PICKUP 2WD	ST	QUAD CAB 4.7L SLT	WHITE
2	3	0	12/7/2009	ADESA	2005	4	DODGE	STRATUS V6	SXT	4D SEDAN SXT FFV	MAROON
3	4	0	12/7/2009	ADESA	2004	5	DODGE	NEON	SXT	4D SEDAN	SILVER
4	5	0	12/7/2009	ADESA	2005	4	FORD	FOCUS	ZX3	2D COUPE ZX3	SILVER

In [459]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72983 entries, 0 to 72982
Data columns (total 34 columns):
RefId                                72983 non-null int64
IsBadBuy                             72983 non-null int64
PurchDate                            72983 non-null object
Auction                              72983 non-null object
VehYear                              72983 non-null int64
VehicleAge                           72983 non-null int64
Make                                 72983 non-null object
Model                                72983 non-null object
Trim                                 70623 non-null object
SubModel                             72975 non-null object
Color                                72975 non-null object
Transmission                         72974 non-null object
WheelTypeID                          69814 non-null float64
WheelType                            69809 non-null object
VehOdo                               72983 non-null int64
Nationality                          72978 non-null object
Size                                 72978 non-null object
TopThreeAmericanName                 72978 non-null object
MMRAcquisitionAuctionAveragePrice    72965 non-null float64
MMRAcquisitionAuctionCleanPrice      72965 non-null float64
MMRAcquisitionRetailAveragePrice     72965 non-null float64
MMRAcquisitonRetailCleanPrice        72965 non-null float64
MMRCurrentAuctionAveragePrice        72668 non-null float64
MMRCurrentAuctionCleanPrice          72668 non-null float64
MMRCurrentRetailAveragePrice         72668 non-null float64
MMRCurrentRetailCleanPrice           72668 non-null float64
PRIMEUNIT                            3419 non-null object
AUCGUART                             3419 non-null object
BYRNO                                72983 non-null int64
VNZIP1                               72983 non-null int64
VNST                                 72983 non-null object
VehBCost                             72983 non-null float64
IsOnlineSale                         72983 non-null int64
WarrantyCost                         72983 non-null int64
dtypes: float64(10), int64(9), object(15)
memory usage: 18.9+ MB
```

In [460]:
```
cars.shape
```

Out[460]:
```
(72983, 34)
```

In [461]:
```
cars.columns
```

Out[461]:
```
Index(['RefId', 'IsBadBuy', 'PurchDate', 'Auction', 'VehYear', 'VehicleAge',
       'Make', 'Model', 'Trim', 'SubModel', 'Color', 'Transmission',
       'WheelTypeID', 'WheelType', 'VehOdo', 'Nationality', 'Size',
       'TopThreeAmericanName', 'MMRAcquisitionAuctionAveragePrice',
       'MMRAcquisitionAuctionCleanPrice', 'MMRAcquisitionRetailAveragePrice',
       'MMRAcquisitonRetailCleanPrice', 'MMRCurrentAuctionAveragePrice',
       'MMRCurrentAuctionCleanPrice', 'MMRCurrentRetailAveragePrice',
       'MMRCurrentRetailCleanPrice', 'PRIMEUNIT', 'AUCGUART', 'BYRNO',
       'VNZIP1', 'VNST', 'VehBCost', 'IsOnlineSale', 'WarrantyCost'],
      dtype='object')
```

Count missing values in all columns in a dataset

In [450]:
```
cars.isnull().sum()
```

Out[450]:
```
RefId                                0
IsBadBuy                             0
PurchDate                            0
Auction                              0
VehYear                              0
VehicleAge                           0
Make                                 0
Model                                0
Trim                              2360
SubModel                             8
Color                                8
Transmission                         9
WheelTypeID                       3169
WheelType                         3174
VehOdo                               0
Nationality                          5
Size                                 5
TopThreeAmericanName                 5
MMRAcquisitionAuctionAveragePrice   18
MMRAcquisitionAuctionCleanPrice     18
MMRAcquisitionRetailAveragePrice    18
MMRAcquisitonRetailCleanPrice       18
MMRCurrentAuctionAveragePrice      315
MMRCurrentAuctionCleanPrice        315
MMRCurrentRetailAveragePrice       315
MMRCurrentRetailCleanPrice         315
PRIMEUNIT                        69564
AUCGUART                         69564
BYRNO                                0
VNZIP1                               0
VNST                                 0
VehBCost                             0
IsOnlineSale                         0
WarrantyCost                         0
dtype: int64
```

Drop two features with 95% missing values

In [462]:
```
cars = cars.drop(['PRIMEUNIT', 'AUCGUART'], axis=1)
# Other way to drop columns
#df.drop(columns=['PRIMEUNIT', 'AUCGUART'])
```

Re-count missing values in all columns in a dataset

In [463]:
```
cars.isnull().sum()
```

Out[463]:
```
RefId                                0
IsBadBuy                             0
PurchDate                            0
Auction                              0
VehYear                              0
VehicleAge                           0
Make                                 0
Model                                0
Trim                              2360
SubModel                             8
Color                                8
Transmission                         9
WheelTypeID                       3169
WheelType                         3174
VehOdo                               0
Nationality                          5
Size                                 5
TopThreeAmericanName                 5
MMRAcquisitionAuctionAveragePrice   18
MMRAcquisitionAuctionCleanPrice     18
MMRAcquisitionRetailAveragePrice    18
MMRAcquisitonRetailCleanPrice       18
MMRCurrentAuctionAveragePrice      315
MMRCurrentAuctionCleanPrice        315
MMRCurrentRetailAveragePrice       315
MMRCurrentRetailCleanPrice         315
BYRNO                                0
VNZIP1                               0
VNST                                 0
VehBCost                             0
IsOnlineSale                         0
WarrantyCost                         0
dtype: int64
```

In [453]:
```
cars.head()
```

Out[453]:

	RefId	IsBadBuy	PurchDate	Auction	VehYear	VehicleAge	Make	Model	Trim	SubModel	Color
0	1	0	12/7/2009	ADESA	2006	3	MAZDA	MAZDA3	i	4D SEDAN I	RED
1	2	0	12/7/2009	ADESA	2004	5	DODGE	1500 RAM PICKUP 2WD	ST	QUAD CAB 4.7L SLT	WHITE
2	3	0	12/7/2009	ADESA	2005	4	DODGE	STRATUS V6	SXT	4D SEDAN SXT FFV	MAROON
3	4	0	12/7/2009	ADESA	2004	5	DODGE	NEON	SXT	4D SEDAN	SILVER
4	5	0	12/7/2009	ADESA	2005	4	FORD	FOCUS	ZX3	2D COUPE ZX3	SILVER

Drop following variables:
"RefId" - unique number with no relations to prediction
"VehYear" - which is year of a vehicle, same information is caputured in "VehicleAge"
"WheelTypeID" - same information is captured in "WheelType" column
"VNZIP1" - zip code showing location of the auction; "VNST" capures state which is more useful
"BYRNO" - provides unique number assigned to the car buyer and hence not useful in the analysis

In [464]:
```
cars = cars.drop(['RefId', 'VehYear', 'WheelTypeID', 'VNZIP1', 'BYRNO'], axis=1)
```

In [465]:
```
cars.head()
```

Out[465]:

	IsBadBuy	PurchDate	Auction	VehicleAge	Make	Model	Trim	SubModel	Color	Transmission	V
0	0	12/7/2009	ADESA	3	MAZDA	MAZDA3	i	4D SEDAN I	RED	AUTO	
1	0	12/7/2009	ADESA	5	DODGE	1500 RAM PICKUP 2WD	ST	QUAD CAB 4.7L SLT	WHITE	AUTO	
2	0	12/7/2009	ADESA	4	DODGE	STRATUS V6	SXT	4D SEDAN SXT FFV	MAROON	AUTO	
3	0	12/7/2009	ADESA	5	DODGE	NEON	SXT	4D SEDAN	SILVER	AUTO	
4	0	12/7/2009	ADESA	4	FORD	FOCUS	ZX3	2D COUPE ZX3	SILVER	MANUAL	

In [466]:
```
cars.isnull().sum()
```

Out[466]:
```
IsBadBuy                                0
PurchDate                               0
Auction                                 0
VehicleAge                              0
Make                                    0
Model                                   0
Trim                                 2360
SubModel                                8
Color                                   8
Transmission                            9
WheelType                            3174
VehOdo                                  0
Nationality                             5
Size                                    5
TopThreeAmericanName                    5
MMRAcquisitionAuctionAveragePrice      18
MMRAcquisitionAuctionCleanPrice        18
MMRAcquisitionRetailAveragePrice       18
MMRAcquisitonRetailCleanPrice          18
MMRCurrentAuctionAveragePrice         315
MMRCurrentAuctionCleanPrice           315
MMRCurrentRetailAveragePrice          315
MMRCurrentRetailCleanPrice            315
VNST                                    0
VehBCost                                0
IsOnlineSale                            0
WarrantyCost                            0
dtype: int64
```

In [467]:
```
#define nominal (categorical) columns manually
nominal_cols = ['PurchDate', 'Auction', 'Make', 'Model', 'Trim', 'SubModel', 'Colo
r',
        'Transmission', 'WheelType', 'Nationality', 'Size',
        'TopThreeAmericanName', 'VNST']
```

In [468]:
```
#define numeric (continuous) columns manually
num_cols = ['VehicleAge', 'VehOdo',
        'MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice',
        'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitonRetailCleanPrice',
        'MMRCurrentAuctionAveragePrice', 'MMRCurrentAuctionCleanPrice',
        'MMRCurrentRetailAveragePrice', 'MMRCurrentRetailCleanPrice', 'VehBCost',
'IsOnlineSale', 'WarrantyCost']
```

In [469]:
```
# Split data into nominal (categorical) and numeric (continuous) features.
def get_cat_cont_cols(df,cols):
    nominal_cols = []
    num_cols = []
    for col in cols:
        if df[col].dtype == 'object':
            nominal_cols.append(col)
        else:
            num_cols.append(col)
    return nominal_cols,num_cols
```

In [470]:
```
total_columns = cars.columns
nominal_cols,num_cols =  get_cat_cont_cols(cars, total_columns)
print ("Nominal (Categorical) columns: \n", nominal_cols)
#print ("\nNumeric (Continuous) columns: \n", num_cols)
```

```
Nominal (Categorical) columns:
 ['PurchDate', 'Auction', 'Make', 'Model', 'Trim', 'SubModel', 'Color', 'Transmi
ssion', 'WheelType', 'Nationality', 'Size', 'TopThreeAmericanName', 'VNST']
```

In [471]:
```
target = ['IsBadBuy']
num_cols = list(set(num_cols) - set(target))
features = nominal_cols + num_cols
print ("Numeric (Continuous) columns after target removal: \n", num_cols)
```

```
Numeric (Continuous) columns after target removal:
 ['MMRAcquisitionAuctionAveragePrice', 'MMRCurrentAuctionCleanPrice', 'MMRAcquis
itionRetailAveragePrice', 'MMRCurrentRetailCleanPrice', 'VehBCost', 'IsOnlineSal
e', 'MMRAcquisitionAuctionCleanPrice', 'MMRCurrentRetailAveragePrice', 'MMRAcqui
sitonRetailCleanPrice', 'VehicleAge', 'WarrantyCost', 'MMRCurrentAuctionAverageP
rice', 'VehOdo']
```

In [472]:
```
#cars = get_cat_cont_cols(cars)
```

Check number of nominal (categorical) attributes in the dataset

In [473]:
```
nominal_df = cars.select_dtypes(include=[np.object])
nominal_df.shape, nominal_df.columns
```

Out[473]:
```
((72983, 13),
 Index(['PurchDate', 'Auction', 'Make', 'Model', 'Trim', 'SubModel', 'Color',
        'Transmission', 'WheelType', 'Nationality', 'Size',
        'TopThreeAmericanName', 'VNST'],
       dtype='object'))
```

Check number of numeric (continuous) attributes in the dataset

In [474]:
```
num_df = cars.select_dtypes(include=[np.number])
num_df.shape, num_df.columns
```

Out[474]:
```
((72983, 14),
 Index(['IsBadBuy', 'VehicleAge', 'VehOdo', 'MMRAcquisitionAuctionAveragePrice',
        'MMRAcquisitionAuctionCleanPrice', 'MMRAcquisitionRetailAveragePrice',
        'MMRAcquisitonRetailCleanPrice', 'MMRCurrentAuctionAveragePrice',
        'MMRCurrentAuctionCleanPrice', 'MMRCurrentRetailAveragePrice',
        'MMRCurrentRetailCleanPrice', 'VehBCost', 'IsOnlineSale',
        'WarrantyCost'],
       dtype='object'))
```

There are 13 columns with nominal (categorical) data and 14 with numerical (continuous) data.

In [475]:
```
# Find null and duplicate values.
def findNullValues(df):
    null_nominalcol = []
    null_numcol = []
    null_vals = df.isnull().sum().sort_values()
    df_null = pd.DataFrame({'nullcols':null_vals.index, 'countval':null_vals.value
s})
    df_null = df_null[df_null.countval > 0]
    print ("Null features with values :",df_null)
    print ("Duplicated values :", df_null.duplicated().sum())
    nullcolumns = list(df_null.nullcols)
    null_nominalcol,null_numcol = get_cat_cont_cols(df,df_null.nullcols)
    return null_nominalcol,null_numcol
```

In [476]:
```
null_nominalcols,null_numcols = findNullValues(cars)
```

```
Null features with values :                              nullcols  countval
11                                 Size         5
12                          Nationality         5
13                TopThreeAmericanName         5
14                             SubModel         8
15                                Color         8
16                         Transmission         9
17       MMRAcquisitonRetailCleanPrice        18
18     MMRAcquisitionAuctionCleanPrice        18
19   MMRAcquisitionAuctionAveragePrice        18
20    MMRAcquisitionRetailAveragePrice        18
21       MMRCurrentAuctionAveragePrice       315
22         MMRCurrentAuctionCleanPrice       315
23        MMRCurrentRetailAveragePrice       315
24          MMRCurrentRetailCleanPrice       315
25                               Trim      2360
26                          WheelType      3174
Duplicated values : 0
```

Dropping columns is only advised to be used if missing values are few (say 0.01–0.5% of our data); our data is missing 5,867 values, out of 72,983 records (8%)
Therefore, we will fill missing values (primitive solution)
with appropirate values for categorical and continuos features

In [477]:
```
# Fill NA values with appropirate values for nominal and numeric features
def fillNAvalues(df,null_nominalcols,null_numcols):
    df_nullnominalcol =  df[null_nominalcols].fillna('NA')
    df_nullnumcol = df[null_numcols]
    df_nullnumcol.fillna(df_nullnumcol.mean(),inplace=True)
    #my_imputer = SimpleImputer()
    #imputed_df_x_cont = my_imputer.fit_transform(df_x[nullcontcols])
    colns = list(set(df.columns) - set(null_nominalcols) - set(null_numcols))
    df_nafill = pd.concat([df[colns],df_nullnominalcol,df_nullnumcol],axis=1)
    return df_nafill
```

In [478]:
```
cars = fillNAvalues(cars, null_nominalcols, null_numcols)
```

```
P:\Anaconda\lib\site-packages\pandas\core\generic.py:6287: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._update_inplace(new_data)
```

In [479]:
```
cars.Color[cars.Color == 'NOT AVAIL'] = 'NA'
cars.Color[cars.Color == 'OTHER'] = 'NA'
cars.TopThreeAmericanName[cars.TopThreeAmericanName == 'OTHER'] = 'NA'
```

```
P:\Anaconda\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
P:\Anaconda\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy

P:\Anaconda\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports unti
l
```

In [35]:
```
#build user-defined function
#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 [203]:
```
#cars = fill_missing_values(cars)
```

In [480]:
```
cars.isnull().sum()
```

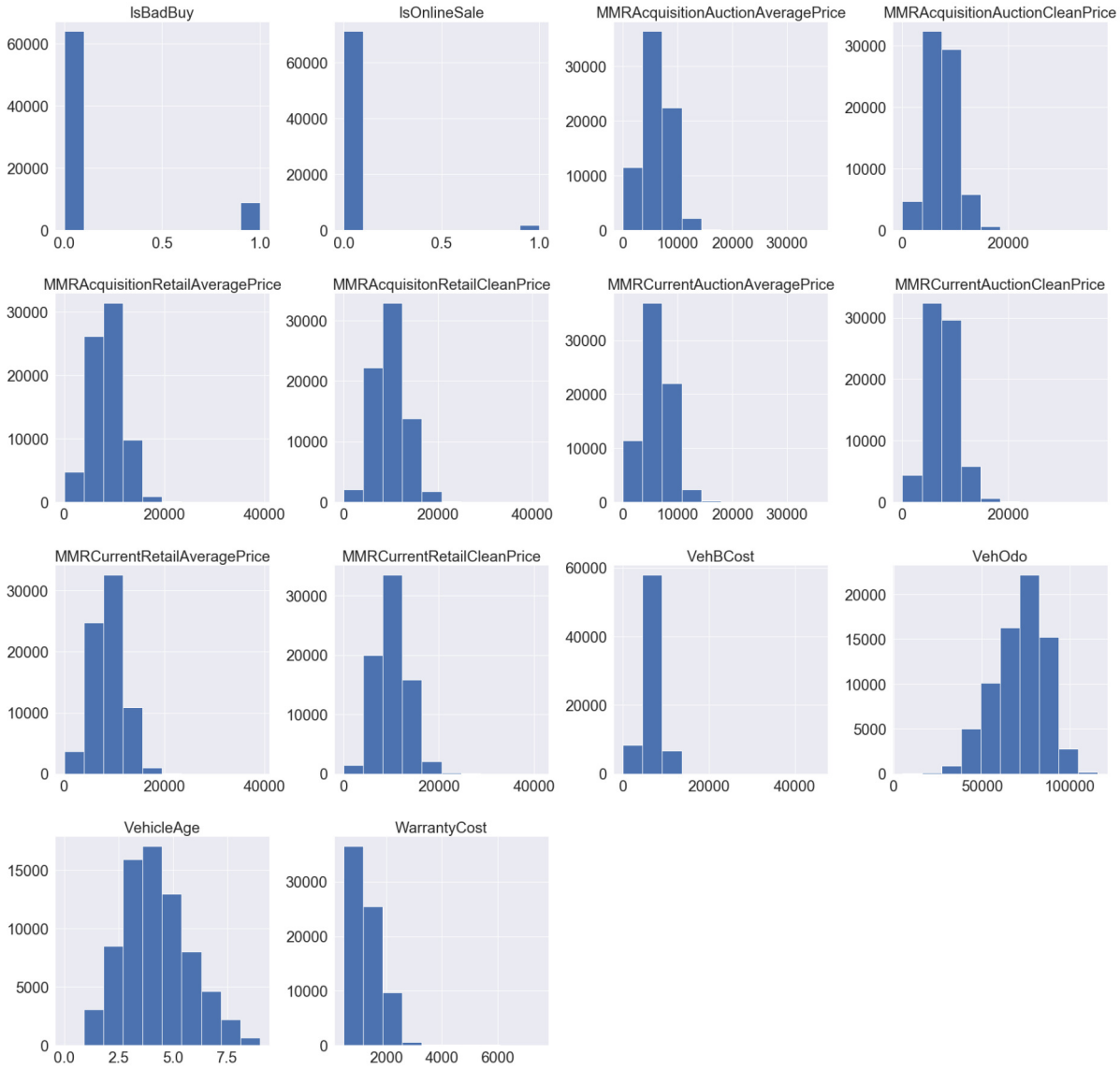
Out[480]:
```
VNST                                0
VehBCost                            0
PurchDate                           0
Auction                             0
IsOnlineSale                        0
IsBadBuy                            0
Make                                0
VehicleAge                          0
WarrantyCost                        0
Model                               0
VehOdo                              0
Size                                0
Nationality                         0
TopThreeAmericanName                0
SubModel                            0
Color                               0
Transmission                        0
Trim                                0
WheelType                           0
MMRAcquisitonRetailCleanPrice       0
MMRAcquisitionAuctionCleanPrice     0
MMRAcquisitionAuctionAveragePrice   0
MMRAcquisitionRetailAveragePrice    0
MMRCurrentAuctionAveragePrice       0
MMRCurrentAuctionCleanPrice         0
MMRCurrentRetailAveragePrice        0
MMRCurrentRetailCleanPrice          0
dtype: int64
```

Missing values have been updated; no missing values are shown in our data above

Data Visualization

Below is the distribution of data on each series, resulting in one histogram per column

In [481]:
```
params = {'axes.titlesize':'24',
          'xtick.labelsize':'24',
          'ytick.labelsize':'24'}
matplotlib.rcParams.update(params)
cars.hist(figsize = (30,30))
plt.show();
```

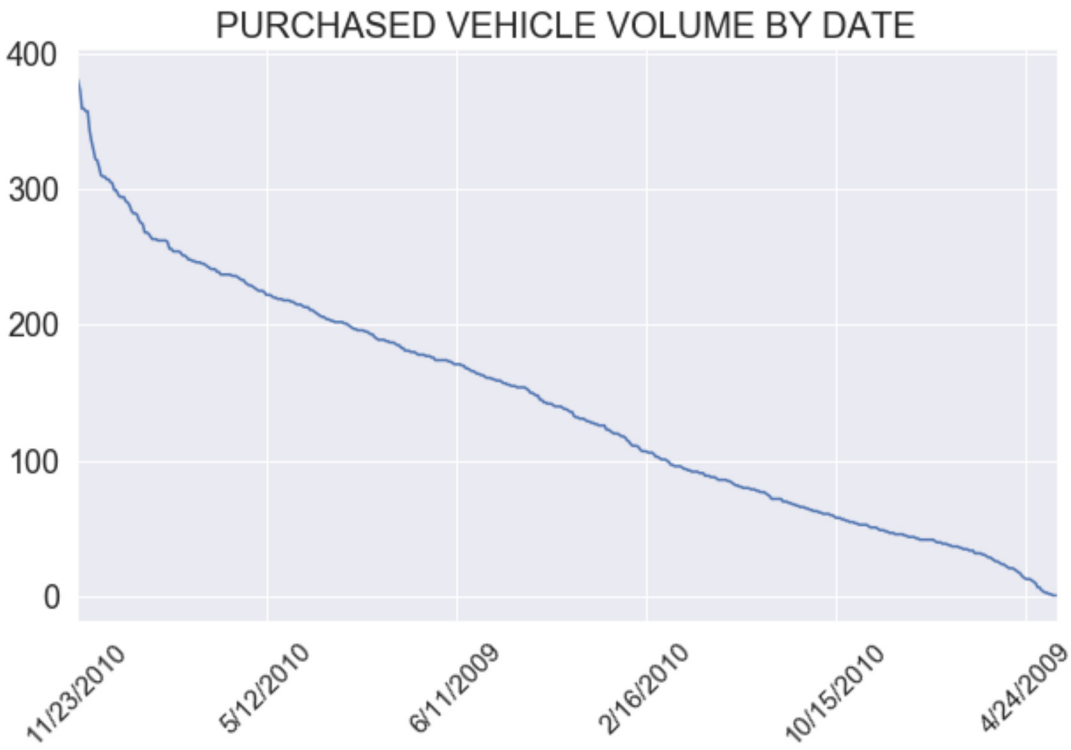


- The original dataset is highly imbalanced. Only about 10% of transactions were a BAD buy, and the rest were GOOD buy.

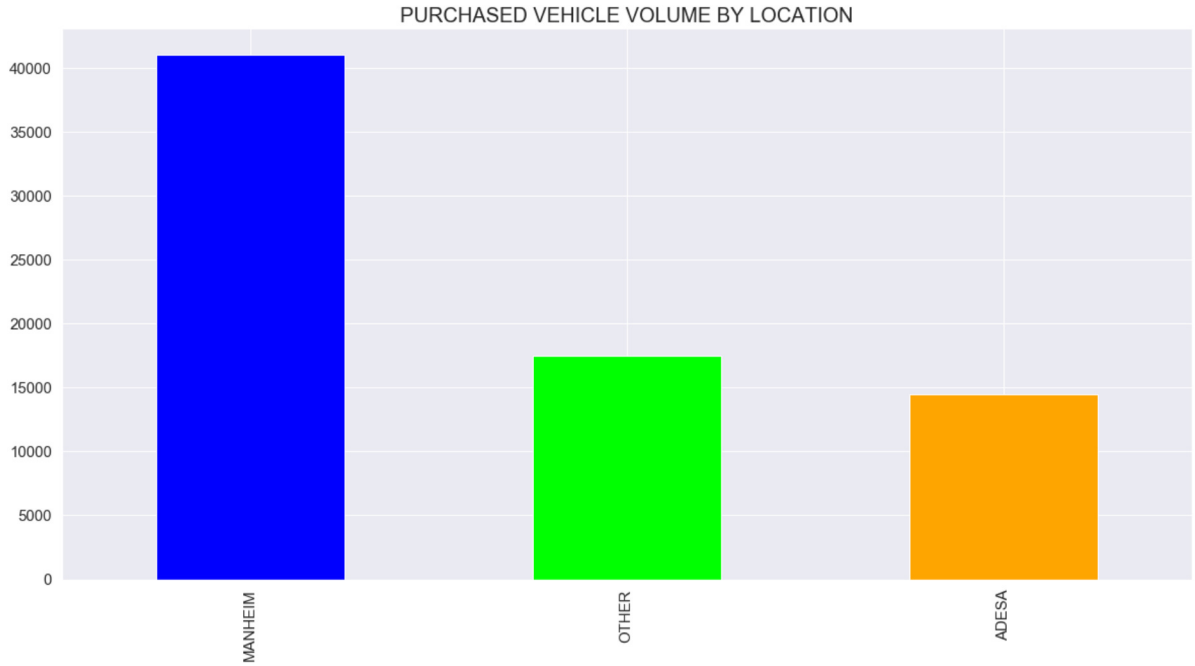
In [482]:
```
cars.select_dtypes(include=['O']).columns.values
```

Out[482]:
```
array(['VNST', 'PurchDate', 'Auction', 'Make', 'Model', 'Size',
       'Nationality', 'TopThreeAmericanName', 'SubModel', 'Color',
       'Transmission', 'Trim', 'WheelType'], dtype=object)
```

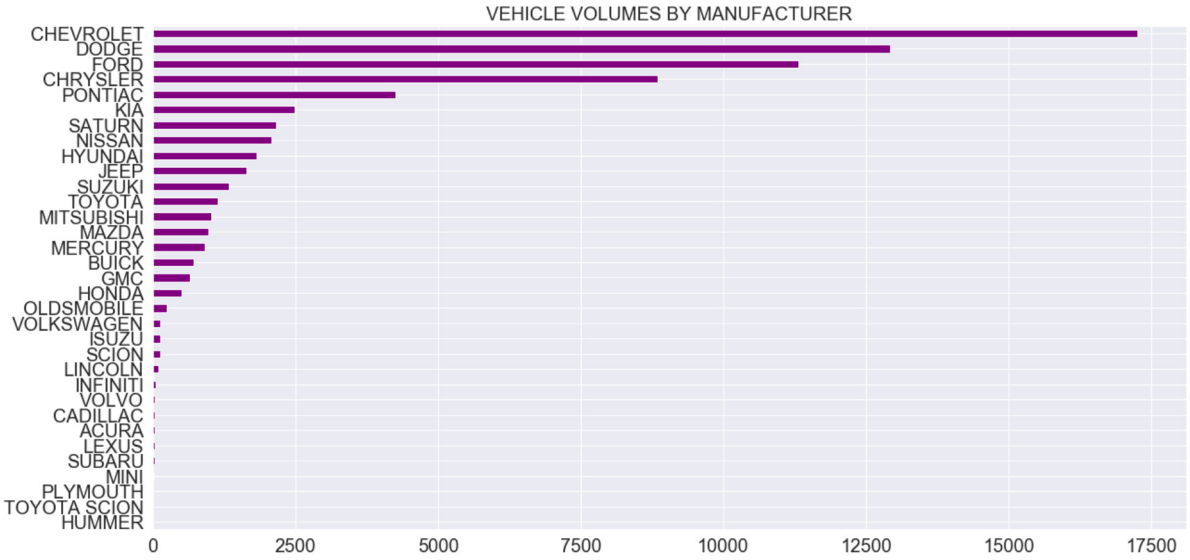
In [483]:
```
cars['PurchDate'].value_counts().sort_values(ascending=False).plot(figsize=(10,6))
plt.title('PURCHASED VEHICLE VOLUME BY DATE', size = 20)
plt.xticks(fontsize=15, rotation=45)
plt.yticks(fontsize=18)
plt.show();
```



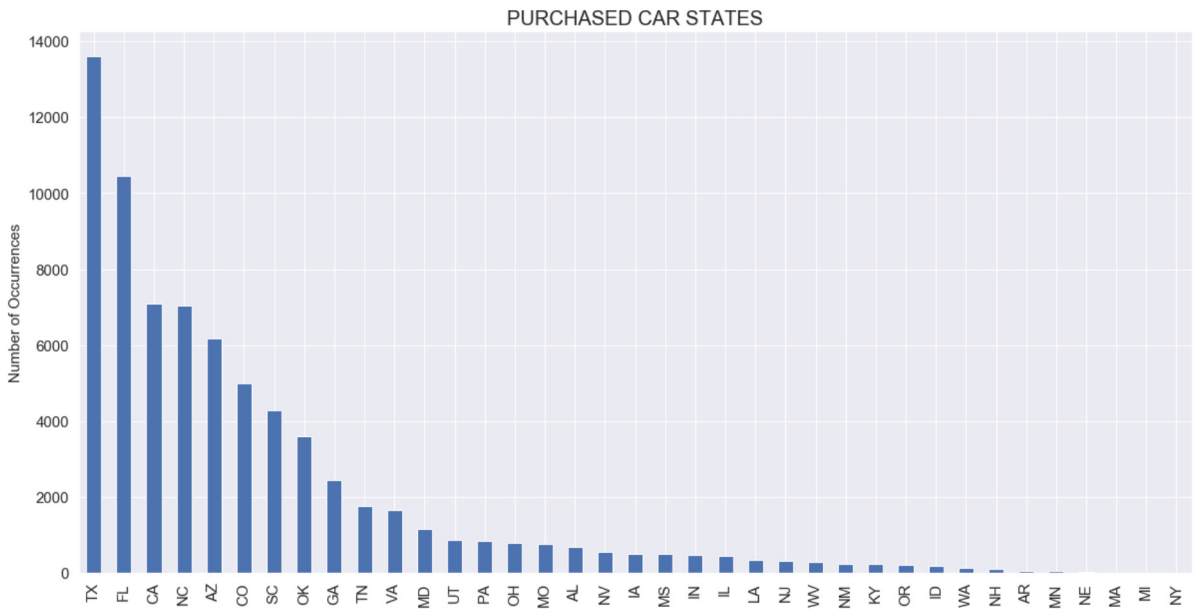
In [484]:
```
colors = ["blue", "lime", "orange"]
cars['Auction'].value_counts().plot(kind='bar', color=colors, figsize=(20,10))
plt.title('PURCHASED VEHICLE VOLUME BY LOCATION', size = 20)
plt.xticks(fontsize=15, rotation=90)
plt.yticks(fontsize=15)
plt.show();
```



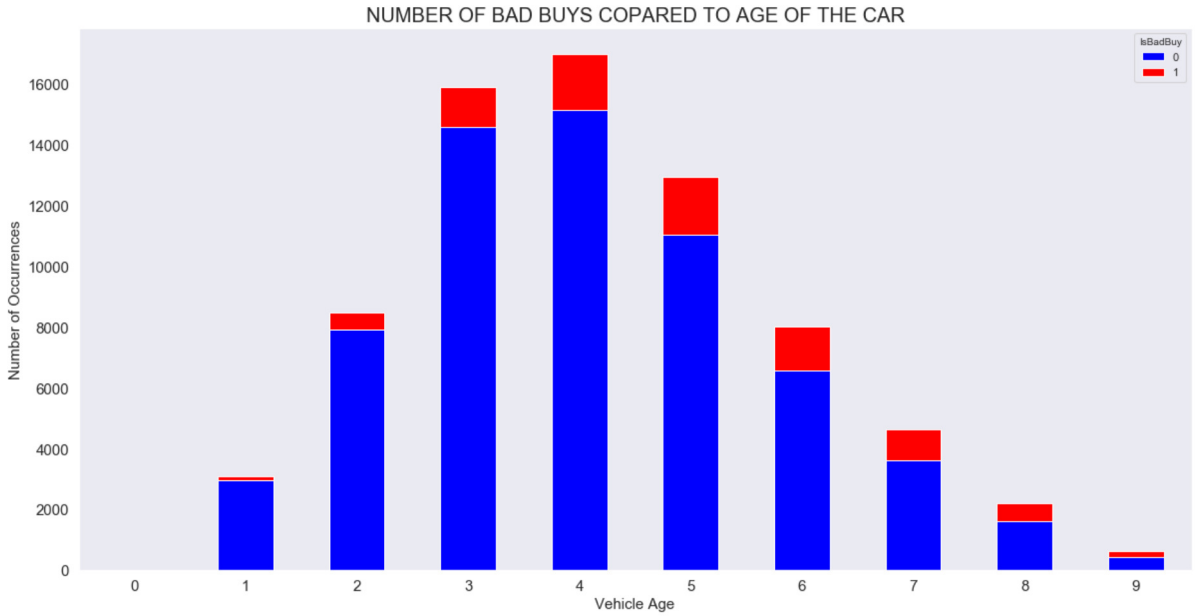
In [485]:
```
colors = ["purple"]
cars['Make'].value_counts().sort_values(ascending=True).plot(kind='barh', color=co
lors, figsize=(20,10))
plt.title('VEHICLE VOLUMES BY MANUFACTURER', size = 20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show();
```



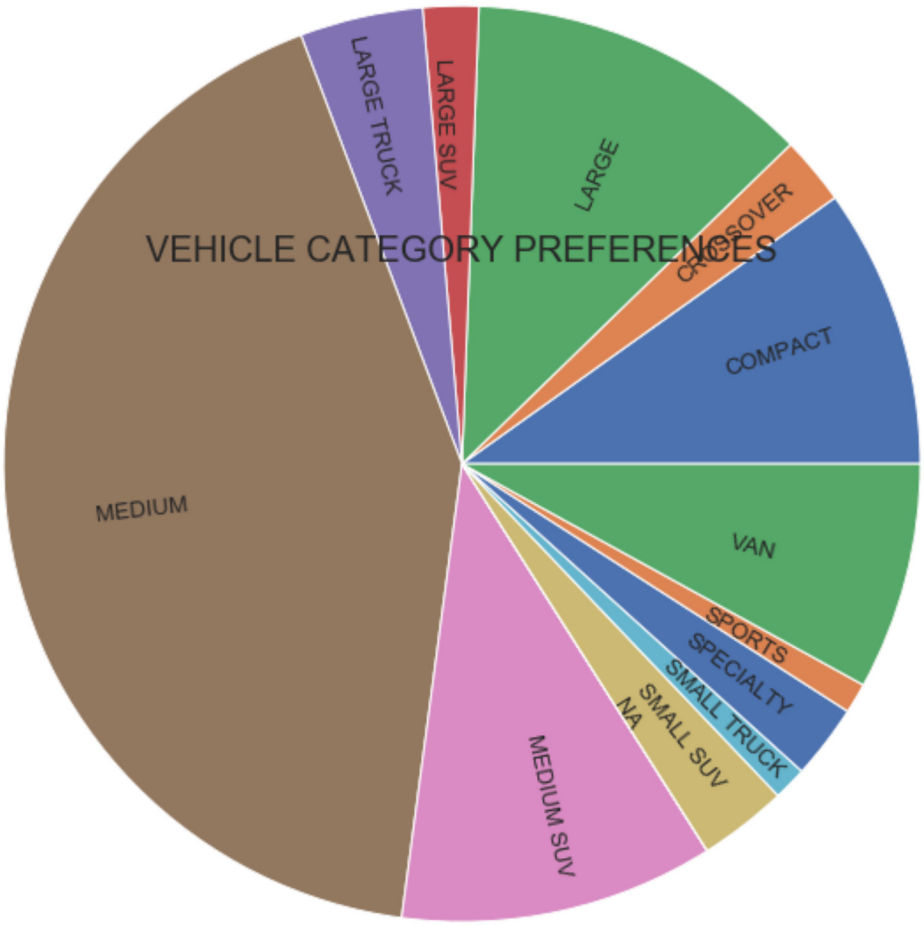
In [486]:
```
state_count = cars['VNST'].value_counts().plot(kind='bar', figsize=(20,10))
#sns.set(style="darkgrid")
#sns.barplot(state_count.index, state_count.values, alpha=0.9, color="limegreen")
#palette="bright"
#sns.set(rc={'figure.figsize':(20,10)})
plt.title('PURCHASED CAR STATES', size=20)
plt.ylabel('Number of Occurrences', size=15)
#plt.xlabel(size=20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```



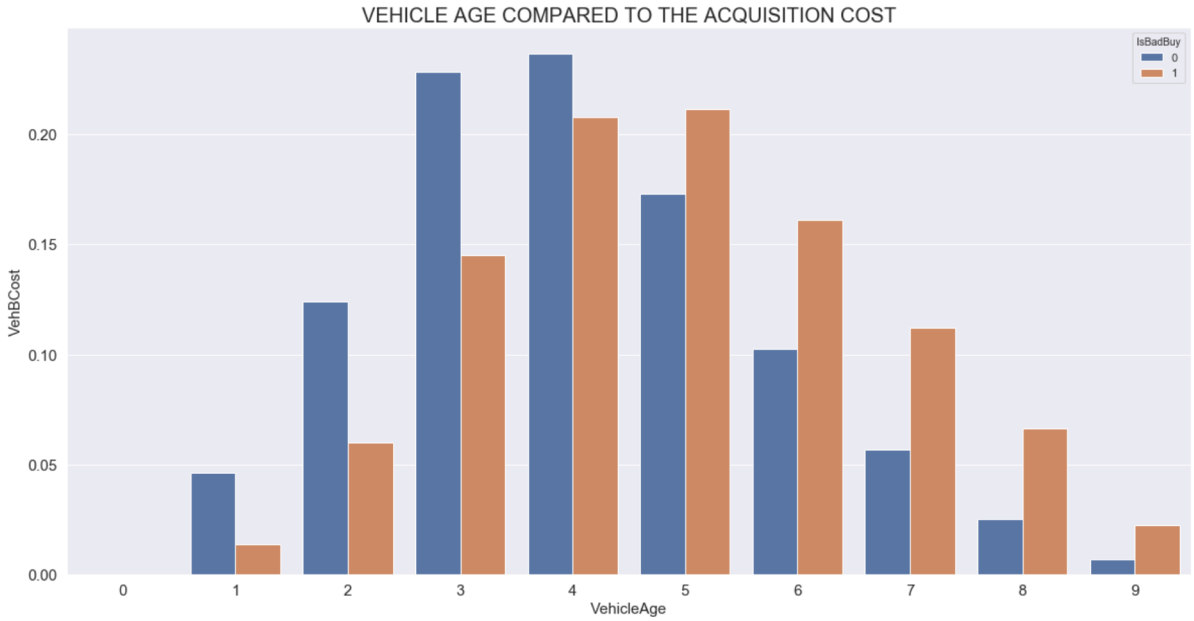
In [487]:
```
var = cars.groupby(['VehicleAge','IsBadBuy']).size()
#.VehBCost.sum()
var.unstack().plot(kind='bar',stacked=True,  color=['blue','red'], grid=False, fig
size=(20,10))
plt.title('NUMBER OF BAD BUYS COPARED TO AGE OF THE CAR', size=20)
plt.ylabel('Number of Occurrences', fontsize=15)
plt.xlabel('Vehicle Age', fontsize=15)
plt.xticks(rotation=0, size=15)
plt.yticks(size=15);
plt.show()
```



In [506]:
```
# Prepare Data
Size = cars.groupby('Size').size()

#fig, ax = plt.subplots(figsize=(6, 3), subplot_kw=dict(aspect="equal"))
Size.plot(kind='pie', subplots=True, radius=3,  labeldistance=0.6, rotatelabels =
270, textprops={'fontsize': 13})
#ax.set_title("VEHICLE CATEGORY PREFERENCES")
plt.title("VEHICLE CATEGORY PREFERENCES", size=20)
plt.show()
```



In [514]:
```
x, y, hue = "VehicleAge", "VehBCost", "IsBadBuy"

plt.figure(figsize=(20, 10))
plt.title('VEHICLE AGE COMPARED TO THE ACQUISITION COST', size=20)
plt.xlabel(x, size=15)
plt.ylabel(y, size=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

(cars[x]
 .groupby(cars[hue])
 .value_counts(normalize=True)
 .rename(y)
 .reset_index()
 .pipe((sns.barplot, "data"), x=x, y=y, hue=hue));
```

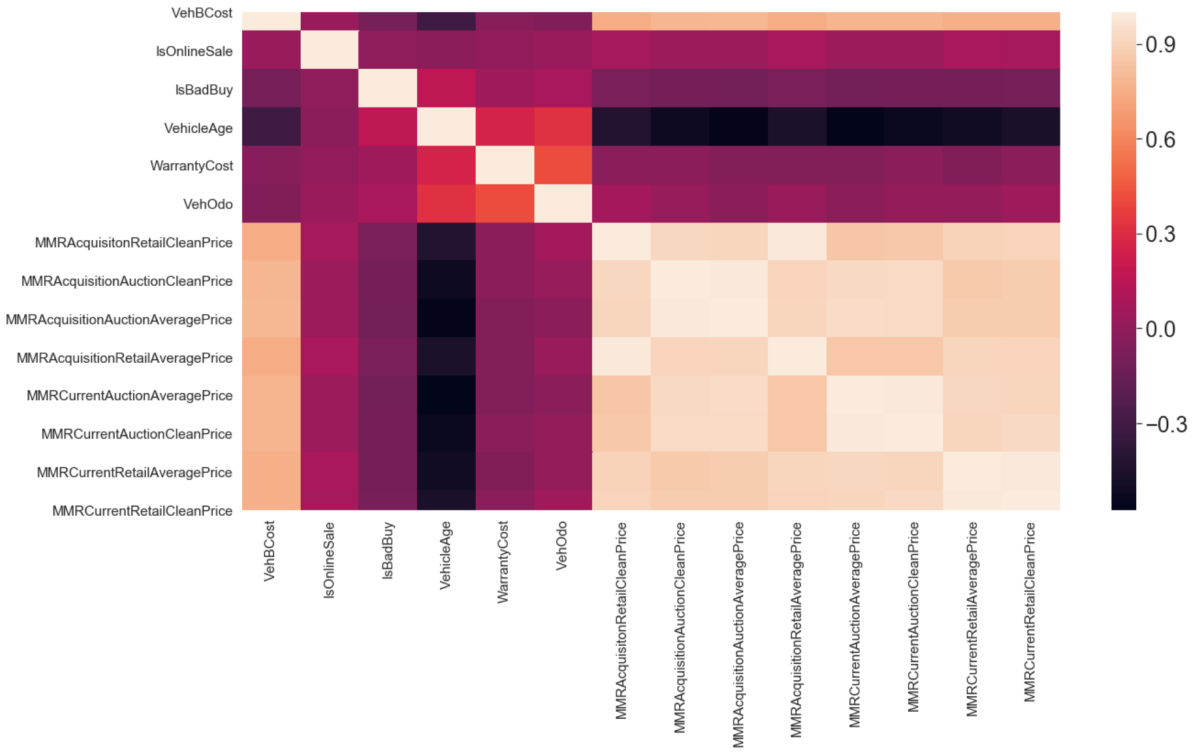


The older vehicle, the greater risk of buying kicked car. Caution should be exercised when purchasing cars older than 4 years.

In [515]:
```
# Correlation matrix for continuous features
def plot_corr_matrix(df):
    print ("Plotting correlation matrix")
    corr = df.corr()
    # plot the heatmap
    fig, ax = plt.subplots(figsize=(20,10))
    sns.heatmap(corr,
            xticklabels=corr.columns,
            yticklabels=corr.columns)
    plt.xlabel("", size=15)
    plt.ylabel("", size=15)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.show()
```

In [374]:
```
plot_corr_matrix(cars)
```

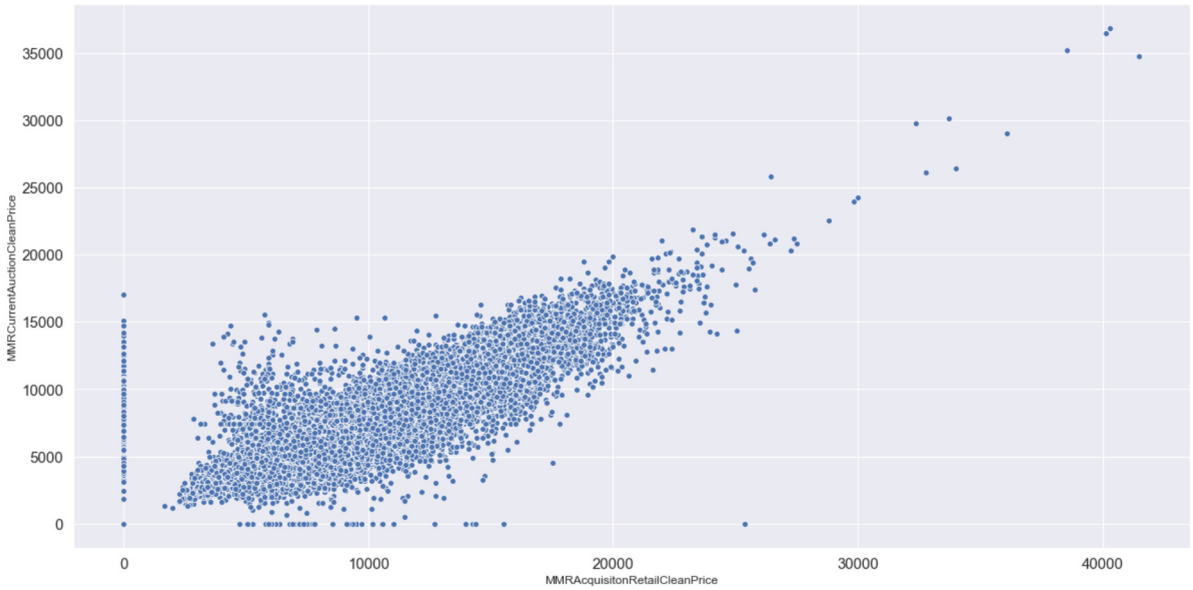
```
Plotting correlation matrix
```

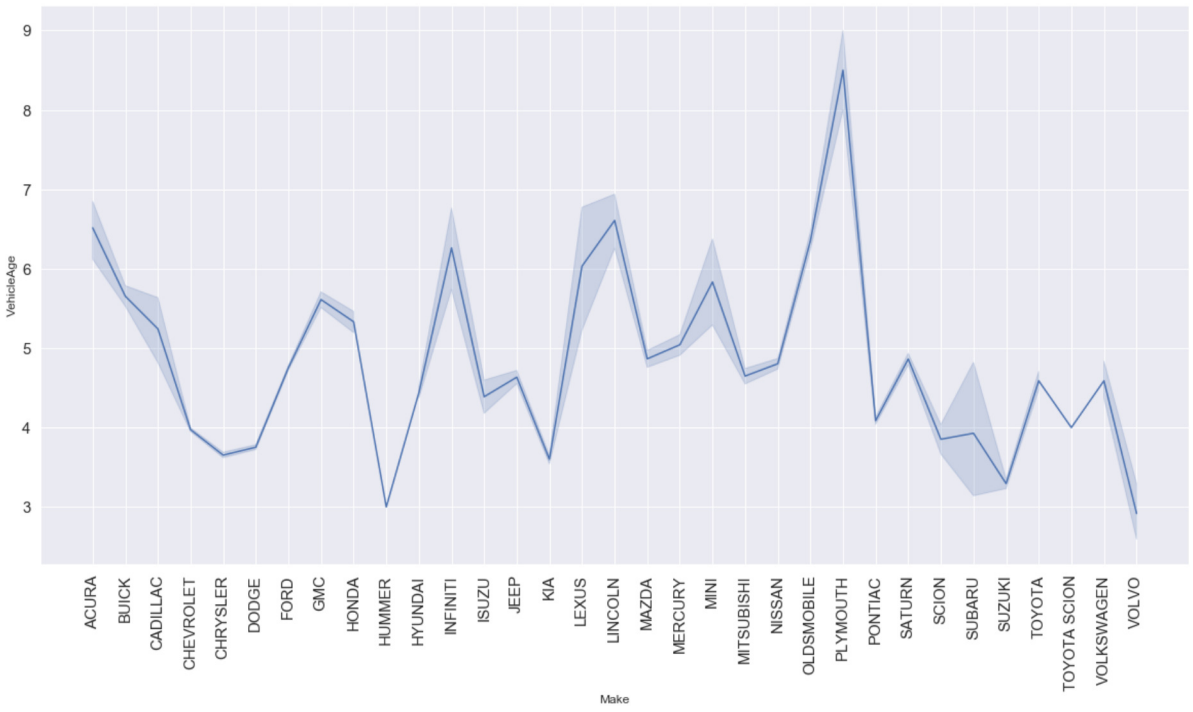


In [516]:
```
# Plotting categorical features
def plot_cat_features(df):
    print ('\033[1m' + "Plotting categorical features")
    plt.figure(figsize = (20, 10))
    sns.scatterplot(x='MMRAcquisitonRetailCleanPrice',y = 'MMRCurrentAuctionCleanPrice',data=df)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.show()

    plt.figure(figsize = (20, 10))
    sns.lineplot(x='Make',y = 'VehicleAge',data=df)
    plt.xticks(fontsize=15, rotation=90)
    plt.yticks(fontsize=15)
    plt.show()
```

In [517]:
```
plot_cat_features(cars)
```

Plotting categorical features

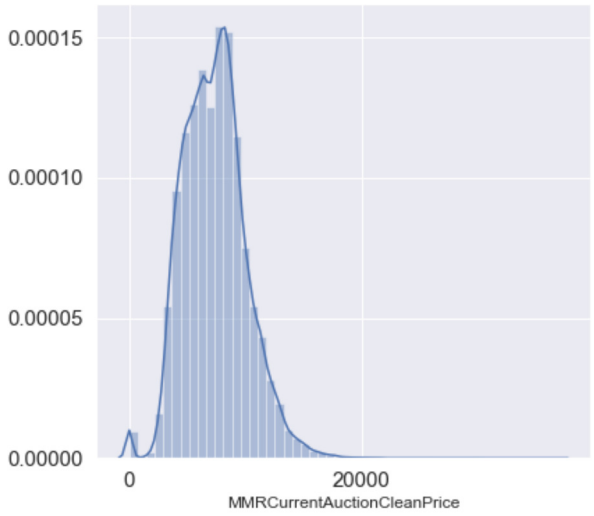


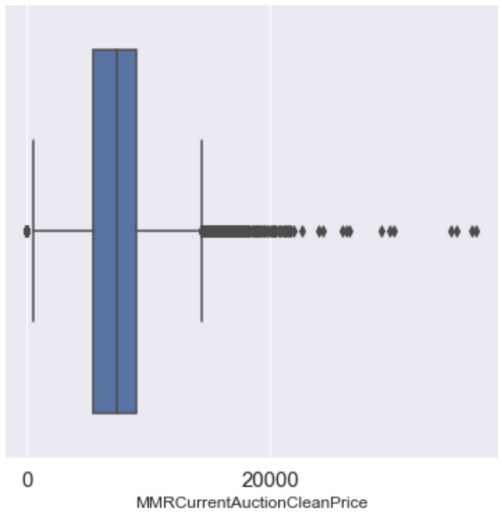


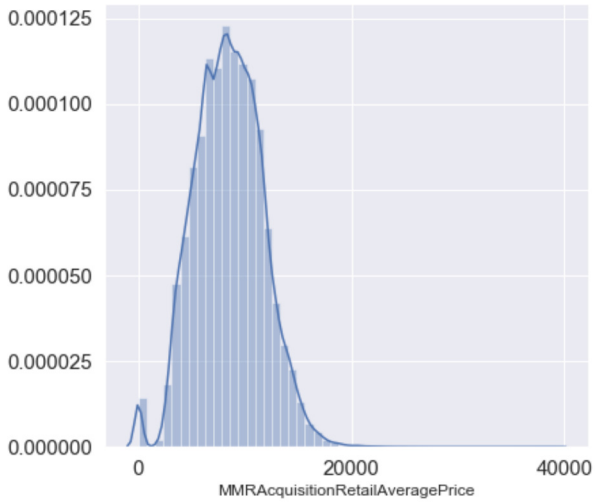
In [518]:
```
# Plotting continuous features
def plot_cont_features(df,num_cols):
    print ('\033[1m' + "Plotting continuous features")
    for i in range (1,len(num_cols)):
        plt.figure(figsize = (14, 6))
        plt.subplot(1, 2, 1)
        sns.distplot(df[num_cols[i]])
        plt.xticks(fontsize=15)
        plt.yticks(fontsize=15)
        plt.subplot(1, 2, 2)
        sns.boxplot(df[num_cols[i]])
        plt.xticks(fontsize=15)
        plt.yticks(fontsize=15)
        plt.show()

    print ('\033[1m' + "Plotting Size vs Warranty cost")
    plt.figure(figsize = (14, 6))
    sns.stripplot(x='Size',y = 'WarrantyCost',data=df)
    plt.xticks(fontsize=10, rotation=45)
    plt.yticks(fontsize=10)
    plt.show()

    print ('\033[1m' + "Plotting Color vs Warranty cost")
    plt.figure(figsize = (14, 6))
    sns.violinplot(x='Color',y = 'WarrantyCost',data=df)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.show()

    plt.figure(figsize = (14, 6))
    sns.boxplot(x='Color',y = 'WarrantyCost',data=df)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.show()

    print ('\033[1m' + "Plotting counts for TopThreeAmericanName")
    plt.figure()
    sns.countplot(x = "TopThreeAmericanName",data=df)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.show()

    print ('\033[1m' + "Plotting Auction vs Warranty cost")
    plt.figure(figsize = (14, 6))
    sns.pointplot(x='Auction',y = 'WarrantyCost',data=df)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.show()

    print ('\033[1m' + "Plotting Color vs VehB Cost")
    plt.figure(figsize = (14, 6))
    sns.barplot(x='Color',y = 'VehBCost',data=df)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.show()

    print ('\033[1m' + "Plotting line plot for VehOdo vs Warranty cost")
    plt.figure(figsize = (14, 6))
    sns.lmplot(x='VehOdo',y = 'WarrantyCost',hue='IsBadBuy',data=df)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.show()
```

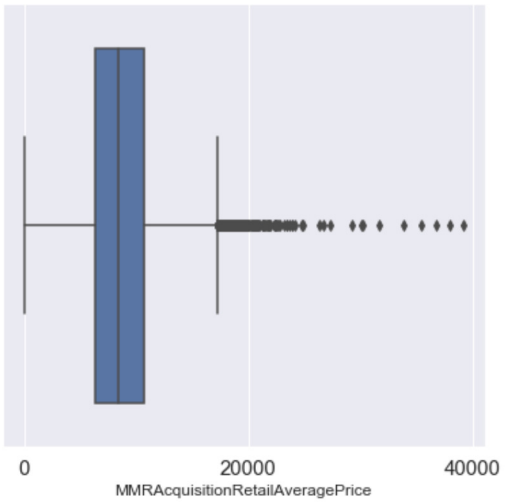
In [519]:
```
plot_cont_features(cars, num_cols)
```

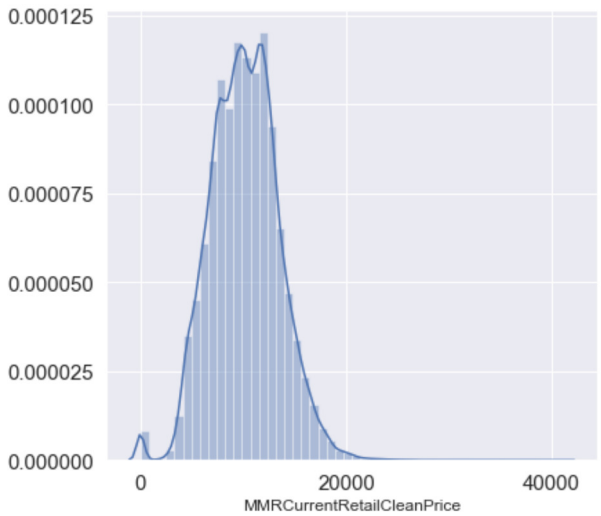
Plotting continuous features

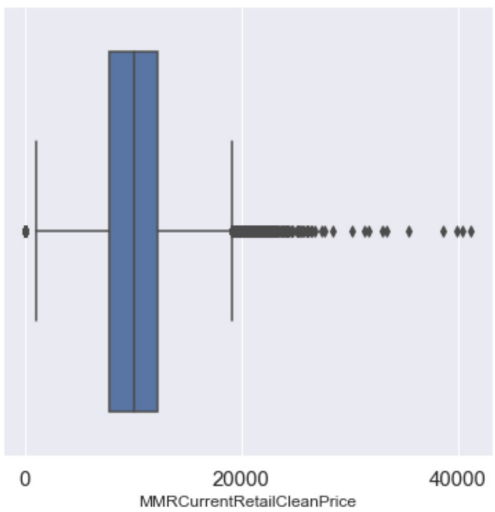


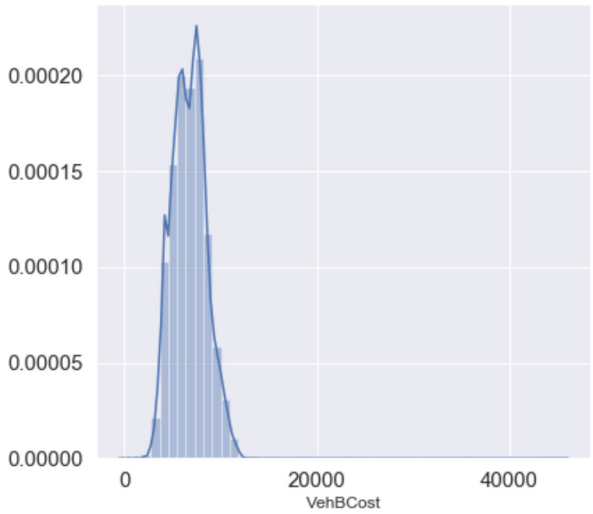


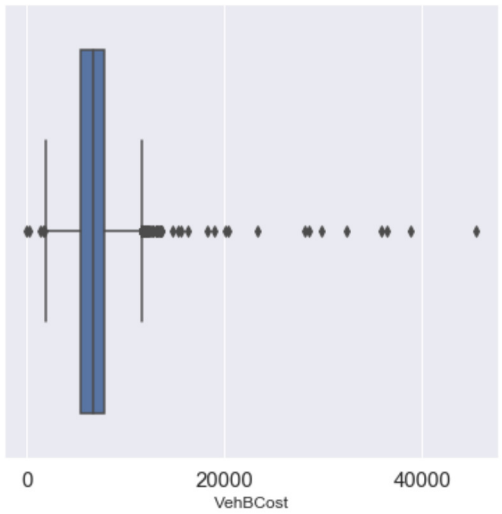


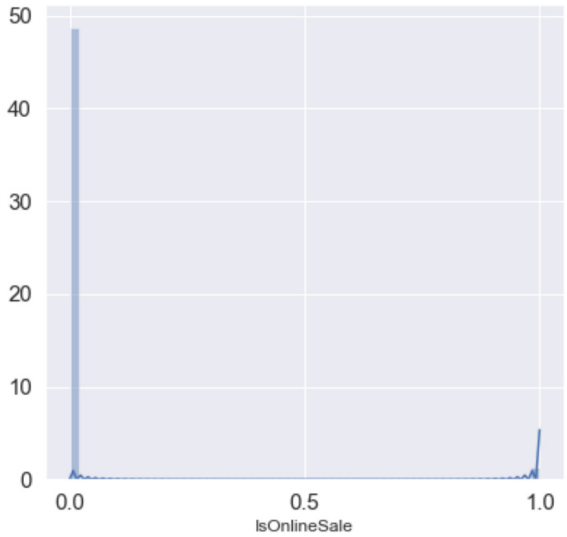


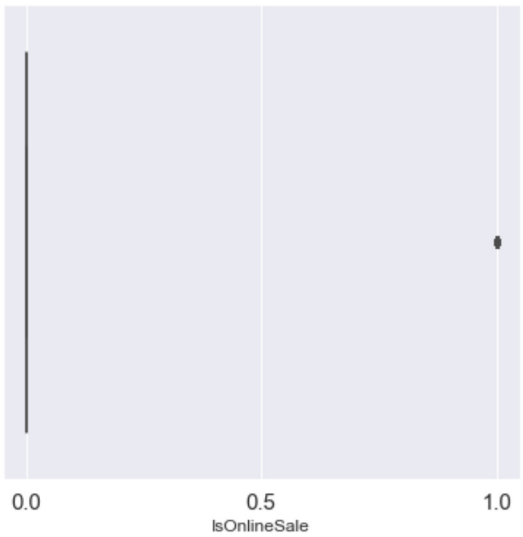


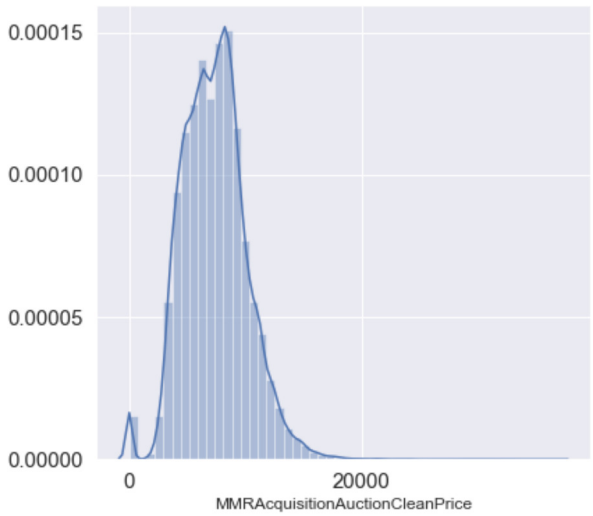


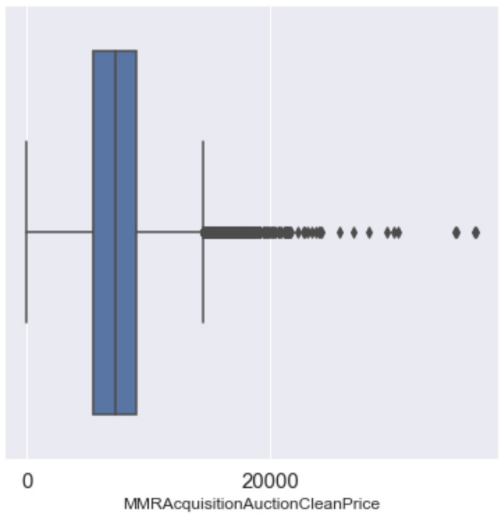

0.00020
0.00015
0.00010
0.00005
0.00000
0
20000
40000
VehBCost

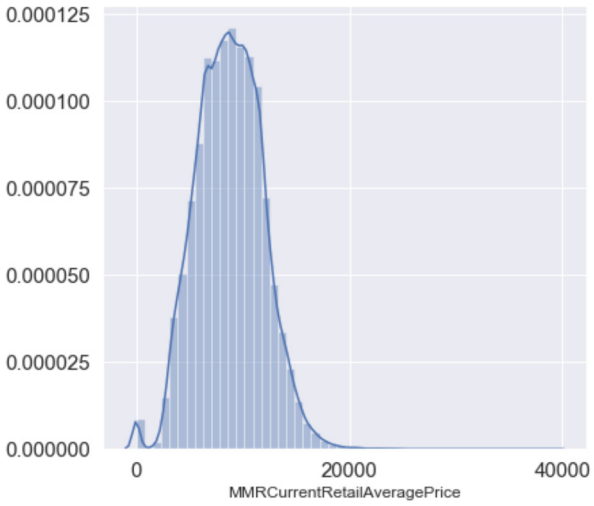

0
20000
40000
VehBCost

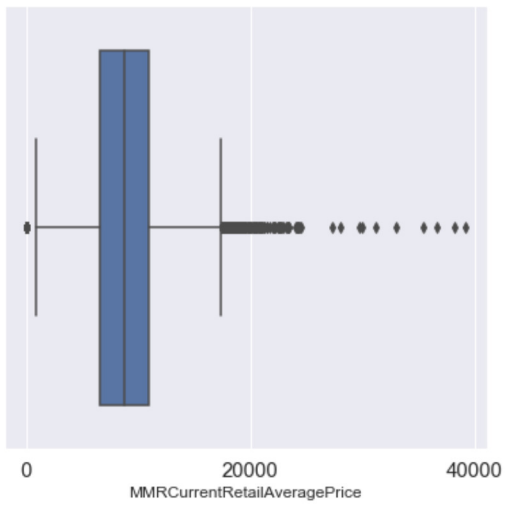

50
40
30
20
10
0
0.0
0.5
1.0
IsOnlineSale

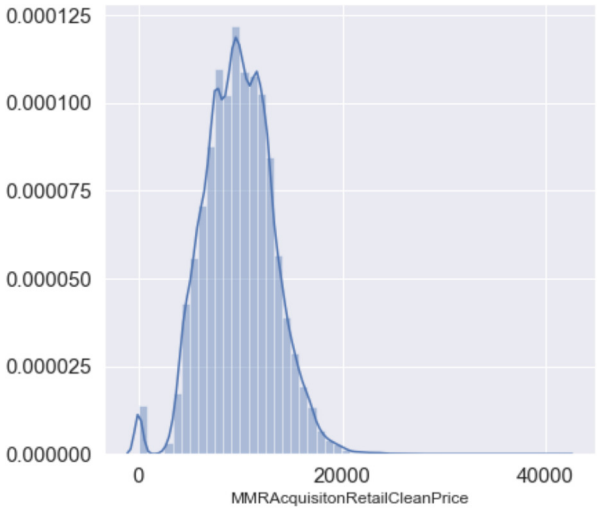

0.0
0.5
1.0
IsOnlineSale

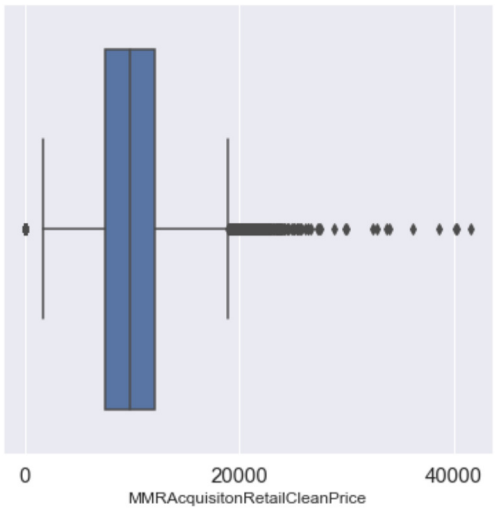

0.00015
0.00010
0.00005
0.00000
0
20000
MMRAcquisitionAuctionCleanPrice

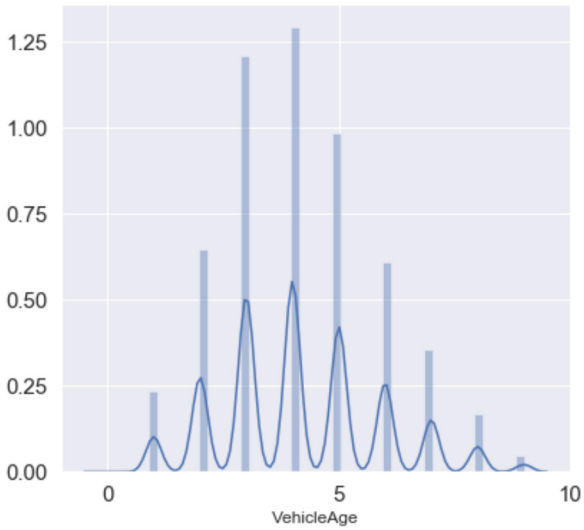

0
20000
MMRAcquisitionAuctionCleanPrice

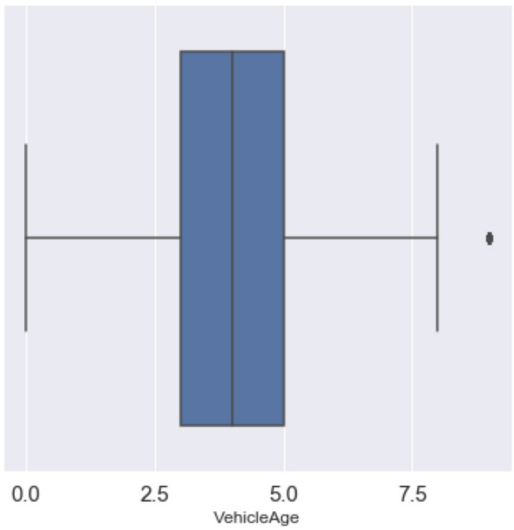

0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
0
20000
40000
MMRCurrentRetailAveragePrice

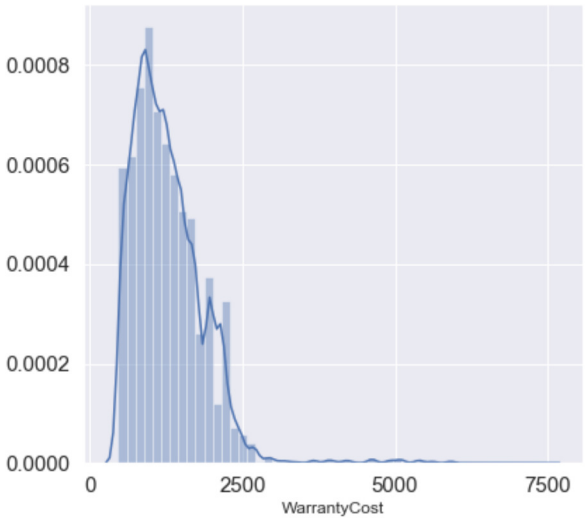

0
20000
40000
MMRCurrentRetailAveragePrice

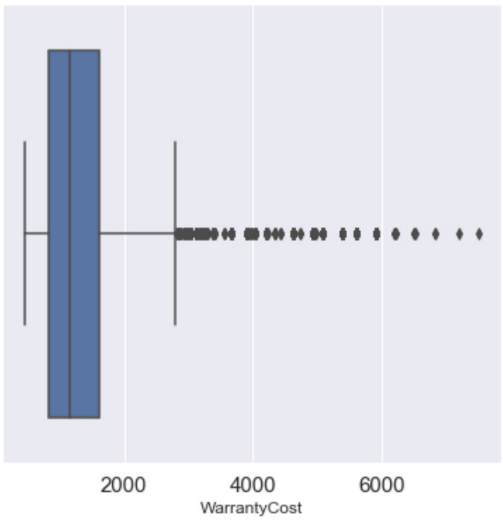

0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
0
20000
40000
MMRAcquisitonRetailCleanPrice

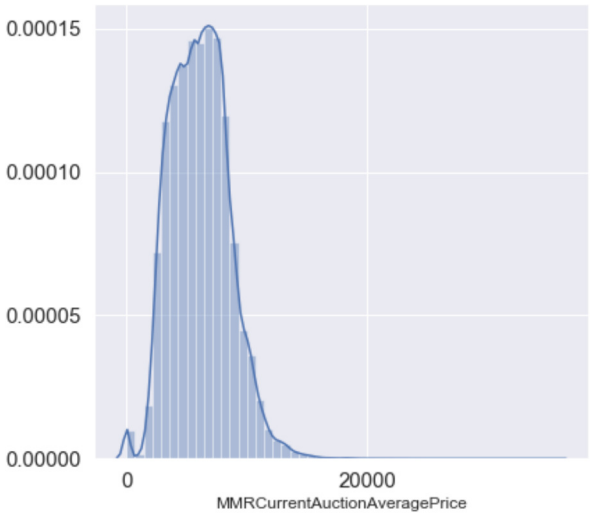

0
20000
40000
MMRAcquisitonRetailCleanPrice

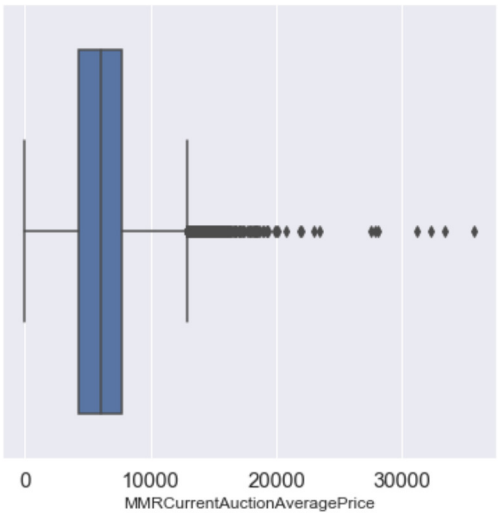

1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
VehicleAge

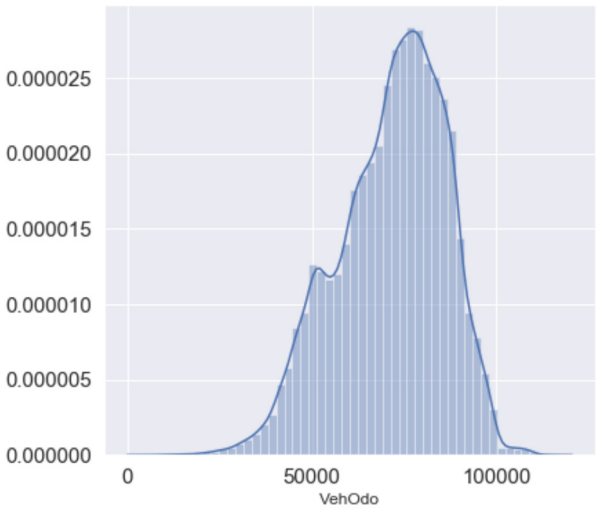

0.0
2.5
5.0
7.5
VehicleAge

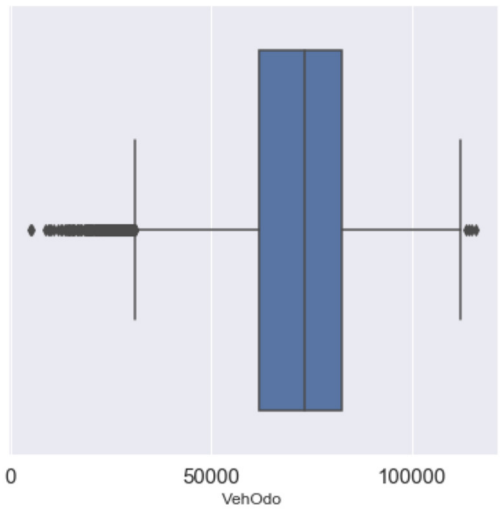




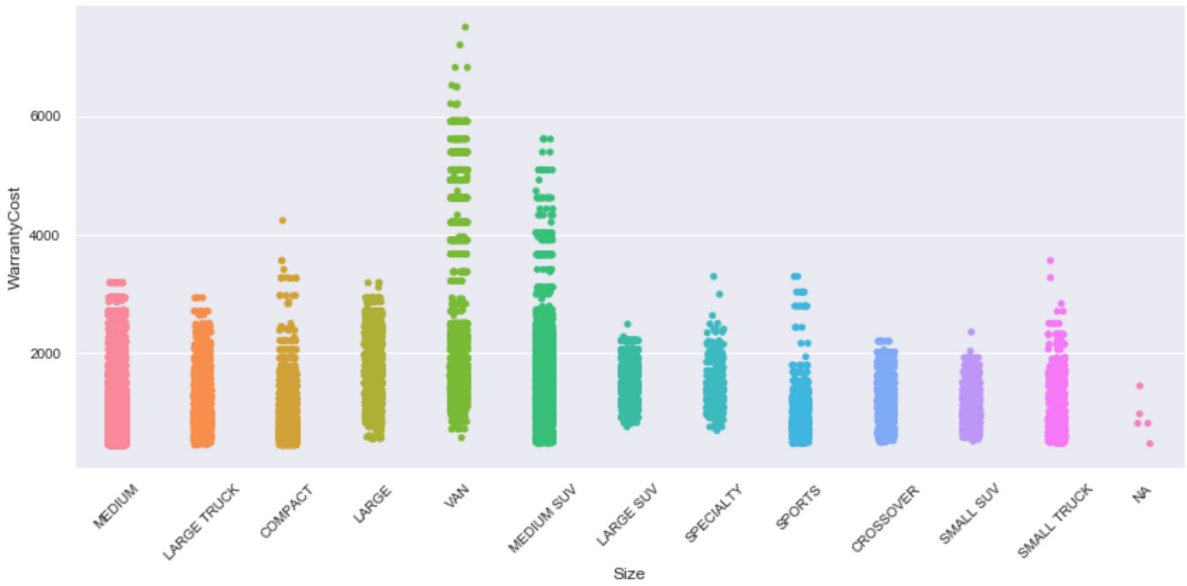




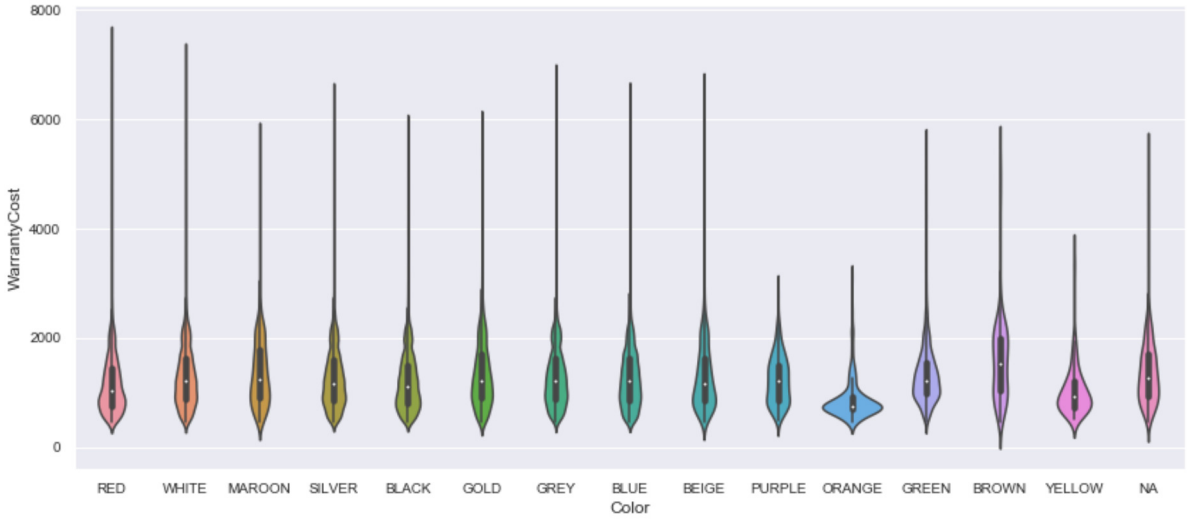


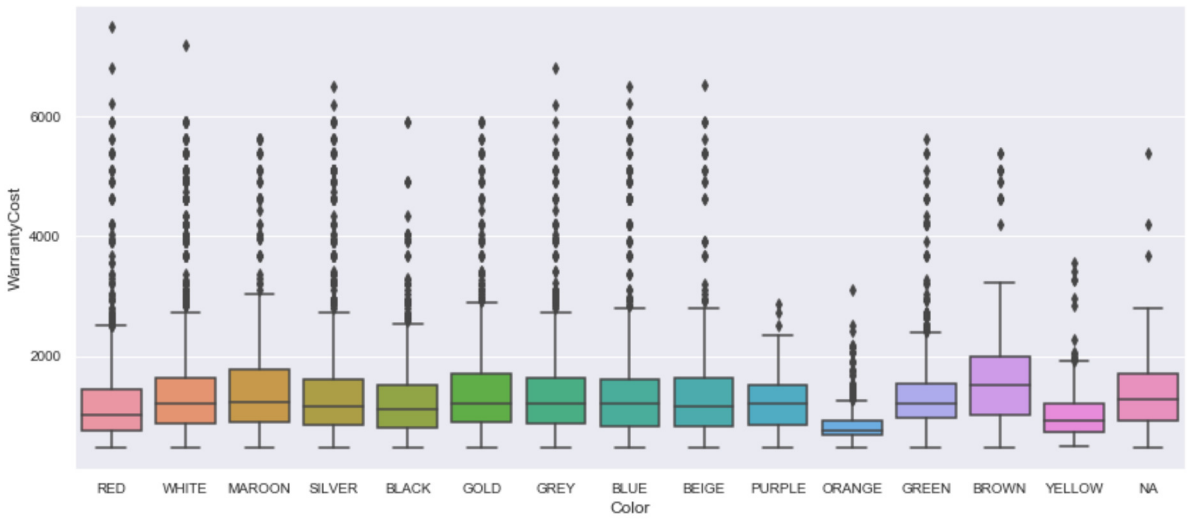


Plotting Size vs Warranty cost

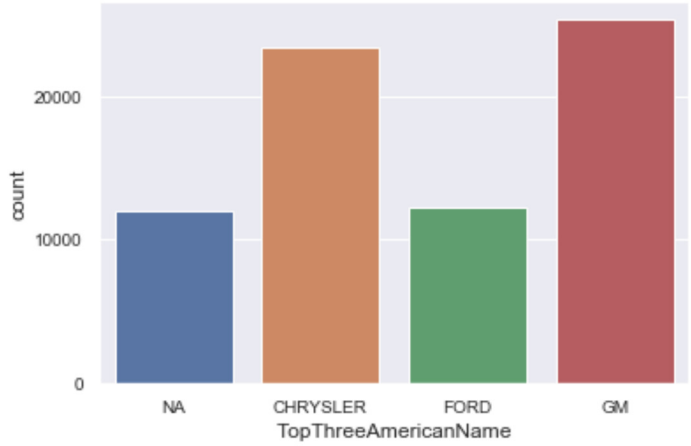


Plotting Color vs Warranty cost

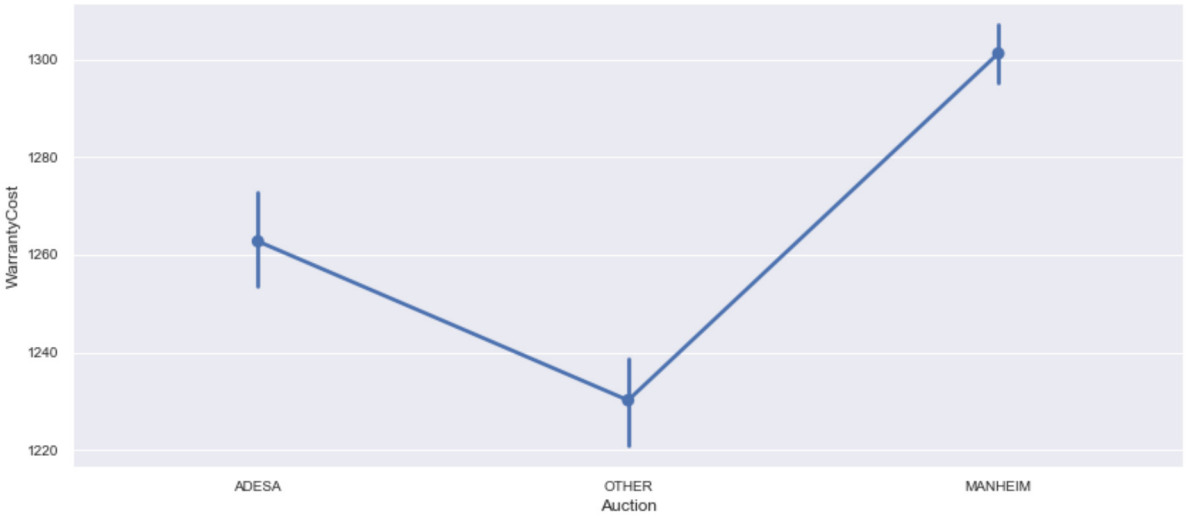




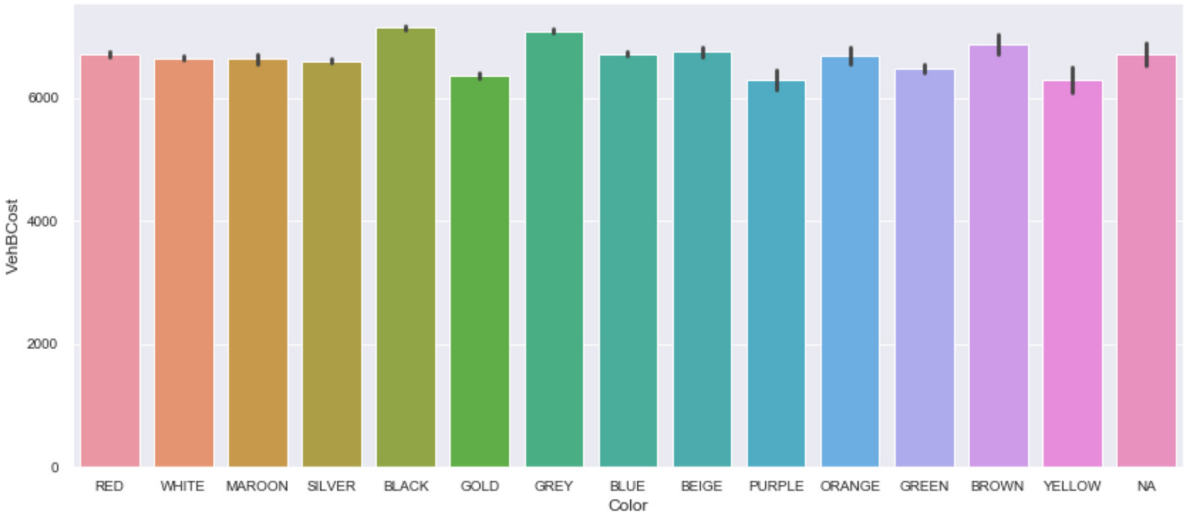
Plotting counts for TopThreeAmericanName



Plotting Auction vs Warranty cost

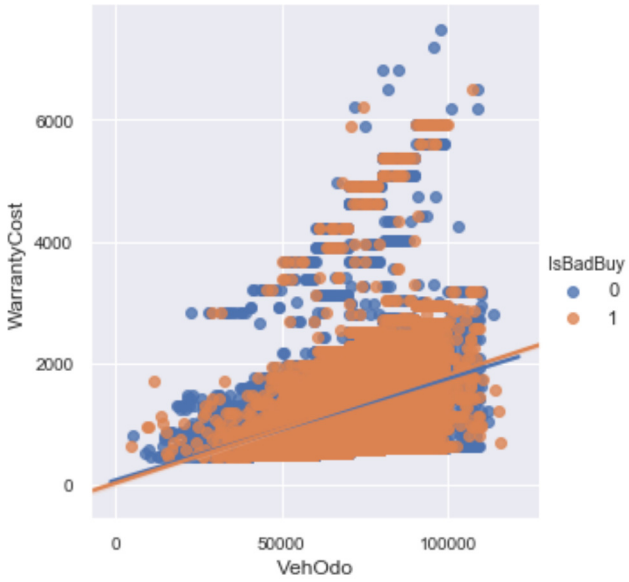


Plotting Color vs VehB Cost



Plotting line plot for VehOdo vs Warranty cost

```
<Figure size 1008x432 with 0 Axes>
```



Transform Data, Tuning Machine Learning Model and Evaluation

In [520]:
```
# Find outliers in numeric (continuous) features and normalize all features into l
og scale
def get_outliers_scale(df,cols):
    for col in cols:
        stat = df[col].describe()
        # print(stat)
        IQR = stat['75%'] - stat['25%']
        upper = stat['75%'] + 1.5 * IQR
        lower = stat['25%'] - 1.5 * IQR
        print('The upper and lower bounds of {} for suspected outliers are {} and
{}.'.format(col,upper,lower))
        print ("Values less than lower bound :" , len(df[df[col] < lower]))
        print ("Values greater than upper  bound : ", len(df[df[col] > upper]))
# converting to logarithmic scale (log scale) to transform data before modeling
        df[col]=np.log1p(df[col])
    return df[cols]
```

In [521]:
```
df_num = get_outliers_scale(cars, num_cols)
```

```
The upper and lower bounds of MMRAcquisitionAuctionAveragePrice for suspected ou
tliers are 13003.0 and -965.0.
Values less than lower bound : 0
Values greater than upper  bound :  526
The upper and lower bounds of MMRCurrentAuctionCleanPrice for suspected outliers
are 14377.5 and 53.5.
Values less than lower bound : 504
Values greater than upper  bound :  846
The upper and lower bounds of MMRAcquisitionRetailAveragePrice for suspected out
liers are 17203.5 and -272.5.
Values less than lower bound : 0
Values greater than upper  bound :  318
The upper and lower bounds of MMRCurrentRetailCleanPrice for suspected outliers
are 19076.0 and 1028.0.
Values less than lower bound : 504
Values greater than upper  bound :  433
The upper and lower bounds of VehBCost for suspected outliers are 11597.5 and 17
37.5.
Values less than lower bound : 5
Values greater than upper  bound :  191
The upper and lower bounds of IsOnlineSale for suspected outliers are 0.0 and 0.
0.
Values less than lower bound : 0
Values greater than upper  bound :  1845
The upper and lower bounds of MMRAcquisitionAuctionCleanPrice for suspected outl
iers are 14442.0 and -14.0.
Values less than lower bound : 0
Values greater than upper  bound :  841
The upper and lower bounds of MMRCurrentRetailAveragePrice for suspected outlier
s are 17417.5 and 29.5.
Values less than lower bound : 504
Values greater than upper  bound :  304
The upper and lower bounds of MMRAcquisitonRetailCleanPrice for suspected outlie
rs are 18979.0 and 603.0.
Values less than lower bound : 828
Values greater than upper  bound :  433
The upper and lower bounds of VehicleAge for suspected outliers are 8.0 and 0.0.
Values less than lower bound : 0
Values greater than upper  bound :  646
The upper and lower bounds of WarrantyCost for suspected outliers are 2802.0 and
-342.0.
Values less than lower bound : 0
Values greater than upper  bound :  838
The upper and lower bounds of MMRCurrentAuctionAveragePrice for suspected outlie
rs are 12902.5 and -885.5.
Values less than lower bound : 0
Values greater than upper  bound :  582
The upper and lower bounds of VehOdo for suspected outliers are 113334.5 and 309
38.5.
Values less than lower bound : 332
Values greater than upper  bound :  4
```

In [522]:
```
df_num.head()
```

Out[522]:

	MMRAcquisitionAuctionAveragePrice	MMRCurrentAuctionCleanPrice	MMRAcquisitionRetailAveragePrice	MMR
0	9.006509	9.054037	9.361945	
1	8.832734	9.129456	9.296335	
2	8.071843	8.622994	8.845633	
3	7.546446	7.881182	8.446556	
4	8.272315	8.385945	8.952088	

In [523]:
```
# Encoding categorical features into numerical values.
# Convert each class under specified feature to a numerical value
def encode_label(df,cols):
    le = preprocessing.LabelEncoder()
    for col in cols:
        df[col] = np.log1p(le.fit_transform(df[col]))
    return df[cols]
```

In [524]:
```
df_nominal = encode_label(cars, nominal_cols)
```

In [525]:
```
df_nominal.head()
```

Out[525]:

	PurchDate	Auction	Make	Model	Trim	SubModel	Color	Transmission	WheelType	Nationali
0	5.105945	0.0	2.890372	6.375025	4.897840	5.402677	2.484907	0.000000	0.000000	1.38629
1	5.105945	0.0	1.791759	0.000000	4.553877	6.642487	2.639057	0.000000	0.000000	0.00000
2	5.105945	0.0	1.791759	6.784457	4.605170	5.680173	2.079442	0.000000	0.693147	0.00000
3	5.105945	0.0	1.791759	6.496775	4.605170	5.030438	2.564949	0.000000	0.000000	0.00000
4	5.105945	0.0	1.945910	5.910797	4.852030	3.970292	2.564949	0.693147	0.693147	0.00000

In [526]:
```
# Merge dataframes with encoded features and target
df_train = pd.concat([df_nominal,df_num,cars[target]],axis=1)
```

In [527]:
```
# Partition data into a training set and a test set
# Percentage of the data that should be held over for testing: 80/20
x_train, x_test, y_train, y_test = train_test_split(df_train[features],df_train[ta
rget],test_size=0.2,random_state=7)
```

In [528]:
```
# Define the Logistic Regression, Random Forest, XGBoost and LightGBM models.
# The AUC value lies between 0.5 to 1 where 0.5 denotes a bad classifer and 1 deno
tes an excellent classifier.
def create_tune_models(x_train,y_train):
    models = {}
    lr = LogisticRegression(class_weight='balanced',random_state=31)
    rf = RandomForestClassifier(n_estimators=75,max_features=5,max_depth=20,
                                min_samples_split=100,class_weight='balanced',rand
om_state=37)
    #xg = xgb.XGBClassifier(objective ='binary:logistic', colsample_bytree = 0.3,
    #                       learning_rate = 0.1,max_depth = 5, alpha = 10, n_estim
ators = 10,random_state=41)
    #lgtbm = lgbm.LGBMClassifier(boosting_type='gbdt', objective='binary',
    #               num_iteration=1000,num_leaves=31,
    #               is_enable_sparse='true',tree_learner='data',min_data_in_leaf=6
00,max_depth=4,
    #               learning_rate=0.01, max_bin=255, subsample_for_bin=5000,
    #               min_split_gain=5, min_child_weight=5, min_child_samples=10, su
bsample=0.995,
    #               subsample_freq=1, colsample_bytree=1, reg_alpha=0,
    #               reg_lambda=0, seed=0, nthread=-1, silent=True,random_state=43)
    models["LogisticRegression"] = lr
    models["RandomForest"] = rf
    #models["XGBoost"] = xg
    #models["LightGBM"] = lgtbm
    for k,v in models.items():
        v.fit(x_train, y_train)
        models[k]=v
    return models
```

In [529]:
```
models = create_tune_models(x_train, y_train)
```

```
P:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarnin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silenc
e this warning.
  FutureWarning)
P:\Anaconda\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWar
ning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
P:\Anaconda\lib\site-packages\ipykernel_launcher.py:22: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
```

In [530]:
```
# Train the models on the 10-Fold Cross Validation and calculate area under the cu
rve (AUC).
# Used in classification analysis in order to determine which of the used models p
redicts the classes best.
# AUC of a classifier is equal to the probability that the classifier will rank a
randomly chosen positive example higher than a randomly chosen negative example
def cv_models(models,x_train,y_train,kfold,metric):
    model_auc = {}
    for k,v in models.items():
        model_results = model_selection.cross_val_score(v, x_train,y_train, cv=kfo
ld, scoring=metric)
        mean_auc = model_results.mean()
        std = model_results.std()
        # print out the mean and standard deviation of the training score
        print('The model {} has AUC {} and STD {}.'.format(k, mean_auc, std))
        model_auc[k] = mean_auc
    return model_auc
```

In [531]:
```
# K-Folds cross-validator
# Provides train/test indices to split data in train/test sets. Split dataset into
k consecutive folds (without shuffling by default).
# Each fold is then used once as a validation while the k - 1 remaining folds form
the training set.
kfold = model_selection.KFold(n_splits=10)
```

In [532]:
```
# Designate a scorer object
# All scorer objects follow the convention that higher return values are better th
an lower return values.
# metrics.roc_auc_score: Compute Area Under the Receiver Operating Characteristic
Curve (ROC AUC) from prediction scores.
metric = 'roc_auc'
```

In [533]:
```
model_auc = cv_models(models, x_train, y_train, kfold, metric)
```

```
P:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarnin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silenc
e this warning.
  FutureWarning)
P:\Anaconda\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWar
ning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
P:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarnin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silenc
e this warning.
  FutureWarning)
P:\Anaconda\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWar
ning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
P:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarnin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silenc
e this warning.
  FutureWarning)
P:\Anaconda\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWar
ning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
P:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarnin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silenc
e this warning.
  FutureWarning)
P:\Anaconda\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWar
ning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
P:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarnin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silenc
e this warning.
  FutureWarning)
P:\Anaconda\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWar
ning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
P:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarnin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silenc
e this warning.
  FutureWarning)
P:\Anaconda\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWar
ning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
P:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarnin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silenc
e this warning.
  FutureWarning)
P:\Anaconda\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWar
ning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
P:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarnin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silenc
e this warning.
  FutureWarning)
P:\Anaconda\lib\site-packages\sklearn\utils\validation.py:724: DataConversionWar
ning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
The model LogisticRegression has AUC 0.6925498200636218 and STD 0.01068585459742
442.
```

```
P:\Anaconda\lib\site-packages\sklearn\model_selection\_validation.py:516: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
P:\Anaconda\lib\site-packages\sklearn\model_selection\_validation.py:516: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
P:\Anaconda\lib\site-packages\sklearn\model_selection\_validation.py:516: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
P:\Anaconda\lib\site-packages\sklearn\model_selection\_validation.py:516: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
P:\Anaconda\lib\site-packages\sklearn\model_selection\_validation.py:516: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
P:\Anaconda\lib\site-packages\sklearn\model_selection\_validation.py:516: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
P:\Anaconda\lib\site-packages\sklearn\model_selection\_validation.py:516: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
P:\Anaconda\lib\site-packages\sklearn\model_selection\_validation.py:516: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
P:\Anaconda\lib\site-packages\sklearn\model_selection\_validation.py:516: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
P:\Anaconda\lib\site-packages\sklearn\model_selection\_validation.py:516: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
```

```
The model RandomForest has AUC 0.7592774072568834 and STD 0.00766712417795856.
```

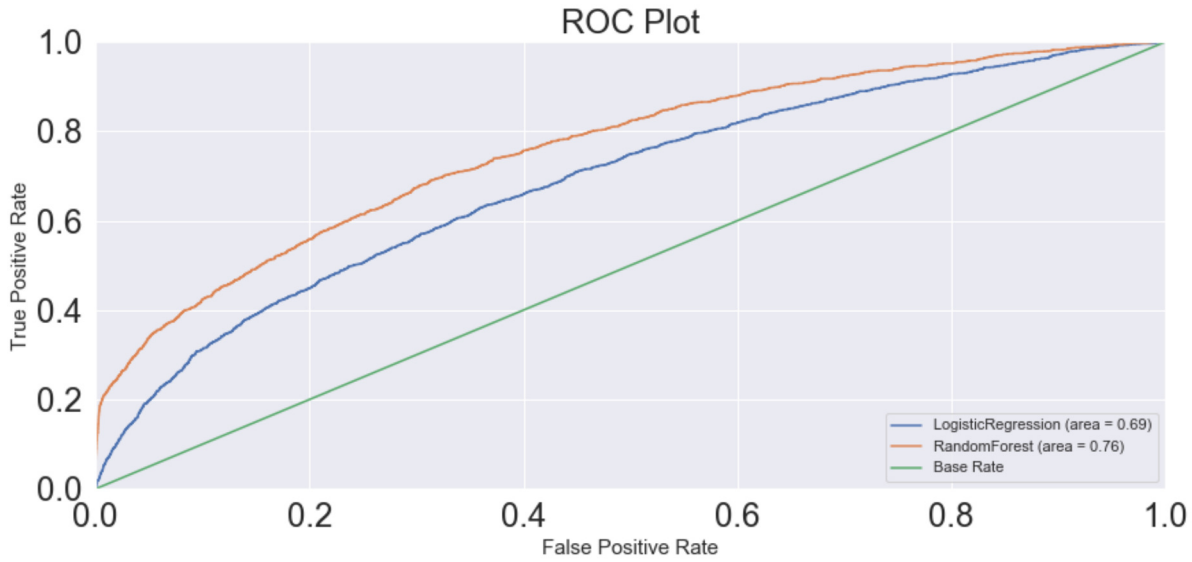
In [560]:
```
# Calculate and print model accuracy,confusion matrix and plot ROC curves.
# confustion matrix - number of correct and incorrect predictions
# confusion matrix shows the ways in which your classification model is confused w
hen it makes predictions
def display_results(model,model_auc,x_test,y_test):
        print ('\n ---Model Summary---')
        plt.figure(figsize = (14, 6))
        for k,v in model.items():
            model_predicted = v.predict(x_test)
            print ('Model accuracy for {} = {}'.format(k,accuracy_score(y_test,mod
el_predicted)))
            model_roc_auc = roc_auc_score(y_test, model_predicted)
            print ('Model ROC AUC for {} = {}'.format(k,model_roc_auc))
            print(classification_report(y_test, model_predicted))
            model_matrix = confusion_matrix(y_test, model_predicted)
            print('Confusion Matrix for model {} : \n {}'.format(k,model_matrix))

            fpr, tpr, thresholds = roc_curve(y_test, v.predict_proba(x_test)[:,1])
            # plot ROC
            plt.plot(fpr, tpr, label='%s (area = %0.2f)' % (k, model_auc[k]))
        # plot Base Rate ROC
        plt.plot([0,1], [0,1],label='Base Rate')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
        plt.xlabel('False Positive Rate', fontsize=15)
        plt.ylabel('True Positive Rate', fontsize=15)
        plt.title('ROC Plot')
        plt.legend(loc="lower right")
        plt.show();
```

In [538]:
```
#Definition of the Terms:
#• Positive (P) : Observation is positive (for example: is an apple).
#• Negative (N) : Observation is not positive (for example: is not an apple).
#• True Positive (TP) : Observation is positive, and is predicted to be positive.
#• False Negative (FN) : Observation is positive, but is predicted negative.
#• True Negative (TN) : Observation is negative, and is predicted to be negative.
#• False Positive (FP) : Observation is negative, but is predicted positive.
```

In [539]:
```
# Precsion tells us about when it predicts yes, how often is it correct
# Recall gives us an idea about when it’s actually yes, how often does it predict
yes
```

In [561]:
```
display_results(models, model_auc, x_test, y_test)
```

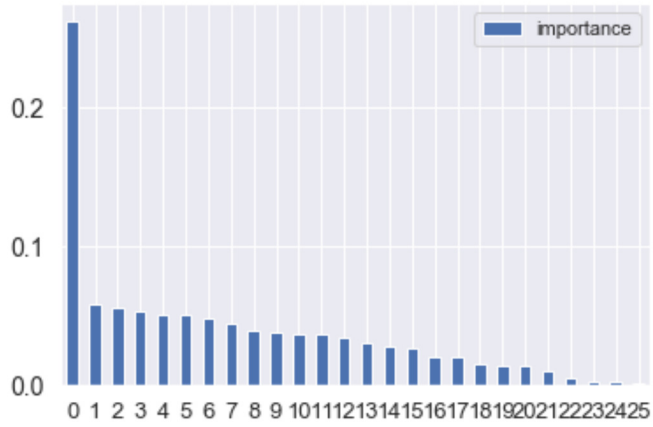
```
 ---Model Summary---
Model accuracy for LogisticRegression = 0.6469822566280743
Model ROC AUC for LogisticRegression = 0.6310866655233319
              precision    recall  f1-score   support

           0       0.92      0.65      0.76     12850
           1       0.19      0.61      0.29      1747

    accuracy                           0.65     14597
   macro avg       0.56      0.63      0.53     14597
weighted avg       0.84      0.65      0.71     14597

Confusion Matrix for model LogisticRegression :
 [[8378 4472]
 [ 681 1066]]
Model accuracy for RandomForest = 0.8344865383297938
Model ROC AUC for RandomForest = 0.6653747502667163
              precision    recall  f1-score   support

           0       0.92      0.89      0.90     12850
           1       0.35      0.44      0.39      1747

    accuracy                           0.83     14597
   macro avg       0.64      0.67      0.65     14597
weighted avg       0.85      0.83      0.84     14597

Confusion Matrix for model RandomForest :
 [[11407  1443]
 [  973   774]]
```



In [629]:
```
# Print feature importance for models.
def get_feature_importance(models,x_train):
       for k,v in models.items():
        if hasattr(v, 'feature_importances_'):
            feature_importances = pd.DataFrame(v.feature_importances_,
                                       index = x_train.columns,
                                        columns=['importance']).sort_values('impor
tance', ascending=False)
            feature_importances = feature_importances.reset_index()
            print ("Feature importances for model {} are \n {}".format(k,feature_i
mportances))
            feature_importances.plot.bar()
            plt.xticks(fontsize=13, rotation=0)
            plt.yticks(fontsize=14)
            plt.show()
        else:
            print ("Feature importances do not exist for model",k)
```

In [630]:
```
get_feature_importance(models, x_train)
```

```
Feature importances do not exist for model LogisticRegression
Feature importances for model RandomForest are
                                index  importance
0                           WheelType    0.261424
1                          VehicleAge    0.058215
2                              VehOdo    0.055395
3       MMRCurrentAuctionAveragePrice    0.053027
4                             VehBCost    0.051040
5   MMRAcquisitionAuctionAveragePrice    0.050996
6     MMRAcquisitionAuctionCleanPrice    0.047886
7         MMRCurrentAuctionCleanPrice    0.044509
8        MMRCurrentRetailAveragePrice    0.039175
9    MMRAcquisitionRetailAveragePrice    0.037636
10                          PurchDate    0.037162
11         MMRCurrentRetailCleanPrice    0.037095
12      MMRAcquisitonRetailCleanPrice    0.034292
13                      WarrantyCost    0.030667
14                          SubModel    0.027712
15                             Model    0.026526
16                              Trim    0.020968
17                              VNST    0.020103
18                              Make    0.016153
19                             Color    0.013711
20                           Auction    0.013670
21                              Size    0.010677
22              TopThreeAmericanName    0.005549
23                       Nationality    0.002864
24                      Transmission    0.002447
25                      IsOnlineSale    0.001104
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: