This notebook finds the features from `listings.csv` that are most correlated with the rating. (See end of notebook for list of these features.)

In [1]:
```
# TODO: better preprocessing of binary features
```

In [2]:
```
import pandas as pd
import numpy as np
import sklearn
import sklearn.preprocessing
import re
import pickle
```

In [3]:
```
train = pd.read_csv('data/listings.csv')

train.shape
```

```
/Users/ChentianJiang/miniconda3/envs/cs317/lib/python3.7/site-packages/IPython/c
ore/interactiveshell.py:2785: DtypeWarning: Columns (43,87,88) have mixed types.
Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[3]:
```
(50914, 96)
```

In [4]:
```
train.head()
```

Out[4]:

	id	listing_url	scrape_id	last_scraped	name	summa
0	2515	https://www.airbnb.com/rooms/2515	20180806171147	2018-08-07	Stay at Chez Chic budget room #1	Step int our artis spaciou apartme and ...
1	2539	https://www.airbnb.com/rooms/2539	20180806171147	2018-08-07	Clean & quiet apt home by the park	Renovat apt hom in elevat building
2	2595	https://www.airbnb.com/rooms/2595	20180806171147	2018-08-07	Skylit Midtown Castle	Find you romantic getaway to this beautifu ...
3	3330	https://www.airbnb.com/rooms/3330	20180806171147	2018-08-07	++ Brooklyn Penthouse Guestroom ++	This is a spaciou clean, furnishe master be...
4	3647	https://www.airbnb.com/rooms/3647	20180806171147	2018-08-07	THE VILLAGE OF HARLEM....NEW YORK !	NaN

5 rows × 96 columns

In [5]:
```
# global vars

# drop useless columns
DROP_COLS = set(["scrape_id",
                 "last_scraped",
                 "thumbnail_url",
                 "medium_url",
                 "picture_url",
                 "xl_picture_url",
                 "host_thumbnail_url",
                 "host_picture_url",
                 "host_total_listings_count",
                 "host_has_profile_pic",
                 "calendar_last_scraped",
                 "availability_30",
                 "availability_60",
                 "availability_90",
                 "availability_365",
                 "first_review",
                 "last_review"])

LABEL = 'review_scores_rating'  # TODO: is this the overall rating?
```

In [6]:
```
train_label = train[LABEL]
train.drop(LABEL, axis=1, inplace=True)
train.shape
```

Out[6]: (50914, 95)

In [7]:
```
train.drop(DROP_COLS, axis=1, inplace=True)
train.shape
```

Out[7]: (50914, 78)

In [8]:
```
# separate out the different *types* of columns
train_type_dict = train.columns.to_series().groupby(train.dtypes).groups
train_type_dict.keys()
```

Out[8]: dict_keys([dtype('int64'), dtype('float64'), dtype('O')])

In [9]:
```
def get_cat_cols(data_df, cat_colnames):
    # find categorical/character columns with <= 10 unique values
    # (features with too many categories is hard to work with)
    cols_cat = []

    for col in cat_colnames:
        if len(data_df[col].unique()) <= 10:  # 3 to include binary values as wel
l as NA/NaN/Null
            cols_cat.append(col)
    return cols_cat
```

In [10]:
```
def create_data_dict(data_df, type_dict, label_df):
    int_data = data_df[type_dict[np.dtype('int')]]
    float_data = data_df[type_dict[np.dtype('float')]]
    num_data = pd.concat([int_data, float_data], axis=1)

    cat_colnames = get_cat_cols(data_df, type_dict[np.dtype('object')])
    cat_data = data_df[cat_colnames]
    return {'num': num_data, 'cat':cat_data, 'label':label_df}
```

In [11]:
```
train_dict = create_data_dict(train, train_type_dict, train_label)

train_dict['num'].shape, train_dict['cat'].shape, train_dict['label'].shape
```

Out[11]:
```
((50914, 23), (50914, 20), (50914,))
```

In [12]:
```
DROP_COLS_NUM = set()
DROP_COLS_CAT = set()
```

In [13]:
```
# drop numerical columns with a high ratio (50%) of missing values

train_num_missing_cols = train_dict['num'].columns[train_dict['num'].isnull().any
()]
for col in train_num_missing_cols:
    if(train_dict['num'][col].isnull().sum() >= 0.5*train.shape[0]):
        DROP_COLS_NUM.add(col)

DROP_COLS_NUM
```

Out[13]:
```
{'host_acceptance_rate', 'square_feet'}
```

In [14]:
```
# doesn't make sense to predict overall review with other reviews --> drop review
-related numerical columns

train_num_review_cols = set([col for col in train_dict['num'].columns if re.searc
h('review', col)])
DROP_COLS_NUM = DROP_COLS_NUM.union(train_num_review_cols)

DROP_COLS_NUM
```

Out[14]:
```
{'host_acceptance_rate',
 'number_of_reviews',
 'review_scores_accuracy',
 'review_scores_checkin',
 'review_scores_cleanliness',
 'review_scores_communication',
 'review_scores_location',
 'review_scores_value',
 'reviews_per_month',
 'square_feet'}
```

In [15]:
```
train_dict['num'].drop(DROP_COLS_NUM, axis=1, inplace=True)
train_dict['num'].shape
```

Out[15]:
```
(50914, 13)
```

In [16]:
```
# fill in the numerical missing values with the column median
# TODO: better ways to impute missing numerical values?
def fill_na(num_df):
    missing_cols = num_df.columns[num_df.isnull().any()]
    for col in missing_cols:
        num_df[col].fillna(num_df[col].median(), inplace=True)

fill_na(train_dict['num'])
```

In [17]:
```
# drop categorical columns with a high ratio (50%) of missing values

train_cat_missing_cols = train_dict['cat'].columns[train_dict['cat'].isnull().any
()]
for col in train_cat_missing_cols:
    if(train_dict['cat'][col].isnull().sum() >= 0.5*train.shape[0]):
        DROP_COLS_CAT.add(col)

DROP_COLS_CAT
```

Out[17]:
```
{'jurisdiction_names', 'license'}
```

In [18]:
```
train_dict['cat'].drop(DROP_COLS_CAT, axis=1, inplace=True)
train_dict['cat'].shape
```

```
/Users/ChentianJiang/miniconda3/envs/cs317/lib/python3.7/site-packages/pandas/co
re/frame.py:3697: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/indexing.html#indexing-view-versus-copy
  errors=errors)
```

Out[18]:
```
(50914, 18)
```

In [19]:
```
train_dict['cat'] = train_dict['cat'].astype('str') # consistent type
```

In [20]:
```
# one-hot encode categorical features

train_dict['cat'] = pd.get_dummies(train_dict['cat']) # this also encodes NaN/Nul
l/NA etc. as its own category
train_dict['cat'].shape
```

Out[20]:
```
(50914, 57)
```

In [21]:
```
# normalize (put into range [0,1]) numerical features
train_num = sklearn.preprocessing.normalize(train_dict['num'], axis=0)
train_num = pd.DataFrame(train_num, columns=train_dict['num'].columns)

# standardize (center to the mean and scale to unit variance) numerical features
train_num = sklearn.preprocessing.scale(train_num, axis=0)
train_num = pd.DataFrame(train_num, columns=train_dict['num'].columns)
```

In [22]:
```
# put together numerical and categorical data (separated from labels)
train_cat = train_dict['cat'].astype('int')
train_data = pd.concat([train_num, train_cat], axis=1)

train_label.shape, train_data.shape
```

Out[22]:
```
((50914,), (50914, 70))
```

In [23]:
```
# impute missing label value with median
# TODO: better impute method?
train_dict['label'].fillna(train_dict['label'].median(), inplace=True)
```

In [24]:
```
# choose features that are relatively more correlated with the label

corrDict = dict()
for col in train_data.columns:
    train_data[col].dtype
    corr = abs(train_data[col].corr(train_label))  # "computes pairwise correlati
on of columns, excluding NA/null values"
    if corr > 0.02:  # most features are barely correlated with the label
        corrDict[col] = corr

# save chosen features
with open('corrDict.pickle', 'wb') as f:
    pickle.dump(corrDict, f)
len(corrDict)  # number of features with correlation above a certain threshold
```

Out[24]:
```
19
```

In [25]:
```
corrDict # top correlated (with label) features with correlation above some thres
hold
```

Out[25]:
```
{'id': 0.04697486373512992,
 'host_id': 0.03647428871160532,
 'accommodates': 0.032057733695012615,
 'calculated_host_listings_count': 0.026567959389538032,
 'longitude': 0.023159218757749976,
 'beds': 0.03612508486889648,
 'host_is_superhost_f': 0.14965465554289106,
 'host_is_superhost_t': 0.14973725471109212,
 'host_identity_verified_f': 0.02716568689308185,
 'host_identity_verified_t': 0.027181575202442035,
 'neighbourhood_group_cleansed_Brooklyn': 0.03238031945441595,
 'neighbourhood_group_cleansed_Manhattan': 0.027002073378334608,
 'room_type_Entire home/apt': 0.03422772761577601,
 'room_type_Private room': 0.03287318388260937,
 'instant_bookable_f': 0.060519618291476764,
 'instant_bookable_t': 0.060519618291476764,
 'cancellation_policy_flexible': 0.021684854503989923,
 'cancellation_policy_moderate': 0.03486031433140535,
 'cancellation_policy_strict_14_with_grace_period': 0.04845450263351716}
```