Alex Hsieh - A11651288 - 4th Year Math and Economics Major
Atharva Fulay - A13140630 - 4th year Applied Math Major
Peter Yao - A12286133 - 3rd year Math-Computer Science Major
Justin Glommen - A13045994 - 2nd year Math-Computer Science Major

Two-Sigma Kaggle Competition - Final

Introduction:

RentHop, a portfolio company of Two-Sigma Ventures, helps home-searchers more easily find quality rental-listings by using data to evaluate each individual listing. We will use both real estate data sets provided by Kaggle, train.json and test.json, which represent the training set of data and testing set of data respectively, to ultimately predict how many inquiries a listing will receive based on its features and other factors. Using the data fields provided in each listing from the training set, we can estimate how desirable each home is to a consumer by formulating an algorithm.

The desirability is measured by the target variable called interest level, and is broken into three categories: low, medium, and high interest. It is our goal to take each listing from the test data set, and correctly predict the probability that it will fall under each respective interest level.

Analysis and Methods:

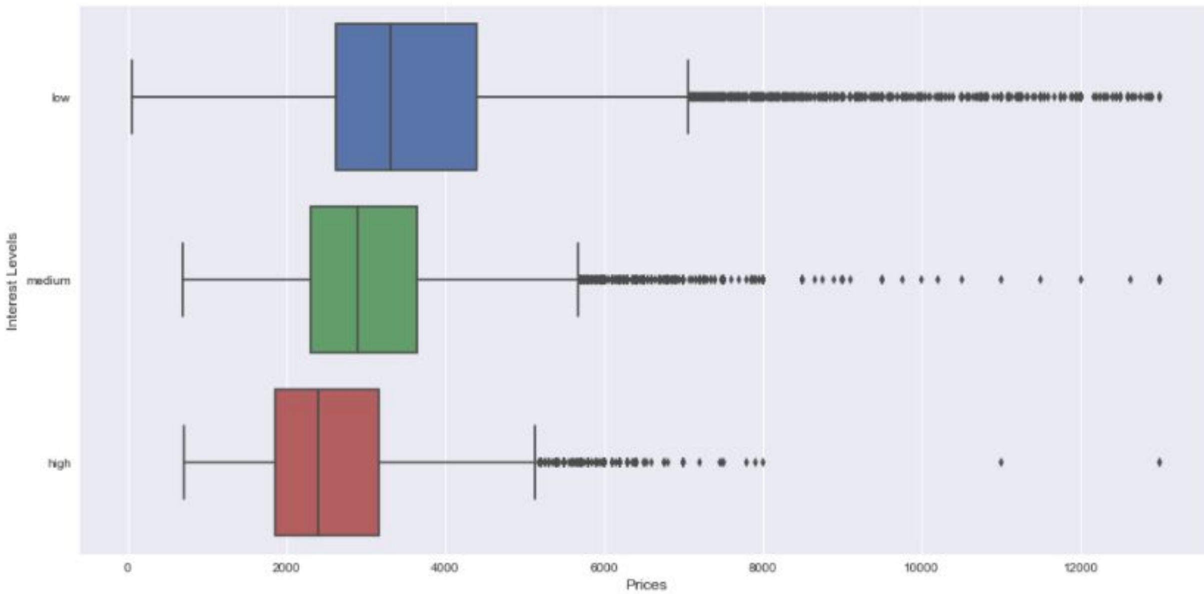
How did we subset our data and why?

Given data about apartment and housing listings throughout the New York City area, we created an algorithm to predict the interest level probabilities of the listing. To create hypothetical interest level values, we use a subset of the data which we deemed most influential (includes the price, features, listing ID, photos and number of bedrooms). This subset of data from the training set also has the given interest levels for the listing, so we partition this subset of factors each by their listing's respective interest level. This way, sample distributions for obtaining p-values for each category can be used.

The origin of this stems from deciding which factors most influence the interest level of a rental. In order to determine this, numerous visualization techniques were performed on the various factors from the training data set. This includes in-depth box plots, kernel density estimation plots, histograms, and joint distribution plots. Ultimately, checking for the most significant differences in distributions against the three interest levels leads to the best chances of predicting correctly. This is sensible, as comparing scores against distributions is only meaningful when the distributions themselves vary and carry information.

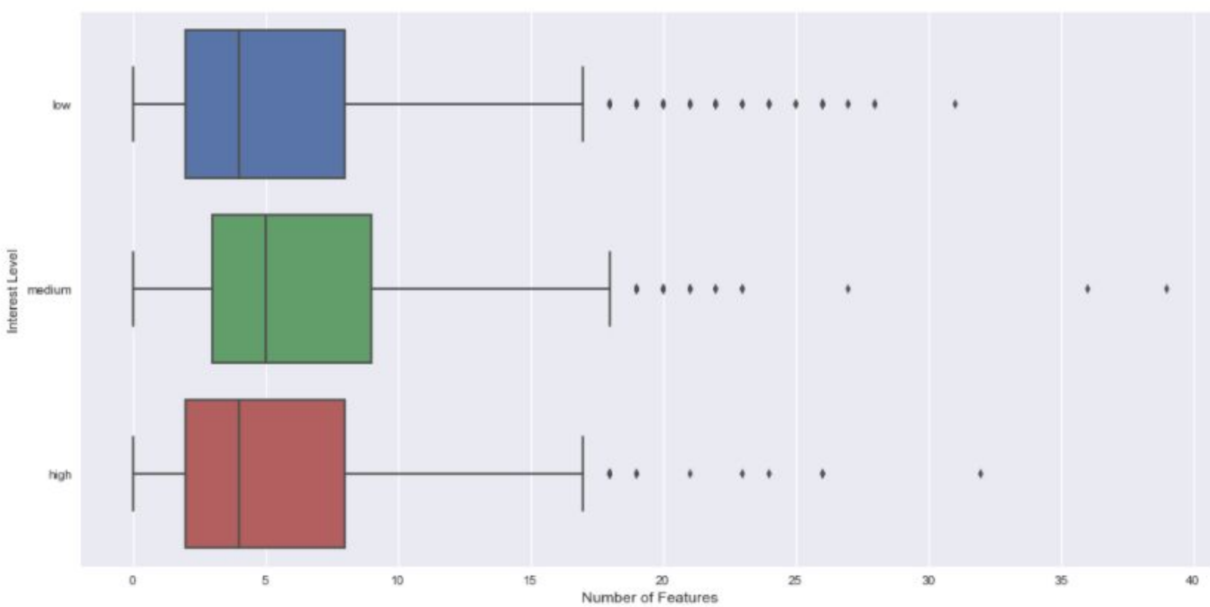
Given the nature of human emotion and investment in rentals and house-buying, we decided to first analyze the factors in order of intuition of what we as analysts feel is most heavily weighted in the decision, the price of a listing. Therefore, that is the first factor that we analyzed, since logically it will carry a significant impact on how popular a home is.

Figure 2.0.3 from the Appendix, also shown below, reaffirms that intuition.



Clearly, the amount of homes in the 'low' interest category that have a price greater than 8000 highly exceeds that of the other two categories. As well, the quartiles of each distribution are quite different from one another, and therefore will be informative in our later calculations.

Next, the number of features of an apartment listing was predicted to have a significant impact on interest level as well, so we began to analyze that using box plots. In order to do this, we ran through each listing in the test set prior to probability prediction and calculated the length of the amount of features and saved that value. Creating a distribution of number of features and partitioning into three categories yielded this from **Figure 2.0.5** from Appendix:



While although not immediately useful, we realize that this helps distinguish the medium interest level from the other two in terms of quantiles and averages. Therefore, we still incorporate this information into the prediction.

A couple other factors seem to have a strong influence on interest level as well, among which are the number of bedrooms and photos, so we decided to incorporate those as well. Again, for the purposes of our algorithm, since the key influencers seem to boil down to the price, number of features, photos, and bedrooms, these will be the main factors used in our analysis. The large training set was broken into a subset of these factors, each with a unique listing ID, for p-values to be generated for each score for each listing.

One factor from the data set that we originally thought was influential was the number of bathrooms stated in a listing. However, the vast majority of listings only had occurrences of either 1 or 2 bathrooms, as shown in **Figure 2.0.2** from the Appendix. After removing listings that had no bathroom number indicated and sorting each listing by interest level, the frequency of number of bathrooms was similar for all three levels. Since there was no visible difference in frequency of number of bathrooms between the categories, we chose not to include this factor in our analysis.

What was the framework behind our algorithm?

To calculate the p-values, we first iterate through each column, keep track of each listing's pricing, number of features, number of photos, listing ID and number of bedrooms and store each value in their own lists based on interest level. The listing ID has no influence on the listing's interest, so we ended up omitting this data in our calculations. The other categories that were originally provided in the data set were also not important factors, since their distributions were too similar to hold any meaning.

Initially, the percentile of each score from each column is found using the Python function stats.percentileofscore. Note that this will create three percentile values, one for each interest level. The percentile is of the test column value compared to that of all the values in each interest level. This will display the different percentiles for each score in each interest level. These values are then used to find individual p-values. These p-values are added up, averaged, and normalized so that their sum is 1. These are the final values used to create the low, medium and high probability predictions. A peek at a few of these p-values can be seen in the appendix under Figure B.

However, generation of these probabilities does not necessarily mean they are accurate. Therefore, in order to improve the algorithm and its results, we ran some tests that we deem useful. This includes counting the number of 'perfect' guesses on the training set: for each listing, if the highest probability was indeed the correct interest level, then it counts as a 'perfect' guess. After running on a large enough fraction of the training set, about forty-one percent of the time the result was 'perfect':

```
Perfect guesses:  413
Total guesses:  1000
Perfect accuracy:  0.413
Logloss:  1.08390205683
```

Obviously it will never reach one-hundred percent perfection as these are probabilities and not guarantees, so this is quite substantial, especially considering that a blind guess would converge on thirty-three percent.

Lastly, as the Kaggle competition checks for the multi-class logarithmic loss, we also calculate that as well and average around 1.05-1.10. For a step-by-step analysis, we have attached our code report (see Appendix).

Conclusion:

From the analysis and testing against the data, we are confident that our prediction model can determine the interest level of a listing given its information. As always, there may be uncertainties that impact the housing market in the future. However, assuming near future listings and market is similar to these listings, we are optimistic of our algorithm.

Theory:

Box and Whisker Plots:

A box-and-whisker plot, or box plot, is typically used to depict multiple groups of data with quartiles. The lower and upper quartiles indicate where the medians of the lower and upper halves of the data lie in the data set, while the lines extending from the boxes, or whiskers, signify variability. Some key statistical information that can be interpreted from a box plot include: the minimum and maximum of the data set, one standard deviation below and above the average of the data set, and the 2nd and 98th percentiles. All of the data in our box and whisker plots that were outliers are represented as points outside of the box and whisker plots.

From our box plots, it was very easy to identify unusual behavior between all three interest levels thanks to outliers, which are any values that are outside of the interval (Q1 - 1.5*IQR, Q3 + 1.5*IQR), where Q1 and Q3 are the medians of the lower and upper halves of the data set, and IQR = Q3 - Q1. For example, from the three box plots that depict the distributions of the price listings, the low interest level box plot had a large amount of high outliers while the medium and high interest level data had lower medians, smaller IQR ranges and much fewer outliers.

P-Value:

The p-value is typically calculated via the t-statistic, t = (score - xBar)/(s/sqrt(n)) or the z-score, z = (score - Mu)/sigma. Given that the factors typically do not follow a normal distribution, and we do not know their distribution, we will calculate p-values using percentiles derived from the t-statistic. It’s however important to note that since the dataset has large n, the t-statistic converges to the z-score. Sort the category of factors, take the index of the score in that subset and divide by the total length, say n/N, and that yields the percentile. Take its distance from 50 and subtract it from 50 and you get the p-value.

Appendix:

Low Interest Rentals:

	bedrooms	features	listing_id	photos	price
6	2	6	7092344	6	3800
15	0	4	7225292	4	2795
16	3	6	7226687	5	7200
18	3	5	7126989	7	6000
23	1	1	7131094	4	2435

Medium Interest Rentals:

	bedrooms	features	listing_id	photos	price
4	1	7	7170325	12	2400
9	2	6	7158677	6	3495
10	3	0	7211212	5	3000
38	0	11	7216312	4	2400
78	1	2	7131812	4	2700

High Interest Rentals:

	bedrooms	features	listing_id	photos	price
19	0	5	7114138	5	1945
46	3	11	7172046	7	4450
88	0	3	7167986	3	1495
93	3	1	7111210	9	2200
115	2	0	7126878	6	2795

Figure A - A snapshot of how the subset of the data looks like with respect to the interest levels.

P-Values:

	bedrooms_high	bedrooms_low	bedrooms_medium	features_high	features_low	\
4	0.537119	0.543285	0.549158	0.718546	0.676000	
6	0.829773	0.793402	0.843397	0.812191	0.861466	
9	0.829773	0.793402	0.843397	0.812191	0.861466	
10	0.336937	0.303465	0.324161	0.180906	0.139759	
15	0.149779	0.139497	0.143156	0.796301	0.768186	

	features_medium	listing_id	photos_high	photos_low	photos_medium	\
4	0.799003	7170325	0.034123	0.045765	0.029700	
6	0.918381	7092344	0.837979	0.808949	0.871137	
9	0.918381	7158677	0.837979	0.808949	0.871137	
10	0.140039	7211212	0.740557	0.790223	0.740315	
15	0.713643	7225292	0.451159	0.536723	0.432185	

	price_high	price_low	price_medium
4	0.765434	0.274180	0.496215
6	0.264913	0.576435	0.580150
9	0.475645	0.640211	0.773800
10	0.780933	0.547996	0.769347
15	0.775723	0.442889	0.733992

Figure B - The calculated p-values of each column separated by interest level.

```
Probabilities:
    listing_id      high    medium       low
4      7170325  0.498385  0.323093  0.178523
6      7092344  0.186362  0.408126  0.405513
9      7158677  0.251710  0.409493  0.338798
10     7211212  0.372178  0.366657  0.261165
15     7225292  0.397276  0.375904  0.226820
```

Figure C - A view of five interest level values calculated by our algorithm.

Further in-depth analysis in the code report on the next page.

RentalListingVisualizations

April 4, 2017

1 Initializing Dataset

We must initialize the dataset first, loading it in as a pandas object from a JSON object.

```
In [43]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns

         colors = sns.color_palette()

         trainData = pd.read_json("train.json")
         testData = pd.read_json("test.json")
         trainData.head()
```

```
Out[43]:         bathrooms  bedrooms                       building_id  \
         10            1.5         3  53a5b119ba8f7b61d4e010512e0dfc85
         10000         1.0         2  c5c8a357cba207596b04d1afd1e4f130
         100004        1.0         1  c3ba40552e2120b0acfc3cb5730bb2aa
         100007        1.0         1  28d9ad350afeaab8027513a3e52ac8d5
         100013        1.0         4                                 0

                             created  \
         10      2016-06-24 07:54:24
         10000   2016-06-12 12:19:27
         100004  2016-04-17 03:26:41
         100007  2016-04-18 02:22:02
         100013  2016-04-28 01:32:41

                                                        description  \
         10      A Brand New 3 Bedroom 1.5 bath ApartmentEnjoy ...
         10000
         100004  Top Top West Village location, beautiful Pre-w...
         100007  Building Amenities - Garage - Garden - fitness...
         100013  Beautifully renovated 3 bedroom flex 4 bedroom...

                     display_address  \
```

```
10       Metropolitan Avenue
10000        Columbus Avenue
100004           W 13 Street
100007       East 49th Street
100013     West 143rd Street

                                                     features interest_level
10                                                         []         medium
10000   [Doorman, Elevator, Fitness Center, Cats Allow...            low
100004  [Laundry In Building, Dishwasher, Hardwood Flo...           high
100007                         [Hardwood Floors, No Fee]            low
100013                                         [Pre-War]            low

        latitude  listing_id  longitude                        manager_id
10       40.7145     7211212   -73.9425  5ba989232d0489da1b5f2c45f6688adc
10000    40.7947     7150865   -73.9667  7533621a882f71e25173b27e3139d83d
100004   40.7388     6887163   -74.0018  d9039c43983f6e564b1482b273bd7b01
100007   40.7539     6888711   -73.9677  1067e078446a7897d2da493d2f741316
100013   40.8241     6934781   -73.9493  98e13ad4b495b9613cef886d79a6291f

                                                   photos  price  \
10      [https://photos.renthop.com/2/7211212_1ed4542e...   3000
10000   [https://photos.renthop.com/2/7150865_be3306c5...   5465
100004  [https://photos.renthop.com/2/6887163_de85c427...   2850
100007  [https://photos.renthop.com/2/6888711_6e660cee...   3275
100013  [https://photos.renthop.com/2/6934781_1fa4b41a...   3350

                  street_address
10      792 Metropolitan Avenue
10000       808 Columbus Avenue
100004          241 W 13 Street
100007     333 East 49th Street
100013    500 West 143rd Street
```

In [44]:
```
testData.head()
```

Out[44]:
```
        bathrooms  bedrooms                       building_id  \
0             1.0         1  79780be1514f645d7e6be99a3de696c5
1             1.0         2                                 0
100           1.0         1  3dbbb69fd52e0d25131aa1cd459c87eb
1000          1.0         2  783d21d013a7e655bddc4ed0d461cc5e
100000        2.0         2  6134e7c4dd1a98d9aee36623c9872b49

                    created  \
0       2016-06-11 05:29:41
1       2016-06-24 06:36:34
100     2016-06-03 04:29:40
1000    2016-06-11 06:17:35
```

```
100000  2016-04-12 05:24:17

                                               description  \
0       Large with awesome terrace--accessible via bed...
1       Prime Soho - between Bleecker and Houston - Ne...
100             New York chic has reached a new level ...
1000    Step into this fantastic new Construction in t...
100000  ~Take a stroll in Central Park, enjoy the ente...

              display_address  \
0              Suffolk Street
1             Thompson Street
100       101 East 10th Street
1000      South Third Street\r
100000  Midtown West, 8th Ave

                                                  features  latitude  \
0       [Elevator, Laundry in Building, Laundry in Uni...   40.7185
1                   [Pre-War, Dogs Allowed, Cats Allowed]   40.7278
100                         [Doorman, Elevator, No Fee]   40.7306
1000    [Roof Deck, Balcony, Elevator, Laundry in Buil...   40.7109
100000  [Common Outdoor Space, Cats Allowed, Dogs Allo...   40.7650

        listing_id  longitude                        manager_id  \
0          7142618   -73.9865  b1b1852c416d78d7765d746cb1b8921f
1          7210040   -74.0000  d0b5648017832b2427eeb9956d966a14
100        7103890   -73.9890  9ca6f3baa475c37a3b3521a394d65467
1000       7143442   -73.9571  0b9d5db96db8472d7aeb67c67338c4d2
100000     6860601   -73.9845  b5eda0eb31b042ce2124fd9e9fcfce2f

                                                    photos  price  \
0       [https://photos.renthop.com/2/7142618_1c45a2c8...   2950
1       [https://photos.renthop.com/2/7210040_d824cc71...   2850
100     [https://photos.renthop.com/2/7103890_85b33077...   3758
1000    [https://photos.renthop.com/2/7143442_0879e9e0...   3300
100000  [https://photos.renthop.com/2/6860601_c96164d8...   4900

                  street_address
0               99 Suffolk Street
1             176 Thompson Street
100          101 East 10th Street
1000    251  South Third Street\r
100000       260 West 54th Street
```

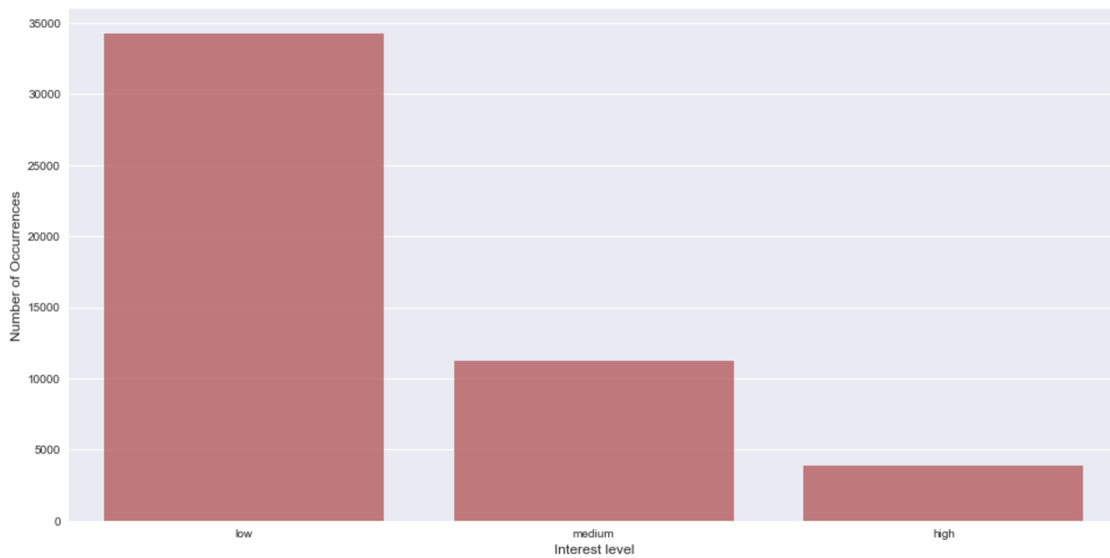
Notice that the training set and the test set have the same factors besides interest_level. That's because interest_level is our target variable, and we'll use it for comparison and analysis in the following below.

2 Visualizing Data

We want to start taking a look at what variables have most influence over the target variable. So, we want to compare the distributions of the varying factors.

2.0.1 Interest Levels

```
In [45]: interestLevelCounts = trainData['interest_level'].value_counts()

         plt.figure(figsize=(16,8))
         sns.barplot(interestLevelCounts.index, interestLevelCounts.values, alpha=(
         plt.ylabel('Number of Occurrences', fontsize=12)
         plt.xlabel('Interest level', fontsize=12)
         plt.show()
```

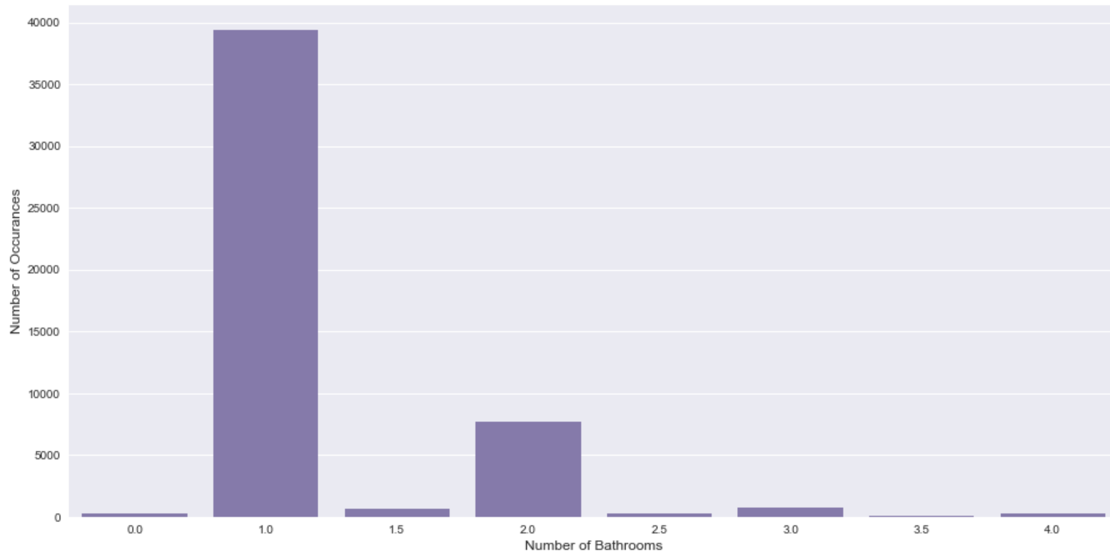


The amount of 'low' interest level houses outnumbers the amount of 'medium' and 'high' occurances. This means the distribution is a bit unbalanced, and we need to be careful when analyzing direct numbers of the factors. We may want to consider using percentages when comparing variables.

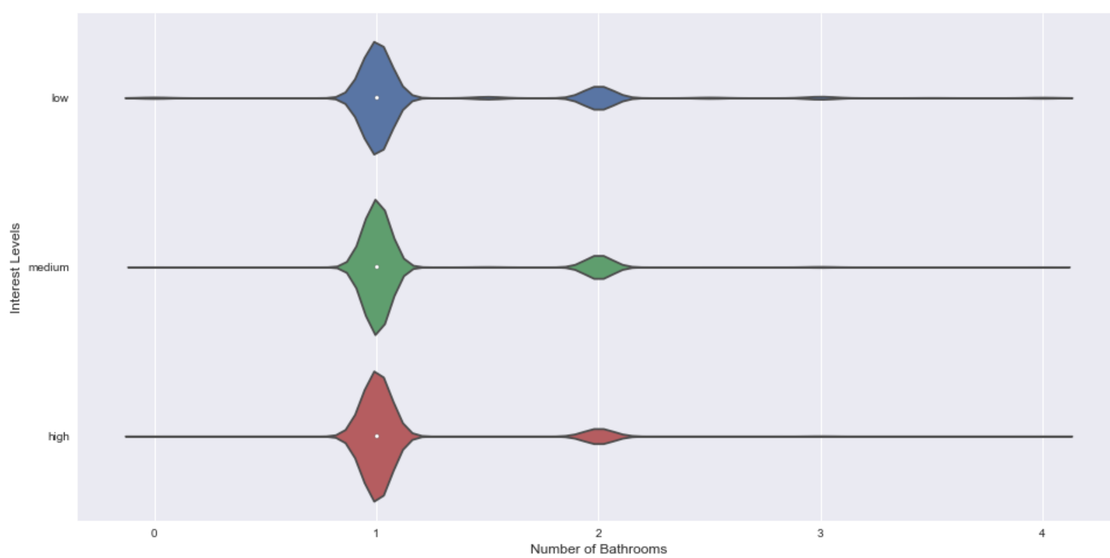
2.0.2 Bathrooms

```
In [46]: # Normalizing the outliers
         outlierBathroomRows = trainData['bathrooms'] > 4
         trainData.loc[outlierBathroomRows, 'bathrooms'] = 4

         bathroomCount = trainData['bathrooms'].value_counts()

         plt.figure(figsize=(16,8))
         sns.barplot(bathroomCount.index, bathroomCount.values, color=colors[3])
```

```
plt.ylabel('Number of Occurances', fontsize=12)
plt.xlabel('Number of Bathrooms', fontsize=12)
plt.show()
```




```
In [47]: #trainData['bathrooms'].ix[trainData['bathrooms'] > 4] = 4

         plt.figure(figsize=(16,8))
         sns.violinplot(trainData['bathrooms'], trainData['interest_level'], order=
         plt.xlabel('Number of Bathrooms', fontsize=12)
         plt.ylabel('Interest Levels', fontsize=12)
         plt.show()
```



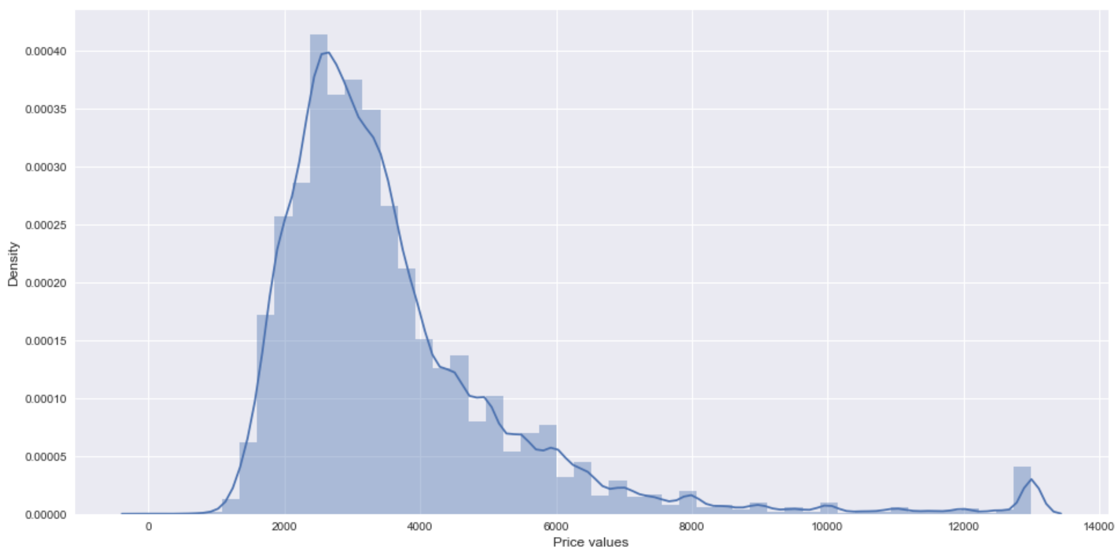
The number of bathrooms appears to be pretty consistent across interest levels. So bathrooms will not be a very important factor in analysis for the target variable, at least at first glance.

2.0.3 Prices

Initially after graphing the prices there were a couple of outliers throwing off the visualization, so we left that out in favor of a more useful approach.

```
In [48]: # Normalizing the outliers
         percentile_99_Price = np.percentile(trainData['price'], 99)
         outlierPriceRows = trainData['price'] > percentile_99_Price
         trainData.loc[outlierPriceRows, 'price'] = percentile_99_Price

         plt.figure(figsize=(16,8))
         sns.distplot(trainData['price'])
         plt.xlabel('Price values', fontsize=12)
         plt.ylabel('Density', fontsize=12)
         plt.show()
```

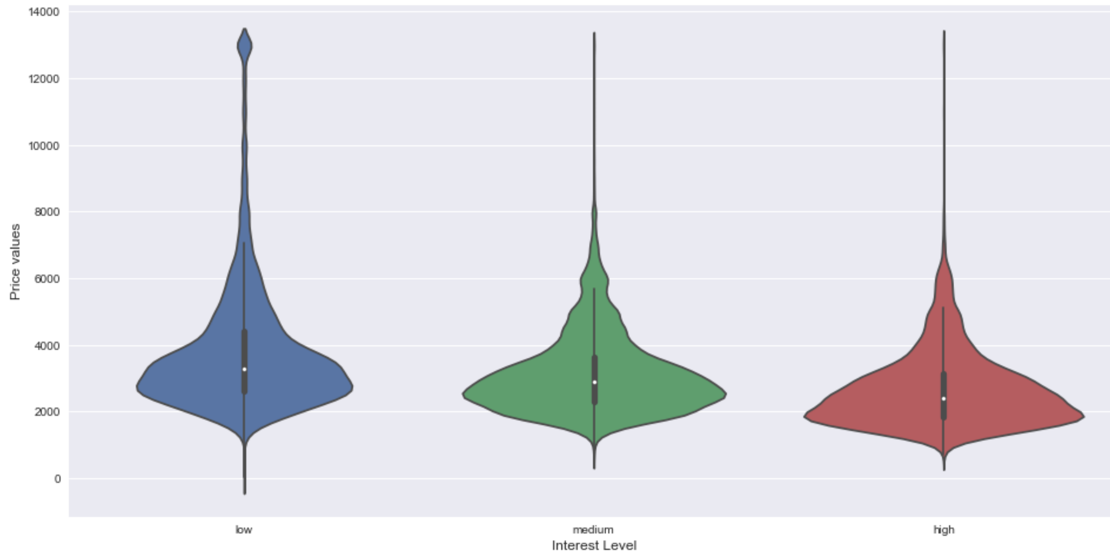
```
/Users/JustinonTG/anaconda/lib/python3.6/site-packages/statsmodels/nonparametric/kc
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```



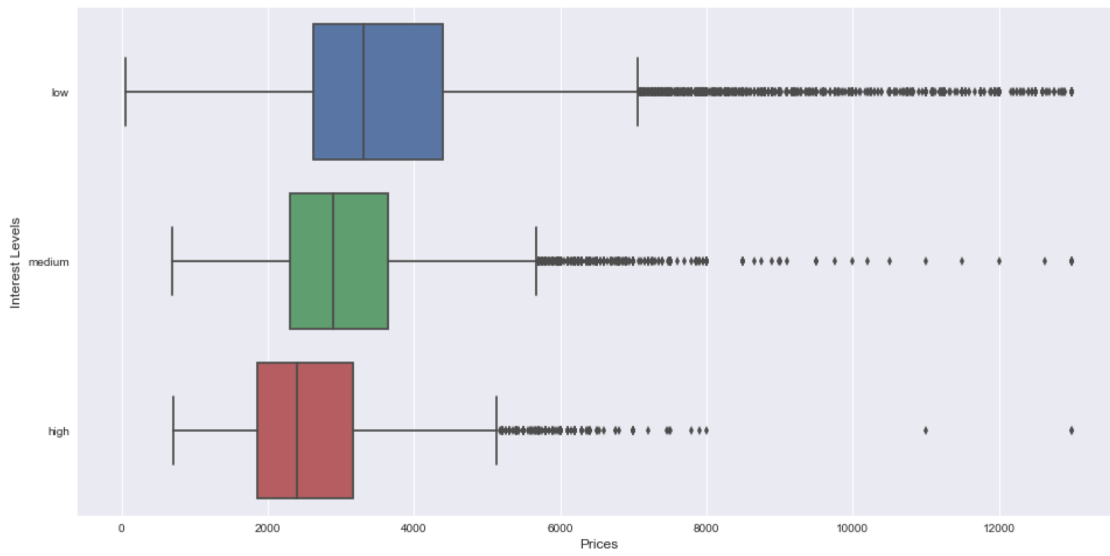
That overall distribution looks quite nice, with a skew right. It should be interesting to visualize the prices against the interest levels.

```
In [49]: plt.figure(figsize=(16,8))
         sns.violinplot(x='interest_level', y='price', data=trainData, order=['low'
```

```
plt.xlabel('Interest Level', fontsize=12)
plt.ylabel('Price values', fontsize=12)
plt.show()
```



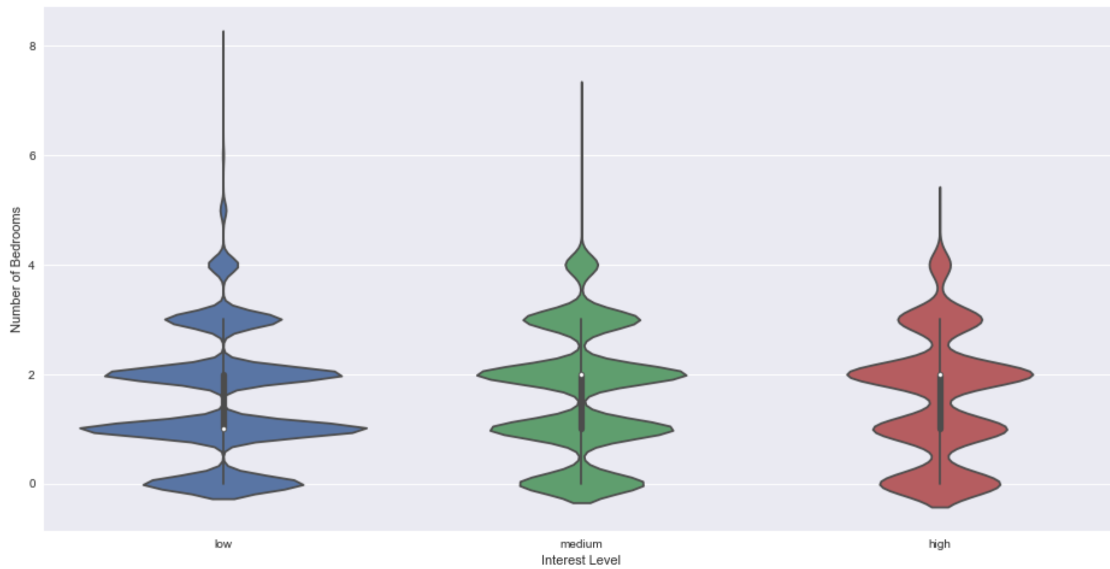

```
In [50]: plt.figure(figsize=(16,8))
         sns.boxplot(x='price', y='interest_level', data=trainData, order=['low','m
         plt.xlabel('Prices', fontsize=12)
         plt.ylabel('Interest Levels', fontsize=12)
         plt.show()
```



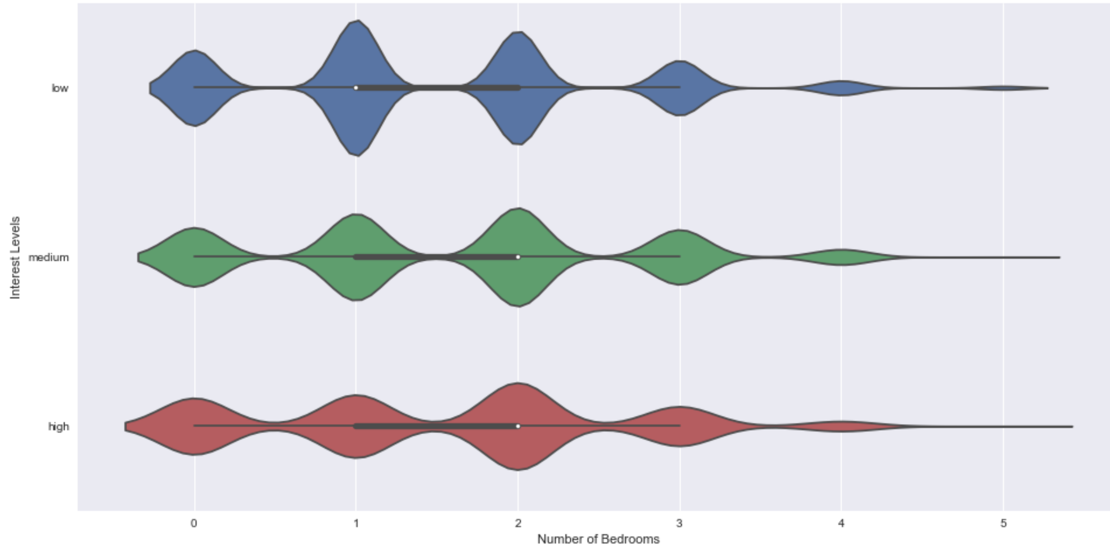
It's easy to notice here that the 'low' interest level flats also are the ones that have the most density in the higher price values. Also, the 'high' interest level flats have a lower average price. This is logical, and helps in our prediction. The price variable therefore should have some more weight than the other factors in predicting the influence level.

2.0.4 Bedrooms

In [51]:
```
plt.figure(figsize=(16,8))
sns.violinplot(x='interest_level', y='bedrooms', data=trainData, order=["l
plt.xlabel('Interest Level')
plt.ylabel('Number of Bedrooms')
plt.show()
```



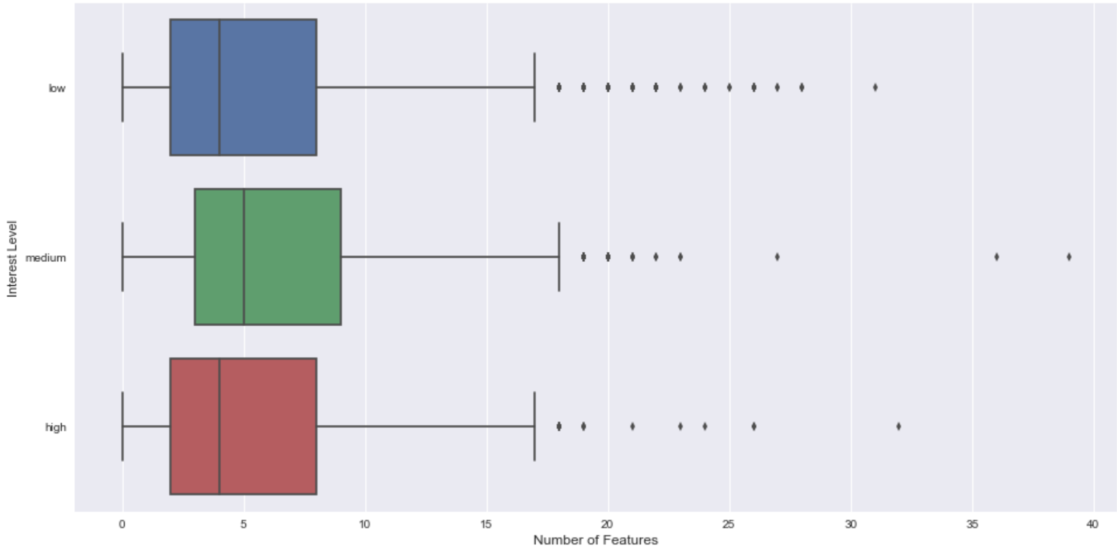
In [62]:
```
# Normalizing the outliers
percentile_995_Bedrooms = np.percentile(trainData['bedrooms'], 99.5)
percentile_005_Bedrooms = np.percentile(trainData['bedrooms'], .5)
outlierHighBedRows = trainData['bedrooms'] > percentile_995_Bedrooms
outlierLowBedRows = trainData['bedrooms'] < percentile_005_Bedrooms
trainData.loc[outlierHighBedRows,'bedrooms'] = percentile_995_Bedrooms
trainData.loc[outlierLowBedRows,'bedrooms'] = percentile_005_Bedrooms

plt.figure(figsize=(16,8))
sns.violinplot(x='bedrooms', y='interest_level', data=trainData, order=["l
plt.xlabel('Number of Bedrooms')
plt.ylabel('Interest Levels')
plt.show()
```



2.0.5 Features

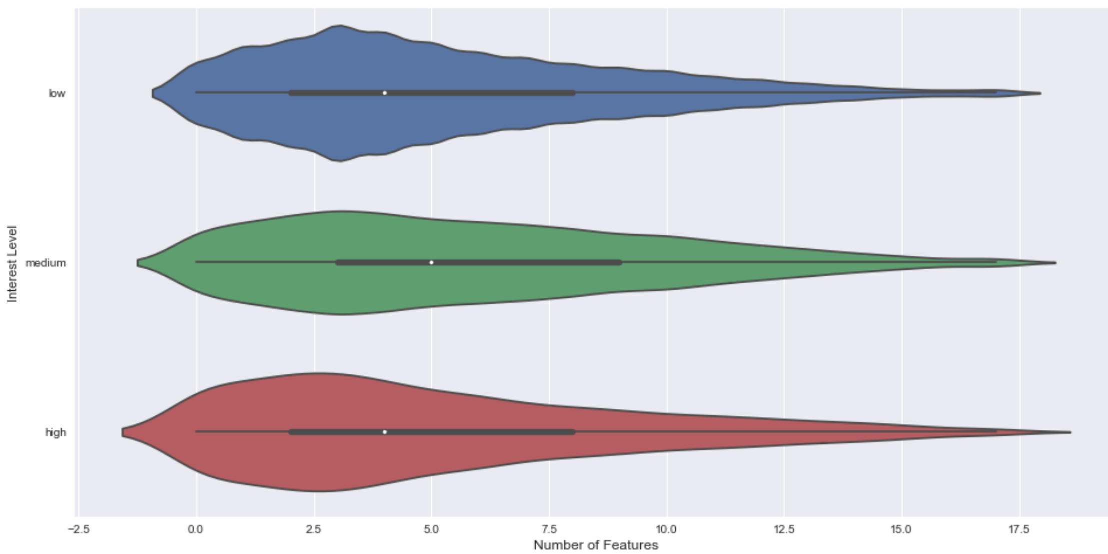
Here we are going to display the various numbers of features and how they're distributed according to the interest level. This will be achieved by counting the number of features provided for each listing and displaying the counts.

```
In [53]: trainData['num_features'] = trainData['features'].apply(len)

         plt.figure(figsize=(16,8))
         listOrder = ['low', 'medium', 'high']
         sns.boxplot(x='num_features', y='interest_level', data=trainData, order=li
         plt.xlabel('Number of Features', fontsize=12)
         plt.ylabel('Interest Level')
         plt.show()
```



Normalizing the outliers yields us the following:

```
In [54]: percentile_995_Features = np.percentile(trainData['num_features'], 99.5)
         percentile_005_Features = np.percentile(trainData['num_features'], .5)
         outlierHighFeatRows = trainData['num_features'] > percentile_995_Features
         outlierLowFeatRows = trainData['num_features'] < percentile_005_Features
         trainData.loc[outlierHighFeatRows,'num_features'] = percentile_995_Feature
         trainData.loc[outlierLowFeatRows,'num_features'] = percentile_005_Features

         plt.figure(figsize=(16,8))
         listOrder = ['low', 'medium', 'high']
         sns.violinplot(x='num_features', y='interest_level', data=trainData, order
         plt.xlabel('Number of Features', fontsize=12)
         plt.ylabel('Interest Level')
         plt.show()
```

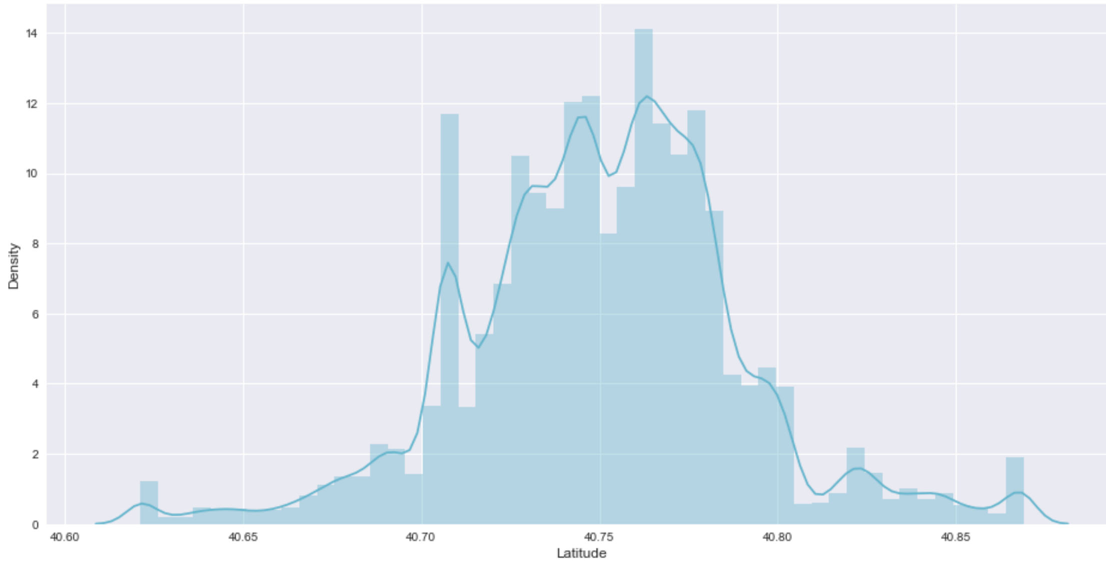


2.1 Geolocation

2.1.1 Latitude

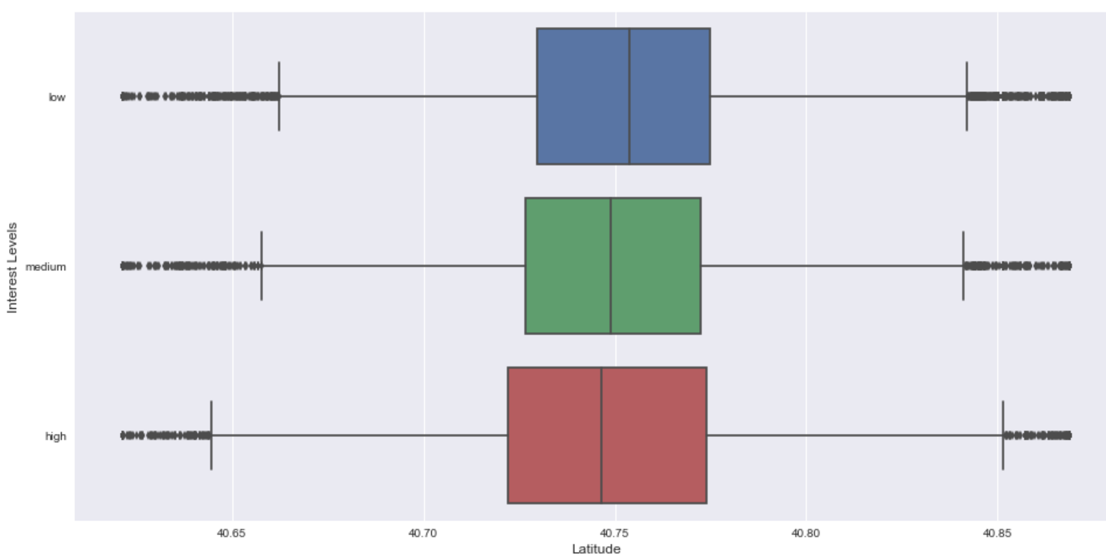
```
In [55]: # Normalizing the outliers
         percentile_995_Latitude = np.percentile(trainData['latitude'], 99.5)
         percentile_005_Latitude = np.percentile(trainData['latitude'], .5)
         outlierUpperLatRows = trainData['latitude'] > percentile_995_Latitude
         outlierLowerLatRows = trainData['latitude'] < percentile_005_Latitude
         trainData.loc[outlierUpperLatRows, 'latitude'] = percentile_995_Latitude
         trainData.loc[outlierLowerLatRows, 'latitude'] = percentile_005_Latitude

         plt.figure(figsize=(16,8))
         sns.distplot(trainData['latitude'], color=colors[5])
         plt.xlabel("Latitude", fontsize=12)
         plt.ylabel("Density", fontsize=12)
         plt.show()
```

```
/Users/JustinonTG/anaconda/lib/python3.6/site-packages/statsmodels/nonparametric/kc
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```



Interestingly, the latitude appears to follow a Gaussian distribution. Further bootstrap testing and Q-Q plotting should be conducted here to compare to the normal.

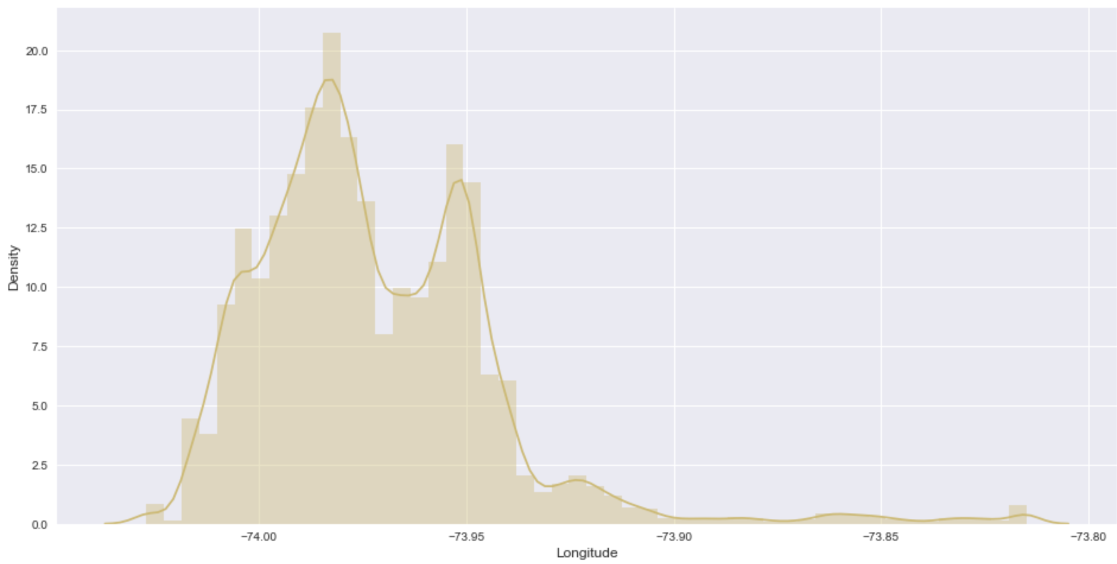
```
In [56]: plt.figure(figsize=(16,8))
         sns.boxplot(x='latitude', y='interest_level', data=trainData, order=["low'
         plt.xlabel("Latitude", fontsize=12)
         plt.ylabel("Interest Levels", fontsize=12)
         plt.show()
```



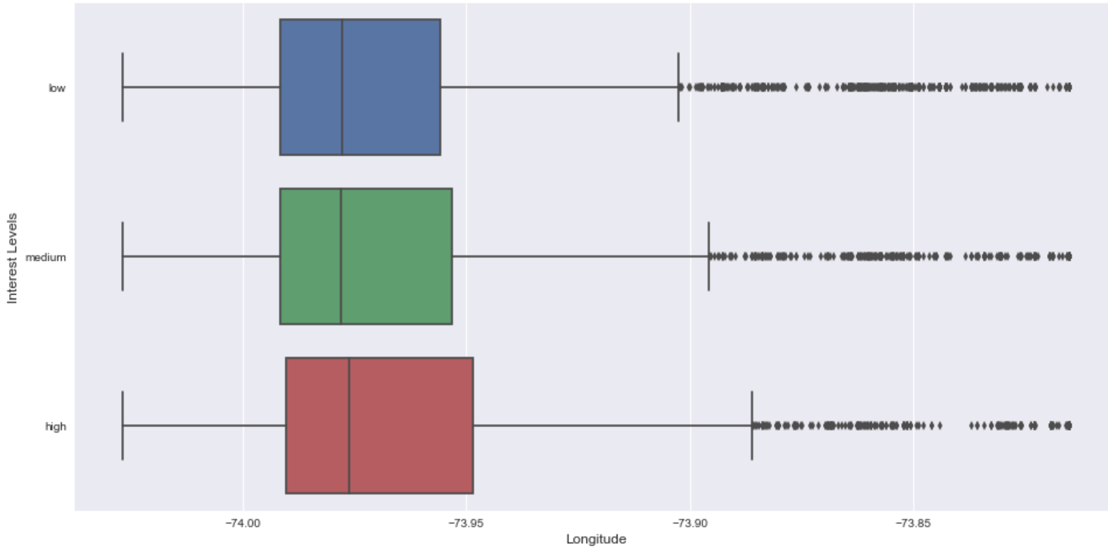
2.1.2 Longitude

```
In [57]: # Normalizing the outliers
         percentile_9975_Longitude = np.percentile(trainData['longitude'], 99.75)
         percentile_0025_Longitude = np.percentile(trainData['longitude'], .25)
         outlierUpperLongRows = trainData['longitude'] > percentile_9975_Longitude
         outlierLowerLongRows = trainData['longitude'] < percentile_0025_Longitude
         trainData.loc[outlierUpperLongRows, 'longitude'] = percentile_9975_Longitu
         trainData.loc[outlierLowerLongRows, 'longitude'] = percentile_0025_Longitu

         plt.figure(figsize=(16,8))
         sns.distplot(trainData['longitude'], color=colors[4])
         plt.xlabel("Longitude", fontsize=12)
         plt.ylabel("Density", fontsize=12)
         plt.show()
```

```
/Users/JustinonTG/anaconda/lib/python3.6/site-packages/statsmodels/nonparametric/kd
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```




```
In [58]: plt.figure(figsize=(16,8))
         sns.boxplot(x='longitude', y='interest_level', data=trainData, order=["low
         plt.xlabel("Longitude", fontsize=12)
         plt.ylabel("Interest Levels", fontsize=12)
         plt.show()
```

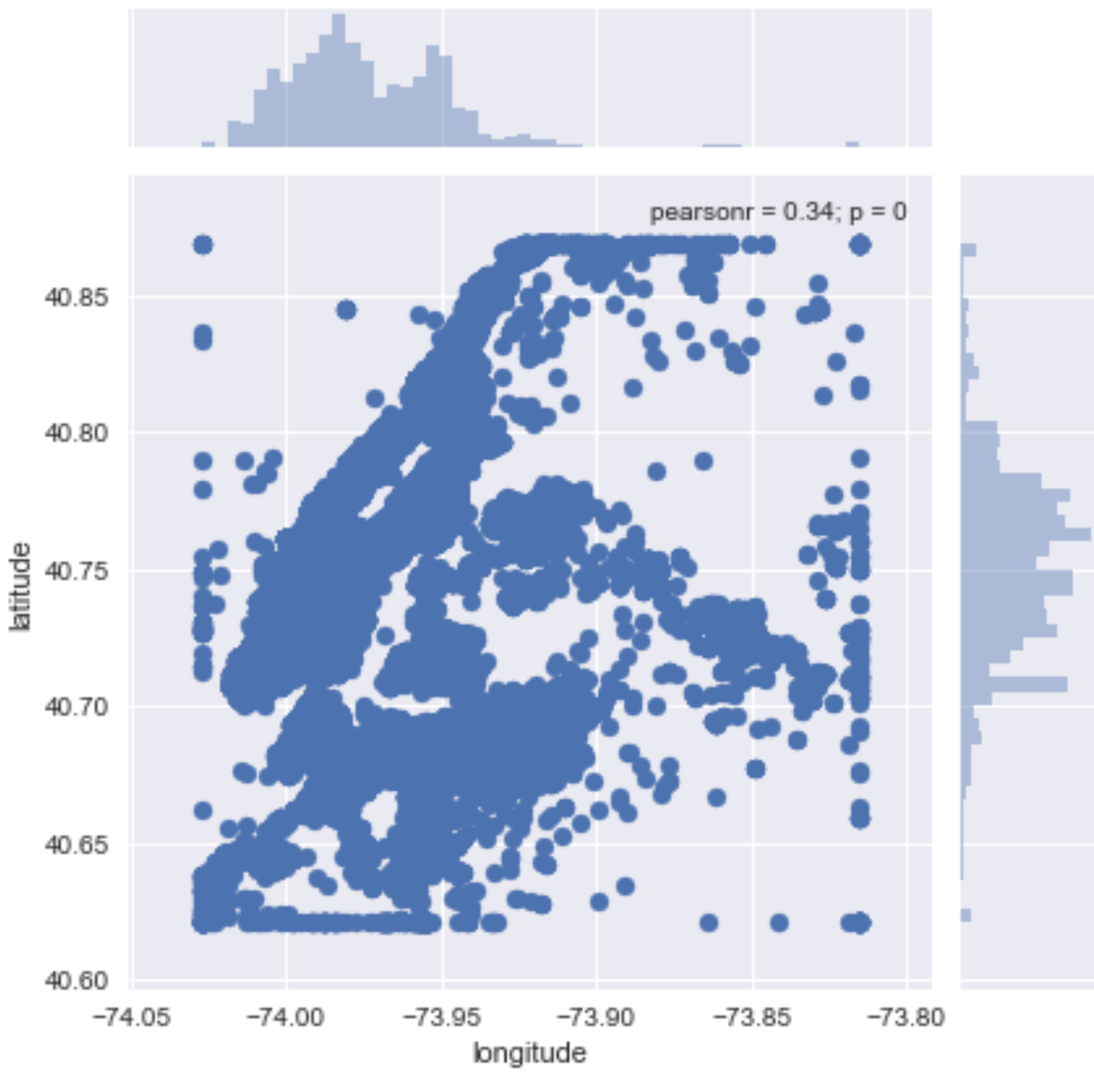


2.1.3 Mapping

First we'll plot the latitude and longitude together to graph the map of the city with which we're dealing.

```
In [59]: plt.figure(figsize=(16,16))
         sns.jointplot(x='longitude',y='latitude', data=trainData)
         plt.show()
```

```
<matplotlib.figure.Figure at 0x13d1772e8>
```

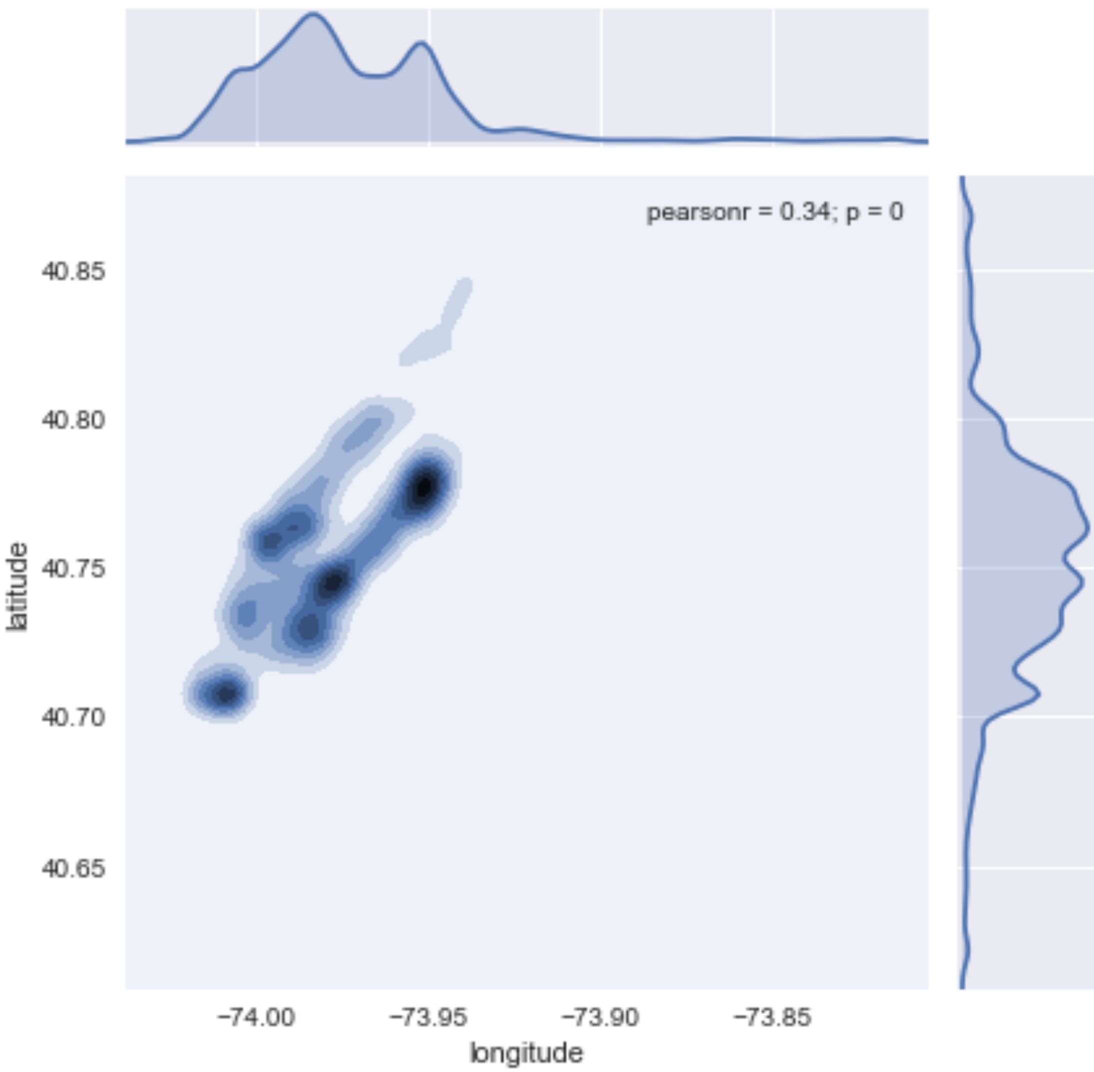


Now to check for the actual densities within the region, we'll use the kernel density estimation joint plot.

```
In [60]: plt.figure(figsize=(16,16))
         sns.jointplot(x='longitude',y='latitude', data=trainData, kind='kde')
         plt.show()
```

```
/Users/JustinonTG/anaconda/lib/python3.6/site-packages/statsmodels/nonparametric/kc
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```

```
<matplotlib.figure.Figure at 0x131521630>
```


pearsonr = 0.34; p = 0
40.85
40.80
40.75
40.70
40.65
latitude
-74.00
-73.95
-73.90
-73.85
longitude

```
In [ ]:
```