Task 1. Data Preparation for Modelling

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

Variable	Role	Measurement level	Reason of usage
Auction	Input	Categorical	▪ Transaction standards may differ for each auction company. ▪ There is a possibility that the defect rate is high in certain auctions.
VehYear	Input	Numerical	▪ Vehicle year reflects the car's age and condition ▪ Older vehicles are more likely to have hidden issues leading to a kick.
Transmission	Input	Categorical	▪ Transmission type (automatic, manual) affects demand and price differences, which can correlate with bad buy risks. ▪ Most consumers have a tendency to use an automatic type of car.
VehOdo	Input	Numerical	▪ A vehicle range finder indicates how much the vehicle has been used ▪ The higher the mileage, the greater the risk of mechanical failure.
Nationality	Input	Categorical	▪ Country of manufacture can imply differences in quality, reliability, and market perception, impacting kick probability.
IsBadBuy	Target	Numerical	▪ The outcome variable to be predicted: 1 = bad buy (kick), 0 = not a bad buy
MMRCurrentAuctionAveragePrice	Input	Numerical	▪ Auction market price in average condition ▪ Large discrepancies with the actual price may indicate abnormal transactions.
MMRCurrentRetailAveragePrice	Input	Numerical	▪ Retail market price in average condition ▪ If too close to or lower than auction price, it signals potential risk of a kick.

2. Did you have to fix any data quality problems, including data imputation? Detail them.

Auction

```
===== Column: Auction========
Before preprocessing:
Auction
MANHEIM     22168
ADESA       11086
OTHER        8178
NaN            44
Name: count, dtype: int64
==============================
After preprocessing:
Auction
MANHEIM     22168
ADESA       11086
OTHER        8222
Name: count, dtype: int64
```

- NaN (empty value) should be changed to 'OTHER' value.
- Code:

```
df['Auction'] = df['Auction'].fillna('OTHER')
```

Transmission

```
===== Column: Transmission=
Before preprocessing:
Transmission
AUTO       39930
MANUAL      1495
NaN           44
?              6
Manual         1
Name: count, dtype: int64
==========================
After preprocessing:
Transmission
AUTO       39930
MANUAL      1496
UNKNOWN       50
Name: count, dtype: int64
```

- String was changed to upper characters.
- '?' value → NaN → 'UNKNOWN'
- Code:

```
df['Transmission'] = df['Transmission'].str.upper()
df['Transmission'] = df['Transmission'].replace('?',np.nan)
df['Transmission'] = df['Transmission'].fillna('UNKNOWN')
```

Make

```
===== Column: Make=========
Before preprocessing:
Make
CHEVROLET      9548
DODGE          7385
FORD           6458
CHRYSLER       5259
PONTIAC        2355
KIA            1337
SATURN         1245
NISSAN         1186
JEEP            985
HYUNDAI         957
SUZUKI          842
TOYOTA          664
MITSUBISHI      569
MAZDA           532
MERCURY         527
BUICK           413
GMC             351
HONDA           263
OLDSMOBILE      146
ISUZU            82
SCION            77
VOLKSWAGEN       73
LINCOLN          54
NaN              44
INFINITI         27
MINI             19
ACURA            19
CADILLAC         17
SUBARU           17
LEXUS            13
VOLVO            12
Name: count, dtype: int64
```

```
==========================
After preprocessing:
Make
CHEVROLET      9548
DODGE          7385
FORD           6458
CHRYSLER       5259
PONTIAC        2355
KIA            1337
SATURN         1245
NISSAN         1186
JEEP            985
HYUNDAI         957
SUZUKI          842
TOYOTA          664
MITSUBISHI      569
MAZDA           532
MERCURY         527
BUICK           413
GMC             351
HONDA           263
OLDSMOBILE      146
ISUZU            82
SCION            77
VOLKSWAGEN       73
LINCOLN          54
UNKNOWN          44
INFINITI         27
MINI             19
ACURA            19
CADILLAC         17
SUBARU           17
LEXUS            13
VOLVO            12
Name: count, dtype: int64
```

- NaN → 'UNKNOWN'
- Code:

```
df['Make'] = df['Make'].fillna('UNKNOWN')
```

Nationality

```
===== Column: Nationality=
Before preprocessing:
Nationality
AMERICAN          34616
OTHER ASIAN        4474
TOP LINE ASIAN     2110
USA                 125
OTHER               104
NaN                  44
?                     3
Name: count, dtype: int64
==========================
After preprocessing:
Nationality
AMERICAN          34741
OTHER ASIAN        4474
TOP LINE ASIAN     2110
OTHER               104
UNKNOWN              47
Name: count, dtype: int64
```

- 'USA' → 'AMERICAN'
- ? → NaN → 'UNKNOWN'
- Code:

```
df['Nationality'] = df['Nationality'].str.upper().str.strip()
df['Nationality'] = df['Nationality'].replace('USA', 'AMERICAN'
df['Nationality'] = df['Nationality'].replace('?', np.nan)
df['Nationality'] = df['Nationality'].fillna('UNKNOWN')
```

TopThreeAmericanName

```
===== Column: TopThreeAmer
Before preprocessing:
TopThreeAmericanName
GM           14075
CHRYSLER     13627
FORD          7039
OTHER         6688
NaN             44
?                3
Name: count, dtype: int64
==========================
After preprocessing:
TopThreeAmericanName
GM           14075
CHRYSLER     13627
FORD          7039
OTHER         6688
UNKNOWN         47
Name: count, dtype: int64
```

- '?' → NaN → 'UNKNOWN'
- Code:

```
df['TopThreeAmericanName'] = df['TopThreeAmericanName'].replace('?', np.nan)
df['TopThreeAmericanName'] = df['TopThreeAmericanName'].fillna('UNKNOWN')
```

ForSale

```
===== Column: ForSale=====
Before preprocessing:
ForSale
Yes      27402
YES       8544
yes       5524
?            3
No           2
0            1
Name: count, dtype: int64
=========================
After preprocessing:
ForSale
0.0        41470
UNKNOWN        4
1.0            2
Name: count, dtype: int64
```

- String → upper string
- '?', '0' → NaN
- 'YES' → 0, 'NO' → 1
- Code:

```
df['ForSale'] = df['ForSale'].str.upper().str.strip(
df['ForSale'] = df['ForSale'].replace('?', np.nan)
df['ForSale'] = df['ForSale'].replace('0', np.nan)
for_sale_map = {'YES': 0, 'NO': 1}
df['ForSale'] = df['ForSale'].map(for_sale_map)
df['ForSale'] = df['ForSale'].fillna('UNKNOWN')
```

IsBadBuy

```
===== Column: IsBadBuy========
Before preprocessing:
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
==============================
After preprocessing:
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
```

- This variable does not need imputation

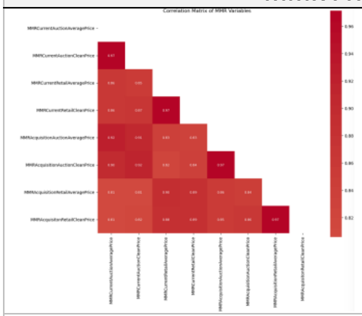
VehYear

```
===== Column: VehYear========
Before preprocessing:
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
=============================
After preprocessing:
count    41476.000000
mean      2005.360232
std          1.729709
min       2001.000000
25%       2004.000000
50%       2005.000000
75%       2007.000000
max       2010.000000
Name: VehYear, dtype: float64
```

- NaN → median value of overall values (year need to change integer)
- Code:

```
df['VehYear'] = df['VehYear'].fillna(df['VehYear'].median())
```

MMR Price



- Strip: remove 'blank space'
- '?', 0 → NaN → median value
- Change to number
- Code:

```
mmr_current_cols = ['MMRCurrentAuctionAveragePrice','MMRCurrentAuctionCleanPrice',
                    'MMRCurrentRetailAveragePrice', 'MMRCurrentRetailCleanPrice']

for col in mmr_current_cols:
    df[col] = df[col].astype(str).str.strip()
    df[col] = df[col].replace('?', np.nan)
    df[col] = pd.to_numeric(df[col], errors='coerce')
    df[col] = df[col].replace(0, np.nan)
    df[col] = df[col].fillna(df[col].median())
```

- According to the heat map, there are nine variable regarding MMRPrice, and these variables high correlation is confirmed in correlation matrix graph. Thus, we chose two variables: MMRCurrentRetailAveragePrice, MMRCurrentAuctionAveragePrice

VehOdo

```
===== Column: VehOdo========
Before preprocessing:
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: VehOdo, dtype: float64
============================
After preprocessing:
count     41476.000000
mean      71293.878918
std       14574.440942
min        4825.000000
25%       61594.750000
50%       73128.500000
75%       82243.750000
max      115717.000000
Name: VehOdo, dtype: float64
```

- If value is lower than lower_bound and is higher than upper_bound, these values replaced median values.
- That is why outliers can affect incorrect analysis

.
- Code:

```
lower_bound = 1000
upper_bound = 400000

vehodo_median = df['VehOdo'].median()

df.loc[(df['VehOdo'] < lower_bound)|(df['VehOdo'] > upper_bound), 'VehOdo'] = np.nan

df['VehOdo'] = df['VehOdo'].fillna(vehodo_median)
```

3. Report the proportion of values of the target variable for the dataset before and after the pre-processing.

Auction

```
=== Auction ===
*** Before preprocessing ***
IsBadBuy  Auction
0         MANHEIM    0.541825
          ADESA      0.258657
          OTHER      0.199518
1         MANHEIM    0.489469
          ADESA      0.327493
          OTHER      0.183038
Name: proportion, dtype: float64
------------------------------
*** After preprocessing ***
IsBadBuy  Auction
0         MANHEIM    0.541255
          ADESA      0.258385
          OTHER      0.200360
1         MANHEIM    0.488922
          ADESA      0.327127
          OTHER      0.183951
Name: proportion, dtype: float64
=================================
```

Make

```
=== Make ===
*** Before preprocessing ***
IsBadBuy  Make
0         CHEVROLET    0.238885
          DODGE        0.182878
          FORD         0.149361
          CHRYSLER     0.126459
          PONTIAC      0.057476
          KIA          0.032329
          SATURN       0.029390
          NISSAN       0.027782
          JEEP         0.022874
          HYUNDAI      0.022763
          SUZUKI       0.019987
          TOYOTA       0.016414
          MITSUBISHI   0.013835
          MAZDA        0.012585
          MERCURY      0.011728
          BUICK        0.009732
          GMC          0.008595
          HONDA        0.006294
          OLDSMOBILE   0.003189
          ISUZU        0.002879
          SCION        0.001941
          VOLKSWAGEN   0.001691
          LINCOLN      0.001026
          INFINITI     0.000471
          CADILLAC     0.000388
          SUBARU       0.000388
          ACURA        0.000333
          VOLVO        0.000333
          MINI         0.000305
          LEXUS        0.000250
1         FORD         0.199627
          CHEVROLET    0.179124
          DODGE        0.152470
          CHRYSLER     0.130103
          PONTIAC      0.052563
          SATURN       0.034483
          NISSAN       0.034296
          KIA          0.031873
          JEEP         0.029823
          HYUNDAI      0.025349
          SUZUKI       0.023113
          MERCURY      0.019385
          MAZDA        0.015098
          TOYOTA       0.013420
          MITSUBISHI   0.013048
          BUICK        0.011556
          GMC          0.007642
          HONDA        0.006710
          OLDSMOBILE   0.005778
          LINCOLN      0.003169
          VOLKSWAGEN   0.002237
          INFINITI     0.001864
          MINI         0.001491
          ACURA        0.001305
          ISUZU        0.001305
          SCION        0.001305
          LEXUS        0.000746
          CADILLAC     0.000559
          SUBARU       0.000559
Name: proportion, dtype: float64
------------------------------
*** After preprocessing ***
IsBadBuy  Make
0         CHEVROLET    0.237834
          DODGE        0.181886
          FORD         0.149204
          CHRYSLER     0.126326
          PONTIAC      0.057416
                          ...
1         SCION        0.001303
          UNKNOWN      0.001117
          LEXUS        0.000745
          CADILLAC     0.000559
          SUBARU       0.000559
Name: proportion, Length: 61, dtype: float64
```

Transmission

```
=== Transmission ===
*** Before preprocessing ***
IsBadBuy  Transmission
0         AUTO            0.963374
          MANUAL          0.036460
          ?               0.000139
          Manual          0.000028
1         AUTO            0.966263
          MANUAL          0.033551
          ?               0.000186
Name: proportion, dtype: float64
------------------------------
*** After preprocessing ***
IsBadBuy  Transmission
0         AUTO            0.962360
          MANUAL          0.036449
          UNKNOWN         0.001191
1         AUTO            0.965183
          MANUAL          0.033513
          UNKNOWN         0.001303
Name: proportion, dtype: float64
------------------------------
```

Nationality

```
=== Nationality ===
*** Before preprocessing ***
IsBadBuy  Nationality
0         AMERICAN          0.837386
          OTHER ASIAN       0.106857
          TOP LINE ASIAN    0.050406
          USA               0.002939
          OTHER             0.002329
          ?                 0.000083
1         AMERICAN          0.822740
          OTHER ASIAN       0.115564
          TOP LINE ASIAN    0.054427
          OTHER             0.003728
          USA               0.003541
Name: proportion, dtype: float64
------------------------------
*** After preprocessing ***
IsBadBuy  Nationality
0         AMERICAN          0.839441
          OTHER ASIAN       0.106744
          TOP LINE ASIAN    0.050353
          OTHER             0.002327
          UNKNOWN           0.001136
1         AMERICAN          0.825358
          OTHER ASIAN       0.115435
          TOP LINE ASIAN    0.054366
          OTHER             0.003724
          UNKNOWN           0.001117
Name: proportion, dtype: float64
======================================
```

TopThreeAmericanName

```
=== TopThreeAmericanName ===
*** Before preprocessing ***
IsBadBuy  TopThreeAmericanName
0         GM                        0.346854
          CHRYSLER                  0.331356
          FORD                      0.162115
          OTHER                     0.159592
          ?                         0.000083
1         CHRYSLER                  0.312395
          GM                        0.291705
          FORD                      0.222181
          OTHER                     0.173719
Name: proportion, dtype: float64
------------------------------
*** After preprocessing ***
IsBadBuy  TopThreeAmericanName
0         GM                        0.346489
          CHRYSLER                  0.331007
          FORD                      0.161944
          OTHER                     0.159424
          UNKNOWN                   0.001136
1         CHRYSLER                  0.312046
          GM                        0.291380
          FORD                      0.221933
          OTHER                     0.173524
          UNKNOWN                   0.001117
Name: proportion, dtype: float64
======================================
```

ForSale

```
=== ForSale ===
*** Before preprocessing ***
IsBadBuy  ForSale
0         Yes        0.666833
          YES        0.202714
          yes        0.130287
          ?          0.000083
          No         0.000055
          0          0.000028
1         Yes        0.619252
          YES        0.228077
          yes        0.152672
Name: proportion, dtype: float64
------------------------------
*** After preprocessing ***
IsBadBuy  ForSale
0         0.0        0.999834
          UNKNOWN    0.000111
          1.0        0.000055
1         0.0        1.000000
Name: proportion, dtype: float64
--------------------------------------
```

VehYear

```
=== VehYear ===
*** Before preprocessing ***
            count         mean       std     min     25%     50%     75%  \
IsBadBuy
0         36067.0  2005.471151  1.697094  2001.0  2004.0  2006.0  2007.0
1          5365.0  2004.617521  1.770015  2001.0  2003.0  2005.0  2006.0

             max
IsBadBuy
0         2010.0
1         2009.0
------------------------------
*** After preprocessing ***
            count         mean       std     min     25%     50%     75%  \
IsBadBuy
0         36105.0  2005.470655  1.696270  2001.0  2004.0  2006.0  2007.0
1          5371.0  2004.617948  1.769072  2001.0  2003.0  2005.0  2006.0

             max
IsBadBuy
0         2010.0
1         2009.0
======================================
```

VehOdo

```
=== VehOdo ===
*** Before preprocessing ***
            count          mean           std     min      25%      50%  \
IsBadBuy
0         36067.0  70848.963263  14737.768855   577.0  61073.0  72635.0
1          5365.0  74332.241193  14267.248465  4825.0  65222.0  76401.0

              75%       max
IsBadBuy
0         81832.5  480444.0
1         84692.0  115717.0
------------------------------
*** After preprocessing ***
            count          mean           std     min      25%      50%  \
IsBadBuy
0         36105.0  70842.090486  14566.926787  8706.0  61090.0  72655.0
1          5371.0  74330.896481  14259.332414  4825.0  65226.5  76390.0

              75%       max
IsBadBuy
0         81820.0  112056.0
1         84687.5  115717.0
======================================
```

MMRCurrentAuctionAveragePrice

```
=== MMRCurrentAuctionAveragePrice ===
*** Before preprocessing ***
          count unique top freq
IsBadBuy
0         36064   8795   0  251
1          5365   3647   0   36
------------------------------
*** After preprocessing ***
            count         mean          std     min     25%     50%     75%
IsBadBuy
0         36105.0  6250.856488  2167.090173  2017.0  4558.0  6138.0  7757.0
1          5371.0  5480.379352  2206.092813  2017.0  3648.5  5232.0  6951.5

              max
IsBadBuy
0         12627.0
1         12617.0
--------------------------------------
```

MMRCurrentRetailAveragePrice

```
=== MMRCurrentRetailAveragePrice ===
*** Before preprocessing ***
          count unique top freq
IsBadBuy
0         36047  10351   0  251
1          5362   3879   0   36
------------------------------
*** After preprocessing ***
            count         mean          std     min     25%     50%      75%
IsBadBuy
0         36105.0  8920.540978  2762.134490  2973.0  6849.0  8760.0  10927.0
1          5371.0  7951.943400  2815.670232  2970.0  5760.0  7838.0   9839.0

              max
IsBadBuy
0         16151.0
1         16098.0
--------------------------------------
```

- After pre-processing, several variables showed skewed distributions in the Kick Car dataset, such as Auction MANHEIM (0.49), Transmission-AUTO(0.96), Nationality – American (0.83).
- The average values of VehYear, VehOdo, MMRCurrentAuctionAveragePrice, and MMRCurrentRetailAveragePrice tend to center around 50%

Task 2. Decision Tree Modelling

1. **Build a decision tree using the default setting. Examine the tree results and answer the following:**
   a. **What parameters have been used to build the tree? Detail them.**

Default tree (DecisionTreeClassifier)

```
****Model parameters*****
 {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'ma
x_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weig
ht_fraction_leaf': 0.0, 'monotonic_cst': None, 'random_state': 10, 'splitter': 'best'}
Number of leaves in the trained model: 4999
```

'criterion': 'gini', min_samples_leaf: 1, min_samples_split: 2, random_state: 10

Small model tree (DecisionTreeClassifier)

```
model_small = DecisionTreeClassifier(
    max_depth=3,
    class_weight='balanced',
    random_state=random_state
)
```

max_depth = 3, class_weight=’balanced’,, random_state=random_state

b. What data split was used to create training and test datasets?

```
from sklearn.model_selection import train_test_split
random_state = 10
test_set_size = 0.3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_set_size, stratify=y, random_state=random_state)

print("Size of training set:", len(X_train))
print("Size of testing set:", len(X_test))

Size of training set: 29033
Size of testing set: 12443
```

- 'random_state (10)' sets a fixed value to ensure that we get the same result regardless of multiple tries. Data for testing is divided by the ratio of 'test_set_size (0.3). Therefore, the size of the training set is 29033, and the testing set is 41476

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

<table>
<tr><td>

```
print("Training set accuracy:", model.score(X_train, y_train))
print("Testing set accuracy:", model.score(X_test, y_test))

Training set accuracy: 0.9998277821788999
Testing set accuracy: 0.7807602668166841
```

</td><td>Training accuracy shows high accuracy (0.99), whereas testing accuracy (0.78) is lower than training accuracy.
This result shows the possibility of overfitting.</td></tr>
<tr><td>

- default tree

```
              precision    recall  f1-score   support

           0       0.88      0.87      0.87     10832
           1       0.18      0.19      0.18      1611

    accuracy                           0.78     12443
   macro avg       0.53      0.53      0.53     12443
weighted avg       0.79      0.78      0.78     12443
```

</td><td>Regarding 'class 1' from this classification_report,
- precision= 0.18, recall=0.19, f1-score=0.18

From this report, the decision tree model identified 19% of kick car (recall), and of those, 18% were correctly (precision). This means this model cannot predict class 1(kick).

Thus, we effort to tune for improvement of model.</td></tr>
<tr><td>

- small model tree

```
              precision    recall  f1-score   support

           0       0.91      0.63      0.74     10832
           1       0.19      0.57      0.28      1611

    accuracy                           0.62     12443
   macro avg       0.55      0.60      0.51     12443
weighted avg       0.82      0.62      0.68     12443
```

</td><td>The default tree has the complexity of model due to unlimited maximum depth, so this model can be enhanced by pruning branches of the decision tree (small model tree).
We set up max_depth = 3.
In the evaluation of model, the result shows that recall = 0.57, and f1-score = 0.28. However, precision remains at a low level.</td></tr>
</table>

d. What is the size of the tree (number of nodes and rules)?

<table>
<tr><th>< default tree></th><th><small model tree></th></tr>
<tr><td>

```
print('Number of leaves in the trained model:', model.get_n_leaves())
print("Number of nodes:", model.tree_.node_count)

Number of leaves in the trained model: 4999
Number of nodes: 9997
```

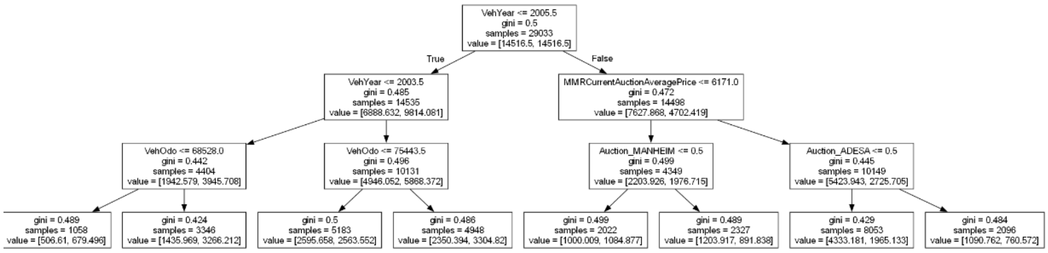
</td><td>

```
# Node, Leaves, depth
print("Nodes:", model_small.tree_.node_count)
print("Leaves:", model_small.get_n_leaves())
print("Depth:", model_small.get_depth())
Nodes: 15
Leaves: 8
Depth: 3
```

</td></tr>
<tr><td colspan="2">Default tree (maximal tree) has 4999 rules and 9997 nodes, whereas small model tree has 8 rules and 15 nodes.</td></tr>
</table>

e. Which variable is used for the first split? What are the variables that are used for the second split?

< small model tree>



- first split: VehYear and MMRCurrentAuctionAveragePrice
- second split: VehOdo and Auction variable

f. What are the five important variables in building the tree?

< default model tree>

```
display_feature_importances(model, feature_names)
```

```
VehOdo : 0.36932140555782883
MMRCurrentRetailAveragePrice : 0.2659883719680745
MMRCurrentAuctionAveragePrice : 0.24943085804225154
VehYear : 0.04296774409041194
Auction_OTHER : 0.015423279634537651
Auction_MANHEIM : 0.0129631752570289
Auction_ADESA : 0.010884338151795542
Nationality_AMERICAN : 0.009273743181650274
Nationality_OTHER ASIAN : 0.009260489473204996
Nationality_TOP LINE ASIAN : 0.006488607453676483
Transmission_AUTO : 0.0036549128945540465
Transmission_MANUAL : 0.0032932079964304224
Nationality_OTHER : 0.001049866298554846
Transmission_UNKNOWN : 0.0
Nationality_UNKNOWN : 0.0
Number of leaves: 4999
```

- VehOdo: 0.36
- MMRCurrentRetailAveragePrice: 0.26
- MMRCurrentAuctionAveragePrice: 0.24
- VehYear: 0.04
- Auction_OTHER: 0.01

< small model tree>

```
display_feature_importances(model_small, feature_names)
visualize_model(model_small)
```

```
VehYear : 0.7661573891371191
MMRCurrentAuctionAveragePrice : 0.11149221549044215
VehOdo : 0.0731100508607268
Auction_ADESA : 0.029442055720801594
Auction_MANHEIM : 0.01979828879091031
Nationality_TOP LINE ASIAN : 0.0
Nationality_UNKNOWN : 0.0
Transmission_MANUAL : 0.0
Transmission_UNKNOWN : 0.0
Nationality_AMERICAN : 0.0
Nationality_OTHER : 0.0
Nationality_OTHER ASIAN : 0.0
Auction_OTHER : 0.0
Transmission_AUTO : 0.0
MMRCurrentRetailAveragePrice : 0.0
Number of leaves: 8
```

- VehYear: 0.76
- MMRCurrentAuctionAveragePrice: 0.11
- VehOdo: 0.073
- Auction_ADESA: 0.029
- Auction_MANHEIM: 0.019

```
def display_feature_importances(model, feature_names, features_to_display=20):
    importances = model.feature_importances_
    indices = np.argsort(importances)
    indices = np.flip(indices, axis=0)

    indices = indices[:features_to_display]
    for i in indices:
        print(feature_names[i], ':', importances[i])
    print("Number of leaves:", model.get_n_leaves())
```

- There are five important variables according to default tree and small tree, this situation can be showed from display_feature_importances' function.

- Auction can be included in five important variables, but low percentage (0.01, 0.019) means that these data are not meaning variables.

g. Report if you see any evidence of model overfitting

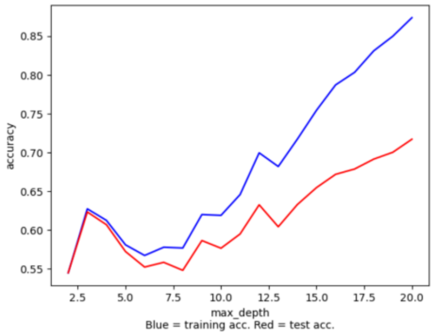
Overfitting means it can fit the data in the training process regardless of the correct prediction result:

- **The number of nodes & complicated and deep model**
  When confirming default maximal tree, there is a complicated and deep decision tree model, which has a lot of nodes.

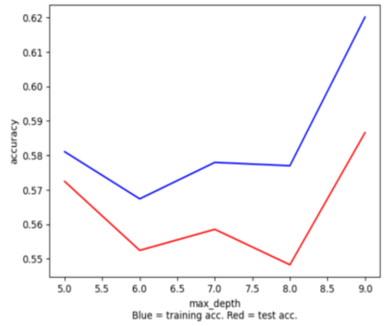
- **Training accuracy (0.99) > testing accuracy (0.78)**
  The maximal tree overfits the training dataset (0.99). That is why testing accuracy is lower than training. Therefore, this model fits in the case of training, not in the case of testing.

2. Build another decision tree tuned with GridSearchCV. Examine the tree results.

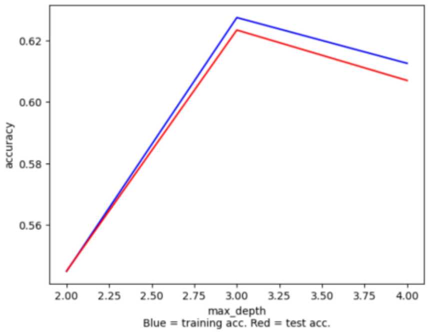
a. What are the optimal parameters for this decision tree? Explain your choice of hyperparameters to search, and the chosen search ranges(s)?



<plot 1>



<plot 2>



<plot 3>

```
params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': range(2, 4),
    'min_samples_leaf': range(5, 50, 10)
}
cv= perform_grid_search(X_train, y_train, X_test, y_test, params)
```

<plot2>

```
params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': range(5,8),
    'min_samples_leaf': range(5, 10)
}
cv = perform_grid_search(X_train, y_train, X_test, y_test, params)
```

<plot3>

The results of 'plot1' and 'plot2' means:
- A larger gap between training and test performance means the indication of overfitting.
- Therefore, we need to choose the search ranges which has the least gap and try to tune by adjusting parameters.

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

```
Fitting 10 folds for each of 20 candidates, totalling 200 fits
Train accuracy: 0.6274584093962043
Test accuracy: 0.6234027163867235
              precision    recall  f1-score   support

           0       0.91      0.63      0.74     10832
           1       0.19      0.57      0.28      1611

    accuracy                           0.62     12443
   macro avg       0.55      0.60      0.51     12443
weighted avg       0.82      0.62      0.68     12443

{'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 5}
```

<1> Overall, training (0.627) and test accuracy (0.623) are similar score. This result may mean overfitting risk is reduced, but it may also indicate underfitting.

<2> Regarding 'class 1':
- precision= 0.19, recall=0.57, f1-score=0.28

From this report, the decision tree model identified 57% of kick car (recall), and of those, 19% were correctly (precision). This means this model cannot predict class 1(kick). F1-score (0.28) is a balance between precision and recall.

Although recall can be improved, precision is still showed to low score, and f1-score also is low score due to this precision score

c. What is the size of the chosen tree (number of nodes and rules)?

```
params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': range(2, 4),
    'min_samples_leaf': range(5, 50, 10)
}
cv= perform_grid_search(X_train, y_train, X_test, y_test, params)
```

```
Fitting 10 folds for each of 20 candidates, totalling 200 fits
Train accuracy: 0.6274584093962043
Test accuracy: 0.6234027163867235
              precision    recall  f1-score   support

           0       0.91      0.63      0.74     10832
           1       0.19      0.57      0.28      1611

    accuracy                           0.62     12443
   macro avg       0.55      0.60      0.51     12443
weighted avg       0.82      0.62      0.68     12443

{'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 5}
```

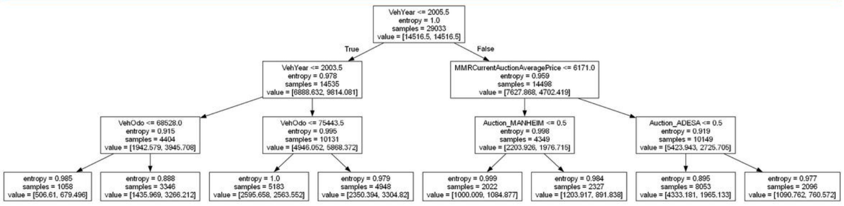
```
Nodes:  15
Leaves:  8
Depth:  3
```

We choose this gridserachCV model with these parameters; this model has 15 nodes and 8 rules, which is similar to a small decision classifier model (small model tree).

However, this model (entropy) has a difference in the criterion from the small model tree (gini).

d. Which variable is used for the first split? What are the variables that are used for the second split?

```
model_cv= cv.best_estimator_
visualize_model(model_cv)
```



First split:
VehYear / MMRCurrentAuction AveragePrice

Second split:
VehOdo / Auction

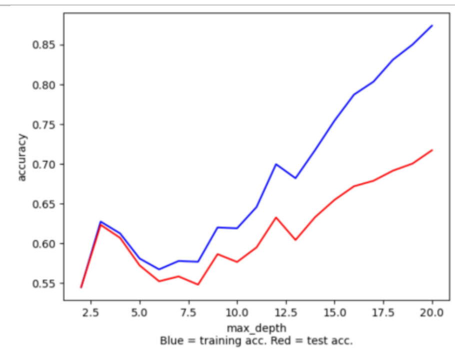
e. What are the five important variables in building the tree?

```
display_feature_importances(model_cv, feature_names)
visualize_model(model_cv)
```

```
VehYear : 0.7598540857360376
MMRCurrentAuctionAveragePrice : 0.1144055334609266
VehOdo : 0.07466742514630023
Auction_ADESA : 0.03162454160340716
Auction_MANHEIM : 0.019448414053328502
Nationality_TOP LINE ASIAN : 0.0
Nationality_UNKNOWN : 0.0
Transmission_MANUAL : 0.0
Transmission_UNKNOWN : 0.0
Nationality_AMERICAN : 0.0
Nationality_OTHER : 0.0
Nationality_OTHER ASIAN : 0.0
Auction_OTHER : 0.0
Transmission_AUTO : 0.0
MMRCurrentRetailAveragePrice : 0.0
Number of leaves: 8
```

- VehYear: 0.75
- MMRCurrent AuctionAveragePrice: 0.11
- VehOdo: 0.07
- Auction_ADESA: 0.03
- Auction_MANHEIM: 0.019

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



This result of the relationship between accuracy and max_depth.

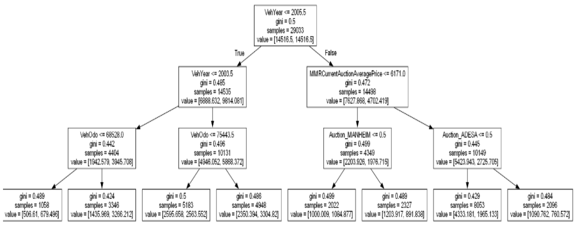
A larger gap between training and test performance means a huge indication of overfitting.

3. **What is the significant difference between these two decision tree models – default (Task 2.1) and using GridSearchCV (Task 2.2)? How do they compare performance-wise? Produce the ROC curve for both DTs. Explain why those changes may have happened**

<default model-DecisionTreeClassifier>

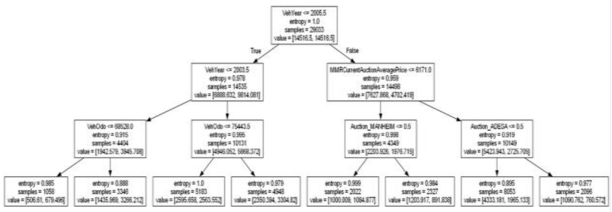
- This model has 9997 nodes.
- It is a complicated and deep model.

<Small model-DecisionTreeClassifier>



- gini = 0.499

<GridSearchCV model>



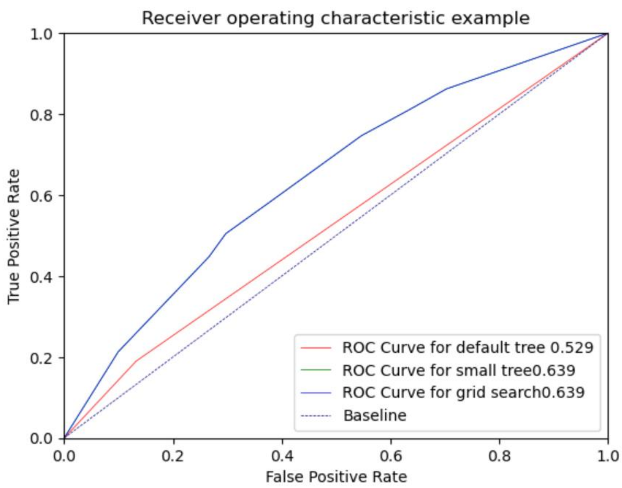
- entropy = 0.985
- Entropy is information gain.
- The higher the entropy, the much information (purity).

<ROC curve>

```
print("ROC index on test for default model:", roc_index_dt)
print("ROC index on test for small model:", roc_index_dt_small)
print("ROC index on test for grid search model:", roc_index_dt_cv)
```

```
ROC index on test for default model: 0.5288166393434355
ROC index on test for small model: 0.6386763716857975
ROC index on test for grid search model: 0.6386763716857975
```

- **Difference**: Small tree is based on ‘gini’, whereas cv model is based on ‘entropy’ (information gain)
- **Comparison of performance:** The ROC curve can help compare models. These models’ index is more than 0.5, and small tree and grid search is the same to 0.639. Therefore, we can choose a model, considering gini and entropy.
- **Why these changes**: The differences between default tree and other models emerge because we set up ‘class_weight=balanced’ for reducing overfitting in small default model. Furthermore, in process of GridSerachCV tuning (cv.best_estimator_), model based on ‘entropy’ is considered better cross-validation quality.



4. **From the better model, can you provide the characteristics of cars most likely to be ‘kicks’? If it is hard to comprehend, discuss why.**

GridSearchCV decision tree (criterion: entropy) ➔ can address important variables.

- Vehicle Year (First split: VehYear <=2005.5): cars that are older than 2005 have tendency to have a high risk of ‘kick(badbuy)’
- Vehicle Odometer (VehOdo <= 68528 and 75443.5): high odometer can be related to kicks
- Auction (= MANHEIM, ADESA): Auction company influences the likelihood of kick.
- MMRCurrentAuctionAveragePrice <= 6171: A low auction price can mean that this car has a high risk regarding kick (bad buy)

It is hard to comprehend because:

- The model is complicated, which has many leaves and rules.
- We cannot confirm the correlation with variables because many variables can simultaneously affect to model's results.
- Data imbalance: The data set is imbalanced with a significant number of negative data and a few positive data extremely.

Task 3. Regression Modelling

1. **Describe what additional processing was required on this dataset to be used in regression modelling. List the variables that need further processing and provide details of the processing.**

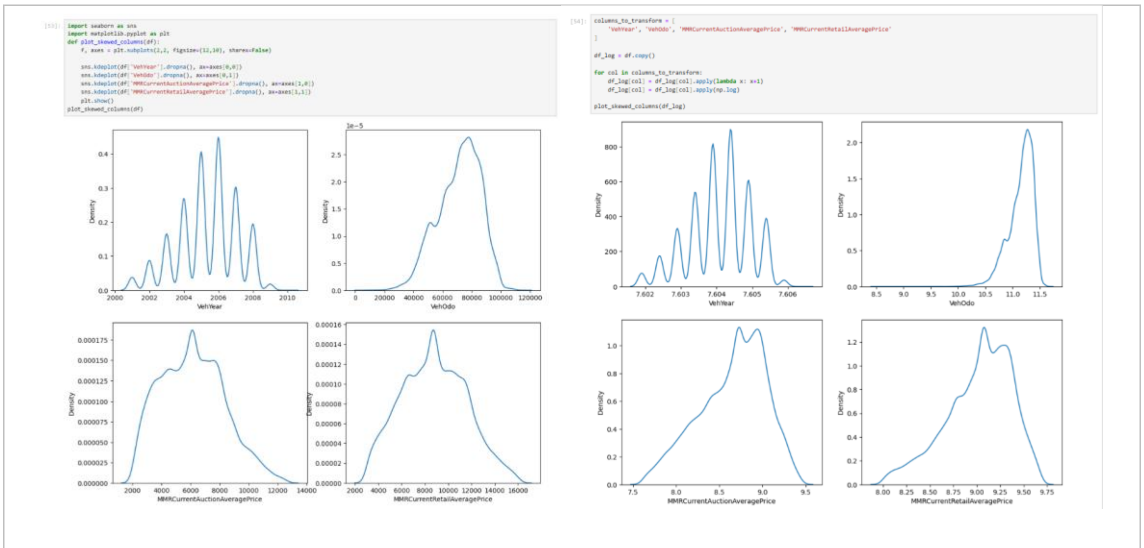
- **Standardisation**

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

print("Before scaling\n-------------")
for i in range(5):
    col = X_train[:,i]
    print("Variable #{}: min {}, max {}, mean {:.2f} and std dev {:.2f}".format(i, min(col), max(col), np.mean(col), np.std(col)))

X_train = scaler.fit_transform(X_train, y_train)

print("After scaling\n-------------")
for i in range(5):
    col = X_train[:,i]
    print("Variable #{}: min {}, max {}, mean {:.2f} and std dev {:.2f}".format(i, min(col), max(col), np.mean(col), np.std(col)))

X_test = scaler.transform(X_test)
```

```
Before scaling
-------------
Variable #0: min 2001.0, max 2010.0, mean 2005.36 and std dev 1.73
Variable #1: min 4825.0, max 115717.0, mean 71311.23 and std dev 14570.61
Variable #2: min 2017.0, max 12623.0, mean 6156.95 and std dev 2188.21
Variable #3: min 2978.0, max 16151.0, mean 8799.11 and std dev 2789.97
Variable #4: min False, max True, mean 0.27 and std dev 0.44
After scaling
-------------
Variable #0: min -2.5182851572014004, max 2.684608421782512, mean -0.00 and std dev 1.00
Variable #1: min -4.563035230327706, max 3.0476255036126663, mean -0.00 and std dev 1.00
Variable #2: min -1.8919368524000326, max 2.954951367499358, mean 0.00 and std dev 1.00
Variable #3: min -2.0864404537378474, max 2.6351132730794915, mean 0.00 and std dev 1.00
Variable #4: min -0.6033342483025691, max 1.6574560499638853, mean 0.00 and std dev 1.00
```

- Numerical variables such as VehYear, VehOdo, MMRCurrentAuctionAveragePrice, and MMRCurrentRetailAveragePrice were standardised using StandardScaler.
- This process ensures mean = 0 and standard deviation = 1 and prevents variables with large-scale differences.

- **Logarithmic transformation**



- Variables such as VehYear, VehOdo, MMRCurrentAuctionAveragePrice, and MMRCurrentRetailAveragePrice show skewed distributions.
- These are normally distributed by applying a logarithmic transformation.

- class_weight='balanced': Most of 'IsBadBuy' variable are 1 (x kick car) in 'kick.vsc' dataset. Therefore, it should be balanced with this processing.

2. Build a regression model tuned with GridSearchCV. Answer the following:

a. Name the Regression function used.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state=random_state)

model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=10)
```

b. What are the optimal parameters for this regression model? Explain your choice of hyperparameters to search, and the chosen search range(s)?

```
from sklearn.model_selection import GridSearchCV
params = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

cv = GridSearchCV(param_grid=params,
                  estimator=LogisticRegression(random_state=random_state, class_weight='balanced'),
                  cv=10, n_jobs=-1)
cv.fit(X_train, y_train)

print("Train accuracy:", cv.score(X_train, y_train))
print("Test accuracy:", cv.score(X_test, y_test))
y_pred = cv.predict(X_test)
print(classification_report(y_test, y_pred))

print(cv.best_params_)
```

```
Train accuracy: 0.6144042985568147
Test accuracy: 0.6149642369203568
              precision    recall  f1-score   support

           0       0.91      0.62      0.74     10832
           1       0.19      0.61      0.29      1611

    accuracy                           0.61     12443
   macro avg       0.55      0.61      0.51     12443
weighted avg       0.82      0.61      0.68     12443

{'C': 1}
```

```
params = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

cv = GridSearchCV(param_grid=params, estimator=LogisticRegression(random_state=random_state, class_weight='balanced'),
                  cv=10, n_jobs=-1)
cv.fit(X_train_log, y_train_log)

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

y_pred = cv.predict(X_test_log)
print(classification_report(y_test_log, y_pred))

print(cv.best_params_)
```

```
Train accuracy: 0.618916405469638
Test accuracy: 0.6204291569557181
              precision    recall  f1-score   support

           0       0.91      0.62      0.74     10832
           1       0.19      0.61      0.29      1611

    accuracy                           0.62     12443
   macro avg       0.55      0.62      0.52     12443
weighted avg       0.82      0.62      0.68     12443

{'C': 0.01}
```

- Smaller C = Stronger regularisation, Larger C = Weaker regularisation → Ideal range = 10^{-6} and 10^{4}
- To be balanced between overfitting and underfitting, we chose search range for Hyperparameters = {'C': {0.001, 0.01, 0.1, 1, 10, 100}
  → Allow the model to identify the optimal complexity level without restricting the search space too narrowly.
- Class_weight='balanced' is applied to mitigate the imbalance issues.
- GridSearchCV_log: {'C': 0.01} has stronger regularisation and not lead to underfitting and to learn in a more stable manner.

c. Report the variables that are included in the regression model.

```
coef = model.coef_[0]

coef = coef[:20]
for i in range(len(coef)):
    print(feature_names[i], ':', coef[i])
```

```
VehYear : -0.3804823517004311
VehOdo : 0.14415402803274585
MMRCurrentAuctionAveragePrice : -0.095570436780107
MMRCurrentRetailAveragePrice : -0.038272248876095455
Auction_ADESA : 0.07183110288718436
Auction_MANHEIM : -0.08946405705092635
Auction_OTHER : 0.03229484736480808
Transmission_AUTO : 0.03133792395509057
Transmission_MANUAL : -0.0318394074556094
Transmission_UNKNOWN : 0.00011396248691719884
Nationality_AMERICAN : -0.03239854940317287
Nationality_OTHER : 0.02095312916143133
Nationality_OTHER ASIAN : 0.041856578538101895
Nationality_TOP LINE ASIAN : -0.009323679205518009
Nationality_UNKNOWN : 0.00012395370954277815
```

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

```
coef = model.coef_[0]

indices = np.argsort(np.absolute(coef))
indices = np.flip(indices, axis=0)

indices = indices[:20]
for i in indices:
    print(feature_names[i], ':', coef[i])
```

```
VehYear : -0.3804823517004311
VehOdo : 0.14415402803274585
MMRCurrentAuctionAveragePrice : -0.095570436780107
Auction_MANHEIM : -0.08946405705092635
Auction_ADESA : 0.07183110288718436
Nationality_OTHER ASIAN : 0.041856578538101895
MMRCurrentRetailAveragePrice : -0.038272248876095455
Nationality_AMERICAN : -0.03239854940317287
Auction_OTHER : 0.03229484736480808
Transmission_MANUAL : -0.0318394074556094
Transmission_AUTO : 0.03133792395509057
Nationality_OTHER : 0.02095312916143133
Nationality_TOP LINE ASIAN : -0.009323679205518009
Nationality_UNKNOWN : 0.00012395370954277815
Transmission_UNKNOWN : 0.00011396248691719884
```

1. VehYear : 0.380
2. VehOdo: 0.144
3. MMRCurrentAuction AveragePrice: 0.096
4. Auction_MANHEIM: 0.0895
5. Auction_ADESA: 0.0718

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

Logistic

Regression_normal

```
Training accuracy: 0.8704921985327042
Test accuracy: 0.8705296150446034
              precision    recall  f1-score   su

           0       0.87      1.00      0.93
           1       0.00      0.00      0.00

    accuracy                           0.87
   macro avg       0.44      0.50      0.47
weighted avg       0.76      0.87      0.81
```

GridSearch_cv

```
Train accuracy: 0.6144042985568147
Test accuracy: 0.6149642369203568
              precision    recall  f1-score   support

           0       0.91      0.62      0.74     10832
           1       0.19      0.61      0.29      1611

    accuracy                           0.61     12443
   macro avg       0.55      0.61      0.51     12443
weighted avg       0.82      0.61      0.68     12443

{'C': 1}
```

GridSearch_cv_log

```
Train accuracy: 0.618916405469638
Test accuracy: 0.6204291569557181
              precision    recall  f1-score   support

           0       0.91      0.62      0.74     10832
           1       0.19      0.61      0.29      1611

    accuracy                           0.62     12443
   macro avg       0.55      0.62      0.52     12443
weighted avg       0.82      0.62      0.68     12443

{'C': 0.01}
```

- Although the normal logistic regression shows a high accuracy (0.870), recall and f1-score are 0.00, which means that it completely failed to learn due to data imbalance.
- After applying GridSearch without log transformation, the accuracy decreased to 0.615, but recall and f1-score increased, which means it started to capture the minority class.
- Additionally, GridSearch with log transformation has slightly higher accuracy (0.620) so that the log transformation stabilised the feature distribution and allowed more effective regularisation.

f. Report any sign of overfitting.

- There is no accurate overfitting sign in the results of GridSearch and GridSearch with log transformation since the differences between train and test accuracy are very small (< 0.01).

3. Build another regression model on the reduced variables set. To minimise variables, either perform dimensionality reduction with Recursive Feature Elimination or select a subset of inputs found significant by the decision tree (use the best decision tree model under Task 2). Tune the model with GridSearchCV to find the best parameter setting. Answer the following:

a. Was dimensionality reduction helpful in identifying a good feature set for building the accurate model? Report the feature selection method used.

- Recursive Feature Elimination with Logistic Regression: Feature set is reduced from 15 to 2.

```
Original feature set 15
Number of features after elimination 2
```

- SelectedFromModel with Decision Tree: Feature set remains 3 from 15.

```
from sklearn.feature_selection import SelectFromModel

selectmodel = SelectFromModel(dt_best, prefit=True)
X_train_sel_model = selectmodel.transform(X_train)
X_test_sel_model = selectmodel.transform(X_test)
print(X_train_sel_model.shape)
```

```
(29033, 3)
```

→ By reducing the feature set, the model is simplified and improves the interpretation possibility.

b. Report the variables that are included in the regression model.

- Recursive Feature Elimination with Logistic Regression: Feature set is reduced from 15 to 2.

```
selected_features = feature_names[rfe.support_]
print("Selected features: ", selected_features)
```

```
Selected features:  Index(['VehYear', 'Auction_MANHEIM'], dtype='object')
```

- SelectedFromModel with Decision Tree: Feature set remains 3 from 15.

```
VehYear : 0.7598540857360376
MMRCurrentAuctionAveragePrice : 0.1144055334609266
VehOdo : 0.07466742514630023
Auction_ADESA : 0.03162454160340716
Auction_MANHEIM : 0.019448414053328502
Nationality_TOP LINE ASIAN : 0.0
Nationality_UNKNOWN : 0.0
Transmission_MANUAL : 0.0
Transmission_UNKNOWN : 0.0
Nationality_AMERICAN : 0.0
Nationality_OTHER : 0.0
Nationality_OTHER ASIAN : 0.0
Auction_OTHER : 0.0
Transmission_AUTO : 0.0
MMRCurrentRetailAveragePrice : 0.0
Number of leaves: 8
```

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

rfe_cv

```
Train accuracy: 0.6316949677952675
Test accuracy: 0.6333681588041469
              precision    recall  f1-score   support

           0       0.91      0.64      0.75     10832
           1       0.19      0.57      0.29      1611

    accuracy                           0.63     12443
   macro avg       0.55      0.61      0.52     12443
weighted avg       0.82      0.63      0.69     12443

{'C': 0.001}
```

rfe_cv_log

```
Original feature set 15
Number of features after elimination 2
Train accuracy: 0.6316949677952675
Test accuracy: 0.6333681588041469
              precision    recall  f1-score   support

           0       0.91      0.64      0.75     10832
           1       0.19      0.57      0.29      1611

    accuracy                           0.63     12443
   macro avg       0.55      0.61      0.52     12443
weighted avg       0.82      0.63      0.69     12443

Best parameters: {'C': 0.001}
```

SelectFromModel_DT

```
Train accuracy: 0.60820445699721
Test accuracy: 0.6144016716225991
              precision    recall  f1-score   support

           0       0.91      0.62      0.74     10832
           1       0.19      0.60      0.29      1611

    accuracy                           0.61     12443
   macro avg       0.55      0.61      0.51     12443
weighted avg       0.82      0.61      0.68     12443

{'C': 0.001}
```

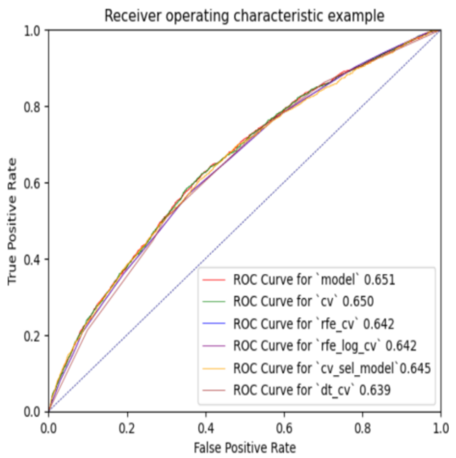
- There are no differences of train (0.632) and test (0.633) accuracy between rfe_cv and rfe_cv_log.
- Applying DT model leads to lower accuracy than RFECV.

→ RFE models have better results than dt model since RFE chose two variables it decreases noise by removing unnecessary features.

d. Report any sign of overfitting.

- The accuracy of the three results is low (around 61-63%), and both train and test accuracy values are similar, which means there is no overfitting

4. Produce the ROC curve for both regression models. Using the best regression model, can you provide the characteristics of cars most likely to be 'kicks'? If it is hard to comprehend, discuss why.



- The best curve → ROC Curve for 'cv_sel_model' ACU = 0.645
- Older vehicles (lower VehYear)
- Higher odometer readings (VehOdo)
- Lower MMR auction average price -> 'kick'
- Variable correlations and class imbalance lead to difficulty in the interpretation of kick cars.

Task 4. Predictive Modelling Using Neural Networks

1. Describe what additional processing was required on this dataset to be used for neural network modelling.

```
from sklearn.preprocessing import StandardScaler
random_seed = 10

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train, y_train)
X_test = scaler.transform(X_test)
```

- Convert categorical data through dummies
- Normalise the numeric data to a 0-1 range by using 'scaler'

2. Build a Neural Network model tuned with GridSearchCV. Answer the following:

a. Explain the parameters in building this model, e.g., network architecture, iterations, activation function, etc. Explain your choice of hyperparameters to search, and the chosen search range(s)?

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report

model = MLPClassifier(random_state=random_state)
model.fit(X_train, y_train)
```

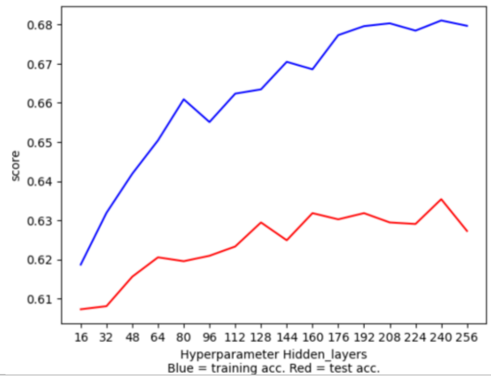
```
MLPClassifier
MLPClassifier(random_state=10)
```

- Import MLP Classifier(Default neural network in sklearn)
- Solver hyperparameter is set by default to 'adam' and also activation is set by 'relu' automatically.

```
model = MLPClassifier(random_state=random_state)
model.fit(X_train_res, y_train_res)
C:\Users\chj91\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
MLPClassifier
MLPClassifier(random_state=10)

model = MLPClassifier(max_iter=500, random_state=random_state)
model.fit(X_train_res, y_train_res)
MLPClassifier
MLPClassifier(max_iter=500, random_state=10)
```

- Due to warning message "convergence is not reached", changed 'max_iter' hyperparameter default 200 to 500.

```
Total number of models:  16
```



- We attempted to tune the model with the lower number of neurons in the hidden layer (less complex model).
- According to the cross-validation results, we set hidden layer size maximum 64. Beyond this point, the gap between test and train increased noticeably, indicating a higher risk of overfitting.

```
params = {'hidden_layer_sizes': [(12,), (16,), (24,), (32,), (40,), (48,), (56,), (64,)],
          'alpha': [0.01, 0.001, 0.0001, 0.00001]}
cv_3 = GridSearchCV(param_grid=params,
                    estimator=MLPClassifier(random_state=random_state), cv=10, n_jobs=-1)
cv_3.fit(X_train_res, y_train_res)
```

- With using ‘alpha’ hyperparameter, reduce overfitting by constraining the size of weights.

b. What is the classification accuracy of the training and test datasets?

```
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))
```

```
Train accuracy: 0.8706644163538043
Test accuracy: 0.8705296150446034
              precision    recall  f1-score   support

           0       0.87      1.00      0.93     10832
           1       0.00      0.00      0.00      1611

    accuracy                           0.87     12443
   macro avg       0.44      0.50      0.47     12443
weighted avg       0.76      0.87      0.81     12443

C:\Users\chj91\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning: Precision is il
l-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\chj91\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning: Precision is il
l-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\chj91\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning: Precision is il
l-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
from imblearn.over_sampling import SMOTE
smote = SMOTE (random_state = 10)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
# For fixing low precision, recall, f1-score
X_train_res = scaler.fit_transform(X_train_res) # scaling X_train_res
```

```
Train accuracy: 0.6623273849562774
Test accuracy: 0.5528409547536768
              precision    recall  f1-score   support

           0       0.91      0.54      0.68     10832
           1       0.17      0.63      0.27      1611

    accuracy                           0.55     12443
   macro avg       0.54      0.58      0.47     12443
weighted avg       0.81      0.55      0.63     12443
```

- **Warning message**
  : Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use zero_division parameter to control this behavior. _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

- This dataset has a very low ratio of IsBadBuy=1 (defective purchase), causing a problem that the model can hardly predict minority classes with basic learning.
- Although Accuracy itself is high, the imbalance between data is quite severe, and to solve this problem, **SMOTE (Synthetic Minority Oversampling Technique)** was applied in the learning stage.
- SMOTE balances between classes while preserving data distribution by generating new synthetic samples among existing samples, rather than simply replicating minority class data.
- Through this process, the model was able to learn both classes evenly, and in particular, the Recall and F1-score for detecting IsBadBuy=1 were improved.

```
print("Train accuracy:", cv_1.score(X_train_res, y_train_res))
print("Test accuracy:", cv_1.score(X_test, y_test))

y_pred = cv_1.predict(X_test)
print(classification_report(y_test, y_pred))
print(cv_1.best_params_)
```

```
Train accuracy: 0.6791239662881335
Test accuracy: 0.5979265450454071
              precision    recall  f1-score   support

           0       0.90      0.60      0.72     10832
           1       0.17      0.55      0.26      1611

    accuracy                           0.60     12443
   macro avg       0.54      0.58      0.49     12443
weighted avg       0.81      0.60      0.66     12443

{'hidden_layer_sizes': (240,)}
```

CV1

- Train accuracy ≈0.68, Test accuracy ≈0.60
- The output of this GridSearchCV returns 240 neurons as the optional number of neurons in the hidden layer.
- But too many hidden nodes can lead overfitting

```
# new parameters
params = {'hidden_layer_sizes': [(12,), (16,), (24,), (32,), (40,), (48,), (56,), (64,)]}
cv_2 = GridSearchCV(param_grid=params,
                    estimator=MLPClassifier(random_state=random_state), cv=10, n_jobs=-1)
cv_2.fit(X_train_res, y_train_res)

print("Train accuracy:", cv_2.score(X_train_res, y_train_res))
print("Test accuracy:", cv_2.score(X_test, y_test))

y_pred = cv_2.predict(X_test)
print(classification_report(y_test, y_pred))
print(cv_2.best_params_)
```

```
Train accuracy: 0.6450362046452736
Test accuracy: 0.5968014144498915
              precision    recall  f1-score   support

           0       0.91      0.60      0.72     10832
           1       0.18      0.58      0.27      1611

    accuracy                           0.60     12443
   macro avg       0.54      0.59      0.50     12443
weighted avg       0.81      0.60      0.66     12443

{'hidden_layer_sizes': (64,)}
```

CV2

- Train accuracy ≈ 0.65, Test accuracy≈0.60
- In previous models, there were cases where the less complex model produced better results. Therefore, as a result of trying a smaller number of hidden layers even in NN, the Accuracy difference decreased, the recall value increased, and a lower number of hidden layers was returned.
- The output of this GridSearchCV returns 64 neurons as the optional number of neurons in the hidden layer.

```
params = {'hidden_layer_sizes': [(12,), (16,), (24,), (32,), (40,), (48,), (56,), (64,)],
          'alpha': [0.01, 0.001, 0.0001, 0.00001]}
cv_3 = GridSearchCV(param_grid=params,
                    estimator=MLPClassifier(random_state=random_state), cv=10, n_jobs=-1)
cv_3.fit(X_train_res, y_train_res)

print("Train accuracy:", cv_3.score(X_train_res, y_train_res))
print("Test accuracy:", cv_3.score(X_test, y_test))

y_pred = cv_3.predict(X_test)
print(classification_report(y_test, y_pred))
print(cv_3.best_params_)
```

```
Train accuracy: 0.6450362046452736
Test accuracy: 0.5968014144498915
              precision    recall  f1-score   support

           0       0.91      0.60      0.72     10832
           1       0.18      0.58      0.27      1611

    accuracy                           0.60     12443
   macro avg       0.54      0.59      0.50     12443
weighted avg       0.81      0.60      0.66     12443

{'alpha': 0.0001, 'hidden_layer_sizes': (64,)}
```

CV3

- Train accuracy ≈ 0.65, Test accuracy≈0.60
- Tuned the second hyperparameter, 'alpha' for strength the L2 regularization term.
- The default value of alpha is 0.0001, thus we tried alpha options around this value.
- There's not a remarkable difference with CV2.

c. Did the training process converge and result in the best model?

- When code is executed with the default MLP Classifier (max_iter=200), a warning message is generated, increased to max_iter=500, and data is collected within 500 times without an error message.

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

```
Train accuracy: 0.6450362046452736
Test accuracy: 0.5968014144498915
              precision    recall  f1-score   support

           0       0.91      0.60      0.72     10832
           1       0.18      0.58      0.27      1611

    accuracy                           0.60     12443
   macro avg       0.54      0.59      0.50     12443
weighted avg       0.81      0.60      0.66     12443

{'alpha': 0.0001, 'hidden_layer_sizes': (64,)}
```

- Although it is difficult to regard it as overfitting with a difference of about 5% in accumulation, it can be interpreted that the accumulation itself is low, and the amount of learning of the model tends to be insufficient.

3. Build another Neural Network model with the reduced feature set. Perform dimensionality reduction by selecting variables with a decision tree (use the best decision tree model under Task 2). Tune the model with GridSearchCV to find the best parameter settings. Answer the following:

a. Did feature selection favour the outcome? Report the changes in the network architecture. What inputs are being used as the network input?

```python
import pickle
from sklearn.feature_selection import SelectFromModel

with open('decision_tree_model.pickle', 'rb') as f:
    dt_best, roc_index_dt_cv, fpr_dt_cv, tpr_dt_cv = pickle.load(f)

selectmodel = SelectFromModel(dt_best, prefit=True)
X_train_sel_model = selectmodel.transform(X_train_res)
X_test_sel_model = selectmodel.transform(X_test)
print(X_train_sel_model.shape)
```

Select only the significant variables selected in the DT model

- VehYear : 0.7661573891371191
- MMRCurrentAuctionAveragePrice : 0.11149221549044215
- VehOdo : 0.0731100508607268
- Auction_ADESA : 0.029442055720801594
- Auction_MANHEIM : 0.01979828879091031

```python
print(X_train_sel_model.shape)
```

```
(50546, 3)
```

- Simplified input data by selecting the feature set.
- Features reduced from 15 to 3

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

```python
params = {'hidden_layer_sizes': [(12,), (16,), (24,), (32,), (40,), (48,), (56,), (64,)],
          'alpha': [0.01,0.001, 0.0001, 0.00001]}
cv_sel_model = GridSearchCV(param_grid=params,
                            estimator=MLPClassifier(random_state=random_state), cv=10, n_jobs=-
cv_sel_model.fit(X_train_sel_model, y_train_res)

print("Train accuracy:", cv_sel_model.score(X_train_sel_model, y_train_res))
print("Test accuracy:", cv_sel_model.score(X_test_sel_model, y_test))

y_pred = cv_sel_model.predict(X_test_sel_model)
print(classification_report(y_test, y_pred))
print(cv_sel_model.best_params_)
```

```
Train accuracy: 0.6176354212004906
Test accuracy: 0.53612472876316
              precision    recall  f1-score   support

           0       0.92      0.51      0.66     10832
           1       0.18      0.71      0.28      1611

    accuracy                           0.54     12443
   macro avg       0.55      0.61      0.47     12443
weighted avg       0.82      0.54      0.61     12443

{'alpha': 1e-05, 'hidden_layer_sizes': (48,)}
```

- Train accuracy ≈ 0.62, Test accuracy≈0.54

- The DT-based reduced feature NN recorded a training accuracy of about 0.61 and a test accuracy of about 0.56.
- Although the overall accuracy is slightly lower than that of the entire feature model, it can be interpreted that the recall of the minority class (IsBadBuy=1) is improved by about 2%, so that a defective purchase vehicle can be detected slightly better

c. **How many iterations are now needed to train this network?**
- Epoch setting remained at max_iter=500, converging stably within 500 times

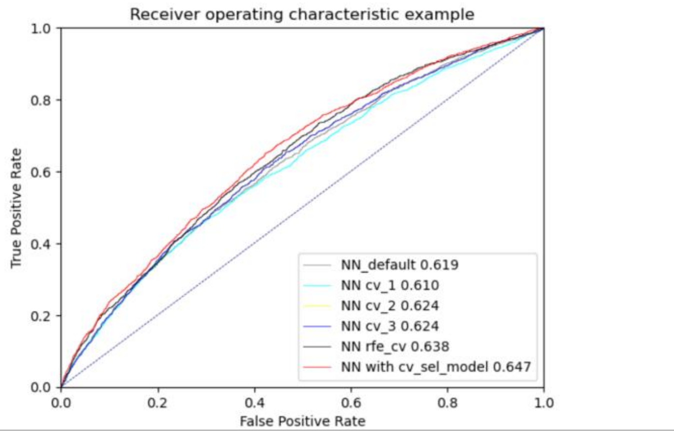
d. **Do you see any sign of over-fitting? Did the training process converge and result in the best model?**

```
Train accuracy: 0.6176354212004906
Test accuracy: 0.53612472876316
              precision    recall  f1-score   support

           0       0.92      0.51      0.66     10832
           1       0.18      0.71      0.28      1611

    accuracy                           0.54     12443
   macro avg       0.55      0.61      0.47     12443
weighted avg       0.82      0.54      0.61     12443

{'alpha': 1e-05, 'hidden_layer_sizes': (48,)}
```

- Although this result has a slightly larger difference in Train/Test acuity than the previous model, it is still not possible to determine the overfitting at 8%.
- In addition, both models converged stably, and although this model is slightly less accurate, our target detection rate of defective purchases itself is a little higher, so this model can be seen as a more optimal model.

4. **Produce the ROC curve for both NNs. Using the best NN model, can you provide the characteristics of cars most likely to be 'kicks'? If it is hard to comprehend, discuss why.**

```
ROC index on test for NN_default: 0.6187085509793728
ROC index on test for NN with gridsearch 1: 0.610088782163248
ROC index on test for NN with gridsearch 2: 0.624033601156011
ROC index on test for NN with gridsearch 3: 0.624033601156011
ROC index on test for NN with feature selection and gridsearch: 0.6377642697408052
ROC index on test for NN with feature selection (model selection) and gridsearch: 0.64719496775
76704
```

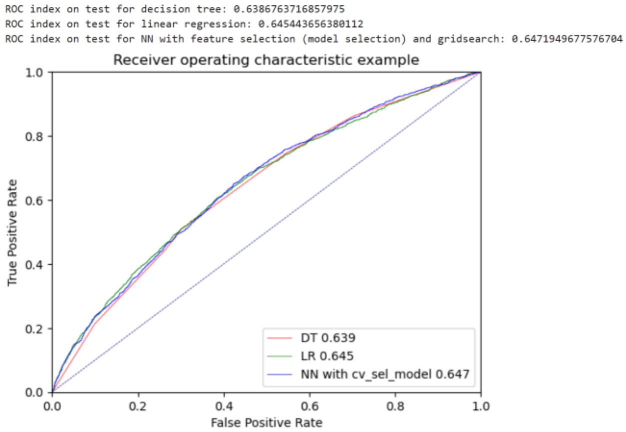


- Based on the ROC curve, 'cv_sel_model (NN model with decision tree selected features)' with index of 0.647, performs better than other models.
- Since the main variables of the Decision Tree model are the same, it can be predicted that the manufacturing year (VehYear), mileage (VehOdo), Auction average price, and the seller are the characteristics of the car that are likely to affect the 'kick'.

Task 5. Final remarks: The decision making

1. **Finally, based on all models and analysis, is there a model you will use in decision-making i.e. the best-performing model? Justify your choice. Draw an ROC chart and an Accuracy Table to support your findings.**

<ROC chart>



<Accuracy Table>

Model	**Training Accuracy**	**Test Accuracy**	**Precision (class 1)**	**Recall (class 1)**	**F1-score (class 1)**	**ROC**
Decision Tree	0.627	0.623	0.19	0.57	0.28	0.639
Linear Regression	0.608	0.614	0.19	0.60	0.29	0.645
Neural Network	0.617	0.536	0.18	0.71	0.28	0.647

Best-performing model: **Neural Network (NN)** with cv_sel_model

- This model has the highest ROC index (0.647), Recall (0.71).
- In the neural network model, the overall accuracy is relatively low, but the recall for IsBadBuy=1 is comparatively high. In other words, it means that the ability to detect kick vehicles is high, and it is a number that matches our goal better. This result will be more helpful for dealers to predict defects.

- According to the prediction accuracy table, this model can predict class 1 (“kick”) better than other models.
- A decision tree has vulnerability related to overfitting and data imbalance. This model cannot be selected for stability because our dataset has these issues.
- Logistic regression cannot capture complex nonlinear relationships, and it can lead to lower accuracy and precision-recall imbalance compared to neural networks.

2. Based on this analysis, can you summarise the positives and negatives of each predictive modelling method?

Model	Pros	Cons
Decision Tree	▪ Can explain reasons about relationship with variables (in the node) ▪ Low necessity of pre-processing (scaling)	▪ Vulnerability of overfitting and data imbalance: our dataset has severe data imbalance, which can lead to overfitting
Logistic Regression	▪ Simple and highly interpretable ▪ Easily identify key variables ▪ Logarithmic transformation: Reduce skewness	▪ Precision-recall imbalance: recall > precision Model Complexity: Linear regression
Neural Network	▪ Neural Network can handle the non-linear data even with missing values.	▪ Takes a long time for training and is sensitive to the number of nodes and other parameters. ▪ Also, cannot interpret exactly decipher the process inside (such as black-box).