Libbing_Final_Project

December 17, 2020

#**Airbnb in NYC: Characteristics of a Good Listing**

###By Jessica Guo (jqg214), Spencer Libbing (stl327) and Harry Wu (zw1869)

###Data Bootcamp Final Group Project

0.1 Topic:

Airbnb is an online vacation rental marketplace that that took its company public on December 10th, 2020 in one of the most highly anticipated IPOs of 2020. Through the service, users can arrange lodging and tourism experiences or list their properties for rental. Airbnb does not own any of the listed properties but instead profits by receiving commission from each booking.

Our project seeks to use listing data for Airbnb properties in NYC in 2019, as found on 'NY_Listings.csv' from https://www.kaggle.com/samyukthamurali/airbnb-ratings-dataset. We aim to determine what characteristics make for the best Airbnb listing to help future hosts optimize where and how they list they properties.

0.1.1 Themes:

What can we learn about different hosts and areas?

What can we learn from predictions? (ex: locations, prices, reviews, etc)

Which hosts are the busiest and why?

Is there any noticeable difference of traffic among different areas and what could be the reason for it?

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
%matplotlib inline
```

[ ]:
```
data = pd.read_csv('NY_Listings.csv', encoding='latin-1')
data.dropna()
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718:
DtypeWarning: Columns (7,19,24) have mixed types.Specify dtype option on import
```

```
or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[ ]:
```
       Listing ID  … Reviews per month
0            2515  …              1.77
1            2539  …              1.54
2            2595  …              3.83
3            3330  …              0.67
4            3647  …              3.70
…             …  …                 …
44301    21176357  …              3.44
44303    21176433  …              0.40
44305    21177066  …              2.25
44306    21177156  …              4.76
44308    21177575  …              0.46

[26743 rows x 35 columns]
```

0.2 Borough Specific Data

[ ]:
```python
data['City'].value_counts()
```

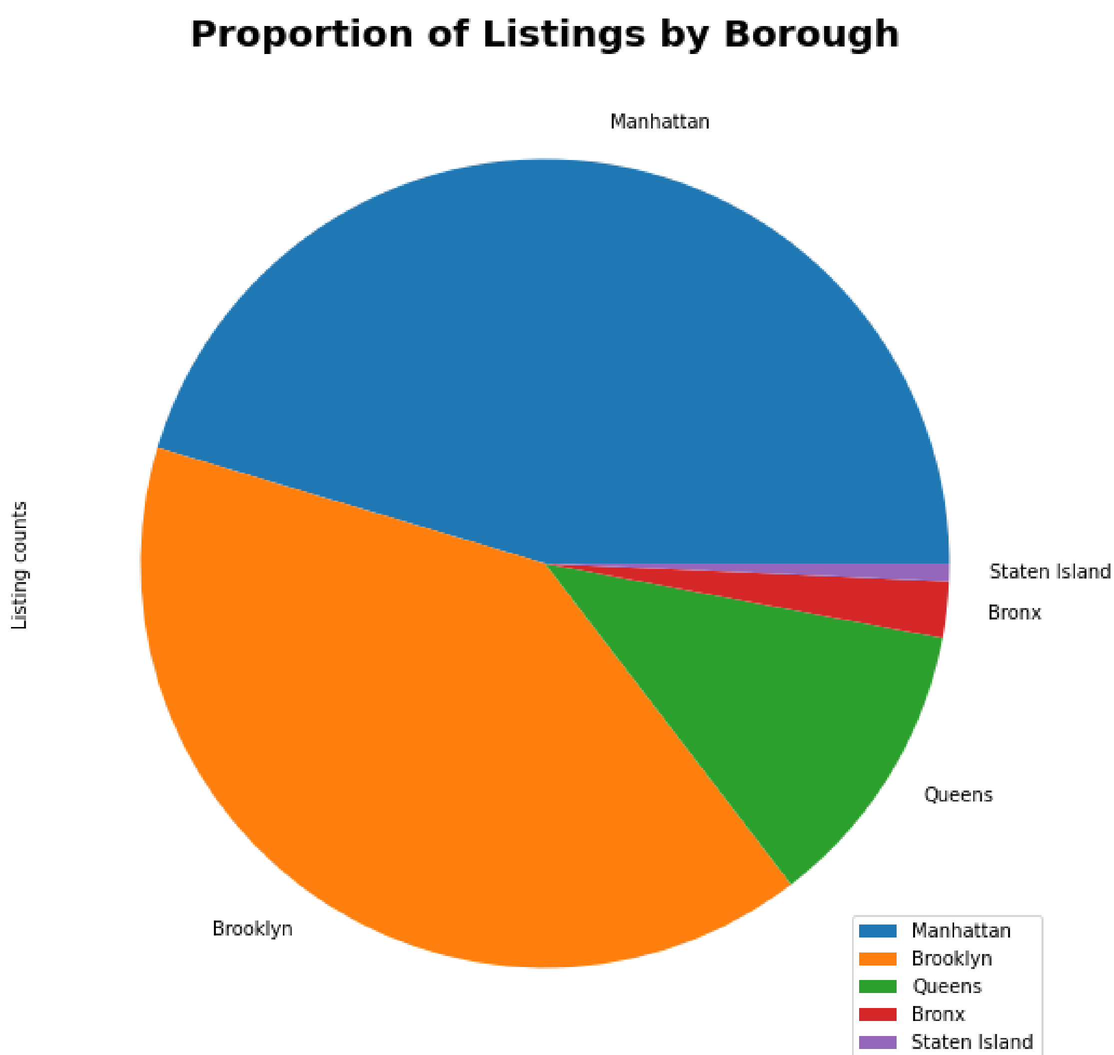
[ ]:
```
Manhattan        34375
Brooklyn         30323
Queens            8816
Bronx             1703
Staten Island      532
Name: City, dtype: int64
```

[ ]:
```python
value_counts = data['City'].value_counts().rename_axis('Borough').
 ↪reset_index(name='Listing counts')
value_counts
```

[ ]:
```
         Borough  Listing counts
0      Manhattan           34375
1       Brooklyn           30323
2         Queens            8816
3          Bronx            1703
4  Staten Island             532
```

[ ]:
```python
fig,ax = plt.subplots()
value_counts.set_index(['Borough']).plot.pie(ax=ax, y='Listing␣
 ↪counts',figsize=(10,10))
ax.set_title('Proportion of Listings by Borough', size = 20, fontweight =␣
 ↪'bold')
```

[ ]:
```
Text(0.5, 1.0, 'Proportion of Listings by Borough')
```




```
[ ]: cities = data.groupby('City')['Price','Review Scores Rating','Review Scores␣
     ↪Location',
                                   'Number of reviews'].mean().
     ↪sort_values('Price',ascending = False)
     cities
```

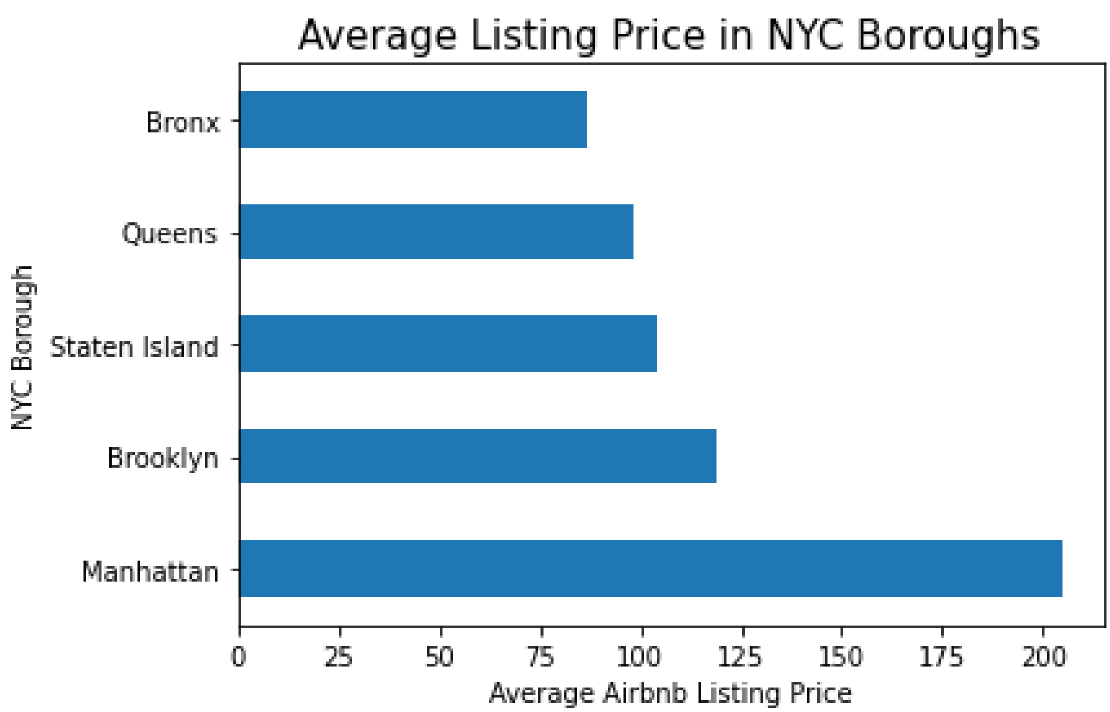
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: FutureWarning:
Indexing with multiple keys (implicitly converted to a tuple of keys) will be
deprecated, use a list instead.
```

```
[ ]:                  Price  …  Number of reviews
     City                   …
     Manhattan   204.686865 …          16.031302
```

```
Brooklyn       118.774561  …          16.299739
Staten Island  103.862782  …          20.541353
Queens          98.211093  …          19.466198
Bronx           86.840869  …          16.869055

[5 rows x 4 columns]
```

[ ]:
```
fig,ax=plt.subplots()

cities['Price'].plot.barh(ax=ax,figsize=(6,4))

ax.set_title('Average Listing Price in NYC Boroughs',size=15)
ax.set_xlabel('Average Airbnb Listing Price')
ax.set_ylabel('NYC Borough')
```

[ ]:
```
Text(0, 0.5, 'NYC Borough')
```



[ ]:
```
manhattan = data.loc[data['City']=='Manhattan', :]
manhattan
```

[ ]:
```
       Listing ID  … Reviews per month
19284    10084117  …               0.0
19285    10084168  …               0.0
19286    10084380  …               0.0
```

```
19287    10084382  …              0.0
19288    10084751  …              2.6
…            …   …            …
75733    42739066  …              0.0
75738    42761614  …              0.0
75739    42770773  …              0.0
75743    42794081  …              0.0
75748    42881423  …              0.0

[34375 rows x 35 columns]
```

[ ]:
```
manhattan = manhattan.loc[manhattan['Review Scores Rating'] != 0]
manhattan
```

[ ]:
```
       Listing ID  … Reviews per month
19288    10084751  …             2.60
19290    10085478  …             1.17
19291    10086029  …             0.23
19292    10086307  …             0.11
19293    10086344  …             1.10
…            …   …            …
39630    19841164  …             0.12
39631    19841329  …             1.62
39632    19841564  …             0.06
39633    19841625  …             0.05
39634    19841797  …             0.14

[15704 rows x 35 columns]
```

[ ]:
```
manhattan['Neighbourhood cleansed'].value_counts()
```

[ ]:
```
Harlem                     2071
East Village               1579
Upper West Side            1449
Hell's Kitchen             1381
Upper East Side            1312
East Harlem                 885
Chelsea                     835
Midtown                     822
Lower East Side             753
Washington Heights          664
West Village                650
Greenwich Village           317
Chinatown                   316
Kips Bay                    307
Financial District          293
SoHo                        277
```

```
Morningside Heights     265
Nolita                  262
Gramercy                239
Murray Hill             199
Inwood                  177
Theater District        137
Tribeca                 104
Little Italy             83
Flatiron District        70
NoHo                     68
Roosevelt Island         60
Two Bridges              48
Battery Park City        31
Civic Center             29
Stuyvesant Town          19
Unionport                 1
Marble Hill               1
Name: Neighbourhood cleansed, dtype: int64
```

[ ]:
```
top_manhattan = list(manhattan['Neighbourhood cleansed'].value_counts()[:10].
 ↪index)
top_manhattan
```

[ ]:
```
['Harlem',
 'East Village',
 'Upper West Side',
 "Hell's Kitchen",
 'Upper East Side',
 'East Harlem',
 'Chelsea',
 'Midtown',
 'Lower East Side',
 'Washington Heights']
```

[ ]:
```
top_manhattan_df = manhattan.loc[manhattan['Neighbourhood cleansed'].
 ↪isin(top_manhattan),:]
top_manhattan_df
```

[ ]:
```
       Listing ID  … Reviews per month
19296    10086969  …              1.00
19297    10087575  …              0.46
19299    10087874  …              0.74
19300    10088464  …              0.04
19302    10088612  …              3.43
…             …   …                …
39615    19831458  …              0.33
39616    19831736  …              0.41
```

```
39617     19833001  …              3.07
39618     19837851  …              0.04
39619     19838251  …              1.02

[11751 rows x 35 columns]
```

[ ]:
```
top_manhattan_neighborhoods = top_manhattan_df.groupby('Neighbourhood␣
↪cleansed')['Price','Review Scores Rating','Review Scores Location'].mean().
↪sort_values('Price',ascending = False)
top_manhattan_neighborhoods
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureWarning:
Indexing with multiple keys (implicitly converted to a tuple of keys) will be
deprecated, use a list instead.
  """Entry point for launching an IPython kernel.
```

[ ]:
```
                             Price  …  Review Scores Location
Neighbourhood cleansed              …
Midtown                 230.171533  …                9.731144
Chelsea                 212.780838  …                9.837126
Hell's Kitchen          194.803041  …                9.743664
Upper West Side         169.569358  …                9.760524
East Village            167.819506  …                9.663711
Upper East Side         159.398628  …                9.687500
Lower East Side         157.629482  …                9.491368
East Harlem             114.879096  …                8.813559
Harlem                  110.273781  …                9.102366
Washington Heights       84.551205  …                9.013554

[10 rows x 3 columns]
```

[ ]:
```
plt.style.use('seaborn-pastel')
```

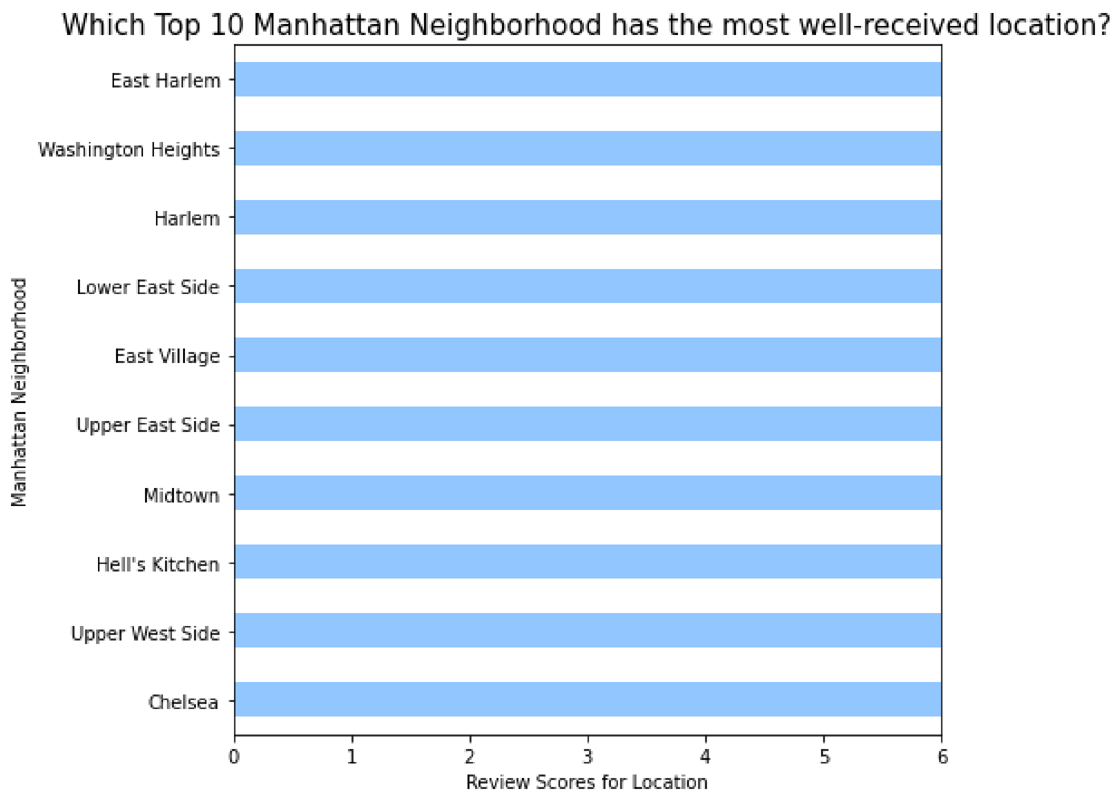
[ ]:
```
fig,ax=plt.subplots()
top_manhattan_neighborhoods['Price'].plot.barh(figsize=(7,7), ax=ax)

ax.set_title('Average Listing Price in Top 10 Manhattan Neighborhoods',size=15)
ax.set_xlabel('Average Price')
ax.set_ylabel('Manhattan Neighborhood')
```

[ ]:
```
Text(0, 0.5, 'Manhattan Neighborhood')
```

Of the 10 Manhattan neighborhoods with the most Airbnb listings, listings in Midtown have a substantially higher average price.

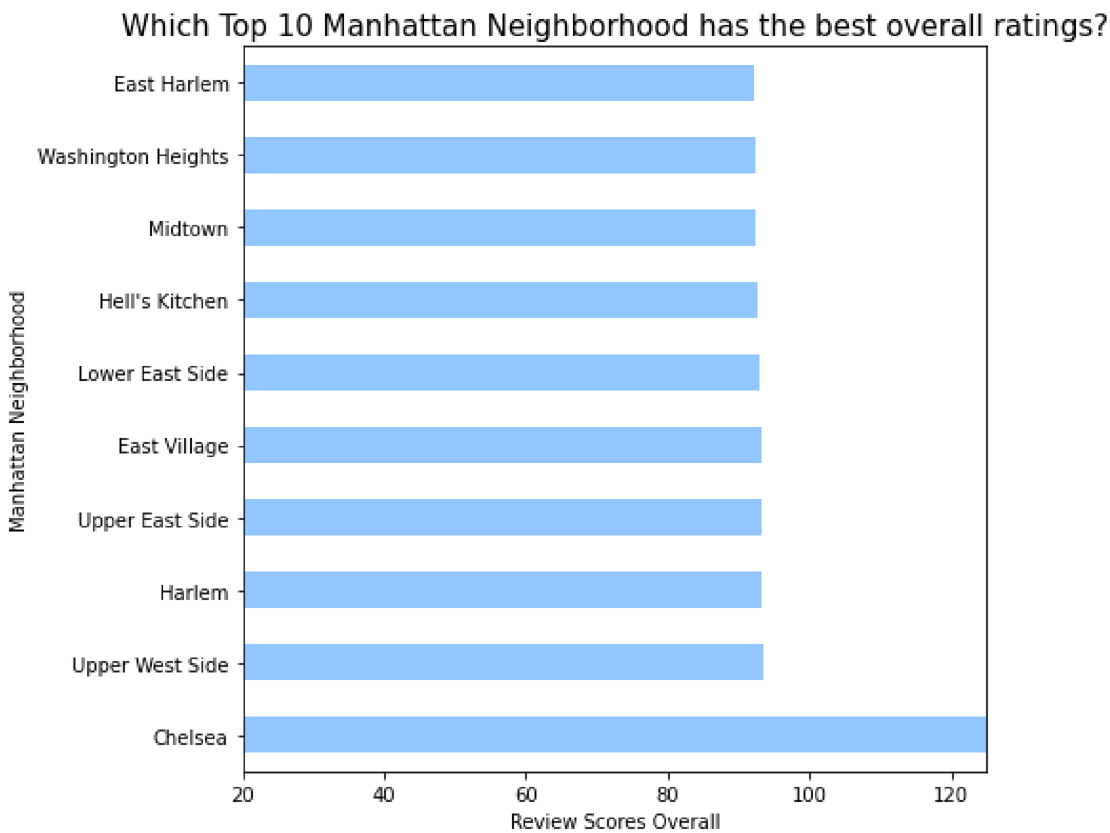
[ ]:
```
fig,ax=plt.subplots()
top_manhattan_neighborhoods['Review Scores Location'].
 ↪sort_values(ascending=False).plot.barh(figsize=(7,7), ax=ax)

ax.set_title('Which Top 10 Manhattan Neighborhood has the most well-received␣
 ↪location?',size=15)
ax.set_xlabel('Review Scores for Location')
ax.set_ylabel('Manhattan Neighborhood')

ax.set_xlim(0,6)
```

[ ]:
```
(0.0, 6.0)
```



[ ]:
```
fig,ax=plt.subplots()
top_manhattan_neighborhoods['Review Scores Rating'].
↪sort_values(ascending=False).plot.barh(figsize=(7,7), ax=ax)

ax.set_title('Which Top 10 Manhattan Neighborhood has the best overall ratings?
↪',size=15)
ax.set_xlabel('Review Scores Overall')
ax.set_ylabel('Manhattan Neighborhood')

ax.set_xlim(20,125)
```

[ ]:
```
(20.0, 125.0)
```



Despite having on average the most expensive listings, Midtown has the worst reviews scores for its location and overall. Chelsea's overall review scores are significantly higher than the other common neighborhoods.

[ ]:
```
all_manhattan_neighborhoods = manhattan.groupby('Neighbourhood␣
↪cleansed')['Price','Review Scores Rating','Review Scores Location'].mean().
↪sort_values('Price',ascending = False)
all_manhattan_neighborhoods
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureWarning:
Indexing with multiple keys (implicitly converted to a tuple of keys) will be
deprecated, use a list instead.
  """Entry point for launching an IPython kernel.
```

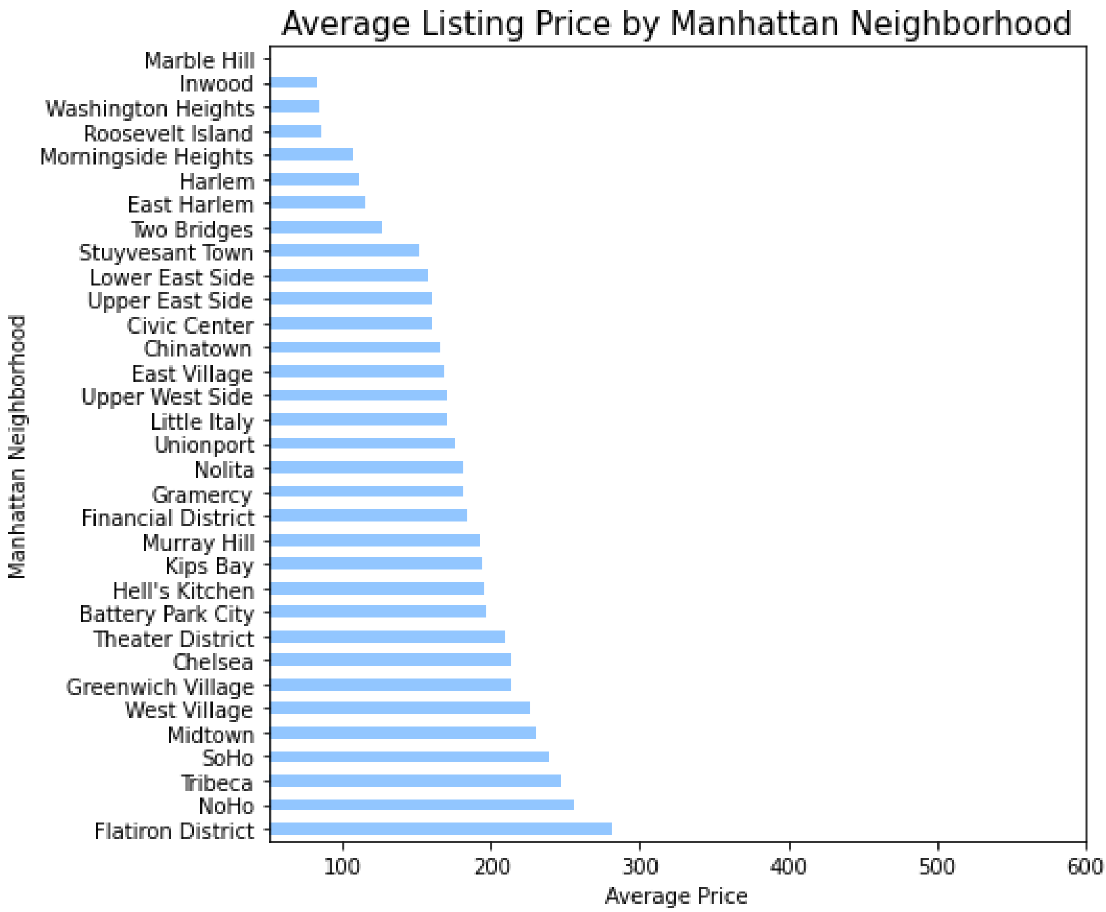
[ ]:

	Price	…	Review Scores Location
Neighbourhood cleansed		…	
Flatiron District	280.814286	…	9.900000
NoHo	256.308824	…	9.970588
Tribeca	246.942308	…	9.682692
SoHo	238.992780	…	9.758123
Midtown	230.171533	…	9.731144

```
West Village            226.041538  …           9.884615
Greenwich Village       214.006309  …           9.905363
Chelsea                 212.780838  …           9.837126
Theater District        209.751825  …           9.824818
Battery Park City       197.193548  …           9.935484
Hell's Kitchen          194.803041  …           9.743664
Kips Bay                193.084691  …           9.680782
Murray Hill             191.718593  …           9.723618
Financial District      183.290102  …           9.791809
Gramercy                181.464435  …           9.769874
Nolita                  180.698473  …           9.774809
Unionport               175.000000  …           9.000000
Little Italy            170.445783  …           9.638554
Upper West Side         169.569358  …           9.760524
East Village            167.819506  …           9.663711
Chinatown               165.515823  …           9.332278
Civic Center            159.862069  …           9.172414
Upper East Side         159.398628  …           9.687500
Lower East Side         157.629482  …           9.491368
Stuyvesant Town         151.421053  …           9.000000
Two Bridges             126.770833  …           9.125000
East Harlem             114.879096  …           8.813559
Harlem                  110.273781  …           9.102366
Morningside Heights     105.977358  …           9.592453
Roosevelt Island         85.016667  …           9.416667
Washington Heights       84.551205  …           9.013554
Inwood                   83.129944  …           9.039548
Marble Hill              40.000000  …           9.000000

[33 rows x 3 columns]
```

[ ]:
```
fig,ax=plt.subplots()
all_manhattan_neighborhoods['Price'].plot.barh(figsize=(7,7), ax=ax)

ax.set_title('Average Listing Price by Manhattan Neighborhood',size=15)
ax.set_xlabel('Average Price')
ax.set_ylabel('Manhattan Neighborhood')

ax.set_xlim(50,600)
```

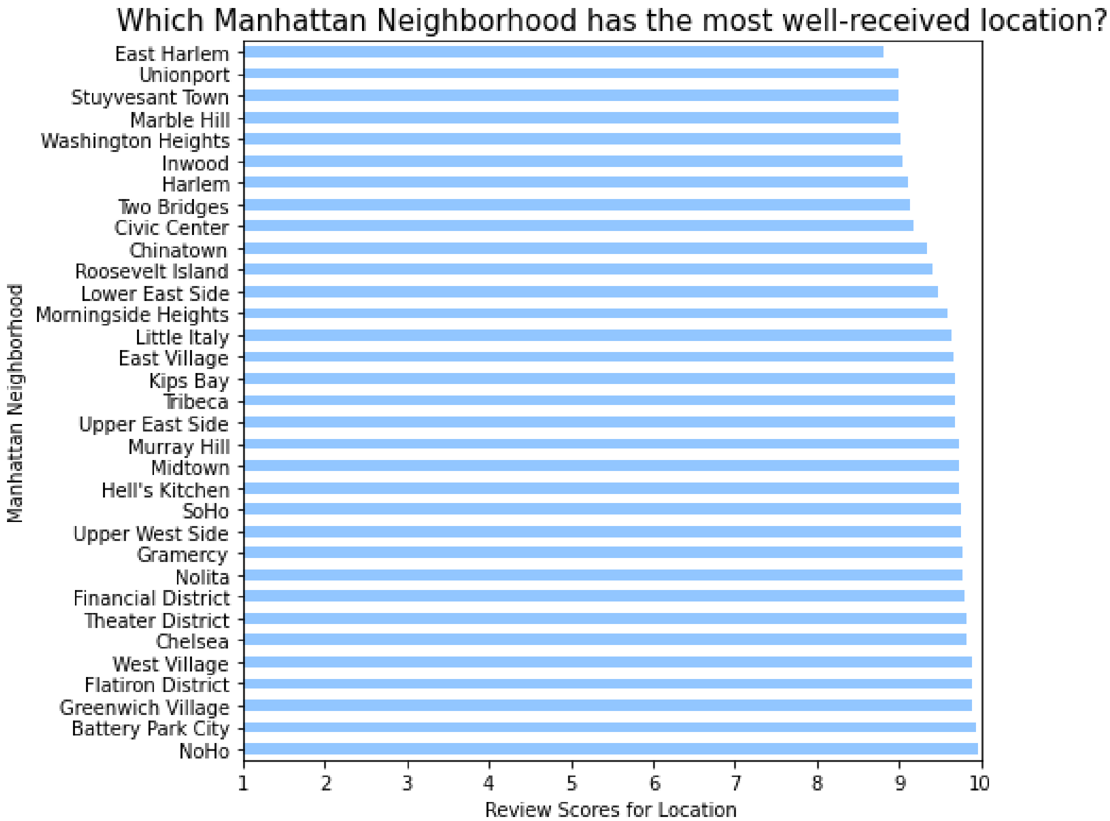
[ ]:
```
(50.0, 600.0)
```



While Midtown is the most pricey of the most popular 10 neighborhoods for listing, the Theater District (with less than 500 listings vs over 2000 in Midtown) has by far the most expensive average listing.

[ ]:
```
fig,ax=plt.subplots()
all_manhattan_neighborhoods['Review Scores Location'].
 ↪sort_values(ascending=False).plot.barh(figsize=(7,7), ax=ax)

ax.set_title('Which Manhattan Neighborhood has the most well-received location?
 ↪',size=15)
ax.set_xlabel('Review Scores for Location')
ax.set_ylabel('Manhattan Neighborhood')

ax.set_xlim(1,10)
```

[ ]:
```
(1.0, 10.0)
```



[ ]:

```

```

0.3 Top Keywords

[ ]:

```
keywords = []
keyword_count = []

def split_name(name):
    spl=str(name).split()
    return spl

for name in data.Name:
    keywords.append(name)

for x in keywords:
    for word in split_name(x):
        word=word.lower()
        keyword_count.append(word)
```

[ ]:
```
from collections import Counter

#show most frequently used words by host to name their listing
top_keywords_count = Counter(keyword_count).most_common()
top_keywords = [x[0] for x in top_keywords_count[:100]]
text = ' '.join(top_keywords)
text
```

[ ]:
```
'in room bedroom private apartment cozy the to studio apt brooklyn spacious 1 2
with of and east & manhattan park sunny beautiful near - williamsburg village
heart a large loft nyc central modern luxury from home new bright west bed for
1br charming one side upper midtown great location bushwick w/ quiet on br
brownstone close harlem clean 3 square huge times + garden by amazing prime
heights bath duplex min | subway 2br city big lovely house best renovated
chelsea york view comfortable apt. space gorgeous bathroom entire comfy prospect
suite train perfect mins floor hill place astoria'
```

[ ]:
```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

[ ]:
```
wordcloud = WordCloud(max_font_size=50, max_words=100,␣
↪background_color="white").generate(text)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

0.4 Regressions

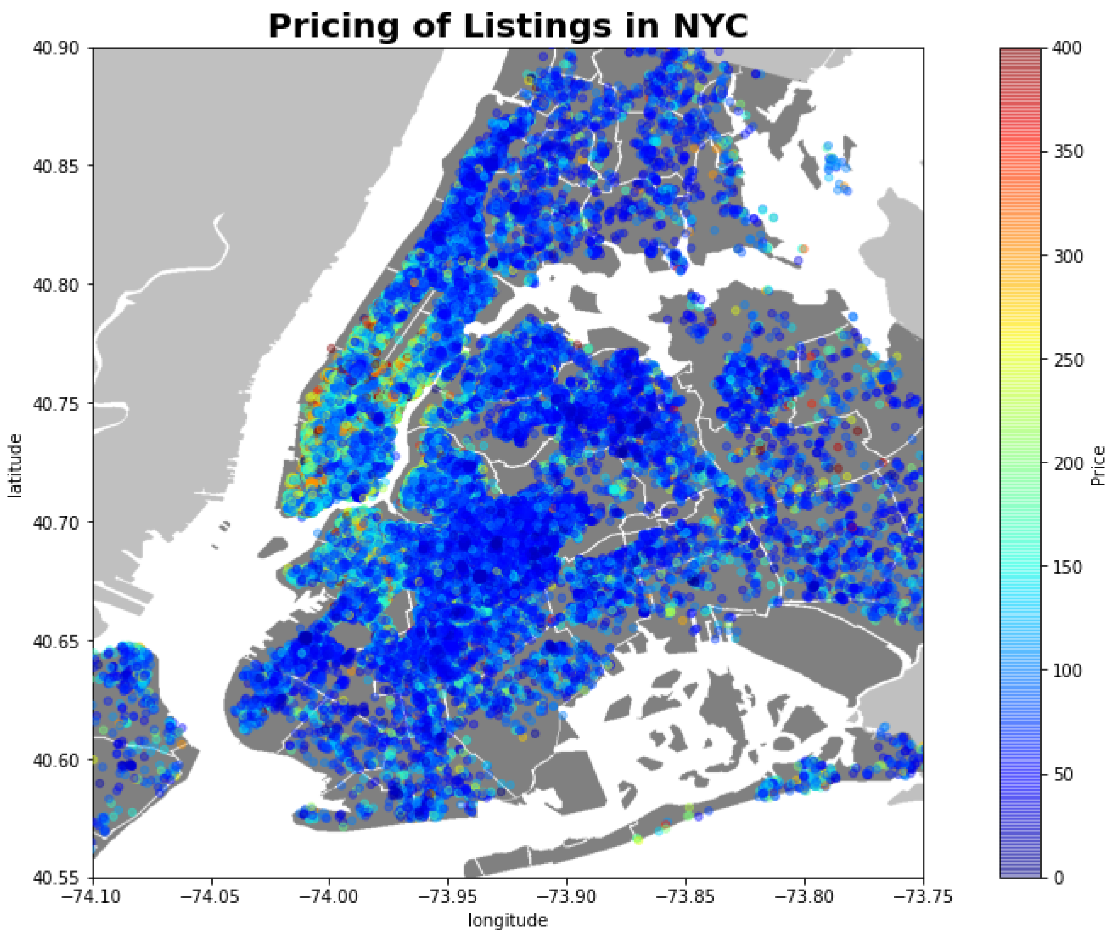
Are more expensive listings more highly received?

[ ]:
```
location = data.loc[data['Price'] <= 400][['latitude','longitude','Number of␣
↪reviews','Price']]
location
```

[ ]:
```
        latitude  longitude  Number of reviews  Price
0      40.866889 -73.857756                 66     43
1      40.829392 -73.865137                 38     28
2      40.869139 -73.895096                 18     80
3      40.868719 -73.891438                  7    140
4      40.863628 -73.894787                 56     60
…            …          …                 …      …
75744  40.641480 -73.960730                  0     22
75745  40.681430 -73.754610                  0     50
75746  40.647210 -74.014180                  0     45
75747  40.691970 -73.930030                  0     20
75748  40.759805 -73.991899                  0     58

[73185 rows x 4 columns]
```

[ ]:
```
import urllib

i=urllib.request.urlopen('https://upload.wikimedia.org/wikipedia/commons/e/ec/
↪Neighbourhoods_New_York_City_Map.PNG')
nyc_map = plt.imread(i)
plt.figure(figsize=(16,9))

plt.imshow(nyc_map, zorder=0, extent=[-74.258, -73.7, 40.49,40.92])
ax=plt.gca()

x = range(300)
ax.imshow(img, extent=[0, 400, 0, 300])

location.plot.scatter(ax=ax, figsize = (16,9), x='longitude',y='latitude',␣
↪c='Price', cmap=plt.get_cmap('jet'), colorbar=True, alpha=0.4, zorder=5)

ax.set_xlim(-74.1,-73.75)
ax.set_ylim(40.55,40.9)
ax.set_title('Pricing of Listings in NYC', size=20, fontweight='bold')
```

[ ]:
```
Text(0.5, 1.0, 'Pricing of Listings in NYC')
```




```
[ ]: reviews = data[['Review Scores Rating','Review Scores Accuracy','Review Scores␣
     ↪Cleanliness','Review Scores Checkin','Review Scores Communication','Review␣
     ↪Scores Location','Review Scores Value','Reviews per month','Price']]

     #drop extreme ratings where rating out of 100 was 0
     reviews.drop(reviews.loc[reviews['Review Scores Rating']==0].index,inplace=True)
     reviews
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:4174:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

```
[ ]:        Review Scores Rating  Review Scores Accuracy  …  Reviews per month
     Price
```

```
0                        96                       10  …                   1.77
43
1                        89                       10  …                   1.54
28
2                        90                        9  …                   3.83
80
3                        85                        9  …                   0.67
140
4                        95                       10  …                   3.70
60
…                      …                      …  …                …
…
44301                    99                       10  …                   3.44
80
44303                    95                        9  …                   0.40
150
44305                   100                       10  …                   2.25
70
44306                    98                       10  …                   4.76
95
44308                    98                       10  …                   0.46
50

[34218 rows x 9 columns]
```

[314]:
```python
reviews[['Review Scores Rating','Price']].corr()
```

[314]:

	Review Scores Rating	Price
Review Scores Rating	1.000000	0.006254
Price	0.006254	1.000000

[319]:
```python
reviews[['Review Scores Accuracy','Price']].corr()
```

[319]:

	Review Scores Accuracy	Price
Review Scores Accuracy	1.0000	0.0124
Price	0.0124	1.0000

[318]:
```python
reviews[['Review Scores Cleanliness','Price']].corr()
```

[318]:

	Review Scores Cleanliness	Price
Review Scores Cleanliness	1.000000	0.065998
Price	0.065998	1.000000

[317]:
```python
reviews[['Review Scores Location','Price']].corr()
```

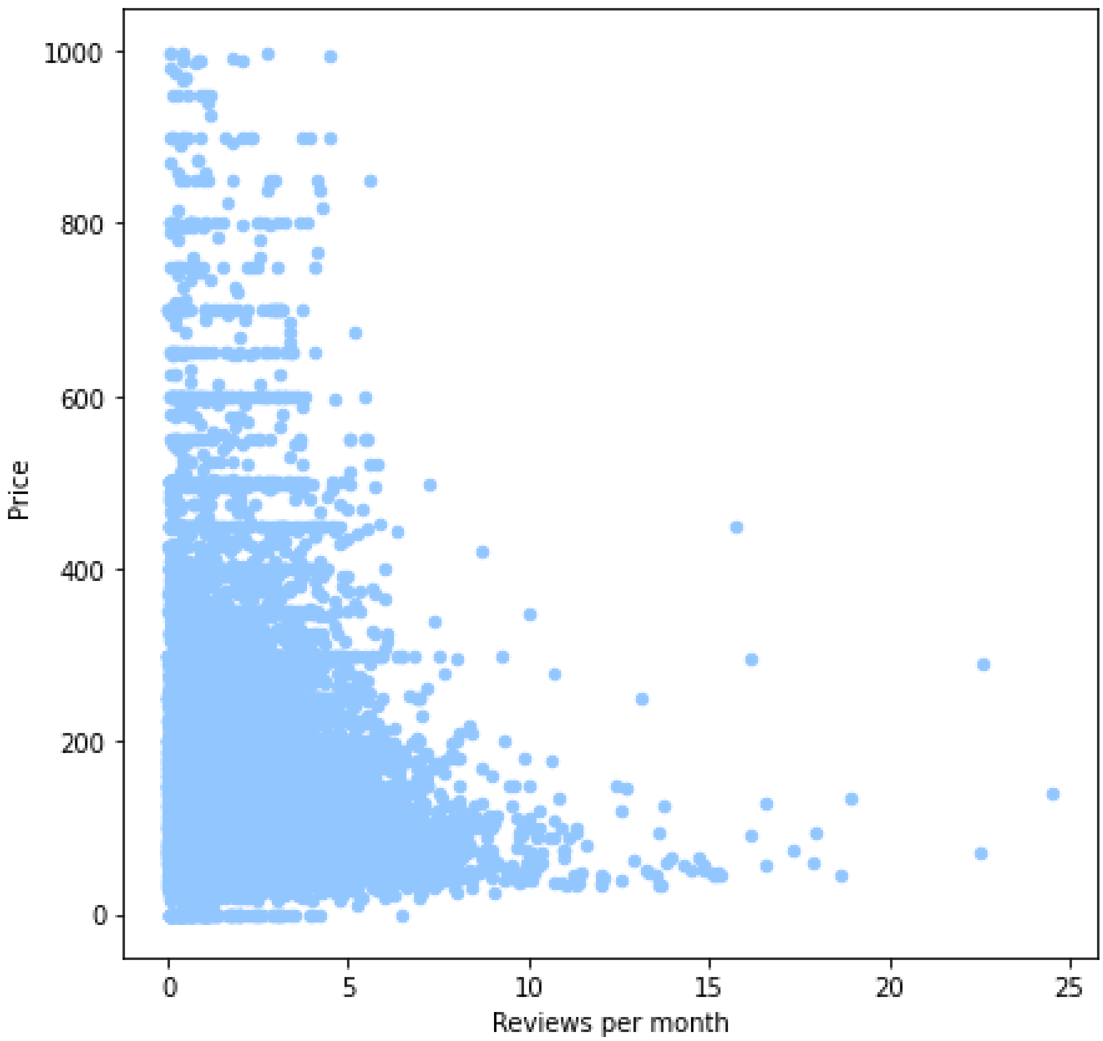
[317]:

	Review Scores Location	Price
Review Scores Location	1.000000	0.131166
Price	0.131166	1.000000

The correlations between these review scor types and price are not substantal/strong.

[316]:
```
reviews.plot.scatter(x='Reviews per month', y='Price', figsize=(7,7)) #FIX
```

[316]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3739b60630>



While many listings have very few reviews per month, the more popular listings (more reviews/month) are all on the cheaper end.

1 ML Introduction

The purpose of this section will be to determine whether we can use machine learning to provide insights to Airbnb hosts about variables keen to their listings. We will examine which numeric dependent variables in the data set can be reliably determined using a KNN machine learning model. The reliability of the models will be demonstrated through value counts distributions, density plots, and various accuracy metrics. Both KNN Classifier and KNN Regressor versions of the models will be tested to determine the effectiveness of each on the different dependent variables.

The variables examined will be review scores rating, review scores accuracy, review scores cleanliness, review scores checkin, review scores communication, review scores location, review scores value, and price. These variables were deemed to be most important to Airbnb hosts regarding their listings.

The non-numeric columns of the data were excluded from the model as their current data types are non-compatible for execution, and the encoding of the non-numeric types was outside the breadth of the class instruction.

For the purpose of runtime speed, the maximum number of n_neighbors for the KNN model was limited to 10. In tests that exceeded this number, the runtime had extended past 10 minutes per variable - totalling a runtime over 80 minutes for all of the dependent variables. Further, the incremental accuracy of n_neighbors past 10 seemed to be non-substantial or non-existent in all of the cases. For robustness, it would be recommended to increase the n_neighbors size, but for convenience/efficiency, not at all.

Density plots were specifically chosen in order to detect overfitting of the model for any of the variables. Given that we are proceeding with a maximum number of n_neighbors on the smaller end, it is possible that the model could use an n_neighbors size of 1 to best fit and predict the data. Another benefit of the density plots is the ability to plot a curve of the data, which will show the skewness of the distribution and provide insight into both the data set and the model's decision making. Each variable contains three unique density plots: 1) a bar density histogram plotting the actual data 2) a bar density histogram plotting the classifier model predictions 3) a step density histogram plotting the regressor model predictions. The step density was used since the regressor model predictions are not required to be classified as A or B - they can be between A and B.

[54]:
```
# import relevant modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.style as style
import statsmodels.formula.api as smf
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import scipy.stats as stats
```

[55]:
```
# read in data set, intialize plot styling
df = pd.read_csv('NY_Listings.csv', encoding='Latin-1')
style.use('ggplot')
plt.rcParams.update({'font.size': 16})
```

```python
# clean data of non-numeric columns (can't encode for ML section)
non_floats = []
for col in df:
    if df[col].dtypes not in ['float64','int64','bool']:
        non_floats.append(col)

df = df.drop(columns=non_floats)

# drop outliers from the data
df = df.dropna()
df.drop(df.loc[df['Review Scores Rating']==38425].index, inplace=True)
```

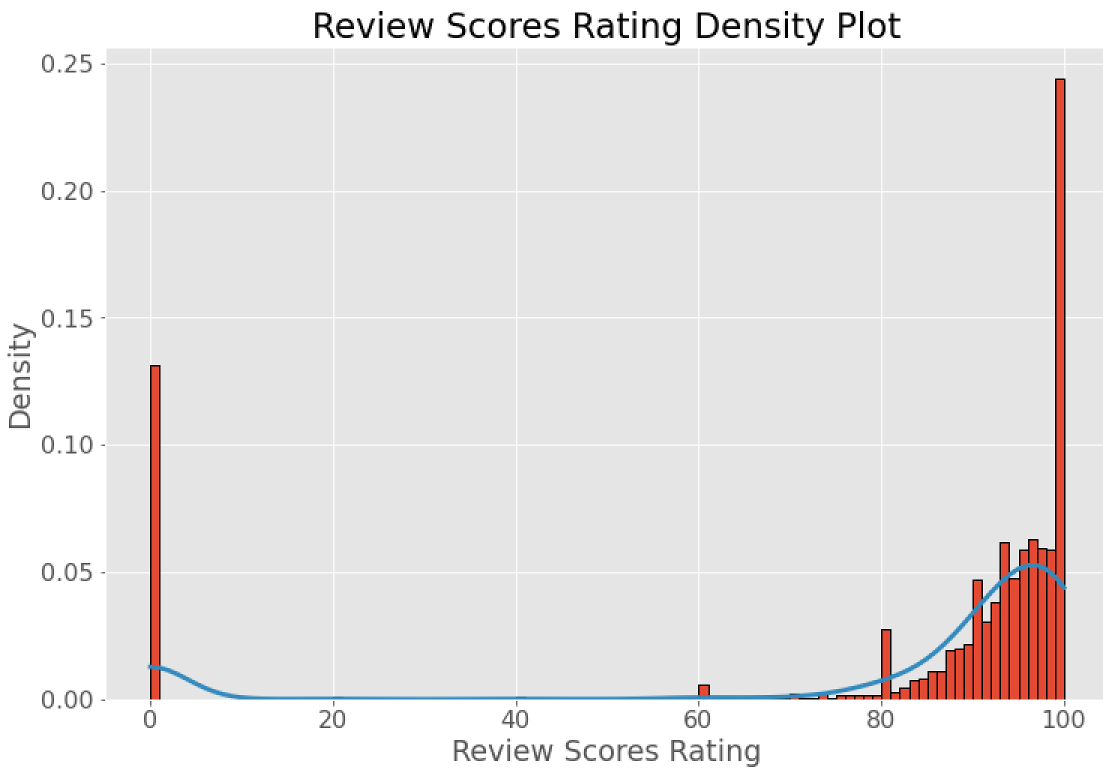
```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/IPython/core/interactiveshell.py:3146: DtypeWarning: Columns (7,19,24)
have mixed types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[56]:
```python
# dependent variables to predict
preds = ['Review Scores Rating','Review Scores Accuracy','Review Scores␣
↪Cleanliness',
         'Review Scores Checkin','Review Scores Communication','Review Scores␣
↪Location',
        'Review Scores Value','Price']

# loop through variables for predictions
for pred in preds:

    # display percentages of values for the data
    print('---------- Beginning of',pred,'----------', end='\n\n')
    #print(df[pred].value_counts(normalize=True).
↪sort_index(ascending=False)*100,end='\n\n')

    # display density plot of the values for the data
    fig = plt.figure(figsize=(12,8))
    noise = df[pred]
    density = stats.gaussian_kde(noise)
    n, x, _ = plt.hist(noise, bins=range(min(noise), max(noise) + 1, 1),␣
↪histtype='bar', density=True, linewidth=1, edgecolor='k')
    plt.xlabel(pred)
    plt.ylabel('Density')
    plt.title(pred+' Density Plot')
    plt.plot(x, density(x), linewidth=3)
    plt.show()

    # drop the variable to predict for train test split initializtion
    x = df.drop(pred, axis=1)
```

```
    y = df[pred]
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.20)

    # scale the data
    scaler = StandardScaler()
    scaler.fit(x_train)
    x_train = scaler.transform(x_train)
    x_test = scaler.transform(x_test)

    # set parameters and initialize KNN
    params = {'n_neighbors': np.arange(1, 11)}
    knn = KNeighborsClassifier()

    # determine the best number of n_neighbors using GridSearchCV
    model = GridSearchCV(knn, params, cv=10, scoring='accuracy')
    model.fit(x_train,y_train)
    print('----------',pred+' KNN Model','----------')
    print('\nModel Best Params:', model.best_params_)
    print('Model Best Score:', model.best_score_*100)

    # use the ideal parameters to fit the training data and make predictions
    best_knn = KNeighborsClassifier(n_neighbors=model.
↪best_params_['n_neighbors'])
    best_knn.fit(x_train, y_train)
    y_pred = best_knn.predict(x_test)
    vals = pd.Series(y_pred)

    best_reg = KNeighborsRegressor(n_neighbors=model.
↪best_params_['n_neighbors'])
    best_reg.fit(x_train, y_train)
    reg_pred = best_reg.predict(x_test)
    reg_vals = pd.Series(reg_pred)

    # summarize results
    #print(vals.value_counts(normalize=True).sort_index(ascending=False)*100)
    print('\n- Classifier -')
    print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
    print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
    print('Root Mean Squared Error:', np.sqrt(metrics.
↪mean_squared_error(y_test, y_pred)))
    print('R2:', metrics.r2_score(y_test, y_pred))
    print('Metrics Accuracy:', accuracy_score(y_pred,y_test)*100)

    print('\n- Regressor -')
    print('Mean Squared Error:', metrics.mean_squared_error(y_test, reg_pred))
    print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, reg_pred))
```

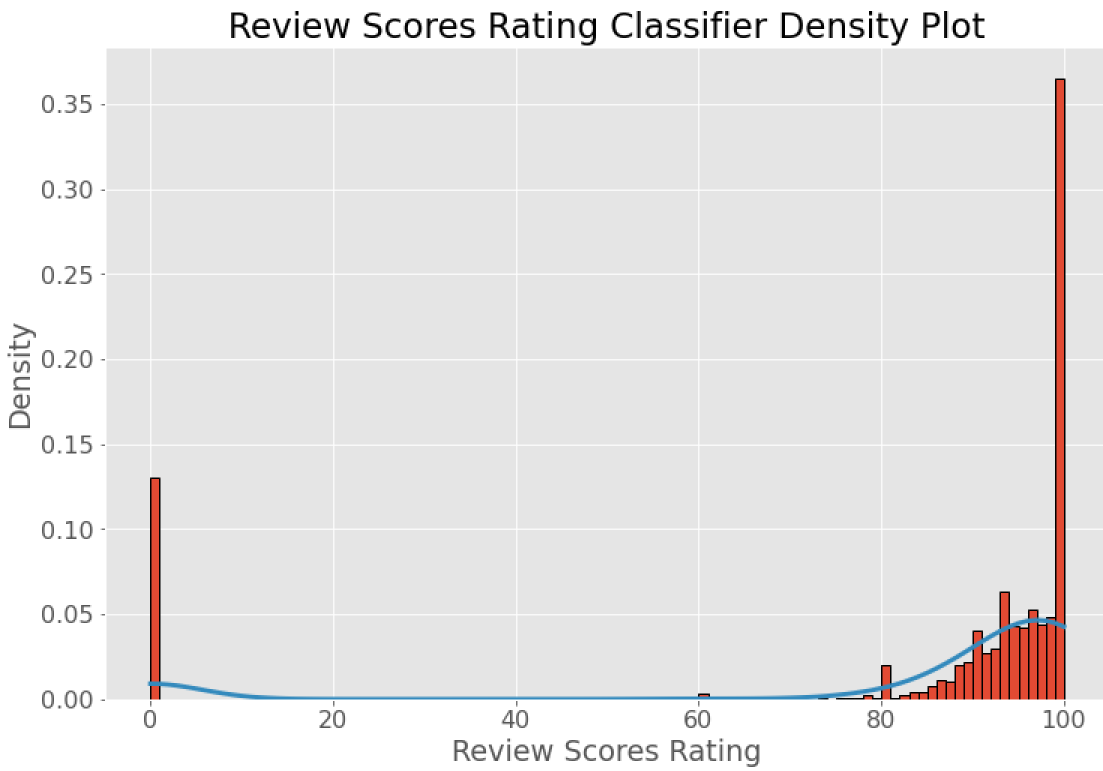
```
    print('Root Mean Squared Error:', np.sqrt(metrics.
↪mean_squared_error(y_test, reg_pred)))
    print('R2:', metrics.r2_score(y_test, reg_pred), end='\n\n')

    # plot results in a density plot
    fig = plt.figure(figsize=(12,8))
    noise = vals
    density = stats.gaussian_kde(noise)
    n, x, _ = plt.hist(noise, bins=range(min(noise), max(noise) + 1, 1),␣
↪histtype='bar', density=True, linewidth=1, edgecolor='k')
    plt.xlabel(pred)
    plt.ylabel('Density')
    plt.title(pred+' Classifier Density Plot')
    plt.plot(x, density(x), linewidth=3)
    plt.show()

    fig = plt.figure(figsize=(12,8))
    noise = reg_vals
    density = stats.gaussian_kde(noise)
    n, x, _ = plt.hist(noise, bins='auto', histtype=u'step', density=True,␣
↪linewidth=3)
    plt.xlabel(pred)
    plt.ylabel('Density')
    plt.title(pred+' Regressor Density Plot')
    plt.plot(x, density(x), linewidth=3)
    plt.show()
    print('---------- End of',pred,'----------', end='\n\n')
```

```
---------- Beginning of Review Scores Rating ----------
```

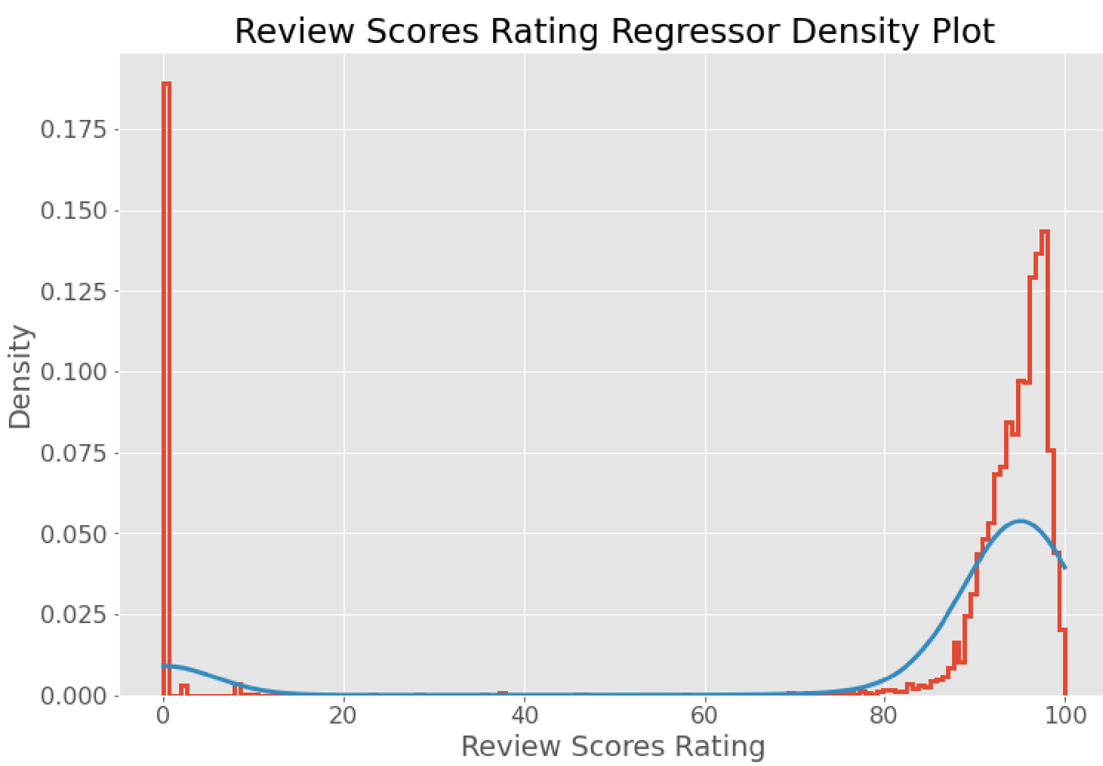



```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:672: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=10.
  % (min_groups, self.n_splits)), UserWarning)
```

```
---------- Review Scores Rating KNN Model ----------

Model Best Params: {'n_neighbors': 10}
Model Best Score: 36.43471142038477

- Classifier -
Mean Squared Error: 45.212180746561884
Mean Absolute Error: 3.6254092992796334
Root Mean Squared Error: 6.724000352956704
R2: 0.9560347400620975
Metrics Accuracy: 34.82318271119843

- Regressor -
Mean Squared Error: 31.464392599869026
Mean Absolute Error: 3.2041093647675183
Root Mean Squared Error: 5.609313023879932
R2: 0.9694033736794117
```

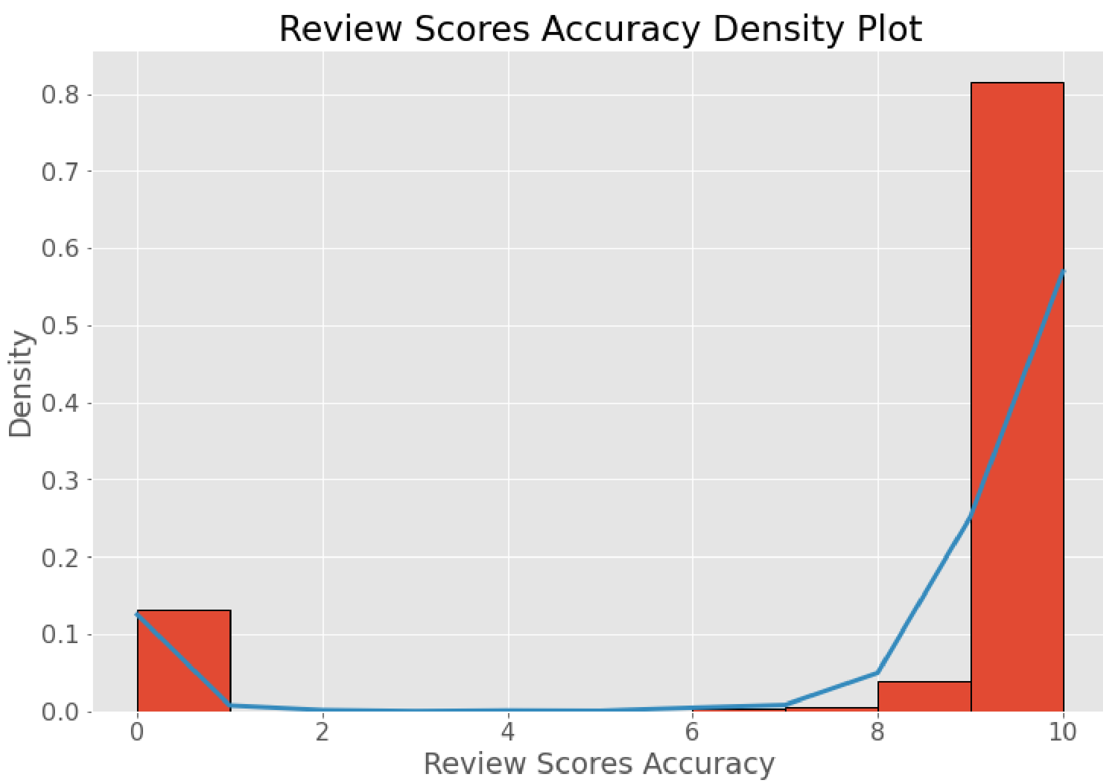


Review Scores Rating Classifier Density Plot
Density
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
Review Scores Rating



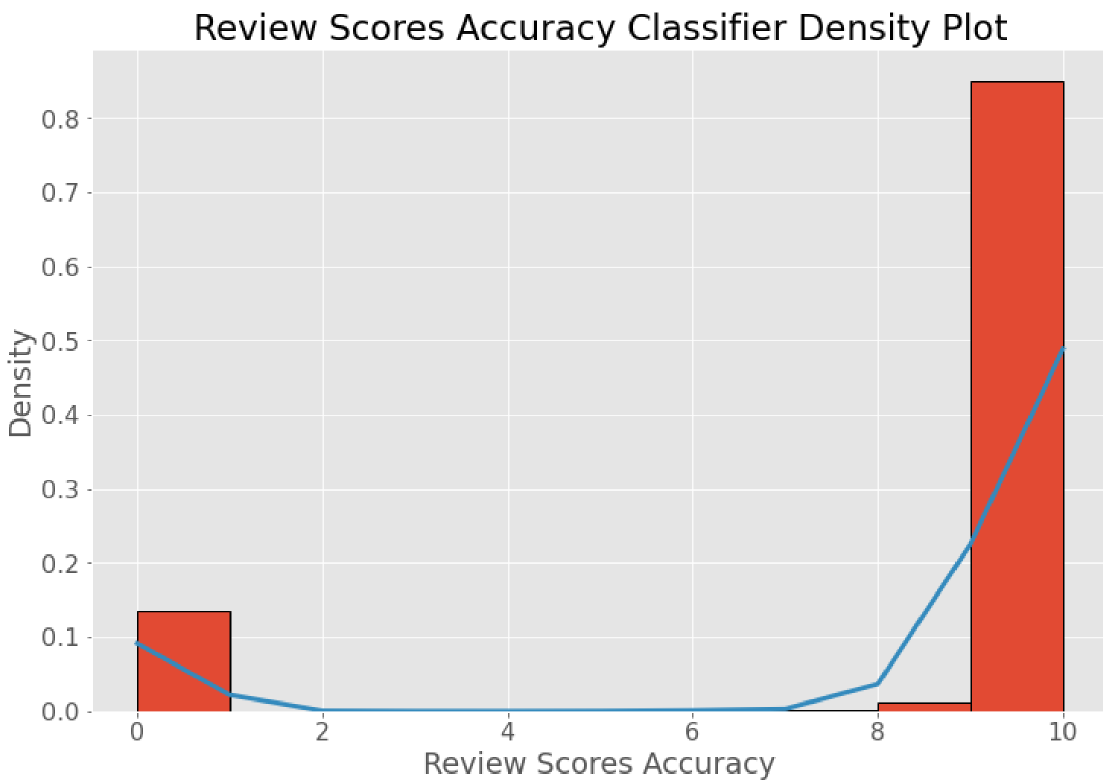
Review Scores Rating Regressor Density Plot
Density
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
20
40
60
80
100
Review Scores Rating

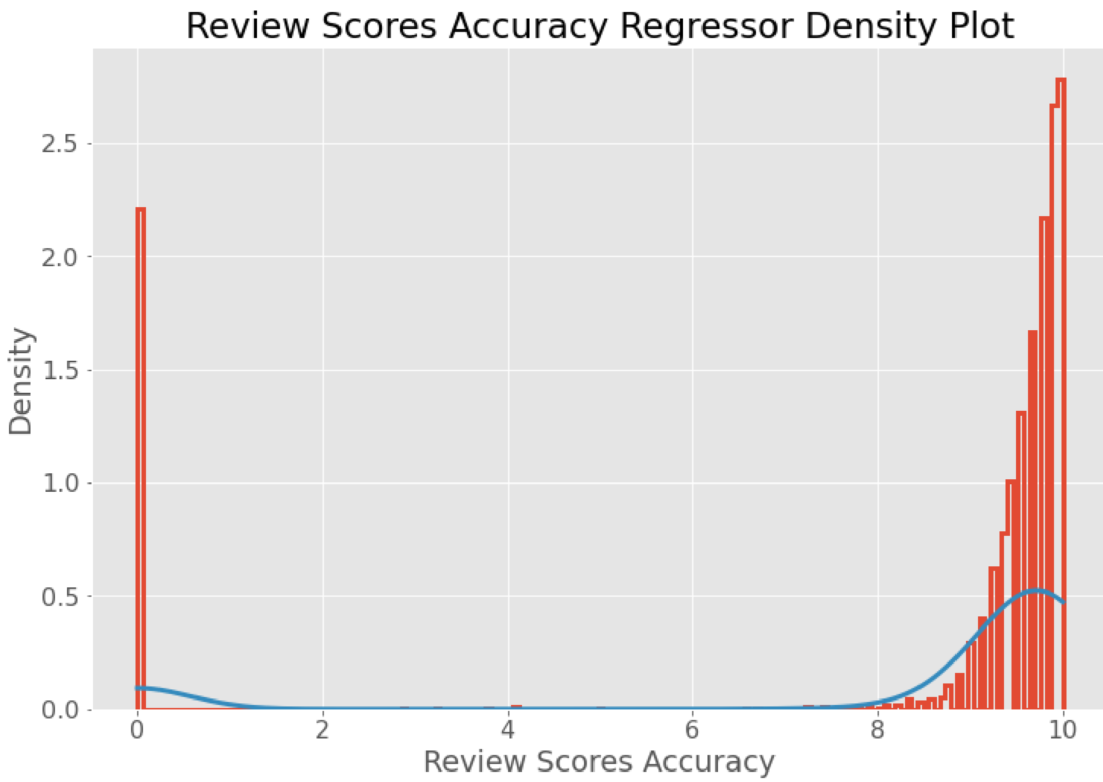
```
---------- End of Review Scores Rating ----------

---------- Beginning of Review Scores Accuracy ----------
```



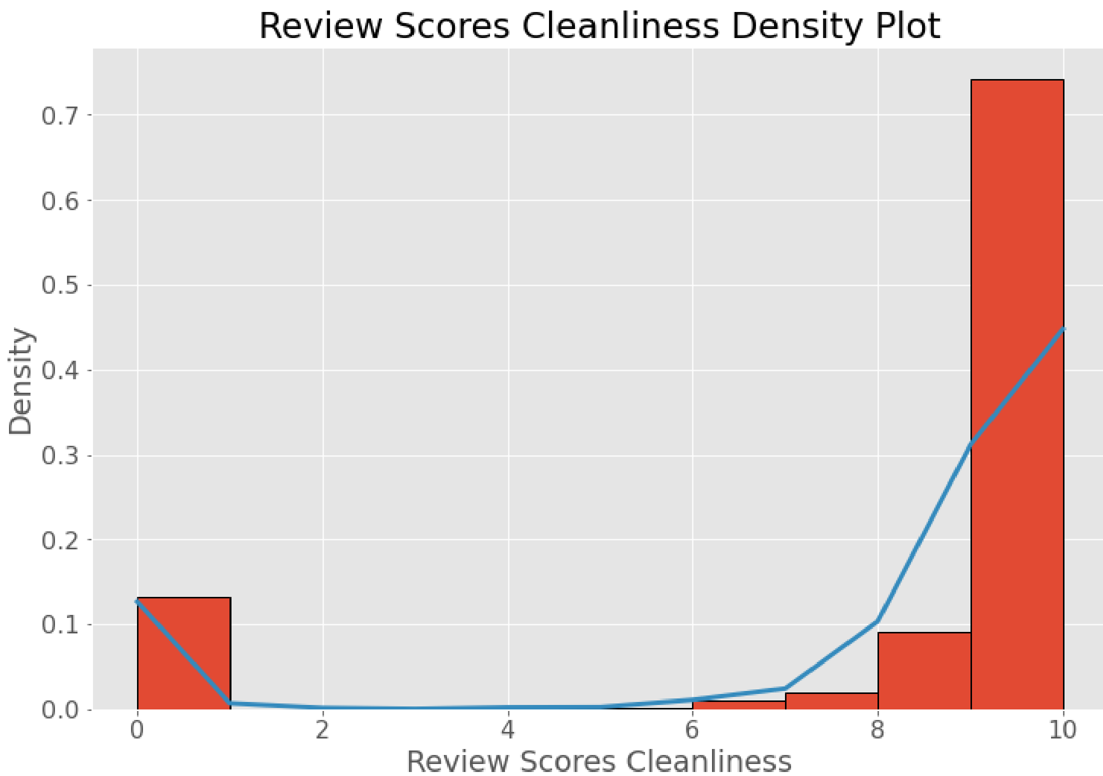

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:672: UserWarning: The least populated
class in y has only 2 members, which is less than n_splits=10.
  % (min_groups, self.n_splits)), UserWarning)

---------- Review Scores Accuracy KNN Model ----------

Model Best Params: {'n_neighbors': 9}
Model Best Score: 75.59148587801883

- Classifier -
Mean Squared Error: 0.47380484610347084
Mean Absolute Error: 0.28323510150622133
Root Mean Squared Error: 0.6883348357474514
R2: 0.9576922531219012
Metrics Accuracy: 76.34250163719712
```

```
- Regressor -
Mean Squared Error: 0.35181142723164116
Mean Absolute Error: 0.33649858109583053
Root Mean Squared Error: 0.5931369380097998
R2: 0.9685854863356791
```



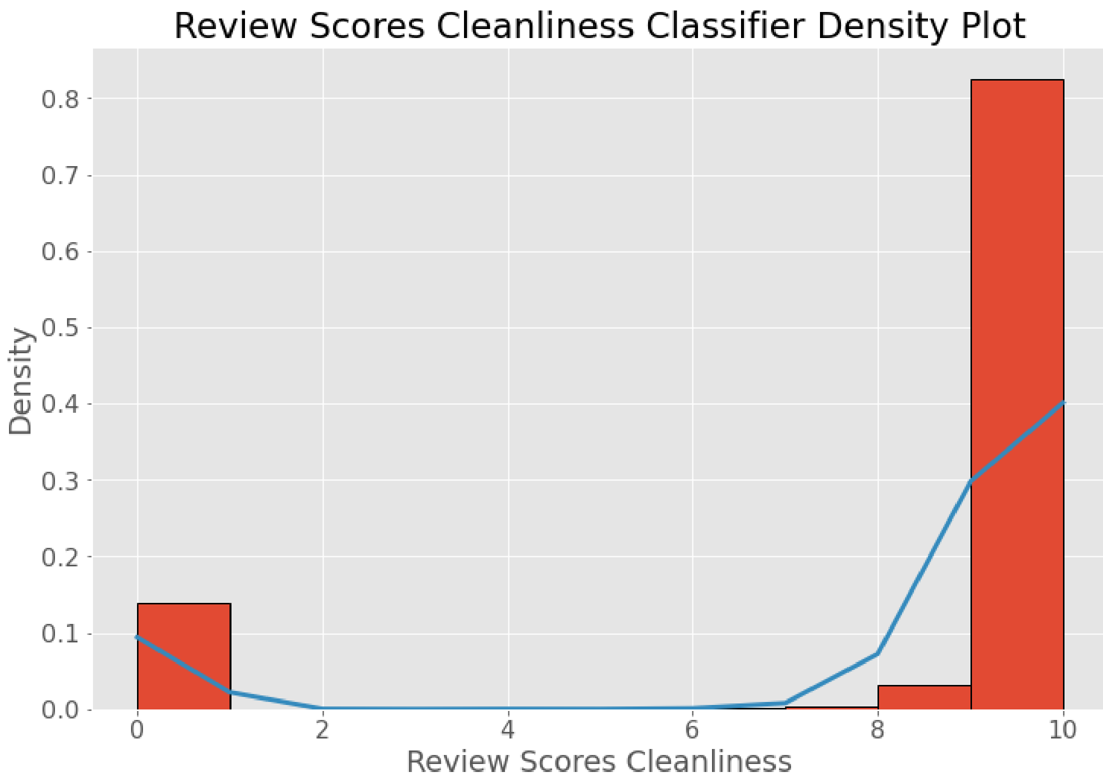


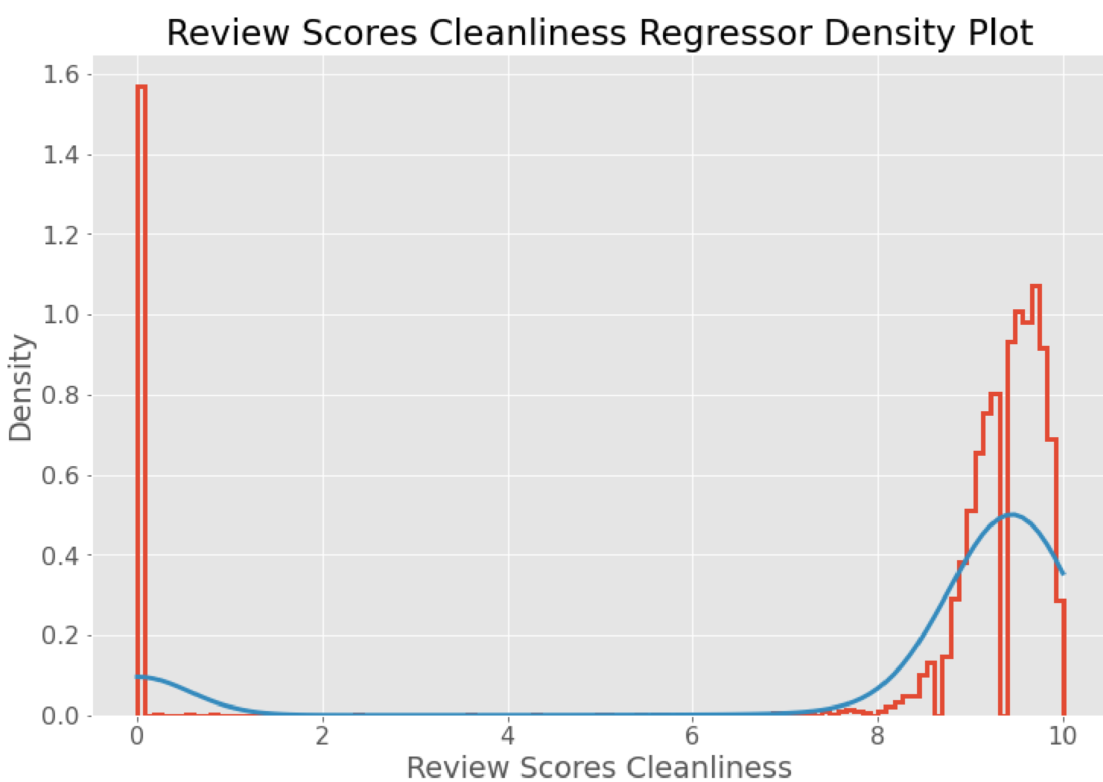

```
---------- End of Review Scores Accuracy ----------

---------- Beginning of Review Scores Cleanliness ----------
```



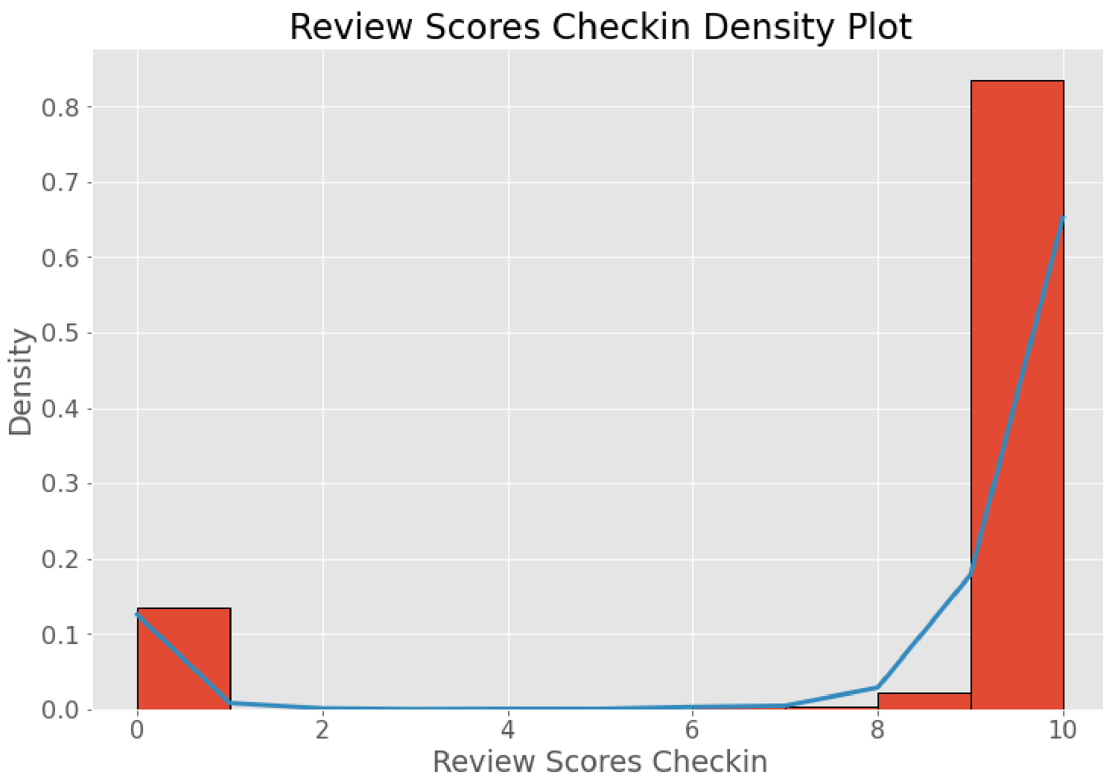

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:672: UserWarning: The least populated
class in y has only 5 members, which is less than n_splits=10.
  % (min_groups, self.n_splits)), UserWarning)
```

```
---------- Review Scores Cleanliness KNN Model ----------

Model Best Params: {'n_neighbors': 10}
Model Best Score: 63.39746213671715

- Classifier -
Mean Squared Error: 0.8115586116568435
Mean Absolute Error: 0.46414538310412573
Root Mean Squared Error: 0.9008654792236428
R2: 0.9258483859315502
Metrics Accuracy: 64.1290111329404

- Regressor -
Mean Squared Error: 0.6108104125736739
Mean Absolute Error: 0.5034872298624754
Root Mean Squared Error: 0.7815436088752015
R2: 0.9441906261216473
```


Review Scores Cleanliness Classifier Density Plot
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2
4
6
8
10
Review Scores Cleanliness

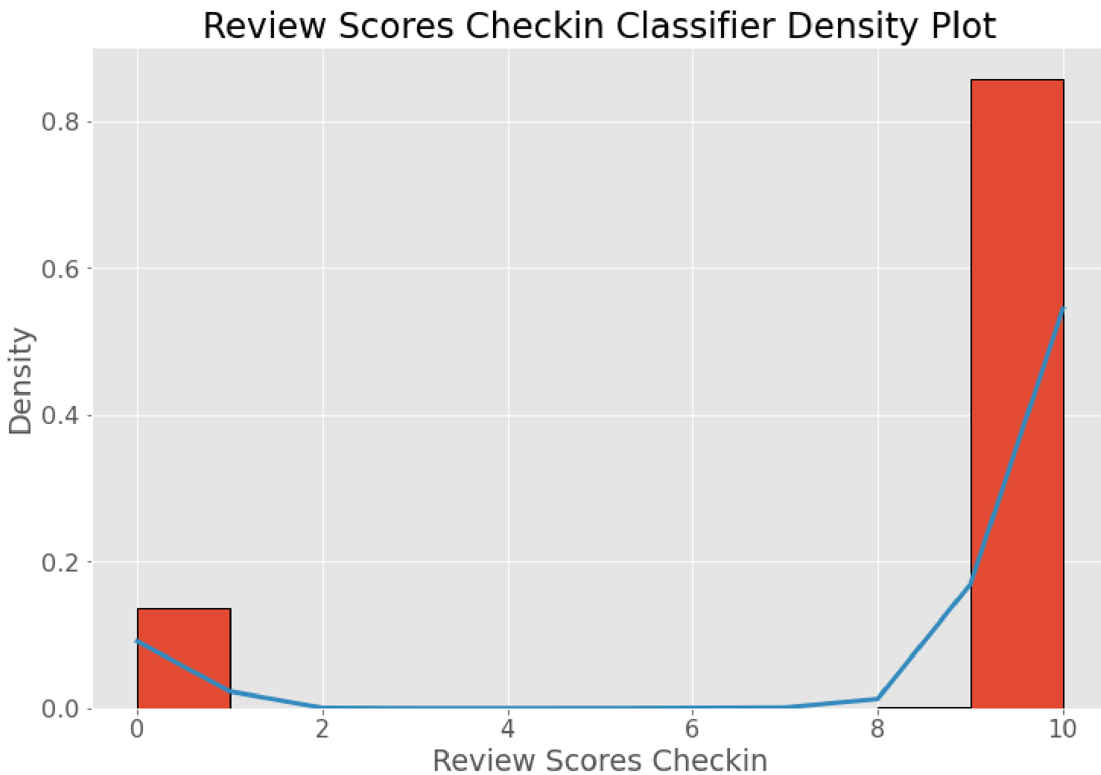

Review Scores Cleanliness Regressor Density Plot
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
2
4
6
8
10
Review Scores Cleanliness

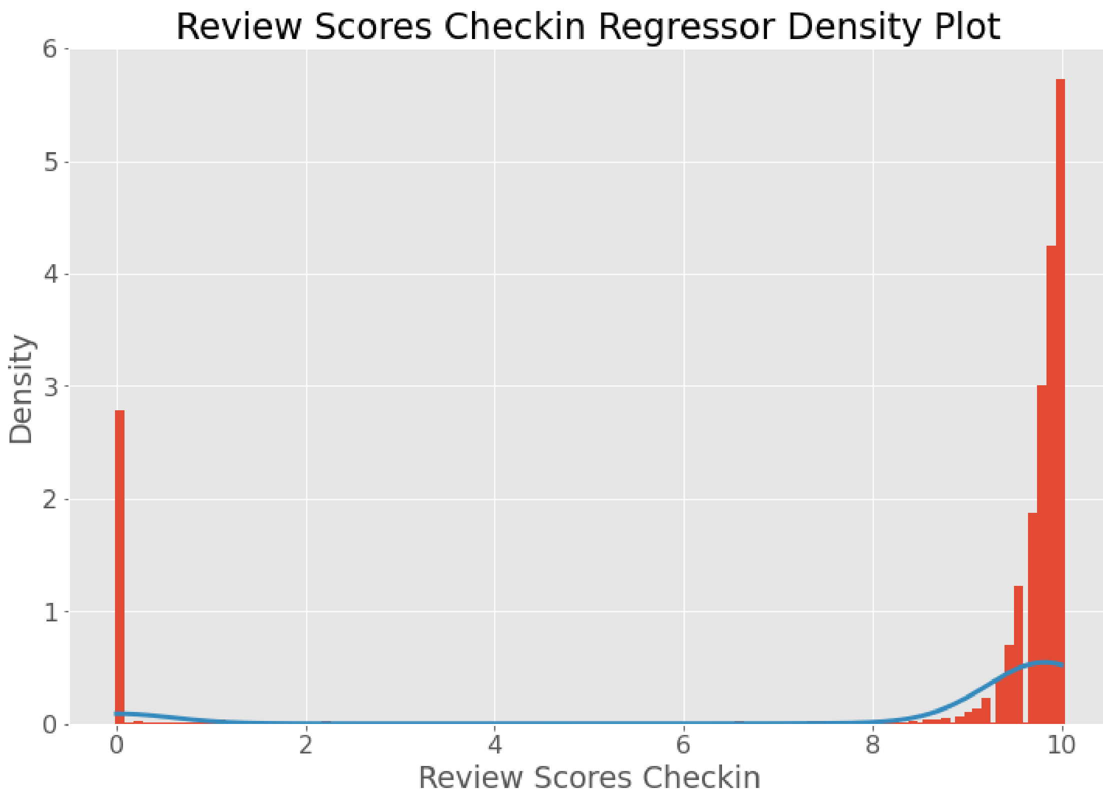
```
---------- End of Review Scores Cleanliness ----------

---------- Beginning of Review Scores Checkin ----------
```




```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:672: UserWarning: The least populated
class in y has only 4 members, which is less than n_splits=10.
  % (min_groups, self.n_splits)), UserWarning)

---------- Review Scores Checkin KNN Model ----------

Model Best Params: {'n_neighbors': 9}
Model Best Score: 83.19688907081458

- Classifier -
Mean Squared Error: 0.44138834315651604
Mean Absolute Error: 0.2154551407989522
Root Mean Squared Error: 0.6643706368861556
R2: 0.9617314388862159
Metrics Accuracy: 83.28421741977733
```

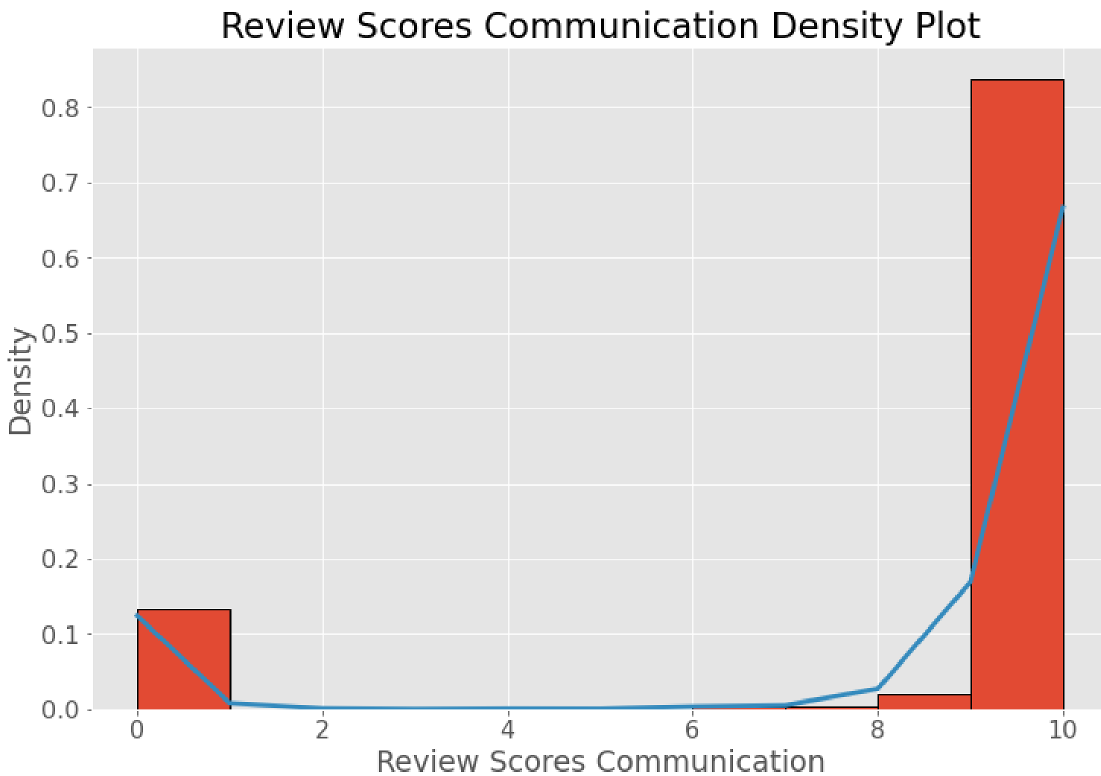
```
- Regressor -
Mean Squared Error: 0.3484056529788902
Mean Absolute Error: 0.2729207596594629
Root Mean Squared Error: 0.5902589711125873
R2: 0.9697930785211456
```





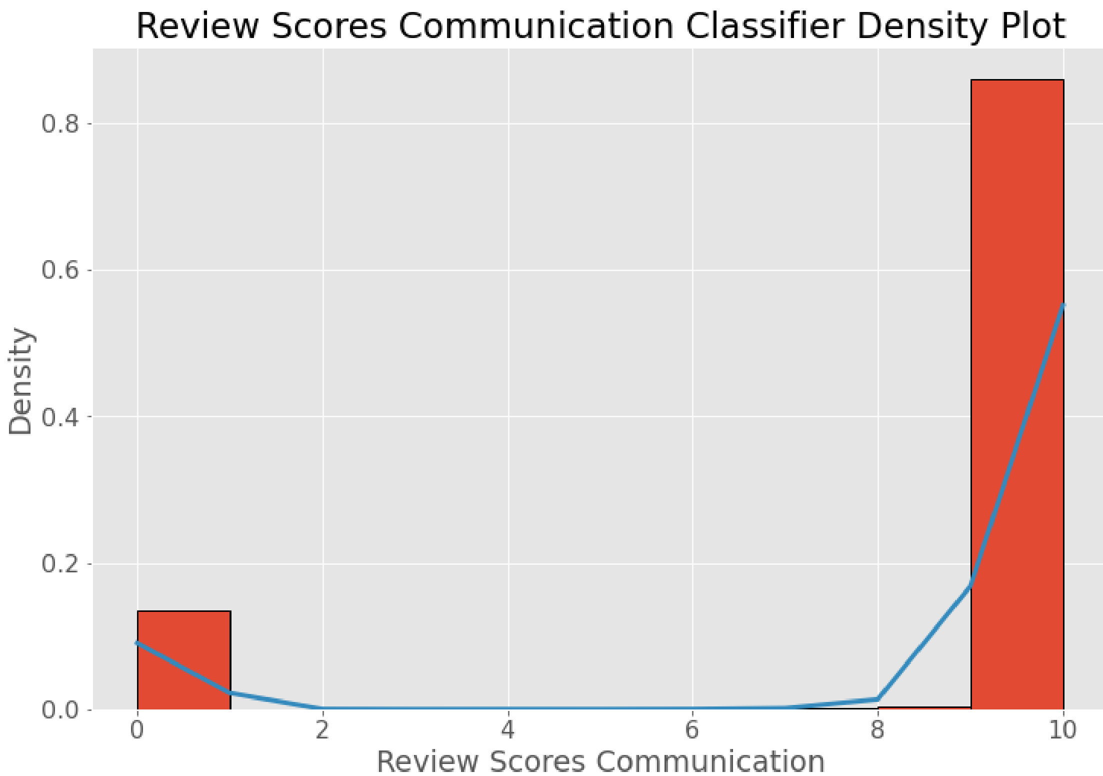
---------- End of Review Scores Checkin ----------

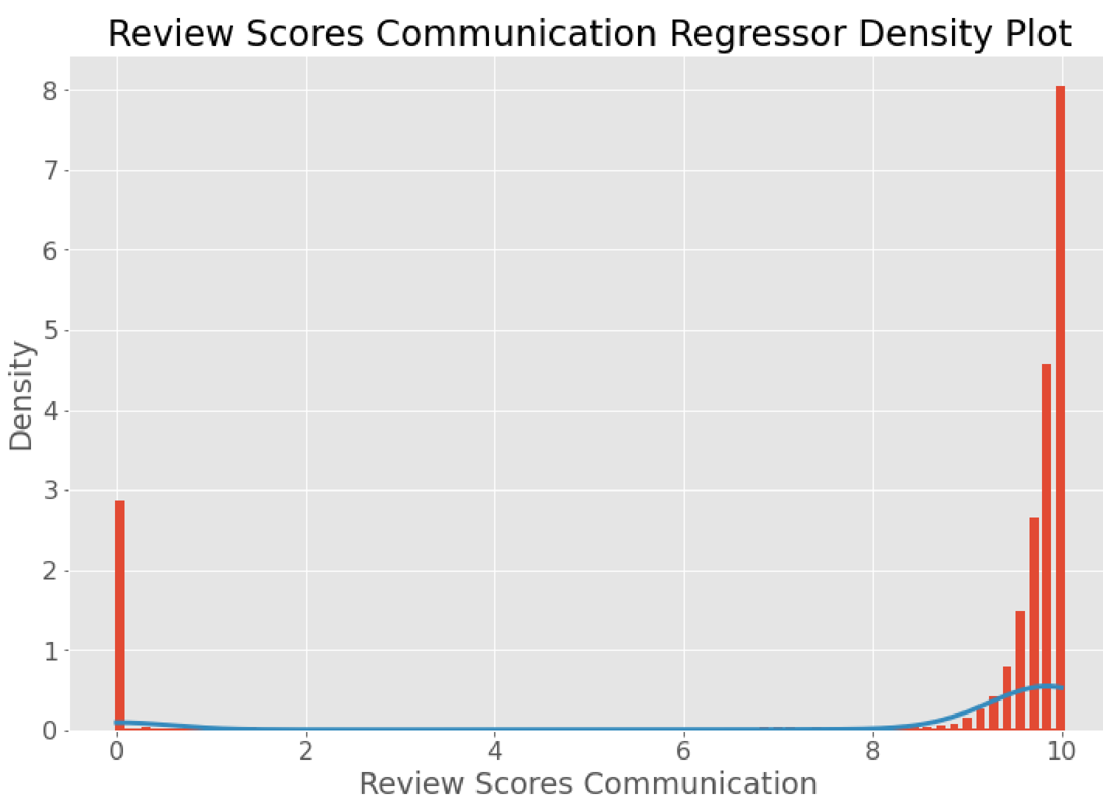
---------- Beginning of Review Scores Communication ----------



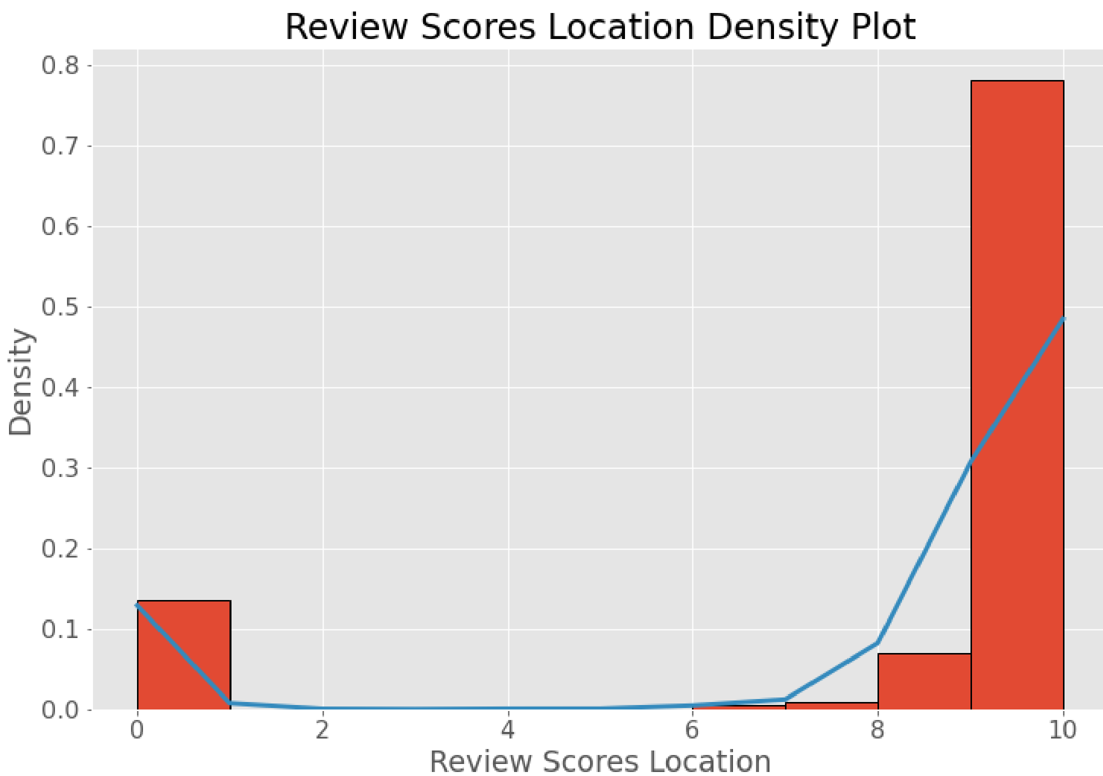

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:672: UserWarning: The least populated
class in y has only 2 members, which is less than n_splits=10.
  % (min_groups, self.n_splits)), UserWarning)
```

```
---------- Review Scores Communication KNN Model ----------

Model Best Params: {'n_neighbors': 7}
Model Best Score: 84.35530085959886

- Classifier -
Mean Squared Error: 0.4212508185985593
Mean Absolute Error: 0.20743287491814014
Root Mean Squared Error: 0.6490383799118195
R2: 0.9629491299492805
Metrics Accuracy: 83.46430910281597

- Regressor -
Mean Squared Error: 0.3120364059179664
Mean Absolute Error: 0.25278323510150624
Root Mean Squared Error: 0.5586021893243585
R2: 0.9725550199161093
```


Review Scores Communication Classifier Density Plot
Density
0.0
0.2
0.4
0.6
0.8
0
2
4
6
8
10
Review Scores Communication

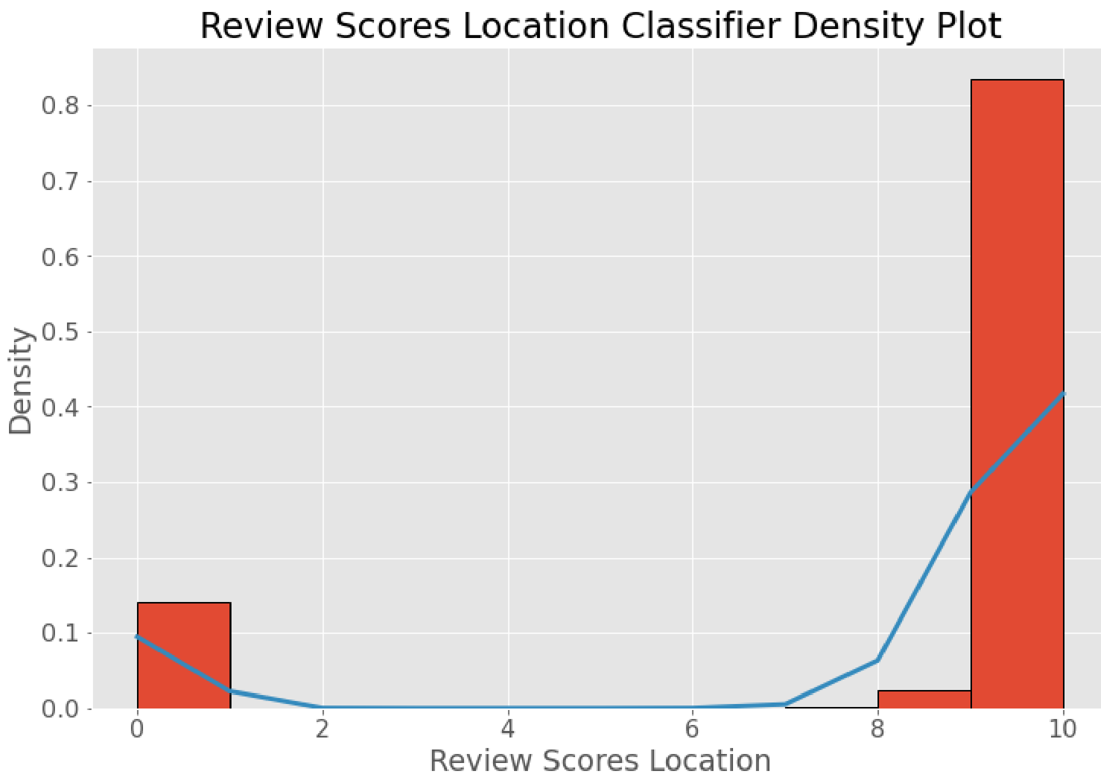

Review Scores Communication Regressor Density Plot
Density
0
1
2
3
4
5
6
7
8
0
2
4
6
8
10
Review Scores Communication

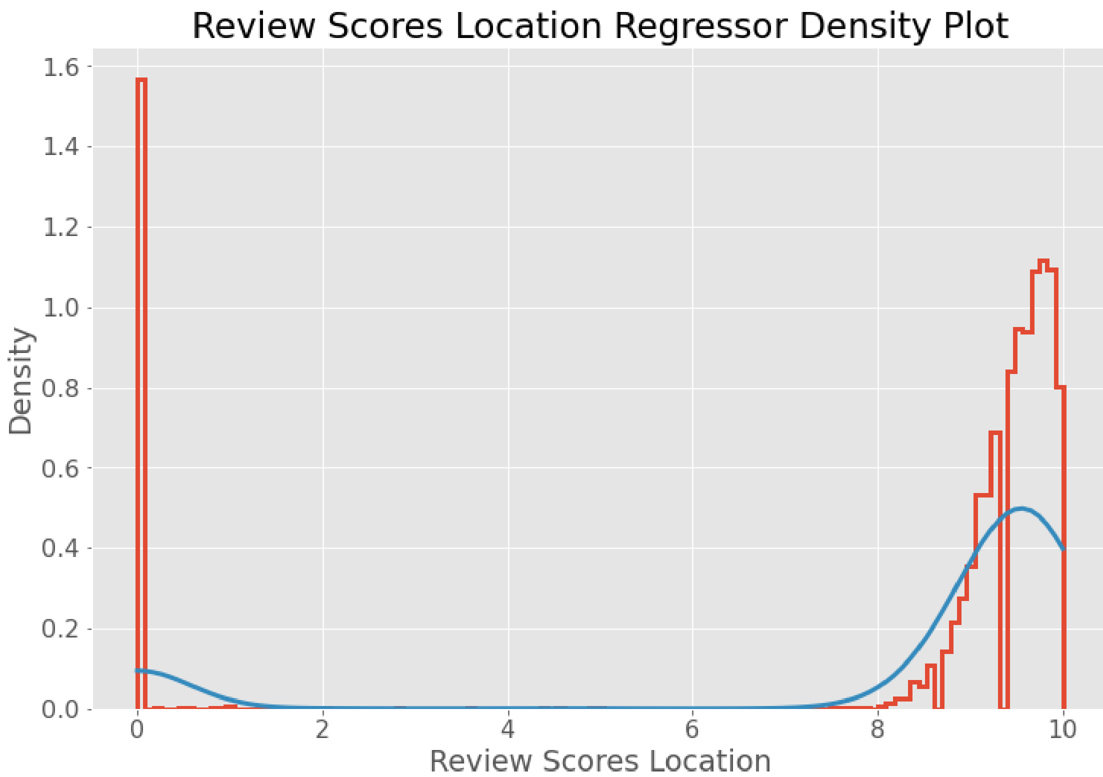
```
---------- End of Review Scores Communication ----------

---------- Beginning of Review Scores Location ----------
```



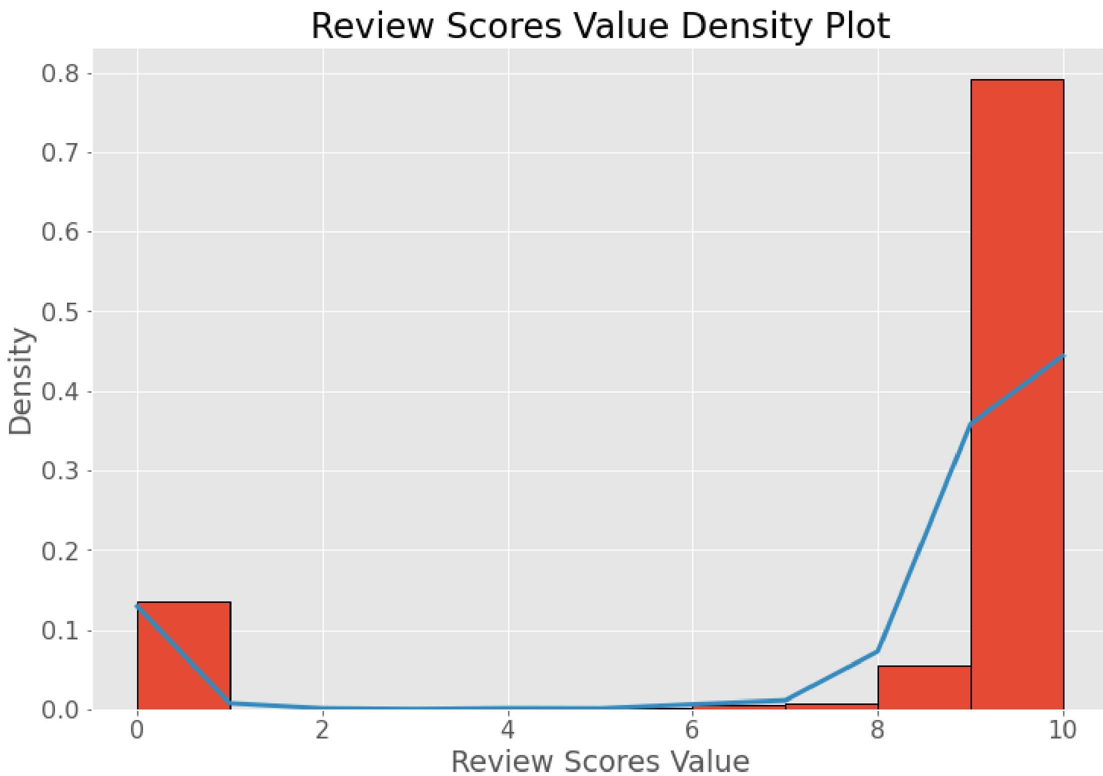

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:672: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=10.
  % (min_groups, self.n_splits)), UserWarning)

---------- Review Scores Location KNN Model ----------

Model Best Params: {'n_neighbors': 10}
Model Best Score: 70.54441260744986

- Classifier -
Mean Squared Error: 0.5985592665356909
Mean Absolute Error: 0.35494433529796987
Root Mean Squared Error: 0.77366612084005
R2: 0.9464804804552184
Metrics Accuracy: 70.48133595284872
```

```
- Regressor -
Mean Squared Error: 0.44739521938441396
Mean Absolute Error: 0.40420759659462996
Root Mean Squared Error: 0.6688760867189183
R2: 0.9599966477393793
```

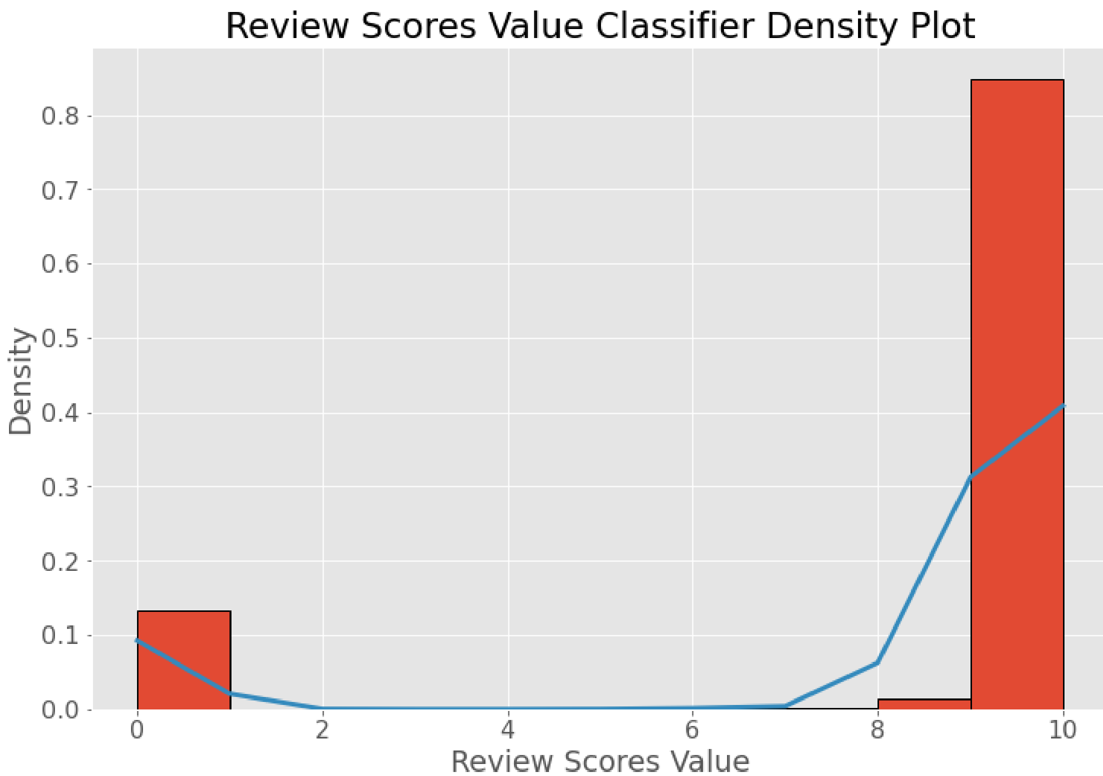


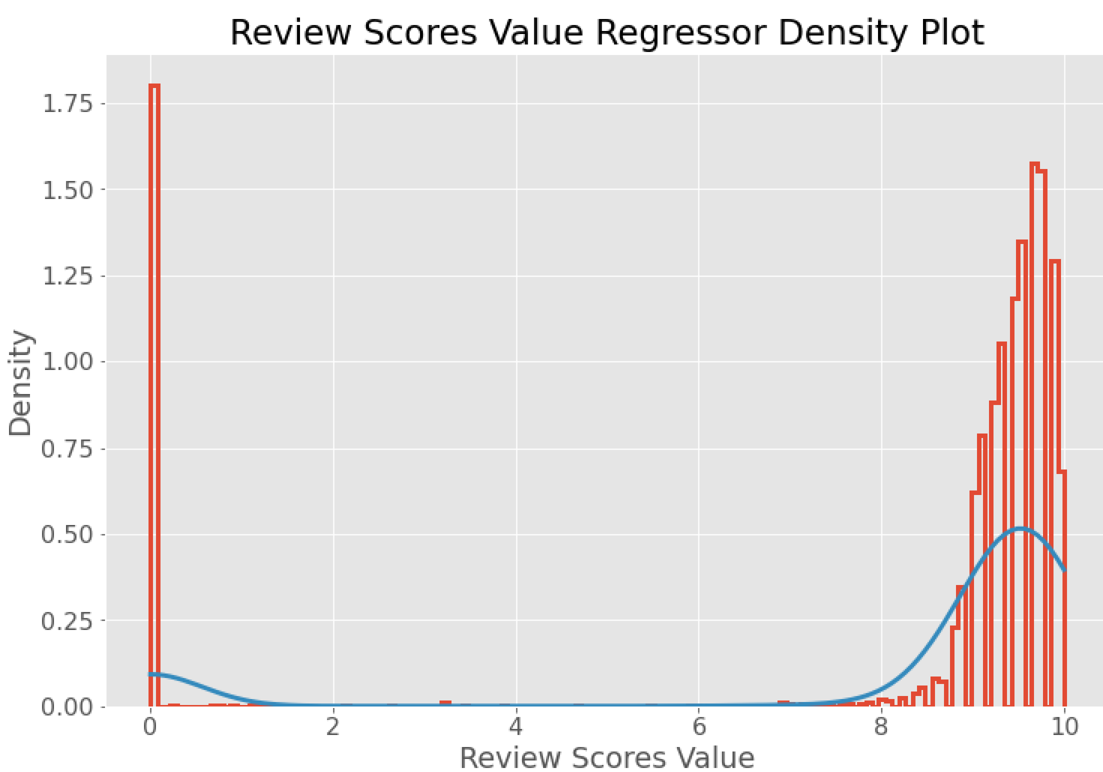



```
---------- End of Review Scores Location ----------

---------- Beginning of Review Scores Value ----------
```

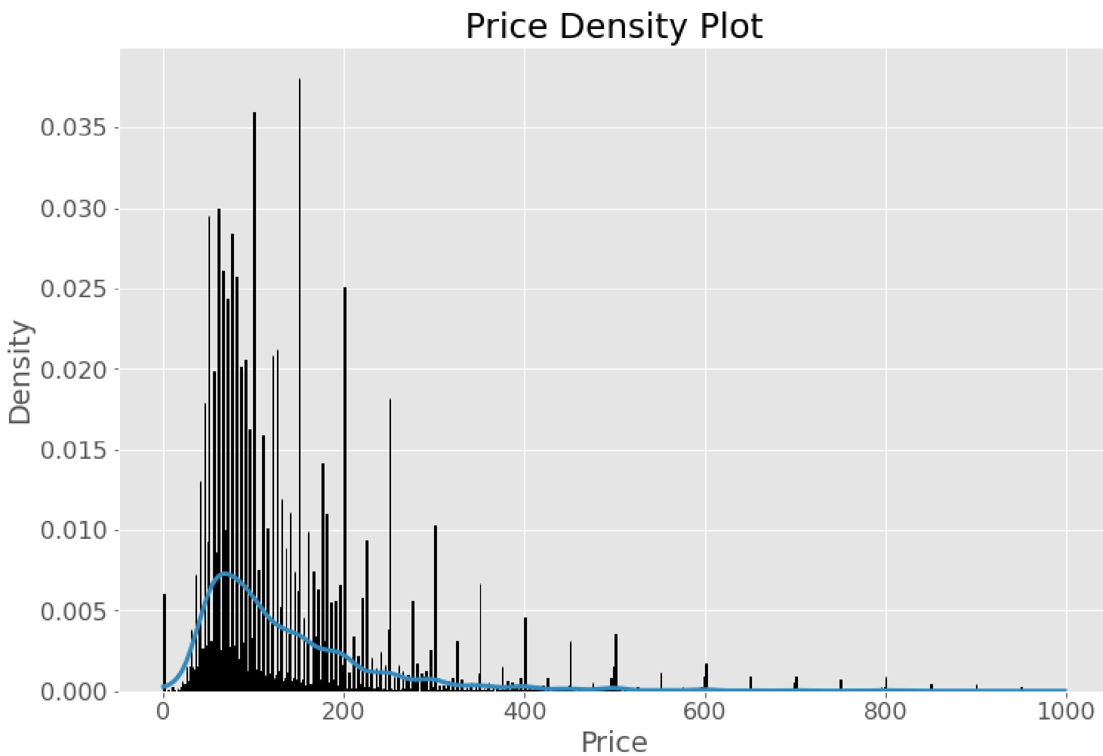



```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:672: UserWarning: The least populated
class in y has only 2 members, which is less than n_splits=10.
  % (min_groups, self.n_splits)), UserWarning)

---------- Review Scores Value KNN Model ----------

Model Best Params: {'n_neighbors': 9}
Model Best Score: 70.26606631191157

- Classifier -
Mean Squared Error: 0.5383104125736738
Mean Absolute Error: 0.3578912901113294
Root Mean Squared Error: 0.7336964035441865
R2: 0.9494761002122212
Metrics Accuracy: 69.33529796987557

- Regressor -
Mean Squared Error: 0.4108637124354217
Mean Absolute Error: 0.40857163646947525
Root Mean Squared Error: 0.6409865150183908
R2: 0.9614377939778733
```


Review Scores Value Classifier Density Plot
Density
Review Scores Value
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2
4
6
8
10

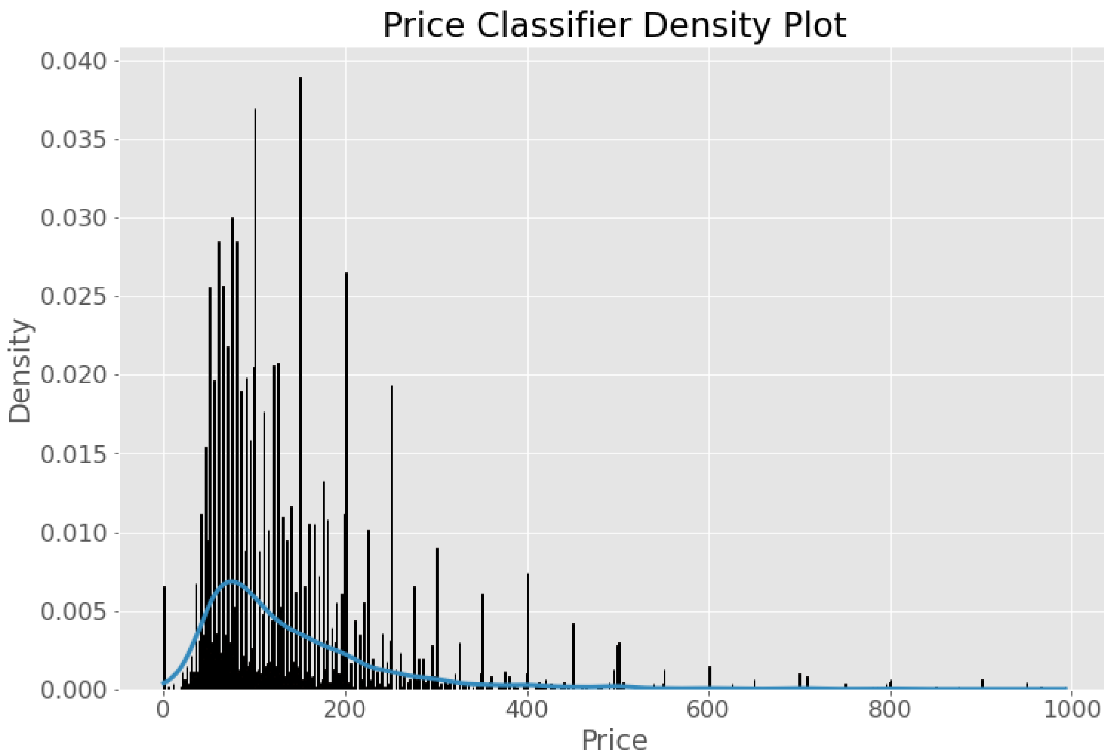

Review Scores Value Regressor Density Plot
Density
Review Scores Value
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
2
4
6
8
10

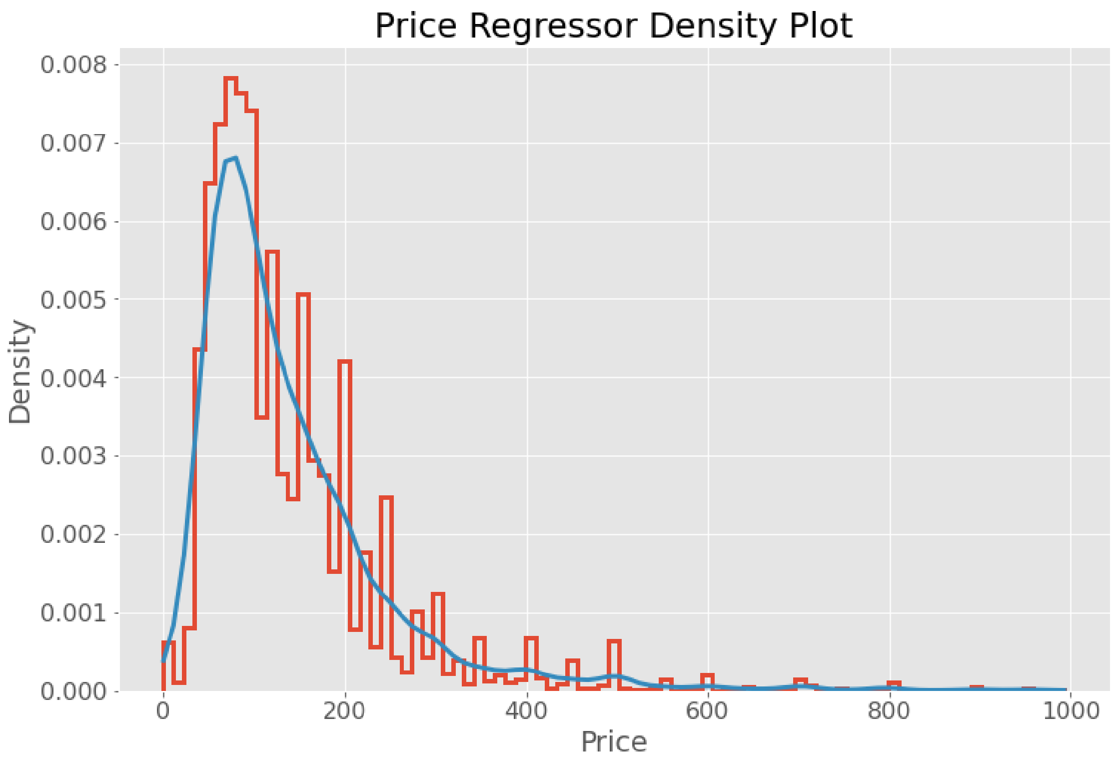
```
---------- End of Review Scores Value ----------

---------- Beginning of Price ----------
```




```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:672: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=10.
  % (min_groups, self.n_splits)), UserWarning)
---------- Price KNN Model ----------

Model Best Params: {'n_neighbors': 1}
Model Best Score: 5.517805976258698

- Classifier -
Mean Squared Error: 10067.880320890636
Mean Absolute Error: 58.28536345776031
Root Mean Squared Error: 100.33882758379548
R2: 0.1765173208062064
Metrics Accuracy: 5.255402750491159
```

```
- Regressor -
Mean Squared Error: 10067.880320890636
Mean Absolute Error: 58.28536345776031
Root Mean Squared Error: 100.33882758379548
R2: 0.1765173208062064
```





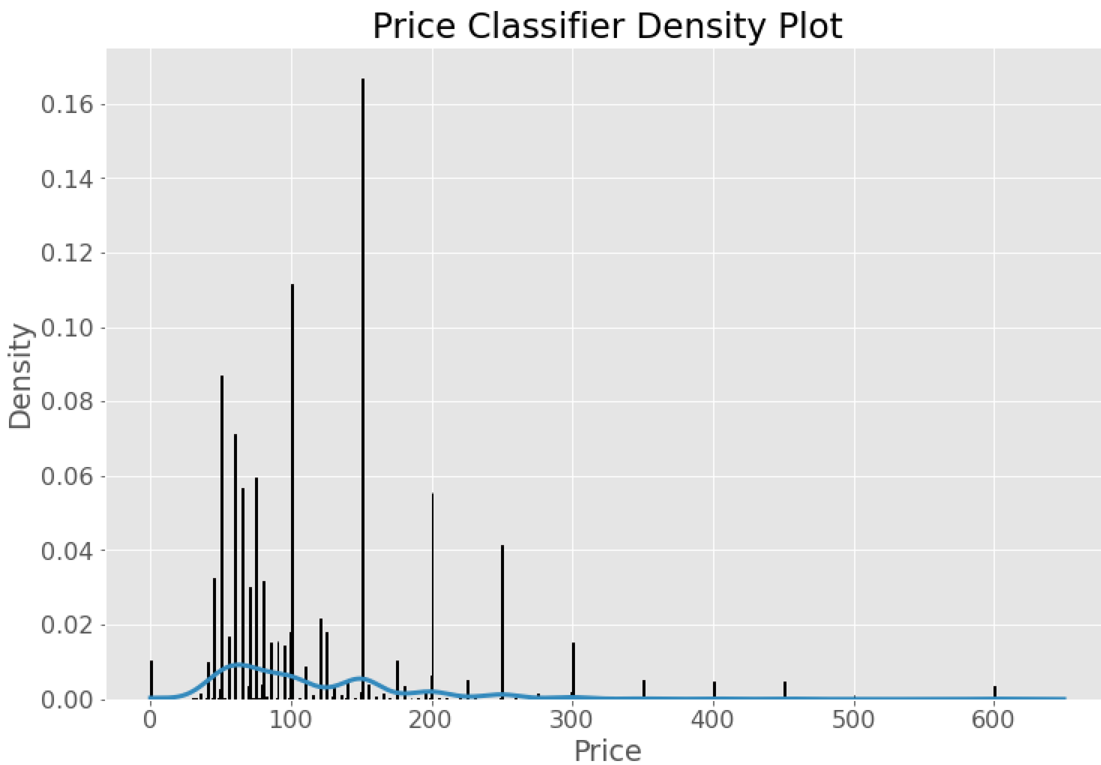

```
---------- End of Price ----------
```

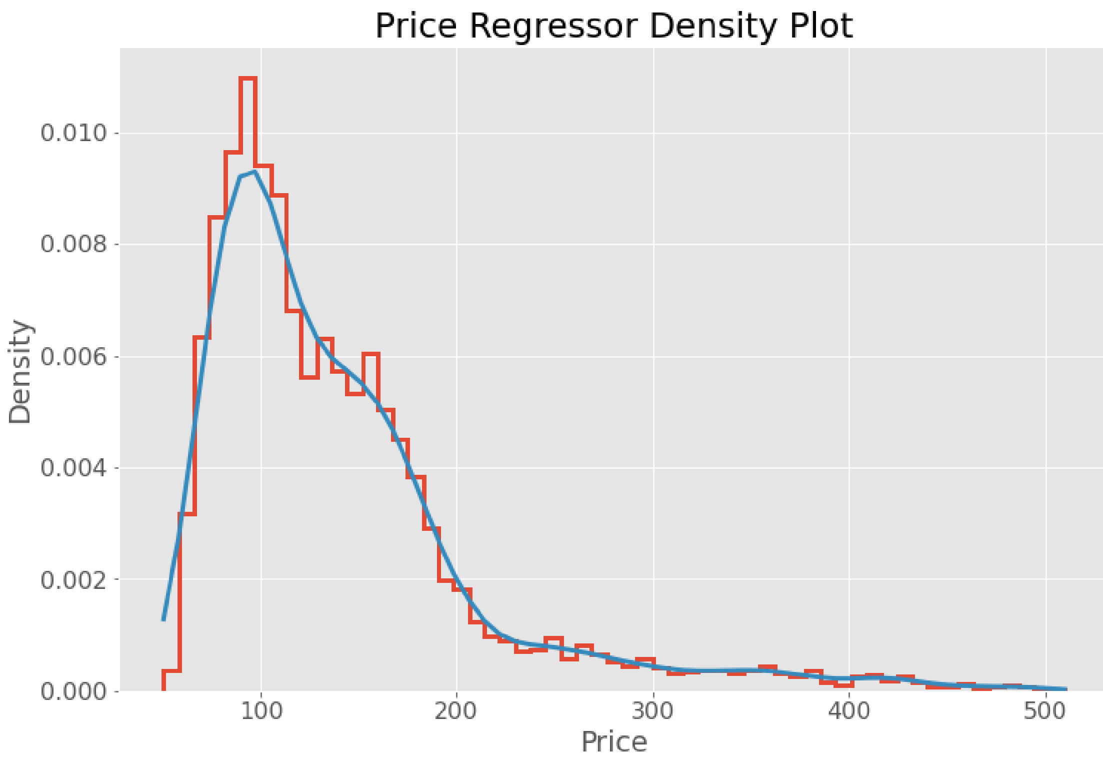
[59]:
```
# test price with non-overfitting n_neighbors value
print('---------- Testing Price with Larger N_Neighbors Value ----------')
# drop the variable to predict for train test split initializtion
x = df.drop('Price', axis=1)
y = df['Price']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.20)

# scale the data
scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

# use the ideal parameters to fit the training data and make predictions
best_knn = KNeighborsClassifier(n_neighbors=75)
best_knn.fit(x_train, y_train)
y_pred = best_knn.predict(x_test)
vals = pd.Series(y_pred)

best_reg = KNeighborsRegressor(n_neighbors=75)
```

```
best_reg.fit(x_train, y_train)
reg_pred = best_reg.predict(x_test)
reg_vals = pd.Series(reg_pred)

# summarize results
print('\n- Classifier -')
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
↪y_pred)))
print('R2:', metrics.r2_score(y_test, y_pred))
print('Metrics Accuracy:', accuracy_score(y_pred,y_test)*100)

print('\n- Regressor -')
print('Mean Squared Error:', metrics.mean_squared_error(y_test, reg_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, reg_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
↪reg_pred)))
print('R2:', metrics.r2_score(y_test, reg_pred), end='\n\n')

# plot results in a density plot
fig = plt.figure(figsize=(12,8))
noise = vals
density = stats.gaussian_kde(noise)
n, x, _ = plt.hist(noise, bins=range(min(noise), max(noise) + 1, 1),␣
↪histtype='bar', density=True, linewidth=1, edgecolor='k')
plt.xlabel(pred)
plt.ylabel('Density')
plt.title(pred+' Classifier Density Plot')
plt.plot(x, density(x), linewidth=3)
plt.show()

fig = plt.figure(figsize=(12,8))
noise = reg_vals
density = stats.gaussian_kde(noise)
n, x, _ = plt.hist(noise, bins='auto', histtype=u'step', density=True,␣
↪linewidth=3)
plt.xlabel(pred)
plt.ylabel('Density')
plt.title(pred+' Regressor Density Plot')
plt.plot(x, density(x), linewidth=3)
plt.show()
```

```
---------- Testing Price with Larger N_Neighbors Value ----------

- Classifier -
Mean Squared Error: 9580.503929273085
```

```
Mean Absolute Error: 54.73117223313687
Root Mean Squared Error: 97.88004867833426
R2: 0.28502768426888936
Metrics Accuracy: 5.697445972495088

- Regressor -
Mean Squared Error: 6502.077484915958
Mean Absolute Error: 49.43537218947827
Root Mean Squared Error: 80.63546046818334
R2: 0.5147640008529067
```





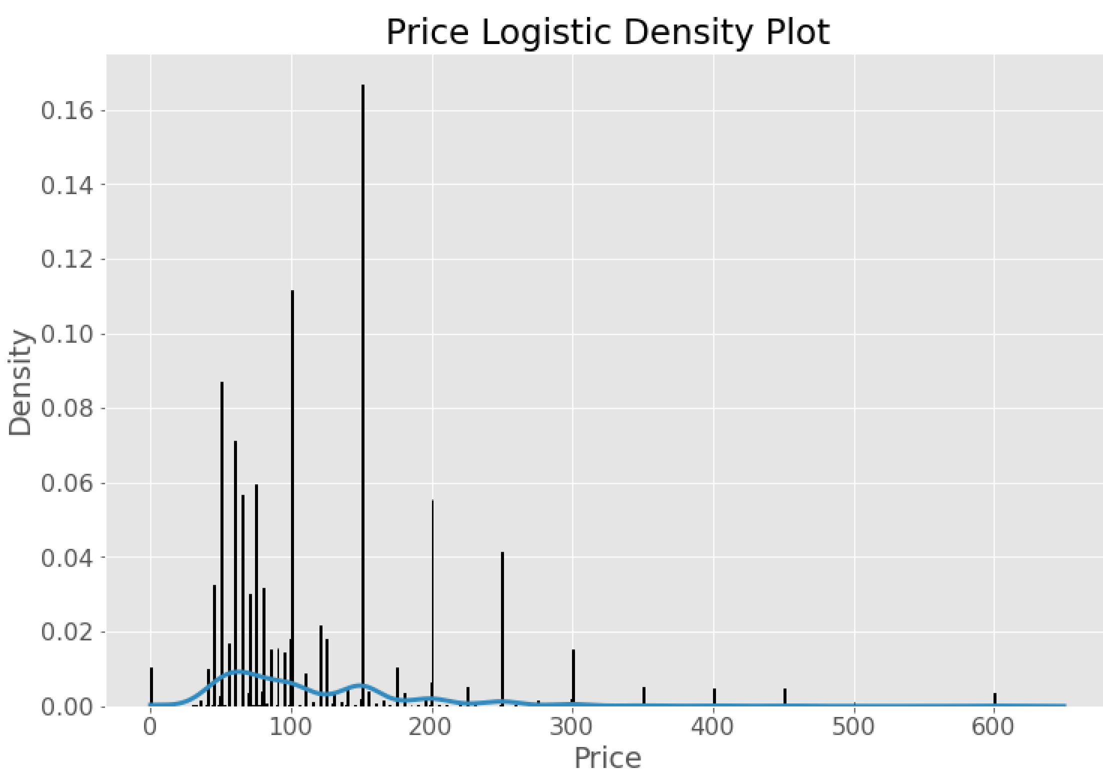
[61]:
```
# test price for logistic regression fit
print('---------- Testing Price with Logistic Model ----------', end='\n')
from sklearn.linear_model import LogisticRegression

# drop dependent variable
x = df.drop('Price', axis=1)
y = df['Price']

# intialize test_train_split and logistic regression
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=4)
logistic_regression = LogisticRegression()

# fit and predict the data using the trains
logistic_regression.fit(x_train, y_train)
print('Model Score:',logistic_regression.score(x_train, y_train)*100)
y_pred = logistic_regression.predict(x_test)
accuracy = metrics.accuracy_score(y_test, y_pred)*100
print('Metrics Accuracy:',accuracy, end='\n\n')

# plot results in a density plot
fig = plt.figure(figsize=(12,8))
noise = vals
```

```
density = stats.gaussian_kde(noise)
n, x, _ = plt.hist(noise, bins=range(min(noise), max(noise) + 1, 1),␣
↪histtype='bar', density=True, linewidth=1, edgecolor='k')
plt.xlabel(pred)
plt.ylabel('Density')
plt.title(pred+' Logistic Density Plot')
plt.plot(x, density(x), linewidth=3)
plt.show()
```

```
---------- Testing Price with Logistic Model ----------
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/sklearn/linear_model/_logistic.py:764: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
Model Score: 4.082434615552548
Metrics Accuracy: 4.191224623444663
```



2 ML Conclusion

Overall, what we observed was that the KNN model was best able to predict data with a higher density in a condensed location (i.e. stronger skew to left or right sides), and data that had fewer predictable values.

For example, the KNN model on price was overfitted - the model chose to use an n_neighbors value of 1. As displayed by the density plot, the model tried to copy the distribution makeup rather than being able to accurately make predictions. However, as price has over 1000 potential values, the model struggled to make overall accurate predictions.

The review scores rating variable plots encountered a similar issue, though the model accuracy was much better given that it only had to predict using a potential 100 values rather than 1000. Further, the regressor version of the review scores rating model was able to significantly improve upon the accuracy of the classifier model, lowering mean squared error from 43.8 to 31.4. This resulted in over a full percentage point increase in the R^2 value of the model. We saw the same effect when we manually increased the n_neighbors to values such as 30 or 75 for the price variable to avoid overfitting. The accuracy of the models both improved compared to the overfit model with an n_neighbor value of 1. Between the 75 n_neighbor classifier and regressor plots, this improvement became more dramatic. The regressor version lowered mean squared error to 5911 and increased R^2 to 52%, compared to 9435 and 23% respectively for the classifier version. The necessity of a higher n_neighbor value was only reinforced by the fact that the price variable poorly fit under a logistic regression model. Thus, the KNN Regressor model for price with n_neighbors larger than 10 provided the best final result.

The review scores checkin and review scores communication variables had the highest model accuracy at approximately 85% each (which the regressor version actually improved upon). What we observe from these plots is that they are very skewed to either 0 or 10 values. These plots, of all other variables, have the least amount (or smallest %) of potential values in the data value distributions. Thus, the KNN model was able to essentially ignore most of the values between 0 and 10 (excluding maybe 8 or 9) when trying to make a prediction using the training sets.

Aside from the number of potential values, all of the review scores models were more effective than that of the price likely due to user response bias. With the review scores, as displayed by the plots, users tend to give scores of either 0 or 10. This is because the actual action of giving a rating is somewhat impulsive. Users may classify their experiences as either positive or negative, not thinking into the gray area between - trying to figure out a true balance of the positive and negtive factors. Thus, the 0 score is frequently used to represent a negative experience - regardless of how negative - and the 10 score vice versa. All the KNN model then has to do is classify whether or not the data provided would lead to a more negative or more positive experience, as then likely assign a 0 or 10 score (possibly delegating an 8 or 9 score for more neutral indicators).

The price variable, on the other hand, is not determined by user feedback. Instead, the hosts set the prices. Given that this price impacts the hosts' incomes, it is likely broken down into much further thought compared to, for example, a cleanliness score. As a result, as seen in the plots, we saw that price had a much larger range of potential values and a less skewed distribution. Although the price distribution is right-skewed, it is not skewed to a potential one or two values like that of the review scores plots. Contrarily, there is a price range of about 300 different values that encompass the right skew. Therfore, it becomes much more difficult for the KNN model to differntiate between the factors that lead to a 225 price versus a 235 price. This is the cause of the overfitting in the original model. However, for these more complex variables, the regressor versions were able to provide much more accurate and useful results as the KNN did not have to try and differentiate between price classification in the scenario above. Rather, it had to determine what was a reasonable regression trend between the prices and map it out. This was much more effective when the total number of potential values were larger and the incremental difference between the values smaller.

To summarize, I would recommend that Airbnb hosts use a KNN Regressor model for predicting dependent variables such as price and review scores rating which are more complex, less skewed, and distributions with more total value possibilities. For these more complex cases, I would make sure to check that the n_neighbors value is sufficiently large to rule out possiblity of overfitting and check the accuracy in a different model (ex. Logistic), such as we did with the price variable. On the other hand, I would recommend using a KNN Classifier model for predicting dependent variables such as review scores checkin and review scores communication which are less complex, more skewed, and distributions with fewer total value possibilties. With these combinations of KNN model types, we were able to effectively and fairly accurately make predictions based on the training data, which should be of great use to Airbnb hosts trying to determine how to improve their review scores ratings or setting a fair price for their listings.

[ ]:

```
```