Our Hotels, in Review

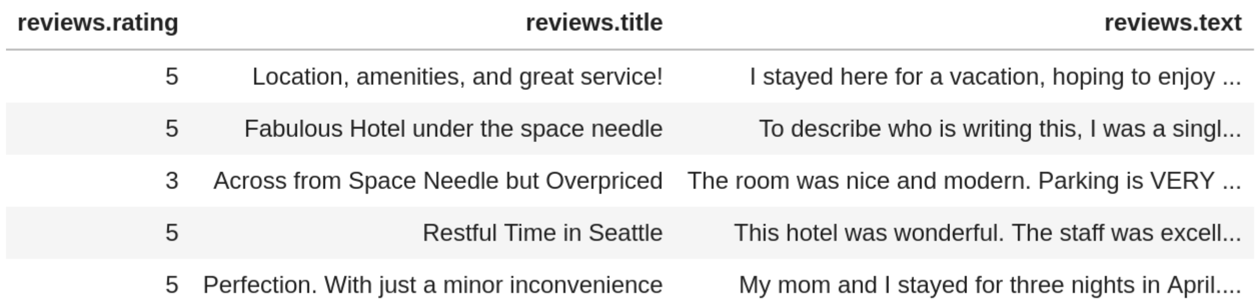
A Report on Guest Feedback

Brodie Heywood

October 28, 2020

Introduction

This report uses a TripAdvisor hotel reviews dataset. The dataset is a collection of 10,000 guest reviews. Each review contains rating, title, and text fields (Figure 1).



reviews.rating	reviews.title	reviews.text
5	Location, amenities, and great service!	I stayed here for a vacation, hoping to enjoy ...
5	Fabulous Hotel under the space needle	To describe who is writing this, I was a singl...
3	Across from Space Needle but Overpriced	The room was nice and modern. Parking is VERY ...
5	Restful Time in Seattle	This hotel was wonderful. The staff was excell...
5	Perfection. With just a minor inconvenience	My mom and I stayed for three nights in April....

Figure 1. An example of the rating, title, and text fields of typical reviews.

We'll use this data to identify features that positively and negatively impact guest experience at a popular hotel (Part A) as well as train models to predict review sentiment and/or rating (Parts B and C). In Part D, we'll compare different types of models to find the most accurate one.

See Appendices A-C for the Python code used to generate the analyses in this report and the above-mentioned models.

Part A. Examining our Most-Reviewed Hotel

In this section, we take a look at the most frequently reviewed hotel in our dataset, **Hyatt House Seattle**[1] (201 5th Ave N, Seattle). We'll use the data to identify some of the positive and negative things about the hotel. See Appendix A for the Python code used to generate the analyses in this section.

Our dataset contains 155 reviews for HHS. Each review contains a rating. Ratings are distributed as follows:

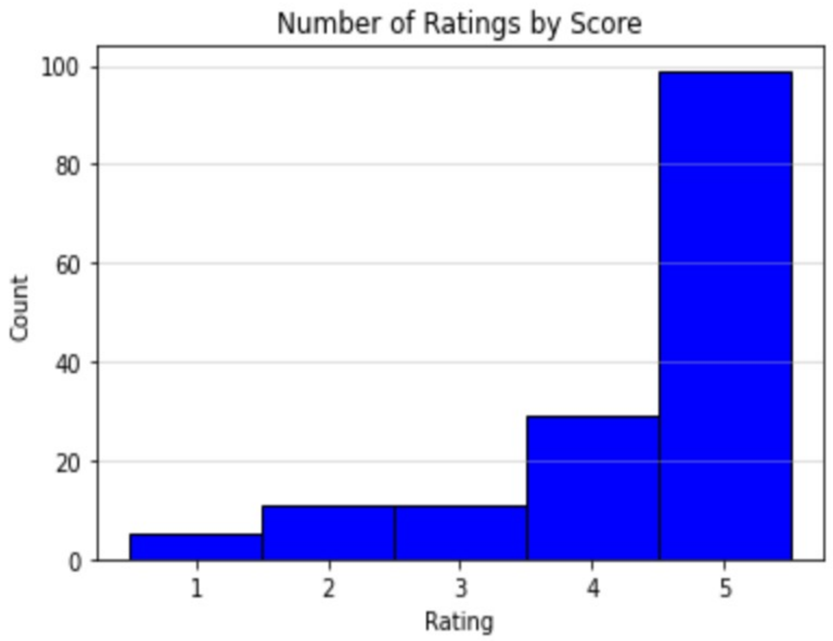


Figure 2. Histogram showing the number of HHS reviews for each rating.

We'll use ratings to approximate the sentiment of reviews and classify reviews as:

1 From here on, we'll refer to Hyatt House Seattle by its abbreviation, "HHS".

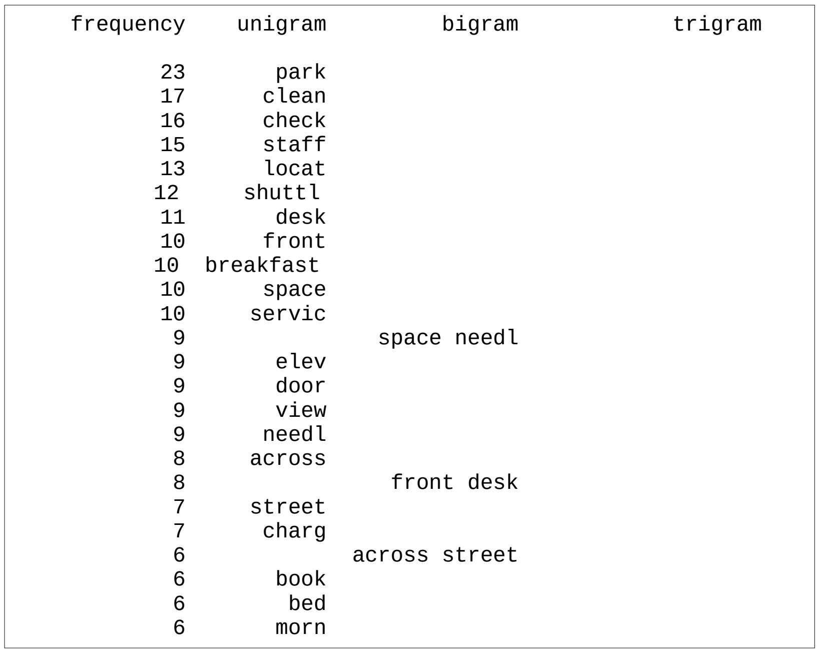
1. favourable (rating of 4 or 5), or
2. unfavourable (rating of 1, 2, or 3).

Under this scheme, 128 reviews are considered favourable and 27 unfavourable.

For each category of reviews, we'll find the most popular n-grams[2] in the title and text fields. These n-grams will help us identify the negative and positive aspects of HHS that most impact guest experience.

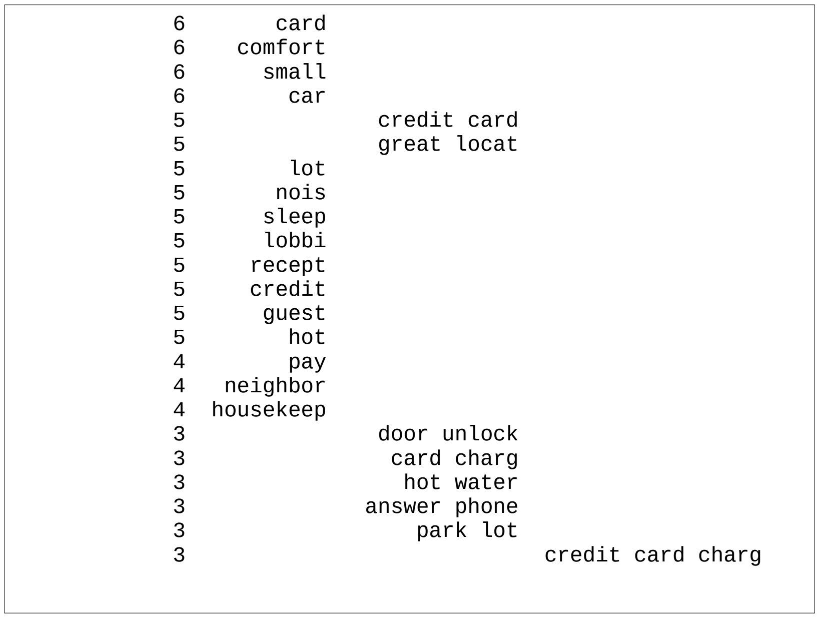
Issues that may need improvement

Figure 3 (below) shows the most frequently used n-grams in unfavourable reviews. We find n-grams for stems[3] in order to more accurately measure the frequency of each *concept*, rather than each specific tense or conjugation. The leftmost column (frequency) describes the number of times each n-gram occurred in review titles and text:



frequency	unigram	bigram	trigram
23	park		
17	clean		
16	check		
15	staff		
13	locat		
12	shuttl		
11	desk		
10	front		
10	breakfast		
10	space		
10	servic		
9		space needl	
9	elev		
9	door		
9	view		
9	needl		
8	across		
8		front desk	
7	street		
7	charg		
6		across street	
6	book		
6	bed		
6	morn		

2 An n-gram is a group of words, n words long; for example, an n-gram of size two contains two words (eg. "nice view").
3 Stems are basically root words; for example, "charg" is the stem of "charge," "charged," and "charging."



```
6      card
6   comfort
6     small
6       car
5                credit card
5                great locat
5       lot
5      nois
5     sleep
5     lobbi
5    recept
5    credit
5     guest
5       hot
4       pay
4  neighbor
4 housekeep
3                door unlock
3                 card charg
3                  hot water
3               answer phone
3                   park lot
3                                     credit card charg
```

Figure 3. N-grams of different sizes and their frequency in the title and text of 27 unfavourable reviews.

From these, we can identify several issues that may need to be addressed:

- parking availability,
- hotel cleanliness,
- front desk (and other staff) customer service, and
- customer billing (“credit card,” “card charge,” “pay,” etc. appear frequently).
- customer satisfaction with the shuttle service.

For reasons that we describe in Appendix A (*Notes* section), the results of n-gram frequencies can’t be taken at face value. The following n-grams can safely be ignored:

- “breakfast” because, as we will see in Figure 4, it is one of the highest frequency n-grams in favourable reviews. The positive reviews far outnumber the negative (111:10), so we believe this n-gram can be safely ignored in this context.
- “view” and “locat” (location) for the same reason as above.

Note that even unfavourable reviews can contain references to favourable aspects of the hotel. I suspect that this may have been the case for both “breakfast” and “view/locat”.

Well-rated features that may be promoted

Figure 4 (below) shows the most popular n-grams in favourable reviews. From these, we can identify the most well-regarded features of HHS. See Appendix A *Notes* for details on how only the most relevant n-grams are selected to be shown here:

```
frequency    unigram                   bigram                 trigram                          4gram                            5gram

      128      locat
      111  breakfast
      108      space
      106      staff
       99      needl
       94                         space needl
       58       walk
       57     across
       54   friendli
       54      clean
       48     shuttl
       47       view
       47     servic
       46     street
       46       area
       41       park
       40       free
       39                       across street
       38      check
       38                         great locat
       38    comfort
       38        bed
       33   downtown
       31   monorail
       29     museum
       28       elev
       28      front
       27      close
       27       desk
       26        bar
       24     center
       24    kitchen
       23  recommend
       23     buffet
       23     famili
       22    conveni
       22       busi
       22        car
       22       wait
       22       pike
       21      small
       21                      staff friendli
       21       food
       20                                                                  across street space needl
       20                          front desk
       20                        street space
       20                        right across
       20                                             across street space
       20    restaur
       20    distanc
       20        lot
       20                                              street space needl
       18                        walk distanc
       16                                             right across street
       16                          pike place
       15                         locat great
       15                                                view space needl
       15                          view space
       15                      free breakfast
       13                    breakfast buffet
       13                         free shuttl
       12                       shuttl servic
```

```
        11                       seattl center
        11                       friendli help
        11                         within mile
         9                                                                   right across street space
         9                                                                                                  right across street space needl
         9                     downtown seattl
         9                      breakfast good
         8                       perfect locat
         8                                                 great locat great
         8                                                 pike place market
         8                                               liter across street
         8                          staff help
         8                    breakfast includ
         8                        place market
         8                        liter across
         7                         locat right
         7                          great view
         7                         excel locat
         7                         locat excel
         7                        needl museum
         7                          happi hour
         7                                                space needl museum
         7                                               within walk distanc
         7                                               space needl chihuli
         7                          emp museum
         7                         within walk
         7                       needl chihuli
         6                     great breakfast
         6                         bed comfort
         6                                                 space needl right
         6                           queen bed
         6                         needl right
         5                                                locat right across
         5                                                 locat space needl
         5                                                 great great locat
         5                                                 right space needl
         5                                                   space needl emp
         5                          nice clean
         5                                                across space needl
         5                        across space
         5                            two elev
         5                            rent car
         5                        servic great
         5                       custom servic
         5                        seattl great
         5                     breakfast great
         5                         locat space
         5             complimentari breakfast
         5                         pike market
         5                       scienc center
         5                          omelet bar
         5                          park garag
         5                          minut walk
```

Figure 4. N-grams of different sizes and their frequency in the title and text of 128 favourable reviews.

According to these n-grams, positive features that HHS might benefit from promoting include:

- the “view of the Space Needle,” “great location,” and other similar concepts (like proximity to downtown and Pike Place Market); and
- anything to do with the complimentary breakfast buffet.

These features constitute the majority of n-grams.

Some n-grams appear frequently in both unfavourable reviews and favourable reviews. For this reason, management should be cautious about promoting the following hotel features:

- customer service,
- hotel cleanliness,
- shuttle service, and
- parking availability.

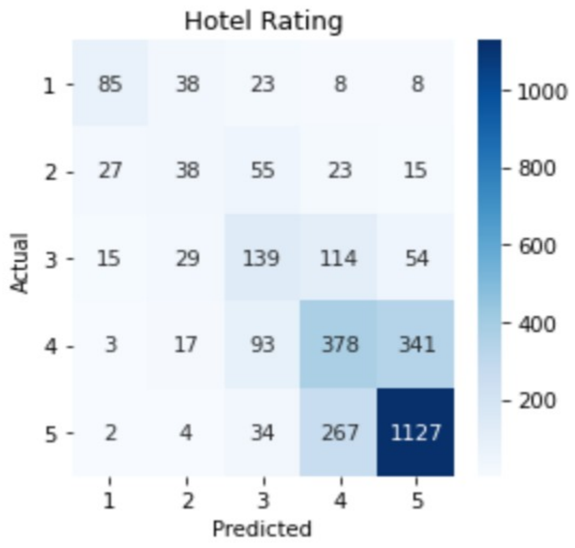
Finally, n-grams like “queen beds” and “elevator,” no matter much they seem to contribute to a positive rating, would be really weird to market. There’s not much we can do with these.

Part B. Logistic Regression Models

In this section, we validate two logistic regression models that were trained with data from the entire hotel dataset. The models predict either review ratings or review sentiment, given vectorized words[4] from review titles and text (see Appendix A *Code* for more details).

Model 1.

This model predicts review rating (1, 2, 3, 4, or 5) with approximately 60% accuracy. Where it is inaccurate, we can see that it's usually only off by one point. For example, notice the darker shaded cross shape around (4, 4) in the confusion matrix for this model (Figure 5):



Actual \ Predicted	1	2	3	4	5
1	85	38	23	8	8
2	27	38	55	23	15
3	15	29	139	114	54
4	3	17	93	378	341
5	2	4	34	267	1127

Figure 5. Confusion matrix for Model 1.

4 Assigned a coordinate in vector space, where like-words have nearer coordinates.

As we can also see from the confusion matrix, the model is best at predicting ratings of 5 (and more generally, high-end ratings like 4 or 5). The model struggles the most with actual ratings of 2. This seems intuitive – I'm sure even most humans would struggle to distinguish 2-rated reviews from those that are rated 1 or 3.

Below (Figure 6) is the precision[5] and recall[6] score for each rating:

```
   precision    recall

1       0.64      0.52
2       0.30      0.24
3       0.40      0.40
4       0.48      0.45
5       0.73      0.79
```

Figure 6. Model 1's precision and recall score for each rating.

Which confirms what we interpreted from the confusion matrix (for example, that the model is best at predicting ratings of 5). What wasn't immediately apparent from the confusion matrix that *is* from the precision and recall scores, however, is how much the model struggles with ratings of 4.

When comparing the precision and recall scores, we see that for all scores the model predicts more false negatives than false positives.

5 "How many selected items are relevant?"
precision score = true positives / (true positives + false positives)

6 "How many relevant items are selected?"
recall score = true positives / (true positives + false negatives)

Model 2.

This model predicts review sentiment (good, bad, or neutral) with approximately 84% accuracy. The reviews used for this model's training are classified as bad, neutral, or good based on their ratings (a rating of 1-2, 3, or 4-5, respectively). Based on the results of the previous model, it's no surprise that this model is more accurate (not to mention, there's less room for error); however, their confusion matrices do bear some similarities:

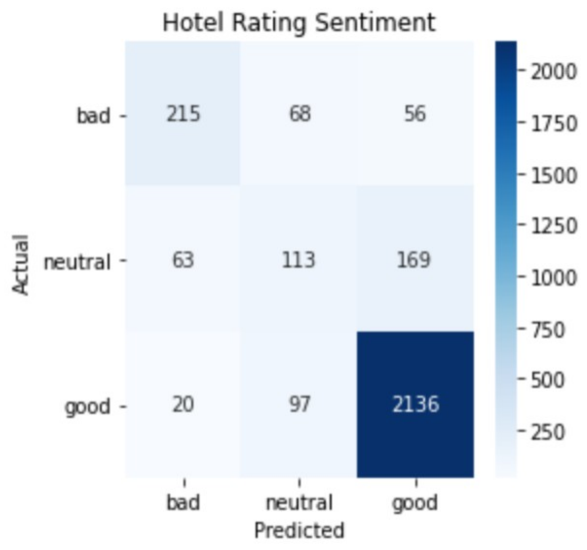


Figure 7. Confusion matrix for Model 2.

Again, the model is better at predicting "scores" on the higher end (in this case, reviews with good sentiment). This makes sense, given that it uses the same data. Again, the model struggles with the lower end.

The precision and recall scores for this model:

```
          precision    recall

bad       0.72         0.65
neutral   0.43         0.36
good      0.91         0.94
```

Figure 8. Model 1’s precision and recall score for each sentiment class.

Something interesting to note from the precision/recall scores is that the model performs slightly better at categorizing ratings of 3, if you still think about the neutral sentiment class in that way (remember, the neutral sentiment class was made up of only 3-rated reviews). Also, although it might be difficult to tell from the confusion matrix, this model is significantly better than the last at predicting positive reviews. Here it reaches an astounding 91% precision and 94% recall when classifying reviews with positive (good) sentiment; that is, it almost identified every single “good” review correctly.

Part C. Long Short-Term Memory (LSTM) Model

In this section, we validate a long short-term memory (LSTM) model that was trained with data from the entire hotel dataset to predict review ratings from review text. The model has an accuracy of 55%. The code for this section is found in Appendix C.

Below (Figure 9) is the confusion matrix for the LSTM model:

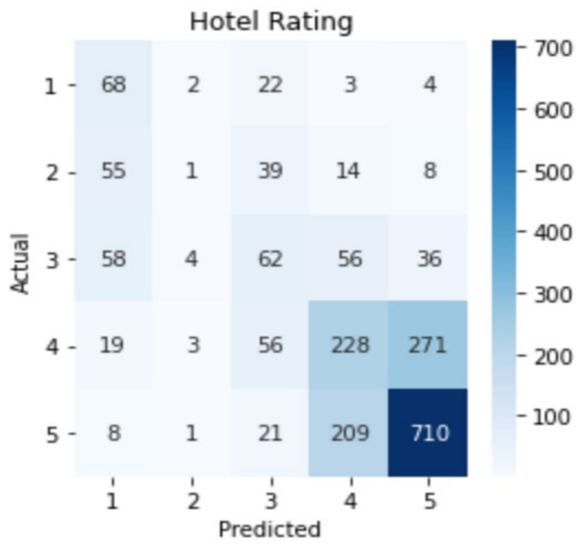


Figure 9. Confusion matrix for LSTM model.

We immediately notice that the LSTM model predicts 1 for actual ratings 1-3 in equal measure, almost entirely avoiding predicting 2 at all. In other words, if the model predicts 1, it is equally likely that the actual value is a 1, 2, or a 3. This isn't the worst case because ratings of 1-3 generally signify a negative review anyways. The model also applied a fairly normal distribution to its 3 predictions, which intuitively seems like a smart move.

Next we look at the LSTM model’s precision and recall score in Figure 10:

```
   precision    recall

1       0.33      0.69
2       0.09      0.01
3       0.31      0.29
4       0.45      0.40
5       0.69      0.75
```

Figure 10. LSTM model’s precision and recall score for each rating.

Here, we can quantify just how terribly the model performs for ratings of 2. Also notice the precision vs recall score for rating 1; it validates what we said above in relation to the confusion matrix. Precision score asks, “how many of the selected elements are relevant?” and we can see that a full two thirds were not. The recall score is better because the total proportion of predictions that the model was assigning to 1 managed to cover enough of the true positives.

In all, we can see that this model’s most urgent problem is predicting too many 1’s, to the detriment of predictions for 2.

Part D. Model Comparison

To wrap up this report, we'll compare the variance of the logistic regression model that predicts ratings (Model 1) and the LSTM model.

Our LSTM model was less accurate than our logistic regression model. We might have guessed this by looking at the two models' confusion matrices, alone. The source of these inaccuracies is predicting a rating of 1 too often for actual ratings 1-3.

We ran each model five times on randomized test data in order to compare them. Figure 11 is an average of those results over five runs:

	Logistic Regression Model (Model 1)	Long Short-Term Memory Model (LSTM)
Accuracy	58.26%	56%
RMSE	0.4954	0.9457
Precision Score	1: 0.59 2: 0.31 3: 0.41 4: 0.44 5: 0.72	1: 0.32 2: 0.07 3: 0.34 4: 0.42 5: 0.70
Recall Score	1: 0.49 2: 0.24 3: 0.36 4: 0.43 5: 0.75	1: 0.71 2: 0.02 3: 0.28 4: 0.40 5: 0.77

Figure 11. Average validation statistics for Model 1 and LSTM model (five runs, randomized test data).

We can see that the logistic regression model (Model 1) slightly outperforms the LSTM model when it comes to overall accuracy. When it comes to variance, the root-mean-square error was higher for the LSTM model meaning that it’s predictions were off by a greater factor than the logistic regression model.

By looking at the precision and recall scores, we see that the logistic regression model did much better than LSTM model when predicting lower ratings (1-3) but had about the same success rate for higher ratings (4-5). One exception to this rule can be seen in LSTM’s higher recall score for rating 1. That means that when the LSTM model predicted a rating of 1, it was more often correct (not as many false negatives). Of course, this is counterbalanced by it’s significantly worse precision score for 1.

The LSTM model does abysmally at predicting a rating of 2. It rarely made this prediction. Both models, however, seem to have considerable difficulty identifying this rating.

In conclusion, with more adjustments, I believe that the LSTM model will ultimately surpass the logistic regression model. For this reason, even though it performs slightly worse, I believe that the LSTM model is the better choice going forward. As it stands in this report, however, they may both be considered essentially equal.

Appendix A

Notes

The original dataset contains exactly 10 000 reviews, 210 of which are duplicates. We removed these duplicates from the dataframe used to generate our report. Surprisingly, 54 of these duplicates came from the “unfavourable” reviews (rated 1, 2, or 3) of the most frequently reviewed hotel in our dataset. A subject matter expert might know why this duplication occurred. We suspect that responses from hotel management (which needed to be removed from some reviews) may have sometimes quoted back the entire review, interfering with accurate data collection. On a related note, we used code to remove hotel management responses from the text body of several of the reviews. This was essential because, as might be expected, the inclusion of highly-similar responses from management was drastically interfering with the results we got back from n-gram analysis.

Note that, when finding n-grams, we are trying to identify the most likely contributors to a low/high rating. For this reason, stopwords[7] and other irrelevant n-grams were programmatically filtered out of the n-gram results, where possible. Furthermore, the n-grams shown in the report (Figures 3 and 4) were manually selected from a larger list of n-grams as they were judged to be more significant contributors to low/high ratings. For the sake of brevity, we will not detail these choices on a case-by-case basis; however, all exclusions were based on common sense. For example, the most commonly used n-gram in unfavourable review text is "room". That could lead one to believe that the rooms themselves are a negative contributor to review rating. In fact, by looking at the full reviews we can see that "room" is most commonly used in a neutral context; for example, when describing the process of check in (as in, "when I arrived at my room").

7 Commonly used words such as "a," "and," and "but" that are excluded from natural language data.

Code

1) Import data into pandas DataFrame

Get only the necessary columns (and rename them so they can be referred to more easily).

In [14]:
```python
from pathlib import Path
import pandas as pd

PATH = Path("Datafiniti_Hotel_Reviews_Jun19.csv")
df = pd.read_csv(
    PATH,
    usecols=[
        'id',
        'reviews.rating',
        'reviews.title',
        'reviews.text',
    ],
)

df.rename(
    columns = {
        'id': 'hotel_id',
        'reviews.rating': 'rating',
        'reviews.title': 'title',
        'reviews.text': 'text',
    },
    inplace = True
)
```

Drop duplicate rows.

In [15]:
```python
og_row_count = df.shape[0]
df = df.drop_duplicates()
new_row_count = df.shape[0]

num_duplicates = og_row_count - new_row_count
print("Dropped {0} duplicate columns".format(num_duplicates))
```

```
Dropped 213 duplicate columns
```

Get a subset of this dataframe - keep only the rows for the most-reviewed hotel.

In [16]:
```python
# find most-reviewed hotel (check by id)
hotel_id = df.hotel_id.value_counts().index[0]

# create df for this hotel and drop [now] redundant 'hotel_id' col
df_hotel = df[df.hotel_id == hotel_id].drop(columns='hotel_id')
df_hotel.head(3)
```

Out[16]:

	rating	text	title
7759	5	I stayed here for a vacation, hoping to enjoy ...	Location, amenities, and great service!
7760	5	To describe who is writing this, I was a singl...	Fabulous Hotel under the space needle
7761	3	The room was nice and modern. Parking is VERY ...	Across from Space Needle but Overpriced

2) Visualize the data

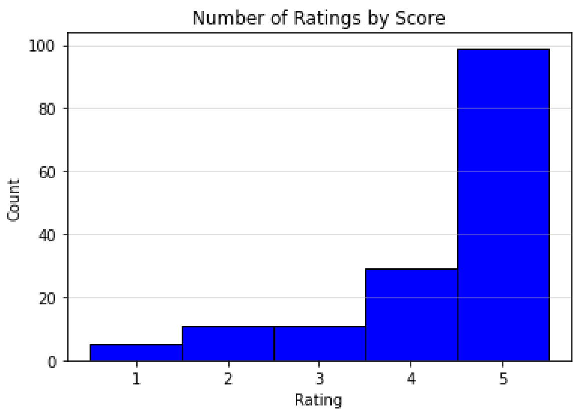
Plot ratings in a histogram.

In [17]:
```python
import matplotlib.pyplot as plt

plt.title('Number of Ratings by Score')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.hist(
    df_hotel["rating"],
    bins=[1, 2, 3, 4, 5, 6],
    align='left',
    color='blue',
    edgecolor='black'
)
plt.grid(
    axis='y',
    alpha=0.5
)

plt.plot()
```

```
Out[17]: []
```



```
In [ ]:
# an alternative, interactive histogram (for use in jupyter)
import plotly.express as px

fig = px.histogram(
    df_hotel['rating'],
    x="rating",
    title='Ratings Count',
    labels={'rating': 'Rating'},
    opacity=0.8,
    color_discrete_sequence=['blue'] # color of histogram bars
)

fig.show()
```

3) Cleanse the data

For the title and text columns:

1. Remove long, unwanted strings (in this case, responses to reviews by hotel management)
2. Tokenize
3. Remove stopwords
4. Remove other irrelevant words
5. Stem words

```
In [18]:
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import re


def remove_mgmt_response(reviews_text: str) -> str:
    """
    Remove management's response from review, return review.

    Keyword arguments:
    reviews_text -- text from all reviews as one long string
    """
    # remove substring: feedback from Sarah Junge and Jonathan
    reviews_text = re.sub(
        r'(Feedback from our guests|,Thank|, Thank|,We|, We|Dear)'  # starts w/
        + '(.*?)'  # include substring between beginning/end
        + '(Sarah Junge|Jonathan)',  # ends w/
        '',
        reviews_text)

    reviews_text = re.sub(
        r'(It was a pleasure|Thank you for completing)'
        + '(.*?)'
        + '(Sarah Junge|Jonathan)',
        '',
        reviews_text)

    # remove substring: feedback from unknown author(s)
    reviews_text = re.sub(
        r'Hello(.*?)(good nights sleep!|wherever you go!)',
        '',
        reviews_text)
    reviews_text = re.sub(
        r'Thank you for taking(.*?)(Cheers!|Seattle!)',
        '',
        reviews_text)

    reviews_text = re.sub(
        r'Thank you for sharing(.*?)(soon!)',
        '',
        reviews_text)

    return reviews_text


def cleanse_df_columns(
        df: pd.DataFrame,
        cols: [str],
        cleaner_function: object,
        filter_list: [str] = []) -> pd.DataFrame:
    """Prepare df column(s) for vectorization and return as a new df.

    For each column: remove long, unwanted strings; tokenize; remove
    stopwords; stem words; and remove stems of words found in stemmed
    filter list, in that order.

    keyword arguments:
    df -- dataframe to tokenize
    cols -- name of column(s) to be tokenized
    cleaner_function -- function specialized for each particular
    dataset that removes long, unwanted strings
    filter_list -- words whose stems will be removed from column(s)
    """
    tokenized_df = df.copy()
    tokenizer = RegexpTokenizer(r'\w+')  # removes punctuation
    stop_words = set(stopwords.words('english'))
    ps = PorterStemmer()

    # get stems of words in filter list
    filter_list_stems = [ps.stem(word) for word in filter_list]

    for col in cols:

        # remove long strings
        tokenized_df[col] = tokenized_df[col].apply(
            lambda row: cleaner_function(str(row)))
        # tokenize
        tokenized_df[col] = tokenized_df.apply(
            lambda row: tokenizer.tokenize(row[col].lower()), axis=1)
        # remove stopwords
        tokenized_df[col] = tokenized_df[col].apply(
            lambda tokens: [word for word in tokens if word not in stop_words])
        # stem words
        tokenized_df[col] = tokenized_df[col].apply(
            lambda tokens: [ps.stem(word) for word in tokens])
        # remove filter list word stems
        tokenized_df[col] = tokenized_df[col].apply(
            lambda stems: [
                stem for stem in stems if stem not in filter_list_stems])

    return tokenized_df


filter_list = ['hotel', 'hyatt', 'house', 'room', 'night', 'stay',
               'day', 'get', 'back', 'ask', 'time', 'would', 'one',
               'left', 'next', 'said', 'told', 'could', 'go', 'take',
               '1', '2', '4', '5', '6']

df_hotel_tokenized = cleanse_df_columns(
    df_hotel, ['text', 'title'], remove_mgmt_response, filter_list)
df_hotel_tokenized.head(3)
```

Out[18]:

	rating	text	title
7759	5	[vacat, hope, enjoy, locat, receiv, much, good...	[locat, amen, great, servic]
7760	5	[describ, write, singl, travel, 3, person, par...	[fabul, space, needl]
7761	3	[nice, modern, park, limit, hard, park, small,...	[across, space, needl, overpr]

4) Classify the data

Sort into two dataframes:

- unfavourable reviews (rating of 1, 2, or 3)
- favourable reviews (rating of 4 or 5)

Author's note: In retrospect, I should have kept reviews in one dataframe and added a column "favourable" that takes a binary value (bit is set to 1 if rating is 1, 2, or 3). Here, I thought it would be cleaner to view unfavourable/favourable n-grams in separate dataframes. I could have accomplished the same thing with a single dataframe by just viewing unfavourable/favourable n-gram columns at separate times.

In [19]:
```python
unfav_rate = [1, 2, 3]
fav_rate = [4, 5]

df_hotel_unfav = df_hotel_tokenized[
    df_hotel_tokenized['rating'].isin(unfav_rate)].drop(columns='rating')

df_hotel_fav = df_hotel_tokenized[
    df_hotel_tokenized['rating'].isin(fav_rate)].drop(columns='rating')
```

5) Find n-grams

Find n-grams of size 1-5 for title and text columns.

In [20]:
```python
from collections import Counter
from nltk.util import ngrams


def get_ngrams(
        df: pd.DataFrame,
        cols: [str],
        ngram_max_size: int = 5,
        min_ngram_freq: int = 2,
        max_ngrams: int = 80) -> pd.DataFrame:
    """Get n-grams from dataframe column(s) and return a dataframe of
    n-grams and their frequency (descending order).

    keyword arguments:
    df -- dataframe to get n-grams from
    cols -- name of column(s) to get n-grams from
    ngram_max_size -- maximum n-gram size to get
    min_ngram_freq -- minimum frequency of n-grams (before inclusion)
    max_ngrams -- maximum number of n-grams to return
    """
    # create one long list of words from column(s)
    words = []
    for col in cols:
        df[col].apply(
            lambda tokens: [words.append(word) for word in tokens])

    # for each n-gram size, create df for n-grams and their frequencies
    ngram_dfs = []
    for ngram_size in range(1, ngram_max_size + 1):

        # count n-gram occurrences
        ngram_frequency_list = Counter(
            ngrams(words, ngram_size)).most_common(max_ngrams)

        frequencies = []
        ngrams_list = []

        # ignore those n-grams below the required frequency
        for i in range(0, len(ngram_frequency_list)):
            if (ngram_frequency_list[i][1] >= min_ngram_freq):
                frequencies.append(ngram_frequency_list[i][1])
                ngrams_list.append(' '.join(ngram_frequency_list[i][0]))

        ngram_data = {
            'frequency': frequencies,
        }

        # get column name for n-grams, based on n-gram size
        if ngram_size == 1:
            ngram_name = 'unigram'
        elif ngram_size == 2:
            ngram_name = 'bigram'
        elif ngram_size == 3:
            ngram_name = 'trigram'
        else:
            ngram_name = str(ngram_size) + 'gram'

        ngram_data.update({ngram_name: ngrams_list})
        ngram_dfs.append(pd.DataFrame(ngram_data))
    ngram_df = ngram_dfs.pop(0)
    for df in ngram_dfs:
        if len(df.index):
            ngram_df = ngram_df.append(df)

    ngram_df = ngram_df.sort_values(
        by='frequency', ascending=False).reset_index(drop=True)
    return ngram_df


df_unfav_ngrams = get_ngrams(df_hotel_unfav, ['title', 'text'], 5, 3)
df_fav_ngrams = get_ngrams(df_hotel_fav, ['title', 'text'], 5, 5)
```

In [21]:
```
df_unfav_ngrams
```

Out[21]:

	frequency	unigram	bigram	trigram
0	23	park	NaN	NaN
1	17	clean	NaN	NaN
2	16	check	NaN	NaN
3	16	call	NaN	NaN
4	15	staff	NaN	NaN
...	...	...	...	...
86	3	NaN	card charg	NaN
87	3	NaN	hot water	NaN
88	3	NaN	answer phone	NaN
89	3	NaN	park lot	NaN
90	3	NaN	NaN	credit card charg

91 rows × 4 columns

In [22]:
```
df_fav_ngrams
```

Out[22]:

	frequency	unigram	bigram	trigram	4gram	5gram
0	157	great	NaN	NaN	NaN	NaN
1	128	locat	NaN	NaN	NaN	NaN
2	111	breakfast	NaN	NaN	NaN	NaN
3	108	space	NaN	NaN	NaN	NaN
4	106	staff	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...
172	5	NaN	omelet bar	NaN	NaN	NaN
173	5	NaN	park garag	NaN	NaN	NaN
174	5	NaN	minut walk	NaN	NaN	NaN
175	5	NaN	earli check	NaN	NaN	NaN
176	5	NaN	needl emp	NaN	NaN	NaN

177 rows × 6 columns

Appendix B

1) Import data into pandas DataFrame

Get only the necessary columns (and rename them so they can be referred to more easily).

In [57]:
```
from pathlib import Path
import pandas as pd

PATH = Path("Datafiniti_Hotel_Reviews_Jun19.csv")
df = pd.read_csv(
    PATH,
    usecols=[
        'id',
        'reviews.rating',
        'reviews.title',
        'reviews.text',
    ],
)

df.rename(
    columns = {
        'id': 'hotel_id',
        'reviews.rating': 'rating',
        'reviews.title': 'title',
        'reviews.text': 'text',
    },
    inplace = True
)
```

Drop duplicate rows.

In [58]:
```
og_row_count = df.shape[0]
df = df.drop_duplicates()
new_row_count = df.shape[0]

num_duplicates = og_row_count - new_row_count
print("Dropped {0} duplicate columns".format(num_duplicates))
```

```
Dropped 213 duplicate columns
```

2) Visualize the data

In [59]:
```
df.shape
```

Out[59]:
```
(9787, 4)
```

In [60]:
```
df.head(3)
```

Out[60]:

	hotel_id	rating	text	title
0	AWE2FvX5RxPSIh2RscTK	3	This hotel was nice and quiet. Did not know, t...	Best Western Plus Hotel
1	AVwcj_OhkufWRAb5wi9T	4	We stayed in the king suite with the separatio...	Clean rooms at solid rates in the heart of Carmel
2	AVwcj_OhkufWRAb5wi9T	3	Parking was horrible, somebody ran into my ren...	Business

In [61]:
```
import plotly.express as px

# Interactive histogram.
fig = px.histogram(
    df,
    x="rating",
)

fig.show()
```

3) Cleanse the data

For the title and text columns:

1. Remove long, unwanted strings (in this case, responses to reviews by hotel management)
2. Tokenize
3. Remove stopwords
4. Remove other irrelevant words
5. Stem words

In [62]:

```
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import re


def remove_mgmt_response(reviews_text: str) -> str:
    """
    Remove management's response from review, return review.

    Keyword arguments:
    reviews_text -- text from all reviews as one long string
    """
    # remove substring: feedback from Sarah Junge and Jonathan
    reviews_text = re.sub(
        r'(Feedback from our guests|,Thank|, Thank|,We|, We|Dear)'  # starts w/
        + '(.*?)'  # include substring between beginning/end
        + '(Sarah Junge|Jonathan)',  # ends w/
        '',
        reviews_text)

    reviews_text = re.sub(
        r'(It was a pleasure|Thank you for completing)'
        + '(.*?)'
        + '(Sarah Junge|Jonathan)',
        '',
        reviews_text)

    # remove substring: feedback from unknown author(s)
    reviews_text = re.sub(
        r'Hello(.*?)(good nights sleep!|wherever you go!)',
        '',
        reviews_text)

    reviews_text = re.sub(
        r'Thank you for taking(.*?)(Cheers!|Seattle!)',
        '',
        reviews_text)

    reviews_text = re.sub(
        r'Thank you for sharing(.*?)(soon!)',
        '',
        reviews_text)

    return reviews_text
```

```
def cleanse_df_columns(
        df: pd.DataFrame,
        cols: [str],
        cleaner_function: object,
        filter_list: [str] = []) -> pd.DataFrame:
    """Prepare df column(s) for vectorization and return as a new df.

    For each column: remove long, unwanted strings; tokenize; remove
    stopwords; stem words; and remove stems of words found in stemmed
    filter list, in that order.

    keyword arguments:
    df -- dataframe to tokenize
    cols -- name of column(s) to be tokenized
    cleaner_function -- function specialized for each particular
    dataset that removes long, unwanted strings
    filter_list -- words whose stems will be removed from column(s)
    """
    tokenized_df = df.copy()
    tokenizer = RegexpTokenizer(r'\w+')  # removes punctuation
    stop_words = set(stopwords.words('english'))
    ps = PorterStemmer()

    # get stems of words in filter list
    filter_list_stems = [ps.stem(word) for word in filter_list]

    for col in cols:

        # remove long strings
        tokenized_df[col] = tokenized_df[col].apply(
            lambda row: cleaner_function(str(row)))
        # tokenize
        tokenized_df[col] = tokenized_df.apply(
            lambda row: tokenizer.tokenize(row[col].lower()), axis=1)
        # remove stopwords
        tokenized_df[col] = tokenized_df[col].apply(
            lambda tokens: [word for word in tokens if word not in stop_words])
        # stem words
        tokenized_df[col] = tokenized_df[col].apply(
            lambda tokens: [ps.stem(word) for word in tokens])
        # remove filter list word stems
        tokenized_df[col] = tokenized_df[col].apply(
            lambda stems: [
                stem for stem in stems if stem not in filter_list_stems])

    return tokenized_df


filter_list = ['hotel', 'hyatt', 'house', 'room', 'night', 'stay',
               'day', 'get', 'back', 'ask', 'time', 'would', 'one',
               'left', 'next', 'said', 'told', 'could', 'go', 'take',
               '1', '2', '4', '5', '6']
```

In [63]:
```
df_tokenized = cleanse_df_columns(
    df, ['text', 'title'], remove_mgmt_response, filter_list)
```

In [64]:
```
df_tokenized.head(3)
```

Out[64]:

	hotel_id	rating	text	title
0	AWE2FvX5RxPSIh2RscTK	3	[nice, quiet, know, train, track, near, train,...	[best, western, plu]
1	AVwcj_OhkufWRAb5wi9T	4	[king, suit, separ, bedroom, live, space, sofa...	[clean, solid, rate, heart, carmel]
2	AVwcj_OhkufWRAb5wi9T	3	[park, horribl, somebodi, ran, rental, car, tr...	[busi]

4) Vectorize the data

Create word vector and vectorize data to make it useable in a logistic regression model.

In [65]:
```
from sklearn.feature_extraction.text import CountVectorizer


def vectorize_cols(df: pd.DataFrame, cols: [str]) -> pd.DataFrame:
    """Vectorize columns and return as a new dataframe.

    keyword arguments:
    df -- dataframe to vectorize
    cols -- name of column(s) to be vectorized
    """
    vectorized_df = df.copy()
    cv = CountVectorizer(binary=True)
    flat_list = []

    # Each row in each col contains a tokenized list;
    # for example, two columns in the first row might look like this:
    #     df[cols[0]][0] = ['nice', 'quiet', 'know']
    #     df[cols[1]][0] = ['train', 'track', 'near']
    # We need the cols in each row to be combined into one string, like this:
    #     'nice quiet know train track near'
    # So that finally strings from each row can be appended to one list, that
    # looks like this:
    #     ['nice quiet know train track near', 'pass best western chang']
    # Then the rows (or 'documents') can be vectorized.
    for index, row in vectorized_df.iterrows():
        row_string = ''

        for col in cols:
            if not row_string:  # if first col to be added to row_string
                join_string = ''
            else:
                join_string = ' '
            row_string = join_string.join([row_string, ' '.join(row[col])])

        flat_list.append(row_string)

    X = cv.fit_transform(flat_list)
#     print(cv.vocabulary_)  # dict of unique vocabulary words
#     print(X.shape)
    return X
```

In [66]:
```
X = vectorize_cols(df_tokenized, ['text', 'title'])
```

5a) Create logistic regression model to predict review rating (Model 1)

Split ratings data into training and testing subsets.

In [67]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, df_tokenized['rating'], train_size = 0.70)
```

Train model.

In [68]:
```
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings("ignore")

rating_model = LogisticRegression().fit(X_train, y_train)
```

Predict ratings.

In [69]:
```
y_predicted = rating_model.predict(X_test)
```

5b) Validate Model 1

Display confusion matrix.

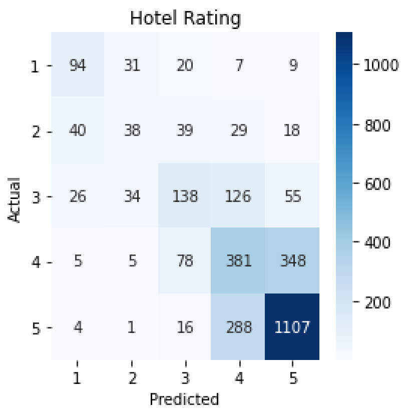
In [70]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from pandas import DataFrame


def display_hotel_rating_pred_confusion_matrix(
    y_test: pd.core.series.Series, y_predicted: pd.core.series.Series) -> None:
    """Display confusion matrix for predictions."""

    cm = metrics.confusion_matrix(y_test.values, y_predicted)

    # Show confusion matrix with colored background.
    labels = [str(num) for num in range(1, 6)]
    index = labels
    cols = labels
    df = DataFrame(cm, index=index, columns=cols)
    plt.figure(figsize = (4,4))
    ax = sns.heatmap(df, cmap='Blues', annot=True, fmt='g')
    bottom, top = ax.get_ylim()
    ax.set(title = 'Hotel Rating')
    ax.set_ylabel('Actual')
    ax.set_xlabel('Predicted')
    ax.set_ylim(bottom, top)
    ax.set_xticklabels(
        ax.get_xticklabels(), rotation=0, horizontalalignment='center')
    ax.set_yticklabels(
        ax.get_yticklabels(), rotation=0, horizontalalignment='right')


display_hotel_rating_pred_confusion_matrix(y_test, y_predicted)
```



Get model accuracy.

In [71]:
```
accuracy_score = rating_model.score(X_test, y_test)
print('{:.2f}% Accuracy'.format(accuracy_score * 100))
```

```
59.86% Accuracy
```

Get precision and recall scores.

In [72]:
```
from sklearn.metrics import classification_report

classification_report = classification_report(y_test, y_predicted)
print(classification_report)
```

```
              precision    recall  f1-score   support

           1       0.56      0.58      0.57       161
           2       0.35      0.23      0.28       164
           3       0.47      0.36      0.41       379
           4       0.46      0.47      0.46       817
           5       0.72      0.78      0.75      1416

    accuracy                           0.60      2937
   macro avg       0.51      0.49      0.49      2937
weighted avg       0.59      0.60      0.59      2937
```

6a) Create logistic regression model to predict review sentiment (Model 2)

Add classification column to dataframes and set value to 0, 1, or 2 (for bad, neutral, and good, respectively).

In [73]:
```
bad = [1, 2]
neutral = [3]
good = [4, 5]

df_classified = df_tokenized.copy()

sentiment_list = []
for index, row in df_classified.iterrows():
    if row['rating'] in bad:
        sentiment_list.append(1)
    elif row['rating'] in neutral:
        sentiment_list.append(2)
    elif row['rating'] in good:
        sentiment_list.append(3)
df_classified['sentiment'] = sentiment_list

df_classified.head()
```

Out[73]:

	hotel_id	rating	text	title	sentiment
0	AWE2FvX5RxPSIh2RscTK	3	[nice, quiet, know, train, track, near, train,...	[best, western, plu]	2
1	AVwcj_OhkufWRAb5wi9T	4	[king, suit, separ, bedroom, live, space, sofa...	[clean, solid, rate, heart, carmel]	3
2	AVwcj_OhkufWRAb5wi9T	3	[park, horribl, somebodi, ran, rental, car, tr...	[busi]	2

	hotel_id	rating	text	title	sentiment
3	AVwcj_OhkufWRAb5wi9T	5	[cheap, excel, locat, price, somewhat, standar...	[good]	3
4	AVwcj_OhkufWRAb5wi9T	2	[advertis, websit, paid, may, lucki, mani, giv...	[low, chanc, come]	1

Split sentiment data into training and testing subsets.

In [74]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, df_classified['sentiment'], train_size = 0.70)
```

Train model.

In [75]:
```
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings("ignore")

sentiment_model = LogisticRegression().fit(X_train, y_train)
```

Predict review sentiment.

In [76]:
```
y_predicted = sentiment_model.predict(X_test)
```

6b) Validate Model 2

Display confusion matrix.

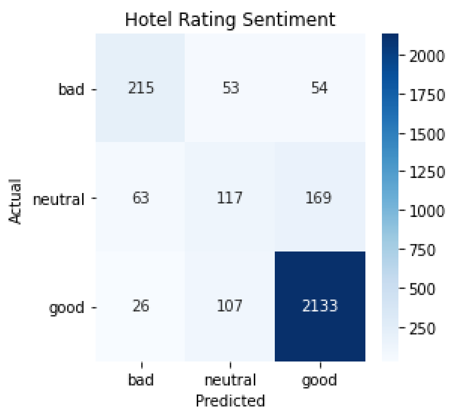
In [77]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from pandas import DataFrame


def display_hotel_sentiment_pred_confusion_matrix(
    y_test: pd.core.series.Series, y_predicted: pd.core.series.Series) -> None:
    """Display confusion matrix for predictions."""

    cm = metrics.confusion_matrix(y_test.values, y_predicted)

    # Show confusion matrix with colored background.
    labels = ['bad', 'neutral', 'good']
    index = labels
    cols = labels
    df = DataFrame(cm, index=index, columns=cols)
    plt.figure(figsize = (4,4))
    ax = sns.heatmap(df, cmap='Blues', annot=True, fmt='g')
    bottom, top = ax.get_ylim()
    ax.set(title = 'Hotel Rating Sentiment')
    ax.set_ylabel('Actual')
    ax.set_xlabel('Predicted')
    ax.set_ylim(bottom, top)
    ax.set_xticklabels(
        ax.get_xticklabels(), rotation=0, horizontalalignment='center')
    ax.set_yticklabels(
        ax.get_yticklabels(), rotation=0, horizontalalignment='right')


display_hotel_sentiment_pred_confusion_matrix(y_test, y_predicted)
```



Get model accuracy.

```
In [78]: accuracy_score = sentiment_model.score(X_test, y_test)
         print('{:.2f}% Accuracy'.format(accuracy_score * 100))
```

```
83.93% Accuracy
```

Get precision and recall scores.

```
In [79]: from sklearn.metrics import classification_report

         classification_report = classification_report(y_test, y_predicted)
         print(classification_report)
```

```
              precision    recall  f1-score   support

           1       0.71      0.67      0.69       322
           2       0.42      0.34      0.37       349
           3       0.91      0.94      0.92      2266

    accuracy                           0.84      2937
   macro avg       0.68      0.65      0.66      2937
weighted avg       0.83      0.84      0.83      2937
```

```
In [80]: import math

         rmse2 = math.sqrt(metrics.mean_squared_error(
             y_test, y_predicted))
         print("RMSE: {:.4f}".format(rmse2))
```

```
RMSE: 0.4924
```

Appendix C

1) Import data into pandas DataFrame

In [2]:

```
from pathlib import Path
import pandas as pd

PATH = Path("Datafiniti_Hotel_Reviews_Jun19.csv")
df = pd.read_csv(
    PATH,
    usecols=[
        'id',
        'reviews.rating',
        'reviews.title',
        'reviews.text',
    ],
)

df.rename(
    columns = {
        'id': 'hotel_id',
        'reviews.rating': 'rating',
        'reviews.title': 'title',
        'reviews.text': 'text',
    },
    inplace = True
)

og_row_count = df.shape[0]
df = df.drop_duplicates()
new_row_count = df.shape[0]

num_duplicates = og_row_count - new_row_count
print("Dropped {0} duplicate columns".format(num_duplicates))
```

```
Dropped 213 duplicate columns
```

2) Create long short-term memory (LSTM) model

In [3]:

```
from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
import pandas as pd
import re
from sklearn.model_selection import train_test_split
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Embedding, LSTM, Dense

data = df.copy()
data['text'] = data['text'].apply(
    (lambda x: re.sub('[^a-zA-z0-9\s]','',x)))

tokenizer = Tokenizer(num_words=2500, lower=True,split=' ')
tokenizer.fit_on_texts(data['text'].values)

X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)

embed_dim = 128
lstm_out = 200
batch_size = 32

model = Sequential()
model.add(Embedding(2500, embed_dim))
model.add(LSTM(lstm_out, dropout=0.2))
model.add(Dense(5, activation='softmax'))

model.compile(
    loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

Y = pd.get_dummies(data['rating']).values
X_train, X_test, Y_train, y_test = train_test_split(X,Y, test_size = 0.20)

history = model.fit(
    X_train, Y_train, batch_size=batch_size, epochs =4, verbose = 1,
    validation_data=(X_test, y_test))

score, acc = model.evaluate(X_test, y_test, verbose=2, batch_size=batch_size)
print("Score: {}".format(score))
print("Accuracy: {0:.2f}".format(acc * 100))
```

```
Epoch 1/4
245/245 [==============================] - 1247s 5s/step - loss: 1.1675 - accuracy: 0.5016 - val_loss: 1.0134 -
val_accuracy: 0.5317
Epoch 2/4
245/245 [==============================] - 1224s 5s/step - loss: 0.9818 - accuracy: 0.5561 - val_loss: 1.0513 -
val_accuracy: 0.4852
Epoch 3/4
245/245 [==============================] - 1218s 5s/step - loss: 0.9015 - accuracy: 0.5970 - val_loss: 1.1405 -
val_accuracy: 0.5107
Epoch 4/4
245/245 [==============================] - 1213s 5s/step - loss: 0.8349 - accuracy: 0.6378 - val_loss: 0.9716 -
val_accuracy: 0.5460
62/62 - 41s - loss: 0.9716 - accuracy: 0.5460
Score: 0.97
Validation Accuracy: 0.55
```

In [4]:
```
y_predicted = model.predict(X_test)
```

3) Validate LSTM model

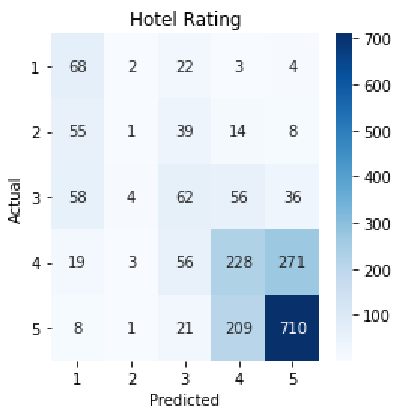
In [11]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from pandas import DataFrame


def display_hotel_rating_pred_confusion_matrix(
    y_test: pd.core.series.Series, y_predicted: pd.core.series.Series) -> None:
    """Display confusion matrix for predictions."""

#     cm = metrics.confusion_matrix(y_test.values, y_predicted)
    cm = metrics.confusion_matrix(
        y_test.argmax(axis=1), y_predicted.argmax(axis=1))

    # Show confusion matrix with colored background.
    labels = [str(num) for num in range(1, 6)]
    index = labels
    cols = labels
    df = DataFrame(cm, index=index, columns=cols)
    plt.figure(figsize = (4,4))
    ax = sns.heatmap(df, cmap='Blues', annot=True, fmt='g')
    bottom, top = ax.get_ylim()
    ax.set(title = 'Hotel Rating')
    ax.set_ylabel('Actual')
    ax.set_xlabel('Predicted')
    ax.set_ylim(bottom, top)
    ax.set_xticklabels(
        ax.get_xticklabels(), rotation=0, horizontalalignment='center')
    ax.set_yticklabels(
        ax.get_yticklabels(), rotation=0, horizontalalignment='right')


display_hotel_rating_pred_confusion_matrix(y_test, y_predicted)
```



In [15]:
```
from sklearn.metrics import classification_report

classification_report = classification_report(
    y_test.argmax(axis=1), y_predicted.argmax(axis=1))
print(classification_report)
```

```
              precision    recall  f1-score   support

           0       0.33      0.69      0.44        99
           1       0.09      0.01      0.02       117
           2       0.31      0.29      0.30       216
           3       0.45      0.40      0.42       577
           4       0.69      0.75      0.72       949
```

```
    accuracy                           0.55      1958
   macro avg       0.37      0.43      0.38      1958
weighted avg       0.52      0.55      0.53      1958
```

In [21]:
```
import math

rmse2 = math.sqrt(metrics.mean_squared_error(
    y_test.argmax(axis=1), y_predicted.argmax(axis=1)))
print("RMSE: {:.4f}".format(rmse2))
```

```
RMSE: 0.9532
```