AirBnb Ratings Classification

Matthew Zhang

Objective:

Often times AirBnb success is determined by customer experience. In this case, I will be a company analyst looking to help AirBnb hosts enhance overall customer satisfaction by classifying listings into three categories: Subpar(0), Good(1) and Best(2); further, determining which aspects of properties can be improved. Through modeling, I hope to gain valuable insights in order to form strategy and help generate company success.

To illustrate which characteristics of listings are important, I hope to perform feature level analysis to gain further insights into the models and how they behave.

Data Cleaning

In [1]:
```
#Import necessary packages
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix,mean_
squared_error, plot_confusion_matrix
from sklearn.metrics import precision_score, recall_score, accuracy_scor
e, f1_score
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')
```

In [9]:
```
#Import relevant dataset
df = pd.read_csv('data/listings (1).csv.gz')
rev = pd.read_csv('data/reviews.csv.gz')
```

In [11]:
```
pd.set_option('display.max_columns', None)
df.head(2)
```

Out[11]:

	id	listing_url	scrape_id	last_scraped	name	description
0	109	https://www.airbnb.com/rooms/109	20201103044428	2020-11-03	Amazing bright elegant condo park front *UPGRA...	*** Unit upgraded with new bamboo flooring, br...
1	2708	https://www.airbnb.com/rooms/2708	20201103044428	2020-11-04	Beautiful Furnish Mirrored Mini-Suite w/ Firep...	CDC Airbnb Standard | Steam, Sanitized, Disinf...

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31471 entries, 0 to 31470
Data columns (total 74 columns):
id                                              31471 non-null int64
listing_url                                     31471 non-null object
scrape_id                                       31471 non-null int64
last_scraped                                    31471 non-null object
name                                            31469 non-null object
description                                     30400 non-null object
neighborhood_overview                           20323 non-null object
picture_url                                     31471 non-null object
host_id                                         31471 non-null int64
host_url                                        31471 non-null object
host_name                                       31467 non-null object
host_since                                      31467 non-null object
host_location                                   31389 non-null object
host_about                                      19477 non-null object
host_response_time                              23183 non-null object
host_response_rate                              23183 non-null object
host_acceptance_rate                            24923 non-null object
host_is_superhost                               31467 non-null object
host_thumbnail_url                              31467 non-null object
host_picture_url                                31467 non-null object
host_neighbourhood                              25331 non-null object
host_listings_count                             31467 non-null float64
host_total_listings_count                       31467 non-null float64
host_verifications                              31471 non-null object
host_has_profile_pic                            31467 non-null object
host_identity_verified                          31467 non-null object
neighbourhood                                   20324 non-null object
neighbourhood_cleansed                          31471 non-null object
neighbourhood_group_cleansed                    31471 non-null object
latitude                                        31471 non-null float64
longitude                                       31471 non-null float64
property_type                                   31471 non-null object
room_type                                       31471 non-null object
accommodates                                    31471 non-null int64
bathrooms                                       0 non-null float64
bathrooms_text                                  31440 non-null object
bedrooms                                        27831 non-null float64
beds                                            31162 non-null float64
amenities                                       31471 non-null object
price                                           31471 non-null object
minimum_nights                                  31471 non-null int64
maximum_nights                                  31471 non-null int64
minimum_minimum_nights                          31469 non-null float64
maximum_minimum_nights                          31469 non-null float64
minimum_maximum_nights                          31469 non-null float64
maximum_maximum_nights                          31469 non-null float64
minimum_nights_avg_ntm                          31469 non-null float64
maximum_nights_avg_ntm                          31469 non-null float64
calendar_updated                                0 non-null float64
has_availability                                31471 non-null object
availability_30                                 31471 non-null int64
availability_60                                 31471 non-null int64
availability_90                                 31471 non-null int64
availability_365                                31471 non-null int64
```

```
calendar_last_scraped                           31471 non-null object
number_of_reviews                               31471 non-null int64
number_of_reviews_ltm                           31471 non-null int64
number_of_reviews_l30d                          31471 non-null int64
first_review                                    24244 non-null object
last_review                                     24244 non-null object
review_scores_rating                            23859 non-null float64
review_scores_accuracy                          23726 non-null float64
review_scores_cleanliness                       23726 non-null float64
review_scores_checkin                           23716 non-null float64
review_scores_communication                     23724 non-null float64
review_scores_location                          23713 non-null float64
review_scores_value                             23709 non-null float64
license                                         6549 non-null object
instant_bookable                                31471 non-null object
calculated_host_listings_count                  31471 non-null int64
calculated_host_listings_count_entire_homes     31471 non-null int64
calculated_host_listings_count_private_rooms    31471 non-null int64
calculated_host_listings_count_shared_rooms     31471 non-null int64
reviews_per_month                               24244 non-null float64
dtypes: float64(22), int64(17), object(35)
memory usage: 17.8+ MB
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['id', 'listing_url', 'scrape_id', 'last_scraped', 'name', 'descr
iption',
       'neighborhood_overview', 'picture_url', 'host_id', 'host_url',
       'host_name', 'host_since', 'host_location', 'host_about',
       'host_response_time', 'host_response_rate', 'host_acceptance_rat
e',
       'host_is_superhost', 'host_thumbnail_url', 'host_picture_url',
       'host_neighbourhood', 'host_listings_count',
       'host_total_listings_count', 'host_verifications',
       'host_has_profile_pic', 'host_identity_verified', 'neighbourhoo
d',
       'neighbourhood_cleansed', 'neighbourhood_group_cleansed', 'latit
ude',
       'longitude', 'property_type', 'room_type', 'accommodates', 'bath
rooms',
       'bathrooms_text', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
       'maximum_minimum_nights', 'minimum_maximum_nights',
       'maximum_maximum_nights', 'minimum_nights_avg_ntm',
       'maximum_nights_avg_ntm', 'calendar_updated', 'has_availabilit
y',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'calendar_last_scraped', 'number_of_review
s',
       'number_of_reviews_ltm', 'number_of_reviews_l30d', 'first_revie
w',
       'last_review', 'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'license', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'reviews_per_mont
h'],
      dtype='object')
```

In [6]:
```
#Examing columns that can be converted to dummy variables
df['host_response_time'].unique()
```

Out[6]:
```
array([nan, 'within an hour', 'within a few hours', 'within a day',
       'a few days or more'], dtype=object)
```

I still want to add host_response and host_acceptance rates even though they are object type. They seem to have relevance. Start by stripping "%" and converting each value to a float.

In [12]:
```
df['host_response_rate'] = df['host_response_rate'].str.rstrip('%').asty
pe('float') / 100.0
df['host_acceptance_rate'] = df['host_acceptance_rate'].str.rstrip('%').
astype('float') / 100.0
```

In [18]:
```
df['price'] = df['price'].str.lstrip('$')
```

In [20]:
```
df['price'] = pd.to_numeric(df['price'], errors='coerce')
```

Since I will be performing classification based modeling, I want all the variables to be numbers. I decided to create a dataframe with all existing numericals and then adding categoricals that will later be converted to dummy variables.

In [22]:
```
#Select all int and float types
df1 = pd.DataFrame()
df1 = df.select_dtypes(include=['int64', 'float64'])
```

In [23]:
```
df1.head(2)
```

Out[23]:

	id	scrape_id	host_id	host_response_rate	host_acceptance_rate	host_listings_count
0	109	20201103044428	521	NaN	0.0	1.0
1	2708	20201103044428	3008	1.0	1.0	2.0

In [24]:
```
#Add relevant categoricals/object types that will be converted to dummie
s with get_dummies
df1['host_response_time'] = df['host_response_time']
df1['room_type'] = df['room_type']
```

This is to convert all boolean values to binary encoding.

In [25]:
```
#Convert t/f to binary encoding.
df1['instant_bookable'] = df['instant_bookable'].map({'f': 0, 't': 1})
df1['has_availability'] = df['has_availability'].map({'f': 0, 't': 1})
df1['host_identity_verified'] = df['host_identity_verified'].map({'f': 0
, 't': 1})
df1['host_has_profile_pic'] = df['host_has_profile_pic'].map({'f': 0,
't': 1})
df1['host_is_superhost'] = df['host_is_superhost'].map({'f': 0, 't': 1})
```

In [26]:
```
df1.head(2)
```

Out[26]:

	id	scrape_id	host_id	host_response_rate	host_acceptance_rate	host_listings_count
0	109	20201103044428	521	NaN	0.0	1.0
1	2708	20201103044428	3008	1.0	1.0	2.0

In [27]:
```
#One hot encode categoricals (neighbourhood_group_cleansed, host_respons
e_time), drop_first = True for logistic
#Although not required for the other models I will be running
df2 = pd.get_dummies(df1)
```

Null values

Null values for this data set were particulularly interesting since some of the columns didn't have a clear value to replace the NaN values with. If I were to simply replace NaN's with any measure of central tendency, the distribution of that column/feature would skew heavily. Also removing the NaN's would potentially remove a lot of important information as many columns had around 8,000/31,000 null values.

A count of the null values will be displayed below. However, as there were also many columns with few null values (2, 4) I made the decision to just drop those rows since there wouldn't be much impact on the distribution of the data for this case. 2/31,000.

In [28]:
```
#Clearly there are a lot of null values that need to be dealt with.
df2.head(2)
```

Out[28]:

	id	scrape_id	host_id	host_response_rate	host_acceptance_rate	host_listings_count
0	109	20201103044428	521	NaN	0.0	1.0
1	2708	20201103044428	3008	1.0	1.0	2.0

In [29]:
```
df2.isna().sum()
```

Out[29]:
```
id                                              0
scrape_id                                       0
host_id                                         0
host_response_rate                           8288
host_acceptance_rate                         6548
host_listings_count                             4
host_total_listings_count                       4
latitude                                        0
longitude                                       0
accommodates                                    0
bathrooms                                   31471
bedrooms                                     3640
beds                                          309
price                                         889
minimum_nights                                  0
maximum_nights                                  0
minimum_minimum_nights                          2
maximum_minimum_nights                          2
minimum_maximum_nights                          2
maximum_maximum_nights                          2
minimum_nights_avg_ntm                          2
maximum_nights_avg_ntm                          2
calendar_updated                            31471
availability_30                                 0
availability_60                                 0
availability_90                                 0
availability_365                                0
number_of_reviews                               0
number_of_reviews_ltm                           0
number_of_reviews_l30d                          0
review_scores_rating                         7612
review_scores_accuracy                       7745
review_scores_cleanliness                    7745
review_scores_checkin                        7755
review_scores_communication                  7747
review_scores_location                       7758
review_scores_value                          7762
calculated_host_listings_count                  0
calculated_host_listings_count_entire_homes     0
calculated_host_listings_count_private_rooms    0
calculated_host_listings_count_shared_rooms     0
reviews_per_month                            7227
instant_bookable                                0
has_availability                                0
host_identity_verified                          4
host_has_profile_pic                            4
host_is_superhost                               4
host_response_time_a few days or more           0
host_response_time_within a day                 0
host_response_time_within a few hours           0
host_response_time_within an hour               0
room_type_Entire home/apt                       0
room_type_Hotel room                            0
room_type_Private room                          0
room_type_Shared room                           0
dtype: int64
```

In [30]:
```
df2 = df2.drop(['bathrooms', 'calendar_updated', 'id', 'scrape_id', 'hos
t_id'], axis=1)
```

In [34]:
```
#Based on median: 1.0
df2['host_response_rate'].replace(np.NaN, 1, inplace=True)
df2['bedrooms'].replace(np.NaN, 1, inplace=True)
df2['beds'].replace(np.NaN, 1, inplace=True)
df2['review_scores_checkin'].replace(np.NaN, 10, inplace=True)
df2['review_scores_communication'].replace(np.NaN, 10, inplace=True)
df2['review_scores_location'].replace(np.NaN, 10, inplace=True)
df2['review_scores_accuracy'].replace(np.NaN, 10, inplace=True)
```

Host Acceptance rate is interesting it has ~8000/31000 null values so making them a single value would skew the data heavily. The values don't lean toward a specific value so I concluded the best way would be to use the np.random_choice() method.

In [35]:
```
#To show that values are being evenly distributed based on previous perc
entage distribution
x = df2['host_acceptance_rate'].value_counts(normalize=True)
print(x)
```

```
1.00    0.274887
0.99    0.088553
0.98    0.067167
0.97    0.044979
0.00    0.038077
          ...
0.15    0.000120
0.12    0.000120
0.24    0.000080
0.05    0.000040
0.02    0.000040
Name: host_acceptance_rate, Length: 99, dtype: float64
```

In [36]:
```
missing = df2['host_acceptance_rate'].isnull()
df2.loc[missing,'host_acceptance_rate'] = np.random.choice(x.index, size
=len(df2[missing]),p=x.values)
```

In [37]:
```
df2['host_acceptance_rate'].value_counts(normalize=True)
```

Out[37]:
```
1.00    0.274221
0.99    0.089130
0.98    0.067205
0.97    0.045502
0.00    0.038162
          ...
0.07    0.000095
0.12    0.000095
0.24    0.000095
0.05    0.000032
0.02    0.000032
Name: host_acceptance_rate, Length: 99, dtype: float64
```

The following is repetitive information that fills in ~8,000 null values for each of these specified columns using random choice to evenly replace NaNs based on percentages each unique values appearance (Hence, normalize=True).

In [38]:
```
a = df2['review_scores_rating'].value_counts(normalize=True)
```

In [39]:
```
missing1 = df2['review_scores_rating'].isnull()
df2.loc[missing1,'review_scores_rating'] = np.random.choice(a.index, siz
e=len(df2[missing1]),p=a.values)
```

In [40]:
```
b = df2['review_scores_cleanliness'].value_counts(normalize=True)
```

In [41]:
```
missing2 = df2['review_scores_cleanliness'].isnull()
df2.loc[missing2,'review_scores_cleanliness'] = np.random.choice(b.index
, size=len(df2[missing2]),p=b.values)
```

In [42]:
```
c = df2['review_scores_value'].value_counts(normalize=True)
```

In [43]:
```
missing3 = df2['review_scores_value'].isnull()
df2.loc[missing3,'review_scores_value'] = np.random.choice(c.index, size
=len(df2[missing3]),p=c.values)
```

In [44]:
```
d = df2['reviews_per_month'].value_counts(normalize=True)
```

In [45]:
```
missing4 = df2['reviews_per_month'].isnull()
df2.loc[missing4,'reviews_per_month'] = np.random.choice(d.index, size=l
en(df2[missing4]),p=d.values)
```

In [47]:
```
e = df2['price'].value_counts(normalize=True)
```

In [48]:
```
missing5 = df2['price'].isnull()
df2.loc[missing5,'price'] = np.random.choice(e.index, size=len(df2[missi
ng5]),p=e.values)
```

Moving forward, with the final few null values (2, 4) in certain columns. I will be just dropping the rows that contain null values as it will not have a significant impact on the data.

In [49]:
```
#Simple line that drops all remaining null values.
df2.dropna(inplace=True)
```

In [50]:
```
df2.head(2)
```

Out[50]:

	host_response_rate	host_acceptance_rate	host_listings_count	host_total_listings_count	latitu
0	1.0	0.0	1.0	1.0	33.982
1	1.0	1.0	2.0	2.0	34.097

In [51]:
```
df2.isna().sum()
```

Out[51]:
```
host_response_rate                              0
host_acceptance_rate                            0
host_listings_count                             0
host_total_listings_count                       0
latitude                                        0
longitude                                       0
accommodates                                    0
bedrooms                                        0
beds                                            0
price                                           0
minimum_nights                                  0
maximum_nights                                  0
minimum_minimum_nights                          0
maximum_minimum_nights                          0
minimum_maximum_nights                          0
maximum_maximum_nights                          0
minimum_nights_avg_ntm                          0
maximum_nights_avg_ntm                          0
availability_30                                 0
availability_60                                 0
availability_90                                 0
availability_365                                0
number_of_reviews                               0
number_of_reviews_ltm                           0
number_of_reviews_l30d                          0
review_scores_rating                            0
review_scores_accuracy                          0
review_scores_cleanliness                       0
review_scores_checkin                           0
review_scores_communication                     0
review_scores_location                          0
review_scores_value                             0
calculated_host_listings_count                  0
calculated_host_listings_count_entire_homes     0
calculated_host_listings_count_private_rooms    0
calculated_host_listings_count_shared_rooms     0
reviews_per_month                               0
instant_bookable                                0
has_availability                                0
host_identity_verified                          0
host_has_profile_pic                            0
host_is_superhost                               0
host_response_time_a few days or more           0
host_response_time_within a day                 0
host_response_time_within a few hours           0
host_response_time_within an hour               0
room_type_Entire home/apt                       0
room_type_Hotel room                            0
room_type_Private room                          0
room_type_Shared room                           0
dtype: int64
```

In [52]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 31465 entries, 0 to 31470
Data columns (total 50 columns):
host_response_rate                             31465 non-null float64
host_acceptance_rate                           31465 non-null float64
host_listings_count                            31465 non-null float64
host_total_listings_count                      31465 non-null float64
latitude                                       31465 non-null float64
longitude                                      31465 non-null float64
accommodates                                   31465 non-null int64
bedrooms                                       31465 non-null float64
beds                                           31465 non-null float64
price                                          31465 non-null float64
minimum_nights                                 31465 non-null int64
maximum_nights                                 31465 non-null int64
minimum_minimum_nights                         31465 non-null float64
maximum_minimum_nights                         31465 non-null float64
minimum_maximum_nights                         31465 non-null float64
maximum_maximum_nights                         31465 non-null float64
minimum_nights_avg_ntm                         31465 non-null float64
maximum_nights_avg_ntm                         31465 non-null float64
availability_30                                31465 non-null int64
availability_60                                31465 non-null int64
availability_90                                31465 non-null int64
availability_365                               31465 non-null int64
number_of_reviews                              31465 non-null int64
number_of_reviews_ltm                          31465 non-null int64
number_of_reviews_l30d                         31465 non-null int64
review_scores_rating                           31465 non-null float64
review_scores_accuracy                         31465 non-null float64
review_scores_cleanliness                      31465 non-null float64
review_scores_checkin                          31465 non-null float64
review_scores_communication                    31465 non-null float64
review_scores_location                         31465 non-null float64
review_scores_value                            31465 non-null float64
calculated_host_listings_count                 31465 non-null int64
calculated_host_listings_count_entire_homes    31465 non-null int64
calculated_host_listings_count_private_rooms   31465 non-null int64
calculated_host_listings_count_shared_rooms    31465 non-null int64
reviews_per_month                              31465 non-null float64
instant_bookable                               31465 non-null int64
has_availability                               31465 non-null int64
host_identity_verified                         31465 non-null float64
host_has_profile_pic                           31465 non-null float64
host_is_superhost                              31465 non-null float64
host_response_time_a few days or more          31465 non-null uint8
host_response_time_within a day                31465 non-null uint8
host_response_time_within a few hours          31465 non-null uint8
host_response_time_within an hour              31465 non-null uint8
room_type_Entire home/apt                      31465 non-null uint8
room_type_Hotel room                           31465 non-null uint8
room_type_Private room                         31465 non-null uint8
room_type_Shared room                          31465 non-null uint8
dtypes: float64(26), int64(16), uint8(8)
memory usage: 10.6 MB
```

Here I will be classifying each of our classes as a number. There are three classes that hint to ternary-class models: and these classes are binned based on rating performance. I have labeled the classes 0 (subpar), 1 (great), 2(best) based on the ranges (0-94], (94-99], (99-100], respectively.

The process I took when experimenting with unique ranges.

Old:

df_best = df.loc[(df['review_scores_rating'] == 100)]

df_great = df.loc[(df['review_scores_rating'] < 100) & (df['review_scores_rating'] >= 95)]

df_good = df.loc[(df['review_scores_rating'] >= 80) & (df['review_scores_rating'] < 95)]

df_bad = df.loc[(df['review_scores_rating'] < 80)]

New:

df_best = df.loc[(df['review_scores_rating'] == 100)]

df_great = df.loc[(df['review_scores_rating'] < 100) & (df['review_scores_rating'] >= 95)]

df_subpar = df.loc[(df['review_scores_rating'] < 95)]

Descriptions:

Best: The best score of 100

Great: 95-100, contains the median, mode

Supbar: Lower than the mean of 95

In [53]:
```
df2['target'] = pd.cut(df2['review_scores_rating'], [0, 95, 99, 100], ri
ght=True, \
                       labels=['0', '1', '2'])
```

In [54]:
```
df2.drop('review_scores_rating', axis=1, inplace=True)
```

In [55]:
```
df2['target'].value_counts()
```

Out[55]:
```
0    11662
1    10479
2     9324
Name: target, dtype: int64
```

In [130]:
```
df2['target'].value_counts(normalize=True)
```

Out[130]:
```
0    0.370931
1    0.334184
2    0.294884
Name: target, dtype: float64
```

As you can see, my cutoffs for each class are not only suitable ranges, but also address class imbalance. I made these cutoff decisions based on measures of central tendency where the 'best'(2) class has only perfect scores, the 'great'(1) class has scores greater than the mean up to, but not including, the best (median and mode) and the 'subpar'(0) class contains all values lower than the mean.

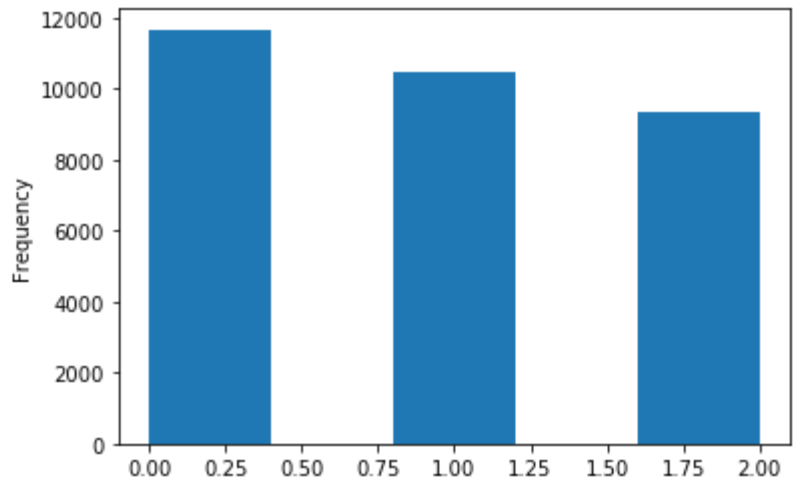
Some additional explanation: After conducting side research on airbnb ratings, all values lower than the mean potentially represent ratings where customers had felt their experience was not the best and could have complaints/suggestions to make.

Class imbalance: For future datasets, my solution to class imbalance will not particularly be the same as it is here. I would definitely implement SMOTE (which creates artificial data in underrepresented classes).

In [56]:
```
#Binned using .cut; however, still category type. Converting to numeric
df2['target'] = pd.to_numeric(df2['target'], errors='coerce')
```

In [57]:
```
#Visualization of target class distribution.
df2['target'].plot.hist(bins=5)
```

Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e7e71d0>



In [58]:
```
df2.head(2)
```

Out[58]:

	host_response_rate	host_acceptance_rate	host_listings_count	host_total_listings_count	latitu
0	1.0	0.0	1.0	1.0	33.982
1	1.0	1.0	2.0	2.0	34.097

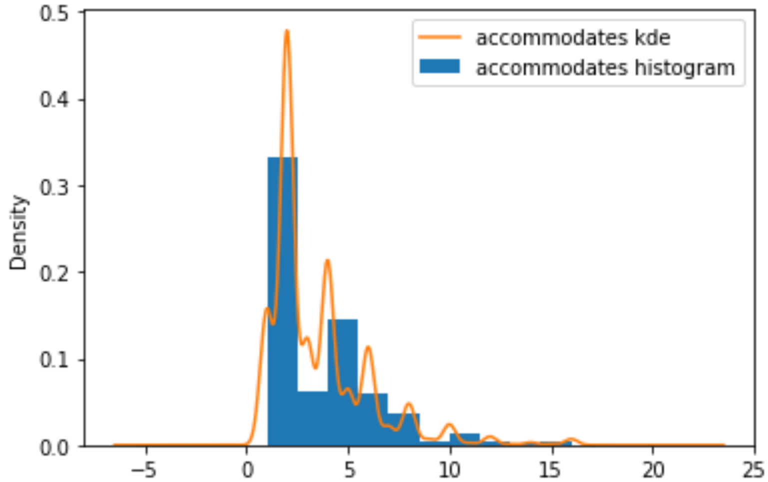
In [59]:
```
#Dropping unnecessary or redundant/repititve columns
df2.drop(['host_listings_count', 'host_total_listings_count', 'minimum_m
inimum_nights', 'maximum_minimum_nights', \
        'minimum_maximum_nights', 'maximum_maximum_nights', 'minimum_nig
hts_avg_ntm', 'maximum_nights_avg_ntm', \
        'number_of_reviews_ltm', 'number_of_reviews_l30d', 'calculated_h
ost_listings_count', \
        'calculated_host_listings_count_entire_homes', 'calculated_host_
listings_count_private_rooms', \
        'calculated_host_listings_count_shared_rooms'], axis=1, inplace=
True)
```

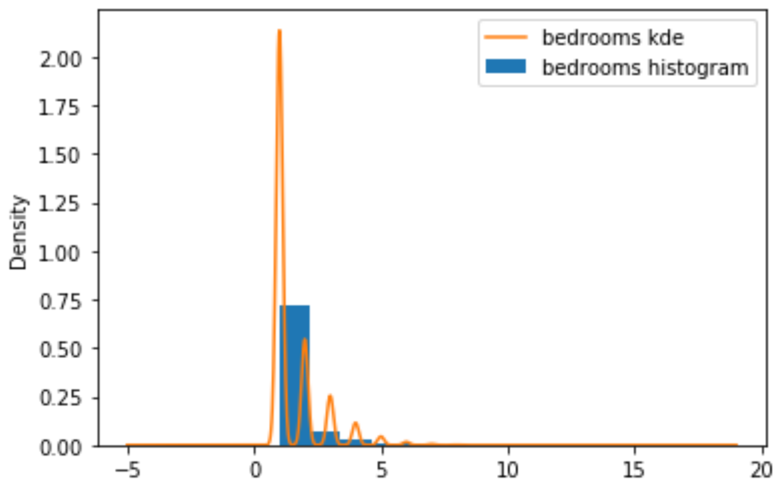
In [60]:
```
#Outlier features checks: ACCOMODATES, BEDROOMS, BEDS, MINIMUM_NIGHTS(re
move single values)
df2['beds'].value_counts()
```

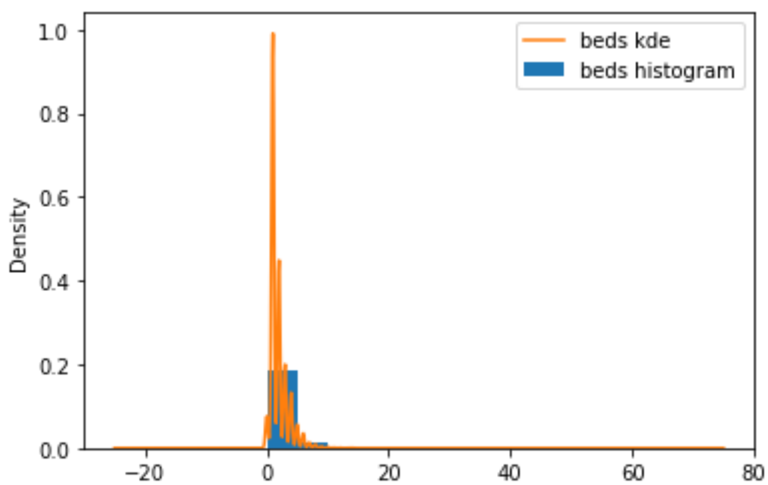
Out[60]:
```
1.0     15800
2.0      7160
3.0      3191
4.0      2121
0.0      1210
5.0       895
6.0       561
7.0       211
8.0       144
9.0        66
10.0       42
12.0       16
11.0       16
14.0       11
16.0        4
22.0        3
13.0        3
19.0        2
15.0        2
32.0        1
50.0        1
44.0        1
18.0        1
20.0        1
21.0        1
23.0        1
Name: beds, dtype: int64
```

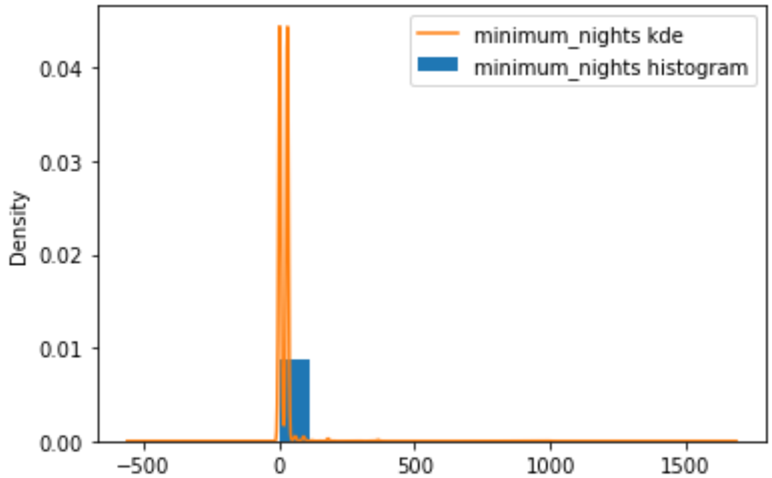
In [61]:
```
#Create a dataframe with non binary/discrete values to look at boxplot d
istribution and remove any outliers.
non_bin = df2[['accommodates', 'bedrooms', 'beds', 'minimum_nights', 'ma
ximum_nights', 'number_of_reviews', \
              'reviews_per_month']]
```

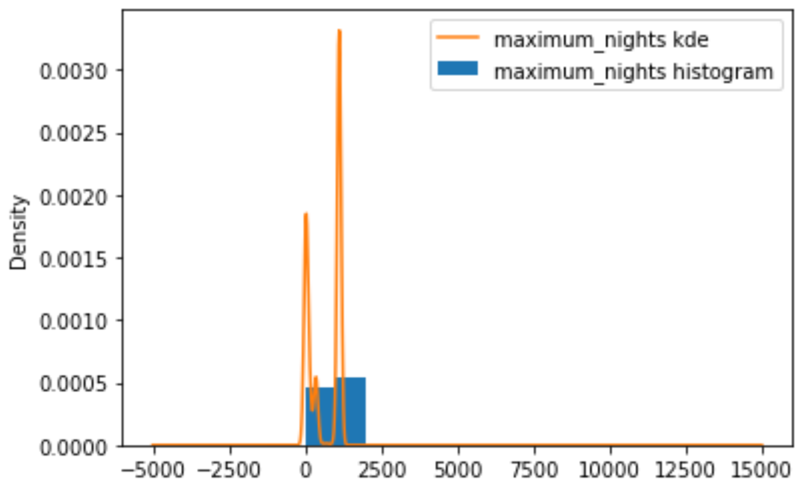
In [62]:
```
for column in non_bin:
    non_bin[column].plot.hist(density=True, label= column+' histogram')
    non_bin[column].plot.kde(label=column+' kde')
    plt.legend()
    plt.show()
```

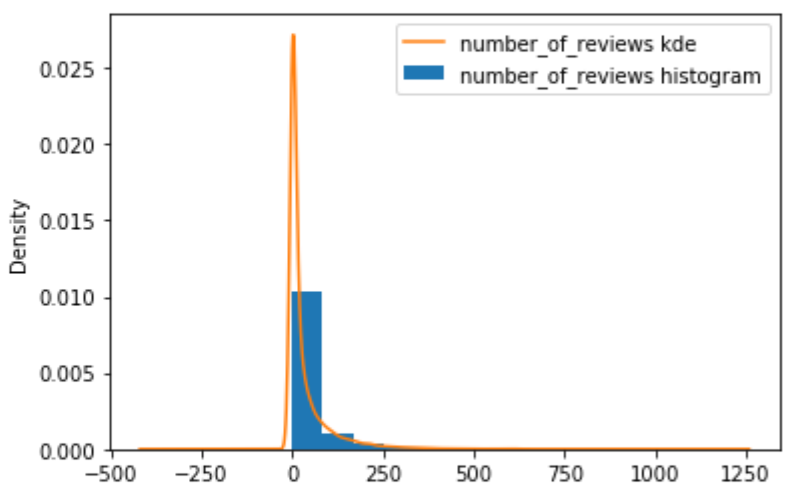

accommodates kde
accommodates histogram
Density
0.5
0.4
0.3
0.2
0.1
0.0
−5
0
5
10
15
20
25

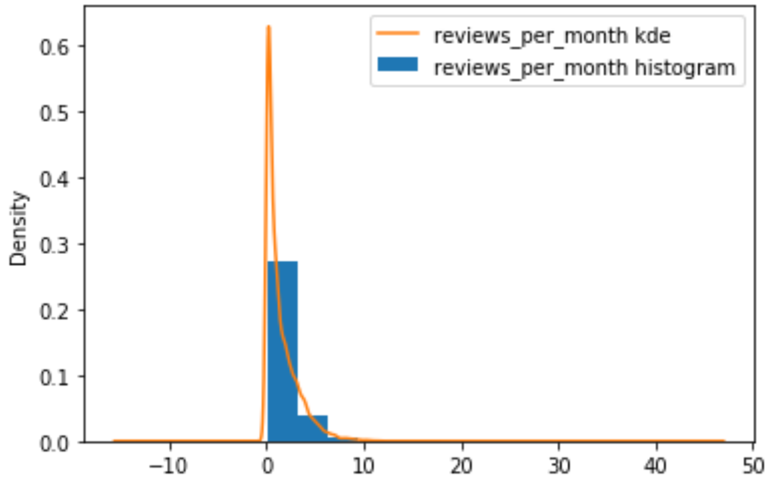

bedrooms kde
bedrooms histogram
Density
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−5
0
5
10
15
20


beds kde
beds histogram
Density
1.0
0.8
0.6
0.4
0.2
0.0
−20
0
20
40
60
80

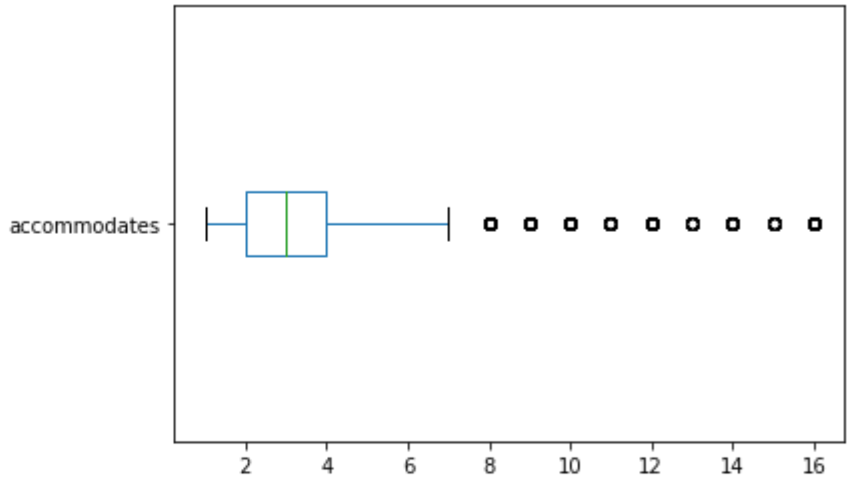

minimum_nights kde
minimum_nights histogram
Density
0.04
0.03
0.02
0.01
0.00
−500
0
500
1000
1500

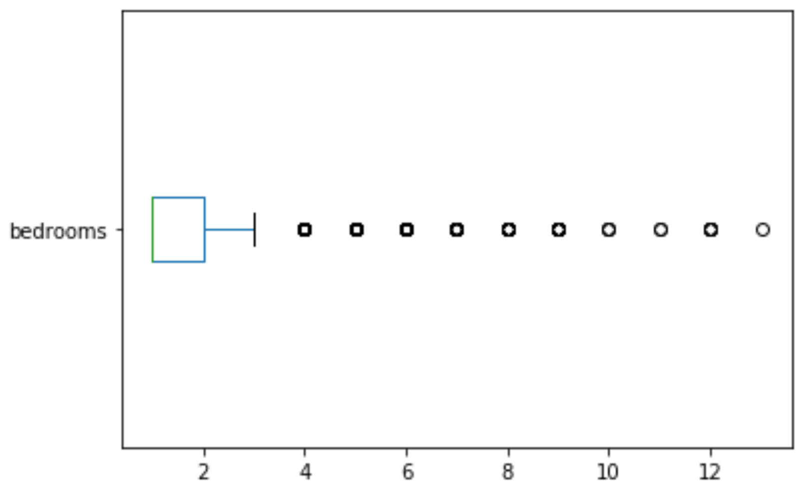

maximum_nights kde
maximum_nights histogram
Density
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
−5000
−2500
0
2500
5000
7500
10000
12500
15000

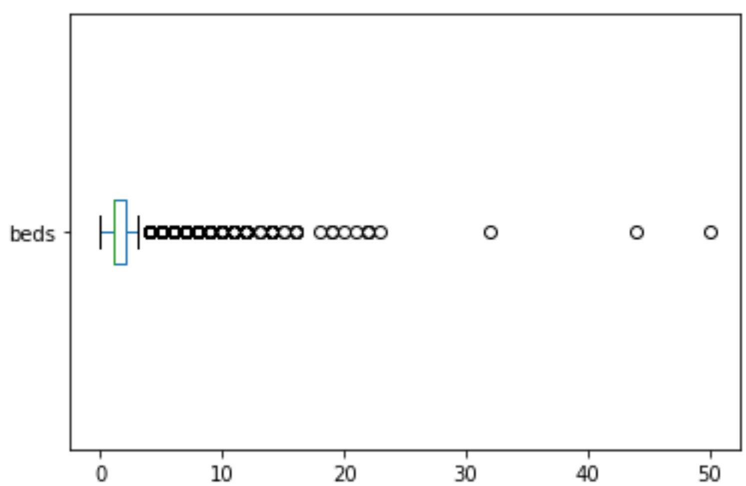

number_of_reviews kde
number_of_reviews histogram
Density
0.025
0.020
0.015
0.010
0.005
0.000
−500
−250
0
250
500
750
1000
1250

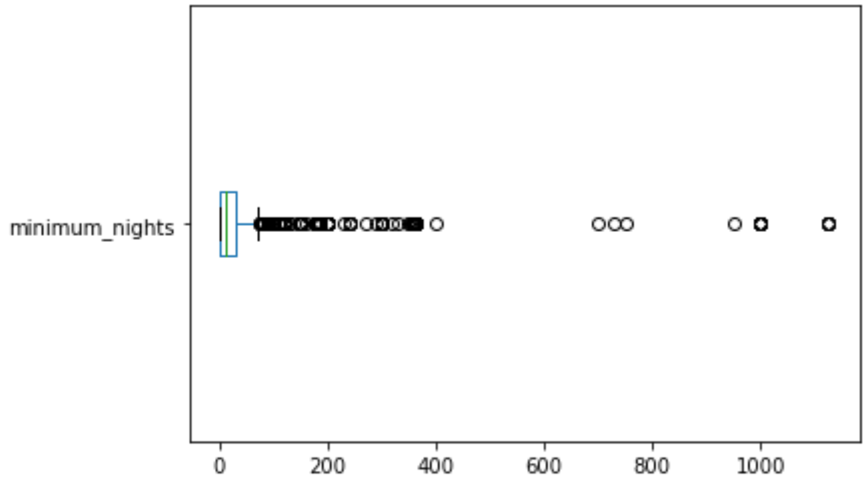

reviews_per_month kde
reviews_per_month histogram
Density
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10
0
10
20
30
40
50

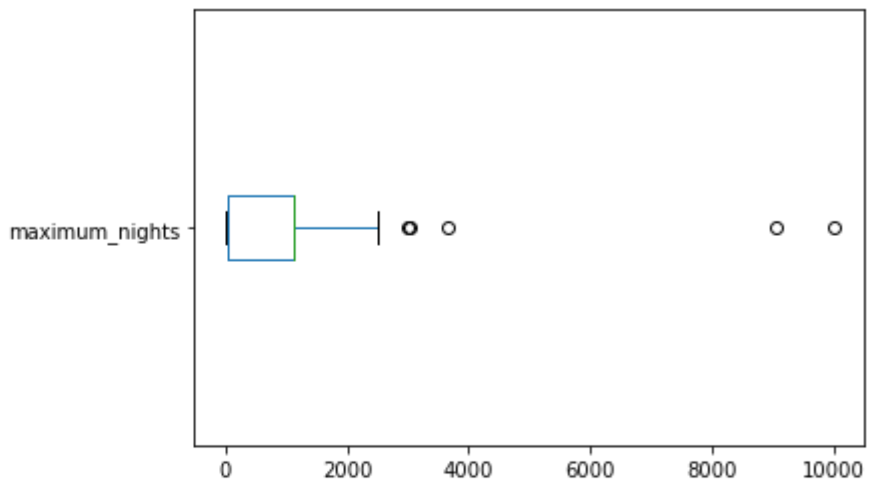
In [63]:
```
for column in non_bin:
    non_bin[column].plot.box(vert=False)
    plt.show()
```

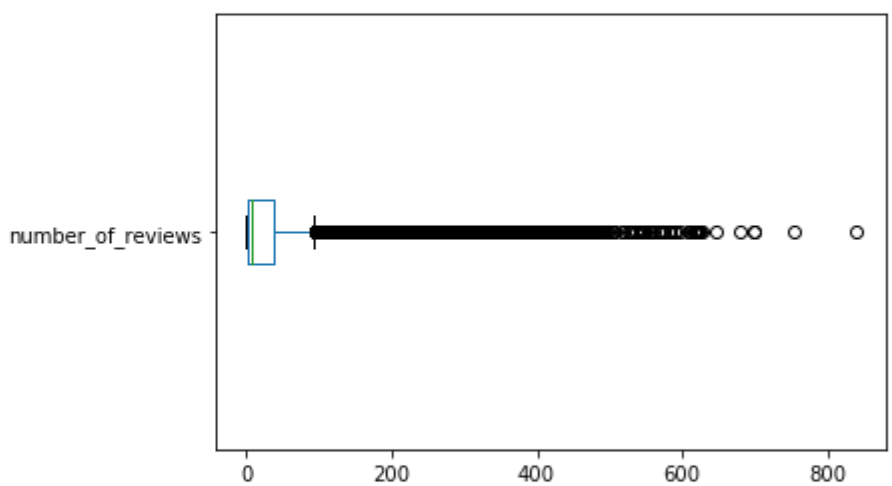

accommodates
2
4
6
8
10
12
14
16

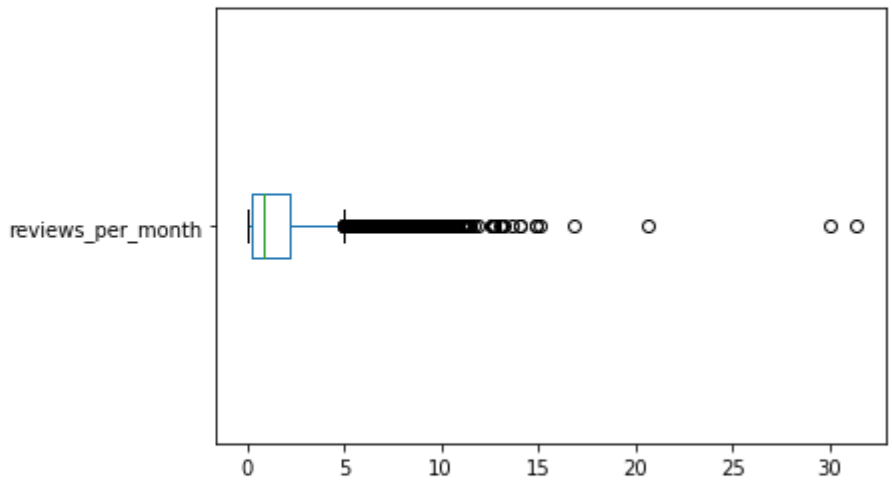

bedrooms
2
4
6
8
10
12


beds
0
10
20
30
40
50


minimum_nights
0
200
400
600
800
1000

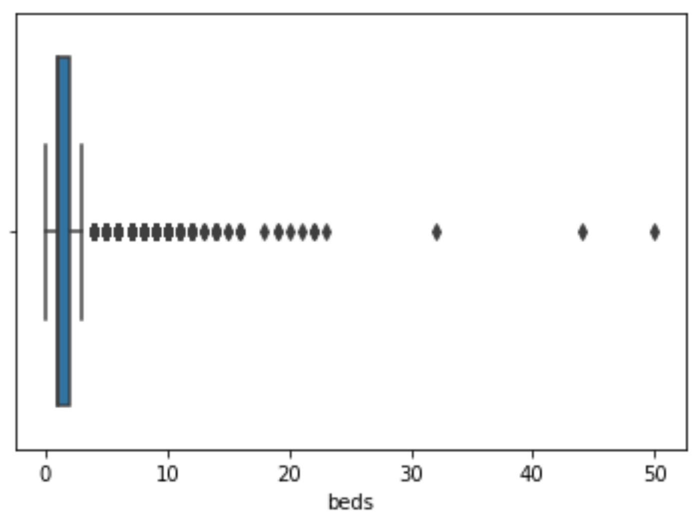

maximum_nights
0
2000
4000
6000
8000
10000


number_of_reviews
0
200
400
600
800



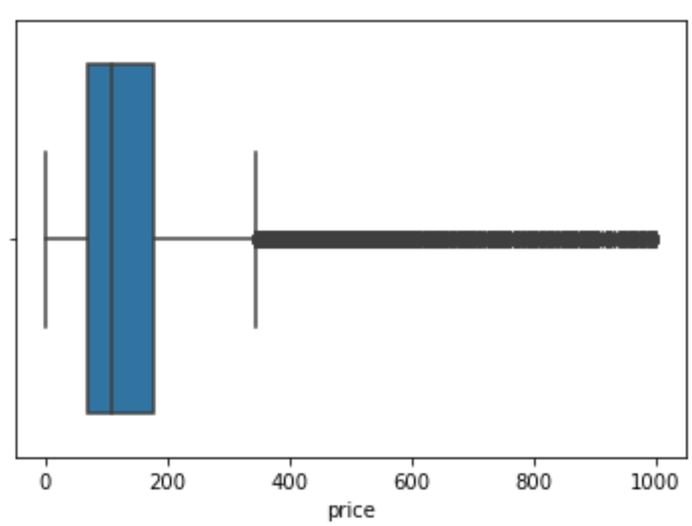
In [64]:
```
sns.boxplot(x=df2['beds'])
```

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2e2c25f8>



In [65]:
```
sns.boxplot(x=df2['price'])
```

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2fcd7a58>



Removing outliers from each column based on visual boxplots:

In [66]:
```
df2 = df2.loc[(df2['beds'] < 10)]
```

In [67]:
```
df2 = df2.loc[(df2['minimum_nights'] < 80)]
```

In [68]:
```
df2 = df2.loc[(df2['accommodates'] < 12)]
```

In [69]:
```
df2 = df2.loc[(df2['bedrooms'] < 10)]
```

In [70]:
```
df2 = df2.loc[(df2['maximum_nights'] < 5000)]
```

In [71]:
```
df2 = df2.loc[(df2['number_of_reviews'] < 600)]
```

In [72]:
```
df2 = df2.loc[(df2['reviews_per_month'] < 15)]
```

In [73]:
```
df2 = df2.loc[(df2['price'] < 600)]
```

EDA

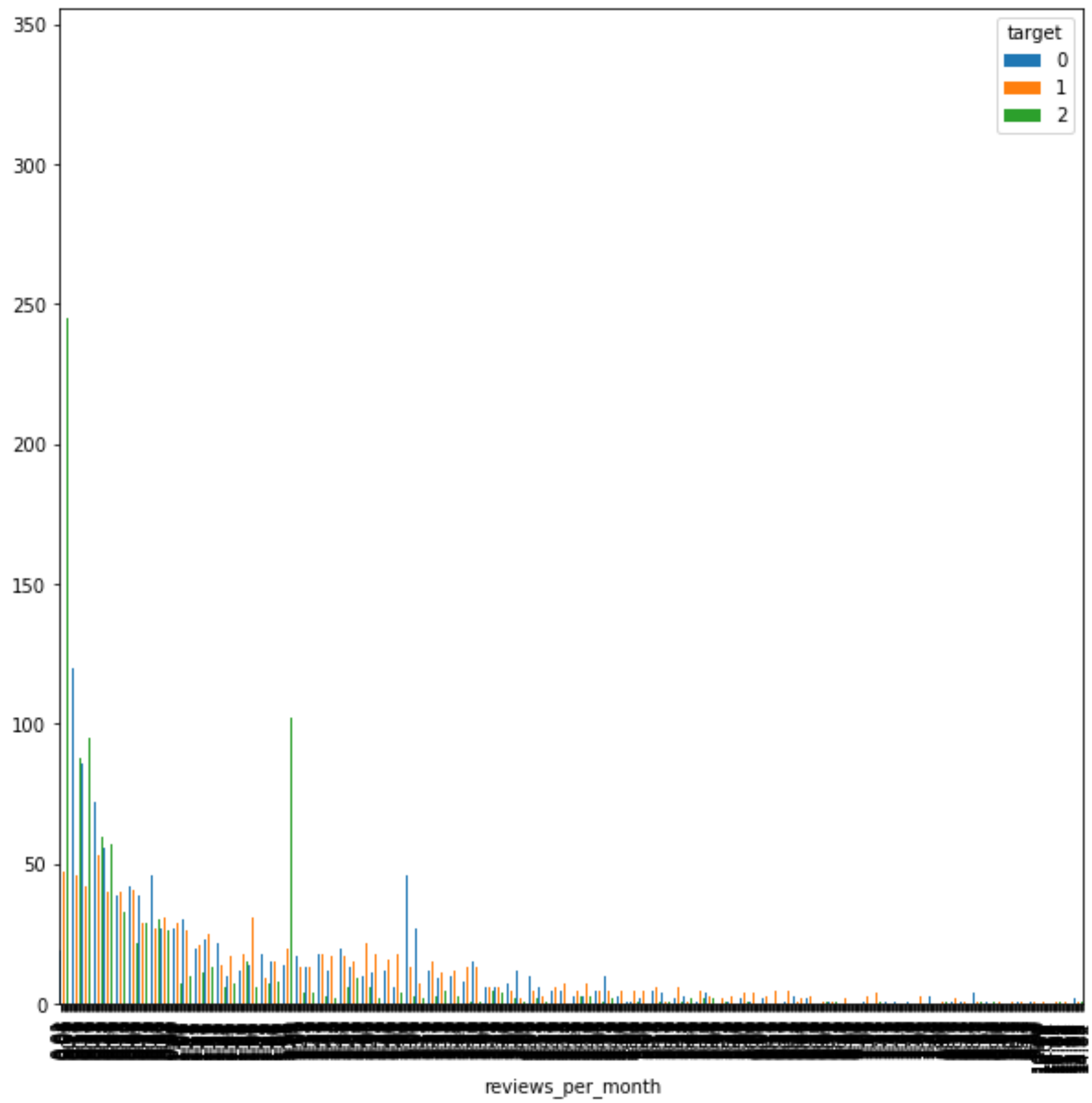
Exploratory Data Analysis (EDA) here consists of creating visualizations in order to understand and summarize key characteristics of my data. I will mostly be illustrating visuals on what my features/columns look across my three classes. Specifically, looking for variance/seperability within classes so the model can make better predictions and can theoretically distinguish the classes better.

In [75]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29771 entries, 0 to 31470
Data columns (total 36 columns):
host_response_rate                         29771 non-null float64
host_acceptance_rate                       29771 non-null float64
latitude                                   29771 non-null float64
longitude                                  29771 non-null float64
accommodates                               29771 non-null int64
bedrooms                                   29771 non-null float64
beds                                       29771 non-null float64
price                                      29771 non-null float64
minimum_nights                             29771 non-null int64
maximum_nights                             29771 non-null int64
availability_30                            29771 non-null int64
availability_60                            29771 non-null int64
availability_90                            29771 non-null int64
availability_365                           29771 non-null int64
number_of_reviews                          29771 non-null int64
review_scores_accuracy                     29771 non-null float64
review_scores_cleanliness                  29771 non-null float64
review_scores_checkin                      29771 non-null float64
review_scores_communication                29771 non-null float64
review_scores_location                     29771 non-null float64
review_scores_value                        29771 non-null float64
reviews_per_month                          29771 non-null float64
instant_bookable                           29771 non-null int64
has_availability                           29771 non-null int64
host_identity_verified                     29771 non-null float64
host_has_profile_pic                       29771 non-null float64
host_is_superhost                          29771 non-null float64
host_response_time_a few days or more      29771 non-null uint8
host_response_time_within a day            29771 non-null uint8
host_response_time_within a few hours      29771 non-null uint8
host_response_time_within an hour          29771 non-null uint8
room_type_Entire home/apt                  29771 non-null uint8
room_type_Hotel room                       29771 non-null uint8
room_type_Private room                     29771 non-null uint8
room_type_Shared room                      29771 non-null uint8
target                                     29771 non-null int64
dtypes: float64(17), int64(11), uint8(8)
memory usage: 6.8 MB
```

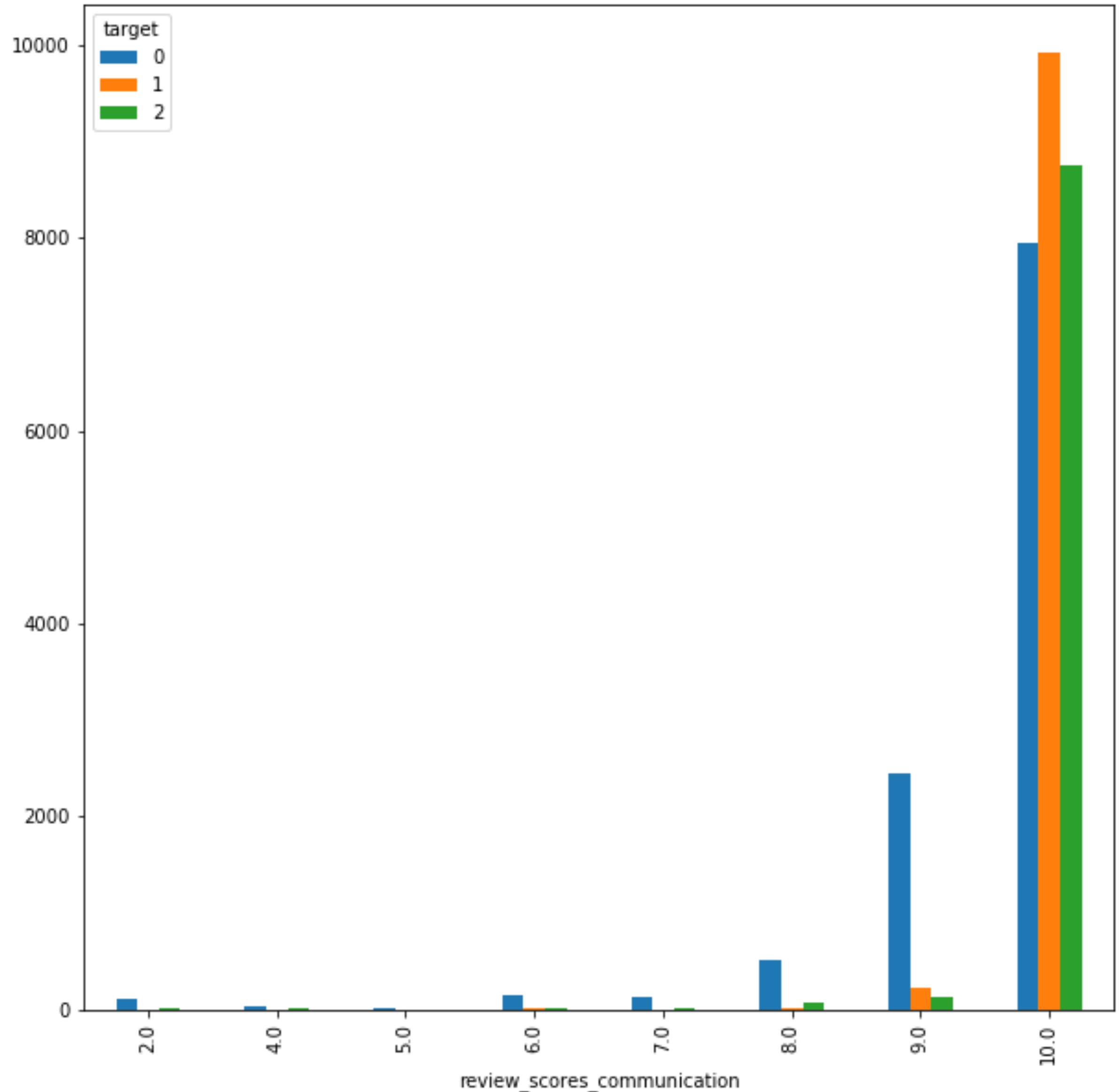
In [76]:
```
#The perfect class receives marginally more reviews than the other 2 classes.
pd.crosstab(df2['reviews_per_month'], df2['target']).plot.bar(figsize=(10,10))
```

Out[76]: <matplotlib.axes._subplots.AxesSubplot at 0x1a301be4e0>



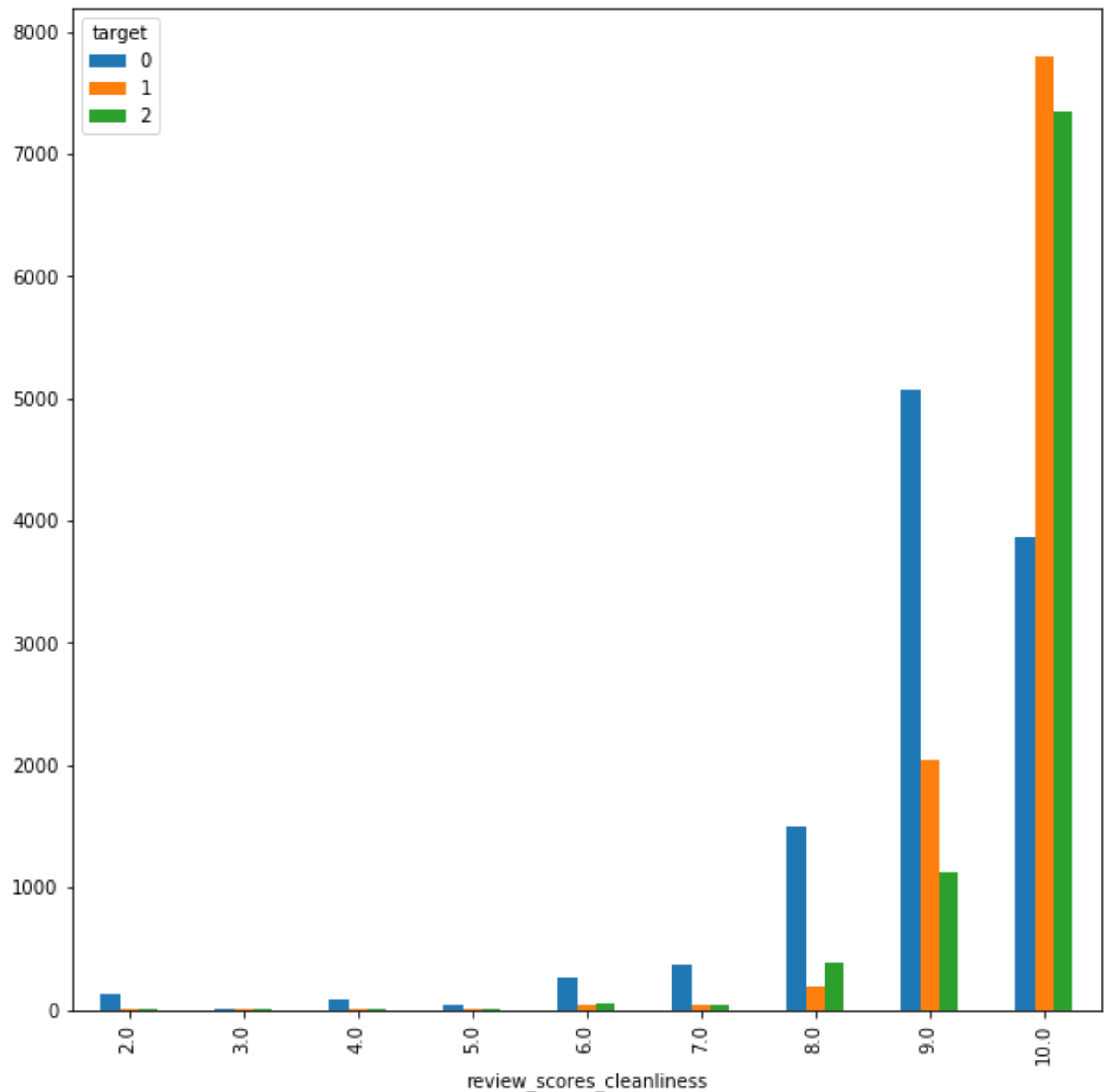
In [54]:
```
pd.crosstab(df2['review_scores_communication'], df2['target']).plot.bar(
figsize=(10,10))
#Noteable trend, for the subpar class, although it has some perfect scor
es, it has scores for every value down to 2 more
#prominent than the other, better classes.
#So, communication was not that great.
```

Out[54]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a22905ef0>
```



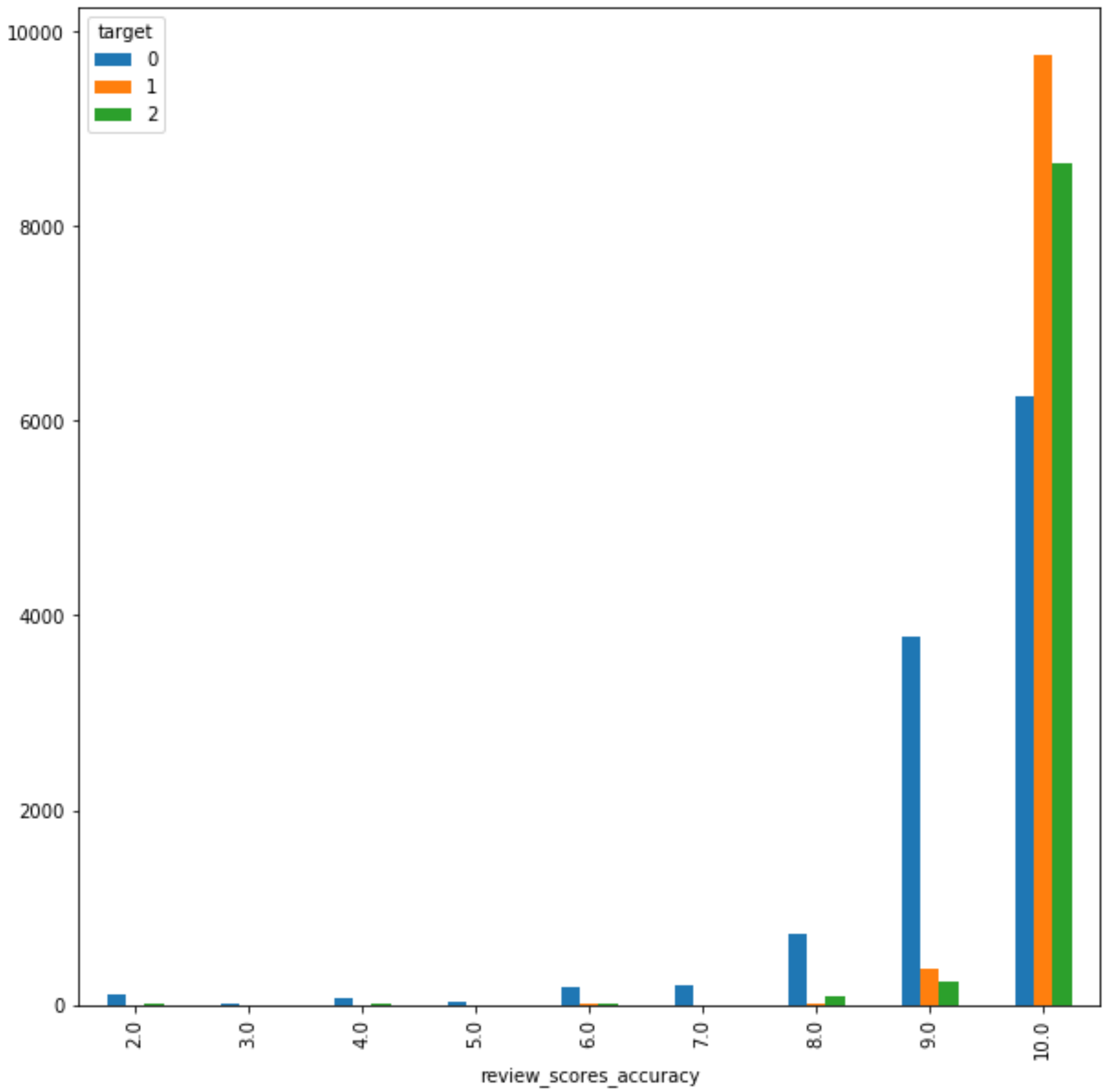
In [55]:
```
pd.crosstab(df2['review_scores_cleanliness'], df2['target']).plot.bar(figsize=(10,10))
#Similar trend, 1,2 have a lot more perfect scores than 0 and 0 is more prominent in the other scores down to 2.
```

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1d9e2898>



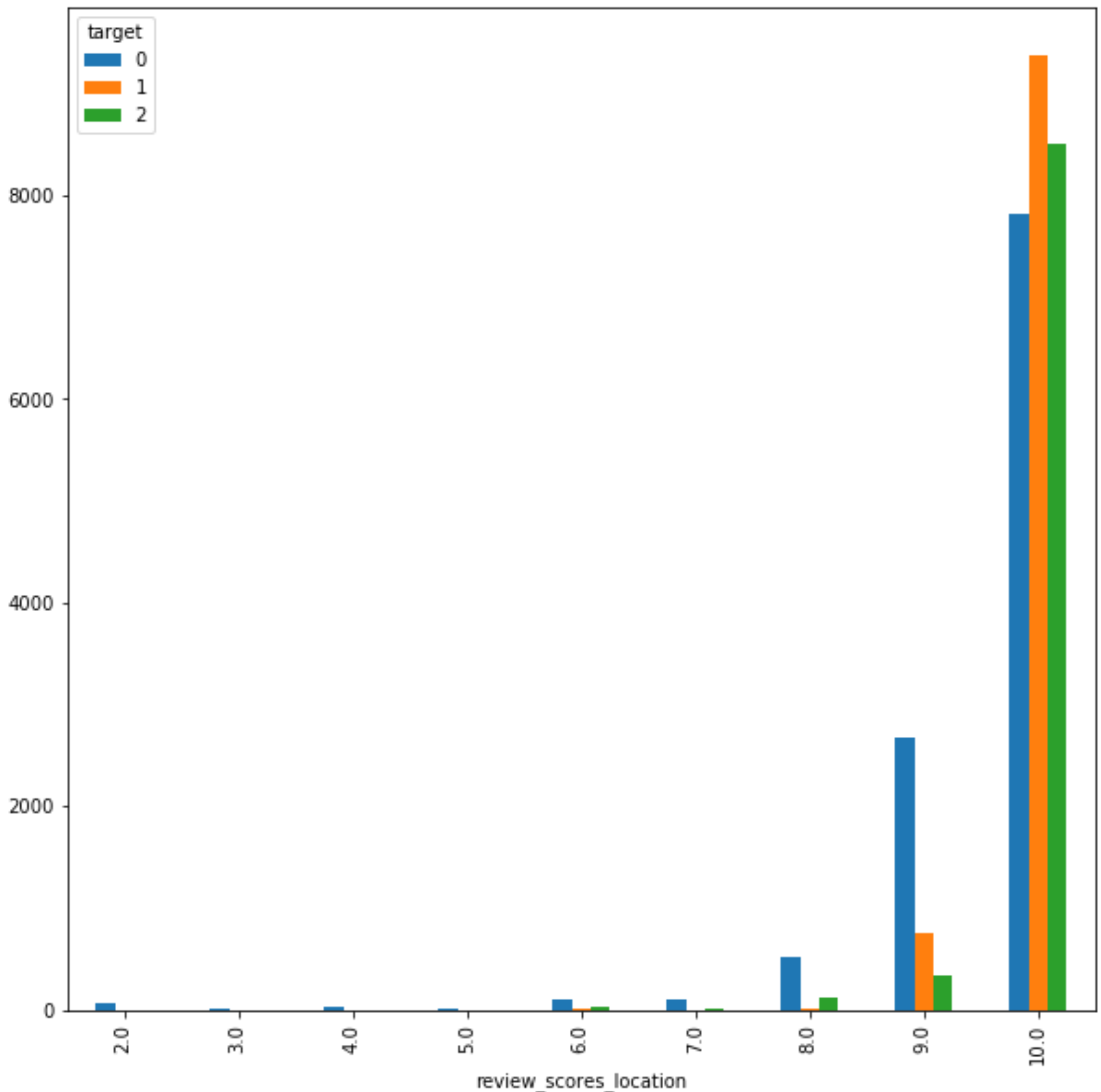
In [56]:
```
pd.crosstab(df2['review_scores_accuracy'], df2['target']).plot.bar(figsi
ze=(10,10))
```

Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x1a24122358>



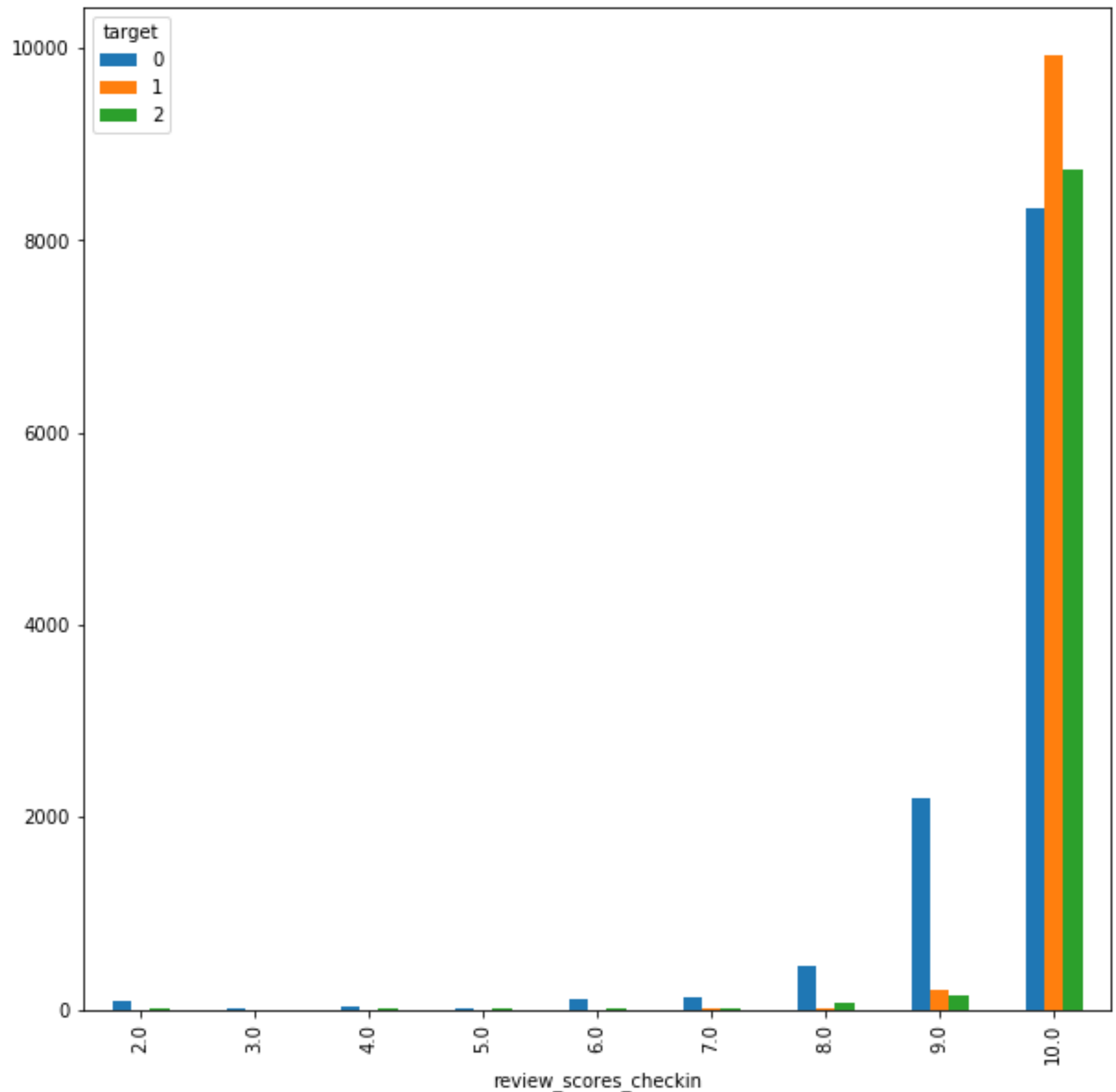
In [57]:
```
pd.crosstab(df2['review_scores_location'], df2['target']).plot.bar(figsize=(10,10))
```

Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x1a23be9240>



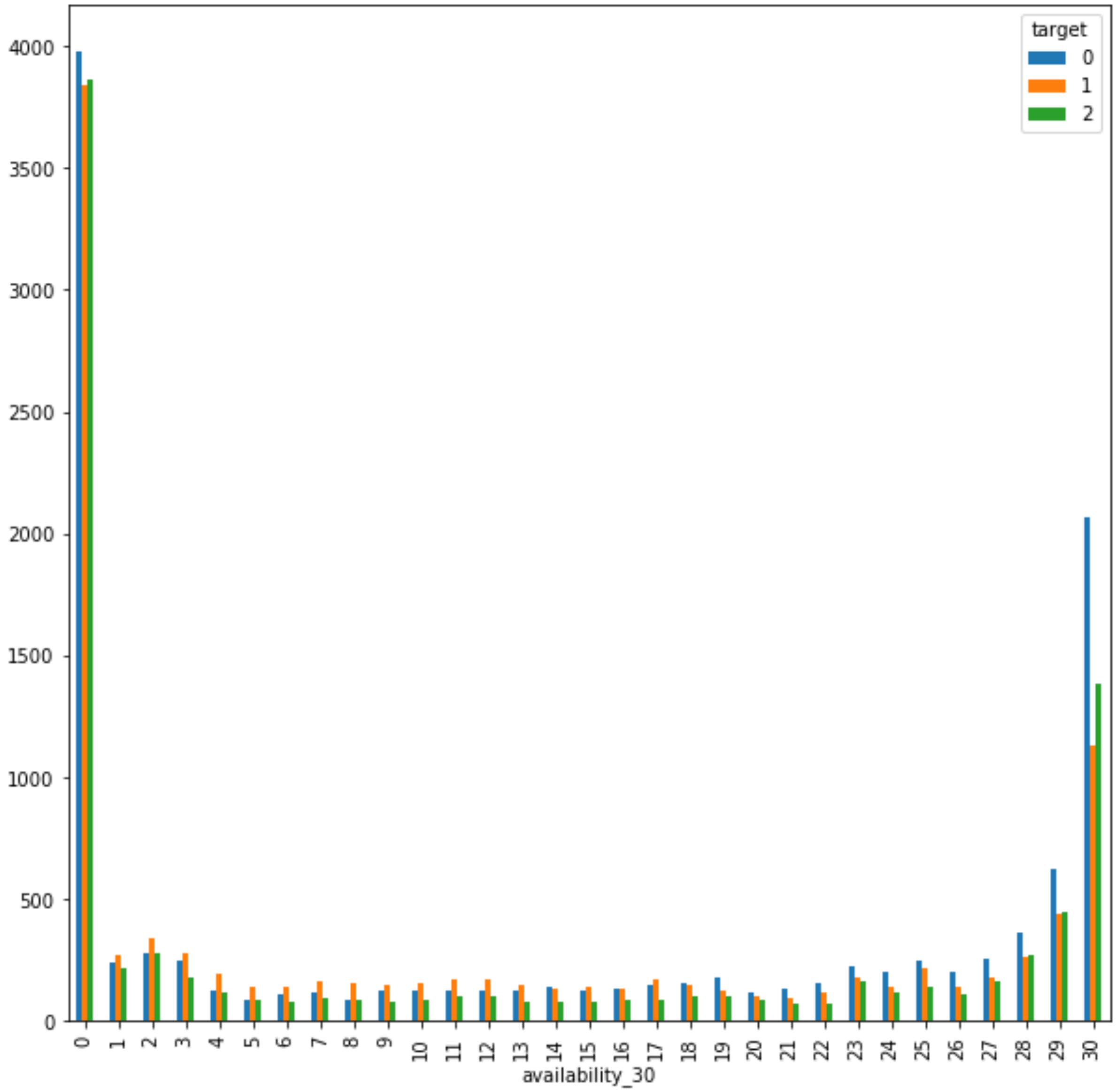
In [58]:
```
pd.crosstab(df2['review_scores_checkin'], df2['target']).plot.bar(figsiz
e=(10,10))
```

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x1a25bca7f0>



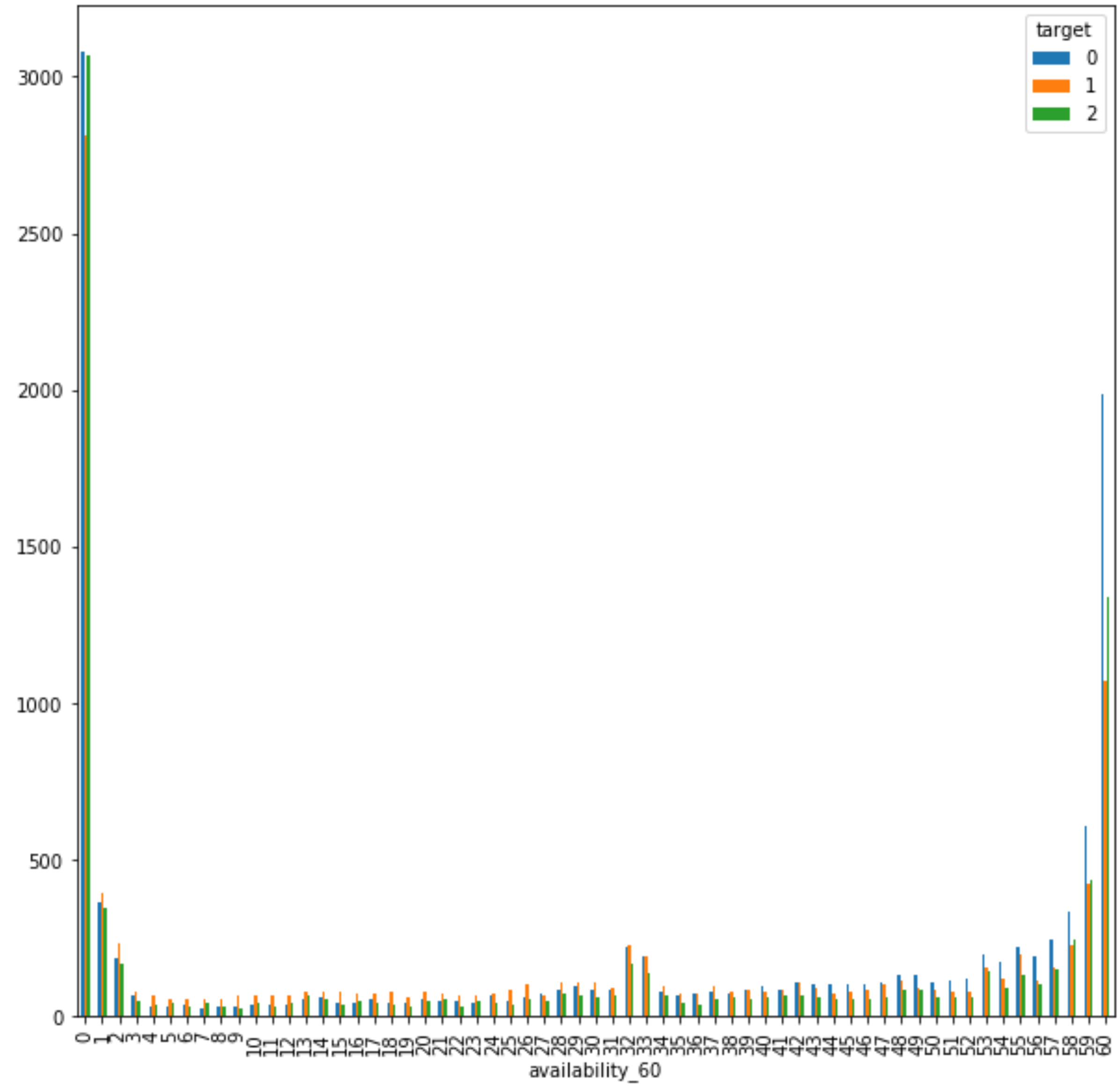
In [59]:
```
pd.crosstab(df2['availability_30'], df2['target']).plot.bar(figsize=(10,
10))
#As you can see within 0 days, essentially none of the classes are booke
d. But, as you progress up to 30, class 0 is
#more available and we can conclude that less people book the lower rate
d listings.
```

Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x1a24bf3b38>



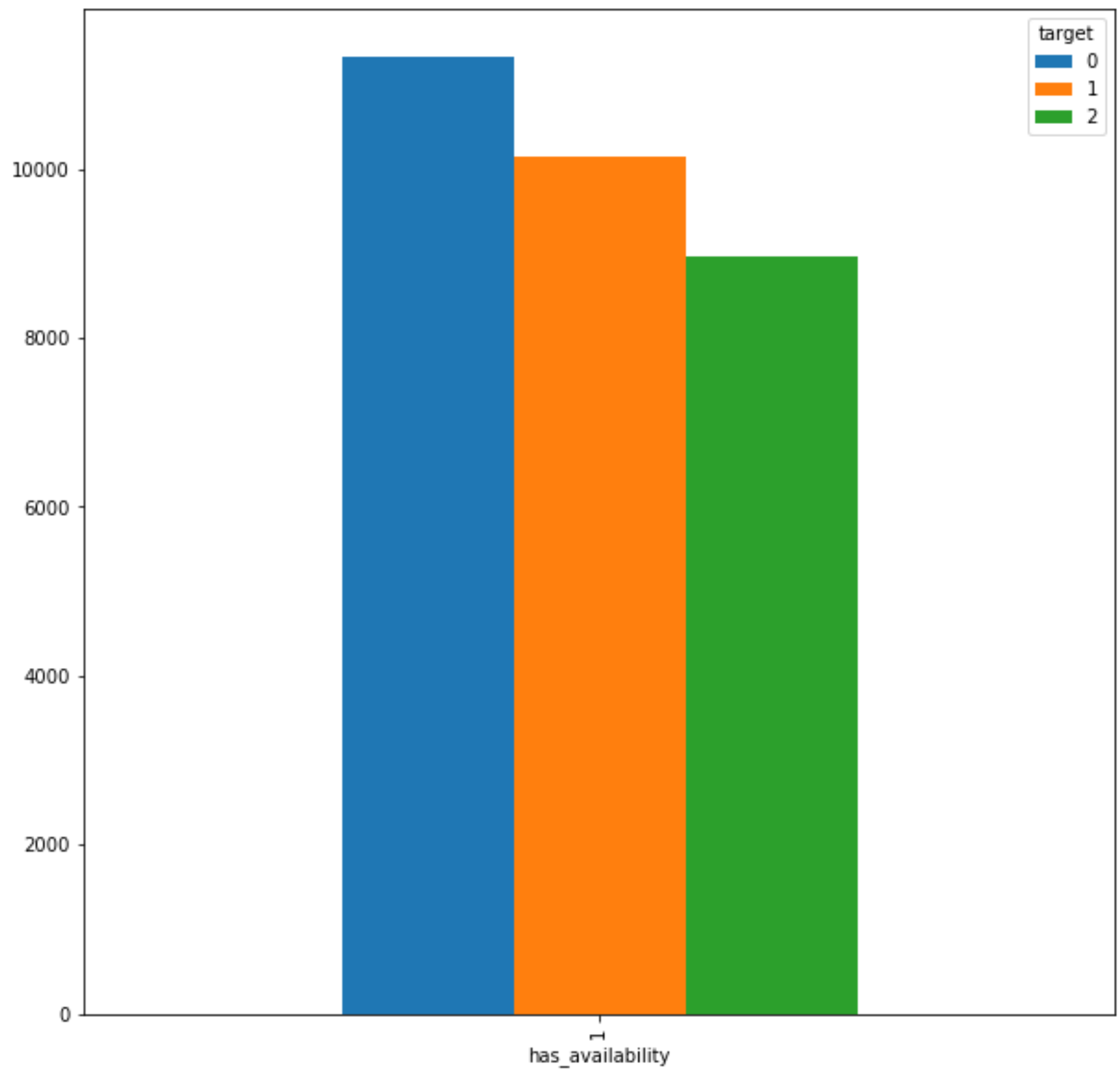
In [60]:
```
pd.crosstab(df2['availability_60'], df2['target']).plot.bar(figsize=(10,
10))
#Very similar trend to availability within 30 days
```

Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x1a24ba2b38>



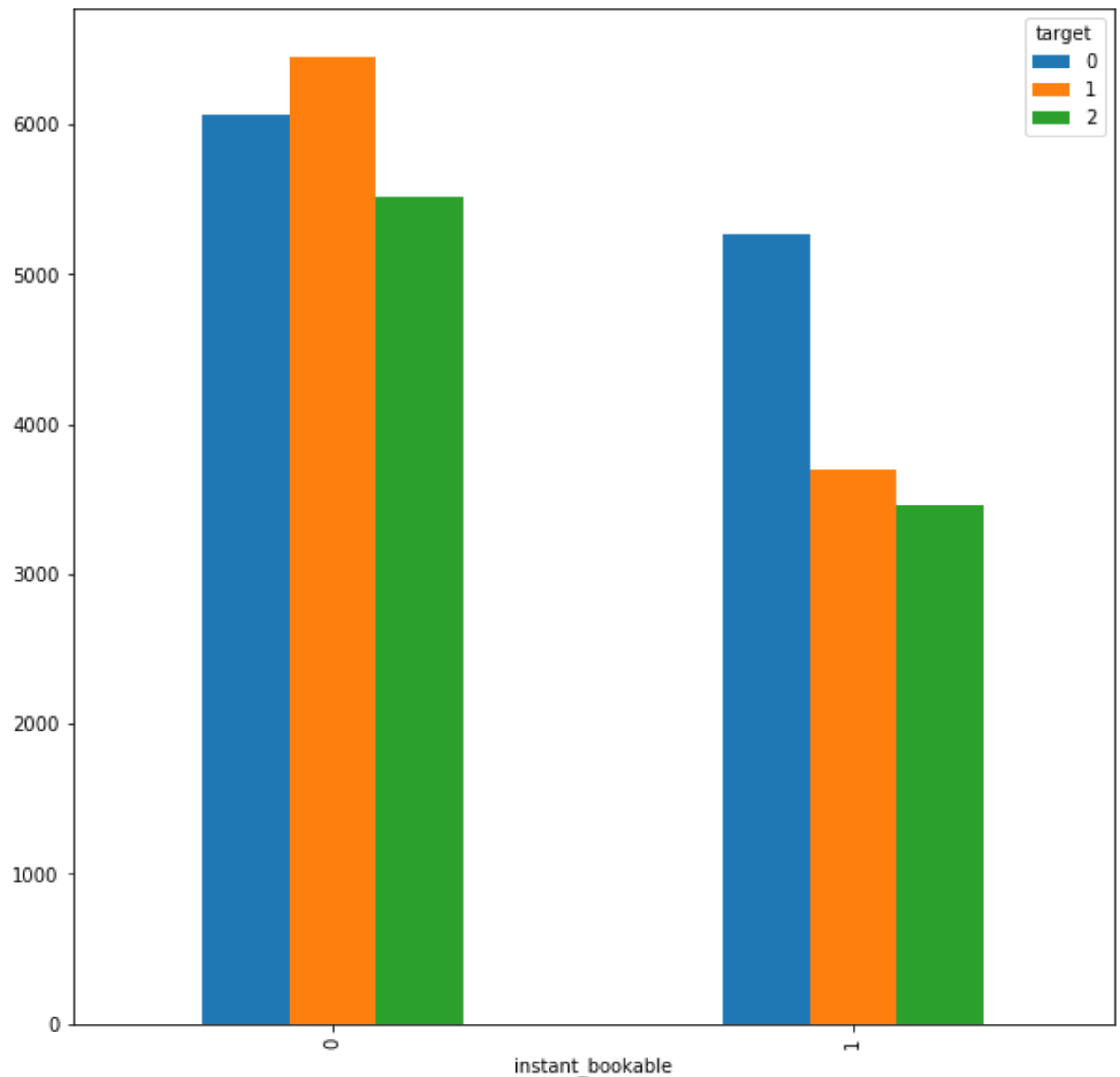
In [61]:
```
pd.crosstab(df2['has_availability'], df2['target']).plot.bar(figsize=(10
,10))
#This graph tells us the lower rating listings have more availability ve
rsus the better rating listings.
#Which intuitively makes perfect sense. The better listings will have lo
wer availability since they are being booked.
```

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2ac725c0>



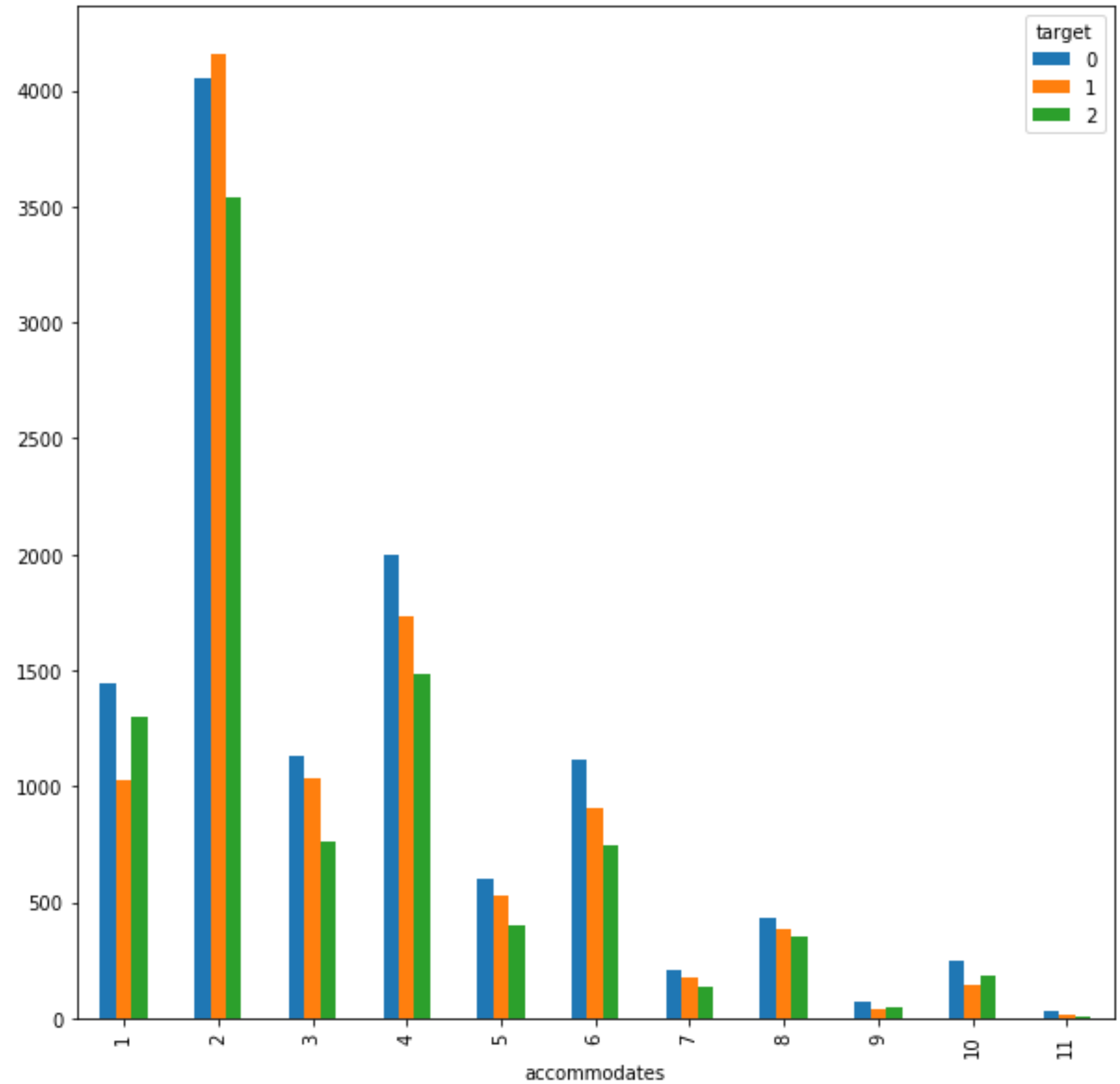
In [62]:
```
pd.crosstab(df2['instant_bookable'], df2['target']).plot.bar(figsize=(10
,10))
#The better rated listings are probably less "instantly bookable" becaus
e people plan ahead to reserve better listings
```

Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2bcda240>



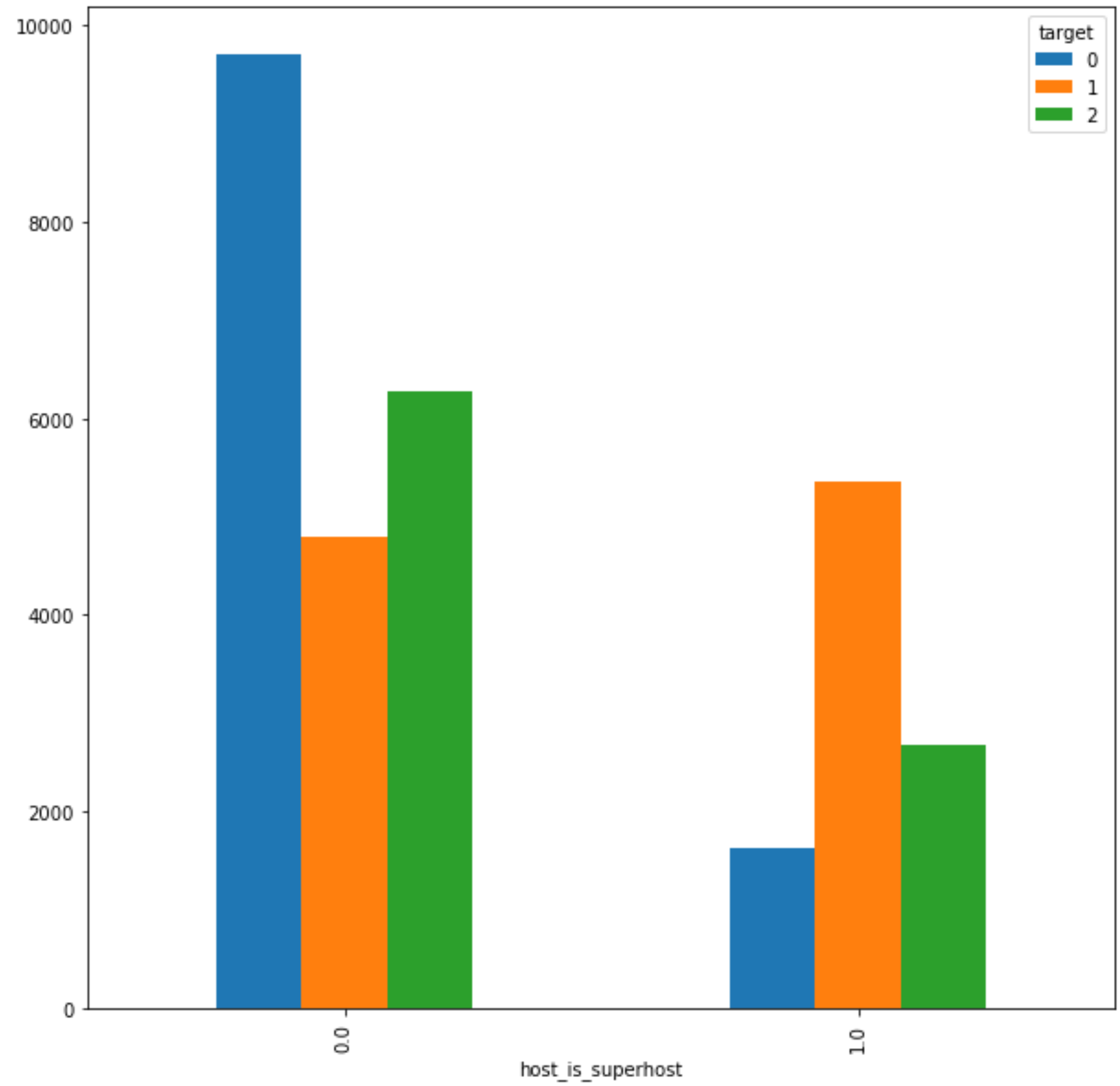
In [63]:
```
pd.crosstab(df2['accommodates'], df2['target']).plot.bar(figsize=(10,10
))
```

Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2bd12dd8>



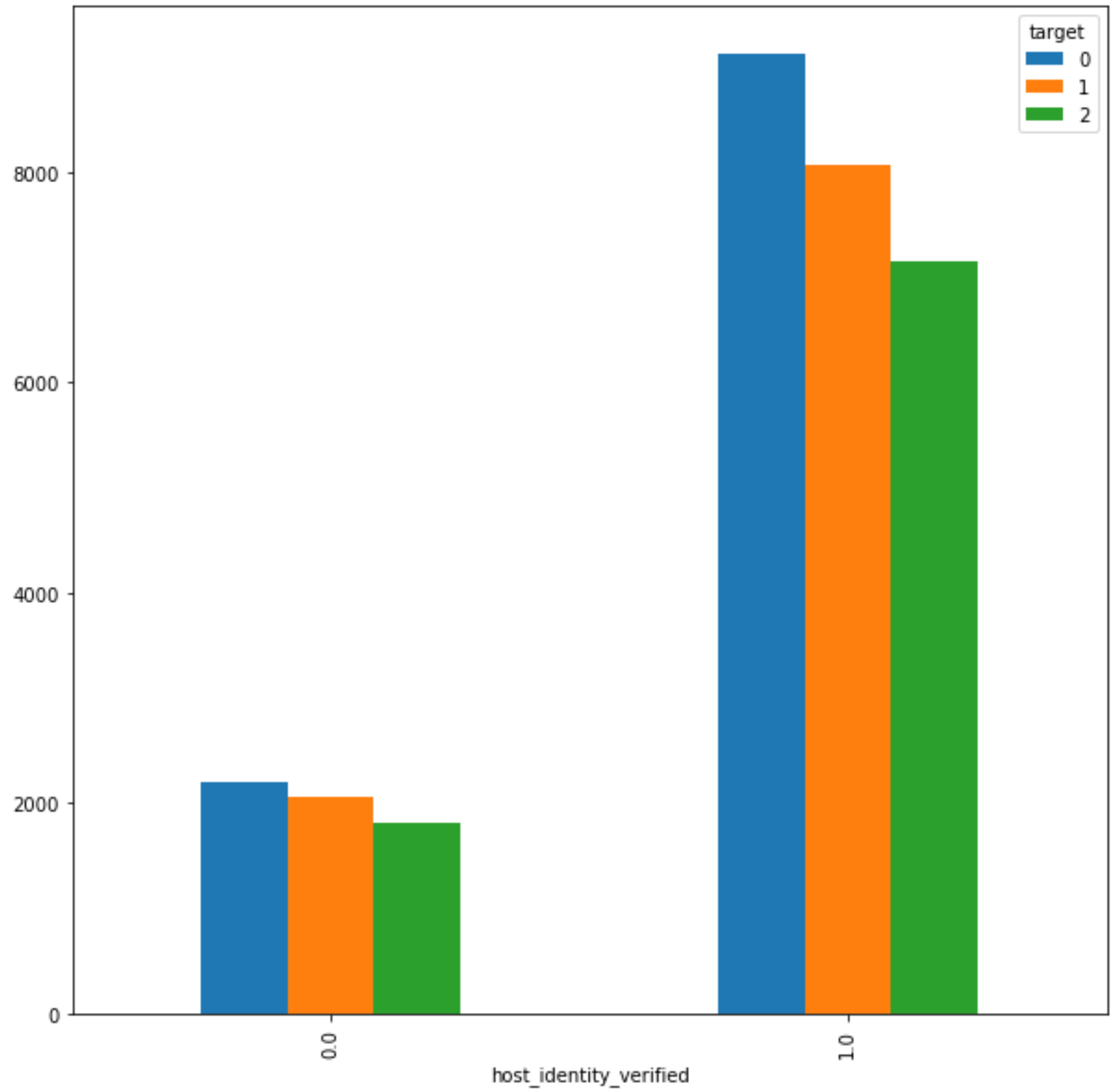
In [64]:
```
pd.crosstab(df2['host_is_superhost'], df2['target']).plot.bar(figsize=(1
0,10))
#Classes 1 and 2 have way more superhost status than class 0. A superhos
t is a verified host that provides excellent
#experiences and stays and is a "role model".
```

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2bd32d30>



In [65]:
```
pd.crosstab(df2['host_identity_verified'], df2['target']).plot.bar(figsi
ze=(10,10))
```

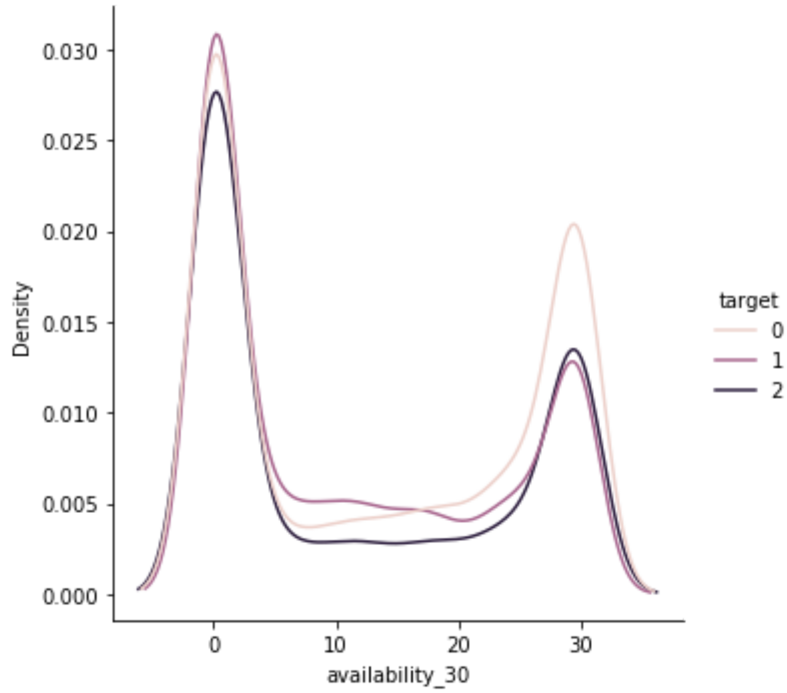
Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2bd32a90>



Exploring different types of graphs

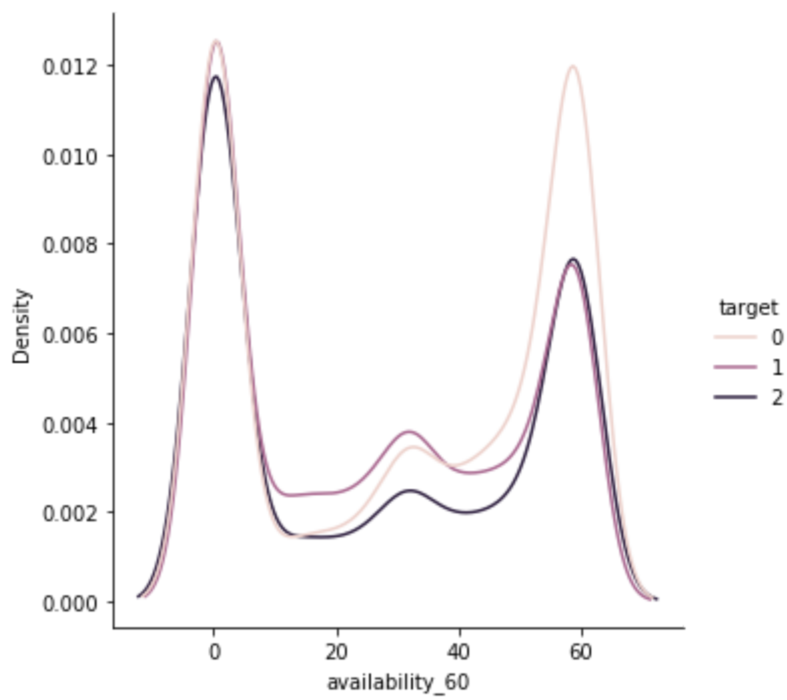
In [66]:
```
sns.displot(data=df2, x='availability_30', hue='target', kind='kde')
```

Out[66]: <seaborn.axisgrid.FacetGrid at 0x1a2c098e10>



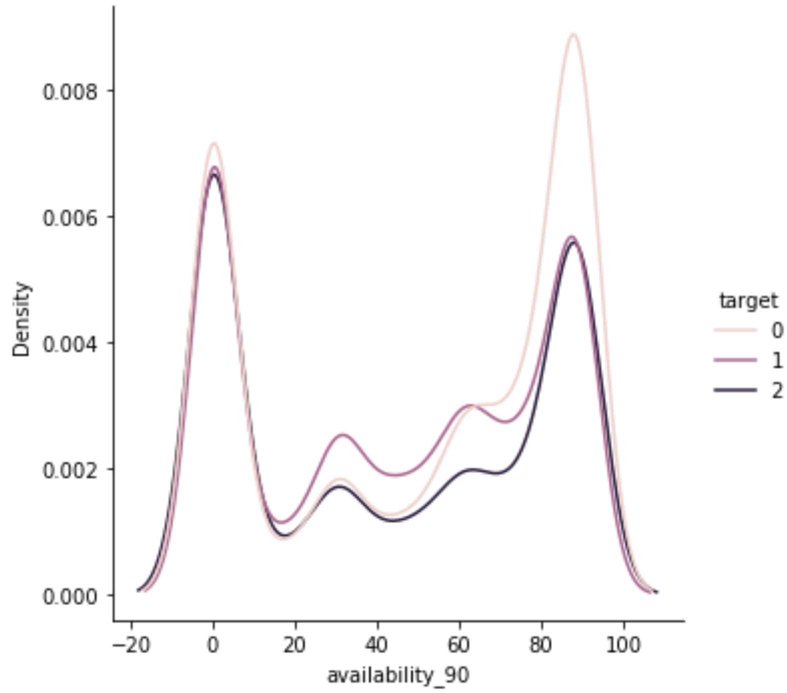
In [67]:
```
sns.displot(data=df2, x='availability_60', hue='target', kind='kde')
```

Out[67]: <seaborn.axisgrid.FacetGrid at 0x1a178b24e0>



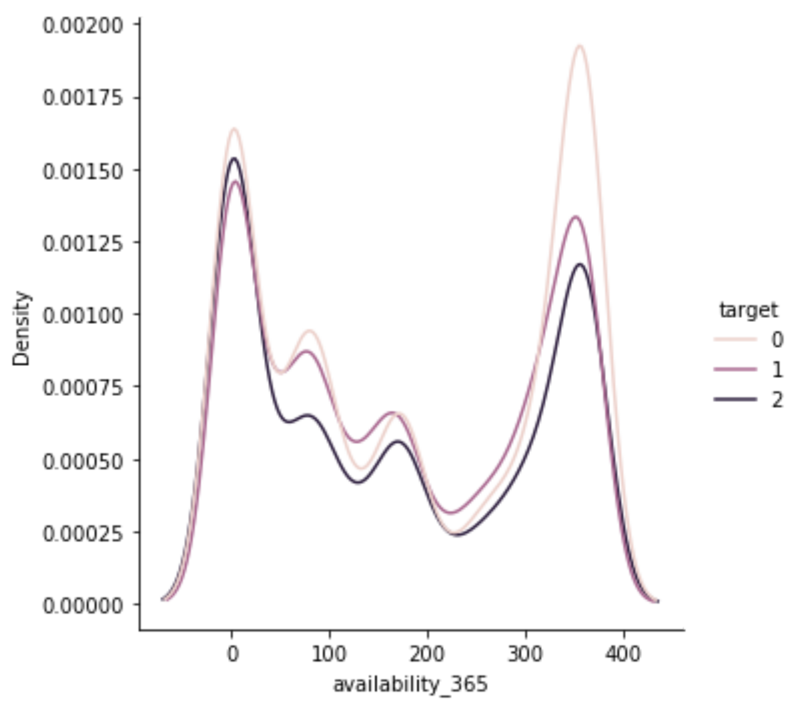
In [68]:
```
sns.displot(data=df2, x='availability_90', hue='target', kind='kde')
```

Out[68]: <seaborn.axisgrid.FacetGrid at 0x1a178b2470>



In [69]:
```
sns.displot(data=df2, x='availability_365', hue='target', kind='kde')
```

Out[69]: <seaborn.axisgrid.FacetGrid at 0x1a2f9094a8>



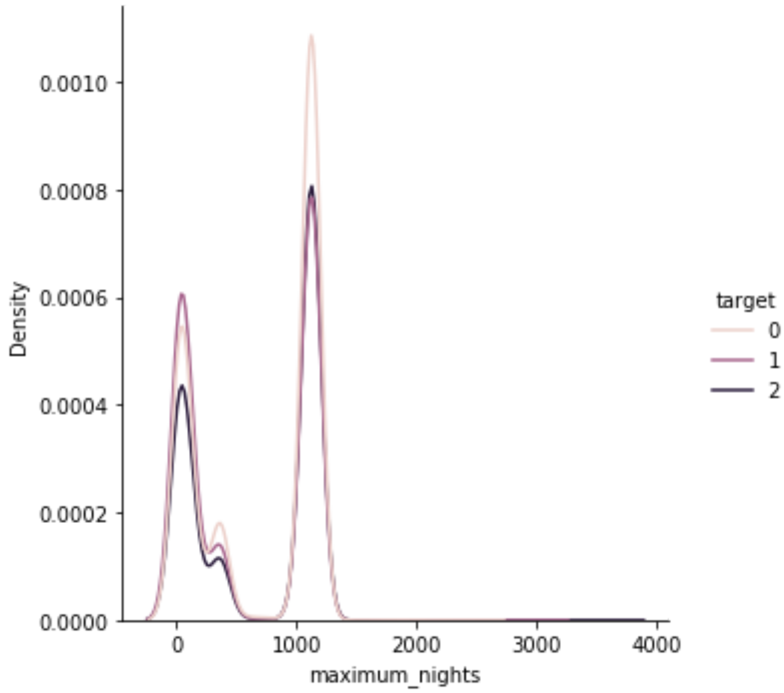
This graph helps visualize the avaibility within 30, 60, 90, 365 days better than the barplots do. A trend to notice is that class 0 (subpar) consistently has more avaibility while class 2(best) is always booked and has the least availability out of the classes.

In [70]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30449 entries, 0 to 31470
Data columns (total 35 columns):
host_response_rate                        30449 non-null float64
host_acceptance_rate                      30449 non-null float64
latitude                                  30449 non-null float64
longitude                                 30449 non-null float64
accommodates                              30449 non-null int64
bedrooms                                  30449 non-null float64
beds                                      30449 non-null float64
minimum_nights                            30449 non-null int64
maximum_nights                            30449 non-null int64
availability_30                           30449 non-null int64
availability_60                           30449 non-null int64
availability_90                           30449 non-null int64
availability_365                          30449 non-null int64
number_of_reviews                         30449 non-null int64
review_scores_accuracy                    30449 non-null float64
review_scores_cleanliness                 30449 non-null float64
review_scores_checkin                     30449 non-null float64
review_scores_communication               30449 non-null float64
review_scores_location                    30449 non-null float64
review_scores_value                       30449 non-null float64
reviews_per_month                         30449 non-null float64
instant_bookable                          30449 non-null int64
has_availability                          30449 non-null int64
host_identity_verified                    30449 non-null float64
host_has_profile_pic                      30449 non-null float64
host_is_superhost                         30449 non-null float64
host_response_time_a few days or more     30449 non-null uint8
host_response_time_within a day           30449 non-null uint8
host_response_time_within a few hours     30449 non-null uint8
host_response_time_within an hour         30449 non-null uint8
room_type_Entire home/apt                 30449 non-null uint8
room_type_Hotel room                      30449 non-null uint8
room_type_Private room                    30449 non-null uint8
room_type_Shared room                     30449 non-null uint8
target                                    30449 non-null int64
dtypes: float64(16), int64(11), uint8(8)
memory usage: 6.7 MB
```

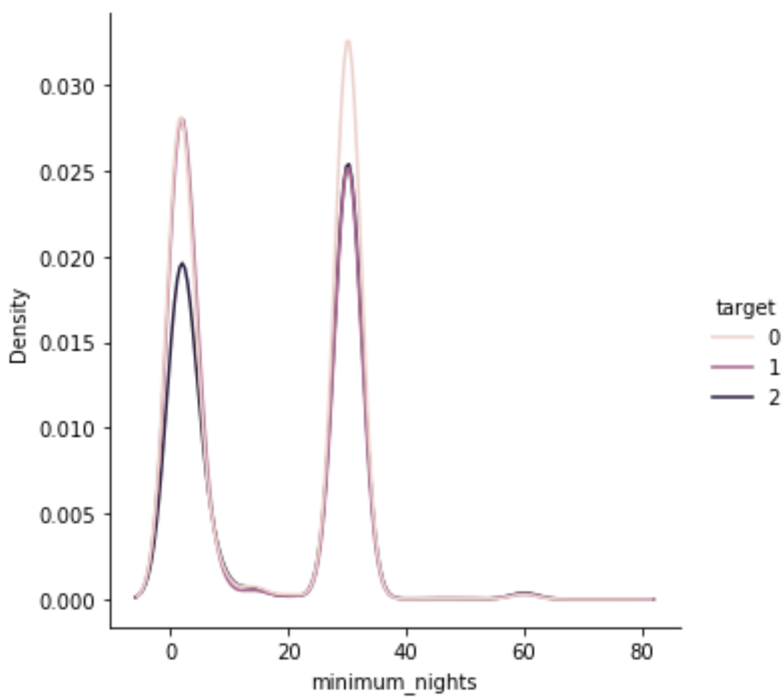
In [71]:
```
sns.displot(data=df2, x='maximum_nights', hue='target', kind='kde')
```

Out[71]: <seaborn.axisgrid.FacetGrid at 0x1a2f8f0c88>



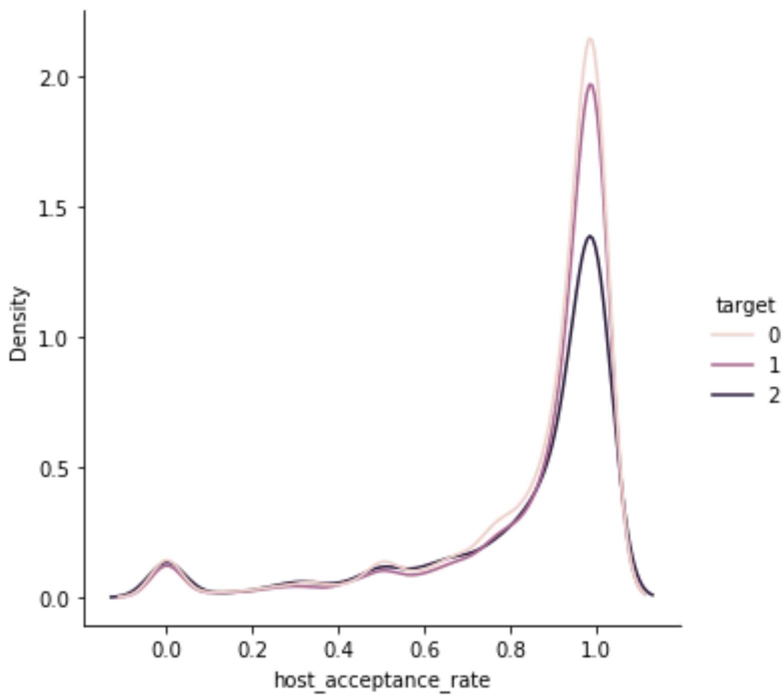
In [72]:
```
sns.displot(data=df2, x='minimum_nights', hue='target', kind='kde')
```

Out[72]: <seaborn.axisgrid.FacetGrid at 0x1a2318a2e8>



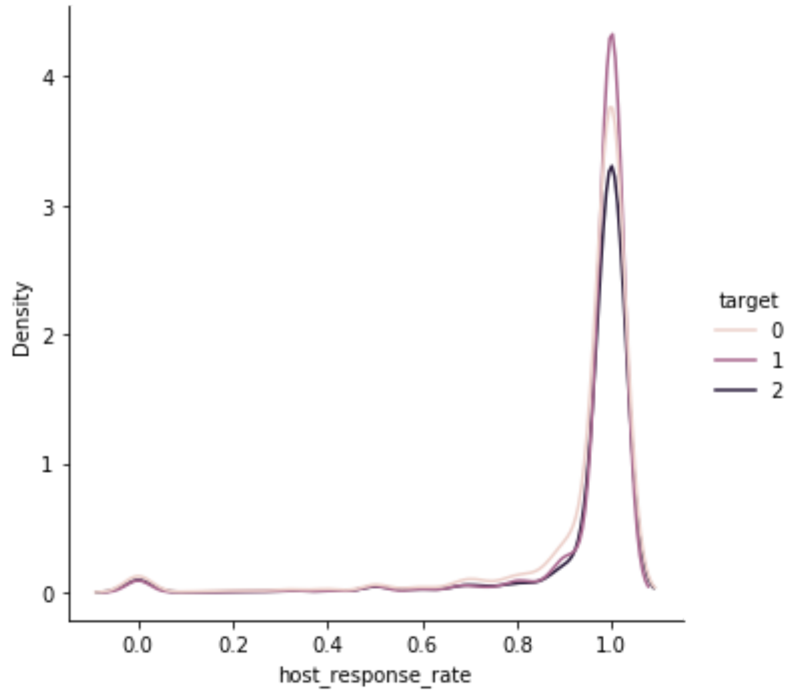
In [73]:
```
sns.displot(data=df2, x='host_acceptance_rate', hue='target', kind='kde'
)
#Maybe the lower rating listings accept for people because they need mon
ey? And therefore the quality of the listing
#is lower. Conversely, the best rating listings are more strict on the r
ating of the guest and whether or not they
#are allowed to stay.
```

Out[73]: <seaborn.axisgrid.FacetGrid at 0x1a231ef6d8>



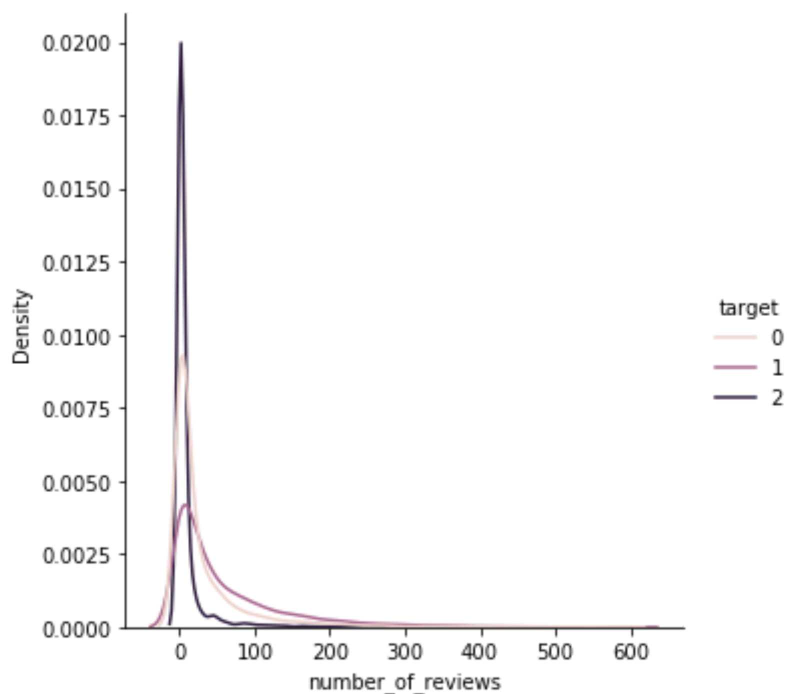
In [74]:
```
sns.displot(data=df2, x='host_response_rate', hue='target', kind='kde')
```

Out[74]: <seaborn.axisgrid.FacetGrid at 0x1a2a172860>



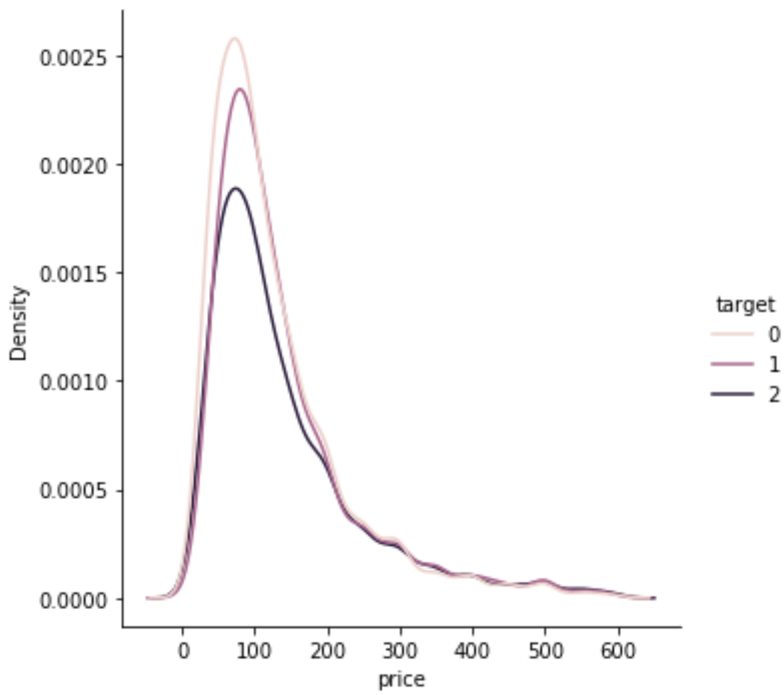
In [75]:
```
sns.displot(data=df2, x='number_of_reviews', hue='target', kind='kde')
#Class 2 seems to have a lot of listings with ~0-30 reviews by a large m
argin.
```

Out[75]: <seaborn.axisgrid.FacetGrid at 0x1a2a195b38>



In [78]:
```
sns.displot(data=df2, x='price', hue='target', kind='kde')
#While the classes seem to remain consistent as price increases, we CAN
 say that class 0 has more listings in the
#less expensive range
```

Out[78]: <seaborn.axisgrid.FacetGrid at 0x1a206d3940>



In [79]:
```
df2.head(2)
```

Out[79]:

	host_response_rate	host_acceptance_rate	latitude	longitude	accommodates	bedrooms	b
0	1.0	0.0	33.98209	-118.38494	6	2.0	
1	1.0	1.0	34.09768	-118.34602	1	1.0	

In [131]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29771 entries, 0 to 31470
Data columns (total 36 columns):
host_response_rate                      29771 non-null float64
host_acceptance_rate                    29771 non-null float64
latitude                                29771 non-null float64
longitude                               29771 non-null float64
accommodates                            29771 non-null int64
bedrooms                                29771 non-null float64
beds                                    29771 non-null float64
price                                   29771 non-null float64
minimum_nights                          29771 non-null int64
maximum_nights                          29771 non-null int64
availability_30                         29771 non-null int64
availability_60                         29771 non-null int64
availability_90                         29771 non-null int64
availability_365                        29771 non-null int64
number_of_reviews                       29771 non-null int64
review_scores_accuracy                  29771 non-null float64
review_scores_cleanliness               29771 non-null float64
review_scores_checkin                   29771 non-null float64
review_scores_communication             29771 non-null float64
review_scores_location                  29771 non-null float64
review_scores_value                     29771 non-null float64
reviews_per_month                       29771 non-null float64
instant_bookable                        29771 non-null int64
has_availability                        29771 non-null int64
host_identity_verified                  29771 non-null float64
host_has_profile_pic                    29771 non-null float64
host_is_superhost                       29771 non-null float64
host_response_time_a few days or more   29771 non-null uint8
host_response_time_within a day         29771 non-null uint8
host_response_time_within a few hours   29771 non-null uint8
host_response_time_within an hour       29771 non-null uint8
room_type_Entire home/apt               29771 non-null uint8
room_type_Hotel room                    29771 non-null uint8
room_type_Private room                  29771 non-null uint8
room_type_Shared room                   29771 non-null uint8
target                                  29771 non-null int64
dtypes: float64(17), int64(11), uint8(8)
memory usage: 6.8 MB
```

Initial Modeling

In [80]:
```
#Define your X and Y for train, test split
X = df2.drop(['target'],axis=1)
y = df2['target']
```

In [81]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=4
2)
```

KNN

KNN (K-Nearest Neighbors) modeling. I implemented a function that will help determine the best K value based on accuracy of the models.

In [82]:
```
from sklearn.neighbors import KNeighborsClassifier
```

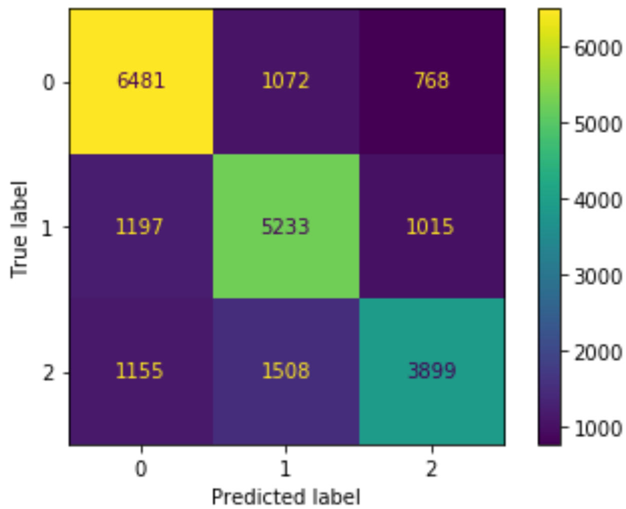
In [83]:
```
def find_best_k(X_train, y_train, X_test, y_test, min_k=1, max_k=25):
    best_k = 0
    best_score = 0.0
    for k in range(min_k, max_k+1, 2):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        preds = knn.predict(X_test)
        acc = accuracy_score(y_test, preds)
        if acc > best_score:
            best_k = k
            best_score = acc

    print("Best Value for k: {}".format(best_k))
    print("acc: {}".format(best_score))
```

In [84]:
```
#This is for KNN because it uses distance so if data is not on the same
 scale, it will be weighted differently
from sklearn.preprocessing import StandardScaler
```

In [85]:
```
#Transforming X_train and X_test using StandardScaler
#Fit is where StandardScaler understand the distribution and variance of
the data a learns how to scale it.
#Transform then applies the "memorized" fit and applies the scaling.
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#Running a baseline KNN model
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

y_predict = knn.predict(X_test)
```

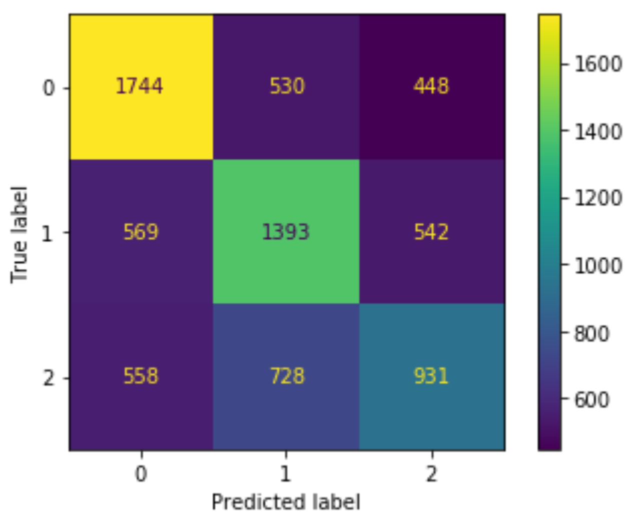
In [86]:
```
plot_confusion_matrix(knn, X_train, y_train)
```

Out[86]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2
3c975f8>
```



In [87]:
```
plot_confusion_matrix(knn, X_test, y_test)
```

Out[87]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2
b2c8ac8>
```



In [88]:
```
print(classification_report(y_train, knn.predict(X_train)))
print(classification_report(y_test, knn.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.73      0.78      0.76      8321
           1       0.67      0.70      0.69      7445
           2       0.69      0.59      0.64      6562

    accuracy                           0.70     22328
   macro avg       0.70      0.69      0.69     22328
weighted avg       0.70      0.70      0.70     22328

              precision    recall  f1-score   support

           0       0.61      0.64      0.62      2722
           1       0.53      0.56      0.54      2504
           2       0.48      0.42      0.45      2217

    accuracy                           0.55      7443
   macro avg       0.54      0.54      0.54      7443
weighted avg       0.54      0.55      0.54      7443
```

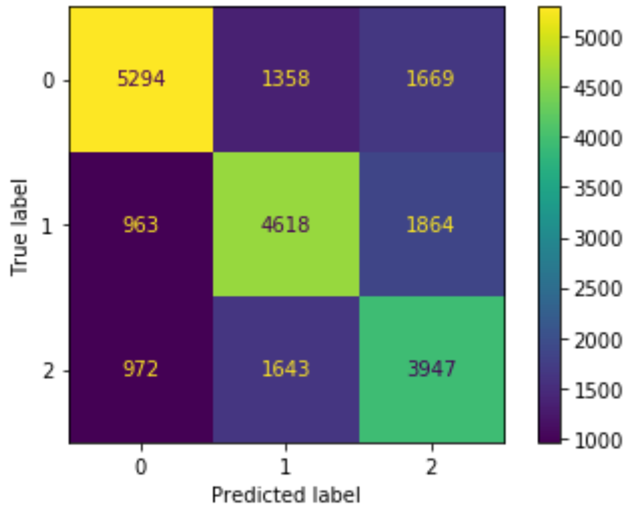
In [89]:
```
#Testing values for the best k using defined function above.
find_best_k(X_train, y_train, X_test, y_test)
```

```
Best Value for k: 25
acc: 0.5851135294907968
```

In [90]:
```
#Running adjusted KNN parameters model.
knn = KNeighborsClassifier(n_neighbors=25)
knn.fit(X_train, y_train)

y_predict = knn.predict(X_test)
```

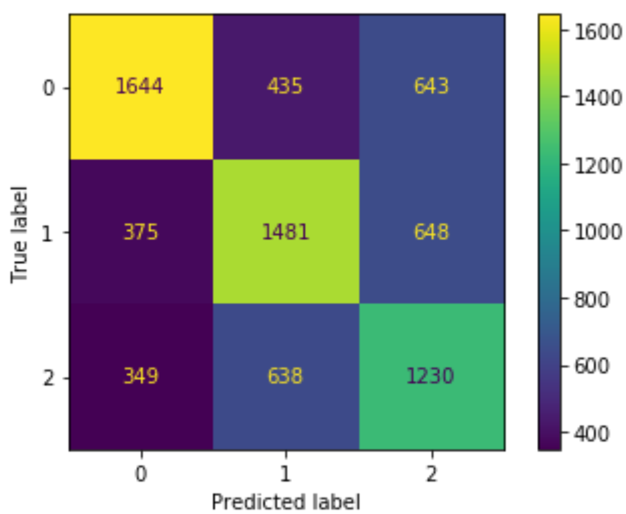
In [91]:
```
plot_confusion_matrix(knn, X_train, y_train)
```

Out[91]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2
c18f240>
```



In [92]:
```
plot_confusion_matrix(knn, X_test, y_test)
```

Out[92]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2
b04f668>
```



In [93]:
```
print(classification_report(y_train, knn.predict(X_train)))
print(classification_report(y_test, knn.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.73      0.64      0.68      8321
           1       0.61      0.62      0.61      7445
           2       0.53      0.60      0.56      6562

    accuracy                           0.62     22328
   macro avg       0.62      0.62      0.62     22328
weighted avg       0.63      0.62      0.62     22328

              precision    recall  f1-score   support

           0       0.69      0.60      0.65      2722
           1       0.58      0.59      0.59      2504
           2       0.49      0.55      0.52      2217

    accuracy                           0.59      7443
   macro avg       0.59      0.58      0.58      7443
weighted avg       0.59      0.59      0.59      7443
```

In [94]:
```
print("Testing Accuracy: {}".format(accuracy_score(y_test, y_predict)))
```

```
Testing Accuracy: 0.5851135294907968
```

Improved by 4% in terms of model accuracy. The adjusted model performs better by .05 looking at the weighted average. The F1 score also increased slightly.

In [ ]:

Decision Trees

Decision Tree modeling.

In [95]:
```
from sklearn.tree import DecisionTreeClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=4
2)
```

In [96]:
```
#Instantiate classifier
ctree = DecisionTreeClassifier()

ctree.fit(X_train, y_train)
```

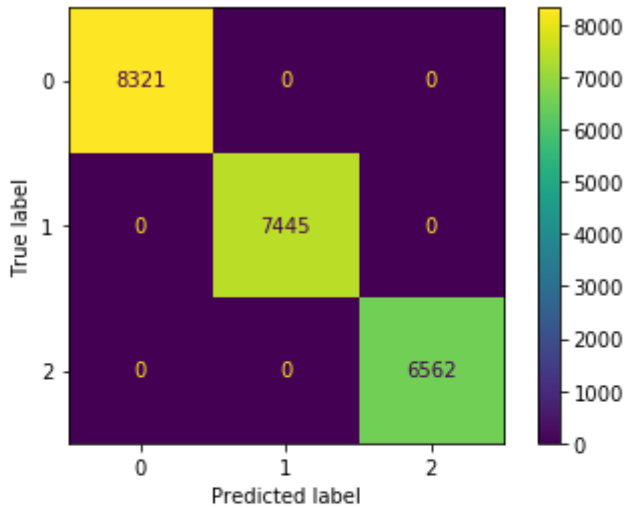
Out[96]:
```
DecisionTreeClassifier()
```

In [97]:
```
ctree_ypred = ctree.predict(X_test)
```

Running a baseline model with default hyperparameters

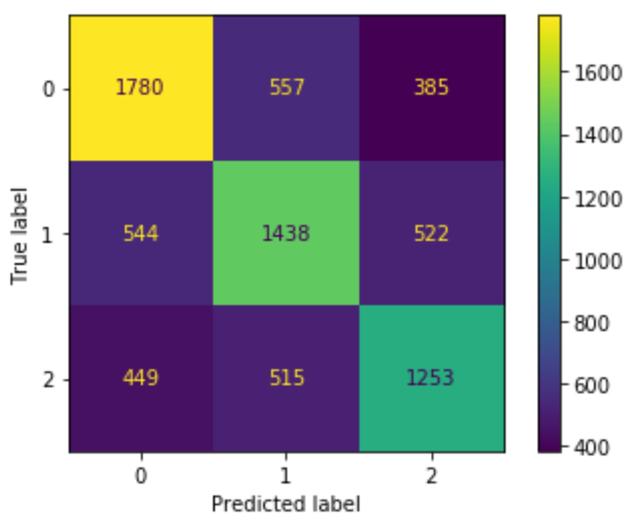
In [98]:
```
plot_confusion_matrix(ctree, X_train, y_train)
```

Out[98]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2
b3990f0>
```



In [99]:
```
plot_confusion_matrix(ctree, X_test, y_test)
```

Out[99]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2
e27feb8>
```



In [100]:
```
print(classification_report(y_train, ctree.predict(X_train)))
print(classification_report(y_test, ctree.predict(X_test)))
```

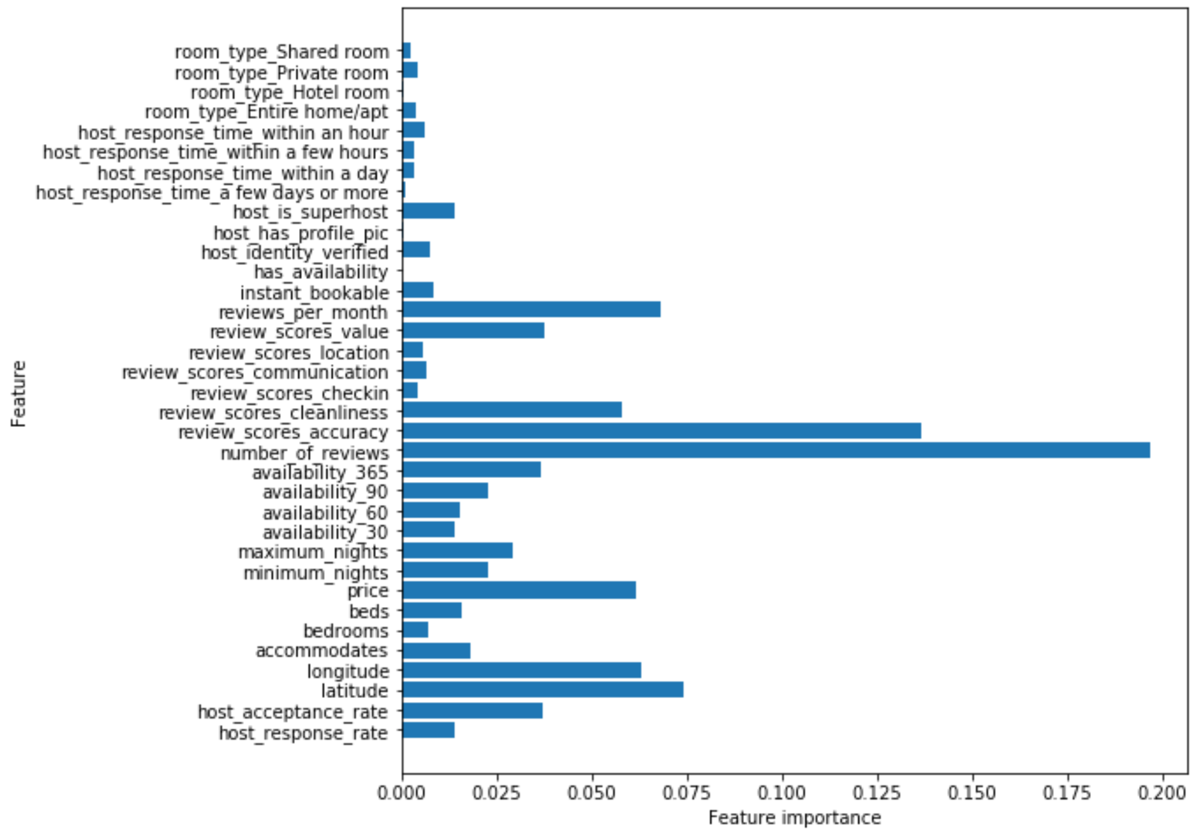
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      8321
           1       1.00      1.00      1.00      7445
           2       1.00      1.00      1.00      6562

    accuracy                           1.00     22328
   macro avg       1.00      1.00      1.00     22328
weighted avg       1.00      1.00      1.00     22328

              precision    recall  f1-score   support

           0       0.64      0.65      0.65      2722
           1       0.57      0.57      0.57      2504
           2       0.58      0.57      0.57      2217

    accuracy                           0.60      7443
   macro avg       0.60      0.60      0.60      7443
weighted avg       0.60      0.60      0.60      7443
```

Interestingly, our training set predicted perfectly with 1.0's across all metrics while the testing set specifically had an accuracy of 60%. These indications tell me that there are signs of overfitting, I safely assumed this was the result of some hyperparameters that could be tuned.

In [102]:
```
#Further analyzing features importances within the decision tree.
ctree.feature_importances_
```

Out[102]:
```
array([0.01407194, 0.03694513, 0.07398842, 0.06305476, 0.01815772,
       0.00683402, 0.01595318, 0.06175618, 0.02263324, 0.02941858,
       0.01388428, 0.01551683, 0.02293637, 0.03675446, 0.19665271,
       0.13668829, 0.05809635, 0.00428663, 0.00678426, 0.00557085,
       0.03761697, 0.0680748 , 0.00817935, 0.        , 0.00733673,
       0.00040626, 0.01383361, 0.00092115, 0.00326617, 0.00335865,
       0.00610037, 0.0037631 , 0.00065564, 0.00416857, 0.00233446])
```

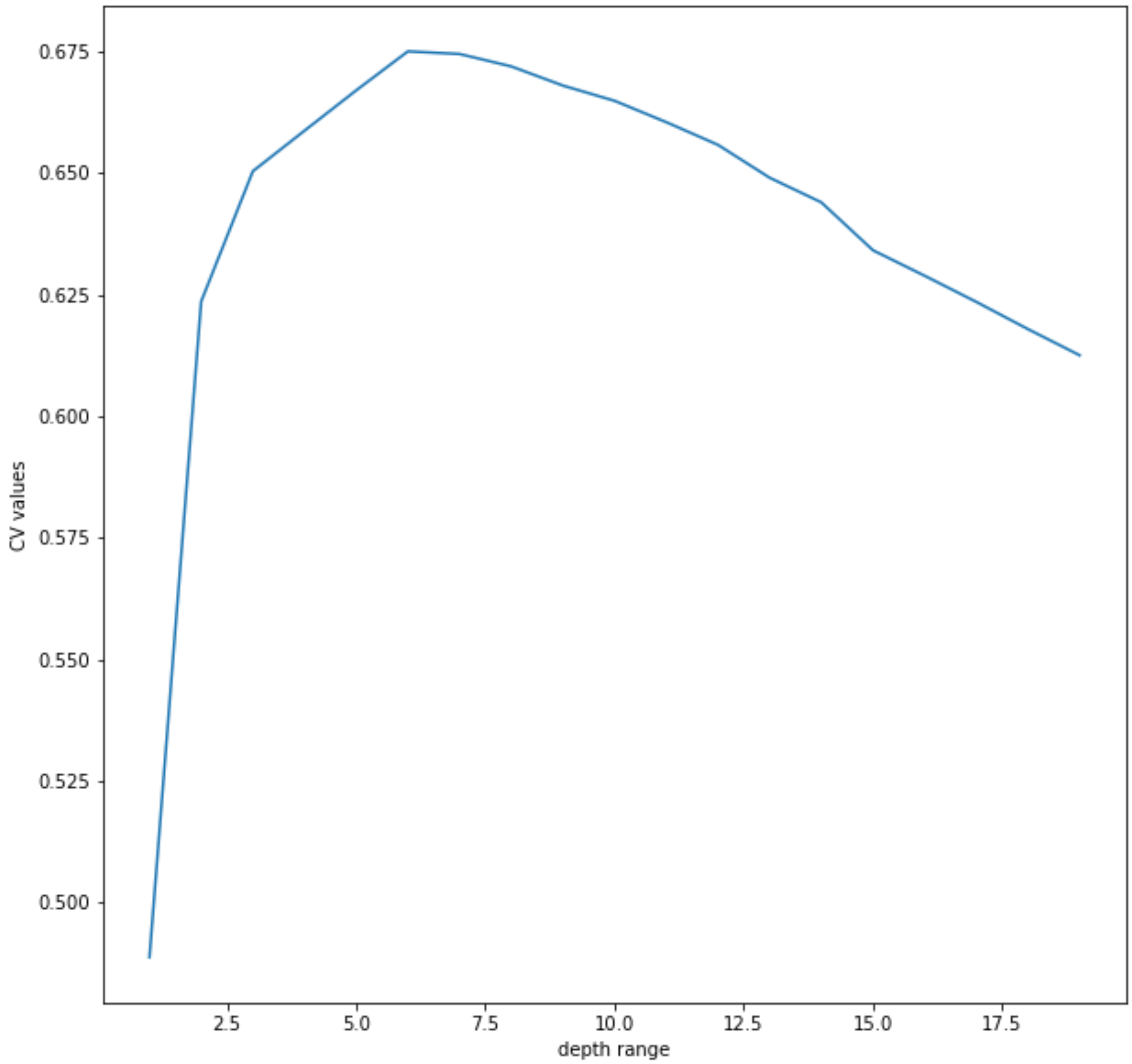
In [103]:
```
def plot_feature_importances(model):
    n_features = X_train.shape[1]
    plt.figure(figsize=(8,8))
    plt.barh(range(n_features), model.feature_importances_, align='cente
r')
    plt.yticks(np.arange(n_features), X_train.columns.values)
    plt.xlabel('Feature importance')
    plt.ylabel('Feature')

plot_feature_importances(ctree)
```



Clear sign of overfitting because there is a max_depth of none on the training set. Test set has an accuracy of .61 For adjusting this model, consider GridSearchCV, and changing the criterion to entropy rather than gini.

In [104]:
```
#Utilizing a loop to illustrate the best depth in a cross validation of
 models.
from sklearn.model_selection import cross_val_score
score = cross_val_score(ctree, X_train, y_train, cv = 20)
score.mean()
depth_range = range(1, 20)
val = []
for depth in depth_range:
    ctree = DecisionTreeClassifier(max_depth = depth)
    depth_score = cross_val_score(ctree, X_train, y_train, cv = 20)
    val.append(depth_score.mean())
print(val)
plt.figure(figsize=(10,10))
plt.plot(depth_range, val)
plt.xlabel('depth range')
plt.ylabel('CV values')
plt.show()
```

```
[0.4888035348138736, 0.6236119133110642, 0.6503916340171286, 0.65872203
93206329, 0.667008203296721, 0.6750249885285406, 0.6744886777498612, 0.
6719352752989799, 0.6679934652791817, 0.6648582673122772, 0.66046935114
85898, 0.655811698000597, 0.6490501551454709, 0.6439882734410848, 0.634
1350920765105, 0.6288960044024733, 0.6235218663663231, 0.61792218981334
41, 0.6125485330971656]
```



In [105]:
```python
from sklearn.model_selection import GridSearchCV
```

In [106]:
```python
#Using GridSearchCV
tree_params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, '6'],
    'max_features': ['auto', 'sqrt', 'log2'],
    'min_samples_split': [None, 5, 7, 9, 10]
}
```

In [107]:
```
gs_tree = GridSearchCV(ctree, tree_params, cv=3)

gs_tree.fit(X_train, y_train)

print(f"Training Accuracy: {gs_tree.best_score_ :.2%}")
print("")
print(f"Optimal Parameters: {gs_tree.best_params_}")
```

```
Training Accuracy: 59.92%

Optimal Parameters: {'criterion': 'gini', 'max_depth': None, 'max_featu
res': 'sqrt', 'min_samples_split': 9}
```

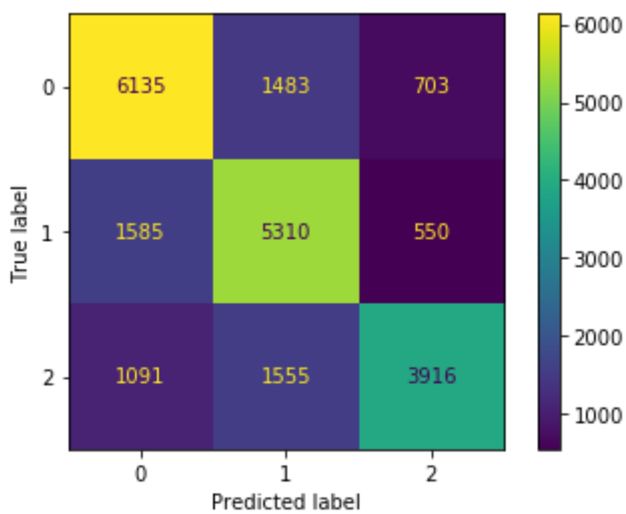
Testing the grid search parameters, there is a training accuracy of 59.92%. While the depth values cross validation gave us 6, I combined some of the grid search values to emphasize a well-rounded model; with hyper parameters from cross validation.

In [108]:
```
ctree = DecisionTreeClassifier(max_depth=6, criterion='gini', min_sample
s_split=10)

ctree.fit(X_train, y_train)
```

Out[108]:
```
DecisionTreeClassifier(max_depth=6, min_samples_split=10)
```

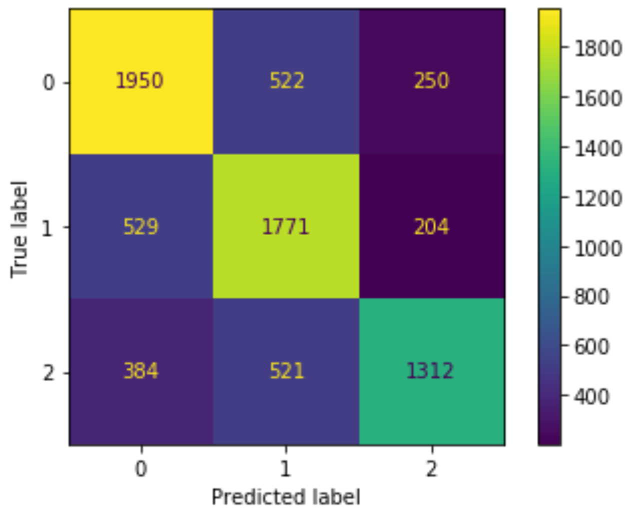
In [109]:
```
plot_confusion_matrix(ctree, X_train, y_train)
```

Out[109]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2
c957b00>
```



In [110]:
```
plot_confusion_matrix(ctree, X_test, y_test)
```

Out[110]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2
e16fcf8>
```



In [111]:
```
print(classification_report(y_train, ctree.predict(X_train)))
print(classification_report(y_test, ctree.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.70      0.74      0.72      8321
           1       0.64      0.71      0.67      7445
           2       0.76      0.60      0.67      6562

    accuracy                           0.69     22328
   macro avg       0.70      0.68      0.69     22328
weighted avg       0.69      0.69      0.69     22328

              precision    recall  f1-score   support

           0       0.68      0.72      0.70      2722
           1       0.63      0.71      0.67      2504
           2       0.74      0.59      0.66      2217

    accuracy                           0.68      7443
   macro avg       0.68      0.67      0.67      7443
weighted avg       0.68      0.68      0.68      7443
```

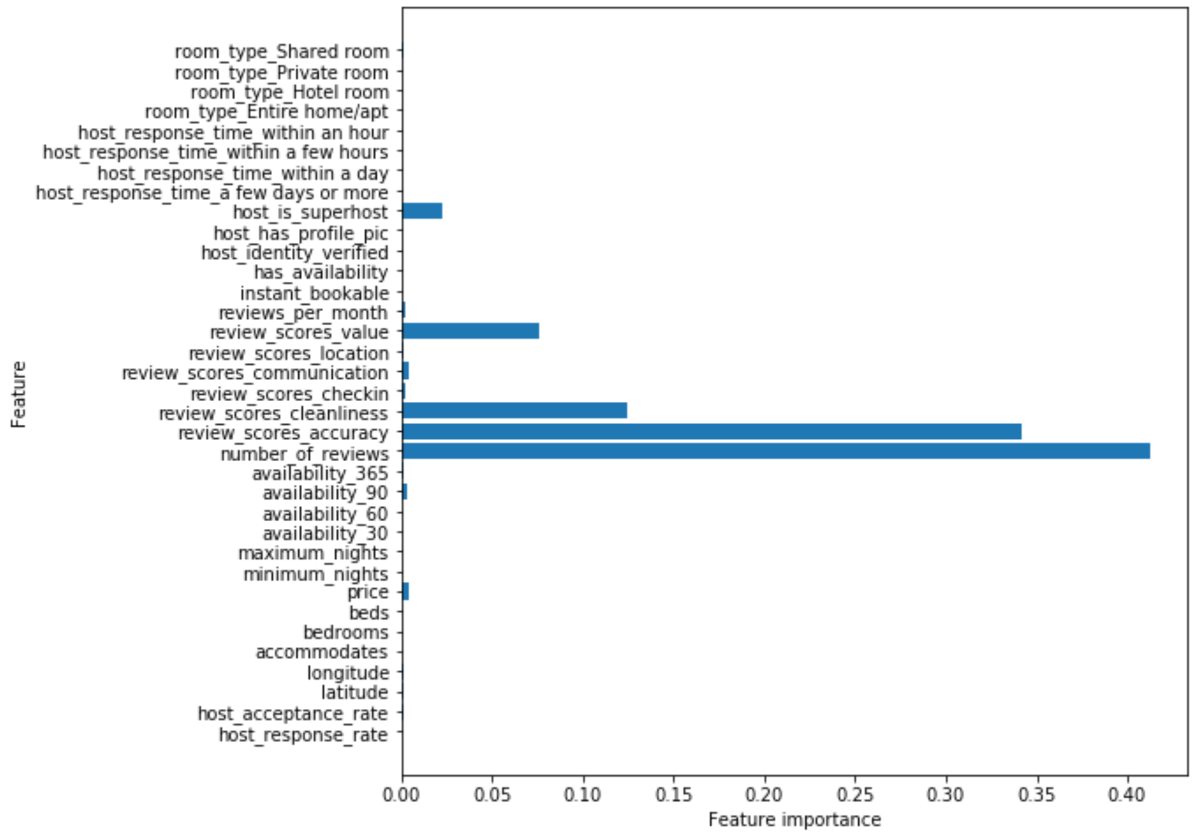
Accuracy improved by .07 and there is no sign of overfitting as the train and test scores are very similar.

In [638]:
```
ctree_ypred = ctree.predict(X_test)
```

So, the final model has an improved accuracy of 68% from 61% with an recall score of .61 for Class 2. The main metric I am looking for in this model is accuracy; however, I thought it would be important to point out that because Class 2 and Class 1 are so close in terms of rating, the ability to distinguish the 2 would be essential.

In terms of misclassified data, with respect to the confusion matrix as well: for Class 0 (2215 predicted correctly and 649 predicted incorrectly). Class 1 (1579 predicted correctly and 938 predicted incorrectly). Class 2 (1370 predicted correctly and 861 predicted incorecctly).

In [112]:
```
plot_feature_importances(ctree)
```

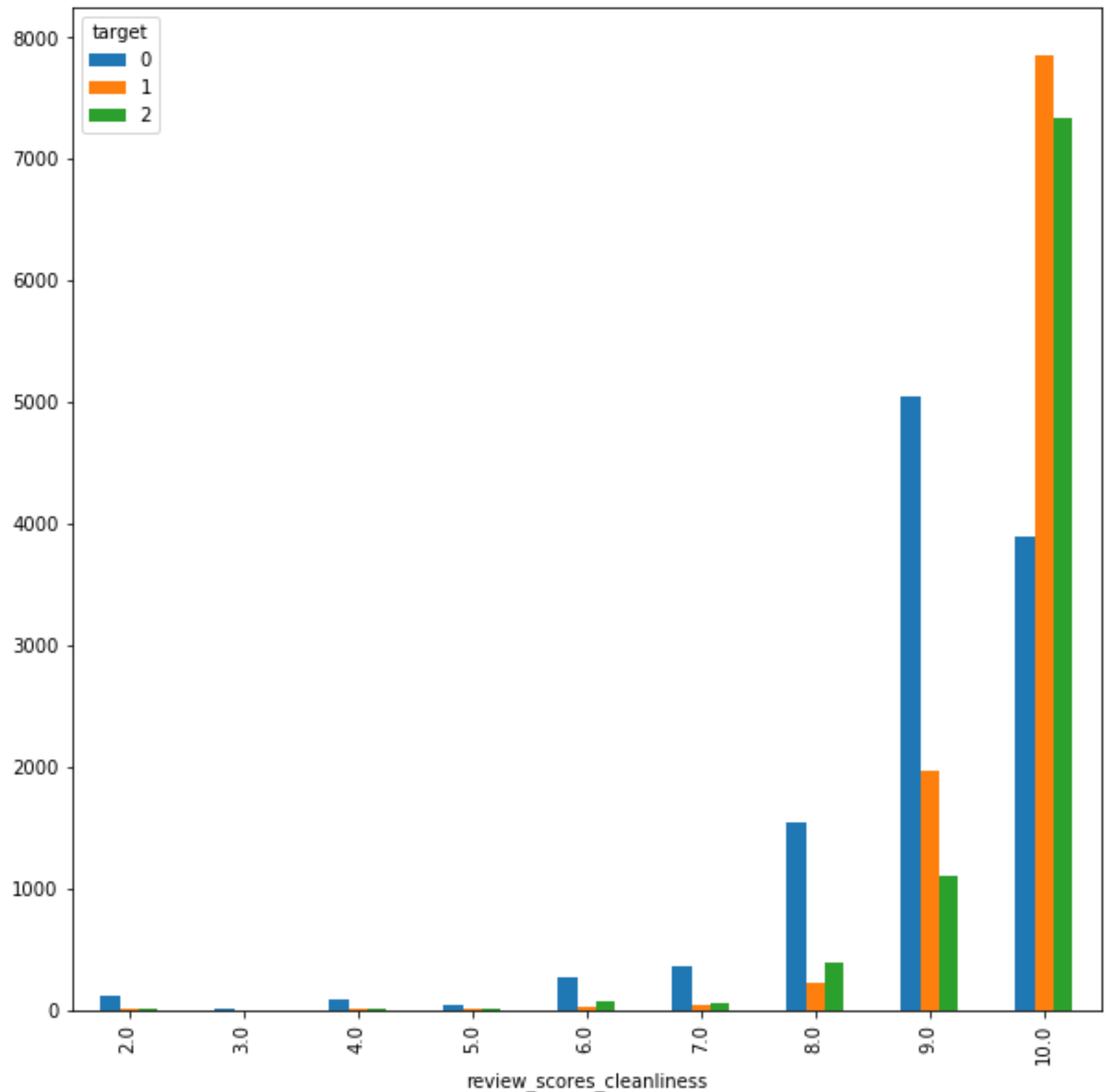


In terms of features, clearly the most important feature is number_of_reviews. Pre-modeling, I observed that Class 2 which is the perfect class had a marginally greater number of reviews than the other 2 classes. As for host improvements, cleanliness, superhost title and communication seem to stand out.

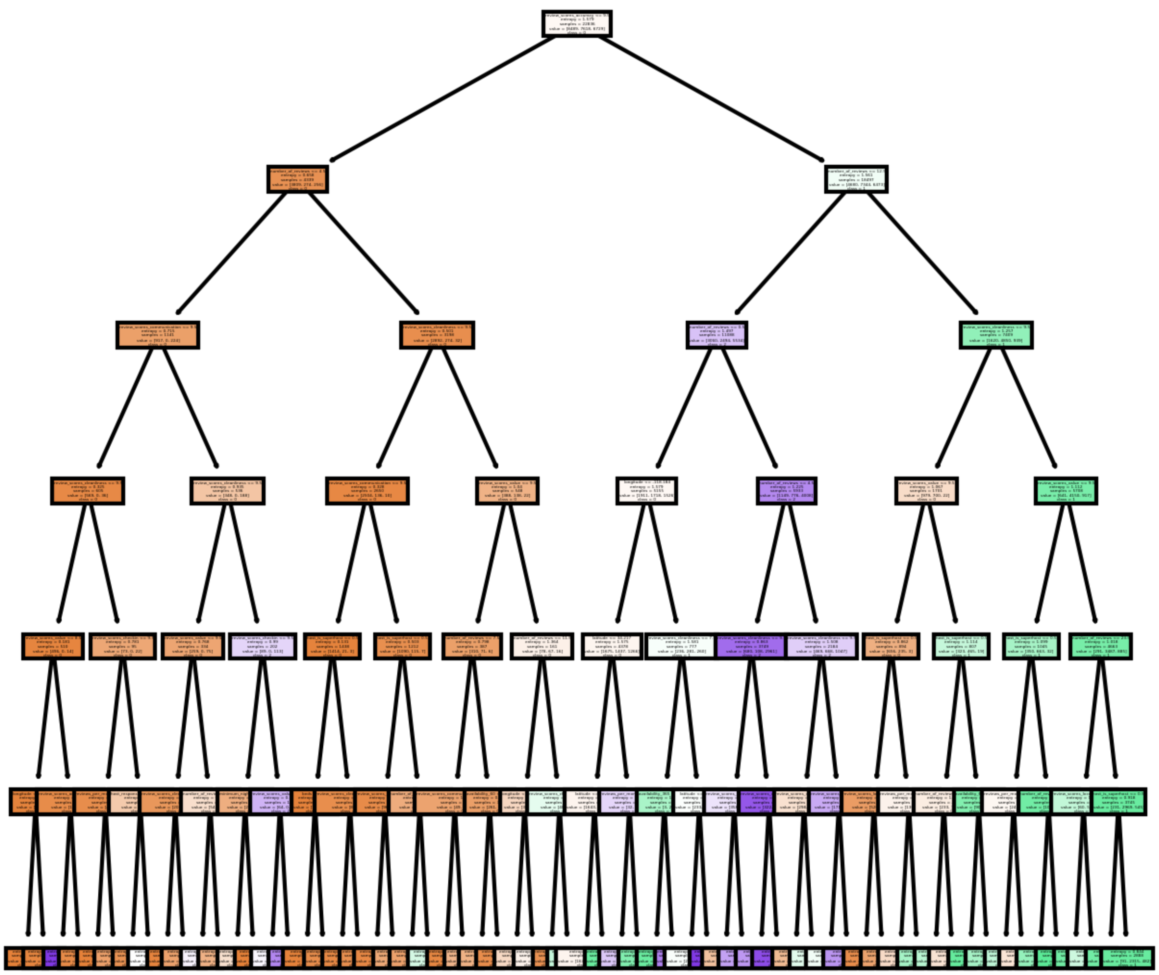
If we look at percentages, number_of_reviewss: 40.5%, accuracy: 31%, cleanliness: 15%, superhost title: 3%, communication: 1%. While from the baseline, there were lots of contributors. This is because the tuning done on the model directly influenced behavior and minimizing the loss function more efficiently.

In [875]:
```
pd.crosstab(df2['review_scores_cleanliness'], df2['target']).plot.bar(fi
gsize=(10,10))
```

Out[875]: <matplotlib.axes._subplots.AxesSubplot at 0x1aa7e1cfd0>



In [637]:
```
#Decision tree visualized.
from sklearn import tree
fig, axes = plt.subplots(nrows = 1,ncols = 1, figsize = (5,5), dpi=300)
tree.plot_tree(ctree,
               feature_names = df2.columns,
               class_names=np.unique(y).astype('str'),
               filled = True)
plt.show()
```



In [ ]:

Random Forests: Final Model

Random forests modeling.

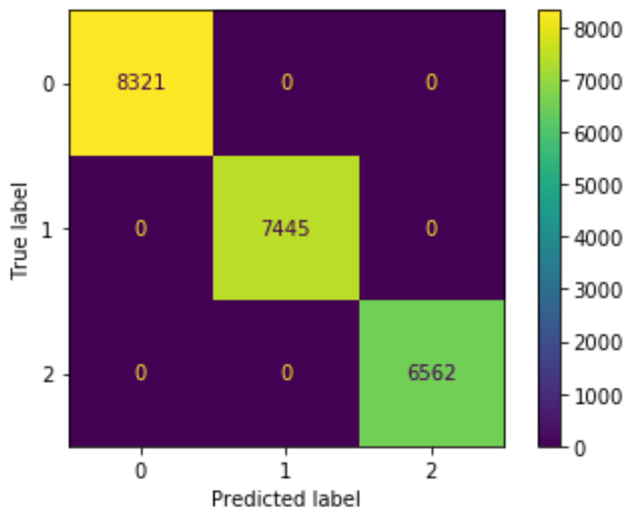
In [113]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [114]:
```
rf = RandomForestClassifier()

# Fit classifier
rf.fit(X_train, y_train)
```

Out[114]:
```
RandomForestClassifier()
```

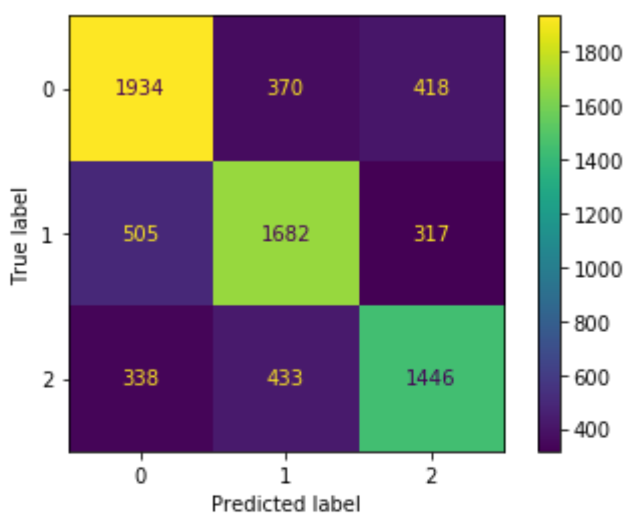
In [115]:
```
plot_confusion_matrix(rf, X_train, y_train)
```

Out[115]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2
c603a20>
```



In [116]:
```
plot_confusion_matrix(rf, X_test, y_test)
```

Out[116]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2
c6354e0>
```



In [117]:
```
print(classification_report(y_train, rf.predict(X_train)))
print(classification_report(y_test, rf.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      8321
           1       1.00      1.00      1.00      7445
           2       1.00      1.00      1.00      6562

    accuracy                           1.00     22328
   macro avg       1.00      1.00      1.00     22328
weighted avg       1.00      1.00      1.00     22328

              precision    recall  f1-score   support

           0       0.70      0.71      0.70      2722
           1       0.68      0.67      0.67      2504
           2       0.66      0.65      0.66      2217

    accuracy                           0.68      7443
   macro avg       0.68      0.68      0.68      7443
weighted avg       0.68      0.68      0.68      7443
```

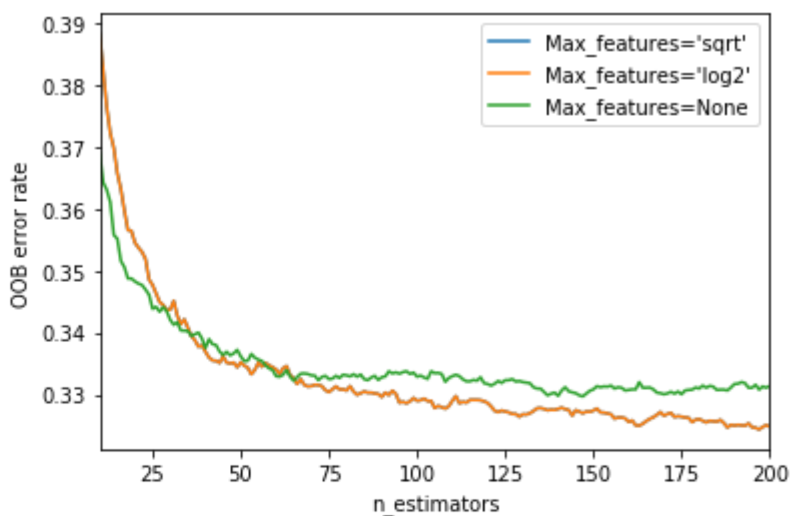
In [118]:
```
#Using GridSearchCV
rf_params = {
    'n_estimators': [30, 100],
    'max_depth': [2, 6, 10, 15],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [3, 6]
}
```

In [119]:
```
gs_rf = GridSearchCV(rf, rf_params, cv=3)

gs_rf.fit(X_train, y_train)

print(f"Optimal Parameters: {gs_rf.best_params_}")
```

```
Optimal Parameters: {'max_depth': 15, 'min_samples_leaf': 3, 'min_sampl
es_split': 5, 'n_estimators': 100}
```

In [715]:
```
from collections import OrderedDict
from sklearn.datasets import make_classification

ensemble_clfs = [
    ("Max_features='sqrt'",
        RandomForestClassifier(warm_start=True, oob_score=True,
                               max_features="sqrt",
                               random_state=123)),
    ("Max_features='log2'",
        RandomForestClassifier(warm_start=True, max_features='log2',
                               oob_score=True,
                               random_state=123)),
    ("Max_features=None",
        RandomForestClassifier(warm_start=True, max_features=None,
                               oob_score=True,
                               random_state=123))
]
```

In [716]:
```
error_rate = OrderedDict((label, []) for label, _ in ensemble_clfs)

min_estimators = 10
max_estimators = 200

for label, clf in ensemble_clfs:
    for i in range(min_estimators, max_estimators + 1):
        clf.set_params(n_estimators=i)
        clf.fit(X_train, y_train)

        oob_error = 1 - clf.oob_score_
        error_rate[label].append((i, oob_error))

for label, clf_err in error_rate.items():
    xs, ys = zip(*clf_err)
    plt.plot(xs, ys, label=label)

plt.xlim(min_estimators, max_estimators)
plt.xlabel("n_estimators")
plt.ylabel("OOB error rate")
plt.legend(loc="upper right")
plt.show()
```

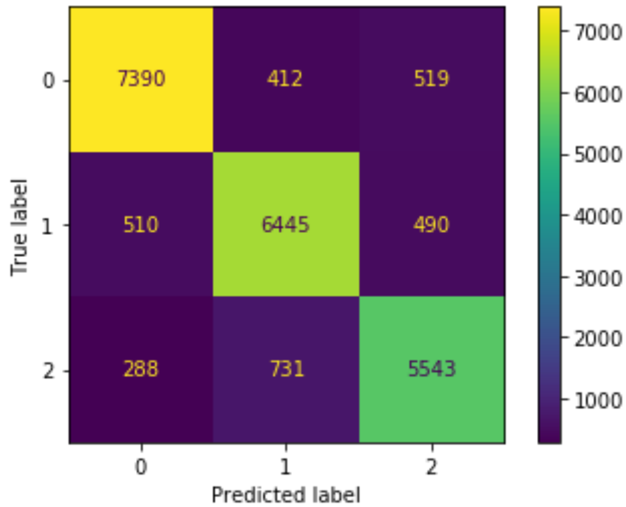


In [167]:
```
rf = RandomForestClassifier(n_estimators=200, max_depth=15, min_samples_
leaf=3, min_samples_split=10, \
                            criterion='entropy', max_features='log2')

# Fit classifier
rf.fit(X_train, y_train)
```

Out[167]:
```
RandomForestClassifier(criterion='entropy', max_depth=15, max_features
='log2',
                       min_samples_leaf=3, min_samples_split=10,
                       n_estimators=200)
```

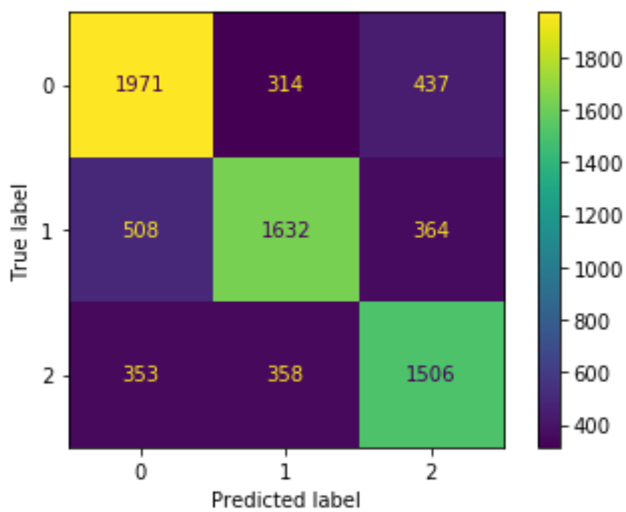
In [168]:
```
plot_confusion_matrix(rf, X_train, y_train)
```

Out[168]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a2e6bfeb8>



In [169]:
```
plot_confusion_matrix(rf, X_test, y_test)
```

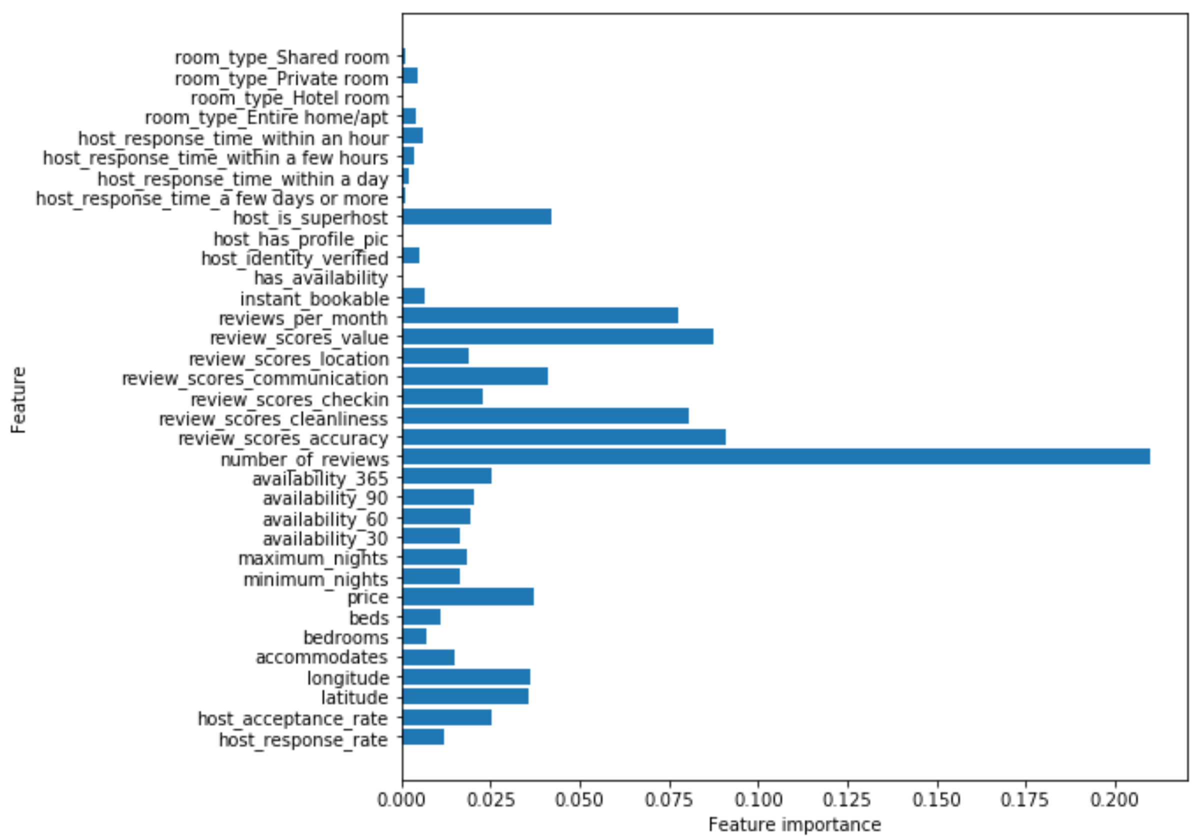
Out[169]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a6e93a4a8>



In [170]:
```
print(classification_report(y_train, rf.predict(X_train)))
print(classification_report(y_test, rf.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.90      0.89      0.90      8321
           1       0.85      0.87      0.86      7445
           2       0.85      0.84      0.85      6562

    accuracy                           0.87     22328
   macro avg       0.87      0.87      0.87     22328
weighted avg       0.87      0.87      0.87     22328

              precision    recall  f1-score   support

           0       0.70      0.72      0.71      2722
           1       0.71      0.65      0.68      2504
           2       0.65      0.68      0.67      2217

    accuracy                           0.69      7443
   macro avg       0.69      0.69      0.68      7443
weighted avg       0.69      0.69      0.69      7443
```

In [158]:
```
plot_feature_importances(rf)
```



As for random forests, feature importances seem to maintain the same levels as for decision trees. Mainly being number_of_reviews, review_scores_accuracy, cleanliness, superhost title and communication. Of course, the feature importances graph has changed as a result of ensembling multiple decision trees.

XGBoost

XGBoost modeling.

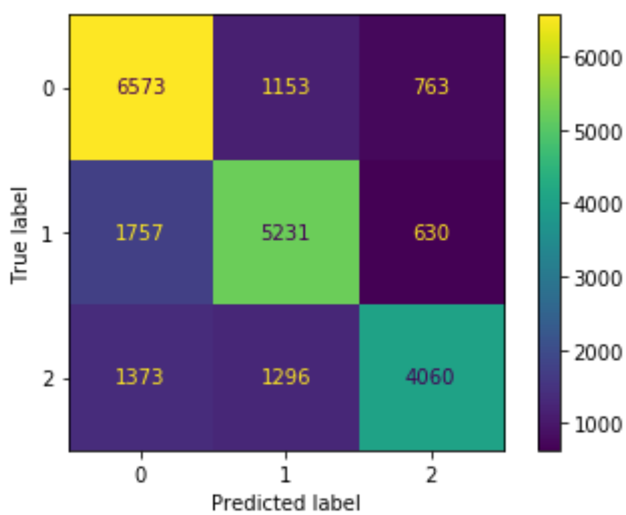
In [767]:
```
from xgboost import XGBClassifier
```

In [859]:
```
xgb = XGBClassifier()

# Fit classifier
xgb.fit(X_train, y_train)
```

Out[859]:
```
XGBClassifier(objective='multi:softprob')
```

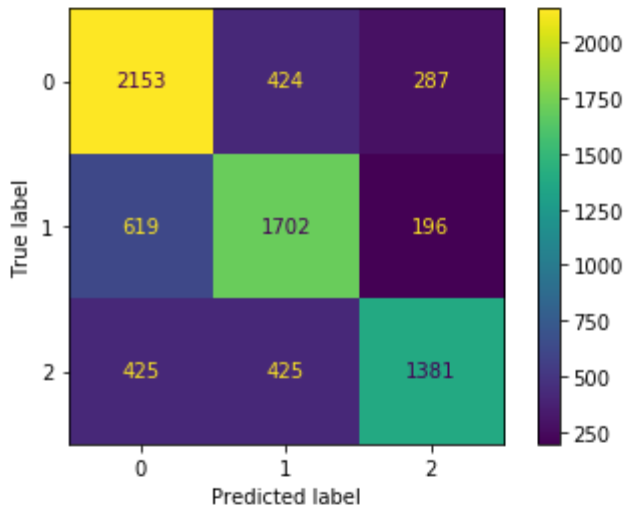
In [860]:
```
plot_confusion_matrix(xgb, X_train, y_train)
```

Out[860]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a7
42f0320>
```



In [861]:
```
plot_confusion_matrix(xgb, X_test, y_test)
```

Out[861]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1b4
a14fc18>
```



In [862]:
```
print(classification_report(y_train, xgb.predict(X_train)))
print(classification_report(y_test, xgb.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.68      0.77      0.72      8489
           1       0.68      0.69      0.68      7618
           2       0.74      0.60      0.67      6729

    accuracy                           0.69     22836
   macro avg       0.70      0.69      0.69     22836
weighted avg       0.70      0.69      0.69     22836

              precision    recall  f1-score   support

           0       0.67      0.75      0.71      2864
           1       0.67      0.68      0.67      2517
           2       0.74      0.62      0.67      2231

    accuracy                           0.69      7612
   macro avg       0.69      0.68      0.69      7612
weighted avg       0.69      0.69      0.69      7612
```

In [815]:
```
xgb_params = {
    'learning_rate': [0.1, 0.2],
    'max_depth': [6],
    'min_child_weight': [1, 2],
    'subsample': [0.5, 0.7],
    'n_estimators': [100],
}
```

In [823]:
```
gs_xgb = GridSearchCV(xgb, xgb_params, scoring='accuracy', cv=None, n_jobs=1)

gs_xgb.fit(X_train, y_train)

print(f"Optimal Parameters: {gs_xgb.best_params_}")
```

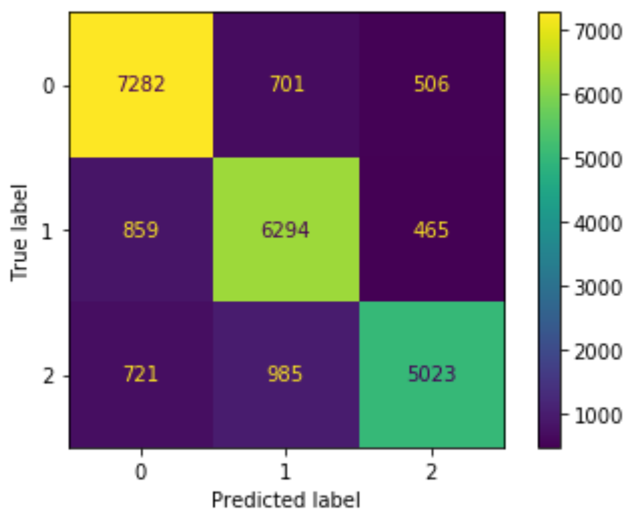
```
Optimal Parameters: {'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 100, 'subsample': 0.7}
```

In [871]:
```
xgb = XGBClassifier(max_depth=6, n_estimators=200)

# Fit classifier
xgb.fit(X_train, y_train)
```

Out[871]:
```
XGBClassifier(max_depth=6, n_estimators=200, objective='multi:softprob')
```

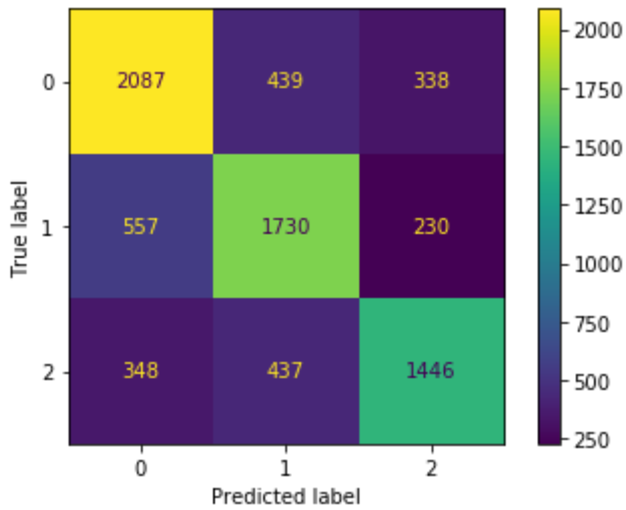
In [872]:
```
plot_confusion_matrix(xgb, X_train, y_train)
```

Out[872]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1aa792e9b0>
```



In [873]:
```
plot_confusion_matrix(xgb, X_test, y_test)
```

Out[873]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a7
820cb70>
```



In [874]:
```
print(classification_report(y_train, xgb.predict(X_train)))
print(classification_report(y_test, xgb.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84      8489
           1       0.79      0.83      0.81      7618
           2       0.84      0.75      0.79      6729

    accuracy                           0.81     22836
   macro avg       0.82      0.81      0.81     22836
weighted avg       0.82      0.81      0.81     22836

              precision    recall  f1-score   support

           0       0.70      0.73      0.71      2864
           1       0.66      0.69      0.68      2517
           2       0.72      0.65      0.68      2231

    accuracy                           0.69      7612
   macro avg       0.69      0.69      0.69      7612
weighted avg       0.69      0.69      0.69      7612
```