Kaggle InClass Competition Report

CompSci 671
Chloe Kang and ck53

12/10/2021

The report is for prediction of Airbnb accommodation based on given 20 attributes from Inside Airbnb data platform. There are 7471 of training data and 2440 of test data. Three different methods were used to predict the reservation: Decision Tree, Logistic Regression, and K-nearest neighbors algorithms.

1 Exploratory Analysis

From the Airbnb dataset, there were 20 attributes at the first time: Host_response_time, Host_is_superhost, Host_has_profile_pic, Host_identity_verified, Neighbourhood, Property_type, Room_type, Accommodates, Bathrooms_test, Bedrooms, Beds, Essentials, Cooking, Balcony, Parking, Price, Number_of_reviews, Review_scores_rating, Instant_bookable, Month. Train data also includes id and Decision data to train algorithms.

First, analysis of data has been executed to recognize which parts should be modified for classification. Table 1 shows the number of each data types of attributes. There are 10 numeric variables including int64 and float 64 data type and 10 object variable columns containing strings feature. Also, the distribution of the decision column was examined to see if there is the class imbalance problem. It is the problem that occur when the total number of positive data class is a lot less than that of negative data class in machine learning. According to Figure 1 and the distribution data that was recognized from value_counts, there are 4735 of 1s (decision of reserved Airbnb) and 2736 of 0s (decision of not reserved Airbnb). The percentage of decision of 1s is 63.4% while that of decision of 0s is 36.6%. In decision data, of course decisions of reserved accommodation outweighs than decisions of not reserved, but it is likely to have too much difference to be modified. Also, one of the important factors in the process of machine learning is examine missing values. In the real-world data, it is impossible to keep away from not getting missing values. To examine the proportion of missing data and decide which data should be kept, it is used that the function that can count missing values and the percentage of total values. From Table 2, it was recognized that Host_response_time is the attribute that has the biggest number of missing data with 858 missing values and 11.5 percent of the total train data. Bedrooms and Review_scores_rating are the following data variables that have 7.8% and 5.3%

missing values of total data values each.

Table 1. Data types of 20 attributes

data types	counts
object	10
int64	7
float64	3

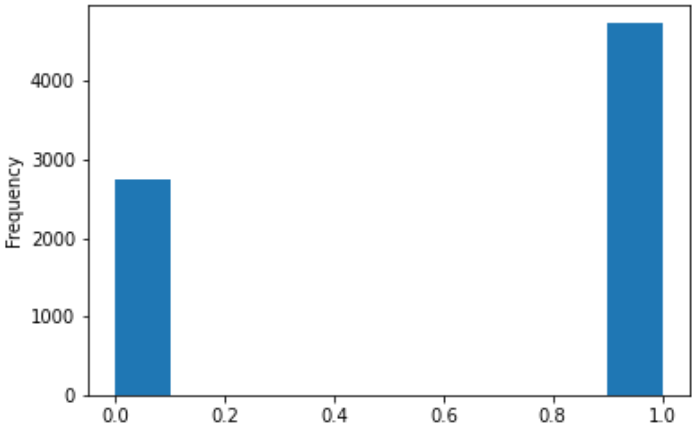


Figure 1. Histogram of Decision in train data

Table 2. The number and Percentage of Missing Values

	Missing Values	% of Total Values
Host_response_time	858	11.5
Bedrooms	585	7.8
Review_scores_rating	395	5.3
Host_is_superhost	96	1.3
Host_has_profile_pic	96	1.3
Host_identity_verified	96	1.3
Beds	13	0.2

Once the analysis of dataset has been completed, the process of modifying and cleaning up data was executed to classify the data effectively and accurately. First, the modification of data type had been executed to let the dataset have the format that can be trained and analyzed. While numeric values including int64 and float64 can be interpreted through the code while categorical values including object cannot be interpreted and analyzed. To make the prediction available, categorical values had been changed to numeric values. There are two methods for encoding categorical values: Label encoding and One-hot encoding. Label encoding is to assign each numeric value to each categorical unique value. In Label encoding process, only a column exists without creating new columns. The numeric value assigned is random and could be reversed every

iteration. In contrast, One-hot encoding create the number of unique values of columns. Each column is for each unique categorical variable. Label encoding has the pros that it has only simple one column but the cons of possibility that the label assigned to the unique values could be reversed due to the randomness at the same time. For the safe option, I decided to use label encoding method for columns that have only 2 unique classes and one-hot encoding method for columns that have more than 2 unique classes.

According to Table 3, it is turned out there 4 categorical attributes that has only 2 unique classes and can apply Label encoding method. With this data, it is likely that Price has too many unique classes, 598 of unique values because of dollar sign and comma. To make price attributes to numeric data rather than encoding all 598 unique values, “replace” function was used by getting rid of ‘$’ and ‘,’ on price data values.

Table 3. The number of unique classes of categorical attributes

Categorical Attributes	**# of unique classes**
Host_response_time	4
Host_is_superhost	2
Host_has_profile_pic	2
Host_identity_verified	2
Property_type	55
Room_type	4
Bathrooms_text	23
Price	598
Instant_bookable	2
Month	4

Once price dataset had been modified to numeric values, Label encoding method was first executed with using LabelEncoder in skelarn.preprocessing package. After getting label encoding conversion, one-hot encoding had been processed. Once all the encoded process was completed, the shape of training features is (7471, 107) that has 107 columns of features and 7471 total data.

The other important modifications that should be done is imputation of missing data. Imputation is the process of missing values with substituted data. Like the missing data had been reviewed on Table 2 earlier, missing values are not too big to drop the entire column data. There are more possibilities to lose valuable data in the process of getting rid of whole column because of small amount of missing data. To visualize the missing data on the plot for better understanding, seaborn is used. According to Figure 2 and Figure 3, of course there are missing data, but it can be possible to check the portion of existing values is much bigger than that of missing values. Therefore, none of any data value was not dropped but imputed.

The method used for imputation is SimpleImputer that allows the empty value to get a

provided constant value or statistical values such as mean, median or most frequent for each column. In this data imputation, I used mean strategy to decrease the impact on the classification and keep the data as much as possible for better training.

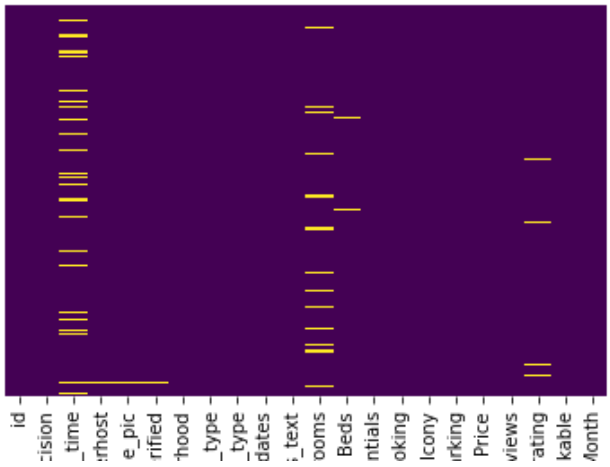


Figure 2. Distribution plot of missing values on training dataset

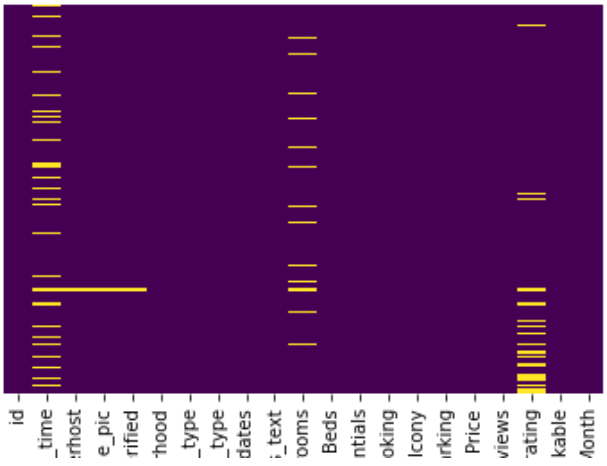


Figure 3: Distribution plot of missing values on test dataset

2 Methods

2.1 Models

There were three methods used to classify the Airbnb accommodation dataset: Decision Tree, Logistic Regression, and K-nearest neighbors algorithms. First, Decision Tree is a simple and efficient supervised learning algorithm by splitting the data points continuously with the

parameters to solve the problem. Even if a decision tree algorithm could be more complex than other algorithms, the usage of decision tree is quite simple because it does not require normalization of data and scaling of data. Because of this characteristic, building decision trees require less effort in data preparation in the preprocessing stage.

Secondly, logistic regression is a learning algorithm that can figure out the probability of true or false. It can be only used when the variable is binary. Of course, logistic regression has the major limitation of the assumption of linearity between the independent variables and the dependent variables. However, this model has couple of strong advantages as well. Logistic regression is one of the easiest models to implement, interpret and train effectively. Also, it is very fast model at classifying unknown records.

Thirdly, K-nearest neighbors (KNN) is a supervised learning algorithm that classify the new data based on a similarity. Even if KNN is slow algorithm in terms of speed and has no capability of handling missing data, KNN algorithm is very simple and intuitive model. Also, this algorithm is memory-based learning, so it adapts very quickly as new training data is coming.

2.2 Training

First of all, for decision tree learning algorithms, data that in train.csv file was separated by train and test data with test size 0.1 by train_test_split from sklearn.model_selection. In other words, from train data file 90% of the data were used as training values while 10% of the data was used as testing values. For y_train data, Decision column was used while for X_train data other attributes were used except id and Decision. The train dataset was trained by DecisionTreeClassifier from sklearn.tree package with 3 of max_depth and 0 of random_state. Once the model had been trained with X and y train dataset, the classification prediction was executed with X_test data from test.csv file. The total wall time for this model was 385ms.

Secondly, for logistic regression learning algorithms, the model was trained by LogisticRegression from sklearn.linear_model. For LogisticRegression, balanced was used as class_weight. In this time, data from train.csv was not separated. The entire data from train.csv was used to train the model. On the trained model, data from test.csv was used as X_test to predict decision. The total wall time for this model was 769 ms.

Lastly, for K-Nearest Negibors learning algorithms, the model was trained by KNeighborsClassifier from sklearn.neighbors package. To train the data, dataset in train.csv file was separated by train and test data with test size 0.1 by train_test_split from sklearn.model_selection package. The total wall time to execute this model was 1.66s.

2.3 Hyper-parameter Selection

Once the data has been modified and clean up, the analysis of correlation was executed to check which attributes are the most relevant to decision either positively or negatively. Through correlations code, it displayed 15 most positive correlations attributes with decision and 15 most negative correlations attributes with decision. From Table 4, it turned out that Host_response_time_within an hour and Instant_bookable have strong correlation with positive decision. Conversely, from Table 5, it can be turned out that Host_response_time_within a day and Host_response_time_within a few hours have quite strong correlation with negative decision. Because there were 85 attributes for total before the data was tuned, I decided to focus on these 30 top most relevant attributes with decision to increase the accuracy of classification. Decision attributes that have 1 as correlations value was omitted because it is redundant.

Table 4. Top 15 of Most Positive Correlations Attributes with Decision

Most Positive Correlations	
Attributes	**Correlations**
Bathrooms_text_1 private bath	0.044426
Property_type_Entire condominium	0.047957
Host_identity_verified	0.05059
Host_is_superhost	0.051808
Room_type_Hotel room	0.053949
Price	0.059403
Property_type_Room in bed and breakfast	0.059617
Property_type_Private room in bed and breakfast	0.065167
Number_of_reviews	0.066657
Property_type_Entire condominium (condo)	0.067271
Month_August	0.070062
Accommodates	0.079489
Instant_bookable	0.218592
Host_response_time_within an hour	0.276863
Decision	1

Table 5. Top 15 of Most Negative Correlation Attribute with Decision

Most Negative Correlations	
Attributes	**Correlations**
Host_response_time_within a day	-0.126946
Host_response_time_within a few hours	-0.113127
Property_type_Entire bungalow	-0.068504
Essentials	-0.066678
Property_type_Entire house	-0.056909
Month_July	-0.054442
Host_response_time_a few days or more	-0.050133
Month_June	-0.049244
Parking	-0.043921
Property_type_Entire guesthouse	-0.041899
Bathrooms_text_1.5 baths	-0.041382
Property_type_Room in hotel	-0.040287
Bathrooms_text_2.5 baths	-0.038033
Property_type_Entire cottage	-0.038009
Room_type_Entire home/apt	-0.037903

3 Results

For the prediction tracks, credits will be awarded based on the classification score. For the interpretability track, half of the credits will be awarded based on the classification score, and the other half is based on the interpretability. We will use subjective criteria to evaluate the interpretability. Note that even if your model does not perform very well in the competition, you can get a good overall grade in this assignment if your writing in the other sections are satisfactory.

3.1 Prediction

For the decision tree model, the accuracy of data was 0.7286. Figure 4 shows all criterion that the decision tree has to classify the dataset. From the logistic regression model, the accuracy of data was 0.64345. Figure 5 shows the proportion of true and false of classification after prediction process. From KNN learning algorithms, the accuracy of training data was 0.67023.

When we compare three algorithms used for classification, decision tree is the most accurate model that produced the highest accuracy and the shortest wall time at the same time. Based on the results of accuracy and time, it is likely that decision tree is the most effective method by handling the Airbnb availability data out of three methods.

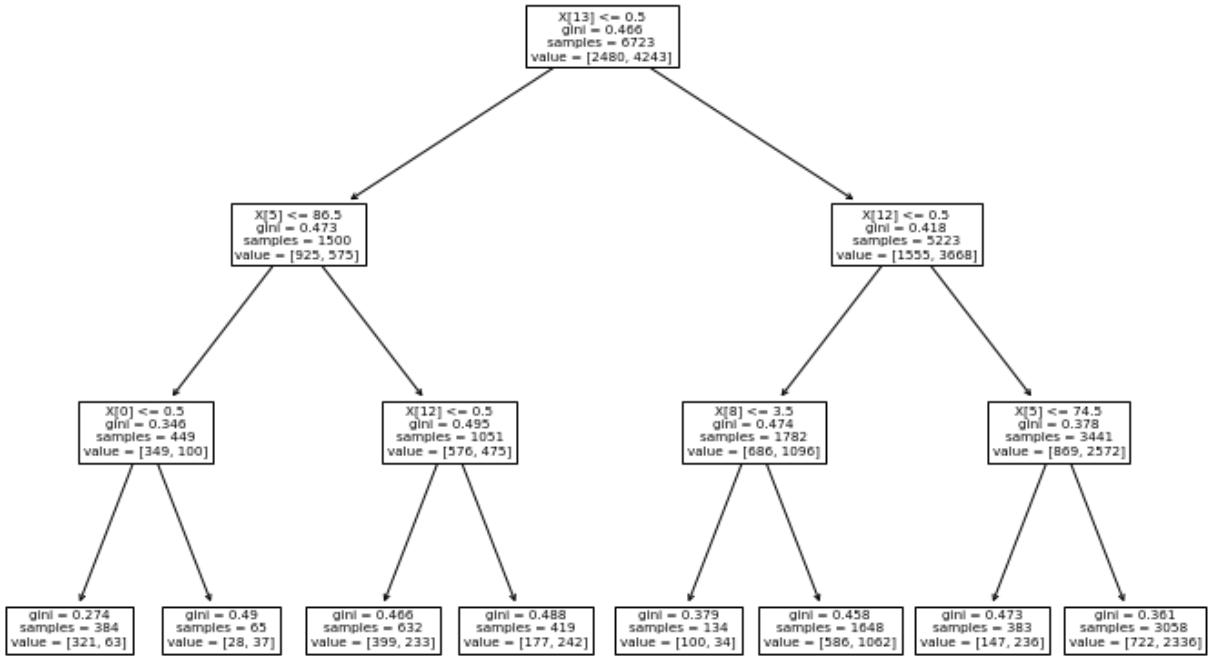


Figure 4. Decision Tree of Airbnb Availability Dataset

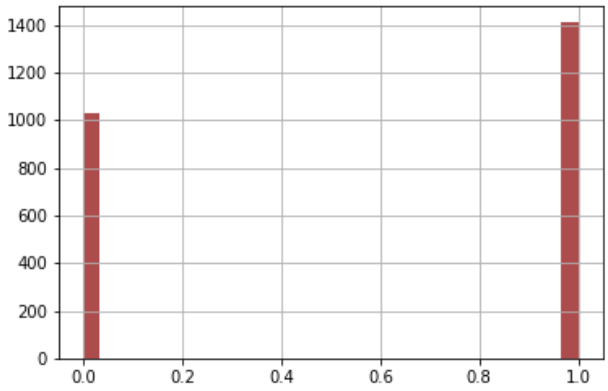


Figure 5. Histogram of Classification from Logistic Regression

3.2 Fixing Mistakes

On the first iteration of Logistic Regression method, the result shows all 1s or 0s which is wrong implementation. On the training process, I separated data from train.csv by 90% train dataset and 10 % test dataset. And then I applied real test dataset from test.csv to trained model. It made the model not applicable to new dataset.

The hardest part of this competition was there are too many unique values on each attribute, so there are going to emerge too many expanded dataset when the categorical attributes are encoded. It made analysis of the problem harder to solve.

Citations

[1] *Logistic regression with python*. DataScience+. (n.d.). Retrieved December 11, 2021, from https://datascienceplus.com/logistic-regression-with-python/.

[2] Willkoehrsen. (2018, August 25). *Start here: A gentle introduction*. Kaggle. Retrieved December 11, 2021, from https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction.

[3] Katari, K. (2020, November 3). *Logistic regression in classification model using Python: Machine Learning*. Medium. Retrieved December 11, 2021, from https://towardsdatascience.com/logistic-regression-in-classification-model-using-python-machine-learning-dc9573e971d0.

[4] Ritesaluja. (2018, June 21). *Logistic regression*. Kaggle. Retrieved December 11, 2021, from https://www.kaggle.com/ritesaluja/logistic-regression.

[5] *K nearest neighbor Classification Algorithm: Knn in python*. Analytics Vidhya. (2021, January 22). Retrieved December 11, 2021, from https://www.analyticsvidhya.com/blog/2021/01/a-quick-introduction-to-k-nearest-neighbor-knn-classification-using-python/.

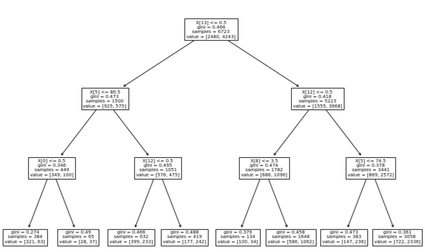
[6] *6.4. imputation of missing values*. scikit. (n.d.). Retrieved December 11, 2021, from https://scikit-learn.org/stable/modules/impute.html.

Code

In [1]:

```
%time
# numpy and pandas for data manipulation
import numpy as np
import pandas as pd

# sklearn preprocessing for dealing with categorical variables (for label encoding)
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer

# File system management
# for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc
import os

# matplotlib and seaborn for plotting
import matplotlib.pyplot as plt

# seaborn is a library for making statistical graphics in Python
import seaborn as sns

## Read in Data

# Training data features
ab_train = pd.read_csv('train.csv')

# Testing data features
ab_test = pd.read_csv('test.csv')

# Cleaning up Price's currency data with pandas
ab_train['Price'] = ab_train['Price'].replace({'\$':'',',':''},regex=True).astype('float')
ab_test['Price'] = ab_test['Price'].replace({'\$':'',',':''},regex=True).astype('float')


## Encoding Categorical Vairables

# Create a label encoder object
le = LabelEncoder()
le_count = 0

# Iterate through the columns
for col in ab_train:
    if ab_train[col].dtype == 'object':
        unique_list = list(ab_train[col].unique())
        unique_list = [x for x in unique_list if pd.isnull(x)==False]
        # If 2 or fewer unique categories
        if len(unique_list) <= 2:
            # Train on the training data
            le.fit(ab_train[col])
            # Transform both training and testing data
            ab_train[col] = le.transform(ab_train[col])
            ab_test[col] = le.transform(ab_test[col])

            # Keep track of how many columns were label encoded
            le_count += 1
            print(col)
print('%d columns were label encoded.' % le_count)

# one-hot encoding of categorical variables

ab_train = pd.get_dummies(ab_train)
ab_test = pd.get_dummies(ab_test)

print('Training Features shape: ', ab_train.shape)
print('Testing Features shape: ', ab_test.shape)

ab_train.fillna(ab_train.mean(), inplace = True)
ab_test.fillna(ab_test.mean(), inplace = True)

## Aligining Training and Testing Data

train_labels = ab_train['Decision']

# Align the training and testing data, keep only columns present in both dataframes
ab_train, ab_test = ab_train.align(ab_test, join = 'inner', axis = 1)

# Add the target back in
ab_train['Decision'] = train_labels

## Corelation

# Find correlations with the target and sort
correlations = ab_train.corr()['Decision'].sort_values()

# Disply correlations
print('Most Positive Correlations:\n', correlations.tail(15))
print('\nMost Negative Correlations:\n', correlations.head(15))

ab_train_par = ab_train[['Bathrooms_text_1 private bath','Property_type_Entire condominium','Host_identity_verified','Host_is_superhost','Room_type
ab_test_par = ab_test[['Bathrooms_text_1 private bath','Property_type_Entire condominium','Host_identity_verified','Host_is_superhost','Room_type_H


## Fitting the Model, Evaluating Result, and Visualizing Trees

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

## Imputing missing values for sklearn decision tree

# retrieve the numpy array
train_values = ab_train_par.values
test_values = ab_test_par.values

#degine the imputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
transformed_train_values = imp.fit_transform(train_values)
transformed_test_values = imp.fit_transform(test_values)

# Set up the train data with columns that are in use
X = ab_train_par
y = ab_train['Decision']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 42)

clf_en = DecisionTreeClassifier(max_depth = 3, random_state = 0)
clf_en.fit(X_train, y_train)

y_pred_en = clf_en.predict(X_test)

## evaluate the classifier
# confusion matrix of the classifier

from sklearn.metrics import accuracy_score
print('Model accuracy score with criterion entropy: {0:0.4f}'.format(accuracy_score(y_test, y_pred_en)))

y_pred_train_en = clf_en.predict(X_train)
y_pred_train_en

plt.figure(figsize=(12,8))

from sklearn import tree
tree.plot_tree(clf_en.fit(X_train, y_train))
plt.savefig('tree.png',bbox_inches = "tight")

## predict classification with test data

#X_test_test = ab_test.drop(['id'], axis=1)
X_test_test = ab_test_par
test_pre = clf_en.predict(X_test_test)

id = ab_test['id'].values.tolist()
df = pd.DataFrame({'id':id,'Decision':test_pre})
print(df)

#df.to_csv('result_DT.csv', index = False)
```

```
Wall time: 0 ns
Host_is_superhost
Host_has_profile_pic
Host_identity_verified
Instant_bookable
4 columns were label encoded.
Training Features shape:  (7471, 107)
Testing Features shape:  (2440, 89)
Most Positive Correlations:
 Bathrooms_text_1 private bath                0.044426
Property_type_Entire condominium              0.047957
Host_identity_verified                        0.050590
Host_is_superhost                             0.051808
Room_type_Hotel room                          0.053949
Price                                         0.059403
Property_type_Room in bed and breakfast       0.059617
Property_type_Private room in bed and breakfast    0.065167
Number_of_reviews                             0.066657
Property_type_Entire condominium (condo)      0.067271
Month_August                                  0.070062
Accommodates                                  0.079489
Instant_bookable                              0.218592
Host_response_time_within an hour             0.276863
Decision                                      1.000000
Name: Decision, dtype: float64

Most Negative Correlations:
 Host_response_time_within a day        -0.126946
Host_response_time_within a few hours   -0.113127
Property_type_Entire bungalow           -0.068504
Essentials                              -0.066678
Property_type_Entire house              -0.056909
Month_July                              -0.054442
Host_response_time_a few days or more   -0.050133
Month_June                              -0.049244
Parking                                 -0.043921
Property_type_Entire guesthouse         -0.041899
Bathrooms_text_1.5 baths                -0.041382
Property_type_Room in hotel             -0.040287
Bathrooms_text_2.5 baths                -0.038033
Property_type_Entire cottage            -0.038009
Room_type_Entire home/apt               -0.037903
Name: Decision, dtype: float64
Model accuracy score with criterion entropy: 0.7286
        id  Decision
0        1         1
1        2         1
2        3         1
3        4         1
4        5         1
...    ...       ...
2435  2436         1
2436  2437         1
2437  2438         1
2438  2439         1
2439  2440         1

[2440 rows x 2 columns]
```



In [3]:

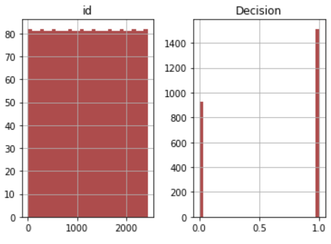
```
%%time

import pandas as pd
import numpy as np

# Seaborn and Matplotlib for data visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# sklearn preprocessing for dealing with categorical variables (for label encoding)
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import log_loss,roc_auc_score, recall_score, precision_score, average_precision_score, f1_score, classification_report, accura

# Read train data
ab_train = pd.read_csv('train.csv')

# Read test data
ab_test = pd.read_csv('test.csv')

## Converting Categorical Features

# Cleaning up Price's currency data with pandas
ab_train['Price'] = ab_train['Price'].replace({'\$':'',',':''},regex=True).astype('float')
ab_test['Price'] = ab_test['Price'].replace({'\$':'',',':''},regex=True).astype('float')


## Encoding Categorical Vairables

# Create a label encoder object
le = LabelEncoder()
le_count = 0

# Iterate through the columns
for col in ab_train:
    if ab_train[col].dtype == 'object':
        unique_list = list(ab_train[col].unique())
        unique_list = [x for x in unique_list if pd.isnull(x)==False]
        # If 2 or fewer unique categories
        if len(unique_list) <= 2:
            # Train on the training data
            le.fit(ab_train[col])
            # Transform both training and testing data
            ab_train[col] = le.transform(ab_train[col])
            ab_test[col] = le.transform(ab_test[col])

            # Keep track of how many columns were label encoded
            le_count += 1
            print(col)
print('%d columns were label encoded.' % le_count)

# one-hot encoding of categorical variables

ab_train = pd.get_dummies(ab_train)
ab_test = pd.get_dummies(ab_test)

print('Training Features shape: ', ab_train.shape)
print('Testing Features shape: ', ab_test.shape)

ab_train.fillna(ab_train.mean(), inplace = True)
ab_test.fillna(ab_test.mean(), inplace = True)

## Aligining Training and Testing Data

train_labels = ab_train['Decision']

# Align the training and testing data, keep only columns present in both dataframes
ab_train, ab_test = ab_train.align(ab_test, join = 'inner', axis = 1)

# Add the target back in
ab_train['Decision'] = train_labels

## Imputing missing values for sklearn decision tree

# retrieve the numpy array
train_values = ab_train.values
test_values = ab_test.values

#degine the imputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
transformed_train_values = imp.fit_transform(train_values)
transformed_test_values = imp.fit_transform(test_values)

ab_train_par = ab_train[['Bathrooms_text_1 private bath','Property_type_Entire condominium','Host_identity_verified','Host_is_superhost','Room_type
ab_test_par = ab_test[['Bathrooms_text_1 private bath','Property_type_Entire condominium','Host_identity_verified','Host_is_superhost','Room_type_H

## Build a Logistic Regression model
X = ab_train_par
y = ab_train['Decision']

## Training and Predicting

from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(class_weight='balanced')
clf.fit(X,y)
X_test = ab_test_par
predictions = clf.predict(X_test)

id = ab_test['id'].values.tolist()
df = pd.DataFrame({'id':id,'Decision':predictions})
print(df)
df.hist(bins=30, color='darkred',alpha=0.7)

#df.to_csv('result_logistic regression.csv', index = False)
```

```
Host_is_superhost
Host_has_profile_pic
Host_identity_verified
Instant_bookable
4 columns were label encoded.
Training Features shape:  (7471, 107)
Testing Features shape:  (2440, 89)
        id  Decision
0        1         0
1        2         1
2        3         1
3        4         1
4        5         1
...    ...       ...
2435  2436         1
2436  2437         1
2437  2438         1
2438  2439         1
2439  2440         1

[2440 rows x 2 columns]
Wall time: 779 ms
```

```
c:\users\owner\appdata\local\programs\python\python39\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[3]:

```
array([[<AxesSubplot:title={'center':'id'}>,
        <AxesSubplot:title={'center':'Decision'}>]], dtype=object)
```



In [ ]:

In [4]:

```
%%time
# numpy and pandas for data manipulation
import numpy as np
import pandas as pd

# sklearn preprocessing for dealing with categorical variables (for label encoding)
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn import linear_model

# File system management
# for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc
import os
import sys

# matplotlib and seaborn for plotting
import matplotlib.pyplot as plt
# seaborn is a library for making statistical graphics in Python
import seaborn as sns

# Read train data
ab_train = pd.read_csv('train.csv')

# Read test data
ab_test = pd.read_csv('test.csv')

## Converting Categorical Features

# Cleaning up Price's currency data with pandas
ab_train['Price'] = ab_train['Price'].replace({'\$':'',',':''},regex=True).astype('float')
ab_test['Price'] = ab_test['Price'].replace({'\$':'',',':''},regex=True).astype('float')

## Encoding Categorical Vairables

# Create a label encoder object
le = LabelEncoder()
le_count = 0

# Iterate through the columns
for col in ab_train:
    if ab_train[col].dtype == 'object':
        unique_list = list(ab_train[col].unique())
        unique_list = [x for x in unique_list if pd.isnull(x)==False]
        # If 2 or fewer unique categories
        if len(unique_list) <= 2:
            # Train on the training data
            le.fit(ab_train[col])
            # Transform both training and testing data
            ab_train[col] = le.transform(ab_train[col])
            ab_test[col] = le.transform(ab_test[col])

            # Keep track of how many columns were label encoded
            le_count += 1
            print(col)
print('%d columns were label encoded.' % le_count)

# one-hot encoding of categorical variables

ab_train = pd.get_dummies(ab_train)
ab_test = pd.get_dummies(ab_test)

print('Training Features shape: ', ab_train.shape)
print('Testing Features shape: ', ab_test.shape)

ab_train.fillna(ab_train.mean(), inplace = True)
ab_test.fillna(ab_test.mean(), inplace = True)

## Aligining Training and Testing Data

train_labels = ab_train['Decision']

# Align the training and testing data, keep only columns present in both dataframes
ab_train, ab_test = ab_train.align(ab_test, join = 'inner', axis = 1)

# Add the target back in
ab_train['Decision'] = train_labels

## Imputing missing values for sklearn decision tree

# retrieve the numpy array
train_values = ab_train.values
test_values = ab_test.values

#degine the imputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
transformed_train_values = imp.fit_transform(train_values)
transformed_test_values = imp.fit_transform(test_values)


# Set up the train data with columns that are in use
X = ab_train.drop(['Decision','id'], axis=1)
y = ab_train['Decision']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 0)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
y_pred=classifier.predict(X_test)

from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test,y_pred)

X_test_test = ab_test.drop(['id'], axis=1)
y_pred = classifier.predict(X_test_test)

id = ab_test['id'].values.tolist()
df = pd.DataFrame({'id':id,'Decision':y_pred})

#df.to_csv('result_KNN.csv', index = False)
```

```
Host_is_superhost
Host_has_profile_pic
Host_identity_verified
Instant_bookable
4 columns were label encoded.
Training Features shape:  (7471, 107)
Testing Features shape:  (2440, 89)
c:\users\owner\appdata\local\programs\python\python39\lib\site-packages\sklearn\base.py:434: UserWarning: X has feature names, but KNeighborsClassi
fier was fitted without feature names
  warnings.warn(
Wall time: 1.31 s
```

In [ ]: