University of Minho
School of Engineering

ISLab

Dados e Aprendizagem Automática

Random Forest and XGboost

DAA @ MEI-1º/MiEI-4º – 1º Semestre

Cesar Analide, Bruno Fernandes, Filipa Ferraz, Filipe Gonçalves, Victor Alves

Part IX

Contents

- Implementing Random Forest
    - Hyperparameter tuning
- Implementing XGBoost
    - Hyperparameter tuning

Implementing Random Forest and XGBoost

- Exercise:
  - **Problem:** Development of a ML model able to classify *if the car purchased at the auction is a good or bad buy (kick)*
  - **Classification Approach:** Random Forest and XGBoost
  - **Dataset:** table with information regarding the buy, the auction and the car:
    - *RefID* - Unique (sequential) number assigned to vehicles
    - *IsBadBuy* - Identifies if the kicked vehicle was an avoidable purchase
    - *PurchDate* - The Date the vehicle was Purchased at Auction
    - *Auction* - Auction provider at which the vehicle was purchased
    - *VehYear* - The manufacturer's year of the vehicle
    - *Make* - Vehicle Manufacturer
    - *Model* - Vehicle Model
    - *Color* - Vehicle Color
    - *Transmission* - Vehicles transmission type (Automatic, Manual)
    - *Nationality* - The Manufacturer's country
    - *Size* - The size category of the vehicle (Compact, SUV, etc.)
    - ...

Loading And Preprocessing The Dataset


```
pd.set_option("display.max_columns", 120)
pd.set_option("display.max_rows", 120)
```

```
train=pd.read_csv('training.csv')
test= pd.read_csv('test.csv')
train.head()
```

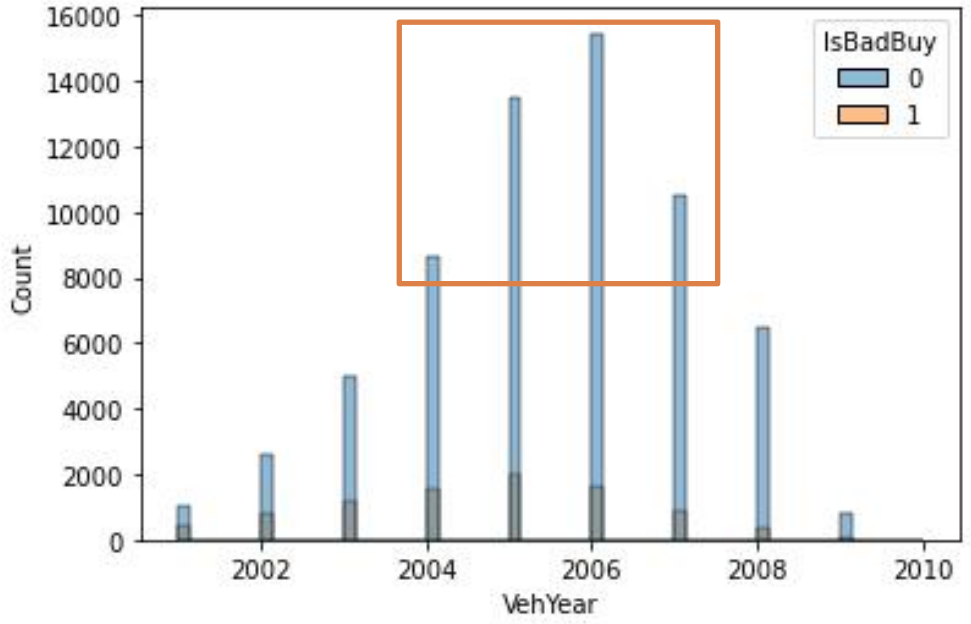
	RefId	IsBadBuy	PurchDate	Auction	VehYear	VehicleAge	Make	Model	Trim
0	1	0	12/7/2009	ADESA	2006	3	MAZDA	MAZDA3	
1	2	0	12/7/2009	ADESA	2004	5	DODGE	1500 RAM PICKUP 2WD	S
2	3	0	12/7/2009	ADESA	2005	4	DODGE	STRATUS V6	SX
3	4	0	12/7/2009	ADESA	2004	5	DODGE	NEON	SX
4	5	0	12/7/2009	ADESA	2005	4	FORD	FOCUS	ZX

...

Exploratory Data Analysis

Understanding the manufacture year of the vehicles

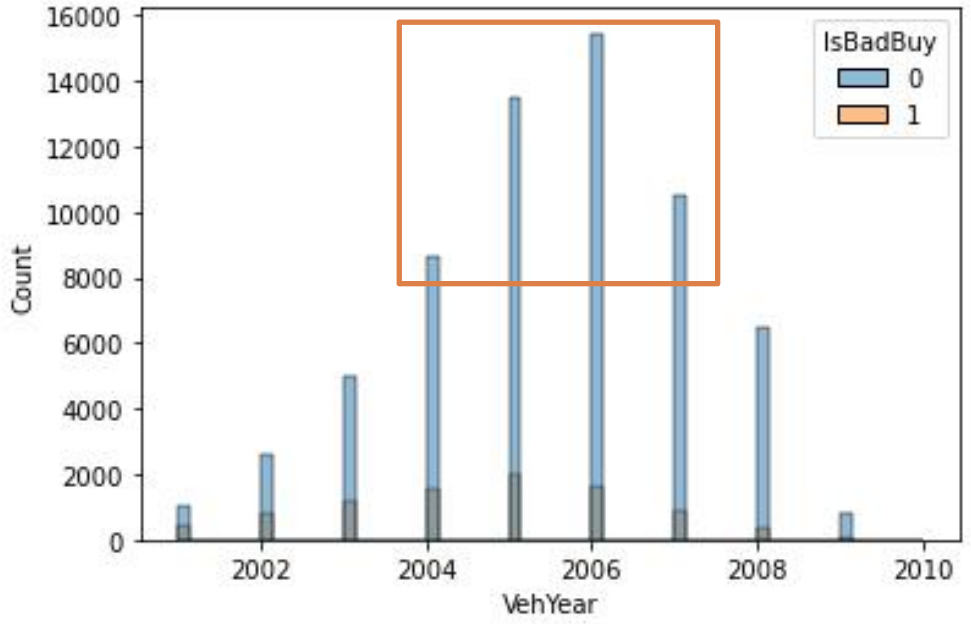
```
sns.histplot(data=train, x="VehYear", hue='IsBadBuy');
```



Exploratory Data Analysis

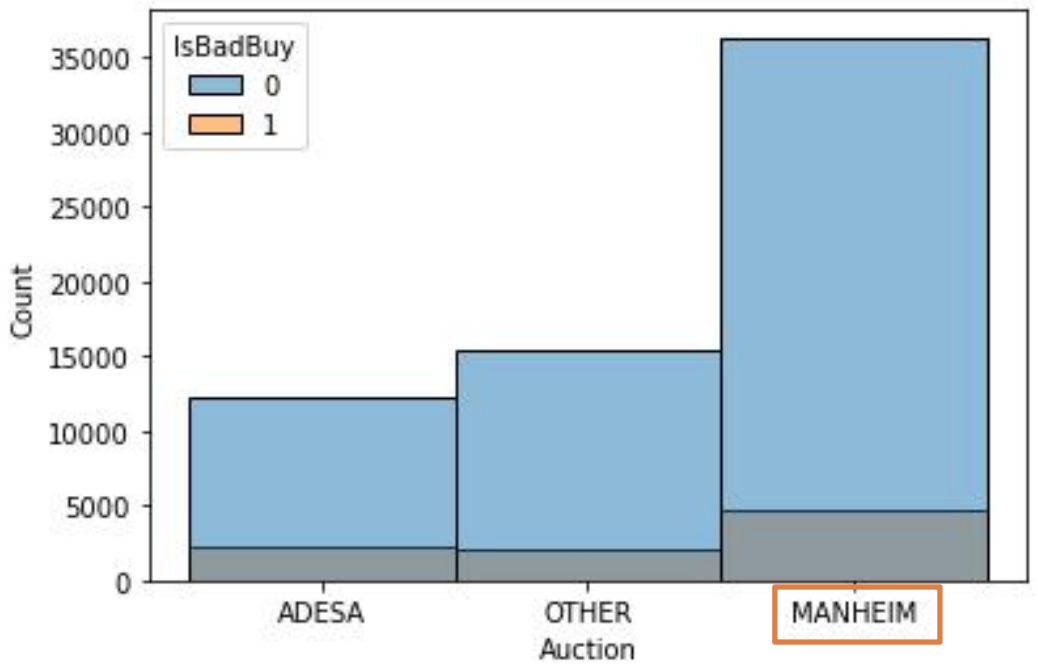
Understanding the manufacture year of the vehicles

```
sns.histplot(data=train, x="VehYear", hue='IsBadBuy');
```



Checking if the auction has any influenceor vehicle being bad

```
sns.histplot(data=train, x="Auction", hue='IsBadBuy');
```



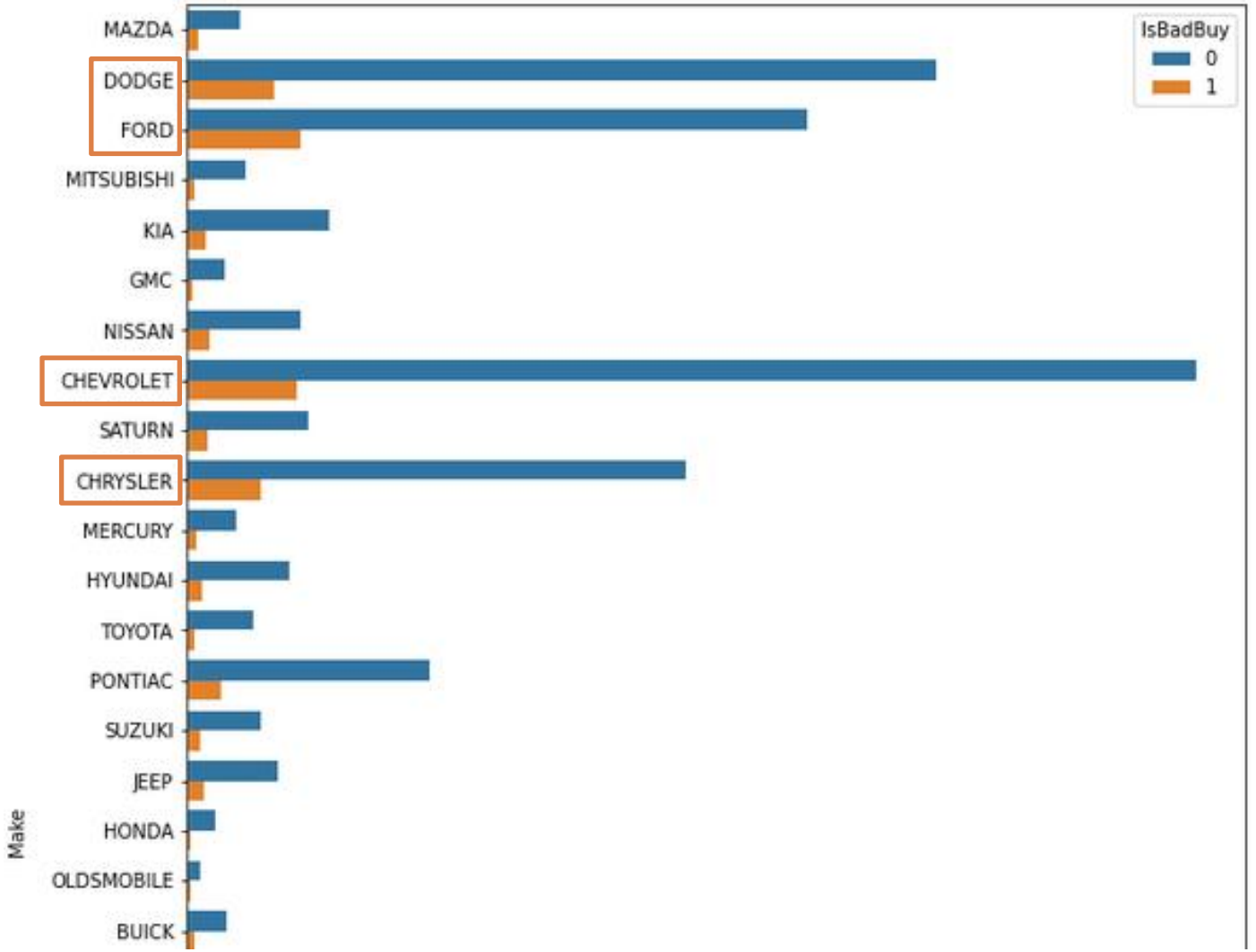
Exploratory Data Analysis

Understanding the manufactures

```
train.Make.value_counts().head(10)
```

```
CHEVROLET    17248
DODGE        12912
FORD         11305
CHRYSLER      8844
PONTIAC       4258
KIA           2484
SATURN        2163
NISSAN        2085
HYUNDAI       1811
JEEP          1644
Name: Make, dtype: int64
```

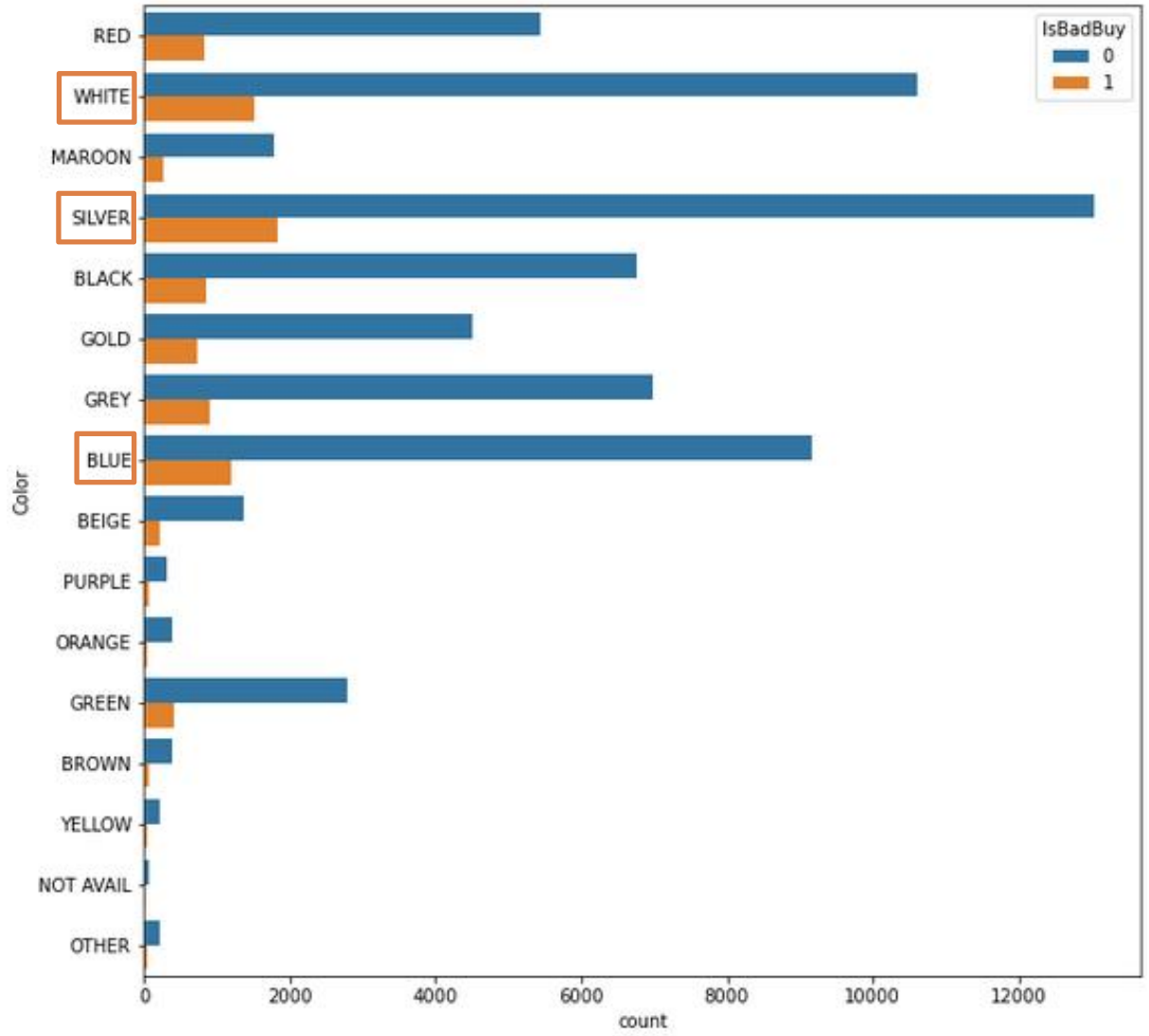
```
fig, ax = plt.subplots(figsize=(10,16))
sns.countplot(y="Make", data=train, hue='IsBadBuy')
plt.show()
```



Exploratory Data Analysis

Impact of color of car being kicket or not

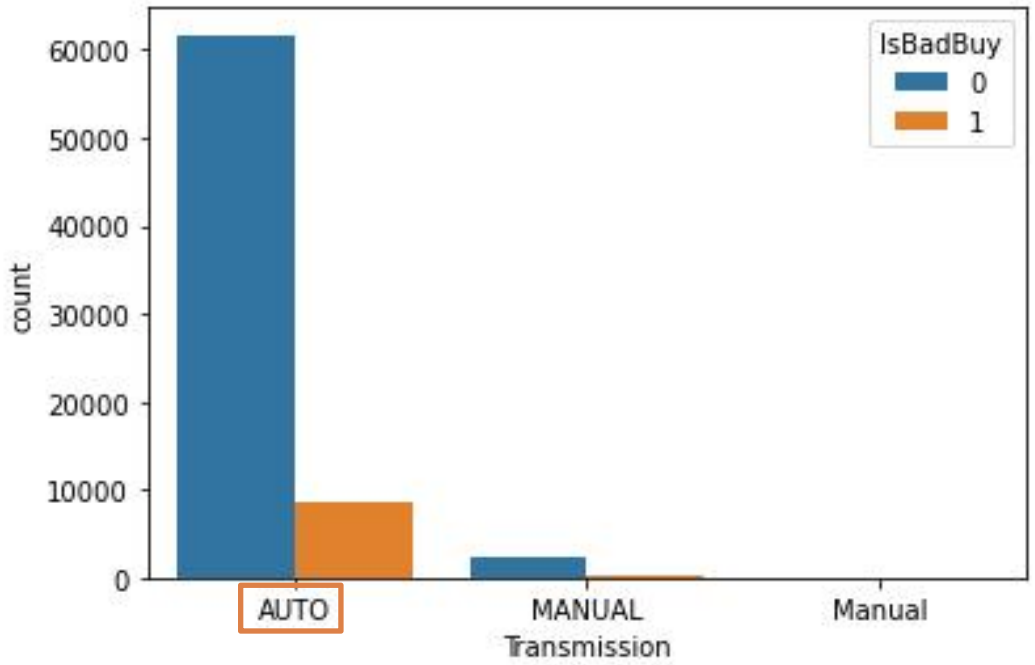
```
fig, ax = plt.subplots(figsize=(10,10))
sns.countplot(y="Color", data=train, hue='IsBadBuy')
plt.show()
```



Exploratory Data Analysis

Impact of transmission type on kicked cars

```
sns.countplot(x="Transmission", data=train, hue='IsBadBuy')
```



Feature Engineering


```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72983 entries, 0 to 72982
Data columns (total 34 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   RefId                             72983 non-null  int64
 1   IsBadBuy                          72983 non-null  int64
 2   PurchDate                         72983 non-null  object
 3   Auction                           72983 non-null  object
 4   VehYear                           72983 non-null  int64
 5   VehicleAge                        72983 non-null  int64
 6   Make                              72983 non-null  object
 7   Model                             72983 non-null  object
 8   Trim                              70623 non-null  object
 9   SubModel                          72975 non-null  object
 10  Color                             72975 non-null  object
 11  Transmission                      72974 non-null  object
 ...
 30  VNST                              72983 non-null  object
 31  VehBCost                          72983 non-null  float64
 32  IsOnlineSale                      72983 non-null  int64
 33  WarrantyCost                      72983 non-null  int64
dtypes: float64(10), int64(9), object(15)
memory usage: 18.9+ MB
```

Feature Engineering

Null values

```
print(train.isnull().sum())
```

```
RefId                                     0
IsBadBuy                                  0
PurchDate                                 0
Auction                                   0
VehYear                                   0
VehicleAge                                0
Make                                      0
Model                                     0
Trim                                   2360
SubModel                                  8
Color                                     8
Transmission                              9
WheelTypeID                            3169
WheelType                              3174
VehOdo                                    0
Nationality                               5
Size                                      5
TopThreeAmericanName                      5
MMRAcquisitionAuctionAveragePrice        18
MMRAcquisitionAuctionCleanPrice          18
MMRAcquisitionRetailAveragePrice         18
MMRAcquisitonRetailCleanPrice            18
MMRCurrentAuctionAveragePrice           315
...
```

```
print(test.isnull().sum())
```

```
RefId                                     0
PurchDate                                 0
Auction                                   0
VehYear                                   0
VehicleAge                                0
Make                                      0
Model                                     0
Trim                                   1550
SubModel                                  5
Color                                     4
Transmission                              3
WheelTypeID                            2188
WheelType                              2188
VehOdo                                    0
Nationality                               7
Size                                      7
TopThreeAmericanName                      7
MMRAcquisitionAuctionAveragePrice        10
MMRAcquisitionAuctionCleanPrice          10
MMRAcquisitionRetailAveragePrice         10
MMRAcquisitonRetailCleanPrice            10
MMRCurrentAuctionAveragePrice           143
...
```

Feature Engineering

Checking duplicates

```
train[train.duplicated()]
```

RefId	IsBadBuy	PurchDate	Auction	VehYear	VehicleAge	Make	Model	Trim	Subl

```
test[test.duplicated()]
```

RefId	PurchDate	Auction	VehYear	VehicleAge	Make	Model	Trim	SubModel	Col

There are no duplicates

Feature Engineering

Drop features since they not seem relevant as they contain specific details which may not help model to learn better

- *PurchDate* (date might not be relevant but year can be), *WheelTypeID*, *Model*, *Trim*, *SubModel*, *Make*, *VNZIP1*, *VNST*

Create additional features

```
def split_date(df):
    df['PurchDate'] = pd.to_datetime(df['PurchDate'])
    df['Year'] = df.PurchDate.dt.year
```

```
def MeanOnFeatures(df):
    df['mean_MMRCurrentAuctionAveragePrice_Make']=train.groupby(['Make'])['MMRCurrentAuctionAveragePrice'].transform('mean')
    df['mean_MMRCurrentAuctionAveragePrice_Model']=train.groupby(['Model'])['MMRCurrentAuctionAveragePrice'].transform('mean')
    df['mean_MMRCurrentAuctionAveragePrice_Trim']=train.groupby(['Trim'])['MMRCurrentAuctionAveragePrice'].transform('mean')
    df['mean_MMRCurrentAuctionAveragePrice_SubModel']=train.groupby(['SubModel'])['MMRCurrentAuctionAveragePrice'].transform('mean')
    df['mean_MMRCurrentAuctionAveragePrice_Color']=train.groupby(['Color'])['MMRCurrentAuctionAveragePrice'].transform('mean')
    df['mean_MMRCurrentAuctionAveragePrice_Transmission']=train.groupby(['Transmission'])['MMRCurrentAuctionAveragePrice'].transform('mean')
```

Feature Engineering


```
print(train.shape, test.shape)
```

```
(72983, 34) (48707, 33)
```

```
split_date(train)
split_date(test)

print(train.shape, test.shape)

MeanOnFeatures(train)
MeanOnFeatures(test)
```

```
(72983, 35) (48707, 34)
```

```
print(train.shape, test.shape)
```

```
(72983, 41) (48707, 40)
```

Feature Engineering


```
print(train.shape, test.shape)
```

```
(72983, 34) (48707, 33)
```

```
split_date(train)
split_date(test)

print(train.shape, test.shape)

MeanOnFeatures(train)
MeanOnFeatures(test)
```

```
(72983, 35) (48707, 34)
```

```
print(train.shape, test.shape)
```

```
(72983, 41) (48707, 40)
```

Handling NaN

```
train.isnull().sum()
```

```
RefId                                  0
IsBadBuy                               0
PurchDate                              0
Auction                                0
VehYear                                0
VehicleAge                             0
Make                                   0
Model                                  0
Trim                                2360
SubModel                               8
Color                                  8
Transmission                           9
WheelTypeID                         3169
WheelType                           3174
VehOdo                                 0
Nationality                            5
Size                                   5
TopThreeAmericanName                   5
MMRAcquisitionAuctionAveragePrice     18
MMRAcquisitionAuctionCleanPrice       18
MMRAcquisitionRetailAveragePrice      18
MMRAcquisitonRetailCleanPrice         18
MMRCurrentAuctionAveragePrice        315
...
```

Feature Engineering


```
print(train.shape, test.shape)
```

```
(72983, 34) (48707, 33)
```

```
split_date(train)
split_date(test)

print(train.shape, test.shape)

MeanOnFeatures(train)
MeanOnFeatures(test)
```

```
(72983, 35) (48707, 34)
```

```
print(train.shape, test.shape)
```

```
(72983, 41) (48707, 40)
```

Merge *MANUAL* and *Manual*

```
train.Transmission.value_counts()
```

```
AUTO      70398
MANUAL     2575
Manual        1
Name: Transmission, dtype: int64
```

```
# Merge MANUAL and Manual
train["Transmission"].replace("Manual", "MANUAL", inplace=True)
```

Handling NaN

```
train.isnull().sum()
```

```
RefId                                  0
IsBadBuy                               0
PurchDate                              0
Auction                                0
VehYear                                0
VehicleAge                             0
Make                                   0
Model                                  0
Trim                                2360
SubModel                               8
Color                                  8
Transmission                           9
WheelTypeID                         3169
WheelType                           3174
VehOdo                                 0
Nationality                            5
Size                                   5
TopThreeAmericanName                   5
MMRAcquisitionAuctionAveragePrice     18
MMRAcquisitionAuctionCleanPrice       18
MMRAcquisitionRetailAveragePrice      18
MMRAcquisitonRetailCleanPrice         18
MMRCurrentAuctionAveragePrice        315
...
```

Feature Engineering

Target, *isBadBuy*, is unbalanced

```
train.IsBadBuy.value_counts()
```

```
0    64007
1     8976
Name: IsBadBuy, dtype: int64
```

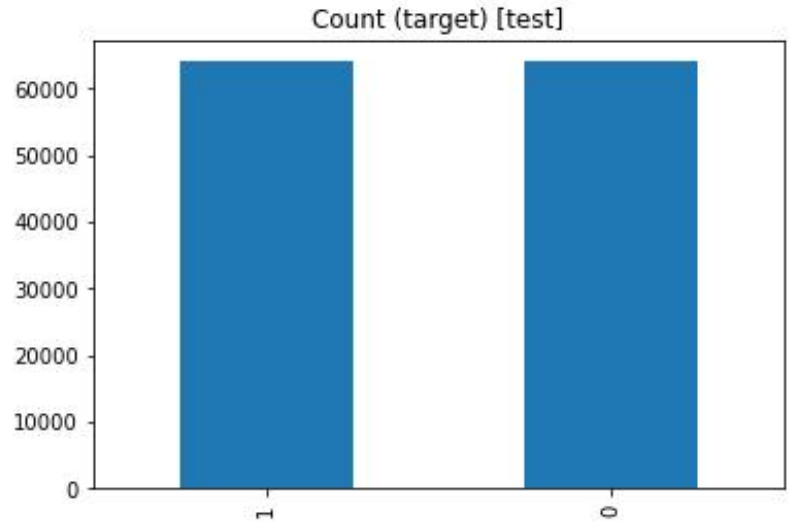
```
count_class_0, count_class_1 = train.IsBadBuy.value_counts()

#separate by value
df_class_0 = train[train['IsBadBuy'] == 0]
df_class_1 = train[train['IsBadBuy'] == 1]

df_class_1_over = df_class_1.sample(count_class_0, replace=True)
df_test_over = pd.concat([df_class_0, df_class_1_over], axis=0)

print(df_test_over.IsBadBuy.value_counts())

df_test_over.IsBadBuy.value_counts().plot(kind='bar', title='Count (target) [test]');
```




```
1    64007
0    64007
Name: IsBadBuy, dtype: int64
```

Data Cleaning And Splitting

Preparation for splitting

```
#create X and y datasets for splitting
X = df_test_over.drop(["RefId",'IsBadBuy'], axis=1)
y = df_test_over['IsBadBuy']
```

```
all_features = X.columns
all_features = all_features.tolist()
```

```
numerical_features = [c for c, dtype in zip(X.columns, X.dtypes)
                      if dtype.kind in ['i','f']]
categorical_features = [c for c, dtype in zip(X.columns, X.dtypes)
                      if dtype.kind not in ['i','f']]
```

Splitting

```
#import train_test_split library
from sklearn.model_selection import train_test_split

# create train test split
X_train, X_test, y_train, y_test = train_test_split(X,  y, test_size=0.3, random_state=42)
```

Random Forest

Implementing Random Forest


```
preprocessor = make_column_transformer(
    (make_pipeline(
        SimpleImputer(strategy = 'median'),
        MinMaxScaler()
    ),
     numerical_features),
    (make_pipeline(
        SimpleImputer(strategy = 'constant', fill_value = 'missing'),
        OneHotEncoder(categories = 'auto', handle_unknown = 'ignore')
    ),
     categorical_features),
)
```

```
preprocessor_best = make_pipeline(
    preprocessor,
    VarianceThreshold(),
    SelectKBest(f_classif, k = 50)
)
```

Simple Imputer to fill the missing values
MinMaxScaler() to normalize the numerical values

Simple Imputer to fill up the missing values
OneHotEncoder to all the categorical columns

Implementing Random Forest


```
RF_Model = make_pipeline(preprocessor_best, RandomForestClassifier(n_estimators = 100))
```

```
RF_Model.fit(X_train, y_train)
RF_Model.score(X_train, y_train)
```

```
1.0
```

The model has learned the training examples excellently and doesn't generalize well to previously unseen examples

```
RF_Model.score(X_test, y_test)
```

```
0.9830230438744955
```

To resolve oversampling, it was induced some overlaps between testing and training set - hence is is observed a very high accuracy

Overfitting

- **Hyperparameter tuning** to overcome the overfitting
- Instead of splitting the data into train and testing set, let us train on the entire set in one go - **K-fold cross validation**

Hyperparameter Tuning


```
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 200, num = 3)]

max_features = ['auto', 'sqrt']

max_depth = [2,6,]

min_samples_split = [2, 5]

min_samples_leaf = [1, 2]

bootstrap = [True, False]
```

Number of trees in random forest

Number of features to consider at every split

Maximum number of levels in tree

Minimum number of samples required to split a node

Minimum number of samples required at each leaf node

Method of selecting samples for training each tree

Hyperparameter Tuning


```
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 200, num = 3)]

max_features = ['auto', 'sqrt']

max_depth = [2,6,]

min_samples_split = [2, 5]

min_samples_leaf = [1, 2]

bootstrap = [True, False]
```

```
# Create the param grid
param_grid = {'randomforestclassifier__n_estimators': n_estimators,
              'randomforestclassifier__max_features': max_features,
              'randomforestclassifier__max_depth': max_depth,
              'randomforestclassifier__min_samples_split': min_samples_split,
              'randomforestclassifier__min_samples_leaf': min_samples_leaf,
              'randomforestclassifier__bootstrap': bootstrap
             }
print(param_grid)
```

```
{'randomforestclassifier__n_estimators': [100, 150, 200], 'randomforestclassifier__max_features': ['auto', '
sqrt'], 'randomforestclassifier__max_depth': [2, 6], 'randomforestclassifier__min_samples_split': [2, 5], 'r
andomforestclassifier__min_samples_leaf': [1, 2], 'randomforestclassifier__bootstrap': [True, False]}
```

Hyperparameter Tuning


```
from sklearn.model_selection import RandomizedSearchCV

rf_RandomGrid = RandomizedSearchCV(estimator = RF_Model, param_distributions = param_grid, cv = 3, verbose=1, n_jobs = -1, n_iter = 5, scoring = 'f1')
```

```
%%time
rf_RandomGrid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
CPU times: user 18.2 s, sys: 288 ms, total: 18.4 s
Wall time: 51.4 s
RandomizedSearchCV(cv=3,
                   estimator=Pipeline(steps=[('pipeline',
                                              Pipeline(steps=[('columntransformer',
                                                               ColumnTransformer(transformers=[('pipeline-1',
                                                                                                Pipeline(steps=[('simpleimputer',
                                                                                                                 SimpleImputer(strategy='median')),
                                                                                                                ('minmaxscaler',
                                                                                                                 MinMaxScaler())]),
                                                                                                ['VehYear',
                                                                                                 'VehicleAge',
                                                                                                 'WheelTypeID',
                                                                                                 'VehOdo',
                                                                                                 'MMRAcquisitionAuctionAveragePrice',
                                                                                                 'MMRAcquisitionAuctio...
                   n_iter=5, n_jobs=-1,
                   param_distributions={'randomforestclassifier__bootstrap': [True,
                                                                              False],
                                        'randomforestclassifier__max_depth': [2,
                                                                              6],
                                        'randomforestclassifier__max_features': ['auto',
                                                                                 'sqrt'],
                                        'randomforestclassifier__min_samples_leaf': [1,
                                                                                     2],
                                        'randomforestclassifier__min_samples_split': [2,
                                                                                      5],
                                        'randomforestclassifier__n_estimators': [100,
                                                                                 150,
                                                                                 200]},
                   scoring='f1', verbose=1)
```

Hyperparameter Tuning


```
rf_RandomGrid.score(X_train, y_train)
```

```
0.6737277128350455
```

```
rf_RandomGrid.score(X_test, y_test)
```

```
0.6696835076294804
```

Overfitting reduced

```
rf_RandomGrid.best_estimator_
```

```
Pipeline(steps=[('pipeline',
                 Pipeline(steps=[('columntransformer',
                                  ColumnTransformer(transformers=[('pipeline-1',
                                                                   Pipeline(steps=[('simpleimputer',
                                                                                    SimpleImputer(strategy='median')),
                                                                                   ('minmaxscaler',
                                                                                    MinMaxScaler())]),
                                                                   ['VehYear',
                                                                    'VehicleAge',
                                                                    'WheelTypeID',
                                                                    'VehOdo',
                                                                    'MMRAcquisitionAuctionAveragePrice',
                                                                    'MMRAcquisitionAuctionCleanPrice',
                                                                    'MMRAcquisitionRetail...
                                                                                    OneHotEncoder(handle_unknown='ignore'))]),
                                                                   ['PurchDate',
                                                                    'Auction',
                                                                    'Make',
                                                                    'Model',
                                                                    'Trim',
                                                                    'SubModel',
                                                                    'Color',
                                                                    'Transmission',
                                                                    'WheelType',
                                                                    'Nationality',
                                                                    'Size',
                                                                    'TopThreeAmericanName',
                                                                    'PRIMEUNIT',
                                                                    'AUCGUART',
                                                                    'VNST'])])),
                                 ('variancethreshold', VarianceThreshold()),
                                 ('selectkbest', SelectKBest(k=50))])),
                ('randomforestclassifier',
                 RandomForestClassifier(bootstrap=False, max_depth=6))])
```

Hyperparameter Tuning

Accuracy

```
print(f'Train : {rf_RandomGrid.score(X_train, y_train):.3f}')
print(f'Test : {rf_RandomGrid.score(X_test, y_test):.3f}')
```

```
Train : 0.674
Test : 0.670
```

Generalized model

XGBoost

Implementing XGBoost


```
preprocessor = make_column_transformer(
    (make_pipeline(
        KNNImputer(n_neighbors=2, weights="uniform"),
        MinMaxScaler()),
     numerical_features),
    (make_pipeline(
        SimpleImputer(strategy = 'constant', fill_value = 'missing'),
        OneHotEncoder(categories = 'auto', handle_unknown = 'ignore')),
     categorical_features),
)

preprocessor_best = make_pipeline(
    preprocessor,
    VarianceThreshold(),
    SelectKBest(f_classif, k = 50)
)
```

KNN Imputer to fill the missing values

```
from xgboost import XGBClassifier
import xgboost as xgb
XG_model = make_pipeline(preprocessor_best, XGBClassifier(n_estimators = 100))
```

Implementing XGBoost


```
%%time
XG_model.fit(X, y)
```

```
CPU times: user 6min 6s, sys: 1min 29s, total: 7min 36s
Wall time: 4min 10s
Pipeline(steps=[('pipeline',
                 Pipeline(steps=[('columntransformer',
                                  ColumnTransformer(transformers=[('pipeline-1',
                                                                   Pipeline(steps=[('knnimputer',
                                                                                    KNNImputer(n_neighbors=2)),
                                                                                   ('minmaxscaler',
                                                                                    MinMaxScaler())]),
                                                                   ['VehYear',
                                                                    'VehicleAge',
                                                                    'WheelTypeID',
                                                                    'VehOdo',
                                                                    'MMRAcquisitionAuctionAveragePrice',
                                                                    'MMRAcquisitionAuctionCleanPrice',
                                                                    'MMRAcquisitionRetailAveragePri...
                                                gamma=0, gpu_id=-1, grow_policy='depthwise',
                                                importance_type=None, interaction_constraints='',
                                                learning_rate=0.300000012, max_bin=256,
                                                max_cat_to_onehot=4, max_delta_step=0,
                                                max_depth=6, max_leaves=0, min_child_weight=1,
                                                missing=nan, monotone_constraints='()',
                                                n_estimators=100, n_jobs=0, num_parallel_tree=1,
                                                predictor='auto', random_state=0, reg_alpha=0,
                                                reg_lambda=1, ...))])
```

```
XG_model.score(X, y)
```

```
0.8300029684253285
```

Implementing XGBoost

KFold Cross Validation

```
from sklearn.model_selection import KFold
```

```
def train_and_evaluate(X_train, train_targets, X_val, val_targets, **params):
    model = make_pipeline(
        preprocessor_best,
        XGBClassifier(random_state=42, n_jobs=-1, **params)
    )
    model.fit(X_train, train_targets)
    train_accuracy = model.score(X_train, train_targets)
    val_accuracy = model.score(X_val, val_targets)
    return model, train_accuracy, val_accuracy
```

Implementing XGBoost


```
models = []

for train_idxs, val_idxs in kfold.split(X):
    X_train, train_targets = X.iloc[train_idxs], y.iloc[train_idxs]
    X_val, val_targets = X.iloc[val_idxs], y.iloc[val_idxs]
    model, train_accuracy, val_accuracy = train_and_evaluate(X_train,
                                                             train_targets,
                                                             X_val,
                                                             val_targets,
                                                             max_depth=4,
                                                             n_estimators=20)

    models.append(model)
    print('Train Accuracy: {}, Validation Accuracy: {}'.format(train_accuracy, val_accuracy))
```

```
Train Accuracy: 0.7179599847672613, Validation Accuracy: 0.3466390657344842
Train Accuracy: 0.7261134057864878, Validation Accuracy: 0.30801077998672033
Train Accuracy: 0.7190633818632765, Validation Accuracy: 0.47986564074522514
Train Accuracy: 0.7396764019490094, Validation Accuracy: 0.4164746318790767
Train Accuracy: 0.7381263914385033, Validation Accuracy: 0.426060464026248
```

High

Low

Implementing XGBoost

Use the average of the 5 models so the errors can be reduced

```
#import numpy as np

def predict_avg(models, inputs):
    return np.mean([model.predict(inputs) for model in models], axis=0)
```

```
preds = predict_avg(models, test.reindex([1,31]))
```

Hyperparameter Tuning


```
def test_params_kfold(n_splits, **params):
    train_accuracys, val_accuracys, models = [], [], []
    kfold = KFold(n_splits)
    for train_idxs, val_idxs in kfold.split(X):
        X_train, train_targets = X.iloc[train_idxs], y.iloc[train_idxs]
        X_val, val_targets = X.iloc[val_idxs], y.iloc[val_idxs]
        model, train_accuracy, val_accuracy = train_and_evaluate(X_train, train_targets, X_val, val_targets, **params)
        models.append(model)
        train_accuracys.append(train_accuracy)
        val_accuracys.append(val_accuracy)
    print('Train accuracy: {}, Validation accuracy: {}'.format(np.mean(train_accuracys), np.mean(val_accuracys)))
    return models
```

Hyperparameter Tuning


```
%%time
test_params_kfold(5, n_estimators=500, max_depth=6, learning_rate=0.9)
```

```
Train accuracy: 0.9999316481391263, Validation accuracy: 0.8401045931571172
CPU times: user 1h 50min 12s, sys: 12min 31s, total: 2h 2min 43s
Wall time: 39min 37s
[Pipeline(steps=[('pipeline',
                  Pipeline(steps=[('columntransformer',
                                   ColumnTransformer(transformers=[('pipeline-1',
                                                                    Pipeline(steps=[('knnimputer',
                                                                                     KNNImputer(n_neighbors=2)),
                                                                                    ('minmaxscaler',
                                                                                     MinMaxScaler())]),
                                                                    ['VehYear',
                                                                     'VehicleAge',
                                                                     'WheelTypeID',
                                                                     'VehOdo',
                                                                     'MMRAcquisitionAuctionAveragePrice',
                                                                     'MMRAcquisitionAuctionCleanPrice',
                                                                     'MMRAcquisitionRetailAveragePri...
                                  gamma=0, gpu_id=-1, grow_policy='depthwise',
                                  importance_type=None, interaction_constraints='',
                                  learning_rate=0.9, max_bin=256,
                                  max_cat_to_onehot=4, max_delta_step=0,
                                  max_depth=6, max_leaves=0, min_child_weight=1,
                                  missing=nan, monotone_constraints='()',
                                  n_estimators=500, n_jobs=-1, num_parallel_tree=1,
```

Hyperparameter Tuning


```
#Putting it all together
XG_model_with_paramter_tuning = make_pipeline(
    preprocessor_best,
    XGBClassifier(n_jobs=-1, random_state=42, n_estimators = 500, learning_rate=0.9, max_depth=6)
)

XG_model_with_paramter_tuning.fit(X,y)
```

```
Pipeline(steps=[('pipeline',
                 Pipeline(steps=[('columntransformer',
                                  ColumnTransformer(transformers=[('pipeline-1',
                                                                   Pipeline(steps=[('knnimputer',
                                                                                    KNNImputer(n_neighbors=2)),
                                                                                   ('minmaxscaler',
                                                                                    MinMaxScaler())]),
                                                                   ['VehYear',
                                                                    'VehicleAge',
                                                                    'WheelTypeID',
                                                                    'VehOdo',
                                                                    'MMRAcquisitionAuctionAveragePrice',
                                                                    'MMRAcquisitionAuctionCleanPrice',
                                                                    'MMRAcquisitionRetailAveragePri...
                                  gamma=0, gpu_id=-1, grow_policy='depthwise',
                                  importance_type=None, interaction_constraints='',
                                  learning_rate=0.9, max_bin=256,
                                  max_cat_to_onehot=4, max_delta_step=0,
                                  max_depth=6, max_leaves=0, min_child_weight=1,
                                  missing=nan, monotone_constraints='()',
                                  n_estimators=500, n_jobs=-1, num_parallel_tree=1,
                                  predictor='auto', random_state=42, reg_alpha=0,
                                  reg_lambda=1, ...))])
```

Sample Prediction

Sample Prediction


```
def predict_input(model, single_input):
    input_df = pd.DataFrame([single_input])
    pred = rf_RandomGrid.predict(input_df)
    prob = rf_RandomGrid.predict_proba(input_df)[0]
    return pred, prob[0]
```

```
predict_input(rf_RandomGrid, new_input)
```

```
(array([0]), 0.6534384764915909)
```

```
new_input = {'RefId':48708,
             'PurchDate':2001-12-2,
             'Auction':'ADESA',
             'VehYear':2004,
             'VehicleAge':6,
             'Make':'DODGE',
             'Model':'DURANGO 2WD V8',
             'Trim':'Adv',
             'SubModel':'4D SUV 4.7L ADVENTURER',
             'Color':'SILVER',
             'Transmission':'AUTO',
             'WheelTypeID':1.0,
             'WheelType':'Alloy',
             'VehOdo':33333,
             'Nationality':'TOP LINE ASIAN',
             'Size':'MEDIUM',
             'TopThreeAmericanName':'OTHER',
             'MMRAcquisitionAuctionAveragePrice':7777,
             'MMRAcquisitionAuctionCleanPrice':8888,
             'MMRAcquisitionRetailAveragePrice':10000,
             'MMRAcquisitonRetailCleanPrice':12000,
             'MMRCurrentAuctionAveragePrice':7777,
             'MMRCurrentAuctionCleanPrice':9999,
             'MMRCurrentRetailAveragePrice':11111,
             'MMRCurrentRetailCleanPrice':11111,
             'PRIMEUNIT':'NaN',
             'AUCGUART':'NaN',
             'BYRNO':17777,
             'VNZIP1': 30212,
             'VNST':'GA',
             'VehBCost':7777,
             'IsOnlineSale':0,
             'WarrantyCost':600,
             'Year':2010,
             'mean_MMRCurrentAuctionAveragePrice_Make':7021.627649,
             'mean_MMRCurrentAuctionAveragePrice_Model':7091.0279,
             'mean_MMRCurrentAuctionAveragePrice_Trim':6171.007828,
             'mean_MMRCurrentAuctionAveragePrice_SubModel':5577.972891,
             'mean_MMRCurrentAuctionAveragePrice_Color':6794.120395,
             'mean_MMRCurrentAuctionAveragePrice_Transmission':6180.060667}
```

Saving The Model

Saving The Model


```
import joblib

car_quality_check = {
    'model': rf_RandomGrid
}

joblib.dump(car_quality_check, 'car_quality_check.joblib')
```

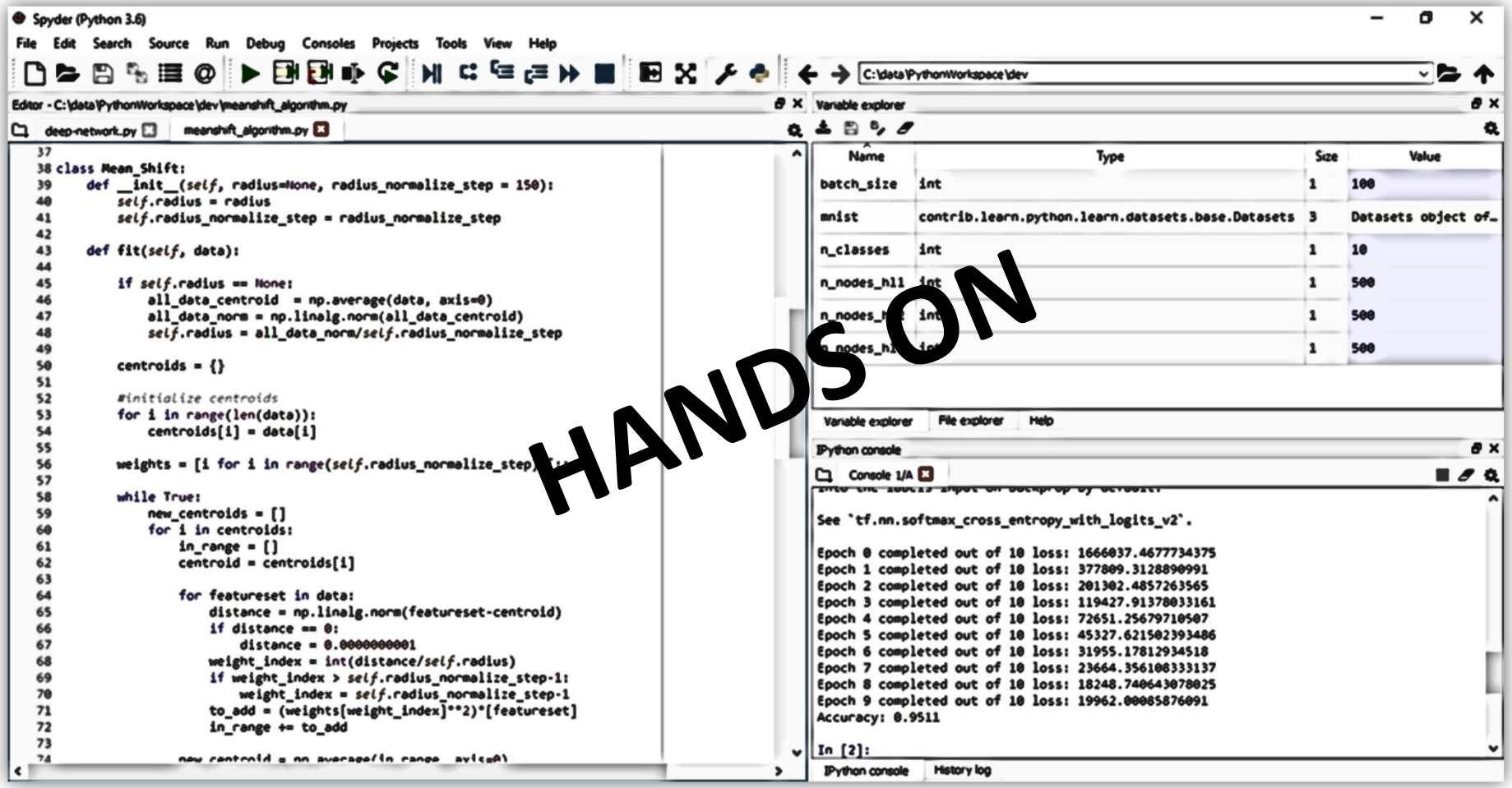
```
['car_quality_check.joblib']
```

```
car_quality_check2 = joblib.load('car_quality_check.joblib')
```

```
test_preds2 = car_quality_check2['model'].score(X_test, y_test)
print(f'Accuracy of the model Random forest is {test_preds2:.3f}')
```

```
Accuracy of the model Random forest is 0.670
```

Hands On


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\data\PythonWorkspace\dev
Editor - C:\data\PythonWorkspace\dev\meanshift_algorithm.py
deep-network.py
meanshift_algorithm.py
class Mean_Shift:
def __init__(self, radius=None, radius_normalize_step = 150):
self.radius = radius
self.radius_normalize_step = radius_normalize_step
def fit(self, data):
if self.radius == None:
all_data_centroid = np.average(data, axis=0)
all_data_norm = np.linalg.norm(all_data_centroid)
self.radius = all_data_norm/self.radius_normalize_step
centroids = {}
#initialize centroids
for i in range(len(data)):
centroids[i] = data[i]
weights = [i for i in range(self.radius_normalize_step
while True:
new_centroids = []
for i in centroids:
in_range = []
centroid = centroids[i]
for featureset in data:
distance = np.linalg.norm(featureset-centroid)
if distance == 0:
distance = 0.0000000001
weight_index = int(distance/self.radius)
if weight_index > self.radius_normalize_step-1:
weight_index = self.radius_normalize_step-1
to_add = (weights[weight_index]**2)*[featureset]
in_range += to_add
Variable explorer
Name Type Size Value
batch_size int 1 100
mnist contrib.learn.python.learn.datasets.base.Datasets 3 Datasets object of…
n_classes int 1 10
n_nodes_hl1 int 1 500
1 500
1 500
Variable explorer File explorer Help
IPython console
Console 1/A
See `tf.nn.softmax_cross_entropy_with_logits_v2`.
Epoch 0 completed out of 10 loss: 1666037.4677734375
Epoch 1 completed out of 10 loss: 377809.3128890991
Epoch 2 completed out of 10 loss: 201302.4857263565
Epoch 3 completed out of 10 loss: 119427.91378033161
Epoch 4 completed out of 10 loss: 72651.25679710507
Epoch 5 completed out of 10 loss: 45327.621502393486
Epoch 6 completed out of 10 loss: 31955.17812934518
Epoch 7 completed out of 10 loss: 23664.356108333137
Epoch 8 completed out of 10 loss: 18248.740643078025
Epoch 9 completed out of 10 loss: 19962.00085876091
Accuracy: 0.9511
In [2]:
IPython console History log
HANDS ON