

# Importing Necessary Libraries

In [1]:

```
import pandas as pd
import numpy as np
import sklearn
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import confusion_matrix
import numpy as np
from collections import defaultdict
import pydot
from io import StringIO
from sklearn.tree import export_graphviz
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import VotingClassifier
from sklearn.feature_selection import RFECV
from sklearn.metrics import roc_curve
from itertools import compress
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler

'''
TODO:

1. Try to improve
2. Desing the replace_val for each column
3. Creat preprocess procedure for every class.
'''

%matplotlib inline

rs = 101
```

## Task 1. Data Selection and Distribution.

In [2]:

```
## Read Data
df = pd.read_csv("CaseStudyData.csv")
```

```
/home/chihcheng/python3env/lib/python3.6/site-packages/IPython/core/
interactiveshell.py:3044: DtypeWarning: Columns (27) have mixed type
s. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

## 1. What is the proportion of cars who can be classified as a “kick”?

In [3]:

```
## Exploring the features in this dataset
print("Number of Columns: ", len(df.columns))
print("Columns: ", list(df.columns))
```

Number of Columns: 31

Columns: ['PurchaseID', 'PurchaseTimestamp', 'PurchaseDate', 'Auction', 'VehYear', 'Make', 'Color', 'Transmission', 'WheelTypeID', 'WheelType', 'VehOdo', 'Nationality', 'Size', 'TopThreeAmericanName', 'MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice', 'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitionRetailCleanPrice', 'MMRCurrentAuctionAveragePrice', 'MMRCurrentAuctionCleanPrice', 'MMRCurrentRetailAveragePrice', 'MMRCurrentRetailCleanPrice', 'MMRCurrentRetailRatio', 'PRIMEUNIT', 'AUCGUART', 'VNST', 'VehBCost', 'IsOnlineSale', 'WarrantyCost', 'ForSale', 'IsBadBuy']

In [4]:

```
print("Number of Observations: ", len(df))
```

Number of Observations: 41476

In [5]:

```
proportionOfKicks = len(df[df['IsBadBuy'] == 1]) / len(list(df['IsBadBuy']))
print("The proportion of kicks: ", proportionOfKicks)
```

The proportion of kicks: 0.1294965763333012

## 2. Did you have to fix any data quality problems? Detail them.

In [6]:

```
#### PREPROCESSING STRATEGY
NEW_STRATEGY = True
ResamplingMethod = 'ros' #['ros', 'rus']
if NEW_STRATEGY:
    print("Using New Preprocessing Strategy")
    using_cat = True
    categorial_cols = ['Auction', 'VehYear', 'Make', 'Color', 'Transmission', 'WheelTypeID', 'WheelType', 'Nationality', 'Size', 'TopThreeAmericanName', 'PRIMEUNIT', 'AUCGUART', 'VNST', 'IsOnlineSale', 'ForSale' ] # Replaced by the most common
    interval_cols = ['VehOdo', 'MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice', 'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitionRetailCleanPrice', 'VehBCost', 'WarrantyCost' ]
    drop_cols = ['PurchaseID', 'PurchaseDate', 'PurchaseTimestamp']
    questionMark_data = ['MMRCurrentAuctionAveragePrice', 'MMRCurrentAuctionCleanPrice', 'MMRCurrentRetailAveragePrice', 'MMRCurrentRetailCleanPrice', 'MMRCurrentRetailRatio']
    replaced_vals = ['?', '#VALUE!']
    if using_cat:
        categorial_cols += questionMark_data
        print("See [MMRCurrentAuctionAveragePrice" +
              "MMRCurrentAuctionCleanPrice, MMRCurrentRetailAveragePrice," +
              " MMRCurrentRetailCleanPrice, MMRCurrentRetailRatio] as Categorical
Data")
    else:
        interval_cols += questionMark_data
        print("See [MMRCurrentAuctionAveragePrice" +
              "MMRCurrentAuctionCleanPrice, MMRCurrentRetailAveragePrice," +
              " MMRCurrentRetailCleanPrice, MMRCurrentRetailRatio] as Interval
Data")
    else:
        print("Using Old Preprocessing Strategy")
        drop_cols = ['PurchaseID', 'PurchaseDate']
        categorial_cols = ['Auction', 'VehYear', 'Make', 'Color', 'Transmission', 'WheelTypeID', 'WheelType', 'Nationality', 'Size', 'TopThreeAmericanName', 'PRIMEUNIT', 'AUCGUART', 'VNST', 'IsOnlineSale', 'ForSale' ] # Replaced by the most common
        interval_cols = ['PurchaseTimestamp', 'VehOdo', 'MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice', 'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitionRetailCleanPrice', 'MMRCurrentAuctionAveragePrice', 'MMRCurrentAuctionCleanPrice', 'MMRCurrentRetailAveragePrice', 'MMRCurrentRetailCleanPrice', 'MMRCurrentRetailRatio', 'VehBCost', 'WarrantyCost' ] # Replaced by the mean
        replaced_vals = ['?', '#VALUE!']

print("Total null before Replacing: ", df.isnull().sum().sum())
```

Using New Preprocessing Strategy

See [MMRCurrentAuctionAveragePriceMMRCurrentAuctionCleanPrice, MMRCurrentRetailAveragePrice, MMRCurrentRetailCleanPrice, MMRCurrentRetailRatio] as Categorical Data

Total null before Replacing: 1691

In [7]:

```

def printColumnInfo():
    '''
    Display the information of this Dataframe
    '''

    for colName in df.columns:
        print("===== " + str(colName) + " =====")
        print("----- FIRST FIVE -----")
        print(df[colName][:5])
        print("----- DESCRIBE -----")
        print(df[colName].describe())
        print("----- COUNTS -----")
        commonList = list(df[colName].value_counts().keys())
        if len(commonList) > 100:
            print("Five Most Common: ", commonList[:5])
        else:
            print("Count List: \n", df[colName].value_counts())
        print("Num of NULL: ", df[colName].isnull().sum())
        for rep in replaced_vals:
            print("Number of "+str(rep)+" : " + str(len(df[df[colName] == rep
])))
printColumnInfo()

```

```
/home/chihcheng/python3env/lib/python3.6/site-packages/pandas/core/ops.py:1649: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
    result = method(y)
```

```
===== PurchaseID =====
----- FIRST FIVE -----
0      0
1      1
2      2
3      3
4      4
```

Name: PurchaseID, dtype: int64

```
----- DESCRIBE -----
count      41476.000000
mean       20737.500000
std        11973.234219
min         0.000000
25%        10368.750000
50%        20737.500000
75%        31106.250000
max        41475.000000
```

Name: PurchaseID, dtype: float64

```
----- COUNTS -----
Five Most Common: [2047, 11567, 15693, 13644, 3403]
Num of NULL: 0
Number of ? : 0
Number of #VALUE! : 0
```

```
===== PurchaseTimestamp =====
----- FIRST FIVE -----
0      1253232000
1      1253232000
2      1253232000
3      1253232000
4      1253232000
```

Name: PurchaseTimestamp, dtype: int64

```
----- DESCRIBE -----
count      4.147600e+04
mean       1.262260e+09
std        1.796895e+07
min        1.231114e+09
25%        1.247530e+09
50%        1.262045e+09
75%        1.277770e+09
max        1.293667e+09
```

Name: PurchaseTimestamp, dtype: float64

```
----- COUNTS -----
Five Most Common: [1235520000, 1259020800, 1234396800, 1264032000,
1287014400]
Num of NULL: 0
Number of ? : 0
Number of #VALUE! : 0
```

```
===== PurchaseDate =====
----- FIRST FIVE -----
0      18/09/2009 10:00
1      18/09/2009 10:00
2      18/09/2009 10:00
3      18/09/2009 10:00
4      18/09/2009 10:00
```

Name: PurchaseDate, dtype: object

```
----- DESCRIBE -----
count      41476
unique      497
top        24/11/2009 10:00
freq       242
```

Name: PurchaseDate, dtype: object

```

----- COUNTS -----
Five Most Common: ['24/11/2009 10:00', '25/02/2009 10:00', '12/02/2
009 10:00', '21/01/2010 10:00', '14/10/2010 10:00']
Num of NULL: 0
Number of ? : 0
Number of #VALUE! : 0
===== Auction =====
----- FIRST FIVE -----
0    OTHER
1    OTHER
2    OTHER
3    OTHER
4    OTHER
Name: Auction, dtype: object
----- DESCRIBE -----
count      41432
unique       3
top      MANHEIM
freq      22168
Name: Auction, dtype: object
----- COUNTS -----
Count List:
MANHEIM      22168
ADESA        11086
OTHER         8178
Name: Auction, dtype: int64
Num of NULL: 44
Number of ? : 0
Number of #VALUE! : 0
===== VehYear =====
----- FIRST FIVE -----
0    2008.0
1    2008.0
2    2008.0
3    2008.0
4    2008.0
Name: VehYear, dtype: float64
----- DESCRIBE -----
count      41432.000000
mean        2005.360615
std          1.730587
min          2001.000000
25%          2004.000000
50%          2005.000000
75%          2007.000000
max          2010.000000
Name: VehYear, dtype: float64
----- COUNTS -----
Count List:
2006.0      9630
2005.0      8682
2007.0      6514
2004.0      5792
2008.0      4177
2003.0      3554
2002.0      1879
2001.0       816
2009.0       387
2010.0        1
Name: VehYear, dtype: int64
Num of NULL: 44

```

Number of ? : 0

Number of #VALUE! : 0

===== Make =====

----- FIRST FIVE -----

0 DODGE

1 DODGE

2 CHRYSLER

3 CHEVROLET

4 DODGE

Name: Make, dtype: object

----- DESCRIBE -----

count 41432

unique 30

top CHEVROLET

freq 9548

Name: Make, dtype: object

----- COUNTS -----

Count List:

CHEVROLET 9548

DODGE 7385

FORD 6458

CHRYSLER 5259

PONTIAC 2355

KIA 1337

SATURN 1245

NISSAN 1186

JEEP 985

HYUNDAI 957

SUZUKI 842

TOYOTA 664

MAZDA 532

MERCUY 527

BUICK 413

GMC 351

HONDA 263

OLDSMOBILE 146

ISUZU 82

SCION 77

VOLKSWAGEN 73

LINCOLN 54

INFINITI 27

MINI 19

ACURA 19

SUBARU 17

CADILLAC 17

LEXUS 13

VOLVO 12

Name: Make, dtype: int64

Num of NULL: 44

Number of ? : 0

Number of #VALUE! : 0

===== Color =====

----- FIRST FIVE -----

0 RED

1 RED

2 SILVER

3 RED

4 SILVER

Name: Color, dtype: object

----- DESCRIBE -----



```
count      41432
unique      17
top         SILVER
freq        8541
```

Name: Color, dtype: object

----- COUNTS -----

Count List:

```
SILVER      8541
WHITE       6890
BLUE        5855
BLACK       4392
GREY        4248
RED         3661
GOLD        3059
GREEN       1796
MAROON      1039
BEIGE       894
ORANGE      255
BROWN       249
PURPLE      205
YELLOW      141
OTHER       136
NOT AVAIL   65
?           6
```

Name: Color, dtype: int64

Num of NULL: 44

Number of ? : 6

Number of #VALUE! : 0

===== Transmission =====

----- FIRST FIVE -----

```
0    AUTO
1    AUTO
2    AUTO
3    AUTO
4    AUTO
```

Name: Transmission, dtype: object

----- DESCRIBE -----

```
count      41432
unique      4
top         AUTO
freq        39930
```

Name: Transmission, dtype: object

----- COUNTS -----

Count List:

```
AUTO       39930
MANUAL     1495
?           6
Manual      1
```

Name: Transmission, dtype: int64

Num of NULL: 44

Number of ? : 6

Number of #VALUE! : 0

===== WheelTypeID =====

----- FIRST FIVE -----

```
0    2
1    2
2    2
3    2
4    2
```

Name: WheelTypeID, dtype: object

----- DESCRIBE -----

```
count      41432
unique      5
top         1
freq       20426
```

Name: WheelTypeID, dtype: object

----- COUNTS -----

Count List:

```
1      20426
2      18791
?       1775
3        437
0         3
```

Name: WheelTypeID, dtype: int64

Num of NULL: 44

Number of ? : 1775

Number of #VALUE! : 0

===== WheelType =====

----- FIRST FIVE -----

```
0      Covers
1      Covers
2      Covers
3      Covers
4      Covers
```

Name: WheelType, dtype: object

----- DESCRIBE -----

```
count      41380
unique      4
top        Alloy
freq       20406
```

Name: WheelType, dtype: object

----- COUNTS -----

Count List:

```
Alloy      20406
Covers     18761
?          1777
Special    436
```

Name: WheelType, dtype: int64

Num of NULL: 96

Number of ? : 1777

Number of #VALUE! : 0

===== Veh0do =====

----- FIRST FIVE -----

```
0      51099.0
1      48542.0
2      46318.0
3      50413.0
4      50199.0
```

Name: Veh0do, dtype: float64

----- DESCRIBE -----

```
count      41432.000000
mean       71300.010427
std        14724.041171
min         577.000000
25%        61578.000000
50%        73128.500000
75%        82259.250000
max        480444.000000
```

Name: Veh0do, dtype: float64

----- COUNTS -----

Five Most Common: [84675.0, 85884.0, 67464.0, 72101.0, 79600.0]

Num of NULL: 44

Number of ? : 0

Number of #VALUE! : 0

===== Nationality =====

----- FIRST FIVE -----

0 AMERICAN

1 AMERICAN

2 AMERICAN

3 AMERICAN

4 AMERICAN

Name: Nationality, dtype: object

----- DESCRIBE -----

count 41432

unique 6

top AMERICAN

freq 34616

Name: Nationality, dtype: object

----- COUNTS -----

Count List:

AMERICAN 34616

OTHER ASIAN 4474

TOP LINE ASIAN 2110

USA 125

OTHER 104

? 3

Name: Nationality, dtype: int64

Num of NULL: 44

Number of ? : 3

Number of #VALUE! : 0

===== Size =====

----- FIRST FIVE -----

0 MEDIUM

1 MEDIUM

2 MEDIUM

3 COMPACT

4 MEDIUM

Name: Size, dtype: object

----- DESCRIBE -----

count 41432

unique 13

top MEDIUM

freq 17540

Name: Size, dtype: object

----- COUNTS -----

Count List:

MEDIUM 17540

LARGE 4968

MEDIUM SUV 4569

COMPACT 4035

VAN 3367

LARGE TRUCK 1897

SMALL SUV 1332

SPECIALTY 998

CROSSOVER 974

LARGE SUV 830

SMALL TRUCK 494

SPORTS 425

? 3

Name: Size, dtype: int64

Num of NULL: 44

Number of ? : 3

Number of #VALUE! : 0

```

===== TopThreeAmericanName =====
----- FIRST FIVE -----
0    CHRYSLER
1    CHRYSLER
2    CHRYSLER
3         GM
4    CHRYSLER
Name: TopThreeAmericanName, dtype: object
----- DESCRIBE -----
count      41432
unique       5
top         GM
freq       14075
Name: TopThreeAmericanName, dtype: object
----- COUNTS -----
Count List:
  GM      14075
CHRYSLER  13627
FORD      7039
OTHER     6688
?          3
Name: TopThreeAmericanName, dtype: int64
Num of NULL:  44
Number of ? : 3
Number of #VALUE! : 0
===== MMRAcquisitionAuctionAveragePrice =====
=====
----- FIRST FIVE -----
0    8566
1    8566
2    8835
3    7165
4    8566
Name: MMRAcquisitionAuctionAveragePrice, dtype: object
----- DESCRIBE -----
count      41416
unique     9271
top         0
freq       502
Name: MMRAcquisitionAuctionAveragePrice, dtype: object
----- COUNTS -----
Five Most Common:  ['0', '5480', '6311', '7811', '7644']
Num of NULL:  60
Number of ? : 7
Number of #VALUE! : 0
===== MMRAcquisitionAuctionCleanPrice =====
=====
----- FIRST FIVE -----
0    9325
1    9325
2    9428
3    7770
4    9325
Name: MMRAcquisitionAuctionCleanPrice, dtype: object
----- DESCRIBE -----
count      41429
unique     10010
top         0
freq       415
Name: MMRAcquisitionAuctionCleanPrice, dtype: object
----- COUNTS -----

```

Five Most Common: ['0', '6461', '7450', '1', '8258']

Num of NULL: 47

Number of ? : 7

Number of #VALUE! : 0

===== MMRAcquisitionRetailAveragePrice =====  
=====

----- FIRST FIVE -----

0 9751

1 9751

2 10042

3 8238

4 9751

Name: MMRAcquisitionRetailAveragePrice, dtype: object

----- DESCRIBE -----

count 41429

unique 11070

top 0

freq 502

Name: MMRAcquisitionRetailAveragePrice, dtype: object

----- COUNTS -----

Five Most Common: ['0', '6418', '7316', '11114', '8756']

Num of NULL: 47

Number of ? : 7

Number of #VALUE! : 0

===== MMRAcquisitonRetailCleanPrice =====  
=====

----- FIRST FIVE -----

0 10571

1 10571

2 10682

3 8892

4 10571

Name: MMRAcquisitonRetailCleanPrice, dtype: object

----- DESCRIBE -----

count 41327

unique 11583

top 0

freq 501

Name: MMRAcquisitonRetailCleanPrice, dtype: object

----- COUNTS -----

Five Most Common: ['0', '7478', '8546', '11562', '10103']

Num of NULL: 149

Number of ? : 7

Number of #VALUE! : 0

===== MMRCurrentAuctionAveragePrice =====  
=====

----- FIRST FIVE -----

0 7781

1 8568

2 8137

3 7074

4 7857

Name: MMRCurrentAuctionAveragePrice, dtype: object

----- DESCRIBE -----

count 41429

unique 9183

top 0

freq 287

Name: MMRCurrentAuctionAveragePrice, dtype: object

----- COUNTS -----

Five Most Common: ['0', '?', '5480', '6311', '7269']

Num of NULL: 47  
 Number of ? : 184  
 Number of #VALUE! : 0

===== MMRCurrentAuctionCleanPrice =====  
 =====

----- FIRST FIVE -----

0 8545  
 1 9325  
 2 8733  
 3 7629  
 4 8711

Name: MMRCurrentAuctionCleanPrice, dtype: object

----- DESCRIBE -----

count 41429  
 unique 9890  
 top 0  
 freq 206

Name: MMRCurrentAuctionCleanPrice, dtype: object

----- COUNTS -----

Five Most Common: ['0', '?', '6461', '1', '7450']

Num of NULL: 47  
 Number of ? : 184  
 Number of #VALUE! : 0

===== MMRCurrentRetailAveragePrice =====  
 =====

----- FIRST FIVE -----

0 11777  
 1 9753  
 2 9288  
 3 8140  
 4 8986

Name: MMRCurrentRetailAveragePrice, dtype: object

----- DESCRIBE -----

count 41409  
 unique 10935  
 top 0  
 freq 287

Name: MMRCurrentRetailAveragePrice, dtype: object

----- COUNTS -----

Five Most Common: ['0', '?', '6418', '7316', '8756']

Num of NULL: 67  
 Number of ? : 184  
 Number of #VALUE! : 0

===== MMRCurrentRetailCleanPrice =====  
 =====

----- FIRST FIVE -----

0 12505  
 1 10571  
 2 9932  
 3 8739  
 4 9908

Name: MMRCurrentRetailCleanPrice, dtype: object

----- DESCRIBE -----

count 41409  
 unique 11363  
 top 0  
 freq 287

Name: MMRCurrentRetailCleanPrice, dtype: object

----- COUNTS -----

Five Most Common: ['0', '?', '7478', '8546', '10103']

Num of NULL: 67

Number of ? : 184

Number of #VALUE! : 0

===== MMRCurrentRetailRatio =====

=

----- FIRST FIVE -----

0 0.941783287

1 0.922618485

2 0.935159082

3 0.931456688

4 0.906943884

Name: MMRCurrentRetailRatio, dtype: object

----- DESCRIBE -----

count 41116

unique 25870

top #VALUE!

freq 178

Name: MMRCurrentRetailRatio, dtype: object

----- COUNTS -----

Five Most Common: ['#VALUE!', '0.858250869', '0.856073017', '0.866673265', '0.949268378']

Num of NULL: 360

Number of ? : 0

Number of #VALUE! : 178

===== PRIMEUNIT =====

----- FIRST FIVE -----

0 ?

1 ?

2 ?

3 ?

4 ?

Name: PRIMEUNIT, dtype: object

----- DESCRIBE -----

count 41432

unique 3

top ?

freq 39634

Name: PRIMEUNIT, dtype: object

----- COUNTS -----

Count List:

? 39634

NO 1764

YES 34

Name: PRIMEUNIT, dtype: int64

Num of NULL: 44

Number of ? : 39634

Number of #VALUE! : 0

===== AUCGUART =====

----- FIRST FIVE -----

0 ?

1 ?

2 ?

3 ?

4 ?

Name: AUCGUART, dtype: object

----- DESCRIBE -----

count 41432

unique 3

top ?

freq 39634

Name: AUCGUART, dtype: object

----- COUNTS -----

Count List:

? 39634

GREEN 1754

RED 44

Name: AUCGUART, dtype: int64

Num of NULL: 44

Number of ? : 39634

Number of #VALUE! : 0

===== VNST =====

----- FIRST FIVE -----

0 NC

1 NC

2 NC

3 NC

4 NC

Name: VNST, dtype: object

----- DESCRIBE -----

count 41432

unique 31

top TX

freq 9076

Name: VNST, dtype: object

----- COUNTS -----

Count List:

TX 9076

FL 5250

CO 3623

NC 3594

AZ 3383

CA 3268

OK 2595

SC 1662

TN 1471

GA 1287

VA 1093

MO 758

PA 700

NV 553

IN 486

MS 412

LA 349

NJ 317

NM 239

KY 230

AL 179

IL 165

UT 165

WV 137

OR 136

WA 136

NH 97

NE 26

OH 25

ID 14

NY 6

Name: VNST, dtype: int64

Num of NULL: 44

Number of ? : 0

Number of #VALUE! : 0

===== VehBCost =====

----- FIRST FIVE -----



```

0    7800
1    7800
2    7800
3    6000
4    7800

```

Name: VehBCost, dtype: object

----- DESCRIBE -----

```

count    41432
unique    1869
top       7500
freq      459

```

Name: VehBCost, dtype: object

----- COUNTS -----

Five Most Common: ['7500', '6500', '7800', '7200', '7000']

Num of NULL: 44

Number of ? : 29

Number of #VALUE! : 0

===== IsOnlineSale =====

----- FIRST FIVE -----

```

0    0
1    0
2    0
3    0
4    0

```

Name: IsOnlineSale, dtype: object

----- DESCRIBE -----

```

count    41432.0
unique      8.0
top       0.0
freq    31368.0

```

Name: IsOnlineSale, dtype: float64

----- COUNTS -----

Count List:

```

0.0    31368
0       8572
1.0       753
-1.0     601
1        134
?         2
4.0        1
2.0        1

```

Name: IsOnlineSale, dtype: int64

Num of NULL: 44

Number of ? : 2

Number of #VALUE! : 0

===== WarrantyCost =====

----- FIRST FIVE -----

```

0    920.0
1    834.0
2    834.0
3    671.0
4    920.0

```

Name: WarrantyCost, dtype: float64

----- DESCRIBE -----

```

count    41432.000000
mean     1273.050758
std       599.188662
min       462.000000
25%       834.000000
50%      1155.000000
75%      1623.000000

```

```

max          7498.000000
Name: WarrantyCost, dtype: float64
----- COUNTS -----
Five Most Common:  [920.0, 1974.0, 2152.0, 1215.0, 1389.0]
Num of NULL:  44
Number of ? : 0
Number of #VALUE! : 0
===== ForSale =====
----- FIRST FIVE -----
0    Yes
1    Yes
2    Yes
3    Yes
4    Yes
Name: ForSale, dtype: object
----- DESCRIBE -----
count      41476
unique       6
top        Yes
freq      27402
Name: ForSale, dtype: object
----- COUNTS -----
Count List:
  Yes      27402
YES       8544
yes       5524
?          3
No         2
0          1
Name: ForSale, dtype: int64
Num of NULL: 0
Number of ? : 3
Number of #VALUE! : 0
===== IsBadBuy =====
----- FIRST FIVE -----
0    0
1    0
2    0
3    0
4    0
Name: IsBadBuy, dtype: int64
----- DESCRIBE -----
count      41476.000000
mean        0.129497
std         0.335753
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000
Name: IsBadBuy, dtype: float64
----- COUNTS -----
Count List:
  0      36105
  1       5371
Name: IsBadBuy, dtype: int64
Num of NULL: 0
Number of ? : 0
Number of #VALUE! : 0

```

In [8]:

```

if NEW_STRATEGY:

    class filling_method():
        MOST_COMMON = "MOST_COMMON"
        MEAN = "MEAN"
        CERTAIN_VALUE = "CERTAIN_VALUE"

    def replaceFunc(colName):
        for replaced, target in preprocessStrategy[colName]['replace_pairs']:
            df[colName].replace(replaced, target, inplace=True)

    def removeOutlier(colName): # FOR THE INTERVAL ONLY
        global df
        df = df[df[colName] < df[colName].quantile(0.999)]

    def replacingValueCol(colName):
        for replaced in preprocessStrategy[colName]['replaced_vals']:
            print("In the Column: " + str(colName) + " : " + str(len(
                df[df[colName] == replaced])) + ", " + str(replaced) + "have been
n replaced by null")
            # Replacing the null in this process #Inplacing for saving the memory
            df[colName].replace(replaced, float('nan'), inplace=True)

    def loweringCol(colName):
        df[colName] = df[colName].str.lower()

    def fillingTheNullValue(colName): # method can be ["MEAN", "MOST_COMMON"]
        if preprocessStrategy[colName]['filling_method'] == filling_method.MEAN:
            df[colName] = df[colName].astype('float')
            df[colName].fillna(df[colName].astype(
                'float').mean(), inplace=True)
        elif preprocessStrategy[colName]['filling_method'] == filling_method.MOST
T_COMMON:
            df[colName] = df[colName].astype('category')
            df[colName].fillna(df[colName].astype(
                'category').describe()['top'], inplace=True)
        elif preprocessStrategy[colName]['filling_method'] == filling_method.CER
TAIN_VALUE:
            df[colName] = df[colName].astype('category')
            df[colName] = df[colName].cat.add_categories(
                [preprocessStrategy[colName]['filling_value']])
            df[colName].fillna(preprocessStrategy[colName]
                ['filling_value'], inplace=True)

    def filterOutRareValue(colName):

        def checkingKeepValue(v, savingValues):
            if v in savingValues:
                return v
            return "LESS_FREQ"

        k = [v for v in df[colName].value_counts().values if v >
            preprocessStrategy[colName]['min_freq']]
        savingValues = df[colName].value_counts().keys()[:len(k)]

        df[colName] = [checkingKeepValue(v, savingValues) for v in df[colName]]

```

```

def changeToType(colName):
    df[colName] = df[colName].astype(
        preprocessStrategy[colName]['changeToType'])

def newData_prep(df):
    """
    For Preprocessing through the whole dictionary
    """
    df.drop(drop_cols, axis=1, inplace=True)

    for colName in df.columns: # df.columns:

        print("Preprocess the col: " + colName)

        for stra in preprocessStrategy[colName]['strategies']:
            if not stra:
                continue
            print(stra)
            stra(colName)

    if not using_cat:
        df['MMRCurrentRetailRatio'] = df['MMRCurrentRetailAveragePrice'] / \
            (df['MMRCurrentRetailCleanPrice']+1e-8) # Prvent divided by 0

    return df

preprocessStrategy = defaultdict(dict)

preprocessStrategy['Auction'] = {
    "strategies":
        [
            replacingValueCol,
            loweringCol,
            fillingTheNullValue,
        ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MOST_COMMON
}

preprocessStrategy['VehYear'] = {
    "strategies":
        [
            fillingTheNullValue,
        ],
    "filling_method": filling_method.CERTAIN_VALUE,
    "filling_value": "UNKNOWN_VALUE"
}

preprocessStrategy['Make'] = {
    "strategies":
        [
            loweringCol,
            fillingTheNullValue,
        ],
    "filling_method": filling_method.MOST_COMMON
}

preprocessStrategy['Color'] = {
    "strategies":
        [
            loweringCol,

```

```

        replacingValueCol,
        fillingTheNullValue,
    ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MOST_COMMON
}

preprocessStrategy['Transmission'] = {
    "strategies":
    [
        loweringCol,
        replacingValueCol,
        fillingTheNullValue,
    ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MOST_COMMON
}

preprocessStrategy['WheelTypeID'] = {
    "strategies":
    [
        fillingTheNullValue,
    ],
    "filling_method": filling_method.MOST_COMMON
}

preprocessStrategy['WheelType'] = {
    "strategies":
    [
        loweringCol,
        fillingTheNullValue,
    ],
    "filling_method": filling_method.MOST_COMMON
}

preprocessStrategy['Veh0do'] = {
    "strategies":
    [
        fillingTheNullValue,
    ],
    "filling_method": filling_method.MEAN
}

preprocessStrategy['Nationality'] = { # Should I merge USA with AMERICAN?
    "strategies":
    [
        replaceFunc,
        loweringCol,
        replacingValueCol,
        fillingTheNullValue,
    ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MOST_COMMON,
    "replace_pairs": [("USA", "AMERICAN")]
}

preprocessStrategy['Size'] = {
    "strategies":
    [
        loweringCol,

```

```

        replacingValueCol,
        fillingTheNullValue,
    ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MOST_COMMON
}

preprocessStrategy['TopThreeAmericanName'] = {
    "strategies":
        [
            loweringCol,
            replacingValueCol,
            fillingTheNullValue,
        ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MOST_COMMON
}

preprocessStrategy['TopThreeAmericanName'] = {
    "strategies":
        [
            loweringCol,
            replacingValueCol,
            fillingTheNullValue,
        ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MOST_COMMON
}

preprocessStrategy['MMRAcquisitionAuctionAveragePrice'] = {
    "strategies":
        [
            replacingValueCol,
            fillingTheNullValue,
        ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MEAN
}

preprocessStrategy['MMRAcquisitionAuctionCleanPrice'] = {
    "strategies":
        [
            replacingValueCol,
            fillingTheNullValue,
        ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MEAN
}

preprocessStrategy['MMRAcquisitionRetailAveragePrice'] = {
    "strategies":
        [
            replacingValueCol,
            fillingTheNullValue,
        ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MEAN
}

preprocessStrategy['MMRAcquisitionRetailCleanPrice'] = {

```

```

    "strategies":
        [
            replacingValueCol,
            fillingTheNullValue,
        ],
    "replaced_vals": ['?'],
    "filling_method": filling_method.MEAN
}

#####

int_stra = {
    "strategies":
        [
            replacingValueCol,
            fillingTheNullValue,
        ],
    "replaced_vals": ['?', '#VALUE!'], # GOT 184 '?'
    "filling_method": filling_method.MEAN,
}

cat_stra = { # HOW DO WE DEAL WITH ? in this column
    "strategies":
        [
            filterOutRareValue,
            fillingTheNullValue,
        ],
    # "replaced_vals": ['?'], # GOT 184 '?'
    "filling_method": filling_method.CERTAIN_VALUE,
    "filling_value": 'NULL',
    "min_freq": 50
}

preprocessStrategy['MMRCurrentAuctionAveragePrice'] \
= preprocessStrategy['MMRCurrentAuctionCleanPrice'] \
= preprocessStrategy['MMRCurrentRetailAveragePrice'] \
= preprocessStrategy['MMRCurrentRetailCleanPrice'] \
= preprocessStrategy['MMRCurrentRetailRatio'] \
= cat_stra if using_cat else int_stra

#####

preprocessStrategy['PRIMEUNIT'] = { # HOW DO WE DEAL WITH ? in this column
    "strategies":
        [
            loweringCol,
            fillingTheNullValue,
        ],
    # "replaced_vals": ['?'], # GOT 184 '?'
    "filling_method": filling_method.CERTAIN_VALUE,
    "filling_value": 'NULL',
}

preprocessStrategy['AUCGUART'] = { # HOW DO WE DEAL WITH ? in this column
    "strategies":
        [
            loweringCol,
            fillingTheNullValue,
        ],
    # "replaced_vals": ['?'], # GOT 184 '?'
    "filling_method": filling_method.CERTAIN_VALUE,

```

```

        "filling_value": 'NULL',
    }

    preprocessStrategy['VNST'] = { # HOW DO WE DEAL WITH ? in this column
        "strategies":
            [
                loweringCol,
                fillingTheNullValue,
            ],
        # "replaced_vals": ['?'], # GOT 184 '?'
        "filling_method": filling_method.CERTAIN_VALUE,
        "filling_value": 'NULL',
    }

    preprocessStrategy['VehBCost'] = { # HOW DO WE DEAL WITH ? in this column
        "strategies":
            [
                replacingValueCol,
                fillingTheNullValue,
            ],
        "replaced_vals": ['?'], # GOT 184 '?'
        "filling_method": filling_method.MEAN
    }

    preprocessStrategy['IsOnlineSale'] = { # HOW DO WE DEAL WITH ? in this column
mn
        "strategies":
            [
                replacingValueCol,
                changeToType,
                fillingTheNullValue,
            ],
        "replaced_vals": ['?', 2.0, 4.0], # GOT 184 '?'
        "filling_method": filling_method.MOST_COMMON,
        "changeToType": 'float'
    }

    preprocessStrategy['WarrantyCost'] = { # HOW DO WE DEAL WITH ? in this column
mn
        "strategies":
            [
                fillingTheNullValue,
            ],
        "replaced_vals": ['?'], # GOT 184 '?'
        "filling_method": filling_method.MEAN,
    }

    preprocessStrategy['ForSale'] = { # HOW DO WE DEAL WITH ? in this column
        "strategies":
            [
                loweringCol,
                replacingValueCol,
                fillingTheNullValue,
            ],
        "replaced_vals": ['?', 0], # GOT 184 '?'
        "filling_method": filling_method.MOST_COMMON,
    }

    # HOW DO WE DEAL WITH ? in this column
    preprocessStrategy['IsBadBuy'] = {"strategies": [None]}

```



```

newData_prep(df)

else:

    def data_prep(df):
        '''
        For Preprocessing the Data (OLD_METHOD)
        '''

        # Check the replaced values are not in the dataset

        for colName in df.columns:

            if colName in categorial_cols:

                if colName == "IsOnlineSale":
                    df[colName] = df[colName].astype(
                        'float').astype('category')
                    df[colName].fillna(df[colName].astype(
                        'category').describe()['top'], inplace=True)

                # Try to lower the data if the data type is string
                try:
                    df[colName] = df[colName].str.lower()
                except:
                    print(colName, " can't be lowered")

                for replaced in replaced_vals:
                    print("In the Column: " + str(colName) + ": " +
                        str(len(df[df[colName] == replaced))) + " -> " + str(r
eplaced))
                    df[colName].replace(replaced, float('nan'), inplace=True)

                df[colName] = df[colName].astype('category')

                # Replacing the null by the most common category
                df[colName].fillna(df[colName].astype(
                    'category').describe()['top'], inplace=True)

            if colName in interval_cols:

                if colName == "MMRCurrentRetailRatio": # Dealing with this calc
ulated value at the last
                    continue

                for replaced in replaced_vals:
                    print("In the Column: " + str(colName) + ": " +
                        str(len(df[df[colName] == replaced))) + " -> " + str(r
eplaced))
                    df[colName].replace(replaced, float('nan'), inplace=True)

                df[colName] = df[colName].astype('float')

                # Removing outlier
                df = df[df[colName] < df[colName].quantile(0.999)]

                # Replacing the null by the mean
                df[colName].fillna(df[colName].astype(
                    'float').mean(), inplace=True)

```

```
df['MMRCurrentRetailRatio'] = df['MMRCurrentRetailAveragePrice'] / \
    (df['MMRCurrentRetailCleanPrice']+1e-8) # Prvent divided by 0

df.drop(drop_cols, axis=1, inplace=True)

return df

df = data_prep(df)
```

```
Preprocess the col: Auction
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: Auction : 0, ?have been replaced by null
<function loweringCol at 0x7f9cc8f4c510>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: VehYear
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: Make
<function loweringCol at 0x7f9cc8f4c510>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: Color
<function loweringCol at 0x7f9cc8f4c510>
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: Color : 6, ?have been replaced by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: Transmission
<function loweringCol at 0x7f9cc8f4c510>
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: Transmission : 6, ?have been replaced by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: WheelTypeID
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: WheelType
<function loweringCol at 0x7f9cc8f4c510>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: Veh0do
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: Nationality
<function replaceFunc at 0x7f9cc8f4c8c8>
<function loweringCol at 0x7f9cc8f4c510>
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: Nationality : 3, ?have been replaced by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: Size
<function loweringCol at 0x7f9cc8f4c510>
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: Size : 3, ?have been replaced by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: TopThreeAmericanName
<function loweringCol at 0x7f9cc8f4c510>
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: TopThreeAmericanName : 3, ?have been replaced by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: MMRAcquisitionAuctionAveragePrice
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: MMRAcquisitionAuctionAveragePrice : 7, ?have been replaced by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: MMRAcquisitionAuctionCleanPrice
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: MMRAcquisitionAuctionCleanPrice : 7, ?have been replaced by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: MMRAcquisitionRetailAveragePrice
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: MMRAcquisitionRetailAveragePrice : 7, ?have been replaced by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: MMRAcquisitionRetailCleanPrice
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: MMRAcquisitionRetailCleanPrice : 7, ?have been replaced
```

```

d by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: MMRCurrentAuctionAveragePrice
<function filterOutRareValue at 0x7f9cc8f4c2f0>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: MMRCurrentAuctionCleanPrice
<function filterOutRareValue at 0x7f9cc8f4c2f0>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: MMRCurrentRetailAveragePrice
<function filterOutRareValue at 0x7f9cc8f4c2f0>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: MMRCurrentRetailCleanPrice
<function filterOutRareValue at 0x7f9cc8f4c2f0>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: MMRCurrentRetailRatio
<function filterOutRareValue at 0x7f9cc8f4c2f0>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: PRIMEUNIT
<function loweringCol at 0x7f9cc8f4c510>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: AUCGUART
<function loweringCol at 0x7f9cc8f4c510>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: VNST
<function loweringCol at 0x7f9cc8f4c510>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: VehBCost
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: VehBCost : 29, ?have been replaced by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: IsOnlineSale
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: IsOnlineSale : 2, ?have been replaced by null
In the Column: IsOnlineSale : 1, 2.0have been replaced by null
In the Column: IsOnlineSale : 1, 4.0have been replaced by null
<function changeToType at 0x7f9cc8f249d8>
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: WarrantyCost
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: ForSale
<function loweringCol at 0x7f9cc8f4c510>
<function replacingValueCol at 0x7f9cc8f4c268>
In the Column: ForSale : 3, ?have been replaced by null
In the Column: ForSale : 0, 0have been replaced by null
<function fillingTheNullValue at 0x7f9cc8f4c730>
Preprocess the col: IsBadBuy

```

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

In [9]:

```
def plotAllCols (df):  
    for colName in df.columns:  
        plt.figure(figsize=(20,10))  
        if colName in categorial_cols:  
            ### if it's categorial column, plot hist diagram  
            sns.countplot(x=colName, data = df, hue="IsBadBuy")  
        elif colName in interval_cols:  
            ### if it's interval column, plot box diagram  
            sns.boxplot(x="IsBadBuy", y=colName, data = df )
```

In [10]:

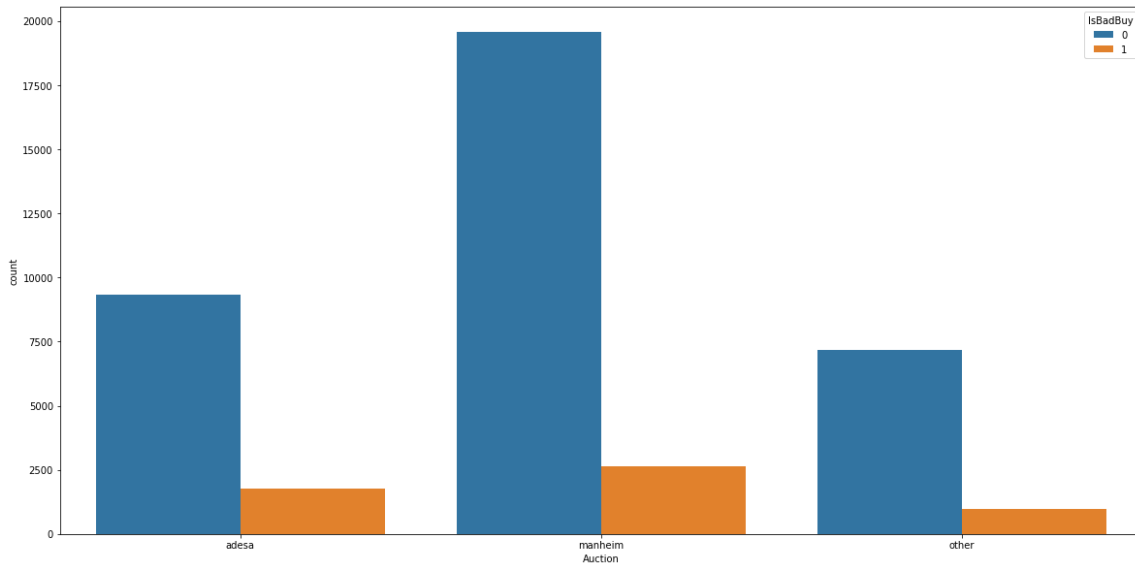
```
plotAllCols(df)
```

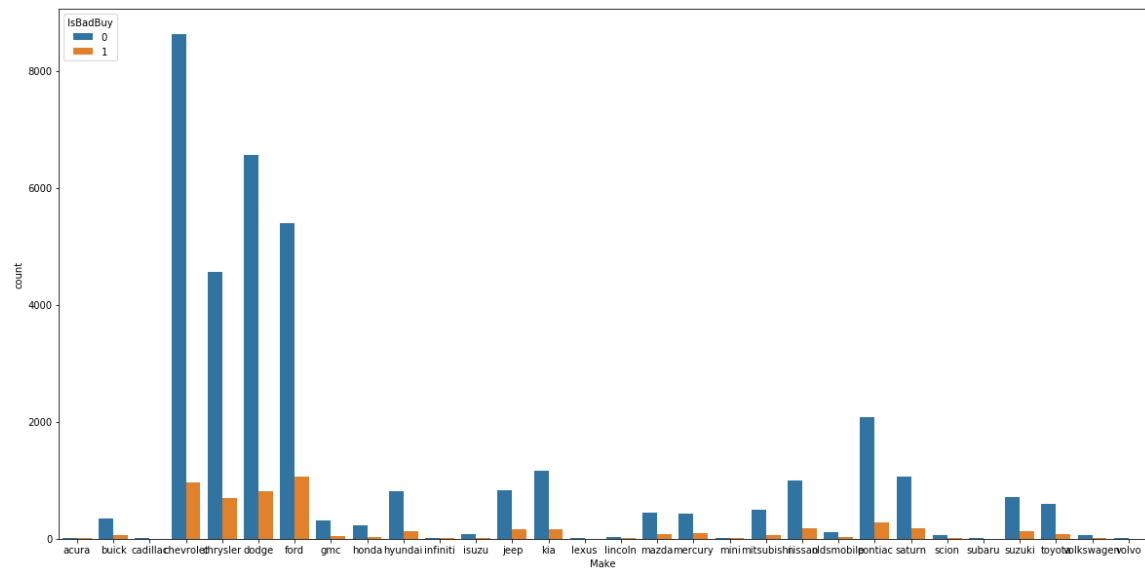
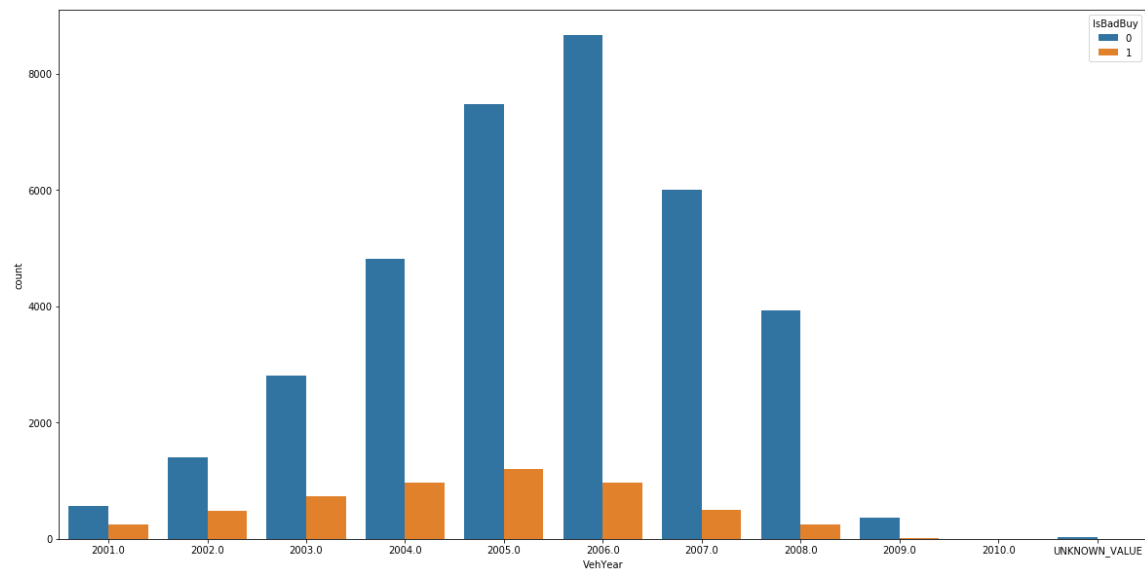
```
/home/chihcheng/python3env/lib/python3.6/site-packages/matplotlib/pyplot.py:514: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
```

```
max_open_warning, RuntimeWarning)
```

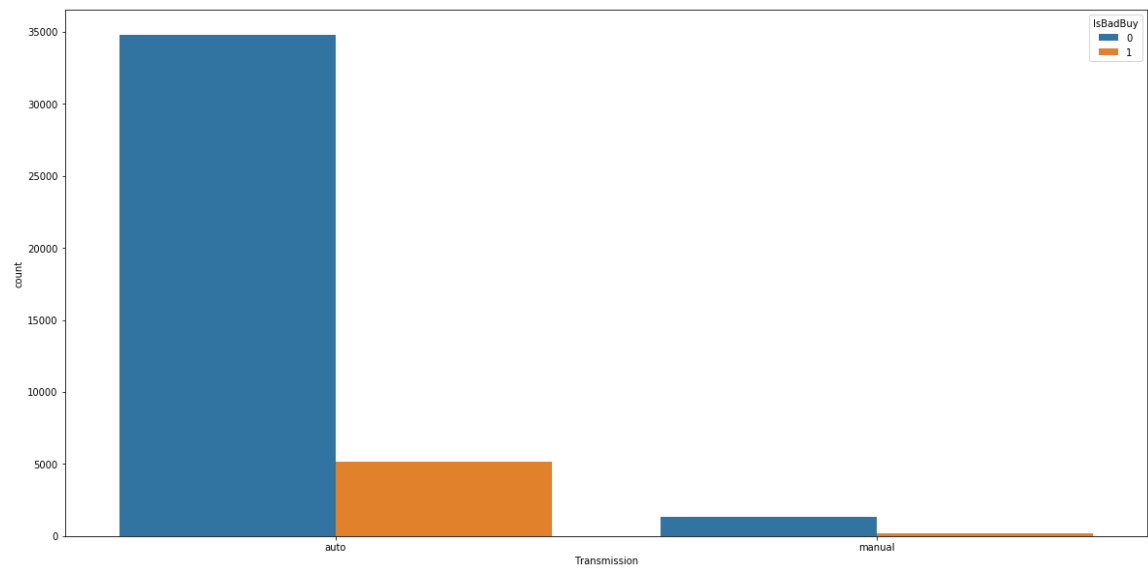
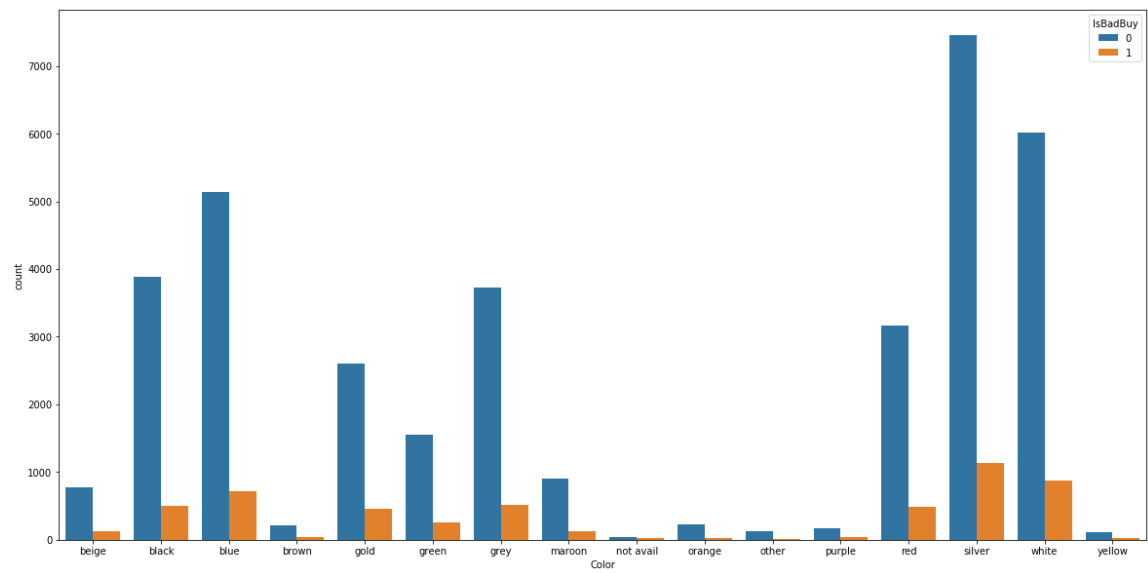
```
/home/chihcheng/python3env/lib/python3.6/site-packages/matplotlib/pyplot.py:514: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warning`).
```

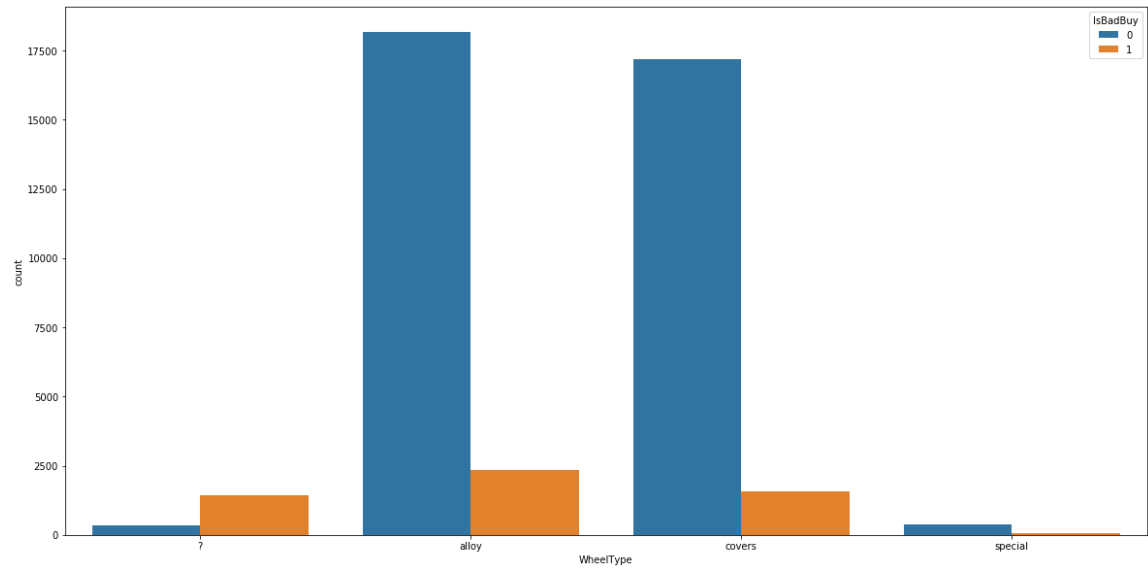
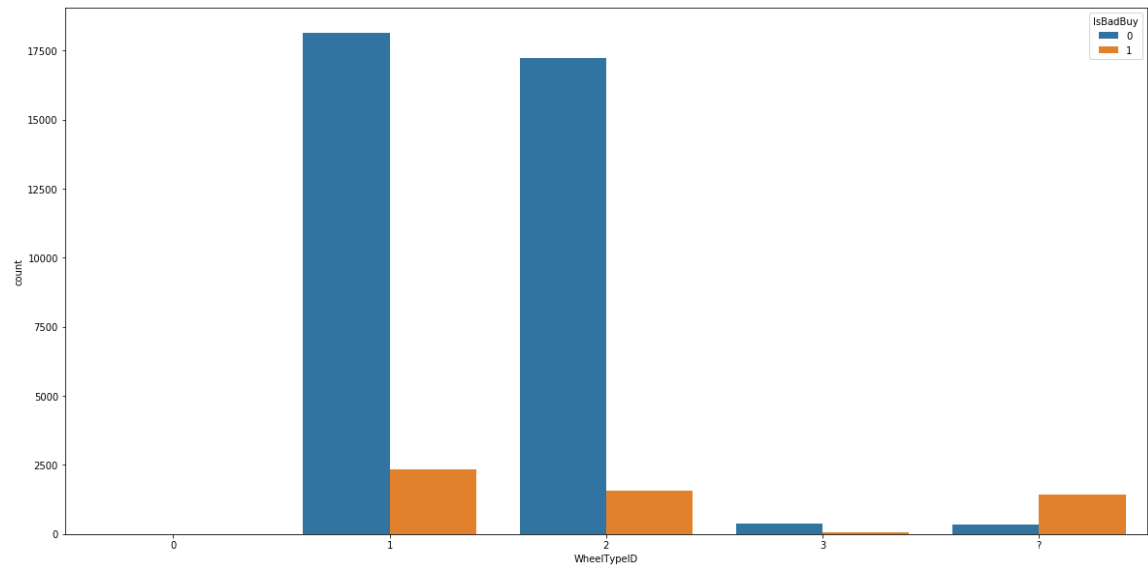
```
max_open_warning, RuntimeWarning)
```

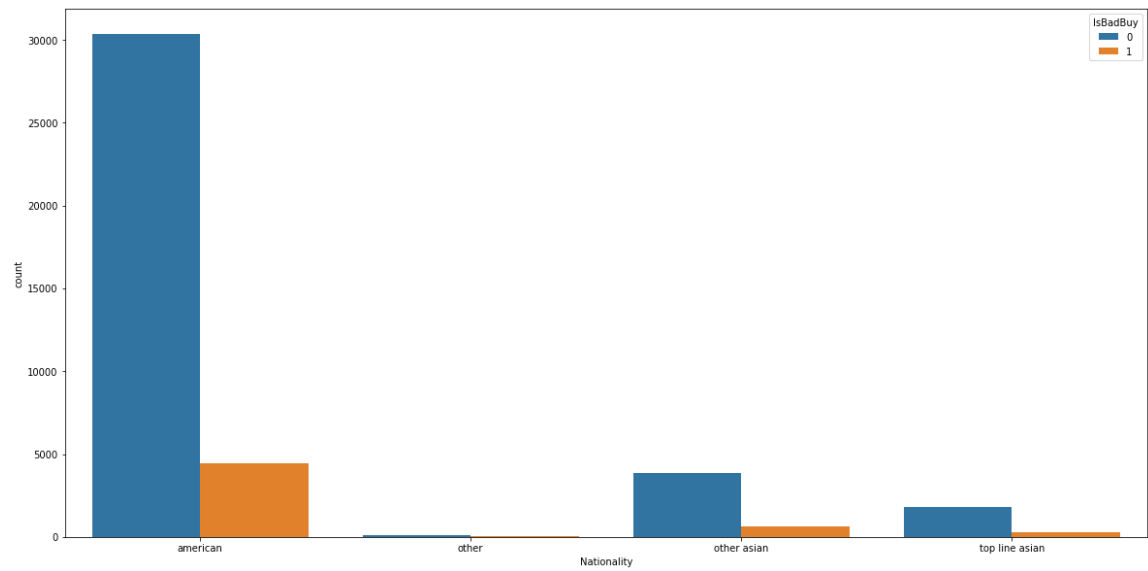
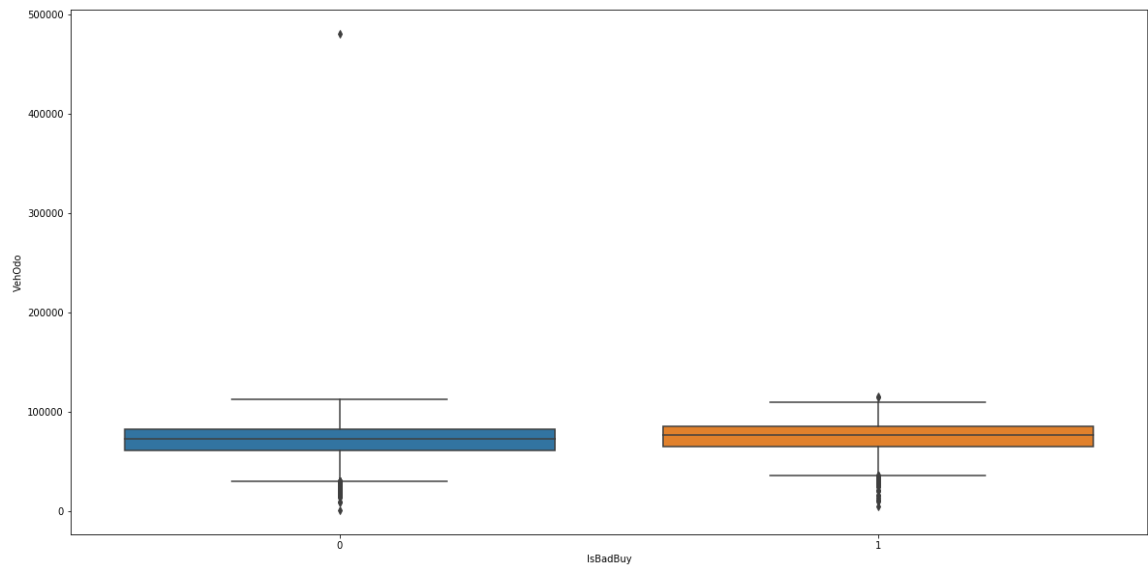


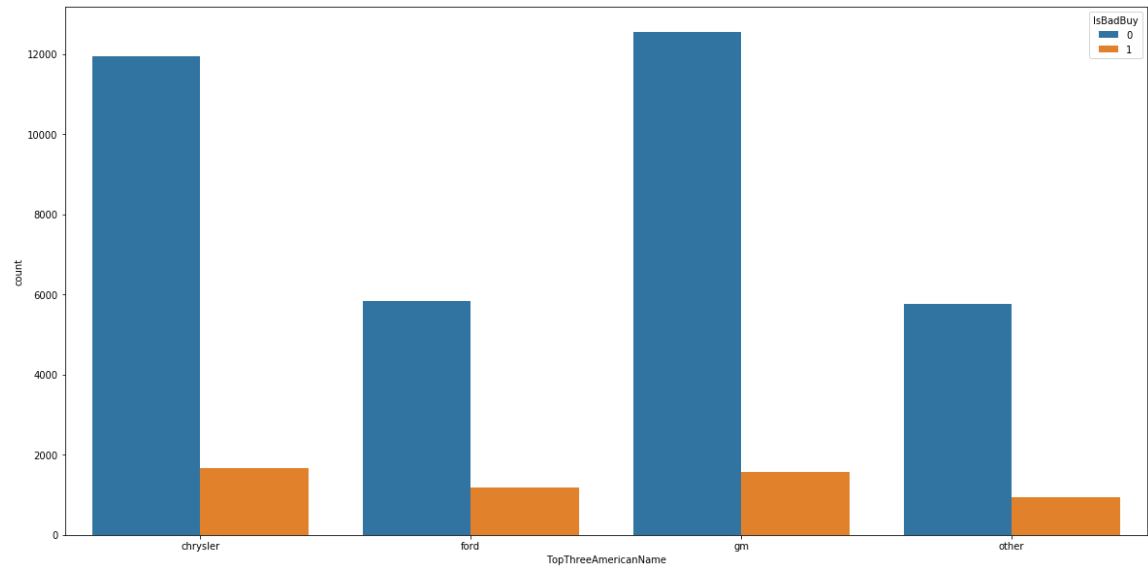
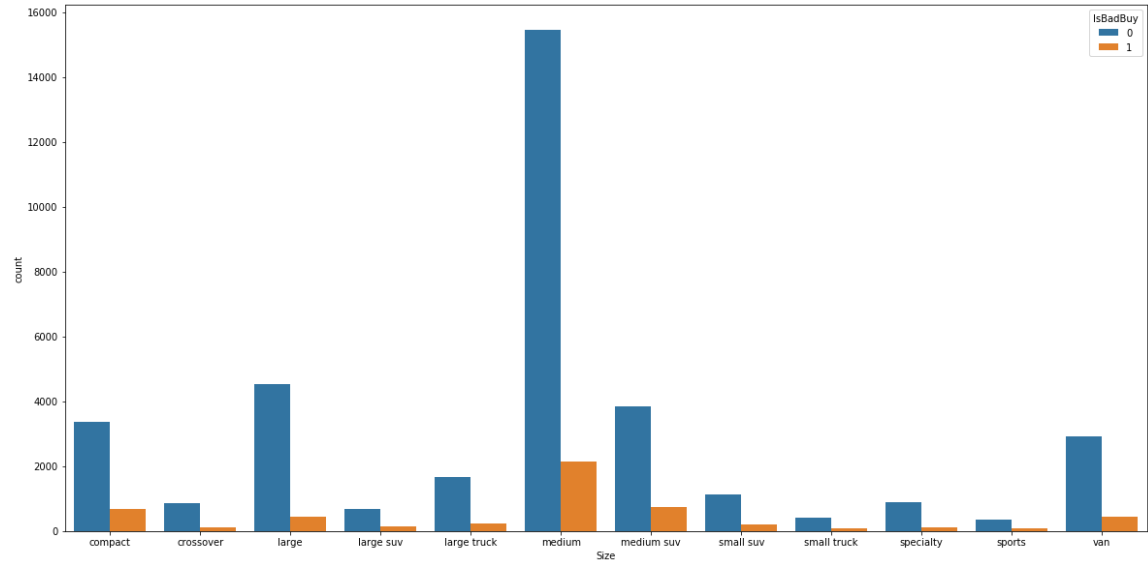


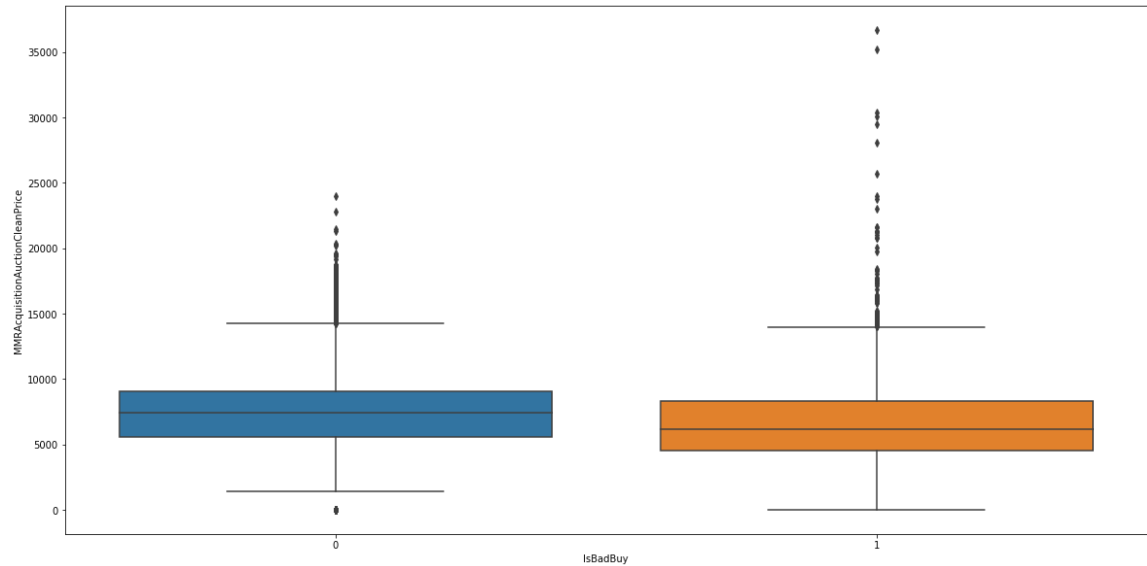
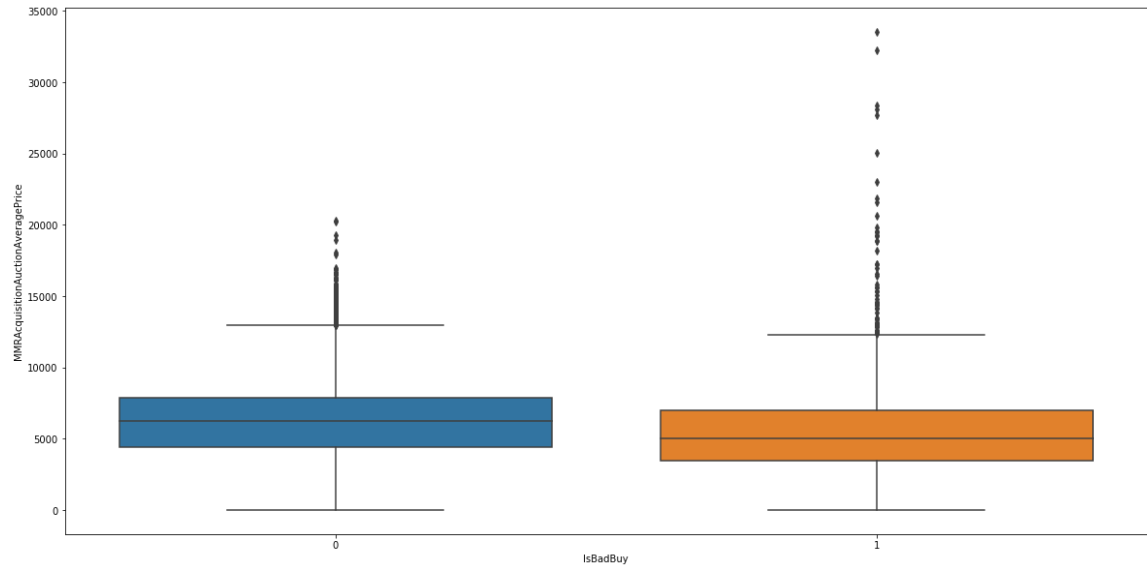


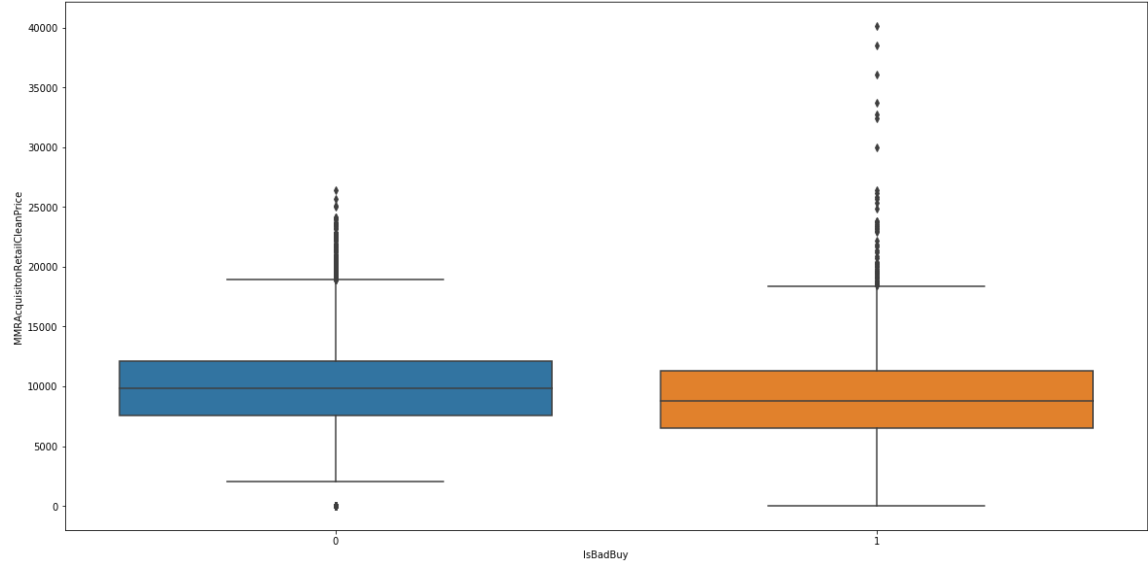
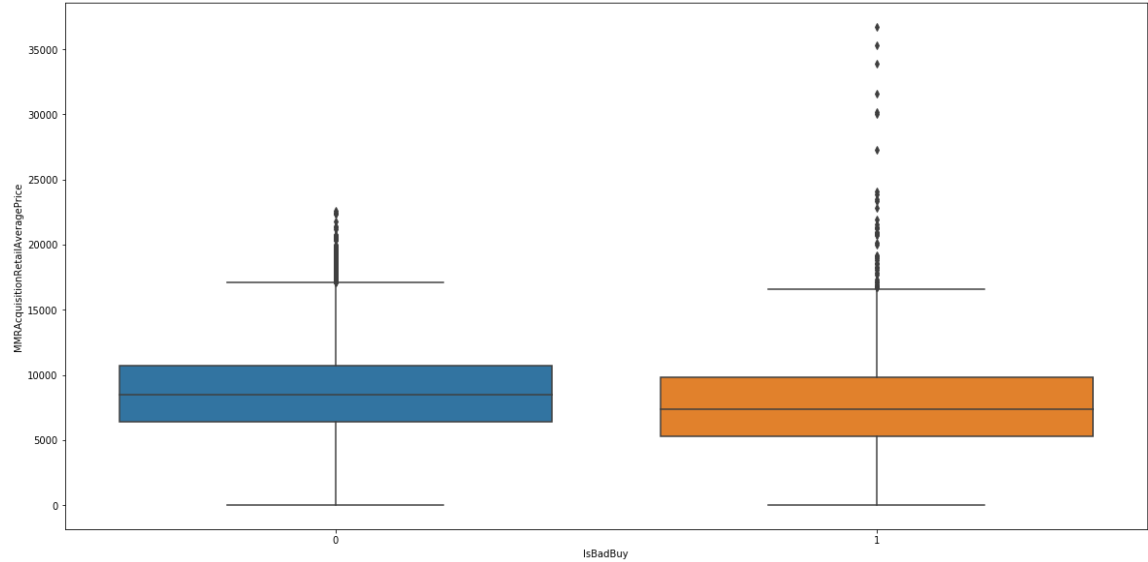


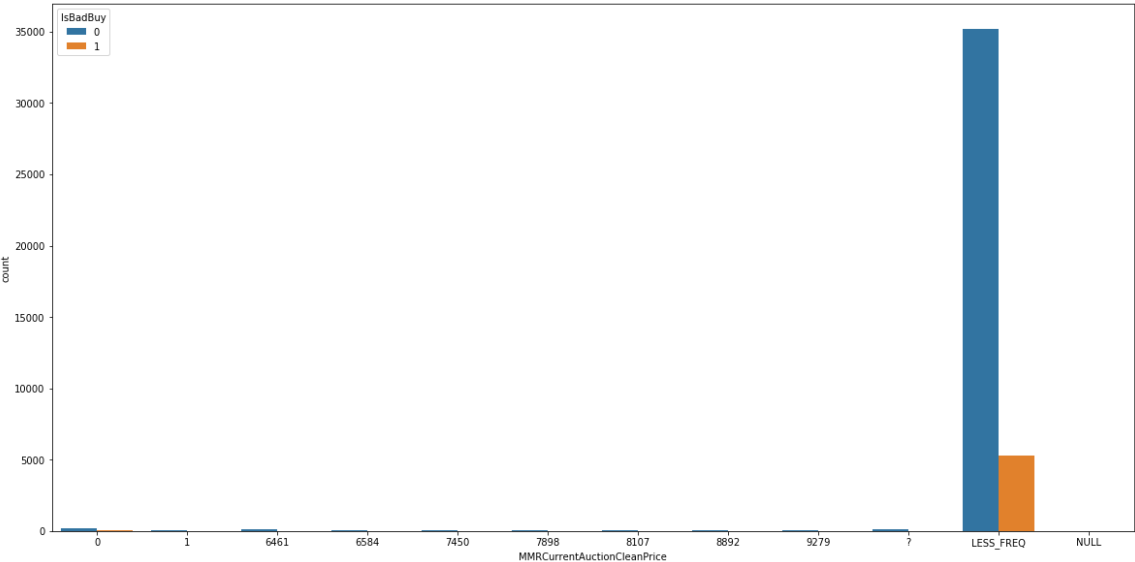
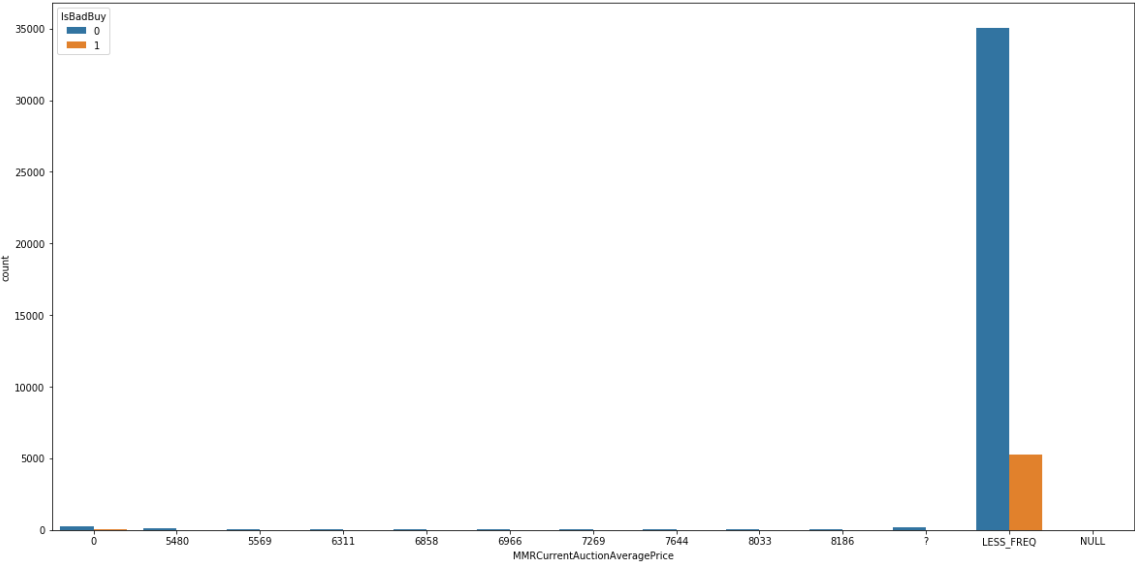


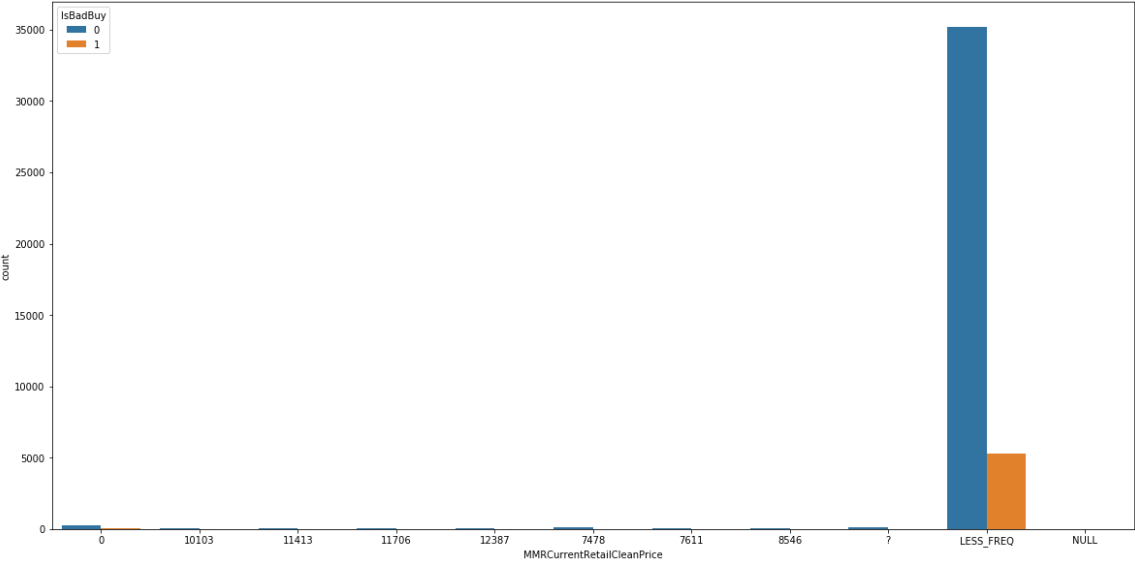
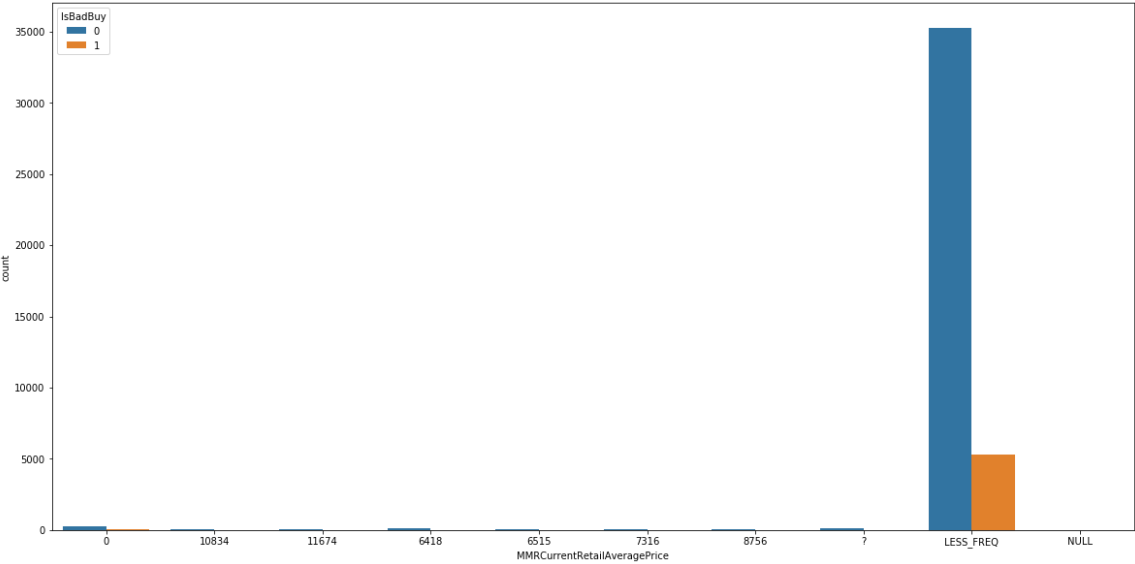




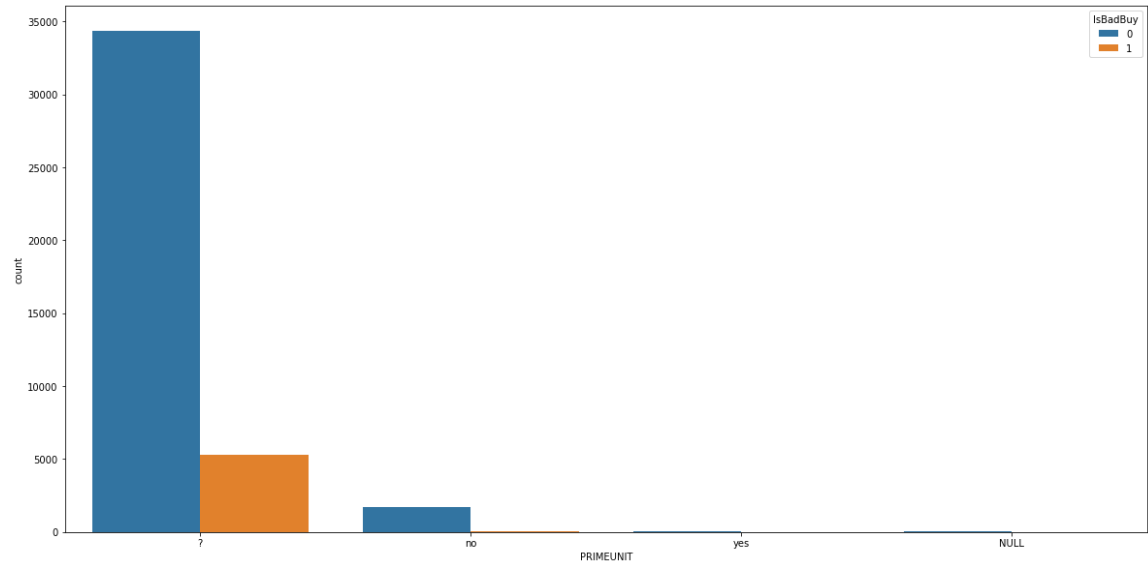
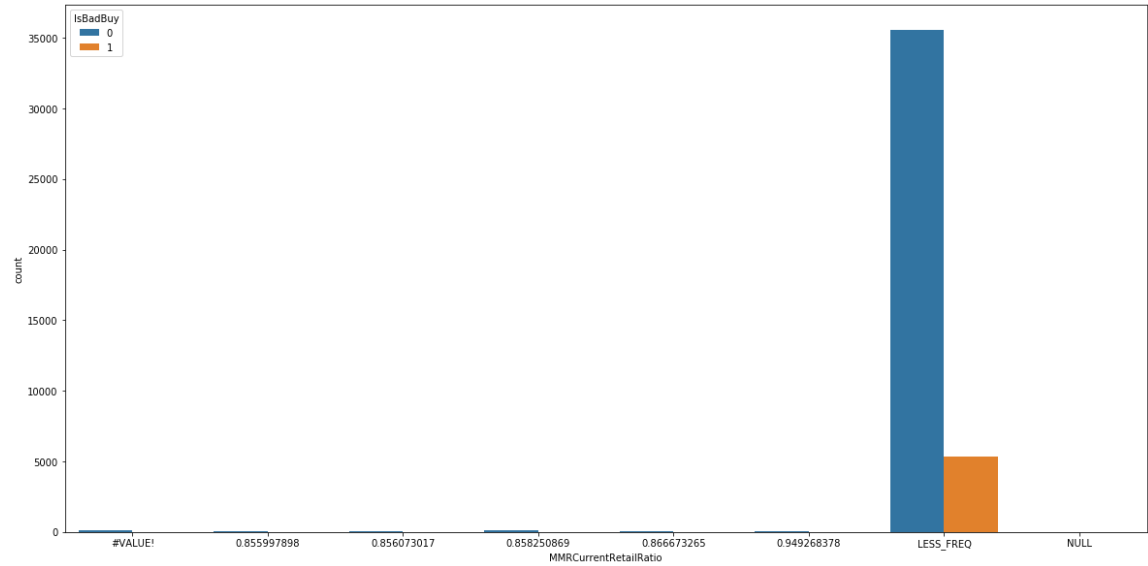


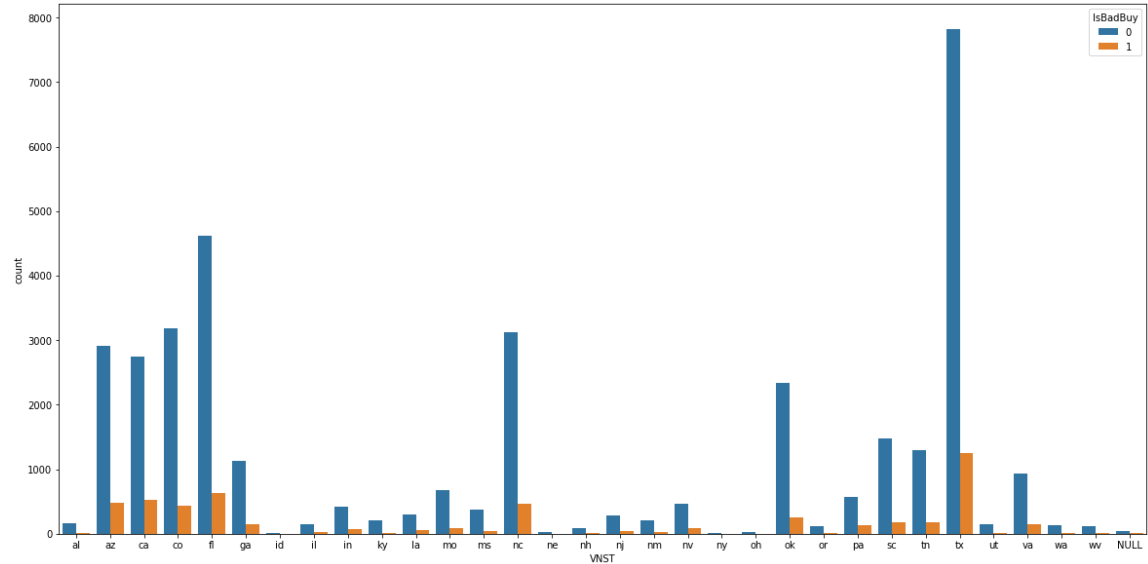
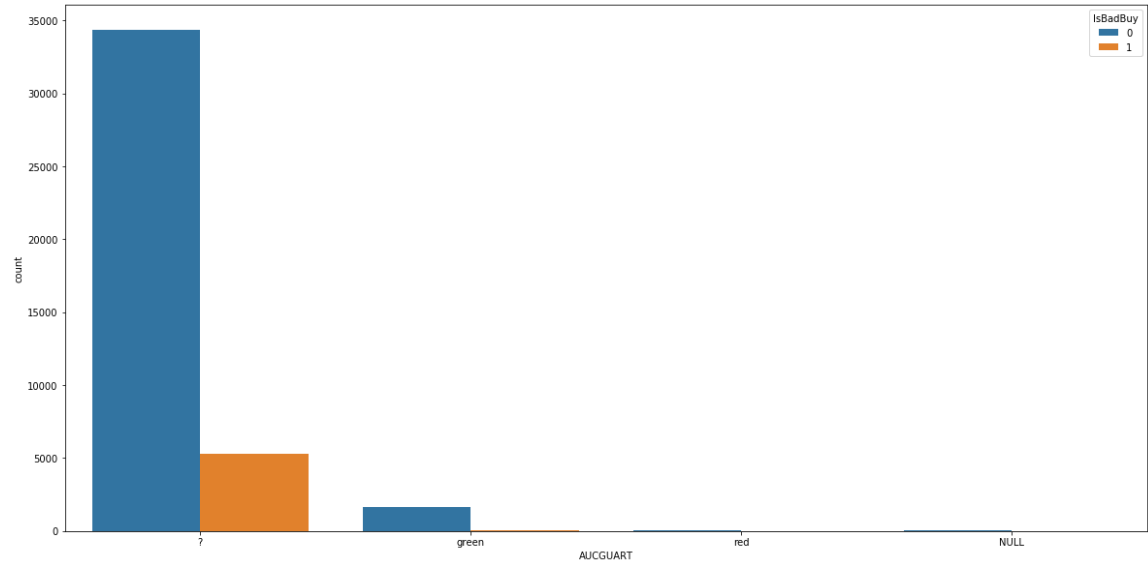


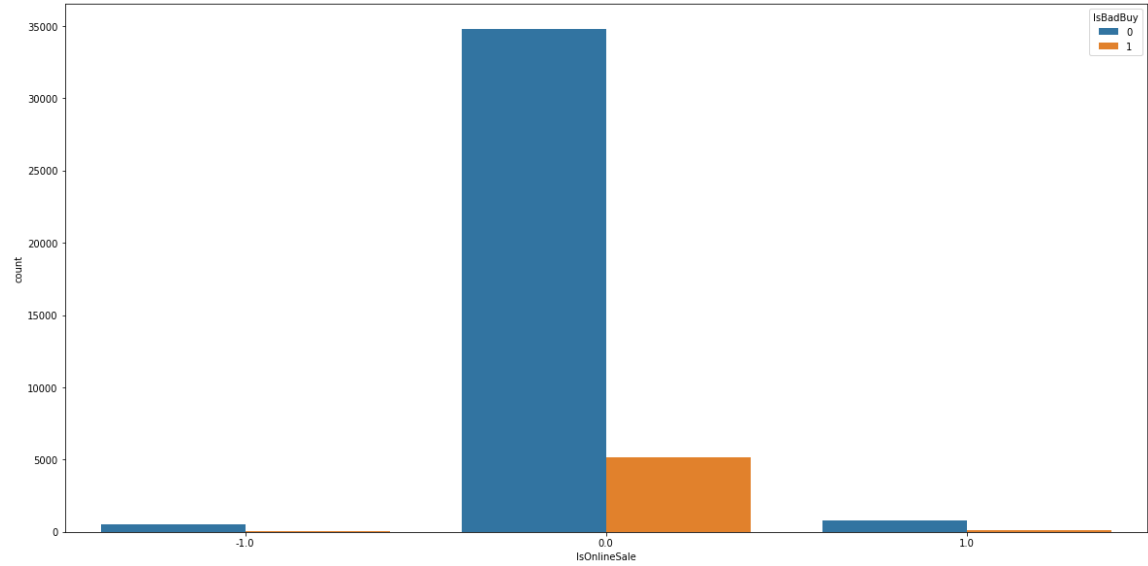
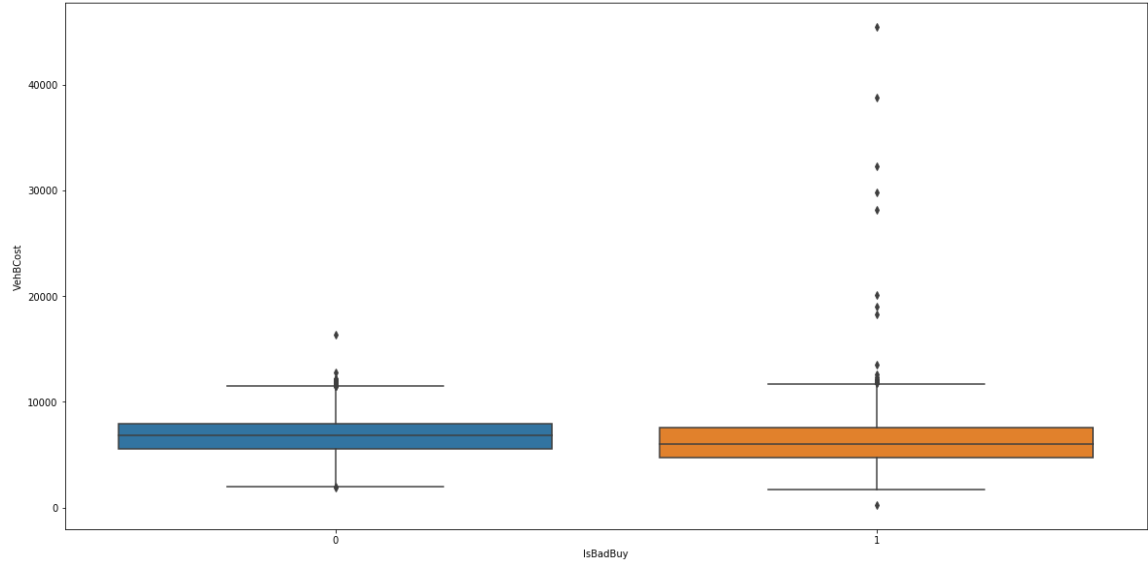


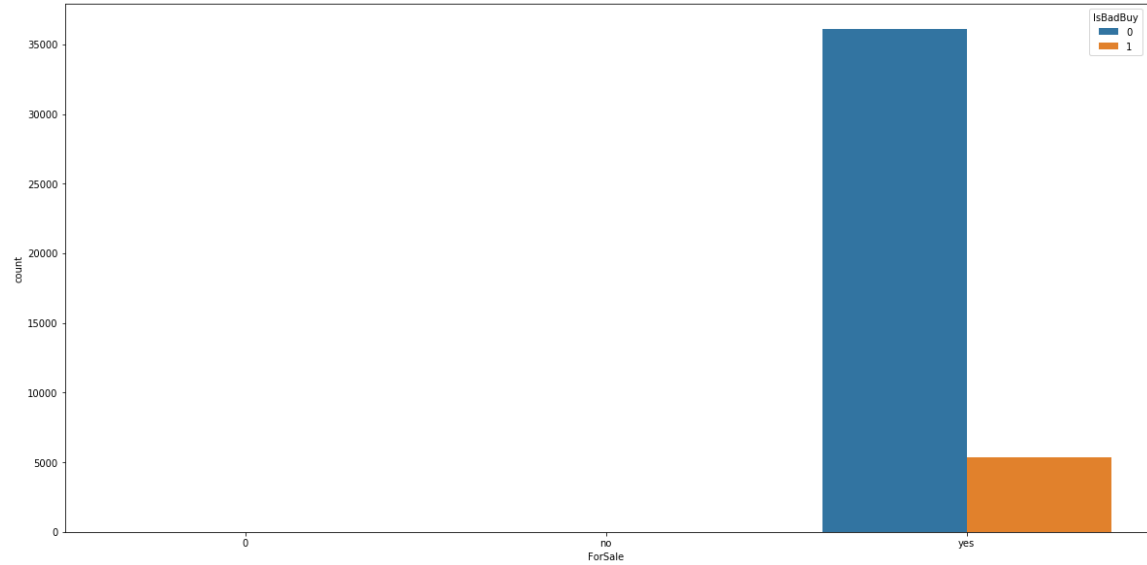
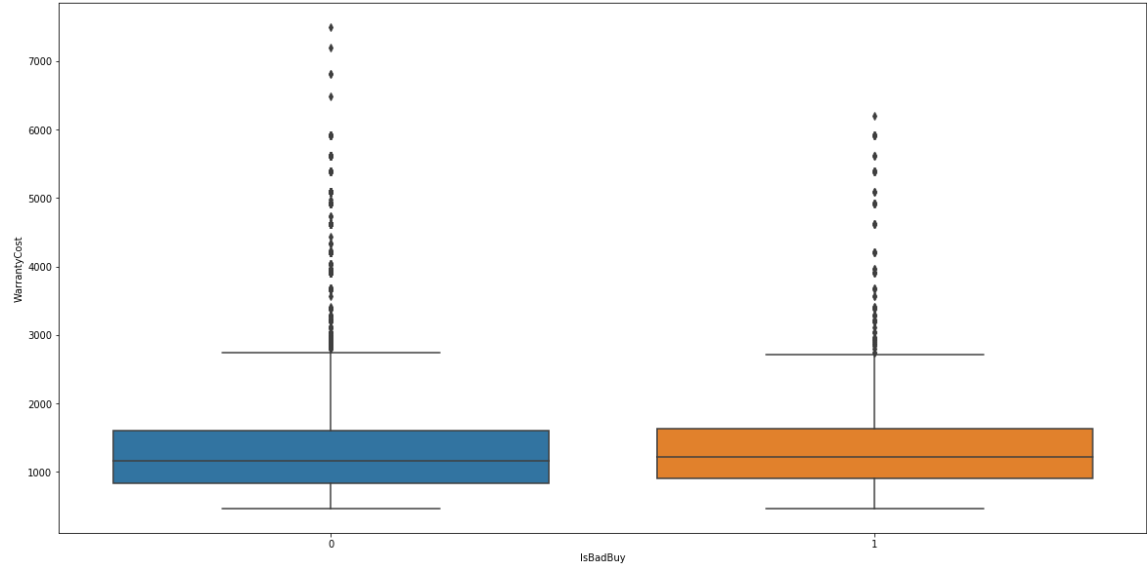












<Figure size 1440x720 with 0 Axes>

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

In [ ]:

## 5. What distribution scheme did you use? What data partitioning allocation did you set? Explain your selection.

In [11]:

```
# Change to the dummy

df = pd.get_dummies(df)

feature_names = df.drop("IsBadBuy", axis=1).columns

### Split to the training and test set.
# The test size is 3%

# y = df['IsBadBuy']
# X = df.drop(['IsBadBuy'], axis=1)
# X_mat = X.as_matrix()

# X_train, X_test, y_train, y_test = train_test_split(X_mat, y, test_size=0.3, s
stratify=y, random_state=rs)

X_train, X_test, y_train, y_test = train_test_split(df.drop("IsBadBuy", axis=1),
df['IsBadBuy'], test_size=0.3, stratify=df['IsBadBuy'], random_state=rs)

if ResamplingMethod == 'ros':
    print("Using ROS Resmapling")
    ros = RandomOverSampler(random_state=rs)
    X_train, y_train = ros.fit_resample(X_train, y_train)
elif ResamplingMethod == 'rus':
    print("Using RUS Resmapling")
    rus = RandomUnderSampler(random_state=rs)
    X_train, y_train = rus.fit_resample(X_train, y_train)
else:
    print("No Resampling Method Used")
```

Using ROS Resmapling

In [12]:

```
print("Number of Training: ", len(X_train))
print("Number of Test: ", len(X_test) )
```

Number of Training: 50546  
Number of Test: 12443

## Task 2. Predictive Modeling Using Decision Trees

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

In [13]:

```
def printLRTopImportant(model, top = 5):

    coef = model.coef_[0]
    indices = np.argsort(np.absolute(coef))
    indices = np.flip(indices, axis=0)
    indices = indices[:top]
    for i in indices:
        print(feature_names[i], ': ', coef[i])

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
```

In [14]:

```
# simple decision tree training
model = DecisionTreeClassifier(random_state=rs)
model.fit(X_train, y_train)
```

Out[14]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False, random_state=101,
                        splitter='best')
```

**a. What is the classification accuracy on training and test datasets?**

In [15]:

```
print("Train accuracy:", model.score(X_train, y_train))
print("Test accuracy:", model.score(X_test, y_test))
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
confusion_matrix(y_test, y_pred) ## Confusion Matrix on the TestSet
```

Train accuracy: 0.9994856170616864

Test accuracy: 0.8256047576950896

	precision	recall	f1-score	support
0	0.90	0.90	0.90	10832
1	0.33	0.33	0.33	1611
micro avg	0.83	0.83	0.83	12443
macro avg	0.61	0.61	0.61	12443
weighted avg	0.83	0.83	0.83	12443

Out[15]:

```
array([[9742, 1090],
       [1080, 531]])
```

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

In [16]:

```
print("Number of nodes: ", model.tree_.node_count)
```

Number of nodes: 7451

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

In [ ]:

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

In [17]:

```
visualize_decision_tree(model, df.drop("IsBadBuy", axis=1).columns, "Tree_Struct.png")
```

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

In [18]:

```
analyse_feature_importance(model, df.drop("IsBadBuy", axis=1).columns, 5)
```

```
WheelTypeID_? : 0.13551426074337208  
VehBCost : 0.12697398520218173  
VehOdo : 0.09322697324736888  
MMRAcquisitionAuctionCleanPrice : 0.0728830956066675  
MMRAcquisitionRetailAveragePrice : 0.06240815240105831
```

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

In [ ]:

**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.**

In [ ]:

## 2. Python: Build another decision tree tuned with GridSearchCV

In [ ]:



In [19]:

```
# grid search CV
params = {'criterion': ['gini', 'entropy'],
          'max_depth': list(range(1, 6000, 1000)) + [None],
          'splitter': ['best', 'random'],
          'min_samples_leaf': range(1, 4),
          'min_samples_split': [2, 0.5, 0.3],
          'max_features': ['auto', 'sqrt', 'log2', None],
          'class_weight': ['balanced', None]
        }

cv = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(random_state=rs), cv=3)
cv.fit(X_train, y_train)
```

Out[19]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None, criterion
='gini', max_depth=None,
             max_features=None, max_leaf_nodes=None,
             min_impurity_decrease=0.0, min_impurity_split=None,
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, presort=False, random_state=101,
             splitter='best'),
             fit_params=None, iid='warn', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'], 'max_depth':
[1, 1001, 2001, 3001, 4001, 5001, None], 'splitter': ['best', 'random'], 'min_samples_leaf': range(1, 4), 'min_samples_split': [2, 0.5, 0.3], 'max_features': ['auto', 'sqrt', 'log2', None], 'class_weight': ['balanced', None]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring=None, verbose=0)
```

**a. What is the classification accuracy on training and test datasets?**

In [20]:

```

print("Train accuracy:", cv.score(X_train, y_train))
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_)

dt_model = cv.best_estimator_

```

Train accuracy: 0.9994856170616864

Test accuracy: 0.8243188941573576

	precision	recall	f1-score	support
0	0.90	0.90	0.90	10832
1	0.33	0.33	0.33	1611
micro avg	0.82	0.82	0.82	12443
macro avg	0.61	0.62	0.61	12443
weighted avg	0.83	0.82	0.83	12443

```

{'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 10
01, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}

```

**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?**

In [21]:

```

print("Number of nodes: ", cv.best_estimator_.tree_.node_count)

```

Number of nodes: 13663

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

In [ ]:

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

In [22]:

```

visualize_decision_tree(cv.best_estimator_, df.drop("IsBadBuy", axis=1).columns,
"Tree_Struct_CV.png")

```

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

In [23]:

```
analyse_feature_importance(cv.best_estimator_, df.drop("IsBadBuy", axis=1).columns, 5)
```

```
MMRAcquisitionAuctionAveragePrice : 0.07556813524312803  
MMRAcquisitionRetailAveragePrice : 0.07453858047632252  
Veh0do : 0.07357030511029244  
MMRAcquisitionAuctionCleanPrice : 0.07355390254270788  
MMRAcquisitionRetailCleanPrice : 0.0672916389998756
```

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

In [ ]:

**g. What are the parameters used? Explain your choices.**

In [ ]:

**3. 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.**

In [ ]:

**4. From the better model, can you identify which cars could potential be “kicks”? Can you provide some descriptive summary of those cars?**

In [ ]:

In [ ]:

## Task 3. Predictive Modeling Using Regression

**1. In preparation for regression, is any imputation of missing values needed for this data set? List the variables that needed this.**

In [24]:

```
# We've already done this in the prep_data function
```

**2. Apply transformation method(s) to the variable(s) that need it. List the variables that needed it**

In [25]:

```
## Doing the log transformation

### Q: It's enough?
columns_to_transform = interval_cols

def logTransformation(df):
    df_log = df.copy()

    for col in columns_to_transform:
        df_log[col] = df_log[col].apply(lambda x: x+1)
        df_log[col] = df_log[col].apply(np.log)

    return df_log

df_log = logTransformation(df)
X_train_log, X_test_log, y_train_log, y_test_log = train_test_split(df_log.drop(
    ['IsBadBuy'], axis=1), df_log['IsBadBuy'], test_size=0.3, stratify=df_log['IsBa
dBuy'],
    random_state=rs)

# Standardise
scaler_log = StandardScaler()
X_train_log = scaler_log.fit_transform(X_train_log, y_train_log)
X_test_log = scaler_log.transform(X_test_log)

/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/prepr
ocessing/data.py:645: DataConversionWarning: Data with input dtype u
int8, float64 were all converted to float64 by StandardScaler.
    return self.partial_fit(X, y)
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/base.
py:467: DataConversionWarning: Data with input dtype uint8, float64
were all converted to float64 by StandardScaler.
    return self.fit(X, y, **fit_params).transform(X)
/home/chihcheng/python3env/lib/python3.6/site-packages/ipykernel_lau
ncher.py:26: DataConversionWarning: Data with input dtype uint8, flo
at64 were all converted to float64 by StandardScaler.
```

### 3. Build a regression model using the default regression method with all inputs. Once you done it, build another one and tune it using GridSearchCV. Answer the followings:

In [26]:

```
### Traing Logistic Regression
model = LogisticRegression(random_state=rs)
model.fit(X_train_log, y_train_log)
```

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[26]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='warn',
                    n_jobs=None, penalty='l2', random_state=101, solver='warn',
                    tol=0.0001, verbose=0, warm_start=False)
```

In [27]:

```
## GridSearch for Logistic Regression
params = {
    'C': [pow(10, x) for x in range(-4, 1)],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    'max_iter': [30, 50, 100],
    'warm_start': [True, False],
    'class_weight': ['balanced', None]
}

cv = GridSearchCV(param_grid=params, estimator=LogisticRegression(random_state=rs), cv=3, n_jobs=-1)
cv.fit(X_train_log, y_train_log)
```

Out[27]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
             fit_intercept=True,
             intercept_scaling=1, max_iter=100, multi_class='warn',
             n_jobs=None, penalty='l2', random_state=101, solver='warn',
             tol=0.0001, verbose=0, warm_start=False),
             fit_params=None, iid='warn', n_jobs=-1,
             param_grid={'C': [0.0001, 0.001, 0.01, 0.1, 1], 'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'], 'max_iter': [30, 50, 100], 'warm_start': [True, False], 'class_weight': ['balanced', None]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring=None, verbose=0)
```

**h. Name the regression function used.**

In [ ]:

**i. How much was the difference in performance of two models build, default and optimal?**

In [28]:

```
print("Train accuracy:", model.score(X_train_log, y_train_log))
print("Test accuracy:", model.score(X_test_log, y_test_log))
print("GridSearch Train accuracy:", cv.score(X_train_log, y_train_log))
print("GridSearch Test accuracy:", cv.score(X_test_log, y_test_log))
```

```
Train accuracy: 0.8963593152619433
Test accuracy: 0.8982560475769509
GridSearch Train accuracy: 0.895945992491303
GridSearch Test accuracy: 0.8984167805191674
```

**j. Show the set parameters for the best model. What are the parameters used? Explain your decision. What are the optimal parameters?**

In [29]:

```
print("The best model parameters: ", cv.best_params_)
```

```
The best model parameters: {'C': 0.001, 'class_weight': None, 'max_
iter': 30, 'solver': 'newton-cg', 'warm_start': True}
```

**k. Report which variables are included in the regression model.**

In [ ]:

**l. Report the top-5 important variables (in the order) in the model.**

In [30]:

```
def printLRTopImportant(model, top = 5):
    coef = model.coef_[0]
    indices = np.argsort(np.absolute(coef))
    indices = np.flip(indices, axis=0)
    indices = indices[:top]
    for i in indices:
        print(feature_names[i], ': ', coef[i])
```

In [31]:

```
printLRTopImportant(model, 5)
```

```
MMRAcquisitionAuctionAveragePrice : -0.7971823901149314
MMRAcquisitionRetailAveragePrice : 0.7882376918654318
WheelTypeID_? : 0.590062075011876
WheelTypeID_1 : -0.42595465288544654
WheelType_covers : -0.3800420459666399
```

### m. What is classification accuracy on training and test datasets?

In [32]:

```
y_pred = model.predict(X_test_log)
print("Classification Report: \n\n",classification_report(y_test_log, y_pred))

y_pred = cv.predict(X_test_log)
print("GridSearch Classification Report: \n\n",classification_report(y_test_log,
y_pred))
log_reg_model = cv.best_estimator_
```

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.84	0.27	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.87	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

GridSearch Classification Report:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.84	0.27	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.87	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

### n. Report any sign of overfitting.

In [33]:

```
## The GridSearch Precision and Recall is weird
```

## 4. Build another regression model using the subset of inputs selected by RFE and selection by model method. Answer the followings:

In [34]:

```
rfe = RFECV(estimator = LogisticRegression(random_state=rs), cv=3)
rfe.fit(X_train_log, y_train_log)
X_train_rfe = rfe.transform(X_train_log)
X_test_rfe = rfe.transform(X_test_log)

selectmodel = SelectFromModel(dt_model, prefit=True)
X_train_sel_model = selectmodel.transform(X_train_log)
X_test_sel_model = selectmodel.transform(X_test_log)
```



file:///home/chihcheng/Downloads/Assignment1.html

58/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

[illegible]

file:///home/chihcheng/Downloads/Assignment1.html

62/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

64/136



65/136

66/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

68/136

file:///home/chihcheng/Downloads/Assignment1.html

70/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

72/136



73/136

74/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

76/136

77/136

78/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

80/136



file:///home/chihcheng/Downloads/Assignment1.html

82/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

84/136

[illegible]

86/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

88/136



file:///home/chihcheng/Downloads/Assignment1.html

90/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

92/136

file:///home/chihcheng/Downloads/Assignment1.html

file:///home/chihcheng/Downloads/Assignment1.html

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

96/136



file:///home/chihcheng/Downloads/Assignment1.html

98/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

100/136

file:///home/chihcheng/Downloads/Assignment1.html

102/136

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```





In [36]:

```
print("The RFE-selected features: \n\n", list(compress(feature_names, rfe.support_)))
print("\n\n")
print("The SelectFromModel features: \n\n", list(compress(feature_names, selectmodel.get_support())))
```

The RFE-selected features:

```
['VehOdo', 'MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice', 'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitionRetailCleanPrice', 'VehBCost', 'WarrantyCost', 'VehYear_2001.0', 'VehYear_2002.0', 'VehYear_2003.0', 'VehYear_2004.0', 'VehYear_2005.0', 'VehYear_2008.0', 'Make_honda', 'Make_nissan', 'Make_toyota', 'Make_volvo', 'Color_other', 'WheelTypeID_1', 'WheelTypeID_3', 'WheelTypeID_?', 'WheelType_?', 'WheelType_alloy', 'WheelType_covers', 'WheelType_special', 'Nationality_top line asian', 'Size_large', 'Size_large suv', 'Size_medium suv', 'Size_van', 'TopThreeAmericanName_gm', 'MMRCurrentAuctionAveragePrice_5480', 'MMRCurrentAuctionAveragePrice_5569', 'MMRCurrentAuctionAveragePrice_6311', 'MMRCurrentAuctionAveragePrice_7269', 'MMRCurrentAuctionAveragePrice_7644', 'MMRCurrentAuctionAveragePrice_8186', 'MMRCurrentAuctionCleanPrice_6461', 'MMRCurrentAuctionCleanPrice_6584', 'MMRCurrentAuctionCleanPrice_7898', 'MMRCurrentRetailAveragePrice_10834', 'MMRCurrentRetailAveragePrice_11674', 'MMRCurrentRetailAveragePrice_6418', 'MMRCurrentRetailAveragePrice_6515', 'MMRCurrentRetailAveragePrice_7316', 'MMRCurrentRetailAveragePrice_8756', 'MMRCurrentRetailCleanPrice_10103', 'MMRCurrentRetailCleanPrice_11413', 'MMRCurrentRetailCleanPrice_7478', 'MMRCurrentRetailCleanPrice_7611', 'MMRCurrentRetailCleanPrice_8546', 'MMRCurrentRetailRatio_#VALUE!', 'MMRCurrentRetailRatio_0.855997898', 'MMRCurrentRetailRatio_0.856073017', 'MMRCurrentRetailRatio_0.858250869', 'MMRCurrentRetailRatio_0.866673265', 'MMRCurrentRetailRatio_0.949268378', 'MMRCurrentRetailRatio_LESS_FREQ', 'PRIMEUNIT_?', 'PRIMEUNIT_no', 'AUCGUART_?', 'VNST_fl', 'VNST_id', 'VNST_ky', 'VNST_nc', 'VNST_ne', 'VNST_or', 'VNST_pa', 'VNST_tn']
```

The SelectFromModel features:

```
['VehOdo', 'MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice', 'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitionRetailCleanPrice', 'VehBCost', 'WarrantyCost', 'Auction_adesa', 'Auction_manheim', 'Auction_other', 'VehYear_2003.0', 'VehYear_2004.0', 'VehYear_2005.0', 'VehYear_2006.0', 'Make_chevrolet', 'Make_chrysler', 'Make_dodge', 'Make_ford', 'Color_black', 'Color_blue', 'Color_gold', 'Color_grey', 'Color_red', 'Color_silver', 'Color_white', 'WheelTypeID_?', 'WheelType_?', 'WheelType_alloy', 'WheelType_covers', 'Size_medium', 'TopThreeAmericanName_chrysler', 'TopThreeAmericanName_gm', 'VNST_az', 'VNST_ca', 'VNST_co', 'VNST_fl', 'VNST_nc', 'VNST_tx']
```

**b. Report the top-5 important variables (in the order) in the model.**

In [37]:

```

params = {
    'C': [pow(10, x) for x in range(-4, 1)],
    'solver' : ['newton-cg', "lbfgs", "liblinear", "sag", "saga"],
    'max_iter': [30, 50, 100],
    'warm_start': [True, False],
    'class_weight': ['balanced', None]
}
rfe_cv = GridSearchCV(param_grid=params, estimator=LogisticRegression(random_state=rs, verbose=True), cv=3, n_jobs=-1)
rfe_cv.fit(X_train_rfe, y_train_log)

selectModel_cv = GridSearchCV(param_grid=params, estimator=LogisticRegression(random_state=rs, verbose=True), cv=3, n_jobs=-1)
selectModel_cv.fit(X_train_sel_model, y_train_log)

```

[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear\_model/logistic.py:758: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.

"of iterations.", ConvergenceWarning)

[Parallel(n\_jobs=1)]: Done 1 out of 1 | elapsed: 0.2s finished

[LibLinear]

Out[37]:

```

GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
             intercept_scaling=1, max_iter=100, multi_class='warn',
             n_jobs=None, penalty='l2', random_state=101, solver='warn',
             tol=0.0001, verbose=True, warm_start=False),
             fit_params=None, iid='warn', n_jobs=-1,
             param_grid={'C': [0.0001, 0.001, 0.01, 0.1, 1], 'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'], 'max_iter': [30, 50, 100], 'warm_start': [True, False], 'class_weight': ['balanced', None]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring=None, verbose=0)

```

In [38]:

```
print("Top-5 important variables for RFE: \n")
printLRTopImportant(rfe_cv.best_estimator_, 5)
print("\n\n")
print("Top-5 important variables for selectModel \n")
printLRTopImportant(selectModel_cv.best_estimator_, 5)
```

Top-5 important variables for RFE:

```
MMRAcquisitionRetailAveragePrice : 0.5910445372774984
Color_not avail : -0.503177053104706
MMRAcquisitionAuctionAveragePrice : -0.4973269163098214
MMRAcquisitionAuctionCleanPrice : -0.4943539763541305
MMRAcquisitonRetailCleanPrice : 0.40966123480199057
```

Top-5 important variables for selectModel

```
Make_dodge : 0.2979660277780559
Make_chrysler : 0.29785458436277856
VehBCost : -0.18512753357969539
Make_gmc : -0.10026521847659335
Veh0do : 0.07440633435578373
```

**c. What are the parameters used? Explain your choices. What are the optimal parameters? Which regression function is being used?**

In [39]:

```
print("Optimal Parameters for RFE", rfe_cv.best_params_)
print("Optimal Parameters for selectModel", selectModel_cv.best_params_)
```

```
Optimal Parameters for RFE {'C': 1, 'class_weight': None, 'max_iter': 30, 'solver': 'lbfgs', 'warm_start': True}
Optimal Parameters for selectModel {'C': 0.001, 'class_weight': None, 'max_iter': 30, 'solver': 'liblinear', 'warm_start': True}
```

**d. Report any sign of overfitting**

In [ ]:

**e. What is classification accuracy on training and test datasets?**

In [40]:

```
print("GridSearch Train accuracy:", cv.score(X_train_log, y_train_log))
print("GridSearch Test accuracy:", cv.score(X_test_log, y_test_log))
print("\n\nRFE:\n")
print("Train accuracy:", rfe_cv.score(X_train_rfe, y_train_log))
print("Test accuracy:", rfe_cv.score(X_test_rfe, y_test_log))
print("\n\nselectModel:\n")
print("Train accuracy:", selectModel_cv.score(X_train_sel_model, y_train_log))
print("Test accuracy:", selectModel_cv.score(X_test_sel_model, y_test_log))
```

GridSearch Train accuracy: 0.895945992491303  
GridSearch Test accuracy: 0.8984167805191674

RFE:

Train accuracy: 0.8965659766472635  
Test accuracy: 0.8987382464036004

selectModel:

Train accuracy: 0.8954982261564427  
Test accuracy: 0.8980953146347344

**f. Did it improve/worsen the performance? Explain why those changes may have happened**

In [41]:

```

y_pred = rfe_cv.predict(X_test_rfe)
print("REF classification report: \n",classification_report(y_test, y_pred))
print("\n\n")
y_pred = selectModel_cv.predict(X_test_sel_model)
print("selectModel classification report: \n",classification_report(y_test, y_pred))

```

REF classification report:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.85	0.27	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.87	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

selectModel classification report:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.83	0.27	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.87	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

## Task4 - Predicting using neural network

**1. Build a Neural Network model using the default setting.  
Answer the following:**

In [42]:

```
model = MLPClassifier(random_state=rs)
model.fit(X_train_log, y_train_log)
```

```
/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
```

```
% self.max_iter, ConvergenceWarning)
```

Out[42]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=101, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

### a. What is the network architecture?

In [43]:

```
def printMLPArchitecture(model):
    print("Number of Layers: ", model.n_layers_)
    print("The First layer is Input Layer, and the last layer is the output layer")
    for i, w in enumerate(model.coefs_):
        print("{} Layer with hidden size {}".format(i+1, w.shape[0]))
        if (i+1) == len(model.coefs_):
            print("{} Layer with hidden size {}".format(i+2, w.shape[1]))

    print("The activation function: ", model.activation)

printMLPArchitecture(model)
```

```
Number of Layers: 3
```

```
The First layer is Input Layer, and the last layer is the output layer
```

```
1 Layer with hidden size 198
```

```
2 Layer with hidden size 100
```

```
3 Layer with hidden size 1
```

```
The activation function: relu
```

### b. How many iterations are needed to train this network?

In [44]:

```
print("Number of iterations it ran: ", model.n_iter_)
```

```
Number of iterations it ran: 200
```

### c. Do you see any sign of over-fitting?

In [45]:

```
# fig = plt.figure(figsize=(10, 5))
# plt.ylabel('Accuracy', fontsize=15)
# plt.xlabel('Number of iterations', fontsize=15)
# plt.title('Validation Accuracy', fontsize=20, fontweight = "bold")
# plt.plot(model.validation_scores_, label="Validation Accuracy")
```

### d. Did the training process converge and resulted in the best model?

In [46]:

```
fig = plt.figure(figsize=(10, 5))
plt.ylabel('Loss', fontsize=15)
plt.xlabel('Number of iterations', fontsize=15)
plt.title('Training Loss', fontsize=20, fontweight = "bold")
plt.plot(model.loss_curve_, label="Training Loss")
```

### The Loss curve is still decreasing

Out[46]:

[<matplotlib.lines.Line2D at 0x7f9c8249ff98>]



### e. What is classification accuracy on training and test datasets?

In [47]:

```
print("MLP Train accuracy:", model.score(X_train, y_train))
print("MLP Test accuracy:", model.score(X_test, y_test))
print("\n\n")
y_pred = model.predict(X_test)
print("MLP classification report: \n", classification_report(y_test, y_pred))
```

MLP Train accuracy: 0.500217623550825

MLP Test accuracy: 0.8704492485734951

MLP classification report:

	precision	recall	f1-score	support
0	0.87	1.00	0.93	10832
1	0.00	0.00	0.00	1611
micro avg	0.87	0.87	0.87	12443
macro avg	0.44	0.50	0.47	12443
weighted avg	0.76	0.87	0.81	12443

## 2. Refine this network by tuning it with GridSearchCV.



In [48]:

```

# Default
# params = {'hidden_layer_sizes': [(3,), (5,), (7,), (9,)], 'alpha': [0.01, 0.001, 0.0001]}

params = [
    {
        'hidden_layer_sizes': [(128,)],
        'activation': ['logistic', 'relu', 'identity'],
        'solver' : ['adam'],
        'batch_size': [32, 64],
        'shuffle': [True],
        'learning_rate_init': [pow(10, x) for x in range(-4, -2)],
        'n_iter_no_change': [10, 20],
        'warm_start': [True, False],
    },
    {
        'hidden_layer_sizes': [(128,)],
        'learning_rate' : ['constant', 'invscaling', 'adaptive'],
        'activation': ['logistic', 'relu', 'identity'],
        'solver' : ['sgd'],
        'shuffle': [True],
        'batch_size': [32, 64],
        'learning_rate_init': [pow(10, x) for x in range(-4, -2)],
        'n_iter_no_change': [10, 20],
        'warm_start': [True, False],
    },
    {
        'hidden_layer_sizes': [(128,)],
        'activation': ['logistic', 'relu', 'identity'],
        'solver' : ['lbfgs'],
        'batch_size': [32, 64],
        'learning_rate_init': [pow(10, x) for x in range(-4, -2)],
        'n_iter_no_change': [10, 20],
        'warm_start': [True, False],
    }
]

cv = GridSearchCV(param_grid=params, estimator=MLPClassifier(random_state=rs, early_stopping = True, verbose=True), cv=3, n_jobs=-1)
# cv = GridSearchCV(param_grid=params, estimator=MLPClassifier(random_state=rs, early_stopping=True, max_iter = max_iter, n_iter_no_change = max_iter ), cv=3, n_jobs=-1)
cv.fit(X_train_log, y_train_log)

```

Iteration 1, loss = 0.33406174  
Validation score: 0.899449  
Iteration 2, loss = 0.31403847  
Validation score: 0.899449  
Iteration 3, loss = 0.31255767  
Validation score: 0.899449  
Iteration 4, loss = 0.31177683  
Validation score: 0.899449  
Iteration 5, loss = 0.31103730  
Validation score: 0.899449  
Iteration 6, loss = 0.31010433  
Validation score: 0.899449  
Iteration 7, loss = 0.30906946  
Validation score: 0.899449  
Iteration 8, loss = 0.30817736  
Validation score: 0.900138  
Iteration 9, loss = 0.30726278  
Validation score: 0.899449  
Iteration 10, loss = 0.30637227  
Validation score: 0.899449  
Iteration 11, loss = 0.30483040  
Validation score: 0.900138  
Iteration 12, loss = 0.30360087  
Validation score: 0.900482  
Iteration 13, loss = 0.30294773  
Validation score: 0.900138  
Iteration 14, loss = 0.30116919  
Validation score: 0.899793  
Iteration 15, loss = 0.29985837  
Validation score: 0.899105  
Iteration 16, loss = 0.29817458  
Validation score: 0.899793  
Iteration 17, loss = 0.29671322  
Validation score: 0.898416  
Iteration 18, loss = 0.29506197  
Validation score: 0.900138  
Iteration 19, loss = 0.29314780  
Validation score: 0.900482  
Iteration 20, loss = 0.29144124  
Validation score: 0.898416  
Iteration 21, loss = 0.28907336  
Validation score: 0.899793  
Iteration 22, loss = 0.28694002  
Validation score: 0.898416  
Iteration 23, loss = 0.28437917  
Validation score: 0.899449  
Iteration 24, loss = 0.28186406  
Validation score: 0.899793  
Iteration 25, loss = 0.27945294  
Validation score: 0.900138  
Iteration 26, loss = 0.27649281  
Validation score: 0.899793  
Iteration 27, loss = 0.27371054  
Validation score: 0.900138  
Iteration 28, loss = 0.27053313  
Validation score: 0.899793  
Iteration 29, loss = 0.26767378  
Validation score: 0.900826  
Iteration 30, loss = 0.26475819  
Validation score: 0.899793  
Iteration 31, loss = 0.26101040

Validation score: 0.900482  
Iteration 32, loss = 0.25803914  
Validation score: 0.900482  
Iteration 33, loss = 0.25452168  
Validation score: 0.900138  
Iteration 34, loss = 0.25113722  
Validation score: 0.898416  
Iteration 35, loss = 0.24768587  
Validation score: 0.897727  
Iteration 36, loss = 0.24420492  
Validation score: 0.897727  
Iteration 37, loss = 0.24099523  
Validation score: 0.899449  
Iteration 38, loss = 0.23717577  
Validation score: 0.897727  
Iteration 39, loss = 0.23445705  
Validation score: 0.895661  
Iteration 40, loss = 0.23030711  
Validation score: 0.896350  
Iteration 41, loss = 0.22705398  
Validation score: 0.897383  
Iteration 42, loss = 0.22393178  
Validation score: 0.895317  
Iteration 43, loss = 0.22061048  
Validation score: 0.899793  
Iteration 44, loss = 0.21711178  
Validation score: 0.893939  
Iteration 45, loss = 0.21407507  
Validation score: 0.892562  
Iteration 46, loss = 0.21096409  
Validation score: 0.896694  
Iteration 47, loss = 0.20773348  
Validation score: 0.895317  
Iteration 48, loss = 0.20389114  
Validation score: 0.896350  
Iteration 49, loss = 0.20139598  
Validation score: 0.892906  
Iteration 50, loss = 0.19787601  
Validation score: 0.891529  
Validation score did not improve more than tol=0.000100 for 20 consecutive epochs. Stopping.

Out[48]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=True, epsilon=1e-08,
                                     hidden_layer_sizes=(100,), learning_rate='constant',
                                     learning_rate_init=0.001, max_iter=200, momentum=0.9,
                                     n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
                                     random_state=101, shuffle=True, solver='adam', tol=0.0001,
                                     validation_fraction=0.1, verbose=True, warm_start=False),
             fit_params=None, iid='warn', n_jobs=-1,
             param_grid=[{'hidden_layer_sizes': [(128,)], 'activation':
                           ['logistic', 'relu', 'identity'], 'solver': ['adam'], 'batch_size':
                           [32, 64], 'shuffle': [True], 'learning_rate_init': [0.0001, 0.001],
                           'n_iter_no_change': [10, 20], 'warm_start': [True, False]}, {'hidden
                           _layer_sizes': [(128,)], 'learning_rate_init': [0.0001, 0.001],
                           'n_iter_no_change': [10, 20], 'warm_start': [True, False]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring=None, verbose=0)
```

### a. What is the network architecture?

In [49]:

```
print("Best Parameters of NN: ", cv.best_params_)
```

```
Best Parameters of NN: {'activation': 'logistic', 'batch_size': 64,
                        'hidden_layer_sizes': (128,), 'learning_rate_init': 0.001, 'n_iter_no_change': 20, 'shuffle': True, 'solver': 'adam', 'warm_start': True}
```

In [50]:

```
printMLPArchitecture(cv.best_estimator_)
```

Number of Layers: 3

The First layer is Input Layer, and the last layer is the output layer

1 Layer with hidden size 198

2 Layer with hidden size 128

3 Layer with hidden size 1

The activation function: logistic

### b. How many iterations are needed to train this network?

In [51]:

```
print("Number of iterations it ran: ", cv.best_estimator_.n_iter_)
```

Number of iterations it ran: 50

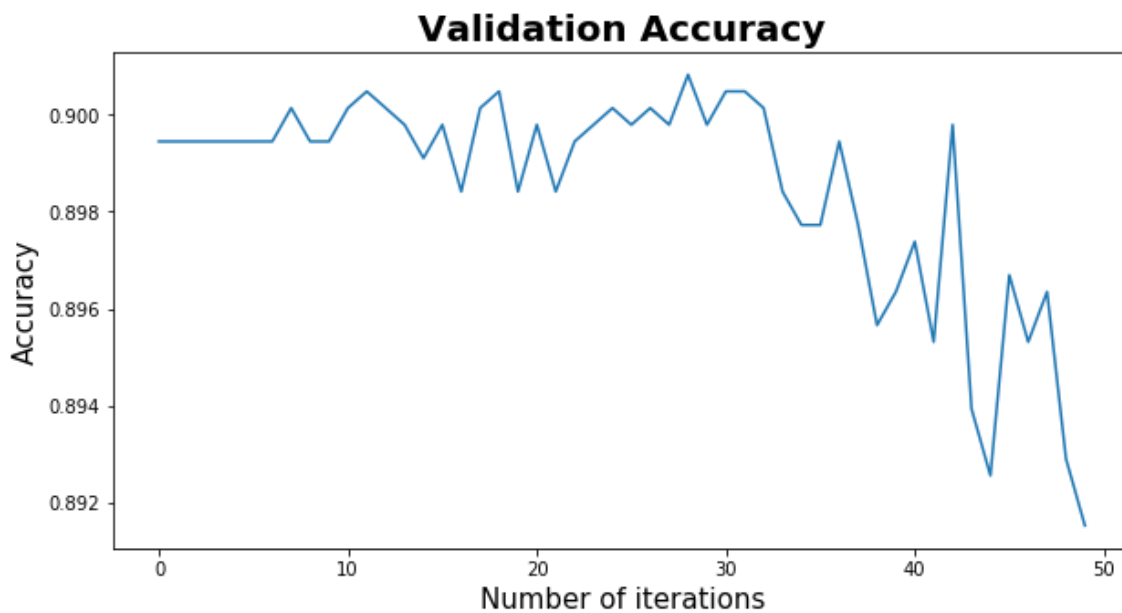
### c. Sign of overfitting?

In [52]:

```
fig = plt.figure(figsize=(10, 5))
plt.ylabel('Accuracy', fontsize=15)
plt.xlabel('Number of iterations', fontsize=15)
plt.title('Validation Accuracy', fontsize=20, fontweight = "bold")
plt.plot(cv.best_estimator_.validation_scores_, label="Validation Accuracy")
```

Out[52]:

[<matplotlib.lines.Line2D at 0x7f9c7a95c0b8>]



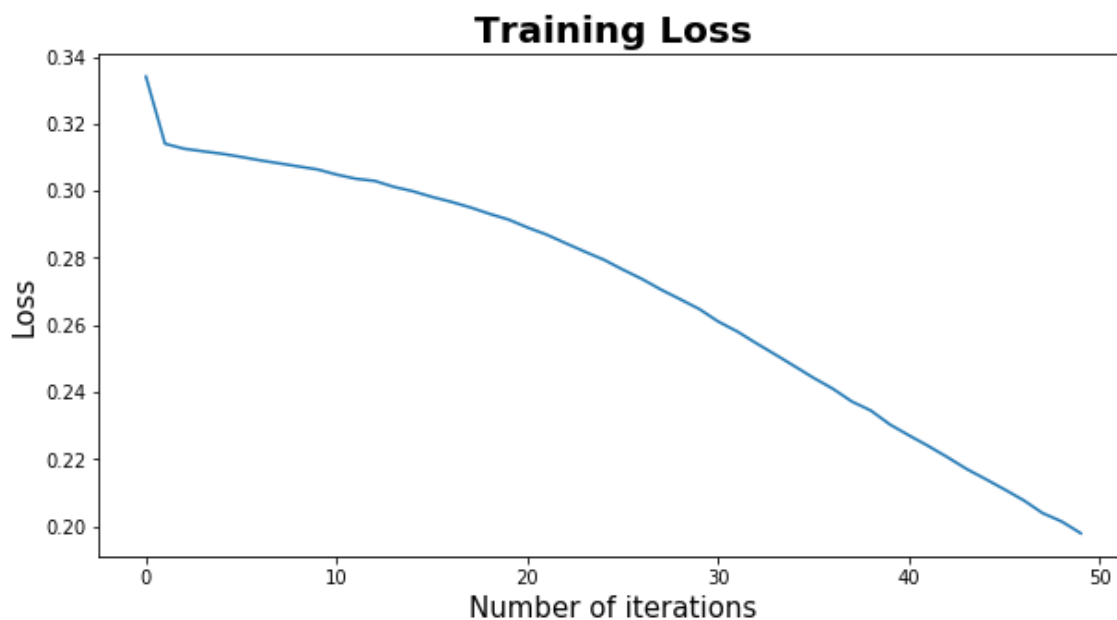
### d. Did the training process converge and resulted in the best model?

In [53]:

```
fig = plt.figure(figsize=(10, 5))
plt.ylabel('Loss',fontsize=15)
plt.xlabel('Number of iterations',fontsize=15)
plt.title('Training Loss',fontsize=20,fontweight="bold")
plt.plot(cv.best_estimator_.loss_curve_, label="Training Loss")
```

Out[53]:

[<matplotlib.lines.Line2D at 0x7f9c8bbbab38>]



**e. What is classification accuracy on training and test datasets? Is there any improvement in the outcome?**

In [54]:

```
print("GridSearch NN Train accuracy:", cv.score(X_train_log, y_train_log))
print("GridSearch NN Test accuracy:", cv.score(X_test_log, y_test_log))

print("\n\n")
y_pred = cv.predict(X_test_log)
print("GridSearch NN Classification Report: \n", classification_report(y_test_log
, y_pred))

print("Best Parameters of NN: ", cv.best_params_)
nn_model = cv.best_estimator_
```

GridSearch NN Train accuracy: 0.9049702063169497

GridSearch NN Test accuracy: 0.8975327493369766

GridSearch NN Classification Report:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.83	0.26	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.86	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

Best Parameters of NN: {'activation': 'logistic', 'batch\_size': 64, 'hidden\_layer\_sizes': (128,), 'learning\_rate\_init': 0.001, 'n\_iter\_no\_change': 20, 'shuffle': True, 'solver': 'adam', 'warm\_start': True}

**3. Would feature selection help here? Build another Neural Network model with inputs selected from RFE with regression (use the best model generated in Task 3) and selection with decision tree (use the best model from Task 2).**

In [55]:

```

params = [
    {
        'hidden_layer_sizes': [(128,)],
        'activation': ['logistic', 'relu', 'identity'],
        'solver' : ['adam'],
        'batch_size': [ 64],
        'shuffle': [True],
        'learning_rate_init': [pow(10, x) for x in range(-4, -2)],
        'n_iter_no_change': [10],
        'warm_start': [True, False],
    },
    {
        'hidden_layer_sizes': [(128,)],
        'learning_rate' : ['constant', 'invscaling', 'adaptive'],
        'activation': ['logistic', 'relu', 'identity'],
        'solver' : ['sgd'],
        'shuffle': [True],
        'batch_size': [64],
        'learning_rate_init': [pow(10, x) for x in range(-4, -2)],
        'n_iter_no_change': [10],
        'warm_start': [True, False],
    },
    {
        'hidden_layer_sizes': [(128,)],
        'activation': ['logistic', 'relu', 'identity'],
        'solver' : ['lbfgs'],
        'batch_size': [64],
        'learning_rate_init': [pow(10, x) for x in range(-4, -2)],
        'n_iter_no_change': [10],
        'warm_start': [True, False],
    }
]

rfe_cv = GridSearchCV(param_grid=params, estimator=MLPClassifier(random_state=rs
, early_stopping=True, verbose=True), cv=3, n_jobs=-1)
rfe_cv.fit(X_train_rfe, y_train_log)
modelSelect_cv = GridSearchCV(param_grid=params, estimator=MLPClassifier(random_
state=rs, early_stopping=True, verbose=True), cv=3, n_jobs=-1)
modelSelect_cv.fit(X_train_sel_model, y_train_log)

```



```
Iteration 1, loss = 0.38896433
Validation score: 0.871212
Iteration 2, loss = 0.36839117
Validation score: 0.871212
Iteration 3, loss = 0.35670821
Validation score: 0.871212
Iteration 4, loss = 0.34582363
Validation score: 0.871556
Iteration 5, loss = 0.33669776
Validation score: 0.878788
Iteration 6, loss = 0.32953443
Validation score: 0.892906
Iteration 7, loss = 0.32452803
Validation score: 0.898072
Iteration 8, loss = 0.32101373
Validation score: 0.898072
Iteration 9, loss = 0.31865502
Validation score: 0.897383
Iteration 10, loss = 0.31691035
Validation score: 0.897039
Iteration 11, loss = 0.31570717
Validation score: 0.896350
Iteration 12, loss = 0.31494236
Validation score: 0.896350
Iteration 13, loss = 0.31420999
Validation score: 0.897039
Iteration 14, loss = 0.31367637
Validation score: 0.896350
Iteration 15, loss = 0.31328425
Validation score: 0.896350
Iteration 16, loss = 0.31303482
Validation score: 0.896350
Iteration 17, loss = 0.31273916
Validation score: 0.896694
Iteration 18, loss = 0.31245818
Validation score: 0.896694
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 1, loss = 0.33545083
Validation score: 0.900826
Iteration 2, loss = 0.31426881
Validation score: 0.901171
Iteration 3, loss = 0.30968716
Validation score: 0.900826
Iteration 4, loss = 0.30638997
Validation score: 0.901860
Iteration 5, loss = 0.30413043
Validation score: 0.901515
Iteration 6, loss = 0.30213946
Validation score: 0.901171
Iteration 7, loss = 0.30001775
Validation score: 0.901171
Iteration 8, loss = 0.29867525
Validation score: 0.900138
Iteration 9, loss = 0.29774148
Validation score: 0.901860
Iteration 10, loss = 0.29688603
Validation score: 0.900482
Iteration 11, loss = 0.29538448
Validation score: 0.901860
Iteration 12, loss = 0.29420375
```

```

Validation score: 0.900482
Iteration 13, loss = 0.29405117
Validation score: 0.901171
Iteration 14, loss = 0.29285984
Validation score: 0.900482
Iteration 15, loss = 0.29152320
Validation score: 0.899105
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.

```

Out[55]:

```

GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=True, epsilon=1e-08,
                                     hidden_layer_sizes=(100,), learning_rate='constant',
                                     learning_rate_init=0.001, max_iter=200, momentum=0.9,
                                     n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
                                     random_state=101, shuffle=True, solver='adam', tol=0.0001,
                                     validation_fraction=0.1, verbose=True, warm_start=False),
             fit_params=None, iid='warn', n_jobs=-1,
             param_grid=[{'hidden_layer_sizes': [(128,)], 'activation': ['logistic', 'relu', 'identity'], 'solver': ['adam'], 'batch_size': [64], 'shuffle': [True], 'learning_rate_init': [0.0001, 0.001], 'n_iter_no_change': [10], 'warm_start': [True, False]}, {'hidden_layer_sizes': [(128,)], 'learning_rate': ['constant', 'adaptive'], 'learning_rate_init': [0.0001, 0.001], 'n_iter_no_change': [10], 'warm_start': [True, False]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring=None, verbose=0)

```

**a. Did feature selection help here? Any change in the network architecture? What inputs are being used as the network input?**

In [56]:

```
print("Best Parameters of NN: ", cv.best_params_)
print("Best Parameters of RFE NN: ", rfe_cv.best_params_)
print("Best Parameters of modelSelect NN: ", modelSelect_cv.best_params_)
print("\n\n")

print("GridSearch:")
printMLPArchitecture(cv.best_estimator_)
print("\n")
print("RFE:")
printMLPArchitecture(rfe_cv.best_estimator_)
print("\n")
print("modelSelect:")
printMLPArchitecture(modelSelect_cv.best_estimator_)
print("\n")
```

```
Best Parameters of NN: {'activation': 'logistic', 'batch_size': 64,
'hidden_layer_sizes': (128,), 'learning_rate_init': 0.001, 'n_iter_no_change': 20, 'shuffle': True, 'solver': 'adam', 'warm_start': True}
Best Parameters of RFE NN: {'activation': 'logistic', 'batch_size': 64, 'hidden_layer_sizes': (128,), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'n_iter_no_change': 10, 'shuffle': True, 'solver': 'sgd', 'warm_start': True}
Best Parameters of modelSelect NN: {'activation': 'relu', 'batch_size': 64, 'hidden_layer_sizes': (128,), 'learning_rate_init': 0.001, 'n_iter_no_change': 10, 'shuffle': True, 'solver': 'adam', 'warm_start': True}
```

GridSearch:

Number of Layers: 3

The First layer is Input Layer, and the last layer is the output layer

1 Layer with hidden size 198

2 Layer with hidden size 128

3 Layer with hidden size 1

The activation function: logistic

RFE:

Number of Layers: 3

The First layer is Input Layer, and the last layer is the output layer

1 Layer with hidden size 69

2 Layer with hidden size 128

3 Layer with hidden size 1

The activation function: logistic

modelSelect:

Number of Layers: 3

The First layer is Input Layer, and the last layer is the output layer

1 Layer with hidden size 38

2 Layer with hidden size 128

3 Layer with hidden size 1

The activation function: relu

**b. What is classification accuracy on training and test datasets? Is there any improvement in the outcome?**

In [58]:

```
print("GridSearch NN Train accuracy:", cv.score(X_train_log, y_train_log))
print("GridSearch NN Test accuracy:", cv.score(X_test_log, y_test_log))
print("RFE NN Train accuracy:", rfe_cv.score(X_train_rfe, y_train_log))
print("RFE NNTest accuracy:", rfe_cv.score(X_test_rfe, y_test_log))
print("modelSelect NN Train accuracy:", modelSelect_cv.score(X_train_sel_model,
y_train_log))
print("modelSelect NN Test accuracmodelSelect_cv:", modelSelect_cv.score(X_test
_sel_model, y_test_log))
```

GridSearch NN Train accuracy: 0.9049702063169497  
 GridSearch NN Test accuracy: 0.8975327493369766  
 RFE NN Train accuracy: 0.8964282023903833  
 RFE NNTest accuracy: 0.8987382464036004  
 modelSelect NN Train accuracy: 0.8982192677298246  
 modelSelect NN Test accuracmodelSelect\_cv: 0.8979345816925178

### c. How many iterations are now needed to train this network?

In [59]:

```
print("Number of iterations GS ran: ",cv.best_estimator_.n_iter_)
print("Number of iterations rfe ran: ",rfe_cv.best_estimator_.n_iter_)
print("Number of iterations modelSelect ran: ",modelSelect_cv.best_estimator_.n_
iter_)
```

Number of iterations GS ran: 50  
 Number of iterations rfe ran: 18  
 Number of iterations modelSelect ran: 15

### d. Do you see any sign of over-fitting?

In [ ]:

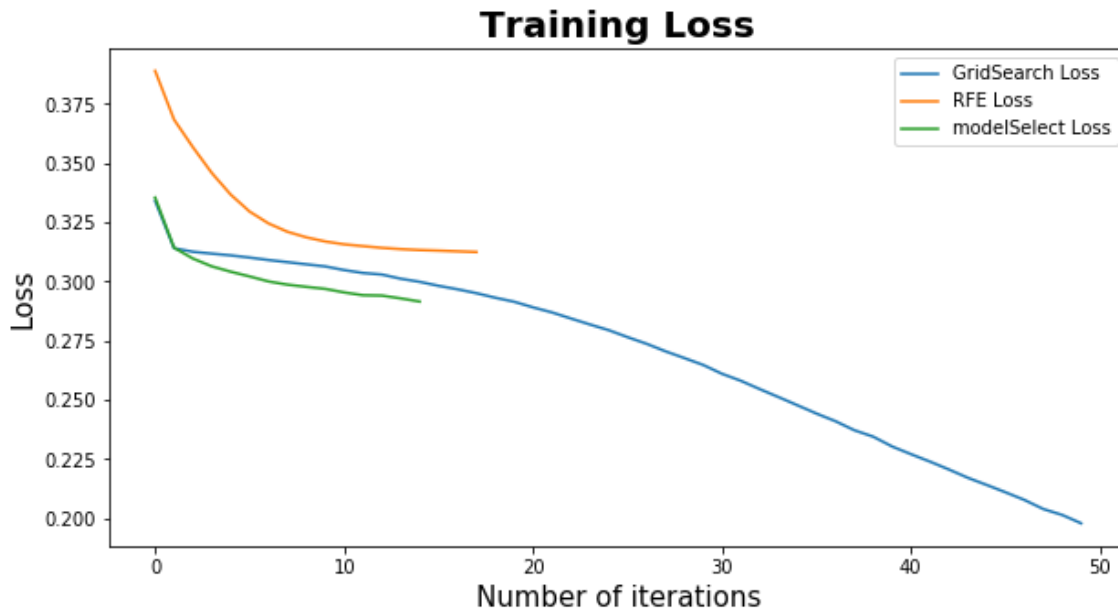
### e. Did the training process converge and resulted in the best model?

In [60]:

```
fig = plt.figure(figsize=(10, 5))
plt.ylabel('Loss',fontsize=15)
plt.xlabel('Number of iterations',fontsize=15)
plt.title('Training Loss',fontsize=20,fontweight = "bold")
plt.plot(cv.best_estimator_.loss_curve_, label="GridSearch Loss")
plt.plot(rfe_cv.best_estimator_.loss_curve_, label="RFE Loss")
plt.plot(modelSelect_cv.best_estimator_.loss_curve_, label="modelSelect Loss")
plt.legend(loc='upper right')
```

Out[60]:

<matplotlib.legend.Legend at 0x7f9c92c77b38>



**4. Using the comparison methods, which of the models (i.e one with selected variables and another with all variables) appears to be better? From the better model, can you identify cars those could potential be “kicks”? Can you provide some descriptive summary of those cars? Is it easy to comprehend the performance of the best neural network model for decision making?**

In [61]:

```
print("GridSearch Classification Report: ")
y_pred = cv.predict(X_test_log)
print(classification_report(y_test_log, y_pred))
print("\n\nRFE Classification Report: ")
y_pred = rfe_cv.predict(X_test_rfe)
print(classification_report(y_test_log, y_pred))
print("\n\nmodelSelect Classification Report: ")
y_pred = modelSelect_cv.predict(X_test_sel_model)
print(classification_report(y_test_log, y_pred))
```

GridSearch Classification Report:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.83	0.26	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.86	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

RFE Classification Report:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.85	0.26	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.88	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

modelSelect Classification Report:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.86	0.25	0.39	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.88	0.62	0.67	12443
weighted avg	0.89	0.90	0.87	12443

## Task 5. Generating an Ensemble Model and Comparing Models

**1. Generate an ensemble model to include the best regression model, best decision tree model, and best neural network model.**

In [62]:

```
voting = VotingClassifier(estimators=[('dt', dt_model), ('lr', log_reg_model), ('nn', nn_model)], voting='soft')
voting.fit(X_train_log, y_train_log)

y_pred_dt = dt_model.predict(X_test_log)
y_pred_log_reg = log_reg_model.predict(X_test_log)
y_pred_nn = nn_model.predict(X_test_log)
y_pred_ensemble = voting.predict(X_test_log)
```



Iteration 1, loss = 0.33406174  
Validation score: 0.899449  
Iteration 2, loss = 0.31403847  
Validation score: 0.899449  
Iteration 3, loss = 0.31255767  
Validation score: 0.899449  
Iteration 4, loss = 0.31177683  
Validation score: 0.899449  
Iteration 5, loss = 0.31103730  
Validation score: 0.899449  
Iteration 6, loss = 0.31010433  
Validation score: 0.899449  
Iteration 7, loss = 0.30906946  
Validation score: 0.899449  
Iteration 8, loss = 0.30817736  
Validation score: 0.900138  
Iteration 9, loss = 0.30726278  
Validation score: 0.899449  
Iteration 10, loss = 0.30637227  
Validation score: 0.899449  
Iteration 11, loss = 0.30483040  
Validation score: 0.900138  
Iteration 12, loss = 0.30360087  
Validation score: 0.900482  
Iteration 13, loss = 0.30294773  
Validation score: 0.900138  
Iteration 14, loss = 0.30116919  
Validation score: 0.899793  
Iteration 15, loss = 0.29985837  
Validation score: 0.899105  
Iteration 16, loss = 0.29817458  
Validation score: 0.899793  
Iteration 17, loss = 0.29671322  
Validation score: 0.898416  
Iteration 18, loss = 0.29506197  
Validation score: 0.900138  
Iteration 19, loss = 0.29314780  
Validation score: 0.900482  
Iteration 20, loss = 0.29144124  
Validation score: 0.898416  
Iteration 21, loss = 0.28907336  
Validation score: 0.899793  
Iteration 22, loss = 0.28694002  
Validation score: 0.898416  
Iteration 23, loss = 0.28437917  
Validation score: 0.899449  
Iteration 24, loss = 0.28186406  
Validation score: 0.899793  
Iteration 25, loss = 0.27945294  
Validation score: 0.900138  
Iteration 26, loss = 0.27649281  
Validation score: 0.899793  
Iteration 27, loss = 0.27371054  
Validation score: 0.900138  
Iteration 28, loss = 0.27053313  
Validation score: 0.899793  
Iteration 29, loss = 0.26767378  
Validation score: 0.900826  
Iteration 30, loss = 0.26475819  
Validation score: 0.899793  
Iteration 31, loss = 0.26101040

Validation score: 0.900482  
Iteration 32, loss = 0.25803914  
Validation score: 0.900482  
Iteration 33, loss = 0.25452168  
Validation score: 0.900138  
Iteration 34, loss = 0.25113722  
Validation score: 0.898416  
Iteration 35, loss = 0.24768587  
Validation score: 0.897727  
Iteration 36, loss = 0.24420492  
Validation score: 0.897727  
Iteration 37, loss = 0.24099523  
Validation score: 0.899449  
Iteration 38, loss = 0.23717577  
Validation score: 0.897727  
Iteration 39, loss = 0.23445705  
Validation score: 0.895661  
Iteration 40, loss = 0.23030711  
Validation score: 0.896350  
Iteration 41, loss = 0.22705398  
Validation score: 0.897383  
Iteration 42, loss = 0.22393178  
Validation score: 0.895317  
Iteration 43, loss = 0.22061048  
Validation score: 0.899793  
Iteration 44, loss = 0.21711178  
Validation score: 0.893939  
Iteration 45, loss = 0.21407507  
Validation score: 0.892562  
Iteration 46, loss = 0.21096409  
Validation score: 0.896694  
Iteration 47, loss = 0.20773348  
Validation score: 0.895317  
Iteration 48, loss = 0.20389114  
Validation score: 0.896350  
Iteration 49, loss = 0.20139598  
Validation score: 0.892906  
Iteration 50, loss = 0.19787601  
Validation score: 0.891529  
Validation score did not improve more than tol=0.000100 for 20 consecutive epochs. Stopping.

**a. Does the Ensemble model outperform the underlying models? Resonate your answer.**

In [63]:

```
print("Report for DT: \n",classification_report(y_test_log, y_pred_dt))
print("\nReport for Logistic Regression: \n",classification_report(y_test_log, y
_pred_log_reg))
print("\nReport for NN: \n",classification_report(y_test_log, y_pred_nn))
print("\nReport for Ensemble: \n",classification_report(y_test_log, y_pred_ensem
ble))
```

Report for DT:

	precision	recall	f1-score	support
0	0.88	1.00	0.93	10832
1	0.80	0.05	0.09	1611
micro avg	0.88	0.88	0.88	12443
macro avg	0.84	0.52	0.51	12443
weighted avg	0.87	0.88	0.82	12443

Report for Logistic Regression:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.84	0.27	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.87	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

Report for NN:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.83	0.26	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.86	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

Report for Ensemble:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.75	0.27	0.40	1611
micro avg	0.89	0.89	0.89	12443
macro avg	0.82	0.63	0.67	12443
weighted avg	0.88	0.89	0.87	12443

**2. Use the comparison methods (or the comparison node) to compare the best decision tree model, the best regression model, the best neural network model and the ensemble model.**

**a. Discuss the findings led by (a) ROC Chart (and Index); (b) Score Ranking (or Accuracy Score); (c) Fit Statistics; (or Classification report) and (4) Output.**

(a) ROC Chart (and Index)

In [64]:

#### ROC

```

y_pred_proba_dt = dt_model.predict_proba(X_test)
y_pred_proba_log_reg = log_reg_model.predict_proba(X_test)
y_pred_proba_nn = nn_model.predict_proba(X_test)
y_pred_proba_ensemble = voting.predict_proba(X_test_log)

roc_index_dt = roc_auc_score(y_test, y_pred_proba_dt[:, 1])
roc_index_log_reg = roc_auc_score(y_test, y_pred_proba_log_reg[:, 1])
roc_index_nn = roc_auc_score(y_test, y_pred_proba_nn[:, 1])
roc_index_ensemble = roc_auc_score(y_test_log, y_pred_proba_ensemble[:, 1])

print("ROC index on test for DT:", roc_index_dt)
print("ROC index on test for logistic regression:", roc_index_log_reg)
print("ROC index on test for NN:", roc_index_nn)
print("ROC index on voting classifier:", roc_index_ensemble)

fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_pred_proba_dt[:,1])
fpr_log_reg, tpr_log_reg, thresholds_log_reg = roc_curve(y_test, y_pred_proba_log_reg[:,1])
fpr_nn, tpr_nn, thresholds_nn = roc_curve(y_test, y_pred_proba_nn[:,1])
fpr_ensemble, tpr_ensemble, thresholds_ensemble = roc_curve(y_test, y_pred_proba_ensemble[:,1])

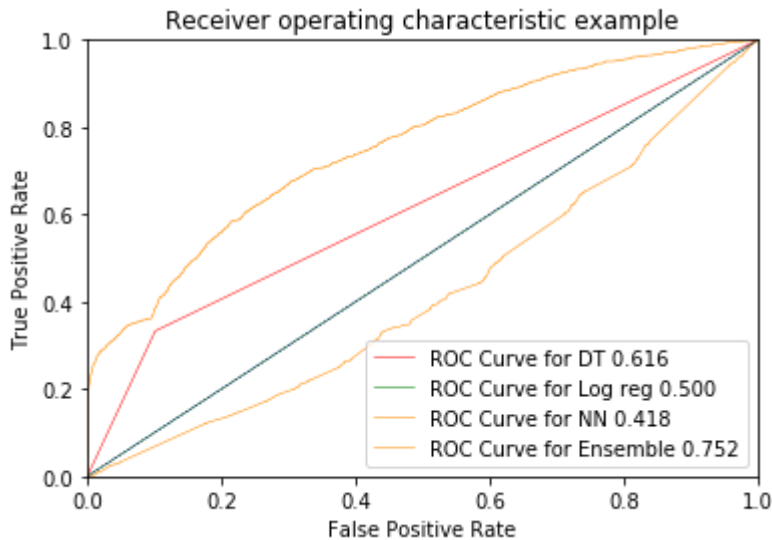
plt.plot(fpr_dt, tpr_dt, label='ROC Curve for DT {:.3f}'.format(roc_index_dt), color='red', lw=0.5)
plt.plot(fpr_log_reg, tpr_log_reg, label='ROC Curve for Log reg {:.3f}'.format(roc_index_log_reg), color='green', lw=0.5)
plt.plot(fpr_nn, tpr_nn, label='ROC Curve for NN {:.3f}'.format(roc_index_nn), color='darkorange', lw=0.5)
plt.plot(fpr_ensemble, tpr_ensemble, label='ROC Curve for Ensemble {:.3f}'.format(roc_index_ensemble), color='darkorange', lw=0.5)

plt.plot([0, 1], [0, 1], color='navy', lw=0.5, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()

```

ROC index on test for DT: 0.6157864609264042  
 ROC index on test for logistic regression: 0.49956381968684643  
 ROC index on test for NN: 0.4175323225571611  
 ROC index on voting classifier: 0.7522671175916681

/home/chihcheng/python3env/lib/python3.6/site-packages/sklearn/linear\_model/base.py:297: RuntimeWarning: overflow encountered in exp  
 np.exp(prob, prob)



(b) Score Ranking (or Accuracy Score)

In [65]:

```
print("Accuracy score on test for DT:", accuracy_score(y_test_log, y_pred_dt))
print("Accuracy score on test for Logistic Regression:", accuracy_score(y_test_log, y_pred_log_reg))
print("Accuracy score on test for NN:", accuracy_score(y_test_log, y_pred_nn))
print("Accuracy score on test for Ensemble:", accuracy_score(y_test_log, y_pred_ensemble))
```

Accuracy score on test for DT: 0.8751105038977739  
 Accuracy score on test for Logistic Regression: 0.8984167805191674  
 Accuracy score on test for NN: 0.8975327493369766  
 Accuracy score on test for Ensemble: 0.8936751587237804

(c) Classification report

In [66]:

```
print("Report for DT: \n",classification_report(y_test_log, y_pred_dt))
print("\nReport for Logistic Regression: \n",classification_report(y_test_log, y
_pred_log_reg))
print("\nReport for NN: \n",classification_report(y_test_log, y_pred_nn))
print("\nReport for Ensemble: \n",classification_report(y_test_log, y_pred_ensem
ble))
```

Report for DT:

	precision	recall	f1-score	support
0	0.88	1.00	0.93	10832
1	0.80	0.05	0.09	1611
micro avg	0.88	0.88	0.88	12443
macro avg	0.84	0.52	0.51	12443
weighted avg	0.87	0.88	0.82	12443

Report for Logistic Regression:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.84	0.27	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.87	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

Report for NN:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.83	0.26	0.40	1611
micro avg	0.90	0.90	0.90	12443
macro avg	0.86	0.63	0.67	12443
weighted avg	0.89	0.90	0.87	12443

Report for Ensemble:

	precision	recall	f1-score	support
0	0.90	0.99	0.94	10832
1	0.75	0.27	0.40	1611
micro avg	0.89	0.89	0.89	12443
macro avg	0.82	0.63	0.67	12443
weighted avg	0.88	0.89	0.87	12443

(d) Output

In [ ]:

**b. Do all the models agree on the cars characteristics? How do they vary?**

In [ ]:

## **Task 6. Final Remarks: Decision Making**

**1. Finally, based on all models and analysis, is there**

**2. Can you summarise positives and negatives of each predictive modelling method based on this analysis?**

**3. How the outcome of this study can be used by decision makers?**

In [ ]:

In [ ]:

In [ ]: