CS 109A/AC 209A/STAT 121A Data Science: Airbnb Project

Milestone #3 - Data Exploration

Harvard University
Fall 2016
Team: Stephen Camera-Murray, Himani Garg, and Vijay Thangella
TF: Christine Hwang

Due Date: Wednesday, November 5th, 2016 at 11:59pm

Data Exploration

We begin by loading the datasets:

- listings.csv.gz - the New York City Airbnb listing data from January 2015
- calendar.csv.gz - listing prices for specific dates-- to be analyzed for seasonality

Import libraries

In [2]:
```
# import libraries
import warnings
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

# suppress warnings
warnings.filterwarnings ( 'ignore' )
%matplotlib inline
```

Load and inspect the data

In [3]:
```
# load listings data into a pandas df
listingsDF = pd.read_csv ( './datasets/listings.csv.gz' )

# display the first two rows
listingsDF.head ( n = 2 )
```

Out[3]:

	id	scrape_id	last_scraped	name	picture_url
0	1069266	20150101184336	2015-01-02	Stay like a real New Yorker!	https://a0.muscache.com/pictures/5
1	1846722	20150101184336	2015-01-02	Apartment 20 Minutes Times Square	https://a1.muscache.com/pictures/3

2 rows × 52 columns

In [4]:
```
listingsDF.shape
```

Out[4]:
```
(27392, 52)
```

The listings dataset is the main dataset we'll be using for prediction. It has 27,392 listings and 52 columns. As we can see from the names below, not all columns are suitable for prediction, but many of them are.

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

Out[5]:
```
Index([u'id', u'scrape_id', u'last_scraped', u'name', u'picture_url',
       u'host_id', u'host_name', u'host_since', u'host_picture_url', u'stree
t',
       u'neighbourhood', u'neighbourhood_cleansed', u'city', u'state',
       u'zipcode', u'market', u'country', u'latitude', u'longitude',
       u'is_location_exact', u'property_type', u'room_type', u'accommodates',
       u'bathrooms', u'bedrooms', u'beds', u'bed_type', u'square_feet',
       u'price', u'weekly_price', u'monthly_price', u'guests_included',
       u'extra_people', u'minimum_nights', u'maximum_nights',
       u'calendar_updated', u'availability_30', u'availability_60',
       u'availability_90', u'availability_365', u'calendar_last_scraped',
       u'number_of_reviews', u'first_review', u'last_review',
       u'review_scores_rating', u'review_scores_accuracy',
       u'review_scores_cleanliness', u'review_scores_checkin',
       u'review_scores_communication', u'review_scores_location',
       u'review_scores_value', u'host_listing_count'],
      dtype='object')
```

The calendar dataset has pricing data for listings at different times of the week. We'll look at this one to see if there are different prices for seasonality.

Note: we added a buffer column to the end of the dataset since there's a bit of messiness in the data.

In [6]:
```
# load listings data into a pandas df
calendarDF = pd.read_csv ( './datasets/calendar.csv.gz', skiprows = 1, usecols
 = [ 0, 1, 2, 3, 4 ], names = [ 'listing_id', 'date', 'available', 'price', 'b
uffer' ] )

# display the first two rows
calendarDF.head ( n = 2 )
```

Out[6]:

	listing_id	date	available	price	buffer
0	3604481	2015-01-01	t	$600.00	NaN
1	3604481	2015-01-02	t	$600.00	NaN

In [7]:
```
calendarDF.shape
```

Out[7]:
```
(9998080, 5)
```

Data Cleansing

Listing dataset

We notice that some listings have weekly and monthly rates, but almost all listings have a daily rate. We start by converting the price column to a float so we can work with it as a number. Next, we remove all columns that we don't plan to use for prediction: 'scrape_id', 'last_scraped', 'picture_url', 'host_id', 'host_name', 'host_picture_url', 'street', 'neighbourhood', 'city', 'state', 'zipcode', 'market', 'country', 'latitude', 'longitude', 'is_location_exact', 'weekly_price', 'monthly_price', 'extra_people', 'calendar_updated', 'calendar_last_scraped'.

In [8]:

```
# convert the price column to a float
listingsDF [ 'price' ] = calendarDF [ 'price' ].replace ( '[\$,)]', '', regex
= True ).replace ( '[(]', '-', regex = True ).astype ( float )

# drop columns we don't need
listingsDF.drop ( [ 'scrape_id', 'last_scraped', 'picture_url', 'host_id', 'ho
st_name', 'host_picture_url', 'street', 'neighbourhood', 'city', 'state', 'zip
code', 'market', 'country', 'latitude', 'longitude', 'is_location_exact', 'wee
kly_price', 'monthly_price', 'extra_people', 'calendar_updated', 'calendar_las
t_scraped' ], axis = 1, inplace = True )

# rename the neighbourhood_cleansed column to neighborhood
listingsDF = listingsDF.rename ( columns = { 'neighbourhood_cleansed' : 'neigh
borhood' } )

# display the first two rows
listingsDF.head ( n = 2 )
```

Out[8]:

	id	name	host_since	neighborhood	property_type	room_type	accommc
0	1069266	Stay like a real New Yorker!	2013-04-10	Midtown East	Apartment	Entire home/apt	2
1	1846722	Apartment 20 Minutes Times Square	2012-06-13	Hamilton Heights	Apartment	Entire home/apt	10

2 rows × 31 columns

Next, we'll convert all of the date fields to an integer representing the month age so we can better work with them.

Note: the first_review and last_review columns may be blank. If they are we'll penalize them by setting it to the "worst" value in the data based on our assumption that the worst case for a first review is it's recent, or the lastest date in our data, and the last review is old, or the earliest date in our data.

In [9]:
```
# fill in missing dates with the "worst" value
listingsDF [ "first_review" ][ listingsDF [ "first_review" ].isnull() ] = list
ingsDF [ "first_review" ][ listingsDF [ "first_review" ].notnull() ].max()
listingsDF [ "last_review"  ][ listingsDF [ "last_review"  ].isnull() ] = list
ingsDF [ "last_review"  ][ listingsDF [ "last_review"  ].notnull() ].min()

# create new date fields based on a months passed
listingsDF [ "months_as_host" ] = ( ( 2014 - listingsDF [ "host_since" ].str [
 : 4 ].astype ( int ) ) * 12 ) + ( 13 - listingsDF [ "host_since" ].str [ 5 :
7 ].astype ( int ) )
listingsDF [ "months_since_first_review" ] = ( ( 2014 - listingsDF [ "first_re
view" ].str [ : 4 ].astype ( int ) ) * 12 ) + ( 13 - listingsDF [ "first_revie
w" ].str [ 5 : 7 ].astype ( int ) )
listingsDF [ "months_since_last_review"  ] = ( ( 2014 - listingsDF [ "last_rev
iew"  ].str [ : 4 ].astype ( int ) ) * 12 ) + ( 13 - listingsDF [ "last_revie
w"  ].str [ 5 : 7 ].astype ( int ) )

# drop columns we don't need
listingsDF.drop ( [ 'host_since', 'first_review', 'last_review' ], axis = 1, i
nplace = True )

# display the first two rows
listingsDF.head ( n = 2 )
```

Out[9]:

	id	name	neighborhood	property_type	room_type	accommodates	bathr
0	1069266	Stay like a real New Yorker!	Midtown East	Apartment	Entire home/apt	2	1.0
1	1846722	Apartment 20 Minutes Times Square	Hamilton Heights	Apartment	Entire home/apt	10	1.0

2 rows × 31 columns

Calendar listing data

We start by converting the price to a float so we can work with it as a number and drop the buffer column.

In [10]:
```
# convert the price column to a float
calendarDF [ 'price' ] = calendarDF [ 'price' ].replace ( '[\$,)]', '', regex
= True ).replace ( '[(]', '-', regex = True ).astype ( float )

# drop the buffer column
calendarDF.drop ( [ 'buffer' ], axis = 1, inplace = True )

# display the first two rows
calendarDF.head ( n = 2 )
```

Out[10]:

	listing_id	date	available	price
0	3604481	2015-01-01	t	600.0
1	3604481	2015-01-02	t	600.0

Next, we add a seasonal column by extracting the month and applying some simple logic.

In [11]:
```
# create a season column for the calendar listing prices by: 1) extract the mo
nth to a separate column,
# and 2) create a season column based on the month
calendarDF [ "month" ] = calendarDF [ "date" ].str [ 5 : 7 ].astype ( int )
calendarDF [ "season" ] = "Winter"
calendarDF [ "season" ][ ( calendarDF [ 'month' ] >= 3 ) &  ( calendarDF [ 'mo
nth' ] <= 5 ) ] = "Spring"
calendarDF [ "season" ][ ( calendarDF [ 'month' ] >= 6 ) &  ( calendarDF [ 'mo
nth' ] <= 8 ) ] = "Summer"
calendarDF [ "season" ][ ( calendarDF [ 'month' ] >= 9 ) &  ( calendarDF [ 'mo
nth' ] <= 11 ) ] = "Fall"

# display the first two rows
calendarDF.head ( n = 2 )
```

Out[11]:

	listing_id	date	available	price	month	season
0	3604481	2015-01-01	t	600.0	1	Winter
1	3604481	2015-01-02	t	600.0	1	Winter

Finally, we group the listings and seasons to get the mean listing price for the season.

In [12]:
```
# get rid of rows with blank or meaningless prices
calendarDF = calendarDF [ np.isfinite ( calendarDF [ 'price' ] ) ]

# create a seasonal dataframe with average seasonal pricing for each listing
seasonalDF = calendarDF.groupby ( [ 'listing_id', 'season' ] ) [ 'price' ].mea
n().to_frame ( name = 'price' ).reset_index()

seasonalDF.head ( n = 2 )
```

Out[12]:

	listing_id	season	price
0	105	Fall	363.285714
1	105	Spring	360.956522

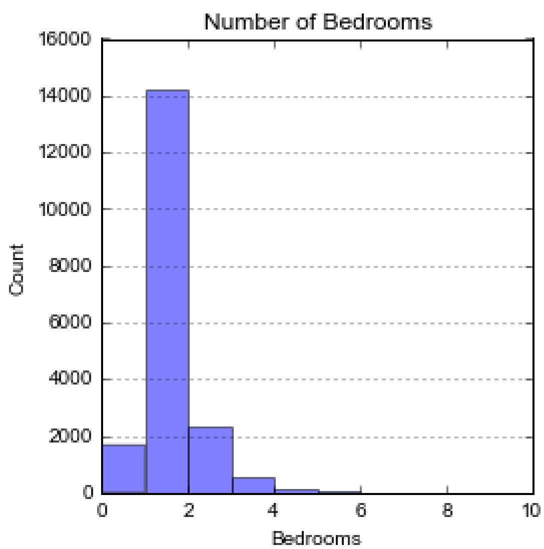
Visualization and Analysis

Pricing Categories

We'll begin by looking at the distribution for price to see if there are any obvious ranges we might choose. We begin by looking at the number of bedrooms and room types to see if New York City has similar phenomenon as San Francisco where most listings are for one bedroom and not shared rooms.

In [13]:
```
# a little cleanup first: remove rows without price or bedroom data
listingsDF = listingsDF [ np.isfinite ( listingsDF [ "bedrooms" ] ) & np.isfin
ite ( listingsDF [ "price" ] ) ]
```

In [14]:
```
# set up our visualization
fig, ax = plt.subplots ( 1, 1, figsize = ( 4, 4 ) )

# set the style
plt.style.use ( [ 'seaborn-white', 'seaborn-muted' ] )
matplotlib.rc ( "font", family = "Times New Roman" )

# create histogram
ax.hist ( listingsDF [ "bedrooms" ], alpha = 0.5 )

# set labels
ax.set_title  ( "Number of Bedrooms" )
ax.yaxis.grid ( True )
ax.set_xlabel ( "Bedrooms" )
ax.set_ylabel ( "Count" )

# display plot
plt.tight_layout()
plt.show()
```



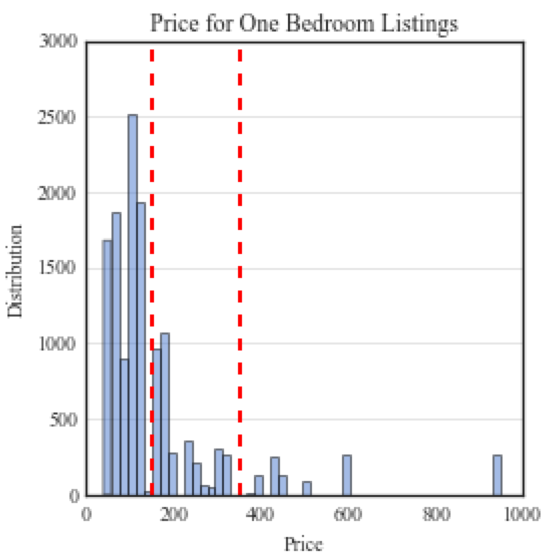
As we suspected, the vast majority of the listings are one bedroom. Let's filter the data to only include those for prediction and look at our price distribution. But we also, look at the room type as well to see if we should filter out shared rooms, which are also likely to be less common.

In [15]:
```
listingsDF.groupby ( [ 'room_type' ] ) [ 'room_type' ].count().to_frame ( name
 = 'count' ).reset_index()
```

Out[15]:

	room_type	count
0	Entire home/apt	11001
1	Private room	7348
2	Shared room	572

In [16]:
```
# filter out non-one bedroom listings
listingsDF = listingsDF [ ( listingsDF [ "bedrooms" ] == 1 ) & ( listingsDF [ 
"room_type" ] != "Shared room" ) ]

# set up our visualization
fig, ax = plt.subplots ( 1, 1, figsize = ( 4, 4 ) )

# create histogram
ax.hist ( listingsDF [ "price" ], alpha = 0.5, bins = 50 )

# set labels
ax.set_title  ( "Price for One Bedroom Listings" )
ax.yaxis.grid ( True )
ax.set_xlabel ( "Price" )
ax.set_ylabel ( "Distribution" )

# set our price grouping cutoffs
plt.axvline ( 150, color = 'r', linestyle = 'dashed', linewidth = 2 )
plt.axvline ( 350, color = 'r', linestyle = 'dashed', linewidth = 2 )

# display plot
plt.tight_layout()
plt.show()
```



Keeping in mind that our goal is to provide pricing guidance to new owners who wish to list their property, we come up with three price groupings based on our data to have a nice balance between the groups **and** user-friendly ranges. We can see the split in the histogram as:

- Low: Up to $150
- Mid: $150 - $350
- High: Over $350

Listing Predictors

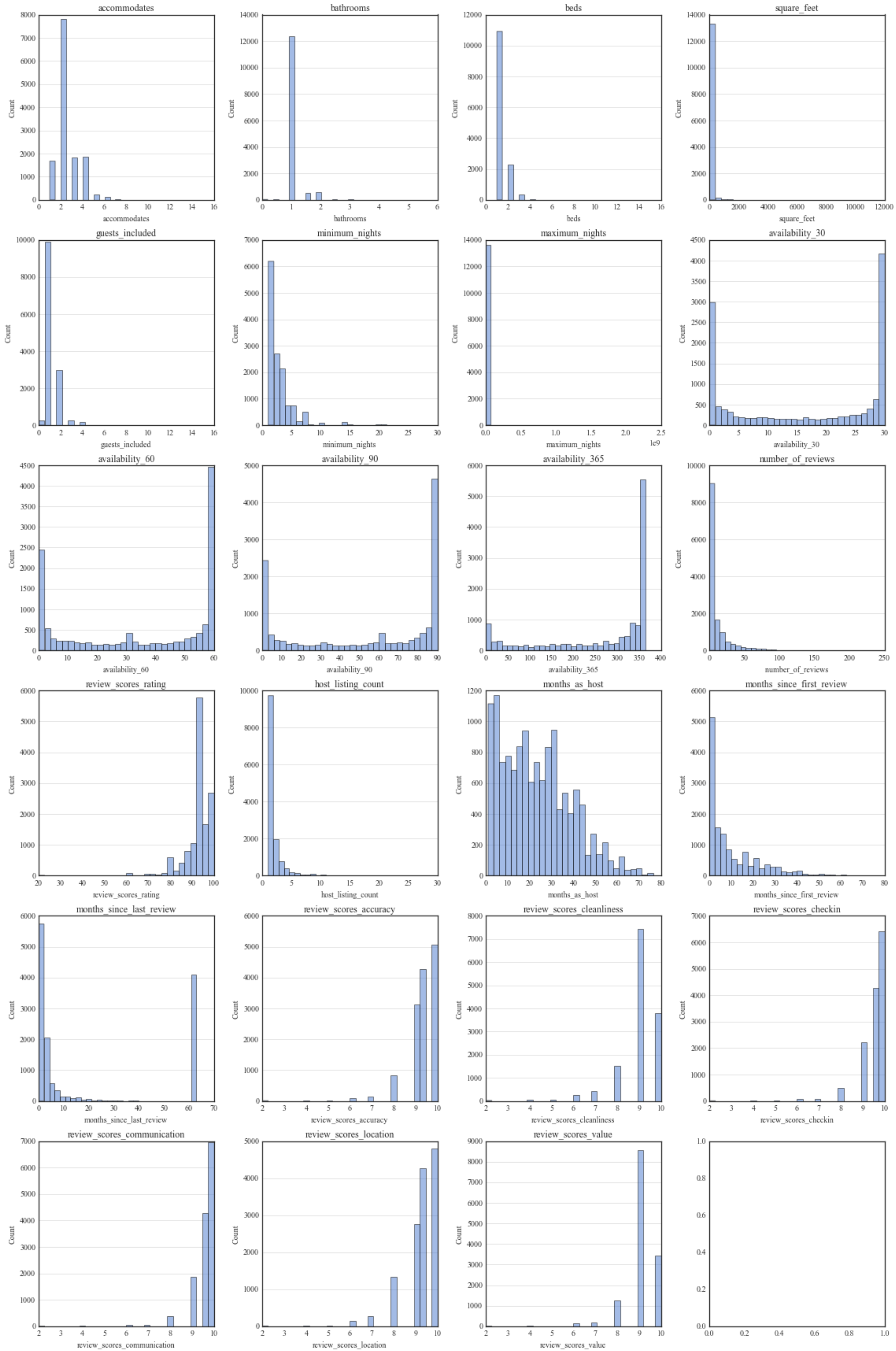
Predictor Distribution

Now, let's look at the distributions of our numeric predictors. But, first, we'll need to do a little cleanup for missing values, setting them to something we believe is appropriate for now.

In [17]:
```
# set up our the list of columns to visualize
cols = [ "accommodates", "bathrooms", "beds", "square_feet",
"guests_included", "minimum_nights", "maximum_nights"
        ,"availability_30", "availability_60", "availability_90", "availabilit
y_365", "number_of_reviews"
        ,"review_scores_rating", "host_listing_count", "months_as_host", "mont
hs_since_first_review", "months_since_last_review"
        ,"review_scores_accuracy", "review_scores_cleanliness", "review_scores
_checkin", "review_scores_communication"
        ,"review_scores_location", "review_scores_value" ]

# fill in missing values for certain columns with 1 or 0, depending
listingsDF [ "bathrooms" ][ listingsDF [ "bathrooms" ].isnull() ] = 1  # assum
e 1 if missing
listingsDF [ "beds" ][ listingsDF [ "beds" ].isnull() ] = 1  # assume 1 if mis
sing
listingsDF [ "square_feet" ][ listingsDF [ "square_feet" ].isnull() ] = 1  # c
an't assum here, set to 0 so it stands out in the viz

# loop through the cols and fill in any other missing value with the mean
for i in cols:
    listingsDF [ i ][ listingsDF [ i ].isnull() ] = listingsDF [ i ][ listings
DF [ i ].notnull() ].mean()  # set to mean if missing
```

In [18]:

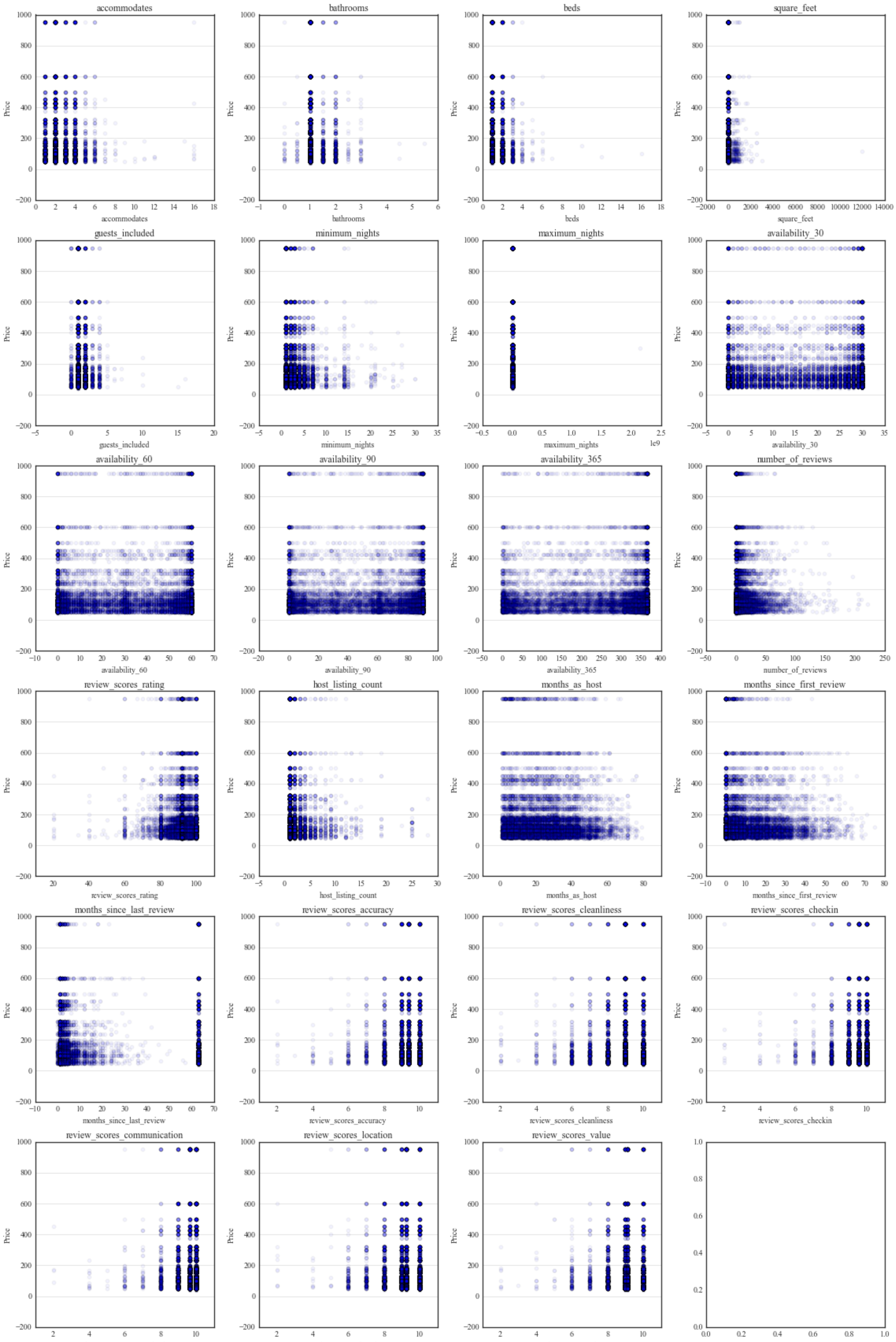
```
# set up our visualization
fig, ax = plt.subplots ( 6, 4, figsize = ( 16, 24 ) )

# loop through cols
for i in range ( len ( cols ) ):

    # create histogram
    ax [ ( i / 4 ), ( i % 4 ) ].hist ( listingsDF [ cols [ i ] ], alpha = 0.5,
 bins = 30 )

    # set labels
    ax [ ( i / 4 ), ( i % 4 ) ].set_title  ( cols [ i ] )
    ax [ ( i / 4 ), ( i % 4 ) ].set_xlabel ( cols [ i ] )
    ax [ ( i / 4 ), ( i % 4 ) ].set_ylabel ( "Count" )
    ax [ ( i / 4 ), ( i % 4 ) ].yaxis.grid ( True )

# display plot
plt.tight_layout()
plt.show()
```


accommodates
bathrooms
beds
square_feet
guests_included
minimum_nights
maximum_nights
availability_30
availability_60
availability_90
availability_365
number_of_reviews
review_scores_rating
host_listing_count
months_as_host
months_since_first_review
months_since_last_review
review_scores_accuracy
review_scores_cleanliness
review_scores_checkin
review_scores_communication
review_scores_location
review_scores_value
Count
1e9

From our numeric predictor distributions we can see that some of them may not have enough variety to make them meaningful for prediction. For instance, square_feet and maximum_nights are missing for most (defaulted to zero) and bathrooms are 1 for most listings. We'll likely drop them for making our predictions.

Price vs. Individual numeric predictors

In [19]:
```
# set up our visualization
fig, ax = plt.subplots ( 6, 4, figsize = ( 16, 24 ) )

# loop through cols
for i in range ( len ( cols ) ):

    # create histogram
    ax [ ( i / 4 ), ( i % 4 ) ].scatter ( listingsDF [ cols [ i ] ], listingsD
F [ "price" ], alpha = 0.05 )

    # set labels
    ax [ ( i / 4 ), ( i % 4 ) ].set_title  ( cols [ i ] )
    ax [ ( i / 4 ), ( i % 4 ) ].set_xlabel ( cols [ i ] )
    ax [ ( i / 4 ), ( i % 4 ) ].set_ylabel ( "Price" )
    ax [ ( i / 4 ), ( i % 4 ) ].yaxis.grid ( True )

# display plot
plt.tight_layout()
plt.show()
```

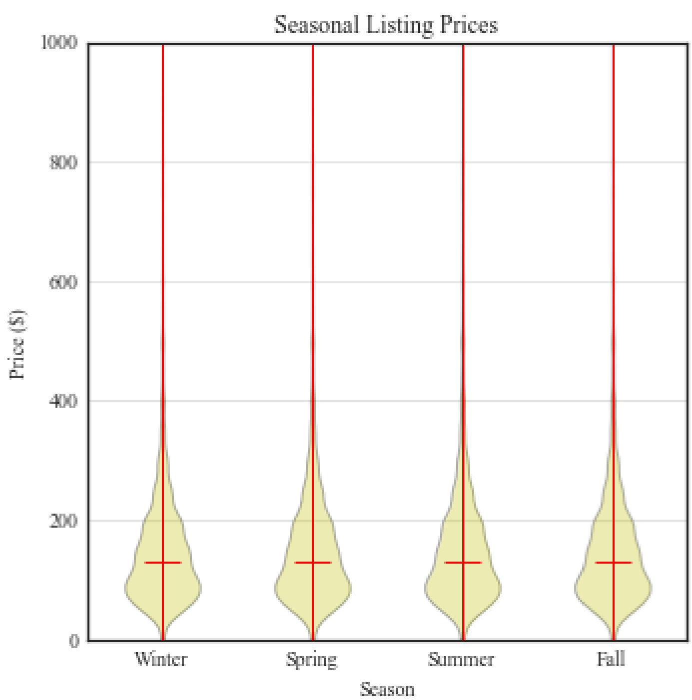

accommodates
bathrooms
beds
square_feet
guests_included
minimum_nights
maximum_nights
1e9
availability_30
availability_60
availability_90
availability_365
number_of_reviews
review_scores_rating
host_listing_count
months_as_host
months_since_first_review
months_since_last_review
review_scores_accuracy
review_scores_cleanliness
review_scores_checkin
review_scores_communication
review_scores_location
review_scores_value
Price

We see some general shape for many of our scatterplots of price vs. individual numeric predictors, indicating there is an association for many of them and they should be included in our model.

Seasonality

Next, we'll look at the effect on seasonality on listing prices.

In [20]:
```
# parse out the seasonal pricing data
seasons = [ "Winter", "Spring", "Summer", "Fall" ]
df = [ seasonalDF [ "price" ][ ( seasonalDF [ "season" ] == season ) ].values
for season in seasons ]

# set up our visualization
fig, ax = plt.subplots ( 1, 1, figsize = ( 5, 5 ) )

# create violin plot
ax.violinplot ( df, showmeans = False, showmedians = True )

# set labels
ax.set_title  ( "Seasonal Listing Prices" )
ax.yaxis.grid ( True )
ax.set_xticks ( [ ( y + 1 ) for y in range ( len ( df ) ) ] )
ax.set_xlabel ( "Season" )
ax.set_ylabel ( "Price ($)" )

# add x-tick labels
plt.setp(ax, xticks = [ ( y + 1 ) for y in range ( len ( df ) ) ], xticklabels
 = seasons )

# display plot
plt.tight_layout()
plt.show()
```



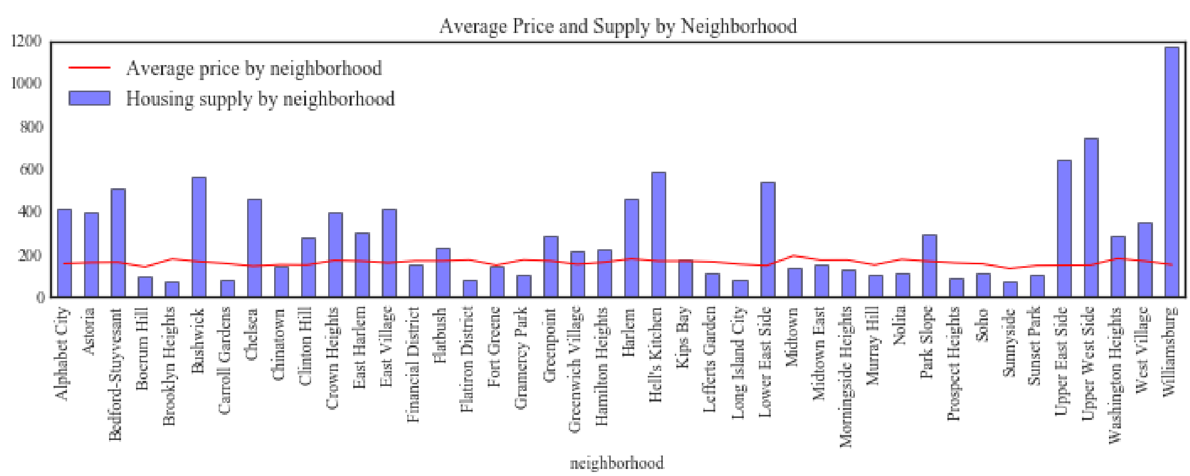
From the visualization, we don't see much shift based on the season, perhaps we need to look at other seasonal effects such as holidays, back-to-school, or summer to see if there's a real effect. We'll delve deeper into the potential for seasonal effects from a holiday persepctive and/or through interactions (e.g. higher prices around the holidays to accommodate friends and relatives).

Supply by neighborhood with their average prices

Now we are going to look into the number of houses (supply) by neighborhood and its relationship to average price. We begin by filtering out neighborhoods with fewer than 70 listings.

In [22]:
```
# get listings for neighborhoods with more than 70 listings
listings_size = listingsDF.groupby ( [ 'neighborhood' ] ).size()
listingsMoreThan70 = listings_size [ listings_size > 70 ].index.values
listingsDF_70 = listingsDF [ listingsDF [ 'neighborhood' ].isin ( listingsMore
Than70 ) ]
```

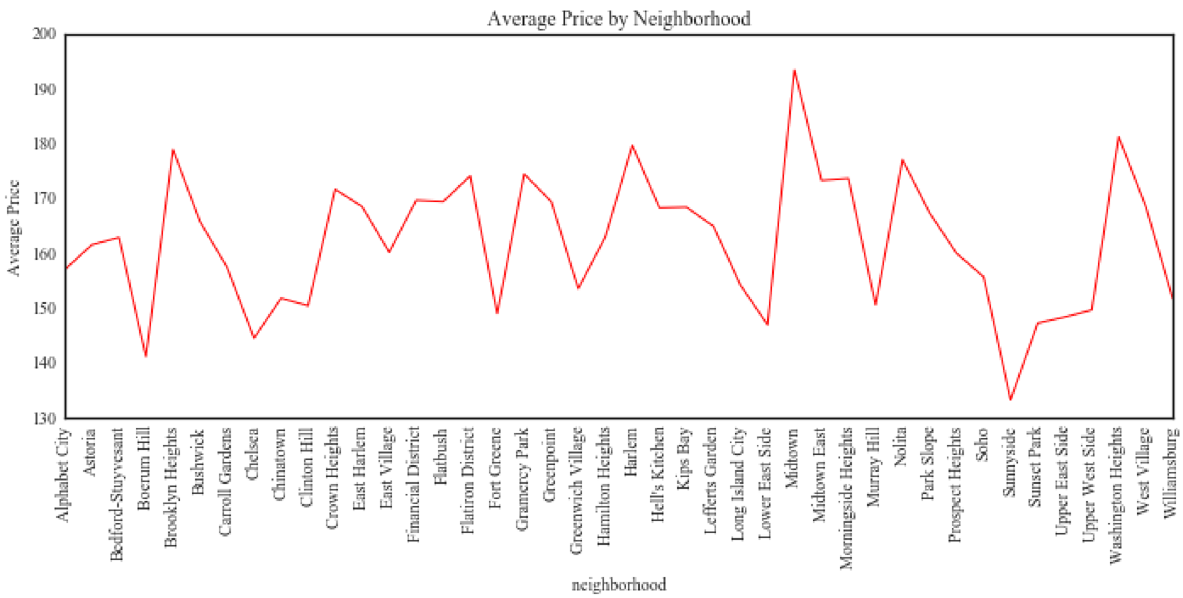
In [48]:
```
# set up visualization
fig1 = plt.figure ( figsize = ( 10, 4 ) )
ax1  = fig1.add_subplot ( 111 )

# plot the supply and mean price
listingsDF_70.groupby ( [ 'neighborhood' ] ) [ 'price' ].mean().plot (  kind
= 'line', color ='r', ax = ax1
                                                                        ,label
= 'Average price by neighborhood' )
listingsDF_70.groupby ( [ 'neighborhood' ] ).size().plot (  kind  = 'bar', ax
= ax1, color = 'b'
                                                            ,label = 'Housing s
upply by neighborhood', alpha = 0.5 )
plt.title  ( 'Average Price and Supply by Neighborhood')

# generate the display
plt.tight_layout()
plt.legend ( loc = 'best' )
plt.show()
```



We can clearly see here the supply of Airbnb house rentals vary by neighborhood and the average house prices by region varies between neighborhoods as well. Let's go in to more detail on average prices by neighborhood as the y-axis range is different for both graphs.

In [47]:

```
# set up the visualization
fig = plt.figure ( figsize = ( 10, 4 ) )

# set up the line graph
y = listingsDF_70.groupby ( [ 'neighborhood' ] )[ 'price' ].mean()
y.plot ( kind = 'line', color = 'r' )
x = range ( len ( y.values ) )

# create labels and
plt.ylabel ( 'Average Price' )
plt.title  ( 'Average Price by Neighborhood')
plt.tight_layout()
labels = list ( y.index.values )
plt.xticks ( x, labels, rotation = 'vertical' )

# display visualization
plt.show()
```



In this plot we can clearly the average price changes between neighborhoods. This may be helpful for predicting prices in those neighborhoods if we decide to use priors to improve our predictions.