In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [9]:
```
data = pd.read_csv('data/listings.csv')
```

```
/home/twang/anaconda3/lib/python3.6/site-packages/IPython/core/interactiveshel
l.py:3020: DtypeWarning: Columns (61,87,88) have mixed types. Specify dtype opt
ion on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [29]:
```
# Concat text columns
text = data[['name', 'summary', 'description']].astype('U').apply(lambda x:
```

In [31]:
```
# Clean text, get rid of punctuations and numbers, and lower letter case
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

def Clean_text(str_input):
    from re import sub
    # from nltk.stem.porter import PorterStemmer
    from string import punctuation

    # Remove punctuations
    str_input = ''.join((char for char in str_input if char not in punctuati

    # Remove any word containing a number (of which there are many)
    words = sub(u'(?ui)\\b[a-zA-Z0-9]*[0-9]+[a-zA-Z0-9]*\\b', " ", str_input

    # porter_stemmer=PorterStemmer()
    # words = [porter_stemmer.stem(word) for word in words if not word in st
    return " ".join(word for word in words if not word in stop_words)
```

In [32]:
```
text=text.apply(Clean_text)
```

In [41]:
```
# Vectorize using tfidf

from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline

tfidf = TfidfVectorizer(max_df=0.9, \
                             min_df=0.01,\
                             ngram_range=(1,3))
svd = TruncatedSVD(n_components=100)
pipeline = make_pipeline(tfidf, svd)

```

In [43]:
```
text_vec = pipeline.fit_transform(text)
```

In [109]:
```
text_vec_df = pd.DataFrame(text_vec, index=data.id)
```

In [49]:
```
# Now, lets select dimensions most correlated with

prices = data.price.apply(lambda x: np.NaN if x == 'NaN' else float(str(x).l
```

In [125]:
```
corrs = []

for column in text_vec_df:
    corrs.append((column, np.corrcoef(text_vec_df[column], prices)[0,1]))
```

In [126]:
```
corrs
```

Out[126]:
```
[(0, -0.06648791206665025),
 (1, -0.028831301490426873),
 (2, 0.1376116399887893),
 (3, -0.14064679796341056),
 (4, -0.015045949884558632),
 (5, -0.017338075788816746),
 (6, -0.15018636924743195),
 (7, -0.03251137010051627),
 (8, -0.03402721371523929),
 (9, 0.002502574276354029),
 (10, -0.017603960019613424),
 (11, -0.012825034604486103),
 (12, 0.015011891979576192),
 (13, -0.008502979562095274),
 (14, -0.021421223808995887),
 (15, 0.08916933204700668),
 (16, 0.03600523544519698),
 (17, 0.05221572799303739),
 (18, 0.07873021129446113),
```

In [127]:
```
# Take the top 10 most correlated columns by absolute value
best_cols = [item[0] for item in sorted([(item[0], abs(item[1])) for item in
```

In [128]:
```
best_cols
```

Out[128]:
```
[6, 3, 2, 15, 18, 0, 24, 17, 20, 27]
```

Quick analysis of the most useful dimensions of svd

NOTE: Check with people knowledgeable with the matter. I am not sure what is the correct interpretation of weights in a factor.

In [102]:
```
vocab = [item[0] for item in sorted(tfidf.vocabulary_.items(), key = operato
```

In [131]:
```
# The best column has over 0.1 correlation
voc_weight_6 = sorted(zip(vocab, svd.components_[6,:]), key = lambda x:x[1],
```

In [135]:
```
voc_weight_6[-20:]
```

Out[135]:
```
[('center', -0.07272840672472593),
 ('high', -0.07314626498261155),
 ('brownstone', -0.07491977130806952),
 ('studio', -0.0783623473510409),
 ('beautiful', -0.08270145726799635),
 ('luxury', -0.08282724873363825),
 ('garden', -0.08313558721756222),
 ('new york city', -0.08584393989391555),
 ('york city', -0.08585334043852712),
 ('views', -0.09038845236272569),
 ('building', -0.09170300233121975),
 ('modern', -0.09791023397410266),
 ('city', -0.09839316504879354),
 ('prospect park', -0.10030319935963242),
 ('prospect', -0.12129553554707352),
 ('loft', -0.12883929977176176),
 ('new york', -0.15294873544812715),
 ('york', -0.1537952360737561),
 ('new', -0.20245435856556063),
 ('brooklyn', -0.23794178814603853)]
```

In [141]:
```
# Top 20 words by weight
vocab_weights = []

for col in best_cols:
    vocab_weights.append(sorted(zip(vocab, svd.components_[col,:]), key = la
```

Hopefully from the loadings in these best factors we can gain insight into what makes an apartment pricey.

In [142]:
```
# Select best columns
text_vec_df = text_vec_df[best_cols]
```

In [143]:
```
text_vec_df.columns = ['descVec_'+str(x) for x in best_cols]
```

In [162]:
```
# Join to cleaned dataset
cleaned_data = pd.read_csv('data/cleaned_with_nlp_listings_2.csv')
```

In [146]:
```
cleaned_with_desc = pd.merge(cleaned_data, text_vec_df.reset_index(), on='id
```

In [147]:
```
cleaned_with_desc.isnull().sum()
```

Out[147]:
```
id                               0
host_response_rate               0
host_is_superhost                0
host_listings_count              0
host_total_listings_count        0
host_has_profile_pic             0
host_identity_verified           0
is_location_exact                0
accommodates                     0
bathrooms                        0
bedrooms                         0
beds                             0
price                            0
security_deposit                 0
cleaning_fee                     0
guests_included                  0
extra_people                     0
minimum_nights                   0
maximum_nights                   0
```

In [151]:
```
%%time
# Make imputer for the missing values using KNN

from fancyimpute import KNN

cleaned_with_desc_filled = KNN(k=5).fit_transform(cleaned_with_desc)
```

```
Imputing row 1/39926 with 10 missing, elapsed time: 524.312
Imputing row 101/39926 with 0 missing, elapsed time: 524.319
Imputing row 201/39926 with 0 missing, elapsed time: 524.324
Imputing row 301/39926 with 0 missing, elapsed time: 524.328
Imputing row 401/39926 with 0 missing, elapsed time: 524.334
Imputing row 501/39926 with 0 missing, elapsed time: 524.343
Imputing row 601/39926 with 0 missing, elapsed time: 524.350
Imputing row 701/39926 with 0 missing, elapsed time: 524.354
Imputing row 801/39926 with 0 missing, elapsed time: 524.359
Imputing row 901/39926 with 0 missing, elapsed time: 524.363
Imputing row 1001/39926 with 0 missing, elapsed time: 524.365
Imputing row 1101/39926 with 0 missing, elapsed time: 524.372
Imputing row 1201/39926 with 0 missing, elapsed time: 524.374
Imputing row 1301/39926 with 0 missing, elapsed time: 524.383
Imputing row 1401/39926 with 0 missing, elapsed time: 524.387
Imputing row 1501/39926 with 0 missing, elapsed time: 524.391
Imputing row 1601/39926 with 0 missing, elapsed time: 524.398
Imputing row 1701/39926 with 0 missing, elapsed time: 524.404
Imputing row 1801/39926 with 10 missing, elapsed time: 524.406
```

In [159]:
```
cleaned_with_desc_filled = pd.DataFrame(cleaned_with_desc_filled, columns=cl
```

In [160]:
```
cleaned_with_desc_filled.to_csv('cleaned_with_desc_filled.csv')
```

In [161]:
```
del cleaned_with_desc, data
```

Model building

In [175]:
```
# Make a pipeline to train with random search tuning

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
import xgboost as xgb
```

Baseline: see performance without appended columns

In [164]:
```
cleaned_data.price.head()
```

Out[164]:
```
0    137.0
1    149.0
2    225.0
3     70.0
4     89.0
Name: price, dtype: float64
```

In [ ]:
```
cleaned_data.set_index('id', inplace=True)
```

In [176]:
```
scaler_baseline = StandardScaler()
```

In [166]:
```
y_baseline = cleaned_data.price
X_baseline = cleaned_data.loc[:, cleaned_data.columns != 'price']
```

In [177]:
```
fit_baseline = xgb.XGBRegressor(objective='reg:linear', random_state = 23)
cv_baseline = cross_val_score(fit_baseline, scaler_baseline.fit_transform(X_
```

```
/home/twang/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/data.p
y:625: DataConversionWarning: Data with input dtype int64, float64 were all con
verted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/twang/anaconda3/lib/python3.6/site-packages/sklearn/base.py:462: DataConv
ersionWarning: Data with input dtype int64, float64 were all converted to float
64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [178]:
```
np.sqrt(-cv_baseline.mean())
```

Out[178]:
```
171.53075105041998
```

Lift: with new columns, see how much lift is generated

In [171]:
```
cleaned_with_desc_filled.set_index('id', inplace=True)
```

In [179]:
```
scaler_lift = StandardScaler()
```

In [172]:
```
y_lift = cleaned_with_desc_filled.price
X_lift = cleaned_with_desc_filled.loc[:, cleaned_with_desc_filled.columns !=
```

In [181]:
```
fit_lift = xgb.XGBRegressor(objective='reg:linear', random_state = 23)
cv_lift = cross_val_score(fit_lift, scaler_lift.fit_transform(X_lift), y_lif
```

In [182]:
```
np.sqrt(-cv_lift.mean())
```

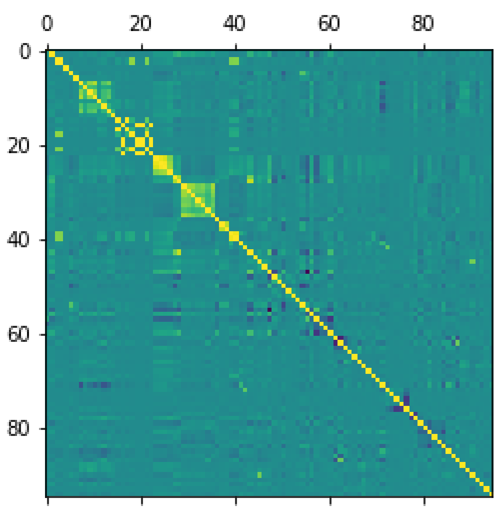
Out[182]:
```
172.6613213053378
```

Conclusion

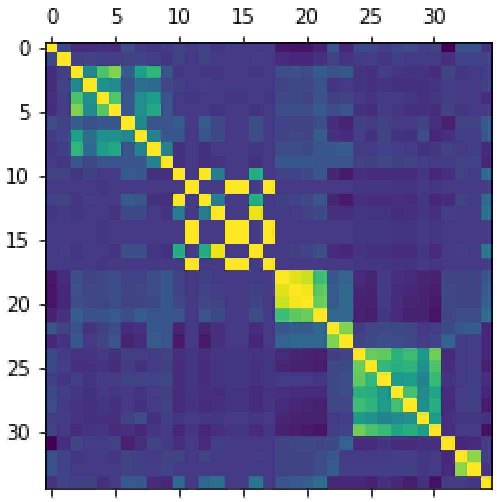
It does seem that the new columns did not add predictive power to the model. This is surprising, but understandable, as these new vectors did not really have a strong correlation with price (in the 10% range). It seems we just don't have enough information in these vectors to really explain variations in price.

I also looked at collinearity and found that we can further process the data to remove some of the super highly collinear columns potentially, and this might merit further investigation. Perhaps these could be result of mistakes in data cleaning.

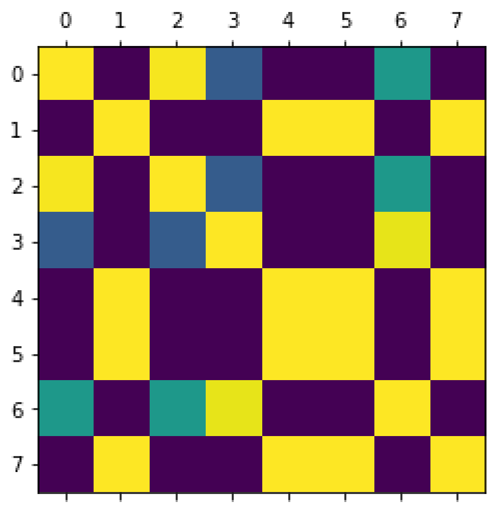
In [184]:
```
plt.matshow(X_lift.corr())
plt.show()
```



In [186]:
```
plt.matshow(X_lift.iloc[:, 5:40].corr())
plt.show()
```



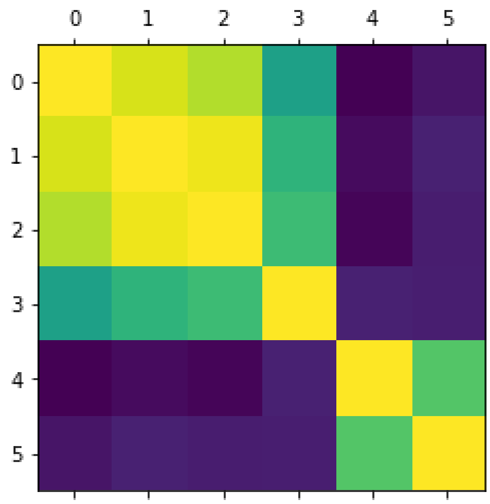
In [188]:
```
plt.matshow(X_lift.iloc[:, 15:23].corr())
plt.show()
```



In [189]:
```
X_lift.iloc[:, 15:23].columns
```

Out[189]:
```
Index(['minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
       'maximum_minimum_nights', 'minimum_maximum_nights',
       'maximum_maximum_nights', 'minimum_nights_avg_ntm',
       'maximum_nights_avg_ntm'],
      dtype='object')
```

In [192]:
```
plt.matshow(X_lift.iloc[:, 23:29].corr())
plt.show()
```



In [193]:
```
X_lift.iloc[:, 23:29].columns
```

Out[193]:
```
Index(['availability_30', 'availability_60', 'availability_90',
       'availability_365', 'number_of_reviews', 'number_of_reviews_ltm'],
      dtype='object')
```

In [ ]:
```

```